
splinter Documentation

Release 0.18.0

andrews medina

Jun 10, 2022

| | | |
|----------|--|-----------|
| 1 | Sample code | 3 |
| 2 | Features | 5 |
| 2.1 | Why use Splinter? | 5 |
| 2.2 | Installation Guide | 6 |
| 2.3 | Splinter Tutorial | 6 |
| 2.4 | Browser | 8 |
| 2.5 | Finding elements | 10 |
| 2.6 | Mouse interactions | 11 |
| 2.7 | Interacting with elements in the page | 13 |
| 2.8 | Matchers | 14 |
| 2.9 | Cookies manipulation | 16 |
| 2.10 | Take screenshot | 16 |
| 2.11 | Take element screenshot | 17 |
| 2.12 | Take html snapshot | 17 |
| 2.13 | Executing javascript | 17 |
| 2.14 | Chrome WebDriver | 18 |
| 2.15 | Firefox WebDriver | 27 |
| 2.16 | Edge WebDriver | 36 |
| 2.17 | Remote WebDriver | 46 |
| 2.18 | zope.testbrowser | 47 |
| 2.19 | Django | 52 |
| 2.20 | Flask | 56 |
| 2.21 | Dealing with HTTP status code and exceptions | 61 |
| 2.22 | Using HTTP Proxies | 61 |
| 2.23 | Frames, alerts and prompts | 62 |
| 2.24 | API Documentation | 63 |
| 2.25 | Selenium Keys | 74 |
| 2.26 | Changelog | 74 |
| 2.27 | Community | 87 |
| 2.28 | Contribute | 88 |
| 2.29 | Writing new splinter drivers | 89 |
| 2.30 | Setting up your splinter development environment | 89 |
| | Python Module Index | 91 |
| | Index | 93 |

Splinter is an open source tool for testing web applications using Python. It lets you automate browser actions such as visiting URLs and interacting with their elements.

CHAPTER 1

Sample code

```
from splinter import Browser

with Browser() as browser:
    # Visit URL.
    browser.visit("http://www.google.com")

    # Find and fill out the search form.
    browser.find_by_name('q').fill('splinter - python acceptance testing for web_
→applications')

    # Find and click the 'search' button.
    button = browser.find_by_name('btnK').click()

    # Check for result on the page.
    if browser.is_text_present('splinter.readthedocs.io'):
        print("Yes, the official website was found!")
    else:
        print("No, it wasn't found... We need to improve our SEO techniques")
```

Note: if you don't provide any driver to the Browser function, firefox will be used.

Features

- Simple API
- Support for multiple drivers
- Support for iframes and alerts
- Can execute javascript
- Works with ajax and async javascript

2.1 Why use Splinter?

Splinter is an abstraction layer on top of existing browser automation tools such as [Selenium](#) and [zope.testbrowser](#). It has a *high-level API* that makes it easy to write automated tests of web applications.

For example, to fill out a form field with Splinter:

```
browser.fill('username', 'janedoe')
```

In Selenium, the equivalent code would be:

```
elem = browser.find_element.by_name('username')
elem.send_keys('janedoe')
```

Because Splinter is an abstraction layer, it supports multiple web automation backends. With Splinter, you can use the same test code to do browser-based testing with Selenium as the backend and “headless” testing (no GUI) with [zope.testbrowser](#) as the backend.

Splinter has drivers for browser-based testing on:

- *Chrome*
- *Firefox*
- *Browsers on remote machines*

For headless testing, Splinter has drivers for:

- *zope.testbrowser*
- *Django client*
- *Flask client*

2.2 Installation Guide

2.2.1 Basic Requirements

Splinter officially supports Python versions 3.6+ for Selenium 3, Django, Flask, and *zope.testbrowser*.

For Selenium 4, Python versions 3.7+ are supported.

2.2.2 Latest version via pip

To install the latest release, run the following command in the terminal:

```
python -m pip install splinter
```

Installing Driver Dependencies

pip can be given extra arguments to automatically install driver dependencies.

The *Drivers* section of the documentation has more information for each driver.

```
$ python -m pip install splinter[django]
```

2.2.3 Source Code

Splinter's source code is hosted on [GitHub](#). You can clone the repository using *git*:

```
$ git clone git://github.com/cobrateam/splinter.git
```

Once you have a copy of the source code, you can manually install the package:

```
$ cd splinter
$ python setup.py install
```

2.3 Splinter Tutorial

Before starting, make sure Splinter is *installed*

This tutorial provides a simple example, teaching step by step how to:

- search for 'splinter - python acceptance testing for web applications' in google.com, and
- find if splinter official website is listed among the search results

2.3.1 Create a Browser instance

First of all, import `Browser` class and instantiate it.

```
from splinter import Browser
browser = Browser()
```

Note: if you don't provide any driver argument to the `Browser` function, `firefox` will be used ([Browser function documentation](#)).

2.3.2 Visit Google website

Visit any website using the `browser.visit` method. Let's go to Google search page:

```
browser.visit('http://google.com')
```

2.3.3 Input search text

After a page is loaded, you can perform actions, such as clicking, filling text input, checking radio and checkbox. Let's fill Google's search field with `splinter - python acceptance testing for web applications`:

```
browser.fill('q', 'splinter - python acceptance testing for web applications')
```

2.3.4 Press the search button

Tell Splinter which button should be pressed. A button - or any other element - can be identified using its `css`, `xpath`, `id`, `tag` or `name`.

In order to find Google's search button, do:

```
button = browser.find_by_name('btnK')
```

Note that this `btnK` was found looking at Google's page source code.

With the button in hands, we can then press it:

```
button.click()
```

Note: Both steps presented above could be joined in a single line, such as:

```
browser.find_by_name('btnK').click()
```

2.3.5 Find out that Splinter official website is in the search results

After pressing the button, you can check if Splinter official website is among the search responses. This can be done like this:

```
if browser.is_text_present('splinter.readthedocs.io'):
    print("Yes, found it! :)")
else:
    print("No, didn't find it :(")
```

In this case, we are just printing something. You might use assertions, if you're writing tests.

2.3.6 Close the browser

When you've finished testing, close your browser using `browser.quit()`:

```
browser.quit()
```

2.3.7 All together

Finally, the source code will be:

```
from splinter import Browser

browser = Browser() # defaults to firefox
browser.visit('http://google.com')
browser.fill('q', 'splinter - python acceptance testing for web applications')
browser.find_by_name('btnK').click()

if browser.is_text_present('splinter.readthedocs.io'):
    print("Yes, the official website was found!")
else:
    print("No, it wasn't found... We need to improve our SEO techniques")

browser.quit()
```

2.4 Browser

To use splinter you need to create a `Browser` instance:

```
from splinter import Browser
browser = Browser()
```

Or, you can use it by a context manager, through the `with` statement:

```
from splinter import Browser
with Browser() as b:
    # stuff using the browser
```

This last example will create a new browser window and close it when the cursor reaches the code outside the `with` statement, automatically.

splinter supports the following drivers: * *Chrome* * *Firefox* * *Browsers on remote machines* * *zope.testbrowser* * *Django client* * *Flask client*

The following examples create new `Browser` instances for specific drivers:

```
browser = Browser('chrome')
browser = Browser('firefox')
browser = Browser('zope.testbrowser')
```

2.4.1 Navigating with `Browser.visit`

You can use the `visit` method to navigate to other pages:

```
browser.visit('http://cobrateam.info')
```

The `visit` method takes only a single parameter - the url to be visited.

You can visit a site protected with basic HTTP authentication by providing the username and password in the url.

```
browser.visit('http://username:password@cobrateam.info/protected')
```

2.4.2 Managing Windows

You can manage multiple windows (such as popups) through the `windows` object:

```
browser.windows          # all open windows
browser.windows[0]       # the first window
browser.windows[window_name] # the window_name window
browser.windows.current   # the current window
browser.windows.current = browser.windows[3] # set current window to window 3

window = browser.windows[0]
window.is_current         # boolean - whether window is current active window
window.is_current = True  # set this window to be current window
window.next               # the next window
window.prev               # the previous window
window.close()            # close this window
window.close_others()     # close all windows except this one
```

This window management interface is not compatible with the undocumented interface exposed in v0.6.0 and earlier.

2.4.3 Reload a page

You can reload a page using the `reload` method:

```
browser.reload()
```

2.4.4 Navigate through the history

You can move back and forward through your browsing history using the `back` and `forward` methods:

```
browser.visit('http://cobrateam.info')
browser.visit('https://splinter.readthedocs.io')
browser.back()
browser.forward()
```

2.4.5 Browser.title

You can get the title of the visited page using the `title` attribute:

```
browser.title
```

2.4.6 Verifying page content with Browser.html

You can use the `html` attribute to get the html content of the visited page:

```
browser.html
```

2.4.7 Verifying page url with Browser.url

The visited page's url can be accessed by the `url` attribute:

```
browser.url
```

2.4.8 Changing Browser User-Agent

You can pass a User-Agent header on Browser instantiation.

```
b = Browser(user_agent="Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en)")
```

2.5 Finding elements

Splinter provides 6 methods for finding elements in the page, one for each selector type: `css`, `xpath`, `tag`, `name`, `id`, `value`, `text`. Examples:

```
browser.find_by_css('h1')
browser.find_by_xpath('//h1')
browser.find_by_tag('h1')
browser.find_by_name('name')
browser.find_by_text('Hello World!')
browser.find_by_id('firstheader')
browser.find_by_value('query')
```

Each of these methods returns a list with the found elements. You can get the first found element with the `first` shortcut:

```
first_found = browser.find_by_name('name').first
```

There's also the `last` shortcut – obviously, it returns the last found element:

```
last_found = browser.find_by_name('name').last
```

2.5.1 Get element using index

You also can use an index to get the desired element in the list of found elements:

```
second_found = browser.find_by_name('name')[1]
```

2.5.2 All elements and `find_by_id`

A web page should have only one id, so the `find_by_id` method returns always a list with just one element.

2.5.3 Finding links

If you want to target only links on a page, you can use the methods provided in the links namespace. This is available at both the browser and element level.

Examples:

```
links_found = browser.links.find_by_text('Link for Example.com')
links_found = browser.links.find_by_partial_text('for Example')
links_found = browser.links.find_by_href('http://example.com')
links_found = browser.links.find_by_partial_href('example')

links_found = browser.find_by_css('.main').links.find_by_text('Link for Example.com')
links_found = browser.find_by_css('.main').links.find_by_partial_text('for Example.com')
links_found = browser.find_by_css('.main').links.find_by_href('http://example.com')
links_found = browser.find_by_css('.main').links.find_by_partial_href('example')
```

As the other `find_*` methods, these returns a list of all found elements.

2.5.4 Chaining find of elements

Finding methods are chainable, so you can find the descendants of a previously found element.

```
divs = browser.find_by_tag("div")
within_elements = divs.first.find_by_name("name")
```

2.5.5 `ElementDoesNotExist` exception

If an element is not found, the `find_*` methods return an empty list. But if you try to access an element in this list, the method will raise the `splinter.exceptions.ElementDoesNotExist` exception.

2.6 Mouse interactions

Note: Most mouse interaction currently works only on Chrome driver and Firefox 27.0.1.

Splinter provides some methods for mouse interactions with elements in the page. This feature is useful to test if an element appears on mouse over and disappears on mouse out (eg.: subitems of a menu).

It's also possible to send a click, double click or right click to the element.

Here is a simple example: imagine you have this `jQuery` event for mouse over and out:

```
$('.menu-links').mouseover(function() {
    $(this).find('.subitem').show();
});

$('.menu-links').mouseout(function() {
    $(this).find('.subitem').hide();
});
```

You can use Splinter to fire the event programmatically:

```
browser.find_by_css('.menu-links').mouse_over()  
# Code to check if the subitem is visible...  
browser.find_by_css('.menu-links').mouse_out()
```

The methods available for mouse interactions are:

2.6.1 mouse_over

Puts the mouse above the element. Example:

```
browser.find_by_tag('h1').mouse_over()
```

2.6.2 mouse_out

Puts the mouse out of the element. Example:

```
browser.find_by_tag('h1').mouse_out()
```

2.6.3 click

Clicks on the element. Example:

```
browser.find_by_tag('h1').click()
```

2.6.4 double_click

Double-clicks on the element. Example:

```
browser.find_by_tag('h1').double_click()
```

2.6.5 right_click

Right-clicks on the element. Example:

```
browser.find_by_tag('h1').right_click()
```

2.6.6 drag_and_drop

Yes, you can drag an element and drop it to another element! The example below drags the `<h1> . . . </h1>` element and drop it to a container element (identified by a CSS class).

```
draggable = browser.find_by_tag('h1')  
target = browser.find_by_css('.container')  
draggable.drag_and_drop(target)
```


2.7 Interacting with elements in the page

2.7.1 Get value of an element

In order to retrieve an element's value, use the `value` property:

```
browser.find_by_css('h1').first.value
```

or

```
element = browser.find_by_css('h1').first  
element.value
```

2.7.2 Clicking links

You can click in links. To click in links by href, partial href, text or partial text you can use this. **IMPORTANT:** These methods return the first element always.

```
browser.links.find_by_href('http://www.the_site.com/my_link').click()
```

or

```
browser.links.find_by_partial_href('my_link').click()
```

or

```
browser.links.find_by_text('my link').click()
```

or

```
browser.links.find_by_partial_text('part of link text').click()
```

2.7.3 Clicking buttons

You can click in buttons. Splinter follows any redirects, and submits forms associated with buttons.

```
browser.find_by_name('send').first.click()
```

2.7.4 Interacting with forms

```
browser.fill('query', 'my name')  
browser.attach_file('file', '/path/to/file/somefile.jpg')  
browser.choose('some-radio', 'radio-value')  
browser.check('some-check')  
browser.uncheck('some-check')  
browser.select('uf', 'rj')
```

To trigger JavaScript events, like `KeyDown` or `KeyUp`, you can use the `type` method.

```
browser.type('type', 'typing text')
```

If you pass the argument `slowly=True` to the `type` method you can interact with the page on every key pressed. Useful for testing field's autocompletion (the browser will wait until next iteration to type the subsequent key).

```
for key in browser.type('type', 'typing slowly', slowly=True):
    pass # make some assertion here with the key object :)
```

You can also use `type` and `fill` methods in an element:

```
browser.find_by_name('name').type('Steve Jobs', slowly=True)
browser.find_by_css('.city').fill('San Francisco')
```

2.7.5 Verifying if element is visible or invisible

To check if an element is visible or invisible, use the `visible` property. For instance:

```
browser.find_by_css('h1').first.visible
```

will be `True` if the element is visible, or `False` if it is invisible.

