Locator's Strategies for Locating WebElement

Locators are used in selenium WebDriver to find an element on a **HTML DOM**. Locating elements in Selenium WebDriver is performed with the help of findElement() and findElements() methods provided by WebDriver Interface.

> **findElement()** returns a WebElement object based on a specified search criteria or ends up throwing an exception if it does not find any element matching the search criteria.

> **findElements()** returns a list of WebElements matching the search criteria. If no elements are found, it returns an empty list.

Types of Locators used for locating WebElements

1. **Id**: Locates elements whose ID attribute matches the search value

```
driver.findElement(By.id("idValue"));
```

2. **CSS Selector** : Locates elements matching a CSS selector

```
driver.findElement(By.cssSelector("input[type='submit']"));
```


3. **Xpath** : Locates elements matching an Xpath expression

```
driver.findElement(By.xpath("//input[@type='submit']"));
```

4. **Name** : Locates elements whose NAME attribute matches the search value

```
driver.findElement(By.name("nameValue"));
```

5. **Class Name** : Locates elements whose class name contains the search value.

```
driver.findElement(By.className ("classValue"));
```

6. **Tag Name** : Locates elements whose Tag Name matches the search values

```
driver.findElement(By.tagName ("html tagName"));
```

7. **Link Text** : Locates anchor elements whose visible text matches the search values

```
driver.findElement(By.linkText ("Text in the Link"));
```

8. **Partial Link Text** : Locates anchor elements whose visible text contains the search values. If multiple elements are matching, only the first one will be selected.

```
driver.findElement(By.partialLinkText ("Partial Text in the Link"));
```

What is Xpath?

> Xapth stands for **XML Path Language**
>
> Xpath is used to find the location of any elements on a webpage using HTML DOM structure.
>
> It was defined by the World Wide Web Consortium (W3C)
>
> Xpath is used to navigate nodes in any XML document
>
> Xpath uses "path like" syntax

Basic Format of Xpath

Syntax :

```
Xpath = // tagname [ @Attribute = 'Value']
```

Types of Xpath

- **Absolute Xpath**

> It contains the complete path from the root element of page to the desired element.

> Absolute Xpath starts with root node - Single Forward Slash (/)

> Drawback of using absolute Xpath - Any slight change in HTML DOM makes the Xpath invalid.

Ex:

```
html/body/div[5]/div[1]/div[2]/div/div/form/div/div/input
```

- **Relative Xpath**

> With relative Xpath, the Xpath starts from the mid of the HTML DOM structure.

> It begins with the Double Forward Slash (//)

> It is less brittle

Ex:

```
//input[@id='First_Name']
```

Xpath function - "starts-with"

> "starts-wwith" function is very useful in finding dynamic web elements

> You can use it to match the starting value of web element which remains static.

Ex :

```
(id = session77482, sesssion43595) -->
Xpath = //input[starts-with(@id,'session')]
```



```
Xpath = //p[starts-with( text(), 'required text here')]
```

> "starts-with" function can also find the element whose attribute value is static.

Syntax:

```
Xpath = //tagname[starts-with(@Attribute,'value')]
```

Xpath function - "contains"

> "contains" function is used for finding dynamic web elements.

Ex:

```
(id=user[first_name])  --> Xpath = //input[contains(@id,'first_name')]
```

> You can provide any partial attribute value to find the web element. Contains can find elements using partial or complete text in the attribute

Syntax :

```
Xpath = //tagname(contains[@Attribute,'Value'])
```

```
Xpath = //p[contains( text(), 'required text here')]
        //p[contains( @id, 'id1')  and text()='unique text']
        --> combination of contains & text()
```

Xpath function - "text()" method

> "text()" method is used to find element with exact text match(exact visible text on UI)

Syntax :

```
Xapth = //tagname[text()='requiredElementText']
```

How to use AND & OR operator in selenium Xpath

> AND & OR expressions can also be use in selenium Xpaath expression.

> Very helpful if you want to use more than two attributes to find element on webpage.

> use **AND** expression, when web element is to be located using both attributes

> use **OR** expression, when web element is to be located using either of two attributes.

Synatx :

```
Xpath = //tagname[@Attribute='Value' or @Attribute='Value']

Xpath = //tagname[@Attribute='Value' and @Attribute='Value']
```

Ex :

```
Xpath = //b[.='Please wait! Loading Data' or 'Please wait! Loading WorkOrder']

Xpath = //b[.='Please wait! Loading Data' and 'Please wait! Loading WorkOrder']
```

Xpath Axes methods (Parent, Child, Self)

> It represents a relationship to the context node(required web element node)

> It is used in locating node's relative to **that** (required node) node in the tree (DOM structure).

