behave-webdriver Documentation

Release 0.0.1a

Spencer Young

Jun 14, 2022

Contents

CHAPTER 1

behave-webdriver

behave-webdriver is a step library intended to allow users to easily run browser automation tests (via selenium) with the behave BDD testing framework.

Inspired by, the webdriverio cucumber-boilerplate project.

1.1 Installation

behave-webdriver can be installed via `pip`.

```
pip install behave-webdriver
```

1.1.1 Using webdrivers

Selenium requires that you provide executables for the webdriver you want to use. Further, unless you specify the path to the binary explicitly, selenium expects that this executable is in PATH. See these driver installation notes for more details.

1.2 Quickstart

Ready to get started testing? This page will give you a quick introduction to behave-webdriver and how to use it. This assumes you have installed behave-webdriver and a webdriver on PATH. We also assume you got at least some familiarity with BDD/behave. If you're brand new to BDD in Python, you may want to check out the behave docs first.

Basic usage of this library with behave requires the following steps:

1. import the step implementations
2. set the `behave_driver` attribute on the behave `context` in your `environment.py` file.

3. write your feature file
4. run `behave`

1.2.1 Importing the step implementations

In order for your feature file steps to match our step implementations, behave needs to find them in your project. This is as simple as importing our step definitions into your own step implementation file.

```python
# features/steps/webdriver_example.py
from behave_webdriver.steps import *
```

For more information about step implementations, see the behave tutorial.

1.2.2 Set behave_driver in the environment

Our step implementations specifically look at the behave context for a `behave_driver` attribute to use to run your tests. In order for that to work, you'll have to provide this attribute in your `environment.py` file.

```python
# features/environment.py
import behave_webdriver

def before_all(context):
    context.behave_driver = behave_webdriver.Chrome()

def after_all(context):
    # cleanup after tests run
    context.behave_driver.quit()
```

The webdriver classes provided by behave-webdriver inherit from selenium's webdriver classes, so they will accept all same positional and keyword arguments that selenium accepts.

Some webdrivers, such as Chrome, provide special classmethods like `Chrome.headless` which instantiates `Chrome` with options to run headless. This is useful, for example in headless testing environments.

```python
def before_all(context):
    context.behave_driver = behave_webdriver.Chrome.headless()
```

In the future, behave-webdriver will provide fixtures for the setup and teardown of webdrivers. See the behave tutorial for more information about environment controls .

1.2.3 Writing the feature file

```gherkin
# my-minimal-project/features/myFeature.feature
Feature: Sample Snippets test
As a developer
I should be able to use given text snippets

Scenario: open URL
    Given the page url is not "http://webdriverjs.christian-bromann.com/"
    And   I open the url "http://webdriverjs.christian-bromann.com/"
    Then  I expect that the url is "http://webdriverjs.christian-bromann.com/"
    And   I expect that the url is not "http://google.com"
```

```
Scenario: click on link
    Given the title is not "two"
    And   I open the url "http://webdriverjs.christian-bromann.com/"
    When  I click on the link "two"
    Then  I expect that the title is "two"
```

1.2.4 Run behave

Then run the tests, just like any other behave test

```
behave
```

You should then see an output as follows:

```
Feature: Sample Snippets test # features/myFeature.feature:2
  As a developer
  I should be able to use given text snippets
  Scenario: open URL                                                         #␣
↪features/myFeature.feature:6
    Given the page url is not "http://webdriverjs.christian-bromann.com/"     # ../../
↪behave_webdriver/steps/given.py:136 0.012s
    And I open the url "http://webdriverjs.christian-bromann.com/"            # ../../
↪behave_webdriver/steps/given.py:10 1.414s
    Then I expect that the url is "http://webdriverjs.christian-bromann.com/" # ../../
↪behave_webdriver/steps/then.py:102 0.007s
    And I expect that the url is not "http://google.com"                      # ../../
↪behave_webdriver/steps/then.py:102 0.007s

  Scenario: click on link                                          # features/
↪myFeature.feature:13
    Given the title is not "two"                                   # ../../behave_
↪webdriver/steps/given.py:81 0.006s
    And I open the url "http://webdriverjs.christian-bromann.com/" # ../../behave_
↪webdriver/steps/given.py:10 0.224s
    When I click on the link "two"                                 # ../../behave_
↪webdriver/steps/when.py:21 0.622s
    Then I expect that the title is "two"                          # ../../behave_
↪webdriver/steps/then.py:10 0.006s

1 feature passed, 0 failed, 0 skipped
2 scenarios passed, 0 failed, 0 skipped
8 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m2.298s
```