2.7.6 Verifying if element has a className

To check if an element has a `className`, use the `has_class` method. For instance:

```
browser.find_by_css('.content').first.has_class('content')
```

2.7.7 Interacting with elements through a `ElementList` object

Don't you like to always use `first` when selecting an element for clicking, for example:

```
browser.find_by_css('a.my-website').first.click()
```

You can invoke any `Element` method on `ElementList` and it will be proxied to the **first** element of the list. So the two lines below are equivalent:

```
assert browser.find_by_css('a.banner').first.visible
assert browser.find_by_css('a.banner').visible
```

2.7.8 Get the shadow root of an element

The `shadow_root` property provides access to the `ShadowRootElement` object. `ShadowRootElement` implements the `find_by_<x>` methods.

```
shadow_root = browser.find_by_css('a.my-website').first.shadow_root
elements = shadow_root.find_by_css('.my-elements')
```

2.8 Matchers

When working with AJAX and asynchronous JavaScript, it's common to have elements which are not present in the HTML code (they are created with JavaScript, dynamically). In this case you can use the methods

`is_element_present` and `is_text_present` to check the existence of an element or text – Splinter will load the HTML and JavaScript in the browser and the check will be performed *before* processing JavaScript.

There is also the optional argument `wait_time` (given in seconds) – it’s a timeout: if the verification method gets `True` it will return the result (even if the `wait_time` is not over), if it doesn’t get `True`, the method will wait until the `wait_time` is over (so it’ll return the result).

2.8.1 Checking the presence of text

The method `is_text_present` is responsible for checking if a text is present in the page content. It returns `True` or `False`.

```
browser = Browser()
browser.visit('https://splinter.readthedocs.io/')
browser.is_text_present('splinter') # True
browser.is_text_present('splinter', wait_time=10) # True, using wait_time
browser.is_text_present('text not present') # False
```

There’s also a method to check if the text *is not* present: `is_text_not_present`. It works the same way but returns `True` if the text is not present.

```
browser.is_text_not_present('text not present') # True
browser.is_text_not_present('text not present', wait_time=10) # True, using wait_time
browser.is_text_not_present('splinter') # False
```

2.8.2 Checking the presence of elements

Splinter provides 6 methods to check the presence of elements in the page, one for each selector type: `css`, `xpath`, `tag`, `name`, `id`, `value`, `text`. Examples:

```
browser.is_element_present_by_css('h1')
browser.is_element_present_by_xpath('//h1')
browser.is_element_present_by_tag('h1')
browser.is_element_present_by_name('name')
browser.is_element_present_by_text('Hello World!')
browser.is_element_present_by_id('firstheader')
browser.is_element_present_by_value('query')
browser.is_element_present_by_value('query', wait_time=10) # using wait_time
```

As expected, these methods returns `True` if the element is present and `False` if it is not present.

There’s also the negative forms of these methods, as in `is_text_present`:

```
browser.is_element_not_present_by_css('h6')
browser.is_element_not_present_by_xpath('//h6')
browser.is_element_not_present_by_tag('h6')
browser.is_element_not_present_by_name('unexisting-name')
browser.is_element_not_present_by_text('Not here :(')
browser.is_element_not_present_by_id('unexisting-header')
browser.is_element_not_present_by_id('unexisting-header', wait_time=10) # using wait_
→time
```

2.8.3 Checking the visibility of elements

There are two methods to check if the element is visible or hidden in the current page using either the selector type `css` or `xpath`. It returns `True` if the element is visible and `False` if the element is not visible.

```
browser.is_element_visible_by_css('h5')
browser.is_element_visible_by_css('h5', wait_time=10)
browser.is_element_visible_by_xpath('//h5')
```

2.9 Cookies manipulation

It is possible to manipulate cookies using the `cookies` attribute from a *Browser* instance. The `cookies` attribute is an instance of the *CookieManager* class that manipulates cookies (ie: adding and deleting).

2.9.1 Add a cookie

```
browser.cookies.add({'cookie_name': 'cookie_value'})
```

2.9.2 Retrieve all cookies

```
cookies = browser.cookies.all()
```

2.9.3 Delete a cookie

```
browser.cookies.delete('cookie_name') # delete the cookie 'cookie_name'
browser.cookies.delete('cookies_name_1', 'cookies_name_2') # delete two cookies
```

2.9.4 Delete all cookies

```
browser.cookies.delete_all()
```

For more details check the API reference of the *CookieManager* class.

Extra Arguments

Each driver accepts various parameters when creating cookies. These can be used with `browser.cookies.add` as extra arguments. For example, *WebDriver* can use `path`, `domain`, `secure`, and `expiry`:

```
:: browser.cookies.add({'cookie_name': 'cookie_value'}, path='/cookiePath')
```

2.10 Take screenshot

Splinter can take a screenshot of the current view:

```
browser = Browser()
screenshot_path = browser.screenshot('absolute_path/your_screenshot.png')
```

You should use the absolute path to save a screenshot. If you don't use an absolute path, the screenshot will be saved in a temporary file.

Take a full view screenshot:

```
browser = Browser()
screenshot_path = browser.screenshot('absolute_path/your_screenshot.png', full=True)
```

2.11 Take element screenshot

If the element is in the current view:

```
browser = Browser()
browser.visit('http://example.com')
element = browser.find_by_xpath('xpath_rule').first
screenshot_path = element.screenshot('absolute_path/your_screenshot.png')
```

If the element is not in the current view:

```
browser = Browser()
browser.visit('http://example.com')
element = browser.find_by_xpath('xpath_rule').first
screenshot_path = element.screenshot('absolute_path/your_screenshot.png', full=True)
```

2.12 Take html snapshot

Splinter can also take a snapshot of the current HTML:

```
browser = Browser()
screenshot_path = browser.html_snapshot('absolute_path/your_screenshot.html')
```

2.13 Executing javascript

You can easily execute JavaScript, in drivers which support it:

```
browser.execute_script("$('body').empty()")
```

You can return the result of the script:

```
browser.evaluate_script("4+4") == 8
```

2.13.1 Example: manipulating text fields with JavaScript

Some text input actions cannot be “typed” thru `browser.fill()`, like new lines and tab characters. Below is an example how to work around this using `browser.execute_script()`. This is also much faster than `browser.fill()` as there is no simulated key typing delay, making it suitable for longer texts.

```
def fast_fill_by_javascript(browser: DriverAPI, elem_id: str, text: str):
    """Fill text field with copy-paste, not by typing key by key.

    Otherwise you cannot type enter or tab.

    :param id: CSS id of the textarea element to fill
    """
    text = text.replace("\t", "\\t")
    text = text.replace("\n", "\\n")

    # Construct a JavaScript snippet that is executed on the browser side
    snippet = f"""document.querySelector("#{elem_id}").value = "{text}";"""
    browser.execute_script(snippet)
```

2.14 Chrome WebDriver

2.14.1 External Requirements

The following applications are required:

- [Google Chrome](#)
- [ChromeDriver](#)

Chromedriver must also be available on your operating system's *PATH* environment variable.

Dependencies

To use Chrome, the python bindings for Selenium 3 or Selenium 4 must be installed.

When splinter is installed via pip, the *selenium3* or *selenium4* extra argument can be provided. This will automatically install the latest version of Selenium 3 or Selenium 4, respectively.

```
python -m pip install splinter[selenium3]
```

2.14.2 Installing Chromedriver

Mac OS X

The recommended way is by using [Homebrew](#):

```
brew install chromedriver
```

Linux

Go to the [download page on the Chromium project](#) and choose the correct version for your Linux installation. Then extract the downloaded file in a directory in the PATH (e.g. `/usr/bin`). You can also extract it to any directory and add that directory to the PATH:

Linux 64bits

```
cd $HOME/Downloads
wget https://chromedriver.storage.googleapis.com/2.41/chromedriver_linux64.zip
unzip chromedriver_linux64.zip

mkdir -p $HOME/bin
mv chromedriver $HOME/bin
echo "export PATH=$PATH:$HOME/bin" >> $HOME/.bash_profile
```

Windows

Note: We don't provide official support for Windows, but you can try it by yourself.

All you need to do is go to [download page on Selenium project](#) and choose “ChromeDriver server for win”. Your browser will download a zip file, extract it and add the .exe file to your PATH.

If you don't know how to add an executable to the PATH on Windows, check these link out:

- [Environment variables](#)
- [How to manage environment variables in Windows XP](#)
- [How to manage environment variables in Windows 8 & 10](#)

Usage

To use the Chrome driver, pass the string `chrome` when you create the Browser instance:

```
from splinter import Browser
browser = Browser('chrome')
```

Note: if you don't provide any driver to the Browser function, `firefox` will be used.

Note: if you have trouble with `$HOME/.bash_profile`, you can try `$HOME/.bashrc`.

Headless mode

Starting with Chrome 59, Chrome can run in a headless mode. Further Information: [google developers updates](#)

To use headless mode, pass the `headless` argument when creating a new Browser instance.

```
from splinter import Browser
browser = Browser('chrome', headless=True)
```

Incognito mode

To use Chrome's incognito mode, pass the `incognito` argument when creating a Browser instance.

```
from splinter import Browser
browser = Browser('chrome', incognito=True)
```

Emulation mode

Chrome options can be passed to customize Chrome's behaviour; it is then possible to leverage the experimental emulation mode.

Further Information: [chrome driver documentation](#)

```
from selenium import webdriver
from splinter import Browser

mobile_emulation = {"deviceName": "Google Nexus 5"}
chrome_options = webdriver.ChromeOptions()
chrome_options.add_experimental_option(
    "mobileEmulation", mobile_emulation,
)
browser = Browser('chrome', options=chrome_options)
```

Custom executable path

Chrome can also be used from a custom path. Pass the executable path as a dictionary to the ***kwargs* argument. The dictionary should be set up with *executable_path* as the key and the value set to the path to the executable file.

```
from splinter import Browser
executable_path = {'executable_path': '</path/to/chrome>'}

browser = Browser('chrome', **executable_path)
```

API docs

```
class splinter.driver.webdriver.chrome.WebDriver (options=None, user_agent=None,
                                                  wait_time=2, fullscreen=False,
                                                  incognito=False, headless=False,
                                                  **kwargs)
```

attach_file (name, value)

Fill the field identified by name with the content specified by value.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (name)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` n times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name*, *value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values 'F' and 'M'.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

evaluate_script (*script*, **args*)

Similar to `execute_script` method.

Execute javascript in the browser and return the value of the expression.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> assert 4 == browser.evaluate_script('2 + 2')
```

execute_script (*script*, **args*)

Execute a piece of JavaScript in the browser.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> browser.execute_script('document.getElementById("body").innerHTML = "<p>
↪Hello world!</p>"')
```

fill (*name*, *value*)

Fill the field identified by `name` with the content specified by `value`.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id*=None, *name*=None, *ignore_missing*=False)

Fill the fields identified by *name* with the content specified by *value* in a dict.

Currently, `fill_form` supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of name.
- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by (*finder*, *finder_kwargs*=None, *original_find*: *str* = None, *original_query*: *str* = None, *wait_time*: *int* = None)

Wrapper for finding elements.

Must be attached to a class.

Returns `ElementList`

find_by_css (*css_selector*, *wait_time*=None)

Return an instance of `ElementList`, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id*, *wait_time*=None)

Find an element on the current page by its id.

Even when only one element is find, this method returns an instance of `ElementList`

Parameters **id** (*str*) – id to use in the search query.

find_by_name (*name*, *wait_time*=None)

Find elements on the current page by their name.

Return an instance of `ElementList`.

Parameters **name** (*str*) – name to use in the search query.

find_by_tag (*tag*, *wait_time*=None)

Find all elements of a given tag in current page.

Returns an instance of `ElementList`

Parameters **tag** (*str*) – tag to use in the search query.

find_by_text (*text*=None, *wait_time*=None)

Find elements on the current page by their text.

Returns an instance of `ElementList`

Parameters **text** (*str*) – text to use in the search query.

find_by_value (*value*, *wait_time*=None)

Find elements on the current page by their value.

Returns an instance of `ElementList`

Parameters **value** (*str*) – value to use in the search query.

find_by_xpath (*xpath*, *original_find*='xpath', *original_query*=None, *wait_time*=None)

Return an instance of `ElementList`, using a xpath selector to query the current page content.

Parameters `xpath` (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds `<option>` elements by their text.

Returns an instance of *ElementList*

Parameters `text` (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find `<option>` elements by their value.

Returns an instance of *ElementList*

Parameters `value` (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert (*wait_time=None*)

Change the context for working with alerts and prompts.

For more details, check the *docs about iframes, alerts and prompts*

get_iframe (*frame_reference*)

Change the context for working with iframes.

For more details, check the *docs about iframes, alerts and prompts*

html

Source of current page.

html_snapshot (*name=*”, *suffix=*’.html’, *encoding=*’utf-8’, *unique_file=True*)

Write the current html to a file.

Parameters

- **name** (*str*) – File name.
- **suffix** (*str*) – File extension.
- **encoding** (*str*) – File encoding.
- **unique_file** (*str*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type `str`

is_element_not_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type `bool`

is_element_not_present_by_id (*id*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_name (*name*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_text (*text*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_value (*value*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_id (*id*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_name (*name*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_text (*text*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_value (*value*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_text_present (*text*, *wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the `text` and False if not.

Return type bool

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters **url** (*str*) – URL path.

quit ()

Quit the browser, closing its windows (if it has one).

reload ()

Revisits the current URL.

screenshot (*name=""*, *suffix='.png'*, *full=False*, *unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type str

select (*name*, *value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name*, *value*, *slowly=False*)

Type a value into an element.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

uncheck (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.uncheck("send-me-emails")
```

If you call `browser.uncheck` n times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the `check` method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters **url** (*str*) – URL path.

2.15 Firefox WebDriver

2.15.1 External Requirements

The following applications are required:

- [Mozilla Firefox](#)
- [Geckodriver](#)

Geckodriver must also be available on your operating system's *PATH* environment variable.

Dependencies

To use Firefox, the python bindings for Selenium 3 or Selenium 4 must be installed.

When splinter is installed via pip, the *selenium3* or *selenium4* extra argument can be provided. This will automatically install the latest version of Selenium 3 or Selenium 4, respectively.

```
python -m pip install splinter[selenium3]
```

2.15.2 Installing Geckodriver

Mac OS X

The recommended way is by using [Homebrew](#):

```
brew install geckodriver
```

Usage

To use the Firefox driver, pass the string `firefox` when you create the `Browser` instance:

```
from splinter import Browser
browser = Browser('firefox')
```

Note: if you don't provide any driver to `Browser` function, `firefox` will be used.

Headless mode

Starting with Firefox 55, Firefox can run in a headless mode.

To use headless mode, pass the *headless* argument when creating a new `Browser` instance.

```
from splinter import Browser
browser = Browser('firefox', headless=True)
```

Incognito mode

To use Firefox's incognito mode, pass the *incognito* argument when creating a `Browser` instance.

