Selenium IDE

Selenium Integrated Development Environment

Objective:

-> It is used to automate the testing with record and playback feature
->It records actions like clicks, text inputs, and navigation steps, which can then be played back to simulate the same sequence.

Environment Setup:

1> Programming Langauge

-> C#, Java, JavaScript, Python, and Ruby

2> Browser

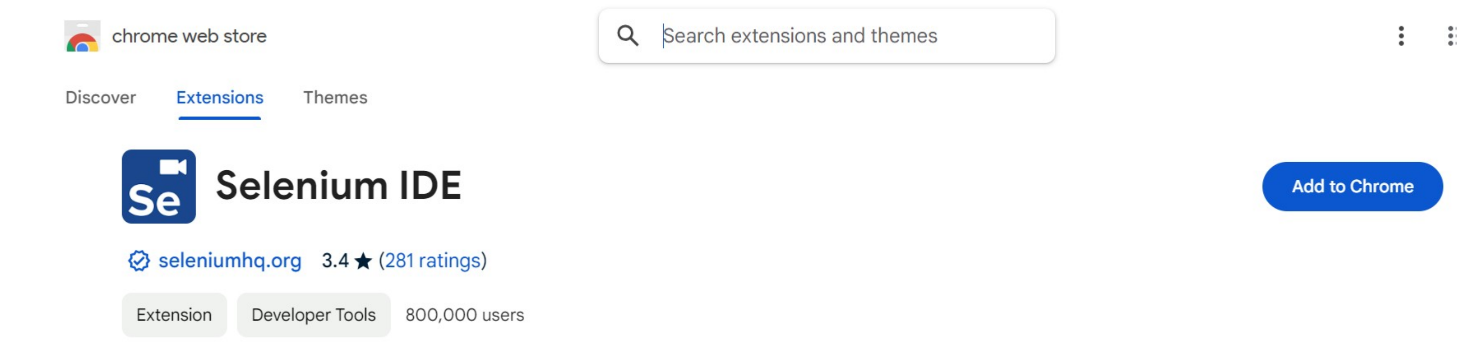
->Chrome, Firefox

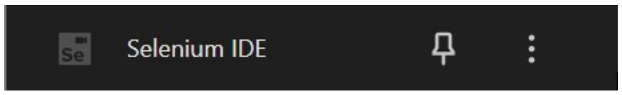
3> Operating System

-> Windows, MacOS

Installation:

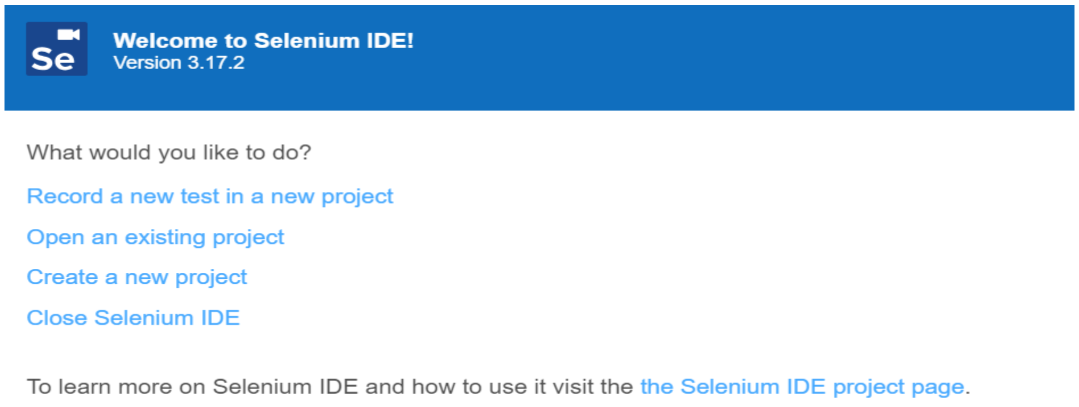
->Go to the Chrome web store (https://chromewebstore.google.com/)
->Search Selenium ide extention
->Click on Add to Chrome
→Add to Extension





Go to extension and Click on this it open the selenium ide

Open Selenium Ide from extension


Welcome to Selenium IDE!
Version 3.17.2

What would you like to do?

