Aloe Webdriver Documentation

Release 0.5.12.dev0+g7efdcad.d20181028

Alexey Kotlyarov

Oct 28, 2018

Contents

aloe_webdriver provides utilities to help write Aloe BDD tests that run in the browser using Selenium webdriver.

CHAPTER 1

Basic Steps

```
import aloe_webdriver
```

1.1 Navigation

Step I visit "(.*?)"$
Navigate to the provided (fully qualified) URL.

Step The browser's URL should be "([^"]*)"$
Assert the absolute URL of the browser is as provided.

Step The browser's URL should contain "([^"]*)"$
Assert the absolute URL of the browser contains the provided.

Step The browser's URL should not contain "([^"]*)"$
Assert the absolute URL of the browser does not contain the provided.

Step The page title should be "([^"]*)"
Assert the page title matches the given text.

1.2 Text

Step I should see "([^"]+)"$
Assert provided text is visible.

Be aware this text could be anywhere on the screen. Also be aware that it might cross several HTML nodes. No determination is made between block and inline nodes. Whitespace can be affected.

Step I should see "([^"]+)" within (d+) seconds?$
Assert provided text is visible within n seconds.

Be aware this text could be anywhere on the screen. Also be aware that it might cross several HTML nodes. No determination is made between block and inline nodes. Whitespace can be affected.

Step I should not see "([^"]+)"$
Assert provided text is not visible.

Be aware that because of the caveats of the positive case, the text MAY be on the screen in a slightly different form.

1.3 Links

Step I click "([^"]*)"$
Click the link with the provided link text.

Step I should see a link with the url "([^"]*)"$
Assert a link with the provided URL is visible on the page.

Step I should see a link that contains the text "([^"]*)" and the url "([^"]*)"$
Assert a link containing the provided text points to the provided URL.

CHAPTER 2

Forms

```
import aloe_webdriver
```

Steps for referring to form controls typically support three methods to identify the control:

1. The label of the control. This is the recommended way to refer to a control as it is the most descriptive.
2. The control's `name`. This can be used if you have multiple controls with the same label (i.e. in formsets).
3. The control's `id`.

`Step` I should see a form that goes to "([^"]*)"$
Assert the existence of a HTML form that submits to the given URL.

`Step` I click on label "([^"]*)"
Click on the given label.

On a correctly set up form this will highlight the appropriate field.

`Step` I submit the only form
Look for a form on the page and submit it.

Asserts if more than one form exists.

`Step` I submit the form with action "([^"]*)"
Submit the form with the given action URL (i.e. the form that submits to `/post/my/data`).

2.1 Text Fields

`Step` I fill in "([^"]*)" with "([^"]*)"$
Fill in the HTML input with given label (recommended), name or id with the given text.

Supported input types are text, textarea, password, month, time, week, number, range, email, url, tel and color.

`Step` Input "([^"]*)" (?:has|should have) value "([^"]*)"
Assert the form input with label (recommended), name or id has given value.

2.2 Buttons

Step I press "([^"]*)"$
Click the button with the given label.

2.3 Checkboxes

Step I check "([^"]*)"$
Check the checkbox with label (recommended), name or id.

Step I uncheck "([^"]*)"$
Uncheck the checkbox with label (recommended), name or id.

Step The "([^"]*)" checkbox should be checked$
Assert the checkbox with label (recommended), name or id is checked.

Step The "([^"]*)" checkbox should not be checked$
Assert the checkbox with label (recommended), name or id is not checked.

2.4 Radio Buttons

Step I choose "([^"]*)"$
Click (and choose) the radio button with the given label (recommended), name or id.

Step The "([^"]*)" option should be chosen$
Assert the radio button with the given label (recommended), name or id is chosen.

Step The "([^"]*)" option should not be chosen$
Assert the radio button with the given label (recommended), name or id is not chosen.

2.5 Selects (Comboboxes)

Step I select "([^"]*)" from "([^"]*)"$
Select the named option from select with label (recommended), name or id.