```
from splinter import Browser
browser = Browser('firefox', incognito=True)
```


Specify Profile

You can specify a [Firefox profile](#) for using on Browser function using the `profile` keyword (passing the name of the profile as a `str` instance):

```
from splinter import Browser
browser = Browser('firefox', profile='my_profile')
```

If you don't specify a profile, a new temporary profile will be created (and deleted when you close the browser).

Firefox Extensions

An extension for firefox is a `.xpi` archive. To use an extension in Firefox webdriver profile you need to give the path of the extension, using the `extensions` keyword (passing the extensions as a `list` instance):

```
from splinter import Browser
browser = Browser('firefox', extensions=['extension1.xpi', 'extension2.xpi'])
```

After the browser is closed, extensions will be deleted from the profile, even if the profile is not a temporary one.

Selenium Capabilities

```
from splinter import Browser
browser = Browser('firefox', capabilities={'acceptSslCerts': True})
```

You can pass any selenium [read-write DesiredCapabilities](#) parameters for Firefox.

API docs

class `splinter.driver.webdriver.firefox.WebDriver` (*options=None, profile=None, extensions=None, user_agent=None, profile_preferences=None, fullscreen=False, wait_time=2, capabilities=None, headless=False, incognito=False, **kwargs*)

attach_file (*name, value*)

Fill the field identified by *name* with the content specified by *value*.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (*name*)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` `n` times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name*, *value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values 'F' and 'M'.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

evaluate_script (*script*, **args*)

Similar to `execute_script` method.

Execute javascript in the browser and return the value of the expression.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> assert 4 == browser.evaluate_script('2 + 2')
```

execute_script (*script*, **args*)

Execute a piece of JavaScript in the browser.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> browser.execute_script('document.getElementById("body").innerHTML = "<p>↵Hello world!</p>"')
```

fill (*name*, *value*)

Fill the field identified by `name` with the content specified by `value`.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id*=None, *name*=None, *ignore_missing*=False)

Fill the fields identified by *name* with the content specified by *value* in a dict.

Currently, fill_form supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of name.
- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by (*finder*, *finder_kwargs*=None, *original_find*: *str* = None, *original_query*: *str* = None, *wait_time*: *int* = None)

Wrapper for finding elements.

Must be attached to a class.

Returns *ElementList*

find_by_css (*css_selector*, *wait_time*=None)

Return an instance of *ElementList*, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id*, *wait_time*=None)

Find an element on the current page by its id.

Even when only one element is find, this method returns an instance of *ElementList*

Parameters **id** (*str*) – id to use in the search query.

find_by_name (*name*, *wait_time*=None)

Find elements on the current page by their name.

Return an instance of *ElementList*.

Parameters **name** (*str*) – name to use in the search query.

find_by_tag (*tag*, *wait_time*=None)

Find all elements of a given tag in current page.

Returns an instance of *ElementList*

Parameters **tag** (*str*) – tag to use in the search query.

find_by_text (*text*=None, *wait_time*=None)

Find elements on the current page by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_by_value (*value*, *wait_time*=None)

Find elements on the current page by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

find_by_xpath (*xpath*, *original_find*='xpath', *original_query*=None, *wait_time*=None)

Return an instance of *ElementList*, using a xpath selector to query the current page content.

Parameters `xpath` (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds `<option>` elements by their text.

Returns an instance of *ElementList*

Parameters `text` (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find `<option>` elements by their value.

Returns an instance of *ElementList*

Parameters `value` (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert (*wait_time=None*)

Change the context for working with alerts and prompts.

For more details, check the *docs about iframes, alerts and prompts*

get_iframe (*frame_reference*)

Change the context for working with iframes.

For more details, check the *docs about iframes, alerts and prompts*

html

Source of current page.

html_snapshot (*name=""*, *suffix='.html'*, *encoding='utf-8'*, *unique_file=True*)

Write the current html to a file.

Parameters

- **name** (*str*) – File name.
- **suffix** (*str*) – File extension.
- **encoding** (*str*) – File encoding.
- **unique_file** (*str*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type `str`

is_element_not_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type `bool`

is_element_not_present_by_id (*id*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_name (*name*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_text (*text*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_value (*value*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_id (*id*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_name (*name*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_text (*text*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_value (*value*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_text_present (*text*, *wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the `text` and False if not.

Return type bool

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters **url** (*str*) – URL path.

quit ()

Quit the browser, closing its windows (if it has one).

reload ()

Revisits the current URL.

screenshot (*name=""*, *suffix='.png'*, *full=False*, *unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type str

select (*name*, *value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name*, *value*, *slowly=False*)

Type a value into an element.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

unchecked (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.unchecked("send-me-emails")
```

If you call `browser.unchecked` n times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the `check` method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters **url** (*str*) – URL path.

2.16 Edge WebDriver

2.16.1 External Requirements

The following applications are required:

- [Microsoft Edge](#)
- [Microsoft Edge Driver](#)

Microsoft Edge Driver must also be available on your operating system's *PATH* environment variable.

Dependencies

To use Edge, the python bindings for Selenium 3 or Selenium 4 must be installed.

When splinter is installed via pip, the *selenium3* or *selenium4* extra argument can be provided. This will automatically install the latest version of Selenium 3 or Selenium 4, respectively.

```
python -m pip install splinter[selenium3]
```

When using Selenium 3, Edge also has the following dependency:

- [msedge-selenium-tools](#)

Using pip, it can be installed automatically as well:

```
python -m pip install splinter[selenium3, edge]
```

2.16.2 Installing Edgedriver

Mac OS X

Modern versions of Edge (79+) are available for Mac OS X. However, no versions of Edge Legacy are available.

Linux

Neither version of Edge is available for Linux, and thus Edge WebDriver cannot be used on Linux systems.

Usage

To use the Edge driver, pass the string *edge* when you create the *Browser* instance:

```
from splinter import Browser
browser = Browser('edge')
```

Edge Options

Selenium Options can be passed to customize Edge's behaviour through the *EdgeOptions* object

It must be imported from the *msedge-selenium-tools* package.

```
from msedge.selenium_tools import EdgeOptions
from splinter import Browser

mobile_emulation = {"deviceName": "Google Nexus 5"}
edge_options = EdgeOptions()
browser = Browser('edge', options=edge_options)
```

Headless mode

To use headless mode, pass the *headless* argument when creating a new Browser instance.

```
from splinter import Browser
browser = Browser('edge', headless=True)
```

Incognito mode

To use Edge's incognito mode, pass the *incognito* argument when creating a Browser instance.

```
from splinter import Browser
browser = Browser('edge', incognito=True)
```

Emulation mode

Since Selenium options can be passed to customize Edge's behaviour; it is then possible to leverage the experimental emulation mode.

```
from msedge.selenium_tools import EdgeOptions
from splinter import Browser

mobile_emulation = {"deviceName": "Google Nexus 5"}
edge_options = EdgeOptions()
edge_options.add_experimental_option(
    "mobileEmulation", mobile_emulation,
)
browser = Browser('edge', options=edge_options)
```

Custom executable path

Edge can also be used from a custom path. Pass the executable path as a dictionary to the ***kwargs* argument. The dictionary should be set up with *executable_path* as the key and the value set to the path to the executable file.

```
from splinter import Browser
executable_path = {'executable_path': '</path/to/edge>'}

browser = Browser('edge', **executable_path)
```

Edge Legacy

By default, Edge WebDriver is configured to use versions of Edge built with Chromium (Version 79 and up).

To use Edge Legacy, pass the *chromium* argument when creating a new Browser instance.

This requires the correct version of Edge and Edge Driver to be installed.

```
from splinter import Browser
browser = Browser('edge', chromium=False)
```

API docs

```
class splinter.driver.webdriver.edge.WebDriver (options=None, user_agent=None,  
                                              wait_time=2, fullscreen=False,  
                                              incognito=False, headless=False,  
                                              chromium=True, **kwargs)
```

attach_file (*name, value*)

Fill the field identified by *name* with the content specified by *value*.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (*name*)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` n times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name, value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values 'F' and 'M'.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

evaluate_script (*script, *args*)

Similar to `execute_script` method.

Execute javascript in the browser and return the value of the expression.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> assert 4 == browser.evaluate_script('2 + 2')
```

execute_script (*script*, **args*)

Execute a piece of JavaScript in the browser.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> browser.execute_script('document.getElementById("body").innerHTML = "<p>↵Hello world!</p>"')
```

fill (*name*, *value*)

Fill the field identified by name with the content specified by value.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id=None*, *name=None*, *ignore_missing=False*)

Fill the fields identified by name with the content specified by value in a dict.

Currently, fill_form supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of name.
- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by (*finder*, *finder_kwargs=None*, *original_find: str = None*, *original_query: str = None*, *wait_time: int = None*)

Wrapper for finding elements.

Must be attached to a class.

Returns `ElementList`

find_by_css (*css_selector*, *wait_time=None*)

Return an instance of `ElementList`, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id*, *wait_time=None*)

Find an element on the current page by its id.

Even when only one element is find, this method returns an instance of `ElementList`

Parameters **id** (*str*) – id to use in the search query.

find_by_name (*name*, *wait_time=None*)

Find elements on the current page by their name.

Return an instance of *ElementList*.

Parameters **name** (*str*) – name to use in the search query.

find_by_tag (*tag*, *wait_time=None*)

Find all elements of a given tag in current page.

Returns an instance of *ElementList*

Parameters **tag** (*str*) – tag to use in the search query.

find_by_text (*text=None*, *wait_time=None*)

Find elements on the current page by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_by_value (*value*, *wait_time=None*)

Find elements on the current page by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

find_by_xpath (*xpath*, *original_find='xpath'*, *original_query=None*, *wait_time=None*)

Return an instance of *ElementList*, using a xpath selector to query the current page content.

Parameters **xpath** (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds <option> elements by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find <option> elements by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert (*wait_time=None*)

Change the context for working with alerts and prompts.

For more details, check the *docs about iframes, alerts and prompts*

get_iframe (*frame_reference*)

Change the context for working with iframes.

For more details, check the *docs about iframes, alerts and prompts*

html

Source of current page.

html_snapshot (*name=""*, *suffix='.html'*, *encoding='utf-8'*, *unique_file=True*)

Write the current html to a file.

Parameters

- **name** (*str*) – File name.
- **suffix** (*str*) – File extension.
- **encoding** (*str*) – File encoding.
- **unique_file** (*str*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type str

is_element_not_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_id (*id*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_name (*name*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_text (*text*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_value (*value*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_id (*id*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_name (*name*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_text (*text*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_value (*value*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_text_present (*text*, *wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the `text` and False if not.

Return type bool

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters **url** (*str*) – URL path.

quit ()

Quit the browser, closing its windows (if it has one).

reload ()

Revisits the current URL.

screenshot (*name=""*, *suffix='.png'*, *full=False*, *unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type *str*

select (*name*, *value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name*, *value*, *slowly=False*)

Type a value into an element.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

uncheck (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.uncheck("send-me-emails")
```

If you call `browser.uncheck` *n* times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the `check` method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters `url` (*str*) – URL path.

2.17 Remote WebDriver

2.17.1 Dependencies

To use Remote WebDriver, the python bindings for Selenium 3 or Selenium 4 must be installed.

When splinter is installed via pip, the *selenium3* or *selenium4* extra argument can be provided. This will automatically install the latest version of Selenium 3 or Selenium 4, respectively.

```
python -m pip install splinter[selenium3]
```

Setting up the Remote WebDriver

To use Remote WebDriver, you need to have access to a Selenium remote WebDriver server. Setting up one of these servers is beyond the scope of this document. However, some companies provide access to a [Selenium Grid](#) as a service.

Usage

To use the Remote WebDriver, use `driver_name="remote"` when you create the `Browser` instance.

The `browser_name` argument should then be used to specify the web browser. The other arguments match Selenium's [Remote WebDriver](#) arguments.

[Desired Capabilities](#) will be set automatically based on Selenium's defaults. These can be expanded and/or replaced by providing your own.

The following example uses [LambdaTest](#) (a company that provides Selenium Remote WebDriver servers as a service) to request an Chrome version 99 browser instance running on Windows 11.

```
# Specify the server URL
remote_server_url = 'http://YOUR_LT_USERNAME:YOUR_LT_ACCESS_KEY@hub.lambdatest.com/
↳wd/hub'

with Browser(
    driver_name="remote",
    browser='Chrome',
    command_executor=remote_server_url,
```

(continues on next page)

(continued from previous page)

```

desired_capabilities = {
    'platform': 'Windows 11',
    'version': '99.0',
    'name': 'Test of Chrome 99 on WINDOWS',
},
keep_alive=True,
) as browser:

    browser.visit("https://www.lambdatest.com/selenium-playground/")
    browser.find_by_text('Simple Form Demo').first.click()

```

The following example uses [Sauce Labs](#) (a company that provides Selenium Remote WebDriver servers as a service) to request an Internet Explorer 9 browser instance running on Windows 7.

```

# Specify the server URL
remote_server_url = 'http://YOUR_SAUCE_USERNAME:YOUR_SAUCE_ACCESS_KEY@ondemand.
↳saucelabs.com:80/wd/hub'

with Browser(
    driver_name="remote",
    browser='internetexplorer',
    command_executor=remote_server_url,
    desired_capabilities = {
        'platform': 'Windows 7',
        'version': '9',
        'name': 'Test of IE 9 on WINDOWS',
    },
    keep_alive=True,
) as browser:
    print("Link to job: https://saucelabs.com/jobs/{}".format(
        browser.driver.session_id))
    browser.visit("https://splinter.readthedocs.io")
    browser.find_by_text('documentation').first.click()

```

2.18 zope.testbrowser

2.18.1 Dependencies

To use the `zope.testbrowser`, the following must be installed:

`zope.testbrowser`, `lxml`, `cssselect`.

When `splinter` is installed via `pip`, the `zope.testbrowser` extra argument can be provided. This will automatically install `Flask`.

