Elemental

Release 0.4.0

Keira & Sky

Feb 17, 2021

GETTING STARTED

Elemental makes Selenium automation faster and easier.

Adds common use-cases Common use-cases missing from Selenium are built into Elemental. *Get an input by its label or placeholder*? Can do. *Get a button by its text or type?* Sure. *Get an element's parent?* Yep. *Get the fourth element in a list?* No problem.

Automatic waiting Elemental has built-in automatic waiting so that your automation is less flaky. It has a sensible default which can be overriden when necessary.

The full power of Selenium For complex operations which require the full power of the Selenium, Elemental gives you easy access to 100% of Selenium's API.

Terse, clean API Write less code than you would if you were using Selenium directly. The Elemental API is terse and internally consistent while still being explicit and unambiguous.

CHAPTER

ONE

THE POWER OF ELEMENTAL

```
import elemental

# Set up.
browser = elemental.Browser()

# Search PyPI for Elemental.
browser.visit("https://www.pypi.org")
browser.get_input(placeholder="Search projects").fill("elemental")
browser.get_button(type="submit").click()

# Click the first search result.
browser.get_element(partial_text="elemental").click()

# Confirm that Elemental's PyPI page was found.
assert browser.title == "elemental · PyPI"

# Tear down.
browser.quit()
```

Now dig in

Check out *Elemental's functionality* or *install it now* and make working with Selenium faster and easier.

1.1 Functionality

Elemental's functionality is grouped into 4 categories.

Actions Functions which perform an action.

For example: clicking a button or filling in an input field.

Getters Functions which get an element or elements.

For example: getting a button or div.

States Functions which get an element's state.

For example: whether an element is displayed or selected.

Values Functions which get a value.

For example: the HTML of a page or the text contained in an element.

When you need more

For anything which Elemental can't do and Selenium can, it's easy to switch to *raw Selenium*.

1.2 First steps with Elemental

1.2.1 Step 1. Install

Elemental can be installed with `pip`:

```
$ pip install elemental
```

1.2.2 Step 2. Start the browser

Getting a browser to drive your automation is this easy:

```
>>> import elemental
>>> browser = elemental.Browser()
```

This will give you the default Firefox browser.

For information about configuring the default browser and using a custom browser, see the *Browser* reference.

Geckodriver required

Note that Mozilla's geckodriver must be installed and available on PATH to use the default Selenium webdriver.

1.2.3 Step 3. Use the browser

Now do whatever you like with it. How about a web search?

```
>>> browser.visit("https://duckduckgo.com")
>>> placeholder = "Search the web without being tracked"
>>> browser.get_input(placeholder=placeholder).fill("python news")
>>> browser.get_input(id="search_button_homepage").click()
```

Enjoy your search results!

1.3 Behave + Elemental

1.3.1 Step 1. Create the browser fixture

Add the `browser` fixture to Behave's `environment.py` file.

```
# features/environment.py
from behave import (
    fixture,
    use_fixture,
)
import elemental

@fixture
def browser(context):
    # Create and yield the browser.
    context.browser = elemental.Browser(headless=True)
    yield context.browser

     # Stop the browser after the tests have finished.
    context.browser.quit()

def before_all(context):
    use_fixture(browser, context)
```

1.3.2 Step 2. Use the fixture

Then use it in your step implementation files.

```
# features/steps/navigation.py
from behave import (
    when,
    then,
 )

@when("I browse to PyPI")
def step_impl(context):
    context.browser.visit("https://pypi.org")

@then("I see PyPI's title")
def step_impl(context):
    assert context.browser.title == "PyPI · The Python Package Index"
```

1.4 Pytest + Elemental

1.4.1 Step 1. Create the browser fixture

Add the `browser` session-scoped fixture to Pytest's `conftest.py` file.

```
# tests/conftest.py
import elemental
import pytest

@pytest.fixture(scope="session")
def browser():
    # Create and yield the browser.
    _browser = elemental.Browser(headless=True)
     yield _browser
```

(continues on next page)

(continued from previous page)

```
    # Stop the browser after the tests have finished.
    _browser.quit()
```

1.4.2 Step 2. Use the fixture

Then use it in your test files.

```
# tests/test_navigation.py
def test_pypi(browser):
    browser.visit("https://pypi.org")
    assert browser.title == "PyPI · The Python Package Index"
```

1.5 Actions

Action functions cause the driver to complete an action, either on an element or the browser.