1. Parent

> Selects the parent of the context (Current) node

Syntax :

```
Xpath = //tagname= [@Attribute='value']//parent::tagname
```

> We can Identify a parent tag uniquely using child attribute.

Ex :

```
Xpath = //h2[.='View Details']//parent::div
```

> We can traverse from child to parent using **(/..)**, when child is unique and parent is not.


```
< a href="https://google.com/">
  <span id="link">Google link</span>
```

```
Xpath = //span[@id='link'] /..
```

2. Child

> Selects all the children of the context (Current) node

Syntax :

```
Xpath = //tagname= [@Attribute='value']//child::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//child::div/textarea
```

> We can traverse from parent to child using **(/)**, when parent is unique and child is not.

```
Xpath = //div[@id='divA'] / div
```

> We can traverse from parent to child or descendant using **(//)**, when parent is unique and child or descendant is not.

```
Xpath = //div[@id='divA'] // div     :- it basically means any where inside the
current element.
```

3. Self

> Selects the Current node

Syntax :

```
Xpath = //tagname= [@Attribute='value']//self::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//self::div
```


4. Indexes

> Only to be used when elements have same attributes and multiple elements are found

Syntax :

```
//p[@id='gridColumn'][2]      --> this will looks for 2nd child of the parent

(//p[@id='gridColumn'])[2]    --> this means this element is present 2 time and we
are looking for 2nd element.
```

5. Elements as attribute

> Identification of an element by usin another element as an attribute

> Child is unique and parent is not unique, so we can identify the parent by giving the child as an attribute.

```
<a href="https://google.com/">
   <span id="link">Google link</span>
</a>
```

```
Xpath = //a[ span[ @id = 'link'] ]  --> here instead of giving attribute in anchor tag we have given element.
```

> if we want to identify parent which is not unique and descendant is unique, we can traverse from descendant to parent by using **(.//)**

```
<a href="https://google.com/">
  <div class="xyz">
     <span id="link">Google link</span>
  </div>
</a>
```



```
Xpath = //a[ .//span[ @id = 'link'] ]  --> (.) here denotes we are searching directly inside the anchor tag
```

Xpath Axes methods (descendant, descendant-or-self)

1. descendant

> It selects all of the descendants (childern, grandchildren, etc) of the context (Current) node

Syntax :

```
Xpath = //tagname= [@Attribute='value']//descendant::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//descendant::div
```

2. descendant-or-self

> It selects context (Current) node and all of its descendants (childern, grandchildren, etc), if tagname for descendants and self are same

Syntax :

```
Xpath = //tagname= [@Attribute='value']//descendant-or-self::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//descendant-or-self::div
```

Xpath Axes methods (ancestor, ancestor-or-self)

1. ancestor

> It selects all of the ancestors (parent, grandparent, etc) of the context (Current) node

Syntax :

```
Xpath = //tagname= [@Attribute='value']//ancestor::tagname
```

Ex :


```
Xpath = //div[.='Add comments']//ancestor::div
```

2. ancestor-or-self

> It selects context (Current) node and all of its ancestor (parent, grandparent, etc), if tagname for ancestors and self are same

Syntax :

```
Xpath = //tagname= [@Attribute='value']//ancestor-or-self::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//ancestor-or-self::div
```

Xpath Axes methods (following, following-sibling)

1. following

> It selects all the nodes that appear after the context (Current) node

Syntax :

```
Xpath = //tagname= [@Attribute='value']//following::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//following::div
```

2. following-sibling

> It selects all the nodes that have the same parent as the context (Current) node and appear after the context (current) node.

Syntax :

```
Xpath = //tagname= [@Attribute='value']//following-sibling::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//following-sibling::div
```


Xpath Axes methods (preceding, preceding-sibling)

1. preceding

> **It selects all the nodes that appear before the context (Current) node**

Syntax :

```
Xpath = //tagname= [@Attribute='value']//preceding::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//preceding::div
```

2. preceding-sibling

> **It selects all the nodes that have the same parent as the context (Current) node and appear before the context (current) node.**

Syntax :

```
Xpath = //tagname= [@Attribute='value']//preceding-sibling::tagname
```

Ex :

```
Xpath = //div[.='Add comments']//preceding-sibling::div
```

Important Xpaths methods & function

1. Position

> **It is used to unique identify the position of the multiple rows or grid in the table. It is similar as indexes.**

Ex :

```
Xpath = //table[@id='table1']//tr[3]  --> by using indexes

Xpath = //table[@id='table1']//tr[position() = 3]  --> by using position method
```