```
python -m pip install splinter[zope.testbrowser]
```

2.18.2 Usage

To use the `zope.testbrowser` driver, all you need to do is pass the string `zope.testbrowser` when you create the `Browser` instance:

```
from splinter import Browser
browser = Browser('zope.testbrowser')
```

By default `zope.testbrowser` respects any `robots.txt` preventing access to a lot of sites. If you want to circumvent this you can call

```
browser = Browser('zope.testbrowser', ignore_robots=True)
```

Note: if you don't provide any driver to `Browser` function, `firefox` will be used.

2.18.3 API docs

class `splinter.driver.zopetestbrowser.ZopeTestBrowser` (*wait_time=2*)

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (*name*)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` *n* times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name*, *value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values 'F' and 'M'.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

fill (*name*, *value*)

Fill the field identified by `name` with the content specified by `value`.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id=None*, *name=None*, *ignore_missing=False*)

Fill the fields identified by *name* with the content specified by *value* in a dict.

Currently, *fill_form* supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of *name*.
- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by_css (*selector*)

Return an instance of *ElementList*, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id_value*)

Find an element on the current page by its id.

Even when only one element is find, this method returns an instance of *ElementList*

Parameters **id** (*str*) – id to use in the search query.

find_by_name (*name*)

Find elements on the current page by their name.

Return an instance of *ElementList*.

Parameters **name** (*str*) – name to use in the search query.

find_by_tag (*tag*)

Find all elements of a given tag in current page.

Returns an instance of *ElementList*

Parameters **tag** (*str*) – tag to use in the search query.

find_by_text (*text*)

Find elements on the current page by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_by_value (*value*)

Find elements on the current page by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

find_by_xpath (*xpath*, *original_find=None*, *original_query=None*)

Return an instance of *ElementList*, using a xpath selector to query the current page content.

Parameters **xpath** (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds `<option>` elements by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find `<option>` elements by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert ()

Change the context for working with alerts and prompts.

For more details, check the *docs about iframes, alerts and prompts*

get_iframe (*name*)

Change the context for working with iframes.

For more details, check the *docs about iframes, alerts and prompts*

html

Source of current page.

html_snapshot (*name: str = "", suffix: str = '.html', encoding: str = 'utf-8', unique_file: bool = True*)
→ *str*

Write the current html to a file.

Parameters

- **name** (*str*) – File name.
- **suffix** (*str*) – File extension.
- **encoding** (*str*) – File encoding.
- **unique_file** (*str*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type *str*

is_text_present (*text, wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the `text` and False if not.

Return type *bool*

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters **url** (*str*) – URL path.

quit()

Quit the browser, closing its windows (if it has one).

reload()

Revisits the current URL.

screenshot (*name=None, suffix=None, full=False, unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type *str*

select (*name, value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name, value, slowly=False*)

Type a value into an element.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

uncheck (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.uncheck("send-me-emails")
```

If you call `browser.uncheck` n times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the `check` method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters **url** (*str*) – URL path.

2.19 Django

2.19.1 Dependencies

To use the `django` driver, the following must be installed:

`django`, `lxml`, `cssselect`.

When splinter is installed via `pip`, the *django* extra argument can be provided. This will automatically install Django.

```
python -m pip install splinter[django]
```

2.19.2 Usage

To use the `django` driver, all you need to do is pass the string `django` when you create the `Browser` instance:

```
from splinter import Browser
browser = Browser('django')
```

Note: if you don't provide any driver to `Browser` function, `firefox` will be used.

2.19.3 API docs

```
class splinter.driver.djangoclient.DjangoClient (user_agent=None, wait_time=2, **kwargs)
```

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (*name*)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` `n` times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name*, *value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values 'F' and 'M'.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

fill (*name*, *value*)

Fill the field identified by *name* with the content specified by *value*.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id*=None, *name*=None, *ignore_missing*=False)

Fill the fields identified by *name* with the content specified by *value* in a dict.

Currently, `fill_form` supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of name.
- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by_css (*selector*)

Return an instance of *ElementList*, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id_value*)

Find an element on the current page by its id.

Even when only one element is found, this method returns an instance of *ElementList*

Parameters *id* (*str*) – id to use in the search query.

find_by_name (*name*)

Find elements on the current page by their name.

Return an instance of *ElementList*.

Parameters *name* (*str*) – name to use in the search query.

find_by_tag (*tag*)

Find all elements of a given tag in current page.

Returns an instance of *ElementList*

Parameters *tag* (*str*) – tag to use in the search query.

find_by_text (*text*)

Find elements on the current page by their text.

Returns an instance of *ElementList*

Parameters *text* (*str*) – text to use in the search query.

find_by_value (*value*)

Find elements on the current page by their value.

Returns an instance of *ElementList*

Parameters *value* (*str*) – value to use in the search query.

find_by_xpath (*xpath*, *original_find=None*, *original_query=None*)

Return an instance of *ElementList*, using a xpath selector to query the current page content.

Parameters *xpath* (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds <option> elements by their text.

Returns an instance of *ElementList*

Parameters *text* (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find <option> elements by their value.

Returns an instance of *ElementList*

Parameters *value* (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert ()

Change the context for working with alerts and prompts.

For more details, check the *docs about iframes, alerts and prompts*

get_iframe (*name*)

Change the context for working with iframes.

For more details, check the *docs about iframes, alerts and prompts*

html

Source of current page.

html_snapshot (*name: str = "", suffix: str = '.html', encoding: str = 'utf-8', unique_file: bool = True*)

→ *str*
Write the current html to a file.

Parameters

- **name** (*str*) – File name.
- **suffix** (*str*) – File extension.
- **encoding** (*str*) – File encoding.
- **unique_file** (*str*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type *str*

is_text_present (*text, wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the *text* and False if not.

Return type *bool*

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters *url* (*str*) – URL path.

quit ()

Quit the browser, closing its windows (if it has one).

reload ()

Revisits the current URL.

screenshot (*name=None, suffix=None, full=False, unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type *str*

select (*name, value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name*, *value*, *slowly=False*)

Type a value into an element.

It's useful to test javascript events like keyPress, keyUp, keyDown, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

unchecked (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.unchecked("send-me-emails")
```

If you call `browser.unchecked` *n* times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the `check` method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters **url** (*str*) – URL path.

2.20 Flask

2.20.1 Dependencies

To use the `flask` driver, the following must be installed:

Flask, lxml, cssselect.

When splinter is installed via pip, the `flask` extra argument can be provided. This will automatically install Flask.

```
python -m pip install splinter[flask]
```

2.20.2 Usage

To use the flask driver, you'll need to pass the string `flask` and an app instances via the `app` keyword argument when you create the `Browser` instance:

```
from splinter import Browser
browser = Browser('flask', app=app)
```

Note: if you don't provide any driver to `Browser` function, `firefox` will be used.

When visiting pages with the Flask client, you only need to provide a path rather than a full URL. For example:

```
browser.visit('/my-path')
```

2.20.3 API docs

class `splinter.driver.flaskclient.FlaskClient` (*app*, *user_agent*=None, *wait_time*=2, *custom_headers*=None)

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (*name*)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` n times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name*, *value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values `'F'` and `'M'`.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

fill (*name*, *value*)

Fill the field identified by *name* with the content specified by *value*.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id*=None, *name*=None, *ignore_missing*=False)

Fill the fields identified by *name* with the content specified by *value* in a dict.

Currently, *fill_form* supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of name.
- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by_css (*selector*)

Return an instance of *ElementList*, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id_value*)

Find an element on the current page by its id.

Even when only one element is find, this method returns an instance of *ElementList*

Parameters **id** (*str*) – id to use in the search query.

find_by_name (*name*)

Find elements on the current page by their name.

Return an instance of *ElementList*.

Parameters **name** (*str*) – name to use in the search query.

find_by_tag (*tag*)

Find all elements of a given tag in current page.

Returns an instance of *ElementList*

Parameters **tag** (*str*) – tag to use in the search query.

find_by_text (*text*)

Find elements on the current page by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_by_value (*value*)

Find elements on the current page by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

find_by_xpath (*xpath*, *original_find=None*, *original_query=None*)

Return an instance of *ElementList*, using a xpath selector to query the current page content.

Parameters **xpath** (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds <option> elements by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find <option> elements by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert ()

Change the context for working with alerts and prompts.

For more details, check the *docs about iframes, alerts and prompts*

get_iframe (*name*)

Change the context for working with iframes.

For more details, check the *docs about iframes, alerts and prompts*

html

Source of current page.

html_snapshot (*name: str = "", suffix: str = '.html', encoding: str = 'utf-8', unique_file: bool = True*)

→ *str*
Write the current html to a file.

Parameters

- **name** (*str*) – File name.
- **suffix** (*str*) – File extension.
- **encoding** (*str*) – File encoding.
- **unique_file** (*str*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type *str*

is_text_present (*text*, *wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the `text` and False if not.

Return type bool

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters **url** (*str*) – URL path.

quit ()

Quit the browser, closing its windows (if it has one).

reload ()

Revisits the current URL.

screenshot (*name=None, suffix=None, full=False, unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type str

select (*name, value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name, value, slowly=False*)

Type a value into an element.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

unchecked (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.unchecked("send-me-emails")
```

If you call `browser.unchecked` *n* times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the `checked` method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters **url** (*str*) – URL path.

2.21 Dealing with HTTP status code and exceptions

Note: After 0.8 version the *webdriver* (firefox, chrome) based drivers does not support http error handling.

2.21.1 Dealing with HTTP status code

It's also possible to check which HTTP status code a `browser.visit` gets. You can use `status_code.is_success` to do the work for you or you can compare the status code directly:

```
browser.visit('http://cobrateam.info')
browser.status_code.is_success() # True
# or
browser.status_code == 200 # True
# or
browser.status_code.code # 200
```

The difference between those methods is that if you get a redirect (or something that is not an HTTP error), `status_code.is_success` will consider your response as successfully. The numeric status code can be accessed via `status_code.code`.

2.22 Using HTTP Proxies

Unauthenticated proxies are simple, you need only configure the browser with the hostname and port.

Authenticated proxies are rather more complicated, (see [RFC2617](#))

2.22.1 Using an unauthenticated HTTP proxy with Firefox

```
profile = {
    'network.proxy.http': YOUR_PROXY_SERVER_HOST,
    'network.proxy.http_port': YOUR_PROXY_SERVER_PORT,
    'network.proxy.ssl': YOUR_PROXY_SERVER_HOST,
    'network.proxy.ssl_port': YOUR_PROXY_SERVER_PORT,
    'network.proxy.type': 1
}
self.browser = Browser(self.browser_type, profile_preferences=profile)
```

2.22.2 Authenticated HTTP proxy with Firefox

If you have access to the browser window, then the same technique will work for an authenticated proxy, but you will have to type the username and password in manually.

If this is not possible, for example on a remote CI server, then it is not currently clear how to do this. This document will be updated when more information is known. If you can help, please follow up on <https://github.com/cobrateam/splinter/issues/359>.

2.23 Frames, alerts and prompts

2.23.1 Using iframes

You can use the `get_iframe` method and the `with` statement to interact with iframes. You can pass the iframe's name, id, index, or web element to `get_iframe`.

```
with browser.get_iframe('iframemodal') as iframe:
    iframe.do_stuff()
```

2.23.2 Handling alerts and prompts

Chrome support for alerts and prompts is new in Splinter 0.4.

IMPORTANT: Only webdriver (Firefox and Chrome) has support for alerts and prompts.

You can interact with alerts and prompts using the `get_alert` method.

```
alert = browser.get_alert()
alert.text
alert.accept()
alert.dismiss()
```

In case of prompts, you can answer it using the `send_keys` method.

```
prompt = browser.get_alert()
prompt.text
prompt.send_keys('text')
prompt.accept()
prompt.dismiss()
```

You can also use the `with` statement to interact with both alerts and prompts.

```
with browser.get_alert() as alert:
    alert.do_stuff()
```

If there's no prompt or alert, `get_alert` will return `None`. Remember to always use at least one of the alert/prompt ending methods (accept and dismiss). Otherwise, your browser instance will be frozen until you accept or dismiss the alert/prompt correctly.

2.24 API Documentation

Welcome to the Splinter API documentation! Check what's inside:

2.24.1 Browser

`splinter.browser.Browser(driver_name='firefox', retry_count=3, *args, **kwargs)`

Get a new driver instance.

Extra arguments will be sent to the driver instance.

If there is no driver registered with the provided `driver_name`, this function will raise a `splinter.exceptions.DriverNotFoundError` exception.

Parameters

- **driver_name** (*str*) – Name of the driver to use.
- **retry_count** (*int*) – Number of times to try and instantiate the driver.

Returns Driver instance

2.24.2 DriverAPI

class `splinter.driver.DriverAPI`

Basic driver API class.

back ()

The browser will navigate to the previous URL in the history.

If there is no previous URL, this method does nothing.

check (*name*)

Check a checkbox by its name.

Parameters **name** (*str*) – name of the element to check.

Example

```
>>> browser.check("agree-with-terms")
```

If you call `browser.check` n times, the checkbox keeps checked, it never get unchecked.

To uncheck a checkbox, take a look in the `uncheck` method.

choose (*name, value*)

Choose a value in a radio buttons group.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to choose.

Example

You have two radio buttons in a page, with the name `gender` and values `'F'` and `'M'`.

```
>>> browser.choose('gender', 'F')
```

Then the female gender will be chosen.

cookies

A *CookieManager* instance.

For more details, check the *cookies manipulation section*.

evaluate_script (*script*, **args*)

Similar to *execute_script* method.

Execute javascript in the browser and return the value of the expression.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> assert 4 == browser.evaluate_script('2 + 2')
```

execute_script (*script*, **args*)

Execute a piece of JavaScript in the browser.

Parameters **script** (*str*) – The piece of JavaScript to execute.

Example