Record a new test in a new project
Open an existing project
Create a new project
Close Selenium IDE

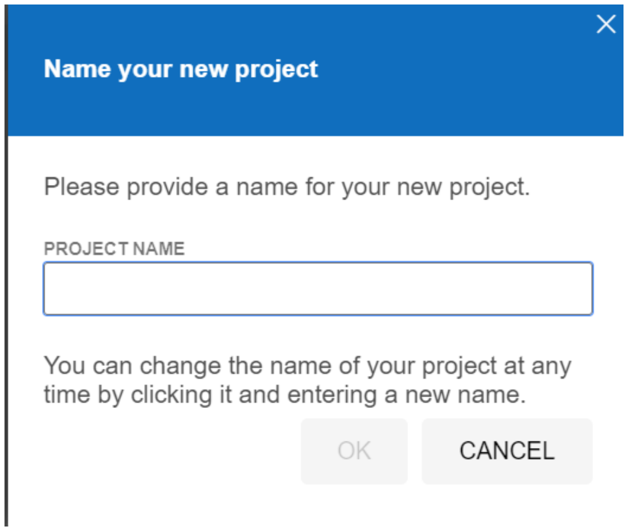
To learn more on Selenium IDE and how to use it visit the the Selenium IDE project page.

There are three option to open

Record a new tests in a new project : Create a new project and immediately start recording a new test.

Create a new project : Create a project without immediately recording a test.

Open Existing project : To open the project that you already make.
->Name it as your project name


Name your new project

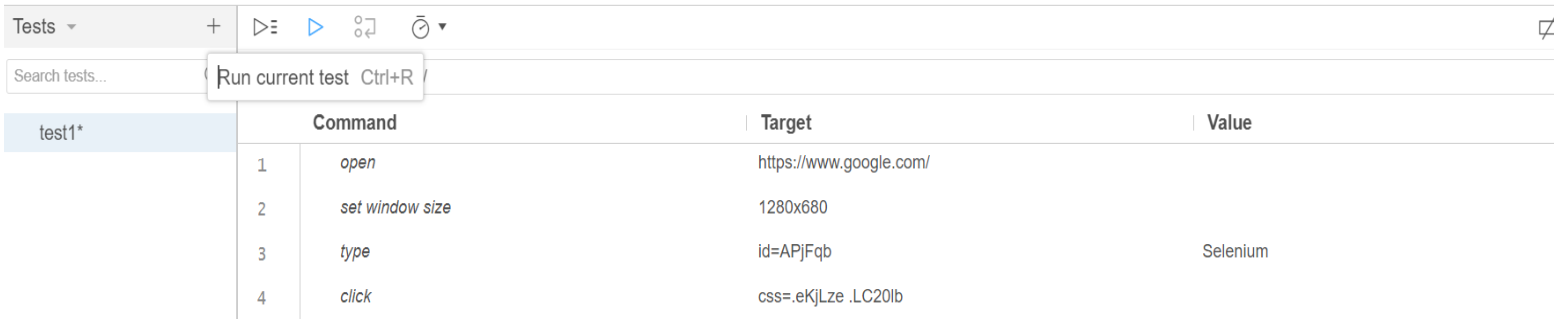
Please provide a name for your new project.

PROJECT NAME

You can change the name of your project at any time by clicking it and entering a new name.

→After this insert base url

Record feature : automatically captures and logs user actions on a web application
-> Enter the base url that you want test
-> Click on record button and start recording
-> Do your test like login,selecting,input etc .
-> After doing this stop the test
for example if we go the url https://www.google.com
In the search type something example selenium and search after this stop



