SeleniumHQ
Browser Automation

Table of Contents

Introduction

Software applications are written as web-based applications to be run in an Internet browser. The effectiveness of testing these applications varies widely among companies and organizations.
Test automation is frequently becoming a requirement for software projects.
Test automation means using a software tool to run repeatable tests against the application to be tested. For regression testing this provides that responsiveness.
There are many advantages to test automation. Most are related to the repeatability of the tests and the speed at which the tests can be executed.
Selenium is possibly the most widely-used open source solution.

Test automation supports:

- Frequent regression testing
- Rapid feedback to developers
- Virtually unlimited iterations of test case execution
- Support for Agile and extreme development methodologies
- Disciplined documentation of test cases
- Customized defect reporting
- Finding defects missed by manual testing

Selenium IDE

Selenium IDE is Selenium Record and Playback tool.
It is designed to record your interactions with websites to help you generate and maintain site automation.
Features include:

- Recording and playing back tests on Firefox and Chrome.
- Organizing tests into suites for easy management.
- Saving and loading scripts, for later playback.
- Support for Selenium 3.

Introducing WebDriver

The primary new feature in Selenium 2.0 is the integration of the WebDriver API. WebDriver is designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API. Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

WebDriver and the Selenium-Server

You may, or may not, need the Selenium Server, depending on how you intend to use Selenium-WebDriver. If your browser and tests will all run on the same machine, and your tests only use the WebDriver API, then you do not need to run the Selenium-Server; WebDriver will run the browser directly.

There are some reasons though to use the Selenium-Server with Selenium-WebDriver.

- You are using Selenium-Grid to distribute your tests over multiple machines or virtual machines (VMs).
- You want to connect to a remote machine that has a particular browser version that is not on your current machine.
- You are not using the Java bindings (i.e. Python, C#, or Ruby) and would like to use HtmlUnit Driver

WebDriver Architecture

Selenium WebDriver works using client server communication. When Selenium test is executed, a new session of the browser is created, and the browser window is launched. For each command in test script, request is sent to the WebDriver API which is REST base service. The WebDriver API Interprets the request and then step is executed in the browser. Which access the server and just wait for the request to come in, once each step is complete the response is sent back to the WebDriver API and this process is continues all steps are complete

Language Supported – C#, Java, Ruby, Python, JavaScript

Platform Supported – macOS, Windows and Linux

Browser Supported – Chrome, Firefox, IE, Edge and Safari

Browser Specific Driver

Each Browser has their own browser driver which is maintained by the browser vendor. All the drivers where written in the same language as browser.

