**Selenium Introduction:**

**What is selenium?**

* Selenium is an open-source tool/framework/set of API’S/project for a range of libraries used to automate web application on different browser & platform.
* Selenium provides remote control of browser instances which emulate user’s interaction with the browser.
* It supports many languages to write automation script
* It supports many IDE (Integrated Development Environment) like eclipse, Visual studio, PyCharm etc. to write automation scripts using different languages.
* It supports only web-application automation.
* Selenium is a well-known antidote for mercury poison.

It contains following tool’s:

1. Selenium IDE (Integrated Development Environment)
2. Selenium RC (Remote Control)
3. Selenium WD (WebDriver)
4. Selenium Grid

Diagram

Description automatically generated

Selenium IDE:

* It is a Firefox extension that can automate web applications by Record & Playback feature.
* In 2006, Selenium IDE is merged with selenium project.

|  |  |
| --- | --- |
| Advantages | Dis-Advantages |
| Very easy to install | Available only for Firefox |
| No programming knowledge is required | No support for customized test operations |
| Built in test reports | Slow in execution compared with RC & WD |
| Open source | We can run the recorded scripts only in Firefox browser |

**Selenium RC :**

* It is a server written in java.
* It is the first automated web-testing tool that support all programming languages.
* Selenium RC merged with web driver in 2008 and formed selenium-2. After new updates & improvements introduced selenium-3.

|  |  |
| --- | --- |
| Advantages | Dis-Advantages |
| Faster than selenium IDE | Slower than selenium WD |
| Support for customized test operations | Need selenium RC server to execute test cases |
| Support parallel testing | Complicated in installation |
| Open source | Must have any programming skills |
| Support Data Driven framework | Internally RC used JavaScript to automate web-applications |
| Selenium RC has in-built HTML report generation features for test results | No support for Headless HTMLUnit browsers (Invisible browser) |

Diagram

Description automatically generated

**Selenium WD :**

* Founded by Simon Stewart in 2006.
* Selenium WebDriver was the first cross-platform testing framework that could control the browser from OS level.

|  |  |
| --- | --- |
| Advantages | Dis-Advantages |
| Faster than selenium RC & IDE | Installation is Complicated then selenium IDE |
| Support for customized test operations | Must have any programming skills |
| support parallel testing | It does not rely on JavaScript for automate web-applications |
| Open source | Does not provide test results directly without third party tools. |
| Interact directly with browser through Driver |  |
| Not required any server to execute testcases |  |
| Customized test results |  |
| Support for Headless HTMLUnit browsers (Invisible browser) |  |

Diagram

Description automatically generated

**Selenium Grid :**

* It is a tool used to run Testcases parallel on different machines, different browsers, and multiple versions of browsers at the same time.
* It has HUB and NODES features.
* Hub is a central source. it provides selenium commands to Nodes.
* Nodes perform actions that are initiated by Hub.

Diagram

Description automatically generated

**WebDriver interface:**it is an interface provided by selenium. It consists constant and abstract methods. These abstract methods are implemented in respective browser driver class like, ChromeDriver , FirefoxDriver etc.

Example :

WebDriver driver = new ChromeDriver();

**Launching Browsers :**

**Chrome Browser :**

**Note : check the chrome browser version first and download chrome driver which is compatibility with that version.**

**Syntax:**

**System.setProperty("webdriver.chrome.driver","chromedriver path");**

**WebDriver chrome = new ChromeDriver();**

* We need to specify driver and its path in set property method so that system can identify driver at the given path.
* Second line will launch the Chrome browser.

**Firefox Browser :**

**Note1 : check the firefox browser version first and download gecko driver which is compatibility with that version.**

**Note2: It's compatible with most browsers**

**Syntax:**

**System.setProperty("webdriver.gecko.driver","geckodriver path");**

**WebDriver firefox = new FirefoxDriver();**

**IE Browser :**

**Note1 :**

* First, we must check the Protection Mode of our IE browser.
* **Launch Internet Explorer → move to Tools Menu → Internet Options → Security tab.**
* **Check** the option Select **Enable Protected Mode**. Also, we must choose the Internet as the zone. We can make the level as Medium-high. The other Protection zones like **Local Intranet** and **Trusted sites** (apart from **Restricted sites**) should also have the **same configuration**. Once all the configurations are done, we have to Apply and then click OK.
* check the ie browser version first and download ie driver which is compatibility with that version.