Congratulations, you've just implemented a behavior-driven test without having to write a single step implementation!

1.3 behave-webdriver API Reference

This reference is meant for those who want to develop upon, extend, or alter the behavior of behave-webdriver. This will contain information regarding the implementation of various methods. Many aspects of the BehaveDriver class deal closely with selenium webdriver instances, but this document will refrain from duplicating information that should be contained in the selenium documentation.

behave-webdriver is designed with **you** in-mind. You are free to extend the behavior of our webdriver classes to suit your unique needs. You can subclass our webdriver classes, use a custom selenium webdriver, write your own mixin, or use a mixin somebody else provides for selenium.

Warning: While every effort is made to not make breaking changes, until a stable release, expect some things here to change, including breaking changes.

1.3.1 The webdriver classes

behave-webdriver provides each of the same webdriver classes provided in `selenium.webdriver`. Each class inherits from the *BehaveDriverMixin* mixin as well as the respective `selenium` counterpart class.

class `behave_webdriver.`**Chrome**(**args*, ***kwargs*)
Chrome driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**Firefox**(**args*, ***kwargs*)
Firefox driver class. Alternate constructors and browser-specific logic is implemented here.

> **doubleclick_element**(*element*)
> Overrides the doubleclick method to first scroll to element, and adds JS shim for doubleclick

class `behave_webdriver.`**Ie**(**args*, ***kwargs*)
Ie driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**Safari**(**args*, ***kwargs*)
Safari driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**PhantomJS**(**args*, ***kwargs*)
PhantomJS driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**Edge**(**args*, ***kwargs*)
Edge driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**Opera**(**args*, ***kwargs*)
Opera driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**BlackBerry**(**args*, ***kwargs*)
BlackBerry driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**Android**(**args*, ***kwargs*)
Android driver class. Alternate constructors and browser-specific logic is implemented here.

class `behave_webdriver.`**Remote**(**args*, ***kwargs*)
Remote driver class. Alternate constructors and browser-specific logic is implemented here.

1.3.2 The BehaveDriverMixin

The mixin class implements all of the general logic. If you want to alter how behave-webdriver behaves, this is probably the place to do it.

class `behave_webdriver.driver.`**BehaveDriverMixin**(**args*, ***kwargs*)
Implements most of the general (I.E. not browser-specific) logic for step implementations.

Intended to be used with subclasses of any selenium webdriver.

```
>>> from behave_webdriver.driver import BehaveDriverMixin
>>> from somewhere import SomeDriver
>>> class MyBehaveDriver(BehaveDriverMixin, SomeDriver):
...     pass
>>> behave_driver = MyBehaveDriver()
>>> behave_driver.get('https://github.com/spyoungtech/behave-webdriver')
```

Can also be used with other mixins designed for selenium, such as selenium-requests

```
>>> from behave_webdriver.driver import BehaveDriverMixin
>>> from seleniumrequests import RequestMixin
>>> from selenium import webdriver
>>> class BehavingRequestDriver(BehaveDriverMixin, RequestMixin, webdriver.
↪Chrome):
...     pass
>>> behave_driver = BehavingRequestDriver()
>>> response = behave_driver.request('GET', 'https://github.com/spyoungtech/
↪behave-webdriver')
```

alert

Property shortcut for an `Alert` object for the driver Note: this will return an Alert instance regardless of whether or not there is actually an alert present. Use `has_alert` to check whether or not there is an alert currently present.