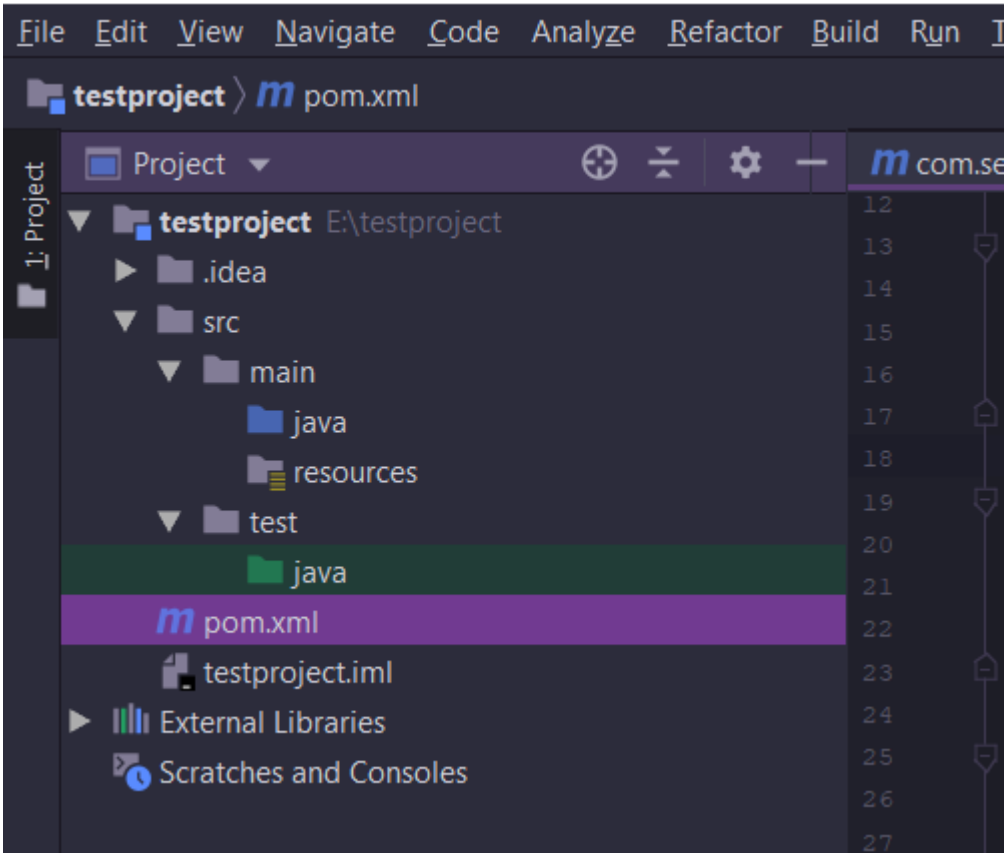
Setting Up a Selenium-WebDriver Project

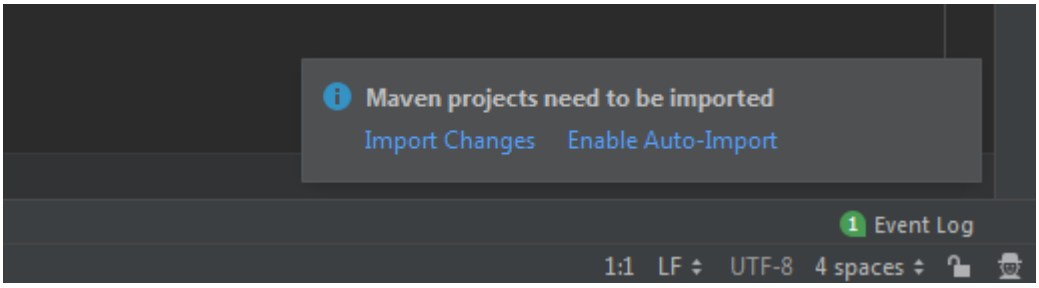
To install Selenium means to set up a project in a development so you can write a program using Selenium. How you do this depends on your programming language and your development environment.

JAVA

The easiest way to set up a Selenium 2.0 Java project is to use Maven. Maven will download the java bindings (the Selenium 2.0 java client library) and all its dependencies, and will create the project for you, using a maven pom.xml (project configuration) file. Once you've done this, you can import the maven project into your preferred IDE, IntelliJ IDEA or Eclipse.

Create Maven Project





Selenium Maven Dependency

Default Maven pom.xml

Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.selenium</groupId>
    <artifactId>com.selenium.automation</artifactId>
    <version>1.0-SNAPSHOT</version>

</project>
```

Pom.xml after adding dependencies

Before Dependencies library download

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.firstproject</groupId>
    <artifactId>com.firstproject.selenium</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-api -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-api</artifactId>
            <version>3.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-server -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-server</artifactId>
            <version>3.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-support -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-support</artifactId>
            <version>3.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-chrome-driver -
->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-chrome-driver</artifactId>
            <version>3.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-remote-driver -
->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-remote-driver</artifactId>
            <version>3.14.0</version>
        </dependency>
    </dependencies>

</project>
```

Auto Resolving Dependencies

```
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-remote-driver -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-remote-driver</artifactId>
            <version>3.14.0</version>
        </dependency>
    </dependencies>

</project>
project > dependencies > dependency
```

Background Tasks

Resolving Maven dependencies...

Resolving dependencies of com.firstproject.selenium...

selenium-chrome-driver-3.14.0.pom [https://repo.maven.apache.or...

1 process running... 47:22 LF UTF-8 4 spaces

Connect to proxy server network

- Create settings.xml
- Add proxy, host: IP address, port: number
- Now we can see below message with progress bar. Wait till complete resolving dependencies.