2. Last

> **It is used to unique identify the last row. it is helpful when last no of row or row count is not known.**

Ex :

```
Xpath = //table[@id='table1']//tr[last()]    --> returns last row element

Xpath = //table[@id='table1']//tr[last()-2]  --> returns 3rd last row ele
```

3. Count

> **It is used to return the total number of instances of an element found.**

Ex :

```
Xpath = //table[count (.//tr) = 6]    --> returns web table having total number of
tr elements equal to 6 rows
```

4. normalize-space

> **It is used to remove the white spaces from preceding & following spaces of web element. It does not remove space in between two words. Ex :**

```
normalize-space(' apple ')  --> output='apple'
```

Syntax :

```
Xpath = //tagName[normalize-space(text())='expectedText']
Xpath = //tagName[normalize-space(@Attribute)='AttributeValue']
```

```
Xpath = //p[normalize-space( text()) = 'Tommy']    --> removes preceding space and
trailing space 'Tommy' word, if any.
```

5. Translate(ignore-case)

> **Using translate function we can make case insensitive. And it is also used to replace text.**

Synatx :


```
Xpath = //tagName[translate(text(),'abc..','ABC...')='expectedText']
```

Ex :

```
translate( String, str1, str2)
translate( "abcd", "ac", "xy") = xbyd
translate( "ANY_TEXT", "ABC....Z", "abc...z") = any_text
```

```
Xpath = //p[translate()( text(),
"ABCDEFGHIJKLMNOPQRSTUVWXYZ","abcdefghijklmnopqrstuvwxyz") = 'tommy']    -->
returns a web element with case insensitve
```

```
Xpath = //p[normalize-space(translate()( text(),
"ABCDEFGHIJKLMNOPQRSTUVWXYZ","abcdefghijklmnopqrstuvwxyz")) = 'tommy']    -->
returns a web element with case insensitve and without preceding & trailing spaces
```

6. String Length

> **It returns the total number of characters which is present in the given string.**

Ex :

```
Xpath = //p[string-length(text())<30] --> returns paragraphs which have less than
30 characters in web page
```

6. Round

> **It returns the web element which have the round off to higher most value(whole number) of the given number. (53.80 --> 54)**

Ex :

```
Xpath = //td[round(text())='54'] --> returns web elements which are rounded off to
54 in web page.
```

7. Floor

> **It returns the web element which have the floor off to lower most value(whole number) of the given number. (53.80 --> 53)**

Ex :


```
Xpath = //td[floor(text())='53'] --> returns returns web elements which are
floored off to 53 in web page.
```

8. Not

> **It returns the either web element which have been selected**

Ex :

```
Xpath = //input[@type='radio' and @id='gender_0'] --> returns male radio button.
Xpath = //input[@type='radio' and not (@id='gender_0')] --> returns female radio
button.
```

9. Sub-string

> **Sub-string after : it return the text after then giving condition & value**

> **Sub-string before : it return the text before then giving condition & value**

> **Text --> Larry has logged in successfuly : 10am**

Ex :

```
Xpath = //p[substring-after(text(),':')='10am'] --> returns web elements which
conatins "Larry has logged in successfuly".

Xpath = //p[substring-before(text(),':')='Larry has logged in successfuly'] -->
returns web elements which conatins "10am".
```

CSS Selectors Tutorial

> **CSS stands for Cascading Style Sheet**

> **CSS is a style sheet language which describes the presentation of the HTML document.**

> **CSS selectors are used to target the HTML documents on the web page**

```
Syntax : tagName[Attribute='AttributeValue']
```



```
Ex : input[id='first_name']
```

CSS Selector : Select by ID

> **If the web element has an ID attribute you can use the ID attribute details in CSS selector.**

> **ID attribute in CSS selector is symbolised by the HASH (#) sign**

```
Syntax : tagName#elementID
```

```
Ex : input#first_name
```

CSS Selector : Select by Class Attribute

> **If the web element has an CLASS attribute you can use the CLASS attribute details in CSS selector.**

> **CLASS attribute in CSS selector is symbolised by the DOT (.) sign**

```
Syntax : tagName.elementClass
```

```
Ex : input.signup
```

CSS Selector : Select by using Other Web element Attribute

> **You can create CSS selecctor for web elements which have other attributes as well like "type", "placeholder", "value", etc.**

```
Syntax : tagName[AttributeName='Attribute.Value']
```

```
Ex : input[value='Sign me up']
```