```
>>> browser.execute_script('document.getElementById("body").innerHTML = "<p>  
↪Hello world!</p>"')
```

fill (*name*, *value*)

Fill the field identified by *name* with the content specified by *value*.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.

fill_form (*field_values*, *form_id*=None, *name*=None)

Fill the fields identified by *name* with the content specified by *value* in a dict.

Currently, *fill_form* supports the following fields: text, password, textarea, checkbox, radio and select.

Checkboxes should be specified as a boolean in the dict.

Parameters

- **field_values** (*dict*) – Values for all the fields in the form, in the pattern of {field name: field value}
- **form_id** (*str*) – Id of the form to fill. Can be used instead of *name*.

- **name** (*str*) – Name of the form to fill.
- **ignore_missing** (*bool*) – Ignore missing keys in the dict.

find_by_css (*css_selector*)

Return an instance of *ElementList*, using a CSS selector to query the current page content.

Parameters **css_selector** (*str*) – CSS Selector to use in the search query.

find_by_id (*id*)

Find an element on the current page by its id.

Even when only one element is find, this method returns an instance of *ElementList*

Parameters **id** (*str*) – id to use in the search query.

find_by_name (*name*)

Find elements on the current page by their name.

Return an instance of *ElementList*.

Parameters **name** (*str*) – name to use in the search query.

find_by_tag (*tag*)

Find all elements of a given tag in current page.

Returns an instance of *ElementList*

Parameters **tag** (*str*) – tag to use in the search query.

find_by_text (*text*)

Find elements on the current page by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_by_value (*value*)

Find elements on the current page by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

find_by_xpath (*xpath*)

Return an instance of *ElementList*, using a xpath selector to query the current page content.

Parameters **xpath** (*str*) – Xpath to use in the search query.

find_option_by_text (*text*)

Finds <option> elements by their text.

Returns an instance of *ElementList*

Parameters **text** (*str*) – text to use in the search query.

find_option_by_value (*value*)

Find <option> elements by their value.

Returns an instance of *ElementList*

Parameters **value** (*str*) – value to use in the search query.

forward ()

The browser will navigate to the next URL in the history.

If there is no URL to forward, this method does nothing.

get_alert()

Change the context for working with alerts and prompts.

For more details, check the [docs about iframes, alerts and prompts](#)

get_iframe(name)

Change the context for working with iframes.

For more details, check the [docs about iframes, alerts and prompts](#)

html

Source of current page.

html_snapshot(name: str = "", suffix: str = '.html', encoding: str = 'utf-8', unique_file: bool = True)

→ str
Write the current html to a file.

Parameters

- **name** (str) – File name.
- **suffix** (str) – File extension.
- **encoding** (str) – File encoding.
- **unique_file** (str) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created html snapshot.

Return type str

is_element_not_present_by_css(css_selector, wait_time=None)

Verify if an element is not present in the current page.

Parameters

- **css** (str) – css selector for the element.
- **wait_time** (int) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_id(id, wait_time=None)

Verify if an element is not present in the current page.

Parameters

- **id** (str) – id for the element.
- **wait_time** (int) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_name(name, wait_time=None)

Verify if an element is not present in the current page.

Parameters

- **name** (str) – name of the element.
- **wait_time** (int) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_text (*text*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_value (*value*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_not_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is not present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is not present and False if is present.

Return type bool

is_element_present_by_css (*css_selector*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **css** (*str*) – css selector for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_id (*id*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **id** (*str*) – id for the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_name (*name*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **name** (*str*) – name of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_tag (*tag*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **tag** (*str*) – tag of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_text (*text*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **text** (*str*) – text in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_value (*value*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **value** (*str*) – value in the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_element_present_by_xpath (*xpath*, *wait_time=None*)

Verify if an element is present in the current page.

Parameters

- **xpath** (*str*) – xpath of the element.
- **wait_time** (*int*) – Number of seconds to search.

Returns True if the element is present and False if is not present.

Return type bool

is_text_present (*text*, *wait_time=None*)

Check if a piece of text is on the page.

Parameters

- **text** (*str*) – text to use in the search query.
- **wait_time** (*int*) – Number of seconds to search for the text.

Returns True if finds a match for the `text` and False if not.

Return type bool

new_tab (*url*)

The browser will navigate to the given URL in a new tab.

Parameters **url** (*str*) – URL path.

quit ()

Quit the browser, closing its windows (if it has one).

reload ()

Revisits the current URL.

screenshot (*name=None*, *suffix=None*, *full=False*, *unique_file=True*)

Take a screenshot of the current page and save it locally.

Parameters

- **name** (*str*) – File name for the screenshot.
- **suffix** (*str*) – File extension for the screenshot.
- **full** (*bool*) – If the screenshot should be full screen or not.
- **unique_file** (*bool*) – If true, the filename will include a path to the system temp directory and extra characters at the end to ensure the file is unique.

Returns Full file name of the created screenshot.

Return type str

select (*name*, *value*)

Select an `<option>` element in an `<select>` element using the name of the `<select>` and the value of the `<option>`.

Parameters

- **name** (*str*) – name of the option element.
- **value** (*str*) – Value to select.

Example

```
>>> browser.select("state", "NY")
```

title

Title of current page.

type (*name*, *value*, *slowly=False*)

Type a value into an element.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

Parameters

- **name** (*str*) – name of the element to enter text into.
- **value** (*str*) – Value to enter into the element.
- **slowly** (*bool*) – If True, this function returns an iterator which will type one character per iteration.

unchecked (*name*)

Uncheck a checkbox by its name.

Parameters **name** (*str*) – name of the element to uncheck.

Example

```
>>> browser.unchecked("send-me-emails")
```

If you call `browser.unchecked` n times, the checkbox keeps unchecked, it never get checked.

To check a checkbox, take a look in the [check](#) method.

url

URL of current page.

visit (*url*)

The browser will navigate to the given URL.

Parameters **url** (*str*) – URL path.

2.24.3 ElementAPI

class `splinter.driver.ElementAPI`

Basic element API class.

Any element in the page can be represented as an instance of `ElementAPI`.

Once you have an instance, you can easily access attributes like a dict:

```
>>> element = browser.find_by_id("link-logo").first
>>> assert element['href'] == 'https://splinter.readthedocs.io'
```

You can also interact with the instance using the methods and properties listed below.

check ()

Check the element, if it's "checkable" (e.g.: a checkbox).

If the element is already checked, this method does nothing. For unchecking elements, take a look in the [unchecked](#) method.

checked

Boolean property that says if the element is checked or not.

Example

```
>>> element.check()
>>> assert element.checked
>>> element.uncheck()
>>> assert not element.checked
```

clear()

Reset the field value.

click()

Click in the element.

fill(value)

Fill the field with the content specified by `value`.

has_class(class_name)

Indicates whether the element has the given class.

mouse_out()

Move the mouse away from the element.

mouse_over()

Put the mouse over the element.

screenshot(name=None, suffix=None, full=False, unique_file=True)

Take screenshot of the element.

select(value, slowly=False)

Select an `<option>` element in the element using the `value` of the `<option>`.

Example

```
>>> element.select("NY")
```

shadow_root

Get the shadow root of an element's shadow DOM.

text

String of all of the text within the element. HTML tags are stripped.

type(value, slowly=False)

Type the `value` in the field.

It's useful to test javascript events like `keyPress`, `keyUp`, `keyDown`, etc.

If `slowly` is `True`, this function returns an iterator which will type one character per iteration.

Also supports selenium.

Example

```
>>> from selenium.webdriver.common.keys import Keys
>>> ElementAPI.type(Keys.RETURN)
```

uncheck()

Uncheck the element, if it's "checkable" (e.g.: a checkbox).

If the element is already unchecked, this method does nothing. For checking elements, take a look in the [check](#) method.

value

Value of the element, usually a form element

visible

Boolean property that says if the element is visible or hidden in the current page.

2.24.4 CookieManager

class `splinter.cookie_manager.CookieManagerAPI` (*driver*)

An API that specifies how a splinter driver deals with cookies.

You can add cookies using the [add](#) method, and remove one or all cookies using the [delete](#) method.

A CookieManager acts like a dict, so you can access the value of a cookie through the `[]` operator, passing the cookie identifier:

```
>>> cookie_manager.add({'name': 'Tony'})
>>> assert cookie_manager['name'] == 'Tony'
```

add (*cookie*, ***kwargs*)

Add a cookie.

Extra arguments will be used to build the cookie.

Parameters **cookie** (*dict*) – Each key is an identifier for the cookie value.

Examples

```
>>> cookie_manager.add({'name': 'Tony'})
>>> browser.cookies.add({'cookie_name': 'cookie_value'}, path='/cookiePath')
```

all (*verbose=False*)

Get all of the cookies.

Note: If you're using any webdriver and want more info about the cookie, set the *verbose* parameter to *True* (in other drivers, it won't make any difference). In this case, this method will return a list of dicts, each with one cookie's info.

Examples

```
>>> cookie_manager.add({'name': 'Tony'})
>>> cookie_manager.all()
[{'name': 'Tony'}]
```

Returns All the available cookies.

delete (**cookies*)

Delete one or more cookies.

You can pass all the cookies identifier that you want to delete.

If identifiers are provided, all cookies are deleted.

Parameters **cookies** (*list*) – Identifiers for each cookie to delete.

Examples

```
>>> cookie_manager.delete() # deletes all cookies
>>> cookie_manager.delete(
    'name', 'birthday', 'favorite_color') # deletes these three cookies
>>> cookie_manager.delete('name') # deletes one cookie
```

delete_all()
Delete all cookies.

2.24.5 ElementList

class `splinter.element_list.ElementList` (*list*, *driver=None*, *find_by=None*, *query=None*)
Bases: `object`

Collection of elements.

Each member of the collection is by default an instance of `ElementAPI`.

Beyond the traditional list methods, `ElementList` provides some other methods, listed below.

There is a peculiar behavior on `ElementList`: you never get an `IndexError`. Instead, you get an `ElementDoesNotExist` exception when trying to access a non-existent item:

```
>>> element_list = ElementList([])
>>> element_list[0] # raises ElementDoesNotExist
```

first
An alias to the first element of the list.

Example

```
>>> assert element_list[0] == element_list.first
```

is_empty() → `bool`
Check if the `ElementList` is empty.

Returns True if the list is empty, else False

Return type `bool`

last
An alias to the last element of the list.

Example

```
>>> assert element_list[-1] == element_list.last
```

2.24.6 Request handling

class `splinter.request_handler.status_code.StatusCode` (*status_code*, *reason*)

code = `None`

200)

Type Code of the response (example

is_success()

Returns True if the response was succeed, otherwise, returns False.

reason = None

Success)

Type A message for the response (example

2.24.7 Exceptions

class splinter.exceptions.DriverNotFoundError

Exception raised when a driver is not found.

Example

```
>>> from splinter import Browser
>>> b = Browser('unknown driver') # raises DriverNotFoundError
```

class splinter.exceptions.ElementDoesNotExist

Exception raised when an element is not found in the page.

The exception is raised only when someone tries to access the element, not when the driver is finding it.

Example

```
>>> elements = browser.find_by_id('unknown-id') # returns an empty list
>>> elements[0] # raises ElementDoesNotExist
```

2.25 Selenium Keys

With Splinter's `type()` method, you can use Selenium's Keys implementation.

```
from selenium.webdriver.common.keys import Keys
from splinter import Browser

browser = Browser()
browser.type(Keys.RETURN)
```

The full list of all supported keys can be found at the official Selenium documentation: [selenium.webdriver.common.keys](#)

2.26 Changelog

2.26.1 What's New in Splinter 0.0.1?

Features

- support to firefox selenium 2 driver
- support to zope test browser
- navigating with Browser.visit
- get the title of the visited page
- get the html content of the visited page
- visited page's url can be accessed by the url attribute
- finding first element by tag, xpath, css selector, name and id
- find first link by xpath or text
- interacting with forms: text input, file, radio and check button
- verifying if element is visible or invisible
- executing and evaluating javascript
- debug with save and open page

2.26.2 What's New in Splinter 0.0.2?

Features

- fill instead of fill_in to fill inputs
- support to google chrome selenium 2 driver
- form interactions now support select
- issue #11: improve find's methods to return all/first/last elements

now finder methods (find_by_name, find_by_css_selector, find_by_tag, find_by_id, find_by_xpath) returns a ElementList object that contains a list of all found elements:

```
browser.find_by_name('name')
```

.first - to find first element

```
browser.find_by_name('name').first
```

.last - to find last element

```
browser.find_by_name('name').last
```

And additionally, using index

```
browser.find_by_name('name')[1]
```

An id should be unique in a web page, so find_by_id() method always returns a list with a single element.

Backward incompatible changes

- issue #24 remove `save_and_open_page` method from splinter api. This feature is out of splinter's scope, hence should be implemented as an external package.
- now finder methods (`find_by_name`, `find_by_css_selector`, `find_by_tag`, `find_by_id`, `find_by_xpath`) returns a list with elements, to get the first element founded use *first* attribute

```
browser.find_by_name('name').first
```

2.26.3 What's New in Splinter 0.0.3?

Features

- now splinter use selenium 2.0b3 for firefox and chrome driver
- `zope.testbrowser.browser` dependency is not required
- new method for reload a page
- `find_by_css_selector` is now deprecated, use `find_by_css` instead
- deprecated methods now throw “DeprecationWarning”
- methods for verify if element or text is present
- `find_by` methods wait for element
- added support for iframes and alerts
- added more specific exception messages for not found elements

Backward incompatible changes

- you should update your selenium to 2.0b3 version

2.26.4 What's New in Splinter 0.1?

Features

- capability to handle HTTP errors (using an exception) in Selenium drivers (Firefox and Chrome)
- capability to work with HTTP status code in Selenium drivers (Firefox and Chrome)
- browsing history (`back` and `forward` methods in `Browser` class)
- improvements in documentation

Bugfixes

- fixed Chrome driver instability
- fixed `Browser.choose` behaviour
- fixed `WebDriver` silencing routine

Backward incompatible changes

- you should update your selenium to 2.0rc2 version

2.26.5 What's New in Splinter 0.1.1?

- compatibility with Firefox 5

2.26.6 What's New in Splinter 0.10.0?

- Scroll to elements before to execute action chains ()
- Using *options* instead *firefox_options* to avoid warnings (<https://github.com/cobrateam/splinter/pull/634>)
- Add support for **args* parameter in *execute_script* (<https://github.com/cobrateam/splinter/issues/436>)
- Implement *__ne__* in *StatusCode* (<https://github.com/cobrateam/splinter/issues/460>)
- Using the new syntax *switch_to_alert* instead *switch_to.alert* to avoid webdriver warnings.
- *CookieManager*. *__eq__* returns a bool value (<https://github.com/cobrateam/splinter/issues/308><Paste>)
- Fix *find_by_text* to be used inside a chain (<https://github.com/cobrateam/splinter/issues/6281>)
- Add support for selenium 3.141.0

2.26.7 What's New in Splinter 0.11.0?

- *Browser.get_alert()* returns *Alert* instead of a wrapper object
- Add *browser.html_snapshot* method
- Allow *browser.get_iframe()* to accept a web element
- Fix *mouse_out* method
- *ElementList* is no longer a subclass of *list*
- *Browser.get_alert()* now waits for alert to present
- Use 'switch_to.alert' instead of deprecated 'switch_to_alert'

2.26.8 What's New in Splinter 0.12.0?

- *find_by_text* now handle strings with quotation marks (<https://github.com/cobrateam/splinter/issues/457>)
- *find_link_by* methods are now chainable (<https://github.com/cobrateam/splinter/pull/699>)
- *ElementList.__getattr__()* no longer hide *ElementNotFound* (<https://github.com/cobrateam/splinter/pull/707>)
- Firefox headless mode now handle custom *firefox_binary* option (<https://github.com/cobrateam/splinter/pull/714>)
- Firefox driver now respects headless option in subsequent calls (<https://github.com/cobrateam/splinter/pull/715>)
- *Browser.get_alert()* returns *None* if no alert exists (<https://github.com/cobrateam/splinter/issues/387>)
- Retry *WebElement.click* if *Exception* is thrown (<https://github.com/cobrateam/splinter/pull/725>)

- *find_by* methods in `WebDriverElement` now uses retry mechanism (<https://github.com/cobrateam/splinter/pull/727>)
- *is_not_present/visible* returns `True` immediately after not finding anything (<https://github.com/cobrateam/splinter/pull/732>)
- Accept all valid arguments for Remote `WebDriver` (<https://github.com/cobrateam/splinter/pull/734>)
- Allow `ActionChains` when using Remote `WebDriver` (<https://github.com/cobrateam/splinter/pull/738>)

2.26.9 What's New in Splinter 0.13.0?

- Patch Remote `WebDriver` to add retry attempts (<https://github.com/cobrateam/splinter/pull/742>)
- Add driver attribute to `WebDriverElement`. This fixes an issue where mouse interaction fails on nested elements (<https://github.com/cobrateam/splinter/pull/740>)
- Fix `WebDriverElement.select` and `.select_by_text` to search only inside the parent element (<https://github.com/cobrateam/splinter/pull/729>)
- *find_by* with 0 second *wait_time* only checks once (<https://github.com/cobrateam/splinter/pull/739>)
- Fix `FlaskClient` redirects (<https://github.com/cobrateam/splinter/pull/721>)

2.26.10 What's New in Splinter 0.14.0?

- Add `FindLinks` api to non-webdrivers (<https://github.com/cobrateam/splinter/pull/762>)
- Add support for `zope` in `python3` (<https://github.com/cobrateam/splinter/pull/771>)
- Fix `WebDriverElement.screenshot` when parent is a `WebDriverElement` (<https://github.com/cobrateam/splinter/pull/769>)
- Improve `firefox` headless support (<https://github.com/cobrateam/splinter/pull/768>)
- Fix mouse out on elements in the left corner of the viewport (<https://github.com/cobrateam/splinter/pull/766>)
- Fix `fullscreen` argument for `firefox` (<https://github.com/cobrateam/splinter/pull/765>)
- Fix unexpected keyword argument '`original_find`' (<https://github.com/cobrateam/splinter/pull/758>)
- Fix incorrect error thrown when missing `chrome/geckodriver` (<https://github.com/cobrateam/splinter/pull/749>)
- Make *find_by_value* works with button elements (<https://github.com/cobrateam/splinter/pull/746>)

2.26.11 What's New in Splinter 0.15.0?

- Add more input types to `Webdriver clear()` (<https://github.com/cobrateam/splinter/pull/780>)
- Standardize init of `CookieManager` (<https://github.com/cobrateam/splinter/pull/795>)
- Add `delete_all` method to `CookieManager` (<https://github.com/cobrateam/splinter/pull/797>)
- Warn user when cookies list is used (<https://github.com/cobrateam/splinter/pull/801>)
- Added `retry_count` to `get_driver` (<https://github.com/cobrateam/splinter/pull/754>)
- Fix full screen screenshot (<https://github.com/cobrateam/splinter/pull/810>)
- Add flag to ignore missing fields in `fill_form` (<https://github.com/cobrateam/splinter/pull/821>)
- Opening a link in a new tab (<https://github.com/cobrateam/splinter/pull/800>)

2.26.12 What's New in Splinter 0.16.0?

- Pin Selenium < 4.0 (<https://github.com/cobrateam/splinter/pull/930>)
- Add support for Microsoft Edge (<https://github.com/cobrateam/splinter/pull/912>)
- Accept extra arguments for cookies (<https://github.com/cobrateam/splinter/pull/895>)
- Fix lxmldriver url join when form action is empty (<https://github.com/cobrateam/splinter/pull/900>)
- Use io.open() to fix encoding issues on some platforms (<https://github.com/cobrateam/splinter/pull/904>)
- allow passing options to Firefox webdriver (<https://github.com/cobrateam/splinter/pull/892>)

Backward incompatible changes

- Remove sending a list of cookie dicts to CookieManager.add() (<https://github.com/cobrateam/splinter/pull/799>)

2.26.13 What's New in Splinter 0.17.0?

- Added parameter to DriverAPI.screenshot and ElementAPI.screenshot to indicate if unique filename should be ensured (<https://github.com/cobrateam/splinter/pull/949>)
- Added Selenium 4 support

Backward incompatible changes

- Removed python 2.7 support (<https://github.com/cobrateam/splinter/pull/952>)
- Selenium 3 is no longer installed by default. To install Selenium 3, use the *selenium3* extra argument