After Resolving Dependencies

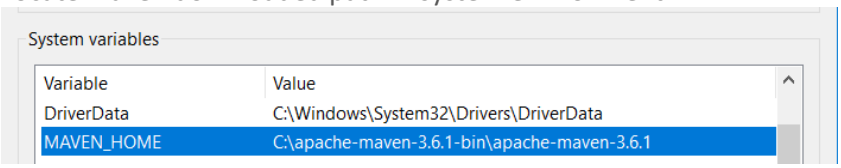
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.selenium</groupId>
    <artifactId>com.selenium.automation</artifactId>
    <version>1.0-SNAPSHOT</version>

<dependencies>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.14.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-api -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-api</artifactId>
        <version>3.14.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-server -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-server</artifactId>
        <version>3.14.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-support -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-support</artifactId>
        <version>3.14.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-chrome-driver -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-chrome-driver</artifactId>
        <version>3.14.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-remote-driver -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-remote-driver</artifactId>
        <version>3.14.0</version>
    </dependency>
</dependencies>
</project>
```

Dependency downloaded location

C:\Users\UserName\.m2\repository\org\seleniumhq\selenium

Maven Download Add MAVEN_HOME to System Environment

Download Maven and extract the zip file [apache-maven-3.6.1-bin.zip]
Locate maven downloaded path in system environment.



Now, from a command-line, CD into the project directory and run maven as follows.
mvn clean install

```
Terminal:  Local ×  +
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\testproject>mvn clean install
```

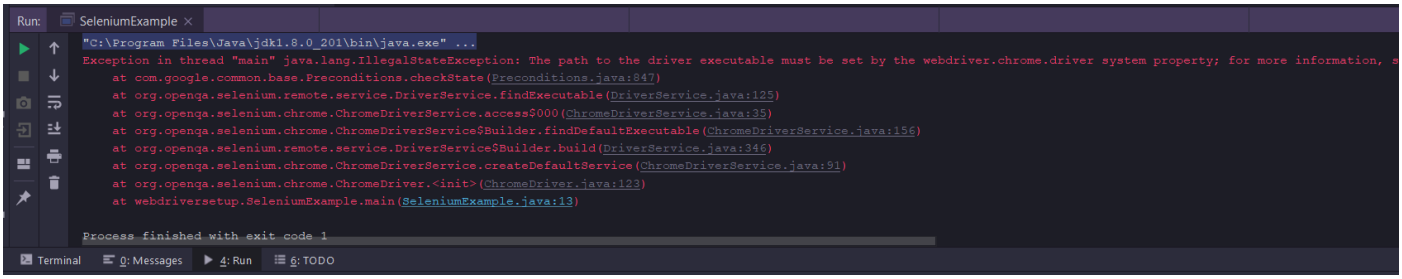
```
Terminal:  Local ×  +
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.173 s
[INFO] Finished at: 2019-07-19T01:40:44+05:30
[INFO] ------------------------------------------------------------------------
```

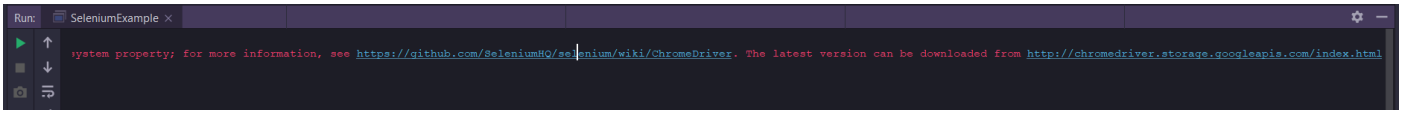
Selenium Example Class

```java
package webdriversetup;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;

public class SeleniumExample {
    public static void main(String[] args) {
        // Create a new instance of the Chrome driver
        WebDriver driver = new ChromeDriver();

        // And now use this to visit Google
        driver.get("https://www.google.com");
        // Alternatively the same thing can be done like this
        // driver.navigate().to("https://www.google.com");

        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));

        // Enter something to search for
        element.sendKeys("Cheese!");

        // Now submit the form. WebDriver will find the form for us from the element
        element.submit();

        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());

        // Google's search is rendered dynamically with JavaScript.
        // Wait for the page to load, timeout after 10 seconds
        (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
            public Boolean apply(WebDriver d) {
                return d.getTitle().toLowerCase().startsWith("cheese!");
```

```
            }
        });

