To create a webpage we need to use tagname <html> and to create a component or element we need to use tagname <input>

To create a listbox we use <select> tag

To create a link we use <a> tag (anchor)

To create a web table we use <table> tag

1. Tag Name - Any keyword which is present after < symbol is known as tagname. E.g. <html> <body> <input>
2. Attribute – Any keyword which is present after tagname with = symbol. E.g. type = ‘text’ value = ‘Login’ (type : attribute name & ‘text’ : attribute value)
3. Text - Any keyword which is present (><) in between greater than symbol (>) & less than symbol (<). E.g. link1, India, Aus

<html>

<body>

UN<input type = ‘text’ id = ‘abc’> </br>

PWD<input type = ‘password’ id = ‘456’> </br>

<a href = ‘url’> Link1 </a>

<input type = ‘Button’ value = ‘Login’> </br>

</body>

</html>

­­­­­­

**Locators :**

1. Locators are used to identify an element present in a webpage with the help of locator types.
2. To identify an element present in a webpage we need to use findElement() method which is present in WebDriver interface.
3. findElement() method will identify an element with the help of “By” class which contains static methods.
4. All the static methods present in a By class are known as locator types.

There are different types of locators:

* 1. ID
  2. Class
  3. Name
  4. Tagname
  5. Linked Text
  6. Partial Linked Text
  7. CSS Selector
  8. X-path

**Xpath :**

WebDriver driver = new ChromeDriver();

driver.get(“url”);

driver.findElement(By.xpath(“xpath\_expression”)).sendkeys(“abc”);

Types of xpath :

1. xpath by attribute

Syntax :

//tagname[@attribute name = ‘attribute value’]

e.g. :- //input[@id = ‘abc’], //input[@id='autocomplete']

driver.findElement(By.xpath(“//input[@id = ‘abc’]”).click();

1. xpath by text

Syntax :

//tagname[text() = ‘textvalue’]

e.g. :- //a[text() = ‘link1’]

1. xpath by contains()
2. Here we can mention the substring without writing a whole text, when link/text is long in size.
3. If any non-breakable space (nbsp) is there.

We can use contains in two ways :

1. Contains with text() :

//tagname[contains(text(),’textvalue’)]

//h2[text()='Facebook helps you connect and share with the people in your life.']🡪 normal text method

//h2[contains(text(),'Facebook ')]🡪using contains text

//a[contains(text(),‘link1’)]

1. Contains with attribute :

//tagname[contains(@attribute name, ‘attribute value’)]

//input[contains(@name,‘first’)], //img[contains(@alt,'gle')]

1. xpath by index

(//tagname[@attribute name = ‘attribute value’] )[2]

(//input[@type = ‘text’] )[2]

(//input[@name='radio'])[2]

(//h1[contains(text(),'Practice Page')])[1]

1. Absolute and Relative xpath

html/body/div[2]/input[1]

//div[2]/input[1]

Root🡪 html/body/ div[1]/ input[1]🡪absolute

// div[1]/input[1]🡪 relative xpath

/html/body/div[2]/input[1]-🡪//div[2]/input[1]

**HTML Tree Diagram**

html

body

div[1]

input[1] – URL

input[2] – PWD

div[2]

input[1] – male

input[2] – female

div[3]

a[1] – link1

a[2] – link2

input – submit

<html>

<body>

<div>

UN<input type = ‘text’>

PWD<input type = ‘password’>

</div>

<div>

male<input type = ‘radio’>

female<input type = ‘radio’>

</div>

<div>

<a href = ‘\_’> link1

<a href = ‘\_’> link2

<button type = ‘submit’ value = ‘login’>

</div>

</body>

</html>

**Absolute Relative**

/html/body/div[3]/a[1] //div[3]//a[1]

/html/body/div[3]/input //div[3]//input or input[5]

/html/body/div[2]/input[2] //div[2]//a[2] or input[4]

**Disadvantages of Absolute xpath**

1. Absolute xpath is too lengthy and time consuming.
2. Identifying an element by developing html tree diagram is difficult.

**Absolute xpath**

Absolute xpath is used to navigate from root of parent to its **immediate** child. To achieve absolute xpath, we need to use ‘/’.

**/html/body/div[2]/input[1]**

**Relative xpath**

Relative xpath is used to navigate from parent to **any child**. To achieve relative xpath, we need to use ‘//’.

//div[2]//input[1]

html/body/section/div[1]/div[1]

//section[2]//div//div

CSS selector

|  |  |  |
| --- | --- | --- |
| **Method** | **Target Syntax** | **Example** |
| Tag and ID | css=*tag*#*id* | css=input#email |
| Tag and Class | css=*tag*.*class* | css=input.inputtext |
| Tag and Attribute | css=*tag*[*attribute*=*value*] | css=input[name=lastName] |
| Tag, Class, and Attribute | css=tag.class[attribute=value] | css=input.inputtext[tabindex=1] |

How to handle dynamic XPath

dynamic XPath expressions in automation testing is a common challenge, as web pages often generate elements with changing attributes or values. Dynamic XPath refers to XPath expressions that are subject to change due to factors like session IDs, timestamps, or random values. Here are strategies to handle this situation:

Using Partial Matching:

Utilize XPath functions like contains() or starts-with() to match a portion of the dynamic attribute value. For instance:

xpath

//input[contains(@id, 'dynamicPart')]

Using text() and ancestor Relationships:

If the dynamic element is identified by its text content or its relationship with another element, you can use these attributes for more stable identification:

xpath

//label[text()='Dynamic Text']/following-sibling::input

Using following and preceding Axes:

These axes allow you to navigate to elements after or before another element. This can be helpful when dealing with sibling elements that have a predictable order.

xpath

//div[@class='dynamicDiv']/following::input

Combining Multiple Attributes:

If an element has multiple attributes that remain stable, use them in combination to form a unique identification:

xpath

//input[@id='staticID' and @class='dynamicClass']

Parent-Child Relationship:

Traverse the hierarchy of parent and child elements to create a more stable XPath:

xpath

//div[@class='parentDiv']//input[@class='dynamicClass']

Using position() Function:

If the element's position in the hierarchy is stable, you can use the position() function to target it:

xpath

(//button[@class='dynamicClass'])[2]

Using Indexed Position:

If you know that the dynamic element is consistently the nth child of a certain type, you can use its index:

xpath

//div[@class='dynamicContainer']/input[3]

Using descendant Axis:

This axis can be useful when the dynamic element is nested within another element:

xpath

//div[@class='parentDiv']//descendant::input[@class='dynamicClass']