```
python -m pip install splinter[selenium3]
```

2.26.14 What's New in Splinter 0.18.0?

Added: * WebDriverElement() now implements the *shadow_root* property. This returns a ShadowRootElement() object to interact with the shadow root of an element. * Failed driver imports are logged at the debug level instead of silently ignored * *browser.html_snapshot()* now takes the optional *unique_file* argument. Setting this to False will disable the addition of random characters to the filename.

Changed: * repr(ElementList()) now returns the repr of the internal container. * Driver.find_link_by_<x> methods have been removed. Use Driver.links.find_by_<x>. * Screenshot taken by WebDriverElement.screenshot() now implements Selenium's element screenshot instead of cropping a full page screenshot. * Flask/Django's back/forward methods more accurately store browsing history * Official Python 3.6 support has been removed

Fixed: * 0.17.0 would report as 0.16.0. 0.18.0 reports correctly. * When using Firefox, extensions can now be installed

2.26.15 What's New in Splinter 0.2?

Features

- *cookies manipulation*
- find elements within an element

- improvements in *ElementList*

Backward incompatible changes

- you should update your selenium to 2.1.0 version and your chrome driver. See more in *suport to new chrome driver*

2.26.16 What's New in Splinter 0.3?

Features

- support for browser extensions on *Firefox driver*
- support for Firefox profiles on *Firefox driver*
- support for mouse over and mouse out on *Chrome driver*
- support for finding and clicking links by partial text and href
- support for *finding by value*

Documentation improvements

- complete *API reference*
- instructions on *new drivers creation*

Backward incompatible changes

- changes on *cookies manipulation*. Affects only who used `cookies.delete` passing the `cookie` keyword.

Before version **0.3**:

```
>>> driver.cookies.delete(cookie='whatever')
```

Now:

```
>>> driver.cookies.delete('whatever')
```

Bugfixes

- Fixed cookies behavior on Chrome driver (it was impossible to delete one cookie, Chrome was always deleting all cookies)

2.26.17 What's New in Splinter 0.4?

Features

- support for double click, right click, drag and drop and other *mouse interactions* (only *Chrome driver*)
- support for Python 2.5

Documentation improvements

- improved *API docs*
- added docs for `is_text_present` method
- added API docs for `is_element_present_by_*` methods
- added docs for *mouse interactions*

Deprecations

- simplified name of Selenium drivers, they're just `chrome` and `firefox` now (instead of `webdriver.chrome` and `webdriver.firefox`). The older names were deprecated.
- changed name of `mouseover` and `mouseout` methods to `mouse_over` and `mouse_out`

IMPORTANT

The following deprecated methods will be **removed** in the next splinter release (0.5) from Browser classes:

- `fill_in`
- `find_by_css_selector`
- `is_element_present_by_css_selector`
- `is_element_not_present_by_css_selector`

2.26.18 What's New in Splinter 0.4.1?

Features

- Partial Windows support
- Internet Explorer driver
- Added `type` and `fill` methods to *ElementAPI*.
- Updated selenium to 2.13.1

2.26.19 What's New in Splinter 0.4.10?

This version does not work with firefox 17.

Improvements

- remove deprecated driver names
- update lxml version
- update selenium version to 2.29

Bugfix

- set user-agent for request_handler requests
- update zope.testbrowser documentation regarding dependencies (cssselect)
- fix URL checking in request_handler (support for HTTPS)

2.26.20 What's New in Splinter 0.4.2?

Features

- added new *browser* method *form_fill* to fill all form fields in one command

Bugfixes

- fixed a bug in setup.py

2.26.21 What's New in Splinter 0.4.3?

Features

- Updated selenium to 2.14

2.26.22 What's New in Splinter 0.4.4?

Features

- Updated selenium to 2.17
- Method to change user-agent
- *dismiss* method in alert element

Bugfixes

- request_handler now works with querystring

2.26.23 What's New in Splinter 0.4.4.1?

Bugfixes

- update selenium version, to work with latest Firefox version

2.26.24 What's New in Splinter 0.4.7?

Features

- `has_class` method on `Element`
- fix documentation

Bugfixes and improvements

- improving `find_by_css` method to use native methods from drivers

2.26.25 What's New in Splinter 0.4.8?

Features

- `html` and `outer_html` property on `Element`
- `profile_preferences` option to Firefox driver
- Support for handling browser pop-up windows for Firefox/Chrome drivers.

2.26.26 What's New in Splinter 0.4.9?

This version does not works with firefox 17.

Features

- support for selenium remote web driver.

Bugfix

- `is_text_present` and `is_text_not_present` works with html without body.
- fixed `zopetestdriver` `attach_file` behaviour.

2.26.27 What's New in Splinter 0.5.0?

Features

- support for phantomjs web driver.
- `zopetestdriver` support `is_text_present`.

Bugfix

- fixed an unicode issue with `setup.py`.

2.26.28 What's New in Splinter 0.5.2?

Improvements

- support password field.

2.26.29 What's New in Splinter 0.5.3?

Improvement

- added kwargs to the Chrome driver constructor
- updated selenium to 2.33.0.

Bugfix

- fixed `about:blank` behaviour #233.

2.26.30 What's New in Splinter 0.5.4?

Improvement

- implemented `browser.cookies.all()` - #240.

Bugfix

- `browser.type()` works with textarea - #216.

2.26.31 What's New in Splinter 0.5.5?

Improvements

- Handle “internet explorer” as remote driver.
- implemented `get_screenshot_as_file`.
- `fill_form` now supports custom field types.
- More robust `find_link_by_partial_text`.
- support for selenium 2.39.0.
- support for zope.testbrowser 4.0.4.

2.26.32 What's New in Splinter 0.6.0?

Features

- support for django test client.

2.26.33 What's New in Splinter 0.7.0?

Features

- Support for `mouse_over`, `mouse_out` in Firefox driver.
- New flask test client driver.
- Better support for browser windows.
- Support for custom headers in PhantomJS driver.
- Added webdriver fullscreen support.
- Added a way to wait until element is visible.

Bugfix

- Support encoding in django client and zopetestbrowser drivers.
- `Browser.cookies.all()` are more consistent and added a verbose mode.

2.26.34 What's New in Splinter 0.7.1?

- support Selenium 2.45.0.
- Django Client supports `**kwargs` parameters on constructor.
- Django Client handle redirects.
- ZopeTestBrowser has the `ignore_robots` parameter.

2.26.35 What's New in Splinter 0.7.2?

- fix Python 3 compatibility, improving encoding/decoding in `browser.title` and `browser.html` - #380

2.26.36 What's New in Splinter 0.7.3?

- support selenium 2.47.1
- add `select_by_text` method
- add `find_by_text`, `is_element_present_by_text`, `is_element_not_present_by_text`
- improved support to python 3
- cookie support for remote webdriver
- get `status_code` by lazy evaluation. It should minimize the proxy and duplicated requests problems

django client

- improved `is_text_present` performance. `djangoclient` doesn't have to wait for load
- support django 1.7 and 1.8
- fixed several bugs with python3 compatibility

- added default extra headers: *SERVER_PORT*, *SERVER_NAME* and *User-Agent*
- support custom headers

2.26.37 What's New in Splinter 0.7.4?

- support Selenium 2.53.6
- `find_by_text` support quotes (#420).
- Selenium capabilities for Firefox driver (#417).
- multi-select support for Django and Flask (#443).
- custom headers support to Flask (#444).
- add *in* operation for cookies (#445).
- Support for *is_element_present_by_** in non-javascript drivers (#463).
- incognito mode for Google Chrome (#465).
- support for clearing text field types (#479).
- allow to pass a chrome Options instance to Browser (#494).
- new `click_link_by_id` method (#498).

Backward incompatible changes

- RequestHandler is removed and the *status* use lazy evaluation.

2.26.38 What's New in Splinter 0.7.5?

- Timeout settings for Firefox driver
- Remove default icognito mode in Chrome driver
- Make input a contro element in *django*, *flask* and *zope.testbrowser*

2.26.39 What's New in Splinter 0.7.6?

- fix *fill_form* for *select* element.
- support chrome headless mode

2.26.40 What's New in Splinter 0.7.7?

- *fill_form* more robust by requiring form ID
- support firefox *headless mode*
- handle exceptions when calling quit on webdriver

2.26.41 What's New in Splinter 0.8.0?

- add support for Firefox incognito mode (<https://github.com/cobrateam/splinter/pull/578>)
- allow return value for `execute_script` to be returned (<https://github.com/cobrateam/splinter/pull/585>)
- `chrome_options` parameter renamed to `options` (<https://github.com/cobrateam/splinter/pull/590>)
- removed deprecated `mouseover` method
- raises `NotImplementedError` on `status_code` in drivers based on webdriver
- `phantomjs` is deprecated (this driver will be removed in 0.9.0)

2.26.42 What's New in Splinter 0.9.0?

- `phantomjs` support was removed (<https://github.com/cobrateam/splinter/issues/592>)
- add options argument for chrome driver (<https://github.com/cobrateam/splinter/pull/345>)
- (bugfix) avoid `element.find_by_text` searches whole dom (<https://github.com/cobrateam/splinter/issues/612>)
- add suport for zope.testbrowser 5+
- handle webdriver `StaleElementReferenceException` (<https://github.com/cobrateam/splinter/issues/541>)
- add support for Flask 1+
- add support for selenium 3.14.0
- update lxml to 4.2.4
- update cssselect to 1.0.3

2.27 Community

2.27.1 mailing list

- [splinter-users](#) - list for help and announcements
- [splinter-developers](#) - where the developers of splinter itself discuss new features

2.27.2 irc channel

#cobrateam channel on irc.freenode.net - chat with other splinter users and developers

2.27.3 ticket system

[ticket system](#) - report bugs and make feature requests

2.27.4 splinter around the world

Projects using splinter

- [salad](#): splinter and lettuce integration

Blog posts

- [Django Full Stack Testing and BDD with Lettuce and Splinter](#)

Slides and talks

- [\[pt-br\] Os complicados testes de interface](#)
- [\[pt-br\] Testes de aceitação com Lettuce e Splinter](#)

2.28 Contribute

- Source hosted at [GitHub](#)
- Report issues on [GitHub Issues](#)

Pull requests are very welcome! Make sure your patches are well tested and documented :)

If you want to add any new driver, check out our *[docs for creating new splinter drivers](#)*.

2.28.1 running the tests

If you are using a virtualenv, all you need is:

```
$ make test
```

You can also specify one or more test files to run:

```
$ make test which=tests/test_webdriver_firefox.py,tests/test_request_handler.py
```

You can pass which test files you want to run, separated by comma, to the `which` variable.

2.28.2 some conventions we like

You can feel free to create and pull request new branches to Splinter project. When adding support for new drivers, we usually work in a separated branch.

2.28.3 writing docs

Splinter documentation is written using [Sphinx](#), which uses [RST](#). We use the [Read the Docs Sphinx Theme](#). Check these tools' docs to learn how to write docs for Splinter.

2.28.4 building docs

In order to build the HTML docs, just navigate to the project folder (the main folder, not the `docs` folder) and run the following on the terminal:

```
$ make doc
```

The requirements for building the docs are specified in `doc-requirements.txt` in the project folder.

2.29 Writing new splinter drivers

The process of creating a new splinter browser is really simple: you just need to implement a `TestCase` (extending `tests.base.BaseBrowserTests`) and make all tests green. For example:

Imagine you're creating the Columbia driver, you would add the `test_columbia.py` file containing some code like...

