holmium.core Documentation

Release 0.3

Ali-Akber Saifee

November 27, 2013

Contents

holmium.core

holmium.core provides utility classes to simplify writing pageobjects for webpages using 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)
        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 occurances / 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 excercise. 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 dont 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"
                                        , 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.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.assertEquals(self.page.header_text.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:
export HO_BROWSER=firefox;nosetests test_selenium_hq.py
# or..
export HO_BROWSER=firefox;python test_selenium_hq.py
# if using unittest.TestCase as the base class run as:
nosetests test_selenium_hq.py --holmium-browser=firefox
```

CHAPTER 1

Building PageObjects

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, "#fll>div>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 Programs', <selenium.webdriver.remote.webelement.WebElement object at
g.google_footer["About Google"]
# <selenium.webdriver.remote.webelement.WebElement object at 0x10b35f450>
g.google_footer["About Google"].get_attribute("href")
# u'http://www.google.ca/intl/en/about.html
driver.get("http://www.google.co.tz")
g.google_footer["Kila Kitu Kuhusu Google"].get_attribute("href")
# u'https://www.google.co.tz/intl/sw/about.html'
```

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.

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.

```
from holmium.core import Page, Section, 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(Locator.CLASS_NAME , "user")
    brief = Element(Locator.CSS_SELECTOR , "div.details div.brief")
    full_text = Element(Locator.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
```

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'
```

1.4 More Examples

1.4.1 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 = 1)
    google_buttons = ElementMap ( Locators.CLASS_NAME, "gbts" , timeout = 1 )
    search_results = Elements( Locators.CSS_SELECTOR, "li.g>div.rc", timeout = 1, 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["Search"].click() # self.google_buttons behaves just like a dictionary
        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"], "Selenium - Web Browser Automation")
        self.assertEquals( self.page.search_results[0]["link"], "http://docs.seleniumhq.org/")

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

1.4.2 Wikipedia text search example

```
import unittest
import selenium.webdriver
from holmium.core import Page, Element, Elements, 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 span[dir=auto]" )
    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.assertEquals(self.page.search("google").article_title.text, "Google", language)

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

CHAPTER 2

Holmium Classes

2.1 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 enter_query(self):
        ....

    def submit_search(self):
        ....

    def get_results(self):
        ....

assert len(Google().enter_query("page objects").submit_search().get_results()) > 0
```

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

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*)

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=1*, *value=<function <lambda> at 0x402e410>*)

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

class holmium.core.**Elements**(*locator_type*, *query_string*, *base_element=None*, *timeout=1*, *value=<function <lambda> at 0x402e410>*)

Utility class to get a collection of selenium.webdriver.remote.webelement.WebElement objects

by querying via one of `holmium.core.Locators`

class `holmium.core.`**`ElementMap`**(*locator_type*, *query_string=None*, *base_element=None*, *timeout=1*, *key=<function <lambda> at 0x402e758>*, *value=<function <lambda> at 0x402e7d0>*)

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

- **key** (*lambda*) – transform function for mapping a key to a WebElement in the collection
- **value** (*lambda*) – transform function for the value when accessed via the key.

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'**

classmethod **`is_valid`**(*by*)

2.2 Deprecated Classes

Earlier versions of `holmium.core` used rather verbose names for Page objects and elements. As of version 0.2 the classes have been renamed but the older names have been retained as aliases to the new classes for backward compatibility (an annoying warning will however, be emitted everytime the old names are used to hopefully convince test authors to update their test code :D).

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=1*, *value=<function <lambda> at 0x402e410>*)

Deprecated alias for `Element`

class `holmium.core.`**`PageElements`**(*locator_type*, *query_string*, *base_element=None*, *timeout=1*, *value=<function <lambda> at 0x402e410>*)

Deprecated alias for `Elements`

class `holmium.core.`**`PageElementMap`**(*locator_type*, *query_string=None*, *base_element=None*, *timeout=1*, *key=<function <lambda> at 0x402e758>*, *value=<function <lambda> at 0x402e7d0>*)

Deprecated alias for `ElementMap`

class `holmium.core.`**`HolmiumTestCase`**(*methodName='runTest'*)

Deprecated alias for `TestCase`

2.3 Utilities

class holmium.core.**TestCase**(*methodName='runTest'*)

holmium.core.**repeat**(*loop*)

CHAPTER 3

Unit Test Integration

Holmium provides two utilities to ease integration with automated tests.

3.1 The `TestCase` base class.

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

- automatically provision a selenium webdriver `driver` to the testcase which is selected based on the environment variable HO_BROWSER.
- 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 at the end of the test class.

3.1.1 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()
```

3.1.2 Execution

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

3.2 HolmiumNose plugin for nosetest.

This plugin registers the following command line options to nose:

option	description
--with-holmium	to enable the use of the holmium plugin
--holmium-browser	one of chrome,firefox,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 of extra desired capabilities to pass to the webdriver.

3.2.1 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")
```

3.2.2 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:5
```

Internal Classes

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=1*, *value=<function <lambda> at 0x402e410>*)
internal class to encapsulate the logic used by holmium.core.Element & holmium.core.Elements

holmium.core.pageobject.**enhanced**(*web_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)

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

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': { 'url': 'http://localhost'
                    , 'path':"{{PATH}}"
                    , 'login_url': '{{default.url}}/{{holmium.environment}}/login'}
```

```
                    , 'username' : '{{holmium.environment}}user'}
        ,'production': {'password': 'sekret'}
        ,'development': {'password': 'password'}
}
```

Python

```
config = {
    {
        'default': { 'url': 'http://localhost'
                    , 'path':"{{PATH}}"
                    , 'login_url': '{{default.url}}/{{holmium.environment}}/login'}
                    , 'username' : '{{holmium.environment}}user'}
        ,'production': {'password': 'sekret'}
        ,'development': {'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'
- `self.config['url']` will always return 'http://localhost'

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://localhost/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://localhost/development/login`

CHAPTER 5

Development

5.1 Contributors

- Sajnikanth Suriyanarayanan sajnikanth@gmail.com, Testing and initial integration and evaluation.
- Wilsen Davil wilsen.davil@gmail.com, original Holmium logo
- Alejandro Perez alejandro@pernixdata.com, Section(s) design.

5.2 Project Resources

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

Code The code is hosted on github.

Bugs, Feature Requests Tracked at the issue tracker.

Questions Freenode irc server : #holmium

5.3 Installation

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
```

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
```

CHAPTER 6

History

6.1 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

6.2 0.2 2013-09-11

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

6.3 0.1.8.4 2013-09-04

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

6.4 0.1.8.3 2013-08-12

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

6.5 0.1.8 2013-03-18

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

CHAPTER 7

License

Python Module Index

h