**Note2:** Among all browsers, Internet Explorer is slower than the other drivers.

**Syntax:**

**System.setProperty("webdriver.ie.driver","iedriver path");**

**WebDriver ie = new InternetExplorerDriver();**

**Headless Browser (Non-UI mode) :**

**Headless browser:**

* A headless browser is same as a real browser except it do not have a GUI.
* All modern browsers provide the capabilities to run them in a headless mode
* when Selenium tests run in headless browsers, they execute in the background.
* Selenium supports headless browser testing using HtmlUnitDriver.
* HtmlUnitDriver is based on java framework HTMLUnit and is the one of the lightweight and fastest among all headless browser.
* In older versions of Selenium, we used the HTMLUnitDriver mainly, a headless driver providing a non-GUI implementation of Selenium WebDriver.
* But with the latest versions of Selenium WebDriver 3 and Selenium WebDriver 4, Selenium also supports headless versions of real browsers like Chrome, Firefox, and Edge.

**Running Selenium test cases using HTMLUnitDriver :**

* Fastest execution of WebDriver compared to other browsers.
* HtmlUnitDriver is platform independent.
* HtmlUnitDriver supports JavaScript.
* Also, HtmlUnitDriver allows you to choose other browser versions to run your scripts. You can mention different browser versions of Chrome or Firefox in the HtmlUnitDriver itself.
* One of the significant benefits of using headless browsers is performance. Since headless browsers don't have a GUI, they are faster than real browsers.

**Syntax:**

HtmlUnitDriver driver = new HtmlUnitDriver();

**Running tests on different browser versions using HtmlUnitDriver :**

* **Chrome syntax:**

HtmlUnitDriver chrome = new HtmlUnitDriver(BrowserVersion.***CHROME***);

* **Firefox syntax:**

HtmlUnitDriver firefox = **new** HtmlUnitDriver(BrowserVersion.***CHROME***);

* **IE syntax:**

HtmlUnitDriver ie = **new** HtmlUnitDriver(BrowserVersion.***INTERNET\_EXPLORER***);

* **Edge syntax:**

HtmlUnitDriver edge = **new**  HtmlUnitDriver(BrowserVersion.***EDGE***);

**Running Selenium headless browser tests using the headless Chrome browser.**

**NOTE** : Google Chrome is available with headless execution since version 59.

* Selenium WebDriver provides a class called "**ChromeOptions**"
* ChromeOptions can specify certain configurations to change the default behavior of Chrome.
* One of those configurations is the "**headless**" mode, which launches the Chrome in headless mode.

**Syntax:**

ChromeOptions options = **new** ChromeOptions();

options.addArgument("headless"); WebDriver driver = **new** ChromeDriver(options);

**Running Selenium headless browser tests using the headless Firefox browser.**

* Selenium WebDriver provides a class called " **FirefoxOptions**"
* FirefoxOptions can specify certain configurations to change the default behavior of Firefox.

**Syntax:**

FirefoxOptions options = **new** FirefoxOptions(); options.setHeadless(**true**); WebDriver driver = **new** FirefoxDriver(options);

**Selenium RemoteWebDriver Methods :**

1. **get**()

This method used to navigate the url passed as parameter. It wait until page load completely.

**Syntax:**

WebDriver driver = new ChromeDriver();

driver.get(“http://google.co.in”);

1. **getTitle**()

This method is used to get the title of the current tab or webpage.

**Syntax:**

WebDriver driver = new ChromeDriver();

String title = driver.getTitle();

System.out.println(title);

1. **getCurrentUrl**()

This method is used to get the URL of the current tab or webpage.

**Syntax:**

WebDriver driver = new ChromeDriver();

String url = driver.getCurrentUrl();

System.out.println(url);

1. **getPageSource**()

This method is used to get the page source code of the current tab or webpage.