Advanced CSS Selector : Select by using mix of Tag, ID and Classname

> **You can create CSS selecctor using mix of ID or Class Name and other attributes of web elements.**


```
Syntax : tagName.classValue[AttributeName='AttributeValue']
         tagName#idValue[AttributeName='AttributeValue']
```

```
Ex : input.signup[type='submit'][value='Sign me up']
     input#submit_btn[type='submit'][value='Sign me up']
```

CSS Selector : Sub-String

> **Sub-string matches are very helpful in identifying dynamic web elements with the help of partial string matches**

> **The 3 important special characters is css sub-string selectors are :**

```
'^' sign - signify's the prefix of the text
'$' sign - signify's the suffix of the text
'*' sign - signify's the sub-string of the text
```

Ex :

```
Match prefix of the text : input[name^='country_c']
Match suffix of the text : input[name$='y_client']
Match sub-string of the text : input[name*='try_cl']
```

Different Ways to Define WebElements

```
1. @FindBy(xpath ="//div[@class='filterIcon']")
   WebElement filterIcon;

2. By mySelector = By.xpath("//div[@class='filterIcon']");
   WebElement myElement = driver.findElement(mySelector);

3. @FindAll({ @FindBy(xpath = "//select[@id='GridID']//option") })
   List<WebElement> gridsIDs;

4. By mySelector = By.xpath("//select[@id='GridID']//option");
   List<WebElement> myElements = driver.findElements(mySelector);

5. @FindBy(how = How.XPATH, using = "//span[.='Assign to User']")
   @CacheLookup
   WebElement reassignAction;

6. String headerXpath="//div[contains(@class,'row head')]//div";
   String rowXpath="//div[contains(@class,'tr row')]//div";

   WebElement headerElement = driver.findElement(By.xpath(headerXpath));
   WebElement rowElement = driver.findElement(By.xpath("//div[contains(@class,'tr
row')]//div"));
```


Different ways to Initialize/Use WebElment

```
1. public boolean isElementDisplayed(WebElement element) {
   boolean isElementDisplayed = element.isDisplayed();
   return isElementDisplayed;
 }

2. public boolean isElementDisplayed(WebDriver driver, By locator) {
   boolean isElementDisplayed = driver.findElement(locator).isDisplayed();
   return isElementDisplayed;
 }

3. public boolean isElementDisplayed((WebDriver driver, String locatorStr) {
   boolean isElementDisplayed =
driver.findElement(By.xpath(locatorStr)).isDisplayed();
   return isElementDisplayed;
 }
```

What is most Preferred Locator to find WebElement?

Locators	When to Use
ID	Use ID if available, Unique & Consistent (Fastest Locator)
CSS Selectors	If Unique ID is not available
Xpath	If CSS Selector can't be used

Relative Locators (Selenum 4 feature)

Selenium 4 introduces Relative Locators *(previously called Friendly Locators)*. These locators are helpful when it is not easy to construct a locator for the desired element, but easy to describe spatially where the element is in relation to an element that does have an easily constructed locator.

Refer: Selenium Documentation detailed explaination

Above

> **If the email text field element is not easily identifiable for some reason, but the password text field element is, we can locate the text field element using the fact that it is an “input” element “above” the password element.**

```
By emailLocator =
RelativeLocator.with(By.tagName("input")).above(By.id("password"));
```

Below

> **If the password text field element is not easily identifiable for some reason, but the email text field element is, we can locate the text field element using the fact that it is an “input” element “below” the email element.**

```
By passwordLocator =
RelativeLocator.with(By.tagName("input")).below(By.id("email"));
```

Left of

> **If the cancel button is not easily identifiable for some reason, but the submit button element is, we can locate the cancel button element using the fact that it is a “button” element to the “left of” the submit element.**

```
By cancelLocator =
RelativeLocator.with(By.tagName("button")).toLeftOf(By.id("submit"));
```

Right of

> **If the submit button is not easily identifiable for some reason, but the cancel button element is, we can locate the submit button element using the fact that it is a “button” element “to the right of” the cancel element.**

```
By submitLocator =
RelativeLocator.with(By.tagName("button")).toRightOf(By.id("cancel"));
```


Near

> **If the relative positioning is not obvious, or it varies based on window size, you can use the near method to identify an element that is at most 50px away from the provided locator. One great use case for this is to work with a form element that doesn’t have an easily constructed locator, but its associated input label element does.**

```
By emailLocator = RelativeLocator.with(By.tagName("input")).near(By.id("lbl-
email"));
```

Chaining relative locators

> **You can also chain locators if needed. Sometimes the element is most easily identified as being both above/below one element and right/left of another.**

```
By submitLocator =
RelativeLocator.with(By.tagName("button")).below(By.id("email")).toRightOf(By.id("
cancel"));
```