These functions operate on an element or browser, and don't return.

1.5.1 check

check()

Checks a checkbox. Always leaves the checkbox in a 'selected' state, no matter whether it was checked or unchecked when the action was performed.

Returns `None`

Example

```
>>> browser.get_element(id="my_checkbox").check()
```

1.5.2 click

click()

Clicks the element.

Returns `None`

Examples

```
>>> browser.get_button(id="my-button").click()
>>> browser.get_element(link_text="Python Package Index (PyPI)").click()
```

1.5.3 fill

fill(*string*)

Fills an input field with a string of text or filepath. Clears any existing input from the field prior to filling.

Parameters

string `str` The text or filepath as a string.

Returns `None`

Examples

```
>>> browser.get_element(id="my_input_field").fill("my_string")
>>> browser.get_input(label="Attach file").fill("my_filepath")
```

1.5.4 quit

quit()

Quits the browser. Operates on a browser, not an element.

Returns `None`

Example

```
>>> browser.quit()
```

1.5.5 screenshot

screenshot(*filepath*)

Takes a screenshot of either the full browser or an element, and saves it to a given filepath.

Parameters

filepath `str` The full filepath, as a string.

Returns `None`

Examples

```
>>> browser.get_element(id="screenshot").screenshot("element.png")
>>> browser.screenshot("page.png")
```

1.5.6 select

select()

Selects a radio button, select option, or checkbox element. Always leaves the target element in a selected state, no matter whether it was already selected.

Returns `None`

Examples

```
>>> browser.get_element(id="my_radio_button").select()
>>> browser.get_element(id="selector_id").get_element(text="option_text").select()
>>> browser.get_element(id="selector_id").get_element(value="option_value").
↪select()
```

1.5.7 uncheck

uncheck()

Unchecks a checkbox. Always leaves the checkbox in a 'not selected' state, no matter whether it was checked or unchecked when the action was performed.

Returns `None`

Example

```
>>> browser.get_element(id="my_checkbox").uncheck()
```

1.5.8 visit

visit(*url*)

Visits a url in the browser.

Parameters

url `str` The URL as a string.

Returns `None`

Example

```
>>> browser.visit("https://redandblack.io")
```

1.6 Getters

Getters get an element.

They operate on an element or browser and return an element or list of elements.

Getters can be chained (see *Chaining getters*).

1.6.1 get_button

get_button(*occurrence=1*, *wait=5*, ***kwargs*)

Gets a button by class, id, partial text, text or type.

Parameters

occurrence `int, optional` The occurrence to get. For instance, if multiple buttons on the page meet the requirements and the 3rd one is required, set this to 3.

wait `int, optional` The time to wait, in seconds, for the button to be found.

kwargs One and only one keyword argument must be supplied. Allowed keys are: "class_name", "id", "partial_text", "text", "type".

Returns `element`

Examples

```
>>> browser.get_button(class_name="primary")
>>> browser.get_button(id="my-button")
>>> browser.get_button(partial_text="Save document")
>>> browser.get_button(text="Save document now", wait=30)
>>> browser.get_button(type="submit")
```

1.6.2 get_element

get_element(*occurrence=1*, *wait=5*, ***kwargs*)

Gets any element.

Parameters

occurrence `int, optional` The occurrence to get. For instance, if multiple links on the page meet the requirements and the 3rd one is required, set this to 3.

wait `int, optional` The time to wait, in seconds, for the element to be found.

kwargs One and only one keyword argument must be supplied. Allowed keys are: "class_name", "css", "id", "link_text", "name", "partial_link_text", "partial_text", "tag", "text", "type", "value", "xpath".

Returns `element`

Examples

```
>>> browser.get_element(class_name="link")
>>> browser.get_element(css="div.container p", wait=10)
>>> browser.get_element(id="heading")
>>> browser.get_element(link_text="Python Package Index (PyPI)")
>>> browser.get_element(name="para-1")
>>> browser.get_element(partial_link_text="PyPI")
>>> browser.get_element(partial_text="Paragraph", occurrence=2)
>>> browser.get_element(tag="a")
>>> browser.get_element(text="Some text")
>>> browser.get_element(type="button")
>>> browser.get_element(value="Sometext")
>>> browser.get_element(xpath="//*[@id='para-2']")
>>> browser.get_element(tag="div").get_element(id="primary")
```

1.6.3 get_elements

get_elements(*min_elements=1*, *wait=5*, ***kwargs*)

Gets a list of elements.