**Syntax:**

WebDriver driver = new ChromeDriver();

String pagesource = driver.getPageSource();

System.out.println(pagesource);

1. **close()**:

This method is used to close the current tab or current window.

**Syntax:**

WebDriver driver = new ChromeDriver();

driver.close();

1. **quit()**:

This method is used to close the current tab or current window.

**Syntax:**

WebDriver driver = new ChromeDriver();

driver.quit();

1. **getSessionID**():

This method used to get session-id of webpage.

Syntax:

WebDriver driver = new ChromeDriver();

String sessionid = ((RemoteWebDriver)driver).getSessionID();

System.out.println(sessionid);

1. **getCapabilities**():

The capabilities property is a dictionary containing information about the configuration**.**

1. **getBrowserName():**

This method used to get name of the browser.

Syntax:

ChromeDriver driver = new ChromeDriver();

String browsername = driver.getCapabilities().getBrowserName();

System.out.println(browsername);

**Or**

WebDriver driver = new ChromeDriver();

String browsername = ((RemoteWebDriver)driver).getCapabilities().getBrowserName();

System.out.println(browsername);

1. **getPlatform**()

This method used to get the name of the platform.

**Syntax:**

WebDriver driver = new ChromeDriver();

String platformname = ((RemoteWebDriver)driver).getCapabilities().getPlatform();

System.out.println(platformname);

1. **getVersion**()

This method used to get the version of the browser.

**Syntax:**

WebDriver driver = new ChromeDriver();

String version = ((RemoteWebDriver)driver).getCapabilities().getVersion);

System.out.println(version);

1. **getCapability**()

This method used to get the version of the browser.

**Syntax:**

WebDriver driver = new ChromeDriver();

String version = ((RemoteWebDriver)driver).getCapabilities().getCapability(“browserVersion”);

System.out.println(version);

1. **getCapabilityNames**()

This method used to get the version of the browser.

**Syntax:**

WebDriver driver = new ChromeDriver();

Set<String> caps = ((RemoteWebDriver)driver).getCapabilities().getCapabilityNames();

for(String cap: caps)

{

System.out.println(cap);

}

**Selenium RemoteNavigation class methods:**

1. **back()**

This method used to go back page from the current page.

**Syntax**:

WebDriver driver = new ChromeDriver();

driver.navigate().back();

1. **forward()**

This method used to go forward page from the current page.

**Syntax**:

WebDriver driver = new ChromeDriver();

driver.navigate().forward();

1. **refresh()**

This method used to refresh the current page.

**Syntax**:

WebDriver driver = new ChromeDriver();

Driver.navigate().refresh();

1. **to()**

This method used to navigate the url or string passed as parameter.

**Syntax**:

WebDriver driver = new ChromeDriver();

driver.navigate().to(“https://www.google.co.in/”);

OR

URL google = new URL(“https://www.google.co.in/”)

WebDriver driver = new ChromeDriver();

driver.navigate().to(google);

**Selenium RemoteWindow class methods:**

1. **maximize()**

This method used to maximize the browser window.

Syntax:

driver.manage().window().maximize();

1. **fullScreen()**

This method used to set the screen as full screen.

**Syntax**:

driver.manage().window().fullScreen();

1. **getSize()**

This method used to get the size of the browser window.

Return type : Dimension

**Syntax**:

Dimension dimension = driver.manage().window().getSize();

System.out.println(dimension.getHeight());

System.out.println(dimension.getWidth());

1. **setSize()**

This method used to set the size of the browser window.

Parameter type : Dimension

**Syntax**:

Dimension dimension = new Dimension(500,600);

driver.manage().window().setSize(dimension);

1. **getPosition()**

This method used to get the co-ordinates of browser window.

Return type : Point

**Syntax**:

Point point = driver.manage().window().getPosition();

System.out.println(point.getX());

System.out.println(point.getY());

1. **setPosition()**

This method used to set the co-ordinates of browser window.

Parameter type : Point

**Syntax**:

Point point = new Point (500,500);

driver.manage().window().setPosition(point);

**Selenium RemoteWebDriverOptions class methods:**

**Selenium cookie commands:**

**Cookie : Cookie Is a temporary file which can store user information.**

1. **getCookies()**

This method used to get all cookies present in a webpage.

