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.

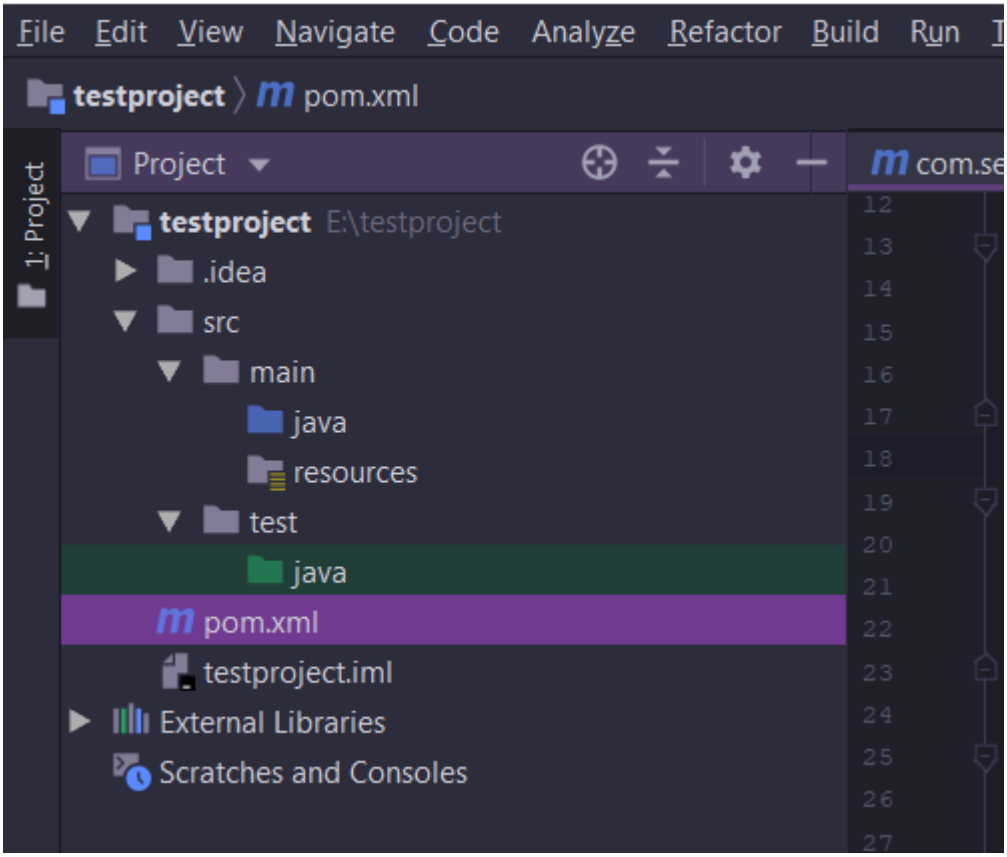
Setting Up a Selenium-WebDriver Project

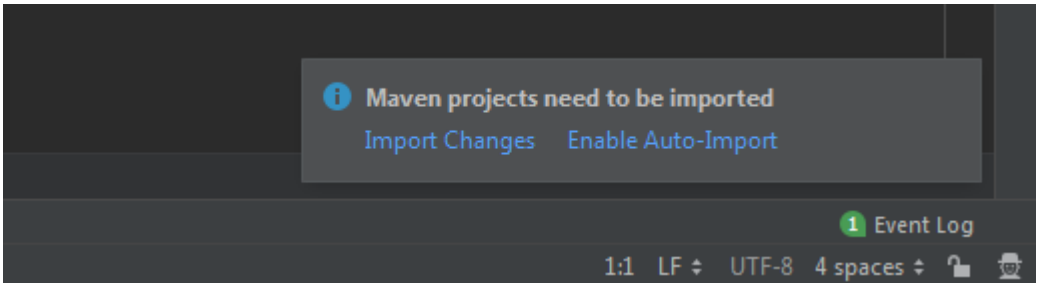
To install Selenium means to set up a project in a development so you can write a program using Selenium. How you do this depends on your programming language and your development environment.

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

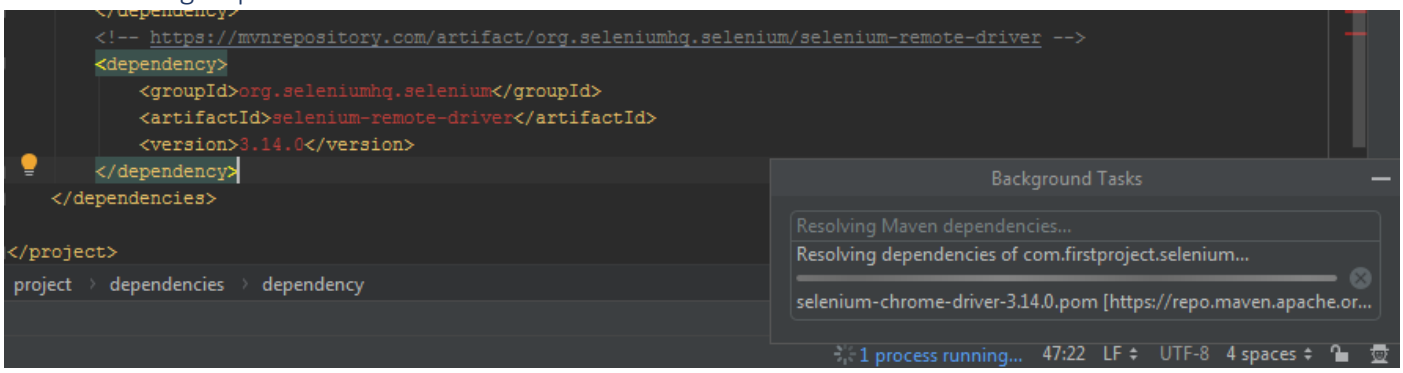
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <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