Step I select the following from "([^"]*?)":?$
Select multiple options from select with label (recommended), name, or id. Pass a multiline string of options. e.g.

```
When I select the following from "Contact Methods":
    """
    Email
    Phone
    Fax
    """
```

Step The "([^"]*)" option from "([^"]*)" should be selected$
Assert the given option is selected from the select with label (recommended), name or id.

If multiple selections are supported other options may be selected.

Step The following options from "([^"]*?)" should be selected:?$

`Step` I should see option "([^"]*)" in selector "([^"]*)"
Assert the select contains the given option.

`Step` I should not see option "([^"]*)" in selector "([^"]*)"
Assert the select does not contain the given option.

CHAPTER 3

Other Steps

3.1 Alerts

```
import aloe_webdriver
```

Validate the behaviour of popup alerts.

Step I should see an alert with text "([^"]*)"
Assert an alert is showing with the given text.

Step I should not see an alert
Assert there is no alert.

Step I accept the alert
Accept the alert.

Step I dismiss the alert
Dismiss the alert.

3.2 Tooltips

```
import aloe_webdriver
```

Step I should see an element with tooltip "([^"]*)"
Assert an element with the given tooltip (title) is visible.

N.B. tooltip may not be visible.

Step I should not see an element with tooltip "([^"]*)"
Assert an element with the given tooltip (title) is not visible.

Step I (?:click|press) the element with tooltip "([^"]*)"
Click on a HTML element with a given tooltip.

This is very useful if you're clicking on icon buttons, etc.

3.3 Checks based on HTML `id`

```
import aloe_webdriver
```

Using the HTML `id` is generally considered bad BDD, but sometimes it is the only way to unambiguously refer to an element. It is strongly recommended to find a more behavioral mechanism to describe your test. See *Writing good BDD steps*.

`Step` The element with id of "([^"]*)" contains "([^"]*)"$
Assert provided content is contained within an element found by `id`.

`Step` The element with id of "([^"]*)" does not contain "([^"]*)"$
Assert provided content is not contained within an element found by `id`.

`Step` I should see an element with id of "([^"]*)"$
Assert an element with the given `id` is visible.

`Step` I should see an element with id of "([^"]*)" within (d+) seconds?$
Assert an element with the given `id` is visible within n seconds.

`Step` I should not see an element with id of "([^"]*)"$
Assert an element with the given `id` is not visible.

`Step` I submit the form with id "([^"]*)"
Submit the form with given id (used to disambiguate between multiple forms).

3.3.1 Focus

`Step` Element with id "([^"]*)" should be focused
Assert the element is focused.

`Step` Element with id "([^"]*)" should not be focused
Assert the element is not focused.

3.3.2 Frames

Use these steps to switch frames if you need to work in a different frame or iframe. It is recommended you wrap these steps up in a more behavioural description. See *Writing good BDD steps*.

`Step` I switch back to the main view
Swap Selenium's context back to the main window.

3.4 CSS Selectors

```
import aloe_webdriver.css
```

Steps for selecting elements using CSS selectors.

Like with steps based on HTML id, these steps should be used cautiously to avoid creating tests that do not describe the behaviours of your application. See *Writing good BDD steps*.

Note: Be aware these steps require jQuery. If jQuery is not present it will be added (v1.12).

`Step` I check $("(.*?)")$
Check the checkbox matching the CSS selector.

`Step` There should be an element matching $("(.*?)")$
Assert an element exists matching the given selector.

`Step` There should not be an element matching $("(.*?)")$
Assert an element does not exist matching the given selector.

`Step` I click $("(.*?)")$
Click the element matching the CSS selector.

`Step` There should be exactly (d+) elements matching $("(.*?)")$
Assert n elements exist matching the given selector.

`Step` I fill in $("(.*?)") with "(.*?)"$
Fill in the form element matching the CSS selector.

`Step` I follow the link $("(.*?)")$
Navigate to the href of the element matching the CSS selector.

N.B. this does not click the link, but changes the browser's URL.

`Step` $("(.*?)") should be selected$
Assert the option matching the CSS selector is selected.