```
from splinter import Browser
from tests.base import BaseBrowserTests

class ColumbiaTest(BaseBrowserTests):

    @classmethod
    def setUpClass(cls):
        cls.browser = Browser('columbia')

    # ...
```

Now, to make the test green, you need to implement methods provided by the [DriverAPI](#) and the [ElementAPI](#).

Use `make test` to run the tests:

```
$ make test which=tests/test_columbia.py
```

2.30 Setting up your splinter development environment

Setting up a splinter development environment is a really easy task. Once you have some basic development tools on your machine, you can set up the entire environment with just one command.

2.30.1 Basic development tools

Let's deal with those tools first.

macOS

If you're a macOS user, you just need to install Xcode, which can be downloaded from Mac App Store (on Snow Leopard or later) or from [Apple website](#).

Linux

If you are running a Linux distribution, you need to install some basic development libraries and headers. For example, on Ubuntu, you can easily install all of them using `apt-get`:

```
$ [sudo] apt-get install build-essential python-dev libxml2-dev libxslt1-dev
```

PIP and virtualenv

Make sure you have pip installed. We manage all splinter development dependencies with [PIP](#), so you should use it too.

And please, for the sake of a nice development environment, use [virtualenv](#). If you aren't using it yet, start now. :)

Dependencies

Once you have all development libraries installed for your OS, just install all splinter development dependencies with `make`:

```
$ [sudo] make dependencies
```

Note: You will need `sudo` only if you aren't using `virtualenv` (which means you're a really bad guy - *no donuts for you*).

Also make sure you have properly configured your [Chrome driver](#).

S

- `splinter.browser`, [63](#)
- `splinter.cookie_manager`, [72](#)
- `splinter.driver`, [63](#)
- `splinter.driver.djangoclient`, [52](#)
- `splinter.driver.flaskclient`, [56](#)
- `splinter.driver.zopetestbrowser`, [47](#)
- `splinter.element_list`, [73](#)
- `splinter.exceptions`, [74](#)
- `splinter.request_handler.status_code`,
[74](#)

A

[add\(\)](#) ([splinter.cookie_manager.CookieManagerAPI method](#)), 72
[all\(\)](#) ([splinter.cookie_manager.CookieManagerAPI method](#)), 72
[attach_file\(\)](#) ([splinter.driver.webdriver.chrome.WebDriver method](#)), 20
[attach_file\(\)](#) ([splinter.driver.webdriver.edge.WebDriver method](#)), 39
[attach_file\(\)](#) ([splinter.driver.webdriver.firefox.WebDriver method](#)), 29

B

[back\(\)](#) ([splinter.driver.djangoclient.DjangoClient method](#)), 52
[back\(\)](#) ([splinter.driver.DriverAPI method](#)), 63
[back\(\)](#) ([splinter.driver.flaskclient.FlaskClient method](#)), 57
[back\(\)](#) ([splinter.driver.webdriver.chrome.WebDriver method](#)), 20
[back\(\)](#) ([splinter.driver.webdriver.edge.WebDriver method](#)), 39
[back\(\)](#) ([splinter.driver.webdriver.firefox.WebDriver method](#)), 29
[back\(\)](#) ([splinter.driver.zopetestbrowser.ZopeTestBrowser method](#)), 48
[Browser\(\)](#) (in module [splinter.browser](#)), 63

C

[check\(\)](#) ([splinter.driver.djangoclient.DjangoClient method](#)), 52
[check\(\)](#) ([splinter.driver.DriverAPI method](#)), 63
[check\(\)](#) ([splinter.driver.ElementAPI method](#)), 70
[check\(\)](#) ([splinter.driver.flaskclient.FlaskClient method](#)), 57
[check\(\)](#) ([splinter.driver.webdriver.chrome.WebDriver method](#)), 20

[check\(\)](#) ([splinter.driver.webdriver.edge.WebDriver method](#)), 39
[check\(\)](#) ([splinter.driver.webdriver.firefox.WebDriver method](#)), 29
[check\(\)](#) ([splinter.driver.zopetestbrowser.ZopeTestBrowser method](#)), 48
[checked](#) ([splinter.driver.ElementAPI attribute](#)), 70
[choose\(\)](#) ([splinter.driver.djangoclient.DjangoClient method](#)), 53
[choose\(\)](#) ([splinter.driver.DriverAPI method](#)), 63
[choose\(\)](#) ([splinter.driver.flaskclient.FlaskClient method](#)), 57
[choose\(\)](#) ([splinter.driver.webdriver.chrome.WebDriver method](#)), 21
[choose\(\)](#) ([splinter.driver.webdriver.edge.WebDriver method](#)), 39
[choose\(\)](#) ([splinter.driver.webdriver.firefox.WebDriver method](#)), 30
[choose\(\)](#) ([splinter.driver.zopetestbrowser.ZopeTestBrowser method](#)), 48
[clear\(\)](#) ([splinter.driver.ElementAPI method](#)), 71
[click\(\)](#) ([splinter.driver.ElementAPI method](#)), 71
[code](#) ([splinter.request_handler.status_code.StatusCode attribute](#)), 73
[CookieManagerAPI](#) (class in [splinter.cookie_manager](#)), 72
[cookies](#) ([splinter.driver.djangoclient.DjangoClient attribute](#)), 53
[cookies](#) ([splinter.driver.DriverAPI attribute](#)), 64
[cookies](#) ([splinter.driver.flaskclient.FlaskClient attribute](#)), 57
[cookies](#) ([splinter.driver.webdriver.chrome.WebDriver attribute](#)), 21
[cookies](#) ([splinter.driver.webdriver.edge.WebDriver attribute](#)), 39
[cookies](#) ([splinter.driver.webdriver.firefox.WebDriver attribute](#)), 30
[cookies](#) ([splinter.driver.zopetestbrowser.ZopeTestBrowser attribute](#)), 48

D

`delete()` (*splinter.cookie_manager.CookieManagerAPI method*), 72

`delete_all()` (*splinter.cookie_manager.CookieManagerAPI method*), 73

`DjangoClient` (class in *splinter.driver.djangoclient*), 52

`DriverAPI` (class in *splinter.driver*), 63

`DriverNotFoundError` (class in *splinter.exceptions*), 74

E

`ElementAPI` (class in *splinter.driver*), 70

`ElementDoesNotExist` (class in *splinter.exceptions*), 74

`ElementList` (class in *splinter.element_list*), 73

`evaluate_script()` (*splinter.driver.DriverAPI method*), 64

`evaluate_script()` (*splinter.driver.webdriver.chrome.WebDriver method*), 21

`evaluate_script()` (*splinter.driver.webdriver.edge.WebDriver method*), 39

`evaluate_script()` (*splinter.driver.webdriver.firefox.WebDriver method*), 30

`execute_script()` (*splinter.driver.DriverAPI method*), 64

`execute_script()` (*splinter.driver.webdriver.chrome.WebDriver method*), 21

`execute_script()` (*splinter.driver.webdriver.edge.WebDriver method*), 40

`execute_script()` (*splinter.driver.webdriver.firefox.WebDriver method*), 30

F

`fill()` (*splinter.driver.djangoclient.DjangoClient method*), 53

`fill()` (*splinter.driver.DriverAPI method*), 64

`fill()` (*splinter.driver.ElementAPI method*), 71

`fill()` (*splinter.driver.flaskclient.FlaskClient method*), 58

`fill()` (*splinter.driver.webdriver.chrome.WebDriver method*), 21

`fill()` (*splinter.driver.webdriver.edge.WebDriver method*), 40

`fill()` (*splinter.driver.webdriver.firefox.WebDriver method*), 30

`fill()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 48

`fill_form()` (*splinter.driver.djangoclient.DjangoClient method*), 53

`fill_form()` (*splinter.driver.DriverAPI method*), 64

`fill_form()` (*splinter.driver.flaskclient.FlaskClient method*), 58

`fill_form()` (*splinter.driver.webdriver.chrome.WebDriver method*), 22

`fill_form()` (*splinter.driver.webdriver.edge.WebDriver method*), 40

`fill_form()` (*splinter.driver.webdriver.firefox.WebDriver method*), 31

`fill_form()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 49

`find_by()` (*splinter.driver.webdriver.chrome.WebDriver method*), 22

`find_by()` (*splinter.driver.webdriver.edge.WebDriver method*), 40

`find_by()` (*splinter.driver.webdriver.firefox.WebDriver method*), 31

`find_by_css()` (*splinter.driver.djangoclient.DjangoClient method*), 53

`find_by_css()` (*splinter.driver.DriverAPI method*), 65

`find_by_css()` (*splinter.driver.flaskclient.FlaskClient method*), 58

`find_by_css()` (*splinter.driver.webdriver.chrome.WebDriver method*), 22

`find_by_css()` (*splinter.driver.webdriver.edge.WebDriver method*), 40

`find_by_css()` (*splinter.driver.webdriver.firefox.WebDriver method*), 31

`find_by_css()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 49

`find_by_id()` (*splinter.driver.djangoclient.DjangoClient method*), 53

`find_by_id()` (*splinter.driver.DriverAPI method*), 65

`find_by_id()` (*splinter.driver.flaskclient.FlaskClient method*), 58

`find_by_id()` (*splinter.driver.webdriver.chrome.WebDriver*

| | | | |
|---|--|--|--|
| <i>method</i>), 22 | | 65 | |
| <code>find_by_id()</code> (<i>splinter.driver.webdriver.edge.WebDriver method</i>), 40 | | <code>find_by_text()</code> (<i>splinter.driver.flaskclient.FlaskClient method</i>), 58 | |
| <code>find_by_id()</code> (<i>splinter.driver.webdriver.firefox.WebDriver method</i>), 31 | | <code>find_by_text()</code> (<i>splinter.driver.webdriver.chrome.WebDriver method</i>), 22 | |
| <code>find_by_id()</code> (<i>splinter.driver.zopetestbrowser.ZopeTestBrowser method</i>), 49 | | <code>find_by_text()</code> (<i>splinter.driver.webdriver.edge.WebDriver method</i>), 41 | |
| <code>find_by_name()</code> (<i>splinter.driver.djangoclient.DjangoClient method</i>), 54 | | <code>find_by_text()</code> (<i>splinter.driver.webdriver.firefox.WebDriver method</i>), 31 | |
| <code>find_by_name()</code> (<i>splinter.driver.DriverAPI method</i>), 65 | | <code>find_by_text()</code> (<i>splinter.driver.zopetestbrowser.ZopeTestBrowser method</i>), 49 | |
| <code>find_by_name()</code> (<i>splinter.driver.flaskclient.FlaskClient method</i>), 58 | | <code>find_by_value()</code> (<i>splinter.driver.djangoclient.DjangoClient method</i>), 54 | |
| <code>find_by_name()</code> (<i>splinter.driver.webdriver.chrome.WebDriver method</i>), 22 | | <code>find_by_value()</code> (<i>splinter.driver.DriverAPI method</i>), 65 | |
| <code>find_by_name()</code> (<i>splinter.driver.webdriver.edge.WebDriver method</i>), 40 | | <code>find_by_value()</code> (<i>splinter.driver.flaskclient.FlaskClient method</i>), 58 | |
| <code>find_by_name()</code> (<i>splinter.driver.webdriver.firefox.WebDriver method</i>), 31 | | <code>find_by_value()</code> (<i>splinter.driver.webdriver.chrome.WebDriver method</i>), 22 | |
| <code>find_by_name()</code> (<i>splinter.driver.zopetestbrowser.ZopeTestBrowser method</i>), 49 | | <code>find_by_value()</code> (<i>splinter.driver.webdriver.edge.WebDriver method</i>), 41 | |
| <code>find_by_tag()</code> (<i>splinter.driver.djangoclient.DjangoClient method</i>), 54 | | <code>find_by_value()</code> (<i>splinter.driver.webdriver.firefox.WebDriver method</i>), 31 | |
| <code>find_by_tag()</code> (<i>splinter.driver.DriverAPI method</i>), 65 | | <code>find_by_value()</code> (<i>splinter.driver.zopetestbrowser.ZopeTestBrowser method</i>), 49 | |
| <code>find_by_tag()</code> (<i>splinter.driver.flaskclient.FlaskClient method</i>), 58 | | <code>find_by_xpath()</code> (<i>splinter.driver.djangoclient.DjangoClient method</i>), 54 | |
| <code>find_by_tag()</code> (<i>splinter.driver.webdriver.chrome.WebDriver method</i>), 22 | | <code>find_by_xpath()</code> (<i>splinter.driver.DriverAPI method</i>), 65 | |
| <code>find_by_tag()</code> (<i>splinter.driver.webdriver.edge.WebDriver method</i>), 41 | | <code>find_by_xpath()</code> (<i>splinter.driver.flaskclient.FlaskClient method</i>), 59 | |
| <code>find_by_tag()</code> (<i>splinter.driver.webdriver.firefox.WebDriver method</i>), 31 | | <code>find_by_xpath()</code> (<i>splinter.driver.webdriver.chrome.WebDriver method</i>), 22 | |
| <code>find_by_tag()</code> (<i>splinter.driver.zopetestbrowser.ZopeTestBrowser method</i>), 49 | | <code>find_by_xpath()</code> (<i>splinter.driver.webdriver.edge.WebDriver method</i>), 41 | |
| <code>find_by_text()</code> (<i>splinter.driver.djangoclient.DjangoClient method</i>), 54 | | <code>find_by_xpath()</code> (<i>splinter.driver.webdriver.firefox.WebDriver method</i>), 31 | |
| <code>find_by_text()</code> (<i>splinter.driver.DriverAPI method</i>), | | <code>find_by_xpath()</code> (<i>splinter-</i> | |

ter.driver.zopetestbrowser.ZopeTestBrowser
method), 49
find_option_by_text() (*splinter.driver.djangoclient.DjangoClient* *method*), 54
find_option_by_text() (*splinter.driver.DriverAPI* *method*), 65
find_option_by_text() (*splinter.driver.flaskclient.FlaskClient* *method*), 59
find_option_by_text() (*splinter.driver.webdriver.chrome.WebDriver* *method*), 23
find_option_by_text() (*splinter.driver.webdriver.edge.WebDriver* *method*), 41
find_option_by_text() (*splinter.driver.webdriver.firefox.WebDriver* *method*), 32
find_option_by_text() (*splinter.driver.zopetestbrowser.ZopeTestBrowser* *method*), 49
find_option_by_value() (*splinter.driver.djangoclient.DjangoClient* *method*), 54
find_option_by_value() (*splinter.driver.DriverAPI* *method*), 65
find_option_by_value() (*splinter.driver.flaskclient.FlaskClient* *method*), 59
find_option_by_value() (*splinter.driver.webdriver.chrome.WebDriver* *method*), 23
find_option_by_value() (*splinter.driver.webdriver.edge.WebDriver* *method*), 41
find_option_by_value() (*splinter.driver.webdriver.firefox.WebDriver* *method*), 32
find_option_by_value() (*splinter.driver.zopetestbrowser.ZopeTestBrowser* *method*), 50
first (*splinter.element_list.ElementList* *attribute*), 73
FlaskClient (*class in splinter.driver.flaskclient*), 57
forward() (*splinter.driver.djangoclient.DjangoClient* *method*), 54
forward() (*splinter.driver.DriverAPI* *method*), 65
forward() (*splinter.driver.flaskclient.FlaskClient* *method*), 59
forward() (*splinter.driver.webdriver.chrome.WebDriver* *method*), 23
forward() (*splinter.driver.webdriver.edge.WebDriver* *method*), 41
forward() (*splinter.driver.webdriver.firefox.WebDriver* *method*), 32
forward() (*splinter.driver.zopetestbrowser.ZopeTestBrowser* *method*), 50
get_alert() (*splinter.driver.djangoclient.DjangoClient* *method*), 54
get_alert() (*splinter.driver.DriverAPI* *method*), 65
get_alert() (*splinter.driver.flaskclient.FlaskClient* *method*), 59
get_alert() (*splinter.driver.webdriver.chrome.WebDriver* *method*), 23
get_alert() (*splinter.driver.webdriver.edge.WebDriver* *method*), 41
get_alert() (*splinter.driver.webdriver.firefox.WebDriver* *method*), 32
get_alert() (*splinter.driver.zopetestbrowser.ZopeTestBrowser* *method*), 50
get_iframe() (*splinter.driver.djangoclient.DjangoClient* *method*), 54
get_iframe() (*splinter.driver.DriverAPI* *method*), 66
get_iframe() (*splinter.driver.flaskclient.FlaskClient* *method*), 59
get_iframe() (*splinter.driver.webdriver.chrome.WebDriver* *method*), 23
get_iframe() (*splinter.driver.webdriver.edge.WebDriver* *method*), 41
get_iframe() (*splinter.driver.webdriver.firefox.WebDriver* *method*), 32
get_iframe() (*splinter.driver.zopetestbrowser.ZopeTestBrowser* *method*), 50
G
H
has_class() (*splinter.driver.ElementAPI* *method*), 71
html (*splinter.driver.djangoclient.DjangoClient* *attribute*), 54
html (*splinter.driver.DriverAPI* *attribute*), 66
html (*splinter.driver.flaskclient.FlaskClient* *attribute*), 59
html (*splinter.driver.webdriver.chrome.WebDriver* *attribute*), 23
html (*splinter.driver.webdriver.edge.WebDriver* *attribute*), 41

`html` (*splinter.driver.webdriver.firefox.WebDriver* attribute), 32
`html` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* attribute), 50
`html_snapshot()` (*splinter.driver.djangoclient.DjangoClient* method), 55
`html_snapshot()` (*splinter.driver.DriverAPI* method), 66
`html_snapshot()` (*splinter.driver.flaskclient.FlaskClient* method), 59
`html_snapshot()` (*splinter.driver.webdriver.chrome.WebDriver* method), 23
`html_snapshot()` (*splinter.driver.webdriver.edge.WebDriver* method), 41
`html_snapshot()` (*splinter.driver.webdriver.firefox.WebDriver* method), 32
`html_snapshot()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* method), 50
I
`is_element_not_present_by_css()` (*splinter.driver.DriverAPI* method), 66
`is_element_not_present_by_css()` (*splinter.driver.webdriver.chrome.WebDriver* method), 23
`is_element_not_present_by_css()` (*splinter.driver.webdriver.edge.WebDriver* method), 42
`is_element_not_present_by_css()` (*splinter.driver.webdriver.firefox.WebDriver* method), 32
`is_element_not_present_by_id()` (*splinter.driver.DriverAPI* method), 66
`is_element_not_present_by_id()` (*splinter.driver.webdriver.chrome.WebDriver* method), 23
`is_element_not_present_by_id()` (*splinter.driver.webdriver.edge.WebDriver* method), 42
`is_element_not_present_by_id()` (*splinter.driver.webdriver.firefox.WebDriver* method), 32
`is_element_not_present_by_name()` (*splinter.driver.DriverAPI* method), 66
`is_element_not_present_by_name()` (*splinter.driver.webdriver.chrome.WebDriver* method), 24
`is_element_not_present_by_name()` (*splinter.driver.webdriver.edge.WebDriver* method), 42
`is_element_not_present_by_name()` (*splinter.driver.webdriver.firefox.WebDriver* method), 33
`is_element_not_present_by_tag()` (*splinter.driver.DriverAPI* method), 67
`is_element_not_present_by_tag()` (*splinter.driver.webdriver.chrome.WebDriver* method), 24
`is_element_not_present_by_tag()` (*splinter.driver.webdriver.edge.WebDriver* method), 42
`is_element_not_present_by_tag()` (*splinter.driver.webdriver.firefox.WebDriver* method), 33
`is_element_not_present_by_text()` (*splinter.driver.DriverAPI* method), 67
`is_element_not_present_by_text()` (*splinter.driver.webdriver.chrome.WebDriver* method), 24
`is_element_not_present_by_text()` (*splinter.driver.webdriver.edge.WebDriver* method), 42
`is_element_not_present_by_text()` (*splinter.driver.webdriver.firefox.WebDriver* method), 33
`is_element_not_present_by_value()` (*splinter.driver.DriverAPI* method), 67
`is_element_not_present_by_value()` (*splinter.driver.webdriver.chrome.WebDriver* method), 24
`is_element_not_present_by_value()` (*splinter.driver.webdriver.edge.WebDriver* method), 43
`is_element_not_present_by_value()` (*splinter.driver.webdriver.firefox.WebDriver* method), 33
`is_element_not_present_by_xpath()` (*splinter.driver.DriverAPI* method), 67
`is_element_not_present_by_xpath()` (*splinter.driver.webdriver.chrome.WebDriver* method), 24
`is_element_not_present_by_xpath()` (*splinter.driver.webdriver.edge.WebDriver* method), 43
`is_element_not_present_by_xpath()` (*splinter.driver.webdriver.firefox.WebDriver* method), 33
`is_element_present_by_css()` (*splinter.driver.DriverAPI* method), 67
`is_element_present_by_css()` (*splinter.driver.webdriver.chrome.WebDriver*

method), 25

`is_element_present_by_css()` (*splinter.driver.webdriver.edge.WebDriver method*), 43

`is_element_present_by_css()` (*splinter.driver.webdriver.firefox.WebDriver method*), 34

`is_element_present_by_id()` (*splinter.driver.DriverAPI method*), 67

`is_element_present_by_id()` (*splinter.driver.webdriver.chrome.WebDriver method*), 25

`is_element_present_by_id()` (*splinter.driver.webdriver.edge.WebDriver method*), 43

`is_element_present_by_id()` (*splinter.driver.webdriver.firefox.WebDriver method*), 34

`is_element_present_by_name()` (*splinter.driver.DriverAPI method*), 68

`is_element_present_by_name()` (*splinter.driver.webdriver.chrome.WebDriver method*), 25

`is_element_present_by_name()` (*splinter.driver.webdriver.edge.WebDriver method*), 43

`is_element_present_by_name()` (*splinter.driver.webdriver.firefox.WebDriver method*), 34

`is_element_present_by_tag()` (*splinter.driver.DriverAPI method*), 68

`is_element_present_by_tag()` (*splinter.driver.webdriver.chrome.WebDriver method*), 25

`is_element_present_by_tag()` (*splinter.driver.webdriver.edge.WebDriver method*), 44

`is_element_present_by_tag()` (*splinter.driver.webdriver.firefox.WebDriver method*), 34

`is_element_present_by_text()` (*splinter.driver.DriverAPI method*), 68

`is_element_present_by_text()` (*splinter.driver.webdriver.chrome.WebDriver method*), 25

`is_element_present_by_text()` (*splinter.driver.webdriver.edge.WebDriver method*), 44

`is_element_present_by_text()` (*splinter.driver.webdriver.firefox.WebDriver method*), 34

`is_element_present_by_value()` (*splinter.driver.DriverAPI method*), 68

`is_element_present_by_value()` (*splinter.driver.webdriver.chrome.WebDriver method*), 25

`is_element_present_by_value()` (*splinter.driver.webdriver.edge.WebDriver method*), 44

`is_element_present_by_value()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`is_empty()` (*splinter.element_list.ElementList method*), 73

`is_success()` (*splinter.request_handler.status_code.StatusCode method*), 74

`is_text_present()` (*splinter.driver.djangoclient.DjangoClient method*), 55

`is_text_present()` (*splinter.driver.DriverAPI method*), 69

`is_text_present()` (*splinter.driver.flaskclient.FlaskClient method*), 59

`is_text_present()` (*splinter.driver.webdriver.chrome.WebDriver method*), 26

`is_text_present()` (*splinter.driver.webdriver.edge.WebDriver method*), 44

`is_text_present()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`is_text_present()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 50

L

`last` (*splinter.element_list.ElementList attribute*), 73

M

`mouse_out()` (*splinter.driver.ElementAPI method*), 71

`mouse_over()` (*splinter.driver.ElementAPI method*), 71

N

`new_tab()` (*splinter.driver.djangoclient.DjangoClient method*), 55

`new_tab()` (*splinter.driver.DriverAPI method*), 69

`new_tab()` (*splinter.driver.flaskclient.FlaskClient method*), 60

`new_tab()` (*splinter.driver.webdriver.chrome.WebDriver method*), 26

`new_tab()` (*splinter.driver.webdriver.edge.WebDriver method*), 44

`new_tab()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`new_tab()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 50

Q

`quit()` (*splinter.driver.djangoclient.DjangoClient method*), 55

`quit()` (*splinter.driver.DriverAPI method*), 69

`quit()` (*splinter.driver.flaskclient.FlaskClient method*), 60

`quit()` (*splinter.driver.webdriver.chrome.WebDriver method*), 26

`quit()` (*splinter.driver.webdriver.edge.WebDriver method*), 45

`quit()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`quit()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 50

R

`reason` (*splinter.request_handler.status_code.StatusCode attribute*), 74

`reload()` (*splinter.driver.djangoclient.DjangoClient method*), 55

`reload()` (*splinter.driver.DriverAPI method*), 69

`reload()` (*splinter.driver.flaskclient.FlaskClient method*), 60

`reload()` (*splinter.driver.webdriver.chrome.WebDriver method*), 26

`reload()` (*splinter.driver.webdriver.edge.WebDriver method*), 45

`reload()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`reload()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 51

S

`screenshot()` (*splinter.driver.djangoclient.DjangoClient method*), 55

`screenshot()` (*splinter.driver.DriverAPI method*), 69

`screenshot()` (*splinter.driver.ElementAPI method*), 71

`screenshot()` (*splinter.driver.flaskclient.FlaskClient method*), 60

`screenshot()` (*splinter.driver.webdriver.chrome.WebDriver method*), 26

`screenshot()` (*splinter.driver.webdriver.edge.WebDriver method*), 45

`screenshot()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`screenshot()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 51

`select()` (*splinter.driver.djangoclient.DjangoClient method*), 55

`select()` (*splinter.driver.DriverAPI method*), 69

`select()` (*splinter.driver.ElementAPI method*), 71

`select()` (*splinter.driver.flaskclient.FlaskClient method*), 60

`select()` (*splinter.driver.webdriver.chrome.WebDriver method*), 26

`select()` (*splinter.driver.webdriver.edge.WebDriver method*), 45

`select()` (*splinter.driver.webdriver.firefox.WebDriver method*), 35

`select()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser method*), 51

`shadow_root` (*splinter.driver.ElementAPI attribute*), 71

`splinter.browser` (*module*), 63

`splinter.cookie_manager` (*module*), 72

`splinter.driver` (*module*), 63

`splinter.driver.djangoclient` (*module*), 52

`splinter.driver.flaskclient` (*module*), 56

`splinter.driver.zopetestbrowser` (*module*), 47

`splinter.element_list` (*module*), 73

`splinter.exceptions` (*module*), 74

`splinter.request_handler.status_code` (*module*), 74

`StatusCode` (*class in splinter.request_handler.status_code*), 73

T

`text` (*splinter.driver.ElementAPI attribute*), 71

`title` (*splinter.driver.djangoclient.DjangoClient attribute*), 56

`title` (*splinter.driver.DriverAPI attribute*), 69

`title` (*splinter.driver.flaskclient.FlaskClient attribute*), 60

`title` (*splinter.driver.webdriver.chrome.WebDriver attribute*), 27

`title` (*splinter.driver.webdriver.edge.WebDriver* attribute), 45
`title` (*splinter.driver.webdriver.firefox.WebDriver* attribute), 36
`title` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* attribute), 51
`type()` (*splinter.driver.djangoclient.DjangoClient* method), 56
`type()` (*splinter.driver.DriverAPI* method), 69
`type()` (*splinter.driver.ElementAPI* method), 71
`type()` (*splinter.driver.flaskclient.FlaskClient* method), 60
`type()` (*splinter.driver.webdriver.chrome.WebDriver* method), 27
`type()` (*splinter.driver.webdriver.edge.WebDriver* method), 45
`type()` (*splinter.driver.webdriver.firefox.WebDriver* method), 36
`type()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* method), 51

U

`uncheck()` (*splinter.driver.djangoclient.DjangoClient* method), 56
`uncheck()` (*splinter.driver.DriverAPI* method), 70
`uncheck()` (*splinter.driver.ElementAPI* method), 71
`uncheck()` (*splinter.driver.flaskclient.FlaskClient* method), 60
`uncheck()` (*splinter.driver.webdriver.chrome.WebDriver* method), 27
`uncheck()` (*splinter.driver.webdriver.edge.WebDriver* method), 45
`uncheck()` (*splinter.driver.webdriver.firefox.WebDriver* method), 36
`uncheck()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* method), 51
`url` (*splinter.driver.djangoclient.DjangoClient* attribute), 56
`url` (*splinter.driver.DriverAPI* attribute), 70
`url` (*splinter.driver.flaskclient.FlaskClient* attribute), 61
`url` (*splinter.driver.webdriver.chrome.WebDriver* attribute), 27
`url` (*splinter.driver.webdriver.edge.WebDriver* attribute), 46
`url` (*splinter.driver.webdriver.firefox.WebDriver* attribute), 36
`url` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* attribute), 52

V

`value` (*splinter.driver.ElementAPI* attribute), 72
`visible` (*splinter.driver.ElementAPI* attribute), 72
`visit()` (*splinter.driver.djangoclient.DjangoClient* method), 56

`visit()` (*splinter.driver.DriverAPI* method), 70
`visit()` (*splinter.driver.flaskclient.FlaskClient* method), 61
`visit()` (*splinter.driver.webdriver.chrome.WebDriver* method), 27
`visit()` (*splinter.driver.webdriver.edge.WebDriver* method), 46
`visit()` (*splinter.driver.webdriver.firefox.WebDriver* method), 36
`visit()` (*splinter.driver.zopetestbrowser.ZopeTestBrowser* method), 52

W

`WebDriver` (class in *splinter.driver.webdriver.chrome*), 20
`WebDriver` (class in *splinter.driver.webdriver.edge*), 39
`WebDriver` (class in *splinter.driver.webdriver.firefox*), 29

Z

`ZopeTestBrowser` (class in *splinter.driver.zopetestbrowser*), 48