        // Should see: "cheese! - Google Search"
        System.out.println("Page title is: " + driver.getTitle());

        //Close the browser
        driver.quit();
    }
}
```

Output





Error message

Exception in thread "main" java.lang.IllegalStateException: The path to the driver executable must be set by the webdriver.chrome.driver system property; for more information, see https://github.com/SeleniumHQ/selenium/wiki/ChromeDriver. The latest version can be downloaded from http://chromedriver.storage.googleapis.com/index.html

Troubleshoot Selenium Example Class

- Download driver from the link from console log
- Create a folder “driver” in the framework root
- Now drag and drop chromedriver.exe in to driver folder
- Add following line above creating driver instance.

```
• System.setProperty("webdriver.chrome.driver","E:\\testproject\\driver\\chromedriver.exe");
```

Find Element

By is a package org.openqa.selenium.By which is used to located element with specified selector.

Way to find an Element

- .By.ClassName
- .By.CssSelector
- .By.Id
- .By.Name
- .By.Xpath

Assign to WebElement

After WebElement is found it is assigned to WebElement called element. This invokes the package org.openqa.selenium.WebElement

Perform Actions

```
element.sendKeys("Cheese!");
element.submit();
```

Other Common Actions

- Click
- Drag and Drop
- Move to element

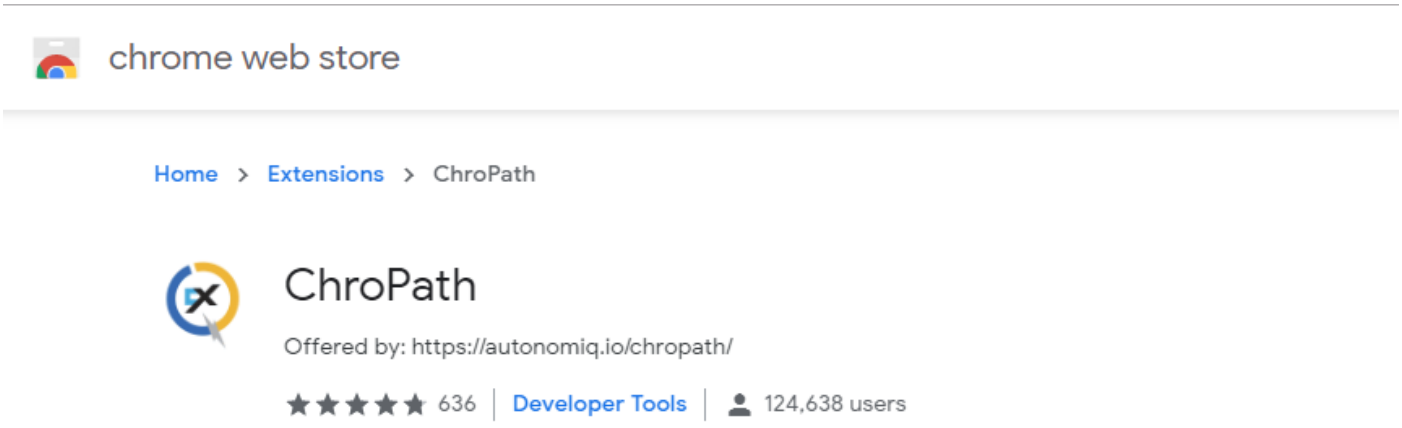
Quit the Driver

```
driver.quit();
```

Which quit driver and close windows

Inspecting Elements

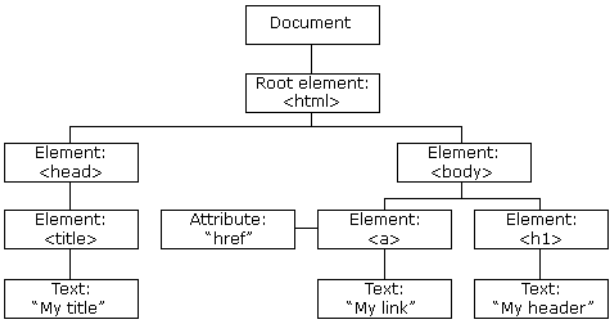
Used to identify web element selectors to use in tests.



The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.
The HTML DOM model is constructed as a tree of Objects:

The HTML DOM Tree of Objects



Katalon Recorder

chrome web store

Home > Extensions > Katalon Recorder

Katalon Recorder

Offered by: Katalon.com

161 | Developer Tools | 127,262 users

Runs offline

Add to Chrome

Selenium-WebDriver API Commands and Operations

Fetching a Page

The first thing you're likely to want to do with WebDriver is navigate to a page