Returns an selenium.webdriver.common.alert.Alert instance

click_element(*element*)

Click on an element. Note: this will not trigger some doubleclick events, even when n=2 with any delay. Instead, if you want to doubleclick, use *doubleclick_element*

Parameters **element** (*str*) – CSS Selector or XPATH used to locate the element

click_link_text(*text*, *partial=False*)

Click on a link, located by matching the text contained in the link. If `partial` is True, the link is located by partial text.

Parameters

- **text** (*str*) – The text contained in the link, used to locate the element.
- **partial** (*bool*) – Whether or not to match link by partial text (as opposed to full match)

Returns

cookies

Shortcut for driver.get_cookies()

doubleclick_element(*element*)

Double click an element

Parameters **element** (*str*) – CSS Selector or XPATH used to locate the element

Returns

drag_element(*element*, *to_element*)

Drag an element to the location of another element.

Parameters

- **element** (*str*) – CSS Selector or XPATH used to locate the element
- **to_element** (*str*) – the selector used to locate the destination element

Returns

`element_contains`(*element*, *value*)
Checks if an element contains (in value/text) a given string/value

Parameters

- **`element`** (*`str`*) – CSS Selector or XPATH used to locate the element
- **`value`** (*`str`*) – the text/value to check for

Returns True or False, whether or not the value was found in the element.

Return type bool

`element_enabled`(*element*)
Checks if an element is enabled or not.

Parameters **`element`** (*`str`*) – CSS Selector or XPATH used to locate the element

Returns True if the element is enabled, else False

Return type bool

`element_exists`(*element*)
Whether or not an element exists. Attempts to locate the element using *get_element* returns True if the element was found, False if it couldn't be located.

Parameters **`element`** (*`str`*) – CSS Selector or XPATH used to locate the element

Returns True if the element could be found, False if it couldn't be found

Return type bool

`element_has_class`(*element*, *cls*)
Checks whether or not an element has a particular css class.

Parameters

- **`element`** (*`str`*) – CSS Selector or XPATH used to locate the element
- **`cls`** (*`str`*) – The css class to check for

Returns True if the element has the specified class, else False

Return type bool

`element_in_viewport`(*element*)
Determines the bounding box (rect) of the window and rect of the element. This information is used to determine whether or not the element is *completely* within the viewport.

Parameters **`element`** – CSS Selector or XPATH used to locate the element

Returns

`element_selected`(*element*)
Checks if an element is selected or not.

Parameters **`element`** (*`str`*) – CSS Selector or XPATH used to locate the element

Returns True if the element is selected, else False

Return type bool

`element_visible`(*element*)
Checks if an element is visible or not.

Parameters **`element`** (*`str`*) – CSS Selector or XPATH used to locate the element

Returns True if the element is visible, else False

Return type bool

get_element (*selector*, *by=None*)

Takes a selector string and uses an appropriate method (XPATH or CSS selector by default) to find a WebElement The optional *by* argument can be supplied to specify any locating method explicitly. This is used to resolve selectors from step definition strings to actual element objects

Parameters

- **selector** (*str*) – The selector to use, an XPATH or CSS selector
- **by** – alternate method used to locate element, e.g. (By.id) See selenium.webdriver.common.by.By attributes

Returns WebElement object

get_element_attribute (*element*, *attr*, *css=False*, *expected_value=None*)

Get the value of an attribute or css attribute from an element.

