Selenium Data Attributes Documentation

Release 1.0.2

John Lane

May 14, 2018

Contents

Contents:

CHAPTER 1

1. Installation

1.1 1.1 Introduction

sda (Selenium Data Attributes) is a simple library built using the selenium-python bindings to approach building testing frameworks in a more intuitive way.

sda (Selenium Data Attributes) only supported python version in 2.7 and is available on PyPi and on GitHub

1.2 1.2 PyPi

To install via PyPi make sure you first install Pip. Afterwards run the following command in your terminal:

```
sudo pip install sda
```

If you are having trouble installing the package use the following command:

```
sudo pip install --no-dependencies sda
```

1.3 1.3 GitHub

To install via Github you have two options for installers: 1. PyPi 2. Setuptools

To install sda using PyPi from Github you would run the following command:

```
sudo pip install git+git://github.com/jlane9/selenium_data_attributes
```

You can add an additional @*branch-name* at the end to install from a specific branch

To install sda using Setuptools:

1. Make sure you have git cli (command line interface) installed on your machine

2. cd to the directory that you want the source to be installed and execute the following command:

   ```
   git clone https://github.com/jlane9/selenium_data_attributes
   ```

3. Move into that directory and install via setuptools:

   ```
   cd selenium_data_attributes
   sudo python setup.py install
   ```

1.4 1.4 Dependencies

In case you are unable to install selenium from the dependencies, install using easy_install:

```
sudo python easy_install selenium
```

1.5 1.5 Updating

To update SDA just run:

```
sudo pip install -U sda
```

To be on the bleeding edge (not recommended) use:

```
sudo pip install git+git://github.com/jlane9/selenium_data_attributes.git@master
↪#egg=sda
```

CHAPTER 2

2. Getting Started

2.1 2.1 Sample project

After installing sda you should be ready to begin.

SDA is built with the intention that it will be used in conjunction with web development. A developer would develop their web site using uniquely identifiable ids or attributes to locate elements within a web page. SDA allows the test builders to create a "framework" that all tests can generally be written on top of so that the tests are not brittle (simple changes easily break operability and fixing requires extensive re-work. When beginning a testing project, it is best practice to already start thinking of how that framework structure will come together. An example would be:

my_site

- **hello_page**
    - __init__.py
    - fixtures.py
    - locators.py
    - page.py
- **goodbye_page**
    - __init__.py
    - fixtures.py
    - locators.py
    - page.py
- **website**
    - __init__.py
    - site.py

Each “page” would have its own locators for elements and fixtures which are just elements or collections of elements with defined structures and have specific behaviours.

2.2 2.2 Using SDA to define pages

Within each page you need to define each element that may appear on that page. And for each element you need to define how one might find that element and only that element. For example on hello page there might be a form that the user would fill out.

```
<form id="form_hello">
   <input id="hello_name" placeholder="What is your name?" />
   <input id="hello_submit" type="submit" />
</form>
```

To define that form, or “fixture”, we would do something similar to the following:

```
# First we would start out in the locators.py file
from sda.locators import Locators
from selenium.webdrivers.common.by import By

class HelloLocators(Locators):

   FORM_HELLO = (By.XPATH, '//form[@id="form_hello"]')
   FORM_NAME = (By.XPATH, '//input[@id="hello_name"]')
   FORM_SUBMIT = (By.XPATH, '//input[@id="hello_submit"]')
```

```
# Then we would move to the fixtures.py file
from sda.element import Element
from sda.structures import *
from locators import HelloLocators

class HelloForm(Element):

   def __init__(self, web_driver, by, path):

      super(HelloForm, self).__init__(web_driver, by, path)

      hello = InputText(web_driver, *HelloLocators.FORM_NAME)
      submit = Button(web_driver, *HelloLocators.FORM_SUBMIT)
```

```
# Lastly we would add that fixture to page.py
from sda.page import Page
from fixtures import HelloForm
from locators import HelloLocators

class HelloPage(Page):

   def __init__(self, web_driver):

      super(HelloPage, self).__init__(web_driver)

      form = HelloForm(web_driver, *HelloLocators.FORM_HELLO)
```

```
# Once the page is complete, add it to the main site
from sda.site import Site
from my_site.hello_page.page import HelloPage

class MySite(Site):

   def __init__(self, web_driver):

      super(MySite, self).__init__(web_driver)

      hello = HelloPage(web_driver)
```

CHAPTER 3

SDA Reference

Contents:

3.1 Element - Base class

Element is the base class for which every other web element type is built from. It is not recommended to use this class directly.