Parameters

min_elements `int, optional` The minimum number of elements which must be found to return before the wait time expires.

wait `int, optional` The time to wait, in seconds, for the elements to be found.

kwargs One and only one keyword argument must be supplied. Allowed keys are: "class_name", "css", "id", "link_text", "name", "partial_link_text", "partial_text", "tag", "text", "type", "value", "xpath".

Returns `list of elements`

Examples

```
>>> browser.get_elements(class_name="link")
>>> browser.get_elements(css="div.container a")
>>> browser.get_elements(id="heading")
>>> browser.get_elements(link_text="Python.org")
>>> browser.get_elements(name="para-1")
>>> browser.get_elements(partial_link_text="Python")
>>> browser.get_elements(partial_text="Paragraph")
>>> browser.get_elements(tag="p", min_elements=5, wait=0)
>>> browser.get_elements(text="Python.org")
>>> browser.get_elements(type="button")
>>> browser.get_elements(value="Sometext")
>>> browser.get_elements(xpath="//*[@class='para']")
>>> browser.get_element(class_name="container").get_elements(tag="p")
```

1.6.4 get_input

get_input(*occurrence=1*, *wait=5*, ***kwargs*)

Gets an input field by class, id, label, or placeholder text.

Parameters

occurrence `int, optional` The occurrence to get. For instance, if multiple inputs on the page meet the requirements and the 3rd one is required, set this to 3.

wait `int, optional` The time to wait, in seconds, for the input field to be found.

kwargs: One and only one keyword argument must be supplied. Allowed keys are: "class_name", "id", "label", "placeholder".

Returns `element`

Examples

```
>>> browser.get_input(class_name="input", wait=0, occurrence=2)
>>> browser.get_input(id="full-name")
>>> browser.get_input(label="Full name")
>>> browser.get_input(placeholder="Enter your full name")
```

1.6.5 get_parent

get_parent()

Gets an element's parent element.

Returns `element`

Examples

```
>>> browser.get_button(type="submit").get_parent()
>>> browser.get_elements(text="Python.org").get_parent()
>>> browser.get_input(label="First name").get_parent()
```

1.6.6 Chaining getters

You can chain getters together to zero in on the element or list of elements you want. Note you cannot chain another getter after *get_elements*.

Examples

```
>>> browser.get_element(id="my-form").get_button(type="submit")
>>> browser.get_element(tag="div").get_element(id="primary")
>>> browser.get_element(class_name="container").get_elements(tag="p")
>>> browser.get_element(tag="form").get_input(label="Full name")
>>> browser.get_element(tag="select").get_element(tag="option", occurrence=3)
```

1.7 States

States functions are used to determine whether an element has a state, such as whether it is displayed or selected.

These functions operate on an element, and return a boolean.

1.7.1 is_displayed

`is_displayed`

Determines whether an element is displayed to the user.

Returns `bool` True if displayed, False otherwise.

Examples

```
>>> browser.get_element(id="not-hiding").is_displayed
True
>>> browser.get_element(id="hiding").is_displayed
False
```

Note: `is_displayed` does not work on hidden select options.

1.7.2 is_enabled

`is_enabled`

Determines whether an element is enabled or disabled.

Returns `bool` True if enabled, False otherwise.

Examples

```
>>> browser.get_element(id="available").is_enabled
True
>>> browser.get_element(id="disabled").is_enabled
False
```

1.7.3 is_selected

is_selected

Determines whether a radio button, checkbox or select option is selected.

Returns `bool` True if selected, False otherwise.

Examples

```
>>> browser.get_element(id="checked").is_selected
True
>>> browser.get_element(id="unchecked").is_selected
False
```

1.8 Values

Values functions get the value of an element, or one of it's attributes.

These functions operate on an element or browser, and return a string or boolean.

1.8.1 attribute

attribute(*attribute_name*)

Gets the value of an attribute.

Parameters

attribute_name `str` The name of the attribute. For instance, "hidden", or "class".

Returns `str` or `bool` For boolean attributes like "hidden" a boolean is returned. For other attributes a string is returned.