Parameters

- **element** (*str*) – CSS Selector or XPATH used to locate the element
- **attr** (*str*) – The attribute to lookup
- **css** (*bool*) – Whether or not this is a CSS atrribute
- **expected_value** –

Returns The value of the attribute

get_element_location (*element*)

Gets the location of the element in the renderable canvas. This is a dict with two keys: 'x' and 'y'

Parameters **element** (*str*) – CSS Selector or XPATH used to locate the element

Returns the element's location

Return type dict

get_element_size (*element*)

Returns a dictionary containing the size information of an element. The dictionary has two keys: 'width' and 'height' which represent the size of the element dimensions in px

Parameters **element** (*str*) – CSS Selector or XPATH used to locate the element

Returns A dictionary with size information

Return type dict

get_element_text (*element*)

Takes in a selector, finds the element, and extracts the text. When present on the WebElement, the element's 'value' property is returned. (For example, this is useful for getting the current text of Input elements) If the element has no 'value' property, the containing text is returned (elem.text)

Parameters **element** (*str*) – CSS Selector or XPATH used to locate the element

Returns the text contained within the element.

Return type str

has_alert

Whether or not there is currently an alert present

Returns True if there is an alert present, else False

Return type bool

static is_color(*str_*)

Whether or not the string represents a color.

Parameters **str** –

Returns

move_to_element(*element*, *offset=None*)

Moves the mouse to the middle of an element

Parameters

- **element** (`str`) – CSS Selector or XPATH used to locate the element
- **offset** (`tuple`) – optional tuple of x/y offsets to offset mouse from center

Returns

open_url(*url*)

Navigate to an absolute URL Behaves same as `driver.get` but serves as a common entry-point for subclasses wanting to change this.

Parameters **url** (`str`) – an absolute URL including the scheme

Returns

pause(*milliseconds*)

Pause for a number of miliseconds. `time.sleep` is used here due to issues with w3c browsers and ActionChain pause feature.

Parameters **milliseconds** (`int`) – number of miliseconds to wait

Returns

press_button(*button*)

Send a keystroke simulating the press of a given button. You can use keys as strings (e.g. 'a', 'z') or any key names (e.g. the 'escape' key). When the length of the button argument is greater than one character, names are checked against selenium.webdriver.common.keys.Keys first.

Parameters **button** (`str`) – A single character or key name

Returns

primary_handle

shortcut for window_handles[0]

Returns the primary (first) window handle

screen_size

Property for the current driver window size. Can also be set by assigning an x/y tuple.

Returns tuple of the screen dimensions (x, y)

scroll_to(*x*, *y*)

Scroll to a particular (x, y) coordinate.

Parameters

- **x** (`int`) – the x coordinate to scroll to.
- **y** (`int`) – the y coordinate to scroll to.

Returns

scroll_to_bottom()
Scrolls the current window to the bottom of the window (0, document.body.scrollHeight).

scroll_to_element(*element*)
Scroll to the location of an element.

Parameters **element** – CSS Selector or XPATH used to locate the element

Returns

secondary_handles
shortcut for window_handles[1:]

Returns list of window handles

Return type list

select_option(*select_element*, *by*, *by_arg*)
Implements features for selecting options in Select elements. Uses selenium's `Select` support class.

Parameters

- **select_element** – CSS Selector or XPATH used to locate the select element containing options
- **by** (*str*) – the method for selecting the option, valid options include any select_by_X supported by `Select`.

Returns

send_keys(*keys*)
Send arbitrary keys. Note: this is different than sending keys directly to an element.

Parameters **keys** – keys to send

Returns

submit(*element*)
Shortcut for submitting an element

Parameters **element** (*str*) – CSS Selector or XPATH used to locate the element

Returns

wait_for_element_condition(*element*, *ms*, *negative*, *condition*)
Wait on an element until a certain condition is met, up to a maximum amount of time to wait.