```
driver.get("https://www.google.com");
```

Implicit and Explicit Waits

Waiting is having the automated task execution elapse a certain amount of time before continuing with the next step. You should choose to use Explicit Waits or Implicit Waits.
WARNING: Do not mix implicit and explicit waits! Doing so can cause unpredictable wait times?

Implicit wait is set for the entire duration of the webDriver object. Suppose, you want to wait for a certain duration, let's say 5 seconds before each element or a lot of elements on the webpage load. Now, you wouldn't want to write the same code again and again. Hence, implicit wait. However, if you want to wait for only one element, use explicit.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement"));
```

Explicit wait the code you define to wait for a certain condition to occur before proceeding further in the code. The worst case of this is Thread.sleep(), which sets the condition to an exact time period to wait.

```
WebDriver driver = new FirefoxDriver();
driver.get("http://somedomain/url that delays loading");
WebElement myDynamicElement = (new WebDriverWait(driver, 10))
  .until(ExpectedConditions.presenceOfElementLocated(By.id("myDynamicElement")));
```

Locating UI Elements (WebElements)

Locating elements in WebDriver can be done on the WebDriver instance itself or on a WebElement. Each of the language bindings exposes a "Find Element" and "Find Elements" method.

By ID

This is the most efficient and preferred way to locate an element.

Example

```
<div id="coolestWidgetEvah">...</div>
```

```
WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
```

By Class Name

"Class" in this case refers to the attribute on the DOM element. Often in practical use there are many DOM elements with the same class name, thus finding multiple elements becomes the more practical option over finding the first element.

Example

```
<div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></div>
```

```
List<WebElement> cheeses = driver.findElements(By.className("cheese"));
```

By Tag Name

The DOM Tag Name of the element.

Example

```
<iframe src="..."></iframe>
```

```
WebElement frame = driver.findElement(By.tagName("iframe"));
```

By Name

Find the input element with matching name attribute.

Example

```
<input name="cheese" type="text"/>
```

```
WebElement cheese = driver.findElement(By.name("cheese"));
```

By Link Text

Find the link element with matching visible text.

Example

```
<a href="http://www.google.com/search?q=cheese">cheese</a>>
```

```
WebElement cheese = driver.findElement(By.linkText("cheese"));
```

By Partial Link Text

Find the link element with partial matching visible text.

Example

```
<a href="http://www.google.com/search?q=cheese">search for cheese</a>>
```

```
WebElement cheese = driver.findElement(By.partialLinkText("cheese"));
```

By CSS

Like the name implies it is a locator strategy by css. Native browser support is used by default

```
<div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></div>
```

Example

```
WebElement cheese = driver.findElement(By.cssSelector("#food span.dairy.aged"));
```

By XPath

At a high level, WebDriver uses a browser's native XPath capabilities wherever possible. On those browsers that don't have native XPath support

Example

```
<input type="text" name="example" />
<INPUT type="text" name="other" />
```

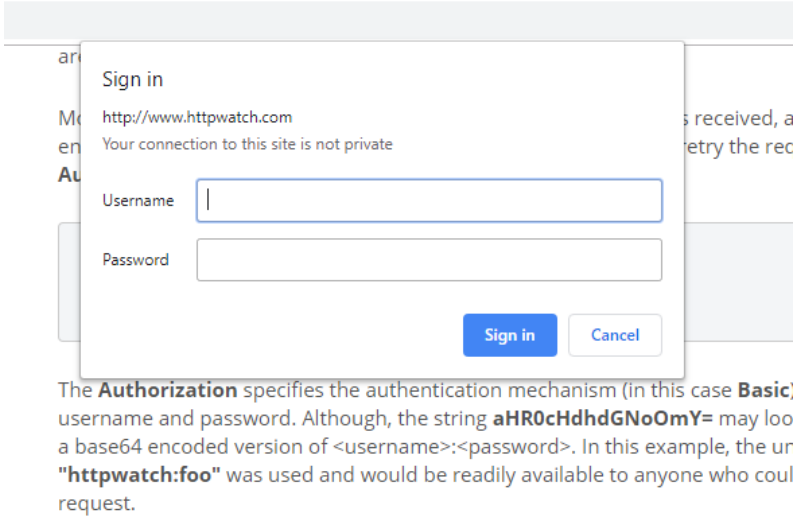
```
List<WebElement> inputs = driver.findElements(By.xpath("//input"));
```