Return type : Set <Cookie>

**Syntax**:

Set<Cookie> cookies = driver.manage().getCookies();

for(Cookie cookie: cookies)

{

System.out.println(cookie.getName());

System.out.println(cookie.getValue());

System.out.println(cookie.getPath());

}

1. **getCookieNamed()**

This method used to get a cookie by its name passed as a parameter.

Parameter Type : String

**Syntax**:

Cookie cookie = driver.manage().getCookieNamed(“cookiename”);

System.out.println(cookie);

1. **addCookie()**

This method used to add a cookie on webpage.

Parameter Type : Cookie

**Syntax**:

driver.manage().addCookie(“key”,”value”);

or

Cookie k = new Cookie(“key” ,”value”);

driver.manage().addCookie(k);

1. **deleteAllCookies()**

This method used to delete all cookies present on a webpage.

**Syntax**:

driver.manage().deleteAllCookies();

1. **deleteCookie()**

This method used to delete a cookie present on a webpage.

Parameter Type : Cookie

**Syntax**:

Cookie k = new Cookie(“key” ,”value”);

driver.manage().deleteCookie(k);

1. **deleteCookieNamed()**

This method used to delete a cookie present on a webpage.

Parameter Type : String

**Syntax**:

driver.manage().deleteCookieNamed(“cookieName”);

**Selenium RemoteWebDriver class methods:**

1. **findElement():**

This method used to locate a WebElement on a webpage.

Parameter Type : By

Return type : WebElement

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.id(“loginbtn”));

1. **findElements():**

This method used to locate multiple WebElement’s on a webpage.

Parameter Type : By

Return type : List<WebElement>

**Syntax**:

WebDriver driver = new ChromeDriver();

List<WebElement> btns = driver.findElements(By.id(“loginbtn”));

**Types of locators in selenium or By class attributes.**

In selenium we have 8 types of locators.

1. Id

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.id(“xxxxxx”));

1. name

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.name(“xxxxxxx”));

1. className

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.className(“xxxxxxx”));

1. tagname

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.tagName(“xxxxx”));

1. linkText

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.linkText(“xxxxxx”));

1. partialLinkText

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.partialLinkText(“xxxxxx”));

1. xpath

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginBtn = driver.findElement(By.xpath(“xxxxx”));

**Xpath in selenium:**

if we are unable to find any locator i.e., id , name, class, text etc. in any tag, then use xpath to locate that tag.

Xpath is categorized into two types

1. **Absolute xpath :**

It starts from the first “/” root tag of the html source code and ends at required tag.it is direct way of finding the WebElement.

Ex: /html/body/div[2]/div/div/div[2]/div[2]/div[1]/form/div[7]/div[2]/div[1]/input

The major dis-advantage of absolute xpath is, if there is any change in elements of the webpage then absolute xpath will not work.

We can get absolute xpath directly by using any tool like Selenium IDE, firebug, chrome inspector etc.

1. **Relative xpath:**

it starts with double forward slash ( // ) and at any tag inside the html source.

Syntax :

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

Ex: //button[@id="hobbies"]

The major advantage of relative xpath is, if there is any change in WebElement of a webpage relative xpath will not affected.

**Different characters in xpath syntax:**

1. Single forward slash ( / ) :

it selects all child tags inside a parent tag.

**Syntax** : //tagname[@attribute=’attributeValue’]/tag

**Example** : //form[@id="userForm"]**/**div

1. Double forward slash( // ) :

it selects all selective tags inside a parent tag.

**Syntax** : //tagname[@attribute=’attributeValue’]//tag

**Example** **: //**form[@id="userForm"]//div

1. Address sigh ( @ ) :

it is used to select a tag based on attribute of a selective tag.

**Syntax** : //tagname[@attribute=’attributeValue’]/tag

**Example** : //form[**@**id="userForm"]

1. Dot ( . ) :

it selects current tag.