Parameters

- **element** – CSS Selector or XPATH used to locate the element
- **ms** – maximum time (in milliseconds) to wait for the condition to be true
- **negative** – whether or not the check for negation of condition. Will coarse boolean from value
- **condition** – the condition to check for. Defaults to checking for presence of element

Returns element

1.4 List of predefined steps

1.4.1 Given Steps

- `I open the site "([^"]*)?"`
- `I open the url "([^"]*)?"`
- `I have a screen that is ([\d]+) by ([\d]+) pixels`
- `I have a screen that is ([\d]+) pixels (broad|tall)`
- `I have closed all but the first (window|tab)`
- `I pause for (\d+)*ms`
- `a (alertbox|confirmbox|prompt) is( not)* opened`
- `the base url is "([^"]*)?"`
- `the checkbox "([^"]*)?" is( not)* checked`
- `the cookie "([^"]*)?" contains( not)* the value "([^"]*)?"`
- `the cookie "([^"]*)?" does( not)* exist`
- `the element "([^"]*)?" contains( not)* the same text as element "([^"]*)?"`
- `the element "([^"]*)?" is( not)* ([\d]+)px (broad|tall)`
- `the element "([^"]*)?" is( not)* empty`
- `the element "([^"]*)?" is( not)* enabled`
- `the element "([^"]*)?" is( not)* positioned at ([\d]+)px on the (x|y) axis`
- `the element "([^"]*)?" is( not)* selected`
- `the element "([^"]*)?" is( not)* visible`
- `the element "([^"]*)?"( not)* contains any text`
- `the element "([^"]*)?"( not)* contains the text "([^"]*)?"`
- `the element "([^"]*)?"( not)* matches the text "([^"]*)?"`
- `the page url is( not)* "([^"]*)?"`
- `the title is( not)* "([^"]*)?"`
- `the( css)* attribute "([^"]*)?" from element "([^"]*)?" is( not)* "([^"]*)?"`
- `there is (an|no) element "([^"]*)?" on the page`

1.4.2 When Steps

- `I open the site "([^"]*)?"`
- `I open the url "([^"]*)?"`
- `I accept the (alertbox|confirmbox|prompt)`
- `I add "{value}" to the inputfield "{element}"`
- `I clear the inputfield "{element}"`

- `I click on the button "{element}"`
- `I click on the element "{element}"`
- `I click on the link "{link_text}"`
- `I close the last opened (tab|window)`
- `I delete the cookie "{cookie_key}"`
- `I dismiss the (alertbox|confirmbox|prompt)`
- `I doubleclick on the element "{element}"`
- `I drag element "{from_element}" to element "{to_element}"`
- `I enter "([^"]*)?" into the (alertbox|confirmbox|prompt)`
- `I focus the last opened (tab|window)`
- `I move to element "{element}" with an offset of {x_offset:d},{y_offset:d}`
- `I move to element "{element}"`
- `I pause for {milliseconds:d}ms`
- `I press "{key}"`
- `I scroll to element "{element}"`
- `I select the option with the (text|value|name) "([^"]*)?" for element "([^"]*)?"`
- `I select the {nth} option for element "{element}"`
- `I set "{value}" to the inputfield "{element}"`
- `I set a cookie "{cookie_key}" with the content "{value}"`
- `I submit the form "{element}"`

1.4.3 Then Steps

- `I expect the screen is ([\d]+) by ([\d]+) pixels`
- `I expect a new (window|tab) has( not)* been opened`
- `I expect that a (alertbox|confirmbox|prompt) is( not)* opened`
- `I expect that a (alertbox|confirmbox|prompt)( not)* contains the text "([^"]*)?"`
- `I expect that checkbox "([^"]*)?" is( not)* checked`
- `I expect that cookie "([^"]*)?"( not)* contains "([^"]*)?"`
- `I expect that cookie "([^"]*)?"( not)* exists`
- `I expect that element "([^"]*)?" (has|does not have) the class "([^"]*)?"`
- `I expect that element "([^"]*)?" becomes( not)* visible`
- `I expect that element "([^"]*)?" does( not)* exist`
- `I expect that element "([^"]*)?" is( not)* ([\d]+)px (broad|tall)`
- `I expect that element "([^"]*)?" is( not)* empty`
- `I expect that element "([^"]*)?" is( not)* enabled`

