**Explaining the code**

**Importing Packages**

To get started, you need to import following two packages:

1. **org.openqa.selenium.\***– contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver**– contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class

If your test needs more complicated actions such as accessing another class, taking browser screenshots, or manipulating external files, definitely you will need to import more packages.

## **Instantiating objects and variables**

Normally, this is how a driver object is instantiated.

First Selenium Webdriver Script: JAVA Code Example

## **Launching a Browser Session**

WebDriver’s **get()** method is used to launch a new browser session and directs it to the URL that you specify as its parameter.

First Selenium Webdriver Script: JAVA Code Example

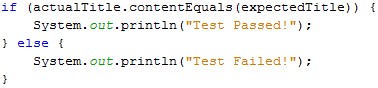
## **Get the Actual Page Title**

The WebDriver class has the **getTitle()** method that is always used to obtain the page title of the currently loaded page.

First Selenium Webdriver Script: JAVA Code Example

## **Compare the Expected and Actual Values**

This portion of the code simply uses a basic Java if-else structure to compare the actual title with the expected one.



## **Terminating a Browser Session**

The “**close()**” method is used to close the browser window.

First Selenium Webdriver Script: JAVA Code Example

## **Terminating the Entire Program**

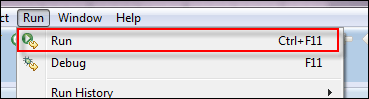
If you use this command without closing all browser windows first, your whole Java program will end while leaving the browser window open.

First Selenium Webdriver Script: JAVA Code Example

## **Running the Test**

There are two ways to execute code in Eclipse IDE.

1. On Eclipse’s menu bar, click **Run > Run.**
2. Press **Ctrl+F11** to run the entire code.



# Locators in Selenium IDE: CSS Selector | DOM | XPath | ID

## **What are Locators?**

Locator is a command that tells Selenium IDE which GUI elements ( say Text Box, Buttons, Check Boxes etc) its needs to operate on.  Identification of correct GUI elements is a prerequisite to creating an automation script. But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all!  Hence, Selenium provides a number of Locators to precisely locate a GUI element

The different types of CSS Locator in Selenium IDE are:

## **Locating by ID**

This is the most common way of locating elements since ID’s are supposed to be unique for each element.

**Target Format:**id=id of the element

## **Locating by Name**

Locating elements by name are very similar to locating by ID, except that we use the **“name=”** prefix instead.

**Target Format:**name=name of the element

## **Locating by Link Text**

This type of CSS locator in Selenium applies only to hyperlink texts. We access the link by prefixing our target with “link=” and then followed by the hyperlink text.

## **Locating by CSS Selector**

CSS Selectors in Selenium are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selectors in Selenium is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.

CSS Selectors in Selenium have many formats, but we will only focus on the most common ones.

* Tag and ID
* Tag and class
* Tag and attribute
* Tag, class, and attribute
* Inner text

## **Locating by XPath**

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use[XPath](https://www.guru99.com/xpath-selenium.html)in locating HTML elements.

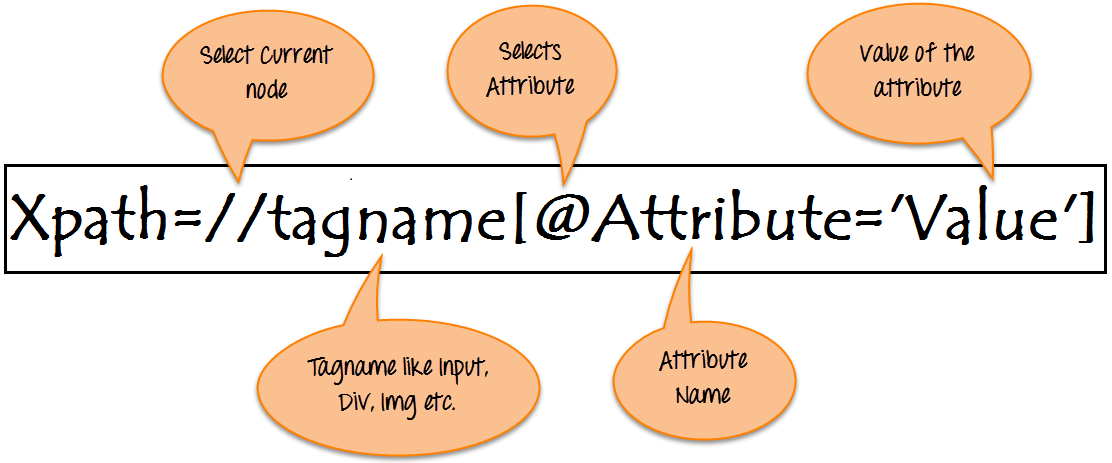
**Advantage:** It can access almost any element, even those without class, name, or id attributes.

**Disadvantage:** It is the most complicated method of identifying elements because of too many different rules and considerations.