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

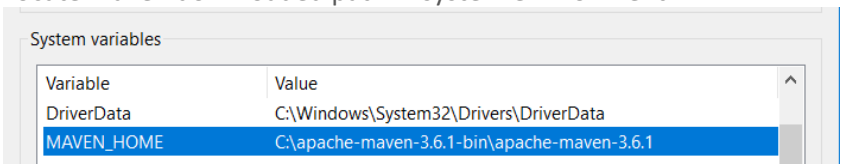
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <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

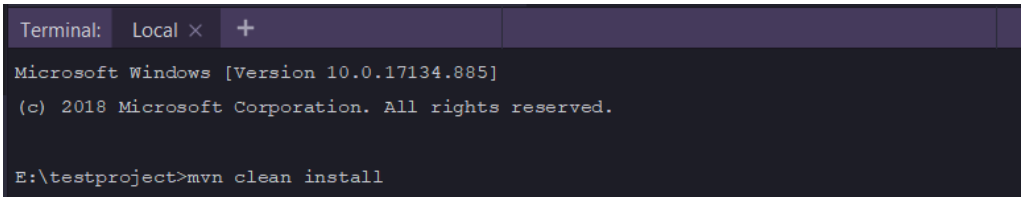
C:\Users\UserName\.m2\repository\org\seleniumhq\selenium

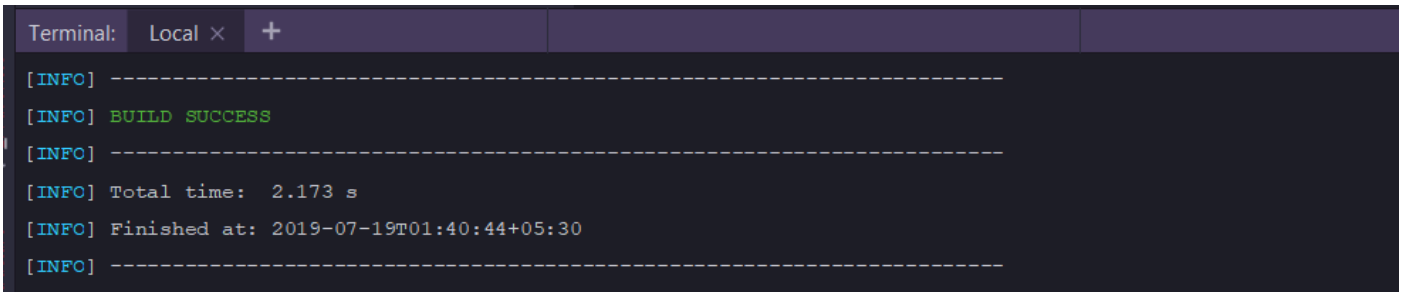
Maven Download Add MAVEN_HOME to System Environment

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



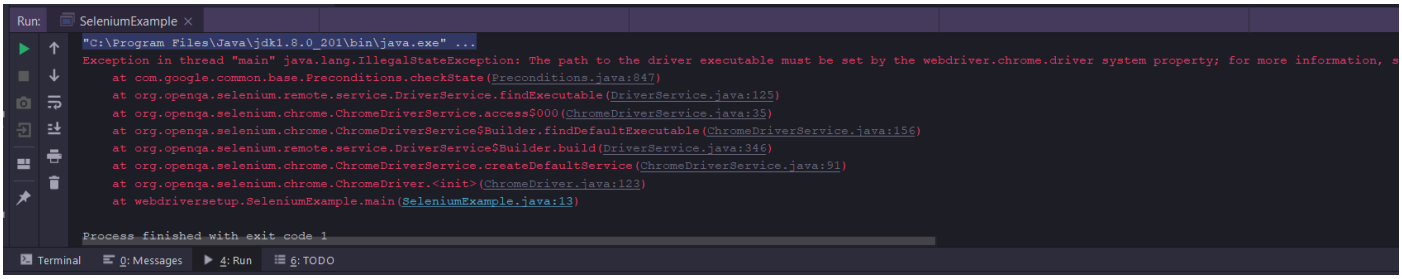