- `I expect that element "([^"]*)?" is( not)* focused`
- `I expect that element "([^"]*)?" is( not)* positioned at ([\d]+)px on the (x|y) axis`
- `I expect that element "([^"]*)?" is( not)* selected`
- `I expect that element "([^"]*)?" is( not)* visible`
- `I expect that element "([^"]*)?" is( not)* within the viewport`
- `I expect that element "([^"]*)?"( not)* contains any text`
- `I expect that element "([^"]*)?"( not)* contains the same text as element "([^"]*)?"`
- `I expect that element "([^"]*)?"( not)* contains the text "([^"]*)?"`
- `I expect that element "([^"]*)?"( not)* matches the text "([^"]*)?"`
- `I expect that the path is( not)* "([^"]*)?"`
- `I expect that the title is( not)* "([^"]*)?"`
- `I expect that the url is( not)* "([^"]*)?"`
- `I expect that the( css)* attribute "([^"]*)?" from element "([^"]*)?" is( not)* "([^"]*)?"`
- `I expect the url "([^"]*)?" is opened in a new (tab|window)`
- `I expect the url to( not)* contain "([^"]*)?"`
- `I wait on element "([^"]*)?"(?: for (\d+)ms)*(?: to( not)* (be checked|be enabled|be selected|be visible|contain a text|contain a value|exist))*`

1.5 Advanced usage; extending behave-webdriver

behave-webdriver is designed with **you** in-mind. You are free to extend the behavior of our webdriver classes to suit your unique needs. You can subclass our webdriver classes, use a custom selenium webdriver, write your own mixin, or use a mixin somebody else provides for selenium.

1.5.1 Example: selenium-requests

selenium-requests is a preexisting project that adds functionality of the popular `requests` library to selenium. It is simple to use `selenium-requests` with behave-webdriver. The following, and other examples, are available in the repo `examples` directory and in the full documentation.

```python
# examples/selenium-requests/features/environment.py
from selenium import webdriver # or any custom webdriver
from behave_webdriver.driver import BehaveDriverMixin
from seleniumrequests import RequestMixin # or your own mixin

class BehaveRequestDriver(BehaveDriverMixin, RequestMixin, webdriver.Chrome):
    pass

def before_all(context):
    context.behave_driver = BehaveRequestDriver()
```

```
# examples/selenium-requests/features/steps/selenium_steps.py
from behave import *
from behave_webdriver.steps import *
from urllib.parse import urljoin

@given('I send a {method} request to the page "{page}"')
def send_request_page(context, method, page):
    url = urljoin(context.base_url, page)
    context.response = context.behave_driver.request(method, url)

@then('I expect the response text contains "{text}"')
def check_response_text_contains(context, text):
    assert text in context.response.text
```

```
# examples/selenium-requests/features/selenium-requests.feature
Feature: Using selenium-requests
  As a developer
  I should be able to extend behave-webdriver with selenium-requests

  Scenario: use selenium-requests with behave-webdriver
    # use a behave-webdriver step
    Given the base url is "http://127.0.0.1:8000"
    # use your own steps using selenium-requests features
    Given I send a GET request to the page "/"
    Then I expect the response text contains "<h1>DEMO APP</h1>"
```

Assuming you're in the repository root (and have the demo app running) just run like any other project with `behave`