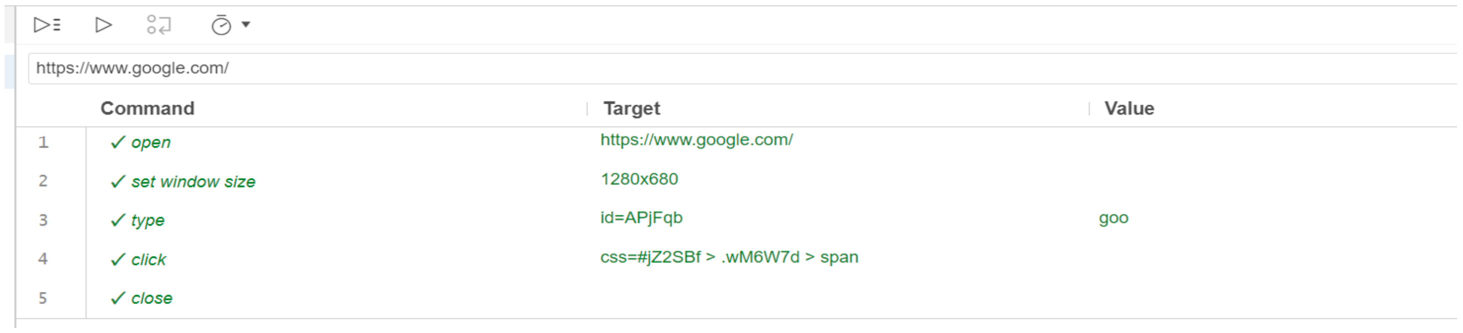
	Command	Target	Value
1	open	https://www.google.com/	
2	set window size	1280x680	
3	type	id=APjFqb	Selenium
4	click	css=.eKjLze .LC20lb	

Run the test

-> after successfully recording , run the test there are two option one is run all tset and other is run current test
-> Run current test run the selected test
-> after run it show pass or fail status

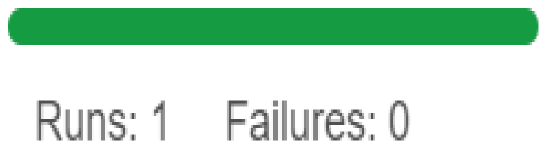
How to Handle Assertions and Validations in Recordings?

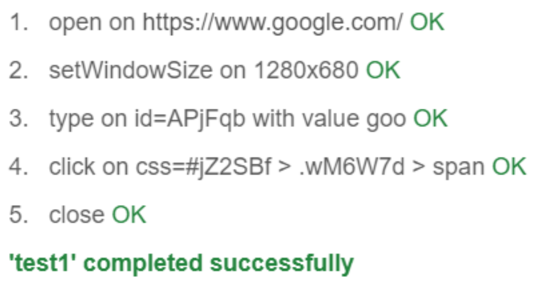
Assertions can be added during the recording process or afterward by editing the test case. Assertions validate if certain conditions are met (like checking if an element exists or if the page displays the expected text). You can insert an assertion by right-clicking on a page element during recording and selecting the "assert" or "verify" option.


https://www.google.com/

	Command	Target	Value
1	✓ open	https://www.google.com/	
2	✓ set window size	1280x680	
3	✓ type	id=APjFqb	goo
4	✓ click	css=#jZ2SBf > .wM6W7d > span	
5	✓ close		

You got successful or failed message or log

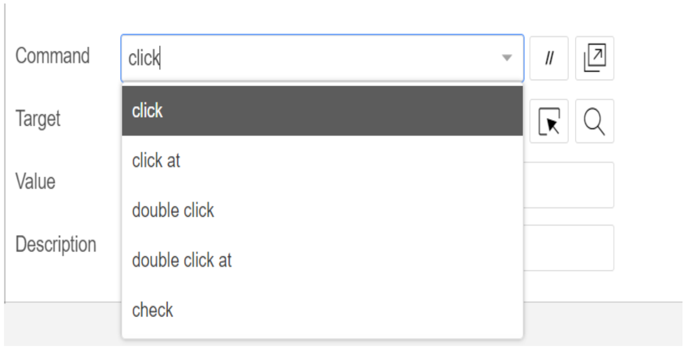


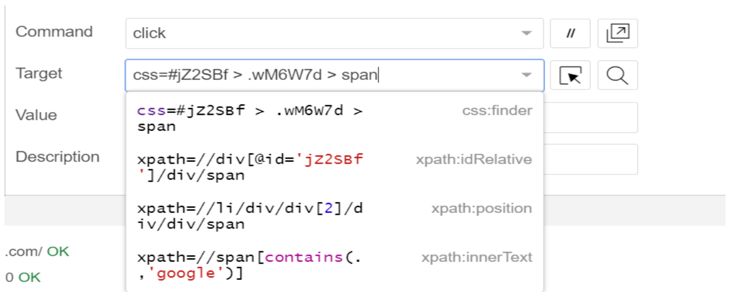

```
1. open on https://www.google.com/ OK
2. setWindowSize on 1280x680 OK
3. type on id=APjFqb with value goo OK
4. click on css=#jZ2SBf > .wM6W7d > span OK
5. close OK
'test1' completed successfully
```