Examples

```
>>> browser.get_element(id="hiding").attribute("hidden")
True
>>> browser.get_element(id="my_id").attribute("id")
"my_id"
```

1.8.2 html

`html`

Gets the html for an element or whole page.

Returns `str` The html for the element or page, as a string.

Example

```
>>> browser.get_element(tag="p").html
"<p>This is a paragraph.</p>"
```

1.8.3 text

`text`

Gets the text of an element.

Returns `str` The text of the element, as a string.

Example

```
>>> browser.get_element(tag="p").text
"This is some text"
```

1.8.4 title

`title`

Gets the title of a page.

Returns `str` The text of the page title, as a string.

Example

```
>>> browser.title
"A Page Title"
```

1.8.5 url

`url`

Gets the url of a page.

Returns `str` The url of the current page, as a string.

Example

```
>>> browser.url
"https://redandblack.io"
```

1.9 Browser

The `Browser` class provides the browser which drives Elemental.

1.9.1 The Browser class

Browser(*selenium_webdriver=None*, *headless=False*)

Parameters

selenium_webdriver `Selenium webdriver, optional` The webdriver being used to drive the browser. Default is a Firefox webdriver.

headless `bool, optional` Whether the default webdriver will run in headless mode. Has no effect if the default webdriver is not used. Default is False.

Example: Default browser

```
import elemental

browser = elemental.Browser()
browser.visit("https://pypi.org/project/elemental/")
```

Example: Headless default browser

```
import elemental

browser = elemental.Browser(headless=True)
browser.visit("https://pypi.org/project/elemental/")
```

1.9.2 Modifying the default browser

The `Browser` class has a `selenium_webelement` property which is the underlying Selenium webdriver.

Use that to modify the default browser when necessary.

For example, to set window size of the default webdriver:

```
import elemental

browser = elemental.Browser(headless=True)
browser.selenium_webdriver.set_window_size(1920, 1080)

browser.visit("https://pypi.org/project/elemental/")
```

1.9.3 Bring-your-own browser

To use your own browser rather than the default browser, instantiate `Browser` with a Selenium WebDriver instance as the only argument.

For example:

```
import elemental
import selenium
from selenium.webdriver.firefox import options as firefox_options

options = firefox_options.Options()
options.headless = True
selenium_webdriver = selenium.webdriver.Firefox(
    executable_path="geckodriver",
    options=options,
)

browser = elemental.Browser(selenium_webdriver)
```

1.10 Raw Selenium

You can use Elemental and still access the full power of Selenium any time you need it.

1.10.1 Selenium webdriver

Every instance of Elemental's `Browser` class has a `selenium_webdriver` property which provides an instance of Selenium's WebDriver.

It can be used like this:

```
import elemental

browser = elemental.Browser()
browser.visit("https://pypi.org")

# Elemental doesn't provide a way to get cookies, so use the Selenium
# webdriver's get_cookies() method.
cookies = browser.selenium_webdriver.get_cookies()
print(cookies)
```

1.10.2 Selenium webelement

Every instance of Elemental's `Element` class has a `selenium_webdriver` property which provides an instance of Selenium's WebDriver.

Every instance of Elemental's `Element` class also has a `selenium_webelement` property which provides an instance of the Selenium WebElement which corresponds to its HTML element.

It can be used like this:

```
import elemental

browser = elemental.Browser()
browser.visit("https://pypi.org")
logo = browser.get_element(class_name="site-header__logo")

# Elemental doesn't provide a way to get the element's size, so use the
# Selenium webelement's size property.
```

(continues on next page)

(continued from previous page)

```
logo_size = logo.selenium_webelement.size
print(logo_size)
```

1.11 Contributions are welcome

1.11.1 Adding functionality is easy

The Elemental source code is simple, clean and well-organised.

Take a look at it and you'll see what we mean!

The four main modules are `actions.py`, `getters.py`, `states.py` and `values.py`. You can guess what sort of functionality each one contains.

As a result it's easy to work with and add to.

1.11.2 Pull requests are encouraged

If Elemental lacks some functionality which you need, fork the repo and add it in.

Then send us a pull request.

We're friendly and happy to work with you – no matter what your level of experience – to get your PR merged into Elemental and released promptly.

1.12 Source

The Elemental source is available under the MIT licence.

You can find it in the Elemental repository on Github.

INDEX