Results

```
(behave-webdriver) $ behave examples/selenium-requests/features

DevTools listening on ws://127.0.0.1:12646/devtools/browser/1fe75b44-1c74-49fa-8e77-
↪36c54d50cd24
Feature: Using selenium-requests # examples/selenium-requests/features/requests.
↪feature:1
  As a developer
  I should be able to extend behave-webdriver with selenium-requests
  Scenario: use selenium-requests with behave-webdriver          # examples/selenium-
↪requests/features/requests.feature:6
    Given the base url is "http://127.0.0.1:8000"                # behave_webdriver/
↪steps/actions.py:162
    Given I send a GET request to the page "/"                   # examples/selenium-
↪requests/features/steps/selenium_steps.py:11
    Then I expect the response text contains "<h1>DEMO APP</h1>" # examples/selenium-
↪requests/features/steps/selenium_steps.py:17

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m1.385s
```

1.6 Browser Support

behave-webdriver is designed so that you *can* use any of the webdriver classes you would normally use with Selenium, e.g. `Chrome`, `Firefox`, `Remote`, etc… However, not all browsers were made equal and attempting to get identical behavior across browsers is… complicated, if not impossible.

This document will aim to describe the status of support with the various webdrivers supported by Selenium. Where there are known issues or quirks related to this step library, there will be an effort to document them here, too. Be sure to also checkout the github issues and projects. browser-specific issues should be tagged accordingly.

More specifics may be revealed in the *behave-webdriver API Reference* and in the source. The ecosystem around selenium/webdrivers is huge. This is not a repository or body of knowledge for all driver-related issues; just the ones that most directly affect this library.

Unless otherwise noted, we are referring to the latest stable release of Selenium and each respective browser and driver. Keep in mind, this documentation may not necessarily be up-to-date with very recent releases.

1.6.1 Chrome (recommended)

Currently, Chrome is essentially the reference implementation. We primarily discuss issues with other webdrivers with respect to how Chrome behaves. In our experience so far, Chrome is the fastest and most well-behaving driver.

We recommend Chrome and fully support the use of the Chrome webdriver with the latest versions of selenium and chrome/chromedriver. At the time of this writing (March 2018) that's selenium 3.10, Chrome 65, and chromedriver 2.36 While earlier versions should work fine and we are willing to support them, they are not tested.

1.6.2 Firefox (beta)

Firefox is officially supported as of v0.1.1

Known issues

- `submit` on form elements is implemented by a (Selenium) JS shim and will not block for page load. Clicking the form button should block properly, however.
- support for window handles is somewhat problematic
- clicking elements requires they are in the viewport (we compensate for this by scrolling to an element before any click)
- moving to an element *with an offset* that is bigger than the viewport is not (yet) supported
- slower than Chrome

Workarounds/Shims

Shims and other workarounds for some known issues are implemented in the Firefox class.

See *behave-webdriver API Reference* for more details.

1.6.3 Ie

We have some preliminary support for Internet Explorer. It is tested in our appveyor CI build.

1.6.4 Safari

We have some preliminary support for Safari on OSX/Mac OS. It is tested as part of our Travis CI build (failures currently allowed).

1.6.5 PhantomJS

Danger: Selenium support for PhantomJS has been deprecated and the PhantomJS development has been suspended. As such, users are recommended to NOT use PhantomJS to begin with.

PhantomJS is a low priority (see above). Users should expect issues with PhantomJS when using modern versions of selenium, and

Known issues

- No support for alerts/modals
- Cookies are problematic (cookies must have domain (and expiry?); setting cookies for localdomain not supported)
- Memory-hungry
- Unsupported (see above)

1.6.6 Remote

Remote is untested at this time.

1.6.7 Edge

Edge is untested at this time.

1.6.8 Opera

Opera is untested at this time.

1.6.9 BlackBerry

BlackBerry is untested at this time.

1.6.10 Android

Android is untested at this time.

1.7 Roadmap

Loosely organized collection of goals/milestones and ideas. Nothing here is necessarily concrete, but should give you an idea of where our heads are at for the development of behave-webdriver.

You can help steer our roadmap in the right direction with suggestions, feedback, and other contributions. Please don't hesitate to raise an issue on Github.

1.7.1 Immediate and Short Term

Immediate and short term goals are some milestones that we are actively working on or in our immediate forefront for development. Ideally, these things have clearly defined requirements and some work in progress.