sda.element

class `sda.element.`**`Element`**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: `object`

The Element implementation

An abstract class for interacting with web elements. Example use below:

Example file structure:

my_project

- __init__.py
- main.py
- **my_web_page**
  - __init__.py
  - fixtures.py
  - locators.py
  - page.py

The following example demonstrates a user creating a custom fixture (SomeElement) for an element on their web page, using a locator class to store the selenium selector and implement a web page view to interact with that web page and its elements:

fixtures.py

```
from selenium_data_attributes.element import Element
from selenium_data_attributes.mixins import ClickMixin

class SomeElement(Element, ClickMixin):

    pass
```

locators.py

```
from selenium_data_attributes.locators import Locators
from selenium.webdriver.common.by import By

class MyWebLocators(Locators):

    EXAMPLE_BUTTON = (By.XPATH, '//some//path[@id="id_example"])
```

page.py

```
from selenium_data_attributes.page import Page

from my_project.my_web_page.fixtures import SomeElement
from my_project.my_web_page.locators import MyWebLocators

class MyWebPage(Page):

    def __init__(self, web_driver):

        self.driver = web_driver

        self.example_button = SomeElement(driver, *MyWebLocators.EXAMPLE_BUTTON)
```

main.py

```
from my_project.my_web_page.page import MyWebPage
from selenium import webdriver

# Instantiate web driver
wd = webdriver.Firefox()

web_page = MyWebPage(wd)

web_page.example_button.click()
```

blur()

Simulate moving the cursor out of focus of this element.

Returns

css_property(*prop*)

Return the value of a CSS property for the element

Parameters **prop** (*str*) – CSS Property

Returns Value of a CSS property

Return type str

drag(*x_offset=0*, *y_offset=0*)

Drag element x,y pixels from its center

Parameters

- **`x_offset`** (*`int`*) – Pixels to move element to
- **`y_offset`** (*`int`*) – Pixels to move element to

Returns

`element``()`

Return the selenium web element object

Returns Selenium WebElement

Return type WebElement

`exists``()`

Returns True if element can be located by selenium

Returns Returns True, if the element can be located

Return type bool

`focus``()`

Simulate element being in focus

Returns

`html``()`

Returns HTML representation of the element

Returns HTML representation of the element

Return type str

`is_displayed``()`

Return True, if the element is visible

Returns True, if element is visible

Return type bool

`parent``()`

Returns the Selenium element for the current element

Returns

`scroll_to``()`

Scroll to the location of the element

Returns

`tag_name`

Returns element tag name

Returns Element tag name

Return type str

`wait_until_appears``(`*timeout=30*`)`

Wait until the element appears

Parameters **`timeout`** (*`int`*) – Wait timeout in seconds

Returns True, if the wait does not timeout

Return type bool

wait_until_disappears(*timeout=30*)
Wait until the element disappears

Parameters **timeout** (*int*) – Wait timeout in seconds

Returns True, if the wait does not timeout

Return type bool

wait_until_present(*timeout=30*)
Wait until the element is present

Parameters **timeout** – Wait timeout in seconds

Returns True, if the wait does not timeout

Return type bool

sda.element.**normalize**(*_by*, *path*, **args*, ***kwargs*)
Convert all paths into a xpath selector

Parameters

- **_by** (*str*) – Selenium selector
- **path** (*str*) – Selector value
- **args** –
- **kwargs** –

Returns

sda.element.**join**(**args*)
Join 'x' locator paths into a single path

Parameters **args** – Locator path tuples (by, path)

Returns Locator path

Return type str

3.2 Locators - Selector Class

Locators is the base class for which location selectors are implemented for a project.

sda.locators

class sda.locators.**Locators**
Bases: object

The Locators implementation

as_dict()
Return all locators

Example:

```
from selenium_data_attributes.locators import Locators

# Let's assume the user uses the Locators class to define some locators
# for the elements on their web page.

class SomeLocators(Locators):
```

(continues on next page)

(continued from previous page)

```
    USER_NAME = (By.ID, 'username')
    PASSWORD = (By.ID, 'password')

# If that user wanted to return all locators associated with this class
# i.e. "USER_NAME" and "PASSWORD" and return the values of both
# they'd use 'as_dict'

l = SomeLocators()

l.as_dict()

# Returns
# {'USER_NAME': (By.ID, 'username'), 'PASSWORD': (By.ID, 'password')}
```

Returns

Return type dict

is_locator(*attrib=None*)