4> Edit the test

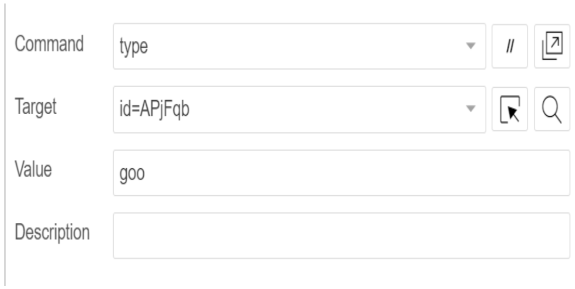
-> we can edit test by using command ,target and value

->add assertion to check like the text found or not etc.,
-> after sometime something is change in the website so we can edit and manage and run as expected

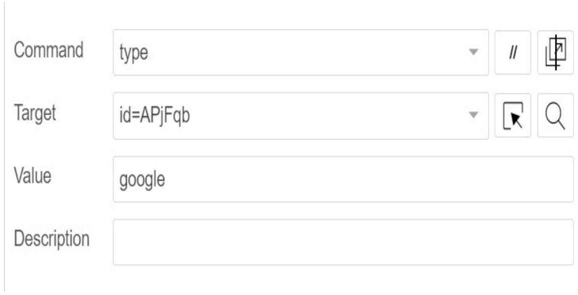

Command: click
Target:
Value:
Description:


Command: click
Target: css=#jZ2SBf > .wM6W7d > span
Value:
Description:

```
css=#jZ2SBf > .wM6W7d > span                  css:finder
xpath=//div[@id='jZ2SBf']/div/span            xpath:idRelative
xpath=//li/div/div[2]/div/div/span            xpath:position
xpath=//span[contains(.,'google')]            xpath:innerText
```

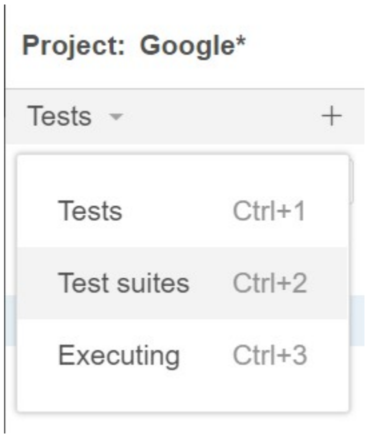

Command: type
Target: id=APjFqb
Value: goo
Description:

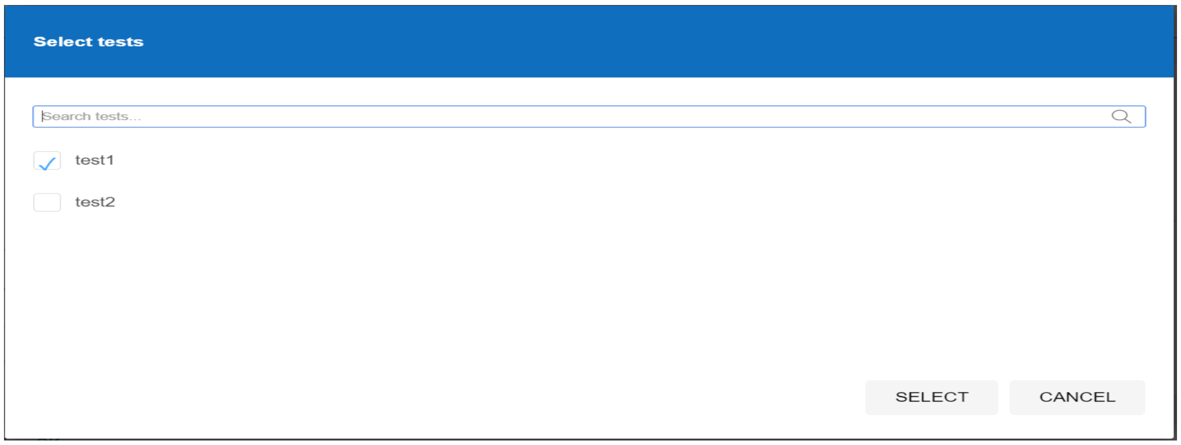
---->


Command: type
Target: id=APjFqb
Value: google
Description:

Manage Test

1> Create a test suite
2> add test in suite




Select tests

Search tests...