Documentation; recipes & tutorials

While documentation is something we'll be working on perpetually through development, we are particularly motivated on immediately providing at least some brief tutorials and recipes.

1.7.2 Medium Term

Medium term goals are things we are committed to working on and implementing in the not-so-distant future. Ideally, this means we're working on these things passively and have at least a basic plan for the implementation.

Device emulation

We want to provide support for steps that use device emulation features of drivers that support this. E.g. steps like `Given I am using an iPhone 6`, `Given I am using a Pixel 2`, etc.

More step definitions

We plan to implement additional step definitions to perform more actions with selenium and provide more robust interfaces for testing and automation. This will include things like taking & saving screenshots, retrieving/saving page source, and more.

If you have ideas for step definitions you'd like to see implemented, raise an issue on Github. These contributions are welcomed and very much appreciated.

Browser support (others)

Chrome and Firefox are in our forefront for browser support. We do however plan to test and provide best-effort support for all the webdrivers supported by selenium.

We hope to get all browsers tested (but not necessarily passing) and attempt to make note of compatibility, behavior differences, and other browser-specific quirks.

Would be nice to have more browsers tested in the CI builds as well.

See *Browser Support* for more information.

1.7.3 Long Term, ongoing, and Ideas

These are some loose long-term milestones or ideas (which may or may not materialize) we have for the future. These are things that we would probably like to do, but have probably not put much effort into implementation or detailed plans. Anything we are remotely considering, but have not committed to, will be here, too.

Assertion matcher

Currently, standard python assertions are used. In the future, we may opt to use an assertion matching library such as pyhamcrest. Some time and effort will need to be put in to research a good choice in this area.

Parallel support

While behave itself is planning to add parallel runner support in the future, its unlikely this will work well for browser testing. As such, we have this parallel support in the back of our minds, but it will probably be some time before it is introduced in a stable release.

Use of other selenium libraries

It may be possible for us to take advantage of previous work in this area, for example requestium, to enhance behave-webdriver. We want to explore these possibilities.

Survey & reflection - path to a stable release

While a stable (LTS) release itself is more of a long-term goal, we are constantly surveying the behave landscape and reviewing our API. We've made (what we feel are good) decisions in the design of behave-webdriver, but there's always room for improvement. Now particularly is a good time for us to ensure we are laying down a solid foundation to build upon for the future.

Your feedback is immensely valuable in this regard and is sincerely appreciated. The best way to make suggestions or general comments is to raise an issue on Github.

Better tests (ongoing)

Our github README boasts its coverage with a shiny badge from coveralls. The truth is that coverage isn't everything. There's undoubtedly cases where functionality is broken or doesn't work quite as expected. We want to find those to build better test cases, and improve the functionality of the library as a whole.

1.7.4 Completed

Browser support (Firefox)

Firefox is officially supported as of v0.1.1

v0.1.0

Complete™ support with Google Chrome. We use the feature files (with modifications or additons in some cases) from cucumber-boilerplate as acceptance tests. While this is bound to be imperfect, it's a great start for v0.1

1.7.5 Deferred

Deferred items are things we previously comitted to but, for some reason or another, have placed on the backburner or suspended entirely.

PhantomJS support

While we will continue to provide best-effort support for all browsers, including PhantomJS, because PhantomJS has been deprecated for selenium and phantomJS development has been suspended, PhantomJS is now a low priority.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

2.1 Goals

- Make writing readable browser automation tests as Gherkin features easy.
- Provide an easily extensible interface to the selenium driver (`BehaveDriver`)
- To be (at least mostly) compatible with feature files written for webdriverio/cucumber-boilerplate

2.2 Status

We currently test against Python2.7 and Python3.5+ using headless chrome. While all selenium's webdrivers are provided, they are not all supported at this time. We plan to add support for additional browsers in the future.

Index