Returns True if the class attribute is a valid locator

Parameters **attrib** – Class attribute

Returns True, if the class attribute is a valid locator

Return type bool

static is_valid(*_by=", path=None*)

Returns true if the selenium selector is valid

Parameters

- **_by** (*str*) – Selenium By locator
- **path** (*str*) – Locator value

Returns True, if the selenium selector is valid

Return type bool

3.3 Mixins - Element functionality extensions

Mixins allow for elements to "share" common functions with other elements. Elements inherit from the Element base class and can be "extended" by any number of mixins. An example would be:

```
from sda.element import Element
from sda.mixins import ElementMixin

class FooMixin(ElementMixin):

    def foo(self):
        return 1

class BarMixin(ElementMixin):

    def bar(self):
```

(continues on next page)

(continued from previous page)

```
        return 0

class Foo(Element, FooMixin, BarMixin):

    pass

f = Foo()

f.bar()

# Returns
0

f.foo()

# Returns
1
```

sda.mixins

class sda.mixins.**ClickMixin**

Bases: sda.mixins.ElementMixin

The ClickMixin Implementation

click()

Click element

Returns

double_click()

Double-click element

Returns

hover()

Simulate hovering over element

Returns

class sda.mixins.**InputMixin**

Bases: sda.mixins.ElementMixin

The InputMixin implementation

input(**args*, ***kwargs*)

Parameters

- **args** – Text to send to the input field
- **kwargs** – clear - True if user wants to clear the field before assigning text

Returns True, if text is assigned

Return type bool

value

Return value of input

Returns Input value

Return type str

class `sda.mixins.`**`SelectMixin`**
Bases: `sda.mixins.ElementMixin`

The SelectMixin implementation

`deselect_all``()`
Deselect all selected options

Returns True, if all options are deselected

Return type bool

`deselect_by_index``(`*option*`)`
Deselect option by index [i]

Parameters **`option`** – Select option index

Returns True, if option is deselected

Return type bool

`deselect_by_text``(`*option*`)`
Deselect option by display text

Parameters **`option`** – Select option

Returns True, if option is deselected

Return type bool

`deselect_by_value``(`*option*`)`
Deselect option by option value

Parameters **`option`** – Select option value

Returns True, if option is deselected

Return type bool

`options``()`
Returns all Select options

Returns List of options

Return type list

`select_by_index``(`*option*`)`
Select option at index [i]

Parameters **`option`** (*`str`*) – Select index

Returns True, if the option is selected

Return type bool

`select_by_text``(`*option*`)`
Select option by display text

Parameters **`option`** (*`str`*) – Select option

Returns True, if the option is selected

Return type bool

`select_by_value``(`*option*`)`
Select option by option value

Parameters **`option`** (*`str`*) – Select option value

Returns True, if the option is selected

Return type bool

`selected_first``()`
Select first option

Returns First option element

Return type WebElement

`selected_options``()`
Returns a list of selected options

Returns List of options

Return type list

`class` `sda.mixins.`**`SelectiveMixin`**
Bases: `sda.mixins.ClickMixin`

The SelectiveMixin implementation

`deselect``()`
Deselect this element

Returns

`select``()`
Select this element

Returns

`selected``()`
Return True if element is selected

Returns True, if the element is selected

Return type bool

`class` `sda.mixins.`**`TextMixin`**
Bases: `sda.mixins.ElementMixin`

The TextMixin implementation

`text``()`
Returns the text within an element

Returns Element text

Return type str

`visible_text``()`
Returns the visible text within an element

Returns Element text

Return type str

3.4 Page - Web page class

Pages are considered the scaffolding for interacting with web pages as a whole. While the class is not necessary in creating testing frameworks, it does contain a few useful functions such as validating that the browser is in view of that page. An example would look like:

```
from sda.page import Page
from sda.structures import *
from selenium import webdriver

class HelloWorld(Page):

    def __init__(self, driver):

        Page.__init__(self, driver, '/category/sub-category/page')  # Make sure that␣
↪this is the path only

        self.foo = Button(driver, '//button[@id="buttonFoo"]')

wd = webdriver.Firefox()
h = HelloWorld(wd)

# Click 'Foo' button
h.foo.click()
```

sda.page

class `sda.page.`**`Page`**(*web_driver*, *url_path=u'/'*)

Bases: `sda.element.SeleniumObject`

The Page Implementation

`elements`()

Returns all testable elements on a page

Returns Dictionary of WebElements

Return type dict

`in_view`()

Returns True if the driver is currently within the scope of this page

Returns True, if driver on page

Return type bool

static **`is_element`**(*attrib=None*)

Returns True if the class attribute is a valid locator

Parameters **`attrib`** – Class attribute

Returns True, if the class attribute is a valid locator

Return type bool

`navigate_to`(**args*)

Navigate to path

Returns

`title`

Return page title

Returns Page title

Return type str

`url`

Current page URL

Returns Page URL

Return type str

3.5 Core functions

Core functions are reusable shortcuts that all elements can use.