`Step` I select $("(.*?)")$
Select the option matching the CSS selector.

`Step` I submit $("(.*?)")
Submit the form matching the CSS selector.

`Step` There should be an element matching $("(.*?)") within (d+) seconds?$
Assert an element exists matching the given selector within the given time period.

3.5 Screenshots

```
import aloe_webdriver.screenshot_failed
```

Hooks to save screenshots and HTML source of the pages when tests fail.

Assumes a browser instance is stored in `world.browser`.

Whenever a step fails, the screen shot and the HTML source of the page displayed in the browser are saved to the current directory. The file names include the feature file name, scenario number and name and, if applicable, the example number.

Consider the following feature:

```
# features/account.feature
Feature: Account management

    Scenario: Log in
        Given I open the site
        And I enter username and password
```

(continues on next page)

(continued from previous page)

```
        And I press "Log in"
        Then I should see "Logged in"
```

If there will be no “Logged in” text when expected, screenshot and the page source will be saved to, respectively:

```
failed_features_account_feature_1_Log_in.png
failed_features_account_feature_1_Log_in.html
```

To change the directory where the screenshots are saved, override the constant `DIRECTORY` as follows:

```
from aloe_webdriver import screenshot_failed

screenshot_failed.DIRECTORY = '/alternative/directory'
```

Note that the given directory should already exist.

CHAPTER 4

Django Integration

```
import aloe_webdriver.django
```

Django-specific extensions for use with aloe_django.

Step I visit site page "([^"]*)"

Visit the specific page of the site, e.g.

```
When I visit site page "/users"
```

CHAPTER 5

Writing good BDD steps

The tools provided in *Aloe-Webdriver* form a reasonably thin wrapper around Selenium and thus make it very easy to write *imperative* tests. While the occasional imperative test is useful, it is frequently more useful to abstract these into sub-steps of a more *declarative test*.

For example, take this example from the BBC essay: Tips for writing better feature files.

Here is a bad, imperative example:

```
Given I am on the login page
When I fill in "username" with "ABC"
And I fill in "password" with "XYZ"
And I checked the "Remember Me" checkbox
And I click on the "Submit" button
Then I should log into the system
And I should see "Welcome"
```

Instead a better, declarative example would be:

```
Given I have logged into the system
Then I should see "Welcome"
```

Use `step.behave_as()` to call the imperative steps from your own step abstracts the mechanics of your website into something more descriptive. This also makes it easier if you ever change the login process.

```
@step("I have logged into the system")
def i_log_in():
    '''Log in to the site'''
    step.behave_as('Given I am on the login page')
    step.behave_as('When I fill in "username" with "ABC"')
    step.behave_as('And I fill in "password" with "XYZ"')
    step.behave_as('And I checked the "Remember Me" checkbox')
    step.behave_as('And I click on the "Submit" button')
    step.behave_as('Then I should log into the system')
```

5.1 Step Writing Utilities

Aloe-Webdriver includes several utilities for writing Selenium tests.

class aloe_webdriver.util.**ElementSelector**(*browser*, *xpath=None*, *elements=None*, *filter_displayed=False*, *filter_enabled=False*)

A set of elements on a page matching an XPath query.

Parameters

- **browser** – world.browser
- **xpath** (*str*) – XPath query
- **elements** (*list*) – list of selenium.WebElement objects
- **filter_displayed** (*bool*) – whether to only return displayed elements
- **filter_enabled** (*bool*) – whether to only return enabled elements

Delays evaluation to batch the queries together, allowing operations on selectors (e.g. union) to be performed first, and then issuing as few requests to the browser as possible.

One of *xpath* or *elements* must be passed. Passing *xpath* creates a selector delaying evaluation until it's needed, passing *elements* stores the elements immediately.

Can behave as an iterable of elements or a single element by proxying all method calls, asserting that there is only one element selected.

Can be combined using the addition operator (+) to *OR* XPath queries together.

evaluated
Whether the selector has already been evaluated.

filter(*displayed=False*, *enabled=False*)
Filter elements by visibility and enabled status.