## **What is XPath in Selenium?**

**XPath in Selenium** is an XML path used for navigation through the HTML structure of the page. It is a syntax or language for finding any element on a web page using XML path expression. XPath can be used for both HTML and XML documents to find the location of any element on a webpage using HTML DOM structure.

The basic format of XPath in selenium is explained below with screen shot.



Basic Format of XPath

**Syntax for XPath selenium:**

XPath contains the path of the element situated at the web page. Standard XPath syntax for creating XPath is.

Xpath=//tagname[@attribute='value']

* **// :** Select current node.
* **Tagname:**Tagname of the particular node.
* **@:** Select attribute.
* **Attribute:** Attribute name of the node.
* **Value:** Value of the attribute.

To find the element on web pages accurately there are different types of locators:

|  |  |
| --- | --- |
| **XPath Locators** | **Find different elements on web page** |
| **ID** | To find the element by ID of the element |
| **Classname** | To find the element by Classname of the element |
| **Name** | To find the element by name of the element |
| **Link text** | To find the element by text of the link |
| **XPath** | XPath required for finding the dynamic element and traverse between  various elements of the web page |
| **CSS path** | CSS path also locates elements having no name, class or ID. |

## **Types of X-path**

There are two types of XPath:

**1) Absolute XPath**

**2) Relative XPath**

### **Absolute XPath:**

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

**Absolute XPath:** /html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]

### **Relative Xpath:**

**Relative Xpath** starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

Relative XPath: //div[@class='featured-box cloumnsize1']//h4[1]//b[1]

**What are XPath axes.**

XPath axes search different nodes in XML document from current context node. XPath Axes are the methods used to find dynamic elements, which otherwise not possible by normal XPath method having no ID , Classname, Name, etc.

Axes methods are used to find those elements, which dynamically change on refresh or any other operations. There are few axes methods commonly used in [Selenium Webdriver](https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html) like child, parent, ancestor, sibling, preceding, self, etc.

## **How To Write Dynamic XPath In Selenium WebDriver**

### **1) Basic XPath:**

XPath expression select nodes or list of nodes on the basis of attributes like **ID , Name, Classname**, etc. from the XML document as illustrated below.

Xpath=//input[@name='uid']

Some more basic xpath expressions:

Xpath=//input[@type='text']

Xpath= //label[@id='message23']

Xpath= //input[@value='RESET']

Xpath=//\*[@class='barone']

Xpath=//a[@href='http://demo.guru99.com/']

Xpath= //img[@src='//guru99.com/images/home/java.png']

### **2) Contains():**

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

The contain feature has an ability to find the element with partial text as shown in below XPath example.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value ‘sub’ is used in place of submit button. It can be observed that the element is found successfully.

Complete value of ‘Type’ is ‘submit’ but using only partial value ‘sub’.

Xpath=//\*[contains(@type,'sub')]

### **3) Using OR & AND:**

In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.

In the below XPath expression, it identifies the elements whose single or both conditions are true.

Xpath=//\*[@type='submit' or @name='btnReset']

### **4) Xpath Starts-with**

**XPath starts-with()** is a function used for finding the web element whose attribute value gets changed on refresh or by other dynamic operations on the webpage. In this method, the starting text of the attribute is matched to find the element whose attribute value changes dynamically. You can also find elements whose attribute value is static (not changes).

Xpath=//label[starts-with(@id,'message')]

### **5) XPath Text() Function**

The **XPath text() function** is a built-in function of selenium webdriver which is used to locate elements based on text of a web element. It helps to find the exact text elements and it locates the elements within the set of text nodes. The elements to be located should be in string form.

In this expression, with text function, we find the element with exact text match as shown below. In our case, we find the element with text “UserID”.

Xpath=//td[text()='UserID']

### **6) XPath axes methods:**

These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods..

### **a) Following:**

Selects all elements in the document of the current node( ) [ UserID input box is the current node] as shown in the below screen.

Xpath=//\*[@type='text']//following::input

### **b) Ancestor:**

The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen.

In the below expression, we are finding ancestors element of the current node(“ENTERPRISE TESTING” node).

Xpath=//\*[text()='Enterprise Testing']//ancestor::div

### **c) Child:**

Selects all children elements of the current node (Java) as shown in the below screen.

Xpath=//\*[@id='java\_technologies']//child::li

### **d) Preceding:**

Select all nodes that come before the current node as shown in the below screen.

In the below expression, it identifies all the input elements before “LOGIN” button that is **Userid** and **password** input element.

Xpath=//\*[@type='submit']//preceding::input

### **e) Following-sibling:**

Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

xpath=//\*[@type='submit']//following-sibling::input

### **f) Parent:**

Selects the parent of the current node as shown in the below screen.

Xpath=//\*[@id='rt-feature']//parent::div

### **g) Self:**

One node matching by using “self ” axis. It always finds only one node as it represents self-element.

Xpath =//\*[@type='password']//self::input

### **h) Descendant:**

Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element ( ‘Main body surround’ frame element) which means down under the node (child node , grandchild node, etc.).

Xpath=//\*[@id='rt-feature']//descendant::a