`sda.shortcuts.`**`generate_elements`**`(`*_class*, *locator*`)`

Iterate through all elements returned and create an instance of _class for each

Parameters

- **_class** (`Element`) – Class to create instances from
- **locator** – SDA Locator. ex. ('xpath', '//element/path/here')

Returns

```
from sda.core import generate_elements
from sda.element import Element
from selenium.webdriver.common.by import By
from selenium import webdriver

# Locator
class FooLocators(object):

    BAR_LOCATOR = (By.XPATH, '//some/locator')

# Can be fixture or structure
class Bar(object):

    def __init__(self, web_driver, by, path):

        self._driver = web_driver
        self.element = Element(web_driver=web_driver, by=by, path=path)

# Can be fixture or page
class Foo(object):

    def __init__(self, web_driver):
        self.driver = web_driver

    # Essentially what generate elements will do is find all elements that return␣
↪from the selector and then append
    # an index at the end of the selector expression. Make sure to use XPATH. There␣
↪will be support for other
    # selector types other than By.XPATH, but this is the only way that it will␣
↪properly work. Always remember to
    # return the web driver element!
    @generate_elements(Bar, FooLocators.BAR_LOCATOR)
    def bars(self):

        return self.driver

wd = webdriver.Firefox()
f = Foo(wd)
```

(continues on next page)

(continued from previous page)

```
# Returns all the foobar instances it can find
bars = f.bars()
```

3.6 Site - Web site project class

Similar to Page, Site contains useful functions on the site-level. An example would look like:

```
from sda.page import Page
from sda.site import Site
from sda.structures import *
from selenium import webdriver

class MyPage(Page):

    def __init__(self, driver):

    Page.__init__(self, driver, '/category/sub-category/page')  # Make sure that this␣
↪is the path only

    self.bar = Button(driver, '//button[@id="buttonBar"]')


class MySite(Site):

    def __init__(self, driver):

    Site.__init__(self, driver)

    self.foo = MyPage(driver)

wd = webdriver.Firefox()
site = MySite(wd)

# Click 'Bar' button on page 'Foo'
site.foo.bar.click()
```

sda.site

class `sda.site.`**`Site`**(*web_driver*, ***kwargs*)

Bases: `sda.element.SeleniumObject`

The Site Implementation

The intention for the Site object is to contain all website pages. An example usage of this might be:

Let's say we have the following file structure

my_project

- __init__.py
- main.py
- **page_1**
  - __init__.py
  - fixtures.py

  - locators.py
  - page.py
- **page_2**
  - __init__.py
  - fixtures.py
  - locators.py
  - page.py
- **site**
  - __init__.py
  - site.py
  - settings.py

site/site.py

```
from sda.site import Site
from page_1.page import Page1
from page_2.page import Page2

class ExampleSite(Site):

    def __init__(self, web_driver):

        super(ExampleSite, self).__init__(web_driver)
        self.page_1 = Page1(web_driver)
        self.page_2 = Page2(web_driver)
```

`domain`
Returns the domain for a website

Returns domain

Return type str

`path`
Returns the website path

Returns path

Return type str

`url`
Current page URL

Returns Page URL

Return type str

3.7 Structures - Web element templates

Structures are classes that represent the functionality of various web 'structures'. For instance, simple web structures might include buttons or text input fields while more complex structures would be forms, tables, dropdown menus. Structures should be generalized and should not rely on 'plugins' like Bootstrap or other custom libraries.

sda.structures

class sda.structures.**Button**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.element.Element*, sda.mixins.ClickMixin, sda.mixins.TextMixin

The Button implementation

Example Use:

Let's take the following example:

```
<button id="someClassId" class="someClass" on-click="javascript.function" >Click␣
↪Me</button>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<button data-qa-id="some.identifier" id="someClassId" class="someClass" on-click=
↪"javascript.function">
    Click Me
</button>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//button[@data-qa-id="some.identifier"]")
b = structures.Button(driver, *locator)

# Example usage
b.click()
```

class sda.structures.**Div**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.element.Element*