**Syntax** : //tagname[@attribute=’attributeValue’]/.

**Example** : //form[@id="userForm"]/**.**

1. Double dot (..) :

it selects parent tag of current tag.

**Syntax** : //tagname[@attribute=’attributeValue’]/..

**Example** : //form[@id="userForm"]/**..**

1. Asterisk ( \* ) :

it selects any tag of html source code.

**Syntax** : //\*[@attribute=’attributeValue’]

**Example** : //**\***[@id="userForm"]

1. Address and Asterisk ( @\* ) :

it selects any attribute of a given tag.

**Syntax** : //tagname[@\*]

**Example** : //form[**@\***]

1. Pipe ( | ) :

it used to select multiple paths.

**Syntax** : //tagname[@attribute=’attributeValue’] | //tagname[@attribute=’attributeValue’]

**Example** : //form[@id="userForm"] **|** //\*[@id="userNumber"]

**Note: xpath and CSS selector use 1-based index.**

**Predicates in xpath** :

Predicates are used to find specific element by its index.

1. index ( [ ] ) :

it selects WebElement based on index from grope of matched WebElement.

**Syntax** : //tagname[@attribute=’attributeValue’]/tag**[index]**

**Example** : //form[@id="userForm"]/div[2]

1. last() :

it selects last WebElement from grope of matched WebElement.

**Syntax** : //tagname[@attribute=’attributeValue’]/tag**[last()]**

**Example** : //form[@id="userForm"]/div[last()]

1. position() :

it selects specific positioned WebElement from grope of matched WebElement.

**Syntax** : //tagname[@attribute=’attributeValue’]/tag**[position()=x]**

**Example** : //form[@id="userForm"]/div[position()=1]

**Functions in xpath** :

1. Contains():

It is used to select WebElement based on partial value of an attribute or text.

**Syntax** : //tagname[contains(@attribute,’partialAttributeValue’)]

( or )

**Syntax** : //tagname[contains(text() , ’partialtext’)]

**Example** : //input[contains(@id,'first')]

**Example** : //label[contains(text(),'Sport')]

1. Starts-with():

It is used to select WebElement based on starting value of an attribute or text.

**Syntax** : //tagname[starts-with(@attribute,’partialAttributeValue’)]

( or )

**Syntax** : //tagname[starts-with(text() , ’partialtext’)]

**Example** : //input[starts-with(@id,'first')]

**Example** : //label[starts-with(text(),'Spo')]

1. text():

It is used to select WebElement based on starting value of an attribute or text.

**Syntax** : //tagname[text() = ’text’]

**Or**

**Syntax** : //tagname[. = ’text’]

**Example** : //label[text()='Sports']

**Example** : //label[.='Sports']

**Operators in xpath** :

1. and :

It is used to combined two different conditions to select WebElement.

**Syntax** : //tagname[@attribute=’AttributeValue’ and @attribute=’AttributeValue’]

( or )

**Syntax** : //tagname[@attribute=’AttributeValue’ and text() =’text’]

**Example** : //input[@placeholder='First Name' and @id='firstName']

**Example** : //input[@placeholder='First Name' and text()='firstName']

1. or :

It is used any one condition to select WebElement.

**Syntax** : //tagname[@attribute=’AttributeValue’ or @attribute=’AttributeValue’]

( or )

**Syntax** : //tagname[@attribute=’AttributeValue’ or text() =’text’]

**Example** : //input[@placeholder='First Name' or @id='firstName']

**Example** : //input[@placeholder='First Name' or text()='firstName']

**Special Attributes in xpath:**

1. child:

It is used in xpath to select all child tags inside the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/child::tag

( or )

**Syntax** : //tagname[@attribute=’AttributeValue’]/tag

**Example** : //\*[@id='userEmail-wrapper']/child::input

**Example** : //\*[@id='userEmail-wrapper']/input

1. descendant:

It is used in xpath to select all selected tags inside the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/descendant::tag

( or )

**Syntax** : //tagname[@attribute=’AttributeValue’]//tag

**Example** : //\*[@id='userEmail-wrapper']/ descendant::input

**Example** : //\*[@id='userEmail-wrapper']//input

1. parent:

It is used in xpath to select parent tag or selective tag of the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/parent::tag

( or )

**Syntax** : //tagname[@attribute=’AttributeValue’]/..

**Example** : //\*[@id='userEmail-wrapper']/ parent::input

**Example** : //\*[@id='userEmail-wrapper']/..

1. Ancestor:

It is used in xpath to select all selective parent tags from the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/ancestor::tag

**Example** : //\*[.='Label']/ancestor::table

1. following:

It is used in xpath to select all selective tags from the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/following::tag

**Example** : //\*[.='Label']/following::table

1. preceding:

It is used in xpath to select all selective tags behind the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/preceding::tag

**Example** : //\*[.='Label']/preceding::table

1. following-sibling:

It is used in xpath to select all selective sibling tags from the current tag.

**Syntax** : //tagname[@attribute=’AttributeValue’]/following-sibling::tag

**Example** : //\*[.='Label']/following-sibling::table

1. CSS Selector

**Syntax**:

WebDriver driver = new ChromeDriver();

WebElement loginbtn = driver.findElement(By.cssSelector(“xxxxx”));

**CSS selector in selenium** : Cascading style sheets

CSS has better performance and speed than xpath because

* Xpath allows bidirectional flow which means the traversal can be both ways from parent to child and child to parent as well
* CSS allows only one directional flow which means the traversal is from parent to child only
* Xpath engines are different in each browser, hence make them inconsistent.
* IE does not have a native xpath engine, therefore selenium injects its own xpath engine for compatibility of its API.
* Every webpage has its CSS at the top in the <head> section and JavaScript at the bottom. This allows the browser to load your CSS first.

**Note: xpath and CSS selector use 1-based index.**

1. **Id**

**Syntax** : tag[id=’idvalue’]

**Example** : input[id=’firstname’] or input**#**firstname

**Note** : **#** symbol indicates ‘ID’ in CSS selector.

1. **Class**

**Syntax** : tag[class=’classvalue’]

**Example** : input[class=’firstname’] or input**.**firstname

**Note** : **.** symbol indicates ‘ID’ in CSS selector.

1. **Any attribute**

**Syntax** : tag[attribute=’attributevalue’]

**Example** : button[placeholder=’acceptcookies’]

**Multiple attribute:**

* **Id & attribute:**

**Syntax** : tag[id=’idvalue’][attribute=’attributevalue’] or tag#idvalue[attribute=’attributevalue’]

**Example** : input[id='firstName'][placeholder='First Name']

or

input#firstName[placeholder='First Name']

* **class & attribute:**

**Syntax** : tag[class=’classvalue’][attribute=’attributevalue’] or tag.classvalue[attribute=’attributevalue’]

**Example** : input[class=’firstname’][placeholder='First Name']

or

input.firstName+[placeholder='First Name']

**Using parent – child hierarchy**

**Syntax** : parenttag>childtag>childtag>…>…>…>

**Example** : form[id='userForm']>div>div>input[id='firstName']

or

form#userForm>div>div>input[id='firstName']

**Note** : **>** symbol indicates child tags in CSS selector.

**Index or nth-of-value()**

**Syntax** : parenttag>childtag : nth-of-type(index)

**Example** : form[id='userForm']>div:nth-of-type(1)>div:nth-of-type(2)>input[id='firstName']

or

form#userForm>div:nth-of-type(1)>div:nth-of-type(2)>input[id='firstName']

**Note** : nth-of-type(index) symbol indicate tags based on index in CSS selector.

**With text**

* Starting text

**Syntax** : tag[attribute^=’attributeStartingValue’]

**Example** : input[id^='firstNa']

**Note** : **^** symbol indicates Start-with text in CSS selector.

* Contains-text

**Syntax** : tag[attribute$=’attributeSomeTextValue’]

**Example** : input[id$='rstNa']

**Note** : **$** symbol indicates Contains-with text in CSS selector.

* Ending text

**Syntax** : tag[attribute\*=’attributeEndTextValue’]

**Example** : input[id\*='tName']

**Note** : **\*** symbol indicates Contains-with text in CSS selector.