**Locators: [**Identifying the different elements [like textbox, select options, links, images, etc.] on web page using various techniques**]**

Locating elements in WebDriver can be achieved by

* WebDriver instance itself
* WebElement.

Each of the language bindings exposes a “Find Element” and “Find Elements” method.

**Find Element** – returns an element or throws exception if no element is found.

**Find Elements** – returns a list of elements or empty list if no elements are found.

The “Find” methods take a locator or query object called “By”.

“By” strategies are listed below:

**Element selection strategies**

The different built-in element location strategies in WebDriver:

|  |  |
| --- | --- |
| **Locator** | **Description** |
| class name | Locates elements whose class name contains the search value (compound class names are not permitted) |
| css selector | Locates elements matching a CSS selector |
| id | Locates elements whose ID attribute matches the search value |
| name | Locates elements whose NAME attribute matches the search value |
| link text | Locates anchor elements whose visible text matches the search value |
| partial link text | Locates anchor elements whose visible text partially matches the search value |
| tag name | Locates elements whose tag name matches the search value |
| xpath | Locates elements matching an XPath expression |

1. **By ID** - efficient and preferred way to locate an element

HTML IDs are available, unique, and consistently predictable, they are the preferred method for locating an element on a page. They tend to work very quickly, and forego much processing that comes with complicated DOM traversals.

Example:

<div id="coolestWidgetEvah">...</div>

**Java:** WebElement element = driver.findElement(By.id("coolestWidgetEvah"));

**Python:** element = driver.find\_element(by=By.ID, value="coolestWidgetEvah")

1. **By Class Name**

“Class” in this case refers to the attribute on the DOM element. Often in practical use there are many DOM elements with the same class name, thus finding multiple elements becomes the more practical option over finding the first element.

Example:

<div class="cheese">

<span>Cheddar</span>

</div>

<div class="cheese">

<span>Gouda</span>

</div>

Java: List<WebElement> cheeses = driver.findElements(By.className("cheese"));

Python: cheeses = driver.find\_elements\_by\_class\_name("cheese") **OR**

from selenium.webdriver.common.by import By

cheeses = driver.find\_elements(By.CLASS\_NAME, "cheese")

1. **By Tag Name**

The DOM Tag Name of the element.

Example:

<iframe src="..."></iframe>

Java: WebElement frame = driver.findElement(By.tagName("iframe"));

Python: frame = driver.find\_element\_by\_tag\_name("iframe") **OR**

from selenium.webdriver.common.by import By

frame = driver.find\_element(By.TAG\_NAME, "iframe")

1. **By Name**

Find the input element with matching name attribute.

<input name="cheese" type="text"/>

Java: WebElement cheese = driver.findElement(By.name("cheese"));

Python: cheese = driver.find\_element\_by\_name("cheese") **OR**

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.NAME, "cheese")

1. **By Link Text**

Find the link element with matching visible text.

Example:

< a href ="http://www.google.com/search?q=cheese">cheese</a>

Java: WebElement cheese = driver.findElement(By.linkText("cheese"));

Python: cheese = driver.find\_element\_by\_link\_text("cheese") OR

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.LINK\_TEXT, "cheese")

1. **By Partial Link Text**

Find the link element with partial matching visible text.

Example:

< a href ="http://www.google.com/search?q=cheese">cheese</a>

Java: WebElement cheese = driver.findElement(By.partialLinkText("cheese"));

Python: cheese = driver.find\_element\_by\_partial\_link\_text("cheese") OR

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.PARTIAL\_LINK\_TEXT, "cheese")

1. **By CSS**

Locator strategy is by CSS [Cascading Style sheet]

All browsers were not created equally, some css that might work in one version may not work in another.

Example:

<div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></div>

**Java:** WebElement cheese = driver.findElement(By.cssSelector("#food span.dairy.aged"));

**Python:** cheese = driver.find\_element\_by\_css\_selector("#food span.dairy.aged") OR

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.CSS\_SELECTOR, "#food span.dairy.aged")

1. **By XPath**

XPath is the language used for locating nodes in an XML document. As HTML can be an implementation of XML (XHTML), Selenium users can leverage this powerful language to target elements in their web applications. XPath extends beyond (as well as supporting) the simple methods of locating by id or name attributes, and opens up all sorts of new possibilities such as locating the third checkbox on the page.

One of the main reasons for using XPath is when you don’t have a suitable id or name attribute for the element you wish to locate. You can use XPath to either locate the element in absolute terms (not advised), or relative to an element that does have an id or name attribute. XPath locators can also be used to specify elements via attributes other than id and name.

Absolute XPaths contain the location of all elements from the root (html) and as a result are likely to fail with only the slightest adjustment to the application. By finding a nearby element with an id or name attribute (ideally a parent element) you can locate your target element based on the relationship. This is much less likely to change and can make your tests more robust.

Example:

<html>

<body>

<form id="loginForm">

<input name="username" type="text" />

<input name="password" type="password" />

<input name="continue" type="submit" value="Login" />

<input name="continue" type="button" value="Clear" />

</form>

</body>

<html>

Java: driver.findelement(By.xpath("/html/body/form[1]"))

driver.findelement(By.xpath("//form[1]"))

driver.findelement(By.xpath("//form[@id='loginForm']"))

Python:

login\_form = driver.find\_element\_by\_xpath("/html/body/form[1]")

login\_form = driver.find\_element\_by\_xpath("//form[1]")

login\_form = driver.find\_element\_by\_xpath("//form[@id='loginForm']")

**Java Selenium Documentation:**

By – Different strategies:

By.ByClassName, By.ByCssSelector, By.ById, By.ByLinkText, By.ByName, By.ByPartialLinkText, By.ByTagName, By.ByXPath, ByAll, ByChained, ByIdOrName, BySelector

**Python Selenium Documentation:**

ID = "id", XPATH = "xpath", LINK\_TEXT = "link text", PARTIAL\_LINK\_TEXT = "partial link text", NAME = "name", TAG\_NAME = "tag name", CLASS\_NAME = "class name", CSS\_SELECTOR = "css selector"

**Tips:**

In general, if HTML IDs are available, unique, and consistently predictable, they are the preferred method for locating an element on a page. They tend to work very quickly, and forego much processing that comes with complicated DOM traversals.

If unique IDs are unavailable, a well-written CSS selector is the preferred method of locating an element. XPath works as well as CSS selectors, but the syntax is complicated and frequently difficult to debug. Though XPath selectors are very flexible, they're typically not performance tested by browser vendors and tend to be quite slow.

Selection strategies based on link text and partial link text have drawbacks in that they only work on link elements. Additionally, they call down to XPath selectors internally in WebDriver.

Tag name can be a dangerous way to locate elements. There are frequently multiple elements of the same tag present on the page. This is mostly useful when calling the findElements (By) method which returns a collection of elements.

The recommendation is to keep your locators as compact and readable as possible. Asking WebDriver to traverse the DOM structure is an expensive operation, and the more you can narrow the scope of your search, the better.

**References:**

Selenium Documentation: <https://www.seleniumhq.org/docs/03_webdriver.jsp>

Java Doc: <https://seleniumhq.github.io/selenium/docs/api/java/>

Python Doc: <https://seleniumhq.github.io/selenium/docs/api/py/api.html>

<https://seleniumhq.github.io/docs/start.html#locating_multiple_elements>