[✓] test1
[ ] test2

3> so that it can be managebale and simply it make in group of test and run all the test in a suite
4> Assert or verify
 -> for example if assertTitle it verify if it is true for the title
  -> continue if the condition is true
5> verification
  ->it check the condition but continue if the condition is false
  ->example verifyText

Commands : Commands are used to automate interactions with web elements on a web page .

Commands can be categorized into three types

Action:
Interact with the web page or modify its state (e.g., clicking a button, entering text).

Accessor:
Used to retrieve data from the web page, such as text or element attributes .

Assertion:
Used to verify that a certain condition is true. If an assertion fails, the test will stop

Action

`click`	Clicks on a specified element (button, link, etc.).
`doubleClick`	Double-clicks on a specified element.
`type`	Types text into a text field.
`select`	Selects an option from a dropdown.
`open`	Opens a specific URL in the browser.
`submit`	Submits a form.
`mouseOver`	Moves the mouse cursor over an element.
`mouseDown`	Simulates pressing the mouse button on an element.
`mouseUp`	Simulates releasing the mouse button on an element.
`dragAndDropToObject`	Drags an element to another target element.

`setWindowSize`	Resizes the browser window to a specified width/height.
`close`	Closes the current browser window.
`refresh`	Refreshes the current page.
`selectFrame`	Selects a frame/iframe by name, index, or locator.

`selectWindow`
Selects a browser window using its name or handle.

Access

`storeTitle`	Retrieves the current page title and stores it.
`storeText`	Retrieves the text of a specified element and stores it.
`storeValue`	Retrieves the value of an input field and stores it.
`storeAttribute`	Retrieves an element's attribute and stores it.
`storeCurrentUrl`	Stores the current URL of the page.
`storeWindowHandle`	Stores the handle of the current window.
`storeElementCount`	Stores the number of elements that match the locator.

Assertion

`assertTitle`	Asserts that the page title matches the expected value.
`assertText`	Asserts that an element's text matches the expected value.
`assertValue`	Asserts that an input field's value matches the expected value.
`assertElementPresent`	Asserts that an element is present on the page.
`assertElementNotPresent`	Asserts that an element is not present on the page.
`assertVisible`	Asserts that an element is visible.
`assertNotVisible`	Asserts that an element is not visible.
`assertAttribute`	Asserts that an element's attribute matches the expected value.
`verifyTitle`	Verifies that the page title matches the expected value (test continues).
`verifyText`	Verifies that an element's text matches the expected value.
`verifyValue`	Verifies that an input field's value matches the expected value.
`verifyElementPresent`	Verifies that an element is present on the page.
`verifyNotVisible`	Verifies that an element is not visible.