Add two test scenarios

```
package katalon;


import java.util.concurrent.TimeUnit;

import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.*;
import static org.testng.Assert.*;
import org.openqa.selenium.*;

public class TestCase {
    private WebDriver driver;
    private String baseUrl;
    private boolean acceptNextAlert = true;
    private StringBuffer verificationErrors = new StringBuffer();

    @BeforeClass(alwaysRun = true)
    public void setUp() throws Exception {
       String sDir =  System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver", sDir + "/driver/chromedriver.exe");
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    }

    @Test
    public void seleniumTestCase() throws Exception {
    driver.get("https://www.seleniumhq.org/docs/01_introducing_selenium.jsp");
    driver.findElement(By.linkText("Browser Automation")).click();
    driver.findElement(By.linkText("Download")).click();
}

    @Test
    public void mvnTestCase() throws Exception {
        driver.get("https://mvnrepository.com/");
        driver.findElement(By.id("navigation")).click();
        driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-
space(.)='Popular'])[1]/preceding::b[1]")).click();
        driver.findElement(By.linkText("TestNG")).click();
    }

    @AfterClass(alwaysRun = true)
    public void tearDown() throws Exception {
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
        } catch (NoSuchElementException e) {
            return false;
        }
    }
```

```
    private boolean isAlertPresent() {
        try {
            driver.switchTo().alert();
            return true;
        } catch (NoAlertPresentException e) {
            return false;
        }
    }

    private String closeAlertAndGetItsText() {
        try {
            Alert alert = driver.switchTo().alert();
            String alertText = alert.getText();
            if (acceptNextAlert) {
                alert.accept();
            } else {
                alert.dismiss();
            }
            return alertText;
        } finally {
            acceptNextAlert = true;
        }
    }
}
```

Handling “Your connection to this site is not private” popup on Chrome using Selenium



Chrome no longer seems to support the ability to interact with the dialog box.

Syntax

```
driver.get("http://username:password@URL1");
```

TestNG DataProvider

A Data Provider is a method on your class that returns an array of array of objects. This method is annotated with `@DataProvider`:

Example1

```
//This method will provide data to any test method that declares that its Data Provider
//is named "test1"
    @DataProvider(name = "test1")
    public Object[][] createData1() {
        return new Object[][] {
                { "Apple", new Integer(220) },
                { "Orange", new Integer(100)},
        };
```

```
    }

    //This test method declares that its data should be supplied by the Data Provider
//named "test1"
    @Test(dataProvider = "test1")
    public void verifyData1(String n1, Integer n2) {
        System.out.println(n1 + " " + n2);
    }
```

Example2

```
//This method will provide data to any test method that declares that its Data Provider
//is named "test1"
    @DataProvider(name = "test1")
    public Object[][] createData1() {
        return new Object[][] {
                { "Apple", new Integer(220) },
                { "Orange", new Integer(100)},
        };
    }

    //This test method declares that its data should be supplied by the Data Provider
//named "test1"
    @Test(dataProvider = "test1")
    public void testUntitledTestCase(String searchText, Integer n2) throws Exception {
        driver.get("https://www.google.com/");
        driver.findElement(By.name("q")).clear();
        driver.findElement(By.name("q")).sendKeys(searchText);
        driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
    }
```