The Div implementation

Example Use:

Let's take the following example:

```
<div id="someClassId" class="someClass">
    ...
</div>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<div data-qa-id="some.identifier" id="someClassId" class="someClass">
    ...
</div>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//button[@data-qa-id="some.identifier"]")
d = structures.Button(driver, *locator)
```

class sda.structures.**Dropdown**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.element.Element*, sda.mixins.ClickMixin, sda.mixins.TextMixin

The Dropdown implementation

Note: This structure is specifically for a Bootstrap dropdown

Example Use:

```
    <div class="dropdown">
        <button class="btn btn-primary dropdown-toggle" type="button" data-toggle=
↪"dropdown">Dropdown Example
        <span class="caret"></span></button>
        <ul class="dropdown-menu">
            <li><a href="#">HTML</a></li>
            ...
        </ul>
    </div>


If the user wants to make the code above recognizable to the testing framework,␣
↪they would add the attribute
"data-qa-id" with a unique value as well as "data-qa-model" with a type.

.. code-block:: html

    <div class="dropdown" data-qa-id="some.identifier" data-qa-model="dropdown">
        <button class="btn btn-primary dropdown-toggle" type="button" data-toggle=
↪"dropdown">Dropdown Example
        <span class="caret"></span></button>
        <ul class="dropdown-menu">
            <li><a href="#">HTML</a></li>
            ...
        </ul>
    </div>


An example on how to interact with the element:

.. code-block:: python

    import selenium
    from selenium.webdriver.common.by import By
    from selenium_data_attributes import structures

    driver = webdriver.FireFox()
```

(continues on next page)

(continued from previous page)

```
    driver.get('http://www.some-url.com')

    locator = (By.XPATH, "//input[@data-qa-id="some.identifier"]")
    d = structures.Dropdown(driver, *locator)

    # Example usage
    d.expand()
```

collapse(*hover=False*)
Hide dropdown

Returns

Return type bool

container
Dropdown container

Returns

expand(*hover=False*)
Show dropdown

Returns

Return type bool

toggle
Show/hide toggle button

Returns

class sda.structures.**Form**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)
Bases: *sda.element.Element*

The Form implementation

Example Use:

Let's take the following example:

```
<form id="someForm">
    <input id="someClassId" type="checkbox" class="someClass">
    ...
</form>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<form id="someForm" data->
    <input id="someClassId" type="checkbox" class="someClass">
    ...
</form>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
```

(continues on next page)

(continued from previous page)

```
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//input[@data-qa-id="some-identifier"]")
form = structures.Form(driver, *locator)

# Example usage
field = form.get_field('someClassId')
```

get_field(*field_name*)

Returns field with id *field_name*

Parameters **field_name** (*basestring*) – Form field to get

Returns

class sda.structures.**Image**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.element.Element*

The Image implementation

Example Use:

Let's take the following example:

```
<img id="someClassId" class="someClass" />
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<img data-qa-id="some.identifier" id="someClassId" class="someClass" />
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//img[@data-qa-id="some.identifier"]")
i = structures.Image(driver, *locator)

# Returns tag attribute 'src'
i.source()
```

source()

Returns image source URL

Returns Image source URL

Return type str

class sda.structures.**InputCheckbox**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: sda.structures.Field, sda.mixins.SelectiveMixin

The InputCheckbox implementation

Example Use:

Let's take the following example:

```
<input id="someClassId" type="checkbox" class="someClass">
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<input data-qa-id="some.identifier" id="someClassId" type="checkbox" class=
↪"someClass">
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//input[@data-qa-id="some.identifier"]")
c = structures.InputCheckbox(driver, *locator)

# Example usage
c.select()
```

class sda.structures.**InputRadio**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.structures.InputCheckbox*, sda.mixins.SelectiveMixin

The InputRadio implementation

Example Use:

Let's take the following example:

```
<input id="someClassId" type="radio" class="someClass">
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<input data-qa-id="some.identifier" id="someClassId" type="radio" class="someClass
↪">
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

r = structures.InputRadio(driver, "//input[@data-qa-id="some.identifier"]")

# Input Radio inherits from InputCheckbox
r.select()
```

class sda.structures.**InputText**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: sda.structures.Field, sda.mixins.InputMixin, sda.mixins.ClickMixin

The InputText implementation

Example Use:

Let's take the following example:

```
<input id="someClassId" type="text" class="someClass">
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<input data-qa-id="some.identifier" id="someClassId" type="text" class="someClass
↪">
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//input[@data-qa-id="some.identifier"]")
t = structures.InputText(driver, *locator)

# Example usage
t.input('Hello World')
```

class sda.structures.**Link**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.structures.Button*, sda.mixins.ClickMixin, sda.mixins.TextMixin

The Link implementation

Example Use:

Let's take the following example:

```
<a id="someClassId" class="someClass" href="/some/link/path">Click Me</a>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<a data-qa-id="some.identifier" id="someClassId" class="someClass" href="/some/
↪link/path">Click Me</a>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//a[@data-qa-id="some.identifier"]")
l = structures.Link(driver, *locator)

# Inherits from Button
l.click()
```

class sda.structures.**MultiSelect**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)

Bases: *sda.element.Element*

The MultiSelect implementation

Example Use:

Let's take the following example:

```
<div id="someClassId" class="someClass" isteven-multi-select input-model="some.
↪model"
output-model="format.model" helper-elements="filter all none">
    ...
</div>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value as well as "data-qa-model" with a type.

```
<div data-qa-id="some.identifier" data-qa-model="multiselect" id="someClassId"␣
↪class="someClass"
isteven-multi-select input-model="some.model" output-model="format.model" helper-
↪elements="filter all none">
    ...
</div>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//a[@data-qa-id="some.identifier"]")
m = structures.MultiSelect(driver, *locator)

# Example usage
l.expand()
```

clear_search()

Click clear search button

Returns

Return type bool

collapse()

Hide iSteven dropdown

Returns

Return type bool

deselect_by_index(*index*)

Deselect option at index 'i'

Parameters **index** (*str*) – Index

Returns

Return type bool

`deselect_by_text`(*text*)
Deselect option that matches text criteria

Parameters **`text`** (*`str`*) – Text criteria

Returns

Return type bool

`expand`()
Show iSteven dropdown

Returns

Return type bool

`options`(*include_group=True*)
Return all available options

Parameters **`include_group`** (*`bool`*) – True, to include groupings

Returns List of options

Return type list

`reset`()
Reset selection to default state

Returns

Return type bool

`search`(*value*, *clear=True*)
Filter selections to those matching search criteria

Parameters

- **`value`** (*`str`*) – Search criteria
- **`clear`** (*`bool`*) – Clear previous search criteria

Returns

Return type bool

`select_all`()
Select all possible selections

Returns

Return type bool

`select_by_index`(*index*)
Select option at index 'i'

Parameters **`index`** (*`str`*) – Index

Returns

Return type bool

`select_by_text`(*text*)
Select option that matches text criteria

Parameters **`text`** (*`str`*) – Text criteria

Returns

Return type bool

select_none()
Deselect all selections

Returns

Return type bool

selected_options()
Return all selected options

Returns List of selected options

Return type list

class sda.structures.**Select**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)
Bases: *sda.element.Element*, sda.mixins.SelectMixin

The Select implementation

Example Use:

Let's take the following example:

```
<select id="someClassId" class="someClass">
    <option value="1">Value 1</option>
    <option value="2">Value 2</option>
    <option value="3">Value 3</option>
    <option value="4">Value 4</option>
</select>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<select data-qa-id="some.identifier" id="someClassId" class="someClass">
    <option value="1">Value 1</option>
    <option value="2">Value 2</option>
    <option value="3">Value 3</option>
    <option value="4">Value 4</option>
</select>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//input[@data-qa-id="some.identifier"]")
s = structures.Select(driver, *locator)

# Example usage. Returns ['Value 1', 'Value 2', 'Value 3', 'Value 4']
s.options()
```

class sda.structures.**Text**(*web_driver*, *by='xpath'*, *path=None*, ***kwargs*)
Bases: *sda.element.Element*, sda.mixins.TextMixin, sda.mixins.ClickMixin

The Text implementation

Example Use:

Let's take the following example:

```
<p id="someClassId" class="someClass">
    ...
</p>
```

If the user wants to make the code above recognizable to the testing framework, they would add the attribute "data-qa-id" with a unique value.

```
<p data-qa-id="some.identifier" id="someClassId" class="someClass">
    ...
</p>
```

An example on how to interact with the element:

```
import selenium
from selenium.webdriver.common.by import By
from selenium_data_attributes import structures

driver = webdriver.FireFox()
driver.get('http://www.some-url.com')

locator = (By.XPATH, "//p[@data-qa-id="some.identifier"]")
d = structures.Text(driver, *locator)

# Prints text inside text elements
print d
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index