Parameters

- **displayed** – whether to filter out invisible elements
- **enabled** – whether to filter out disabled elements

Returns: an *ElementSelector*

aloe_webdriver.util.**element_id_by_label**(*browser*, *label*)
The ID of an element referenced by a *label's ''for''* attribute. The label must be visible.

Parameters

- **browser** – world.browser
- **label** – label text to return the referenced element for

Returns: for attribute value

aloe_webdriver.util.**find_any_field**(*browser*, *field_types*, *field_name*)
Find a field of any of the specified types.

Parameters

- **browser** – world.browser
- **field_types** (*list*) – a list of field type (i.e. *button*)
- **value** (*string*) – an id, name or label

Returns: an *ElementSelector*

See also: *find_field()*.

aloe_webdriver.util.**find_button**(*browser*, *value*)
Find a button with the given value.

Searches for the following different kinds of buttons:

<input type="submit"> <input type="reset"> <input type="button"> <input type="image"> <button> <{a,p,div,span,...} role="button">

Returns: an *ElementSelector*

aloe_webdriver.util.**find_field**(*browser*, *field_type*, *value*)
Locate an input field.

Parameters

- **browser** – world.browser
- **field_type** (*string*) – a field type (i.e. *button*)
- **value** (*string*) – an id, name or label

This first looks for *value* as the id of the element, else the name of the element, else as a label for the element.

Returns: an *ElementSelector*

aloe_webdriver.util.**find_field_by_id**(*browser*, *field_type*, *id*)
Locate the control input with the given id.

Parameters

- **browser** – world.browser
- **field_type** (*string*) – a field type (i.e. *button*)
- **id** (*string*) – id attribute

Returns: an *ElementSelector*

aloe_webdriver.util.**find_field_by_label**(*browser*, *field_type*, *label*)
Locate the control input that has a label pointing to it.

Parameters

- **browser** – world.browser
- **field_type** (*string*) – a field type (i.e. *button*)
- **label** (*string*) – label text

This will first locate the label element that has a label of the given name. It then pulls the id out of the 'for' attribute, and uses it to locate the element by its id.

Returns: an *ElementSelector*

aloe_webdriver.util.**find_field_by_name**(*browser*, *field_type*, *name*)
Locate the control input with the given name.

Parameters

- **browser** – world.browser
- **field_type** (*string*) – a field type (i.e. *button*)
- **name** (*string*) – name attribute

Returns: an *ElementSelector*

aloe_webdriver.util.**find_field_by_value**(*browser*, *field_type*, *name*)
Locate the control input with the given `value`. Useful for buttons.

Parameters

- **browser** – `world.browser`
- **field_type** (*string*) – a field type (i.e. *button*)
- **name** (*string*) – `value` attribute

Returns: an *ElementSelector*

aloe_webdriver.util.**string_literal**(*content*)
Choose a string literal that can wrap our string.

If your string contains a `'` the result will be wrapped in `"`. If your string contains a `"` the result will be wrapped in `'`.

Cannot currently handle strings which contain both `"` and `'`.

aloe_webdriver.util.**wait_for**(*func*)
A decorator to invoke a function, retrying on assertion errors for a specified time interval.

Adds a kwarg *timeout* to *func* which is a number of seconds to try for (default 15).

CHAPTER 6

Installing

```
pip install aloe_webdriver
```

CHAPTER 7

Getting Started

Create a file in your *steps/* directory, i.e. *steps/browser.py* and import the *Aloe-Webdriver* steps.

You are also responsible for building and maintaining the lifecycle of your `selenium.webdriver` referenced `world.browser`.

```
from contextlib import contextmanager

import aloe_webdriver
from aloe import around, world
from selenium import webdriver

@around.all
@contextmanager
def with_browser():
    world.browser = webdriver.Firefox()
    yield
    world.browser.quit()
    delattr(world, 'browser')
```

CHAPTER 8

History

Aloe-Webdriver originally started life as the library lettuce_webdriver. This is a fork of that library for Aloe.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index