Assert – actual vs expected result

Example

```
//This method will provide data to any test method that declares that its Data Provider
//is named "test1"
    @DataProvider(name = "test1")
    public Object[][] createData1() {
        return new Object[][] {
                { "mango", "Images for mango" },
                { "Orange", "Images for Orange"},
        };
    }

    //This test method declares that its data should be supplied by the Data Provider
//named "test1"
    @Test(dataProvider = "test1")
    public void testUntitledTestCase(String searchText, String expectedText) throws Exception {
        driver.get("https://www.google.com/");
        driver.findElement(By.name("q")).clear();
        driver.findElement(By.name("q")).sendKeys(searchText);
        driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
        String actualText = driver.findElement(By.className("iu-card-header")).getText();
        Assert.assertEquals(actualText,expectedText,"Search result displayed as expected");
    }
```

Parameters from testng.xml

Testng.xml

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name="Suite1" verbose="1" >
    <parameter name="search" value="orange"></parameter>
    <test name="Regression1"   >
        <packages>
            <package name="katalon" />
        </packages>
    </test>
</suite>
```

Example

```
package katalon;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.*;
import static org.testng.Assert.*;
import org.openqa.selenium.*;


public class TestCase {
    private WebDriver driver;
    private StringBuffer verificationErrors = new StringBuffer();
    @BeforeClass(alwaysRun = true)
    public void setUp() throws Exception {
       String sDir =  System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver", sDir + "/driver/chromedriver.exe");
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();
    }
    @Parameters({"search"})
    @Test
    public void testUntitledTestCase(String searchText) throws Exception {
        driver.get("https://www.google.com/");
        driver.findElement(By.name("q")).clear();
        driver.findElement(By.name("q")).sendKeys(searchText);
        driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
    }

    @AfterClass(alwaysRun = true)
    public void tearDown() throws Exception {
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }
}
```

Read Test Data from Property file

Properties file will look like below

To create a properties file – filename.properties
Key and value

```
mango=mango
apple=apple
```

Example

```
package katalon;


import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.Properties;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.*;
import static org.testng.Assert.*;
import org.openqa.selenium.*;


public class TestCase {
    private WebDriver driver;
    private String baseUrl;
    private boolean acceptNextAlert = true;
    private StringBuffer verificationErrors = new StringBuffer();
    Properties properties = new Properties();
```

```
    @BeforeClass(alwaysRun = true)
    public void setUp() throws Exception {
       String sDir =  System.getProperty("user.dir");
        System.setProperty("webdriver.chrome.driver", sDir + "/driver/chromedriver.exe");
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        try{
            File configFile = new File(sDir+"/src/test/config.properties");
            InputStream stream = new FileInputStream(configFile);
            properties.load(stream);
        }catch (Exception e){
            System.out.println("e = " + e);}

    }

//    This method will provide data to any test method that declares that its Data Provider is
named "test1"
    @DataProvider(name = "test1")
    public Object[][] createData1() {
        return new Object[][] {
                { properties.getProperty("mango"), "Images for mango" },
                { properties.getProperty("apple"), "Images for Orange"},
        };
    }

    //This test method declares that its data should be supplied by the Data Provider
//named "test1"
    @Test(dataProvider = "test1")
    public void testUntitledTestCase(String searchText, String expectedText) throws Exception {
        driver.get("https://www.google.com/");
        driver.findElement(By.name("q")).clear();
        driver.findElement(By.name("q")).sendKeys(searchText);
        driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
        String actualText = driver.findElement(By.className("iu-card-header")).getText();
        Assert.assertEquals(actualText,expectedText,"Search result displayed as expected");
    }


    @AfterClass(alwaysRun = true)
    public void tearDown() throws Exception {
        driver.quit();
        String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }
}
```

Switch to frame with in the page – Example registration page has Live Chat

```
driver.switchTo().defaultContent(); // you are now outside both frames
 driver.switchTo().frame(driver.findElement(By.id("launcher")));
```

Switch to different window

```
    ArrayList<String> tabs2 = new ArrayList<String> (driver.getWindowHandles());
driver.switchTo().window(tabs2.get(1));
```

After assertion close the window and switch back to original window

```
driver.close();
        driver.switchTo().window(tabs2.get(0))
```

Switch to new tab

```
// Store the current window handle
String winHandleBefore = driver.getWindowHandle();
```

```
// Switch to new window opened
for(String winHandle : driver.getWindowHandles()){
        driver.switchTo().window(winHandle);

// Close the new window, if that window no more required
        driver.close();

// Switch back to original browser (first window)
        driver.switchTo().window(winHandleBefore);
```