Browser

splinter.browser.Browser(*driver_name='firefox', *args, **kwargs*)

Returns a driver instance for the given name.

When working with `firefox`, it's possible to provide a profile name and a list of extensions.

If you don't provide any driver_name, then `firefox` will be used.

If there is no driver registered with the provided `driver_name`, this function will raise a `splinter.exceptions.DriverNotFoundError` exception.

DriverAPI

class **splinter.driver.DriverAPI**

Basic driver API class.

> **back**()
>
> Back to the last URL in the browsing history.
>
> If there is no URL to back, this method does nothing.

> **check**(*name*)
>
> Checks a checkbox by its name.
>
> Example:
>
> ```
> >>> browser.check("agree-with-terms")
> ```
>
> If you call `browser.check` n times, the checkbox keeps checked, it never get unchecked.
>
> To unckech a checkbox, take a look in the `uncheck` method.

> **choose**(*name*, *value*)

Chooses a value in a radio buttons group.

Suppose you have the two radio buttons in a page, with the name `gender` and values 'F' and 'M'. If you use the `choose` method the following way:

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

Then you're choosing the female gender.

`click_link_by_href`(*href*)

Clicks in a link by its `href` attribute.

`click_link_by_id`(*id*)

Clicks in a link by id.

`click_link_by_partial_href`(*partial_href*)

Clicks in a link by looking for partial content of `href` attribute.

`click_link_by_partial_text`(*partial_text*)

Clicks in a link by partial content of its text.

`click_link_by_text`(*text*)

Clicks in a link by its `text`.

`cookies`

A `CookieManager` instance.

For more details, check the cookies manipulation section.

`evaluate_script`(*script*, **args*)

Similar to `execute_script` method.

Executes javascript in the browser and returns the value of the expression.

e.g.: ::

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

execute_script(*script*, **args*)

Executes a given JavaScript in the browser.

e.g.:: ::

```
>>>
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`.

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.

find_by_css(*css_selector*)

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

find_by_id(*id*)

Finds an element in current page by its id.

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

find_by_name(*name*)

Finds elements in current page by their name.

Returns an instance of `ElementList`.

find_by_tag(*tag*)

Find all elements of a given tag in current page.

Returns an instance of `ElementList`

find_by_text(*text*)

Finds elements in current page by their text.

Returns an instance of `ElementList`

find_by_value(*value*)

Finds elements in current page by their value.

Returns an instance of `ElementList`

find_by_xpath(*xpath*)

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

find_link_by_href(*href*)

Find all elements of a given tag in current page.

Returns an instance of `ElementList`

find_link_by_partial_href(*partial_href*)

Find links by looking for a partial `str` in their href attribute.

Returns an instance of `ElementList`

find_link_by_partial_text(*partial_text*)

Find links by looking for a partial `str` in their text.

Returns an instance of `ElementList`

find_link_by_text(*text*)

Find links querying for their text.

Returns an instance of `ElementList`

find_option_by_text(*text*)

Finds `<option>` elements by their text.

Returns an instance of `ElementList`

`find_option_by_value`(*value*)

Finds `<option>` elements by their value.

Returns an instance of `ElementList`

`forward`()

Forward to the next URL in the browsing history.

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

`get_alert`()

Changes the context for working with alerts and prompts.

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

`get_iframe`(*name*)

Changes the context for working with iframes.

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

`html`

Source of current page.

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

Verify if the element is not present in the current page by css, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by id, and wait the specified time in `wait_time`.

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

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

Verify if the element is not present in the current page by name, and wait the specified time in `wait_time`.

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

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

Verify if the element is not present in the current page by tag, and wait the specified time in `wait_time`.

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

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

Verify if the element is not present in the current page by text, and wait the specified time in `wait_time`.

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

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

Verify if the element is not present in the current page by value, and wait the specified time in `wait_time`.

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

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

Verify if the element is not present in the current page by xpath, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by css, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by id, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by name, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by tag, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by text, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by value, and wait the specified time in `wait_time`.

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

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

Verify if the element is present in the current page by xpath, and wait the specified time in `wait_time`.

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

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

Searchs for `text` in the browser and wait the seconds specified in `wait_time`.

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

quit()

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

After quit the browser, you can't use it anymore.

reload()

Revisits the current URL

`screenshot`(*name=None, suffix=None*)

Takes a screenshot of the current page and saves it locally.

`select`(*name, value*)

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

Example:

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

`title`

Title of current page.

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

Types the `value` in the field identified by `name`.

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.

`uncheck`(*name*)

Unchecks a checkbox by its name.

Example:

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

If you call `brower.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*)

Visits a given URL.

The `url` parameter is a string.

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()

Checks 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 loot in the `uncheck` 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()

Clicks 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`()

Moves the mouse away from the element.

`mouse_over`()

Puts the mouse over the element.

`screenshot`()

Take screenshot of the element.

`select`(*value*, *slowly=False*)

Selects an `<option>` element in the element using the `value` of the `<option>`.

Example:

```
>>> element.select("NY")
```

`text`

String of all of the text within the element. HTML tags are stripped.

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

Types 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.

`uncheck`()

Unchecks 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 loot 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.