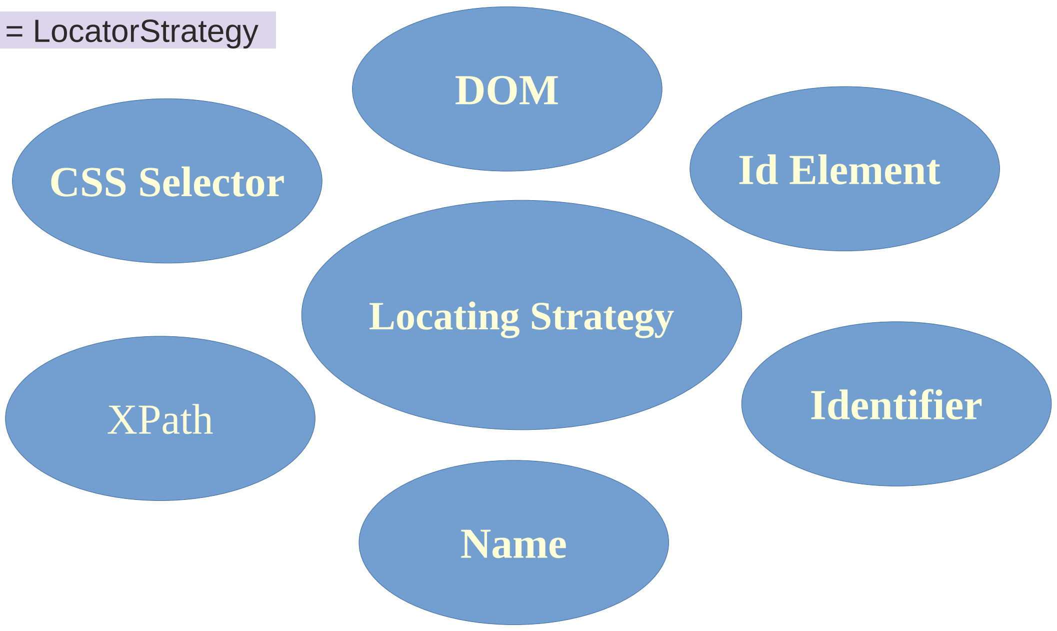
Other

`pause`	Pauses the test execution for a specified amount of time.
`waitForElementPresent`	Waits until a specified element appears on the page.
`waitForElementVisible`	Waits until a specified element becomes visible on the page.
`waitForText`	Waits until a specified text appears on the page.
`waitForValue`	Waits until an input field has a specific value.
`executeScript`	Executes a custom JavaScript script on the page.

Target :

A target consists of a locating strategy and has a format like:

LocatorType = LocatorStrategy



ID :

The `id` locator selects elements by their unique `id` attribute

Command: click

Target: id=loginButton

Name:

The `name` locator selects elements by their `name` attribute

Command: type

Target: name=username

Value: testuser

CSS Selector:

CSS selectors allow selecting elements using CSS rules. This is very powerful and flexible for selecting elements based on attributes, classes, hierarchy, etc.

Command: click

Target: css=input[type='submit']

XPath

XPath is an XML path language used to navigate through elements and attributes in an HTML document. It's useful for selecting elements that don't have unique attributes.

1. Command: click

Target: xpath=//input[@id='username']

2. Command: click

Target: xpath=//div[@class='login']//button[@type='submit']

Link Text

This locator selects anchor (`<a>`) elements based on the exact text within the link.

Command: click

Target: linkText=Login

Partial Link Text Locator

This locator selects anchor (`<a>`) elements by matching a partial text within the link. It's useful when the text is dynamic or long.

Command: click

Target: partialLinkText=Sign

DOM

This uses the Document Object Model (DOM) structure to locate elements. You can use JavaScript-style notation to interact with elements.

Command: click

Target: dom=document.forms[0].username

Class Name (class)

The `class` locator selects elements by their `class` attribute. It works well when an element has a unique class or when you want to target multiple elements with the same class.

Command: click

Target: class=submit-button

Tag Name Locator (`tagName`)

This locator selects elements by their HTML tag. It's typically used when you want to select an element based on the tag alone.

Command: click

Target: tagName=button

Attribute-Based Locator (`css` or `xpath`)

You can also target elements based on various attributes like `type`, `value`, etc., using CSS selectors or XPath.

Command: click

Target: css=input[type='text']

Custom Attribute Locator (CSS Selector or XPath)

Custom attributes can also be selected using CSS selectors or Xpath.

CSS Example:

```
Command: click
Target: css=[data-test='submitButton']
```

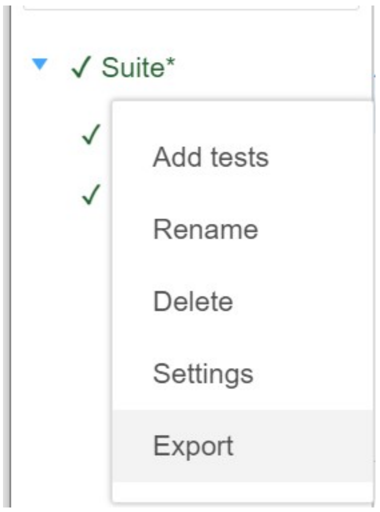
`XPath Example`:

```
Command: click
Target: xpath=//button[@data-test='submitButton']
```

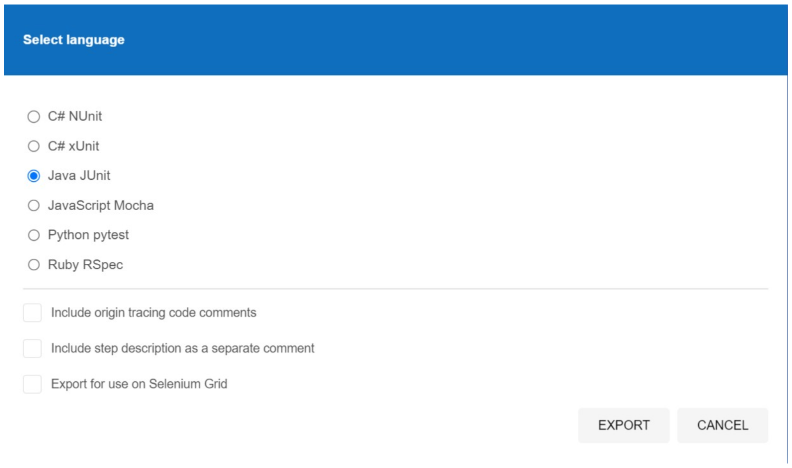
Locator	Format	Example
ID	`id=elementID`	
Name	`name=elementName`	`name=username`
CSS Selector	`css=cssExpression`	`css=input[type='submit']`
XPath	`xpath=xpathExpression`	`xpath=//input[@id='password']`
Link Text	`linkText=exactText`	`linkText=Sign In`
Partial Link Text	`partialLinkText=partialText`	`partialLinkText=Sign`
DOM	`dom=document.forms[0].elementName`	`dom=document.forms[0].username`
Class Name	`class=className`	`class=submit-button`
Tag Name	`tagName=elementTag`	`tagName=button`
Custom Attribute (CSS)	`css=[attribute=value]`	`css=[data-test='submitButton']`
Custom Attribute (XPath)	`xpath=//tag[@attribute='value']`	`xpath=//button[@data-test='submitButton']`

Export

-> export the test for customisation and manageable
-> export in different programming langauge and framework (nunit)



Export different programming langauge

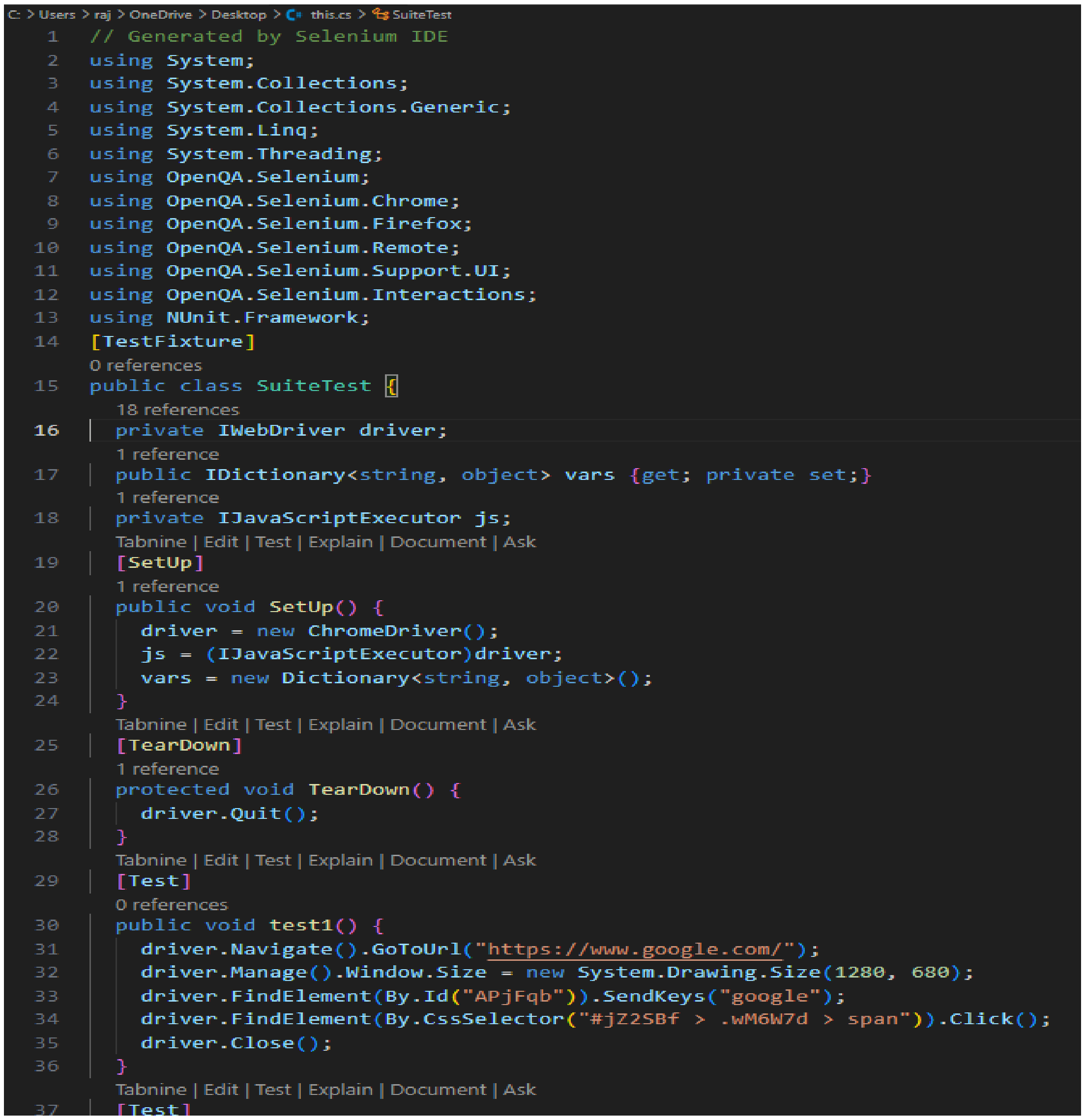


Select language

- ○ C# NUnit
- ○ C# xUnit
- [✓ Java JUnit]
- ○ JavaScript Mocha
- ○ Python pytest
- ○ Ruby RSpec

- ☐ Include origin tracing code comments
- ☐ Include step description as a separate comment
- ☐ Export for use on Selenium Grid

Export Selenium IDE Test to C#



```csharp
// Generated by Selenium IDE
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Remote;
using OpenQA.Selenium.Support.UI;
using OpenQA.Selenium.Interactions;
using NUnit.Framework;
[TestFixture]
0 references
public class SuiteTest {
  18 references
  private IWebDriver driver;
  1 reference
  public IDictionary<string, object> vars {get; private set;}
  1 reference
  private IJavaScriptExecutor js;
  Tabnine | Edit | Test | Explain | Document | Ask
  [SetUp]
  1 reference
  public void SetUp() {
    driver = new ChromeDriver();
    js = (IJavaScriptExecutor)driver;
    vars = new Dictionary<string, object>();
  }
  Tabnine | Edit | Test | Explain | Document | Ask
  [TearDown]
  1 reference
  protected void TearDown() {
    driver.Quit();
  }
  Tabnine | Edit | Test | Explain | Document | Ask
  [Test]
  0 references
  public void test1() {
    driver.Navigate().GoToUrl("https://www.google.com/");
    driver.Manage().Window.Size = new System.Drawing.Size(1280, 680);
    driver.FindElement(By.Id("APjFqb")).SendKeys("google");
    driver.FindElement(By.CssSelector("#jZ2SBf > .wM6W7d > span")).Click();
    driver.Close();
  }
  Tabnine | Edit | Test | Explain | Document | Ask
  [Test]
```

1> First export the test or projet in c# (NUnit)
2> Set Up C# Project with Selenium WebDriver

-> Go to Visual Studion
->serch Nunit Test Project
-> Name the test project
-> Install necessary packages
 * Selenium.WebDriver
 * Selenium.WebDriver.ChromeDriver
 * NUnit (if not already included)
-> Add using in .cs file
 * using NUnit.Framework;
 * using OpenQA.Selenium;
 * using OpenQA.Selenium.Chrome;
 * using OpenQA.Selenium.Firefox;

3> After this open the exported .cs file from selenium

->Right click on project
->Add
->Existing project
->Open the exported file
->Build the solution
->Open the test explorer in test
->Select the that you want to run
->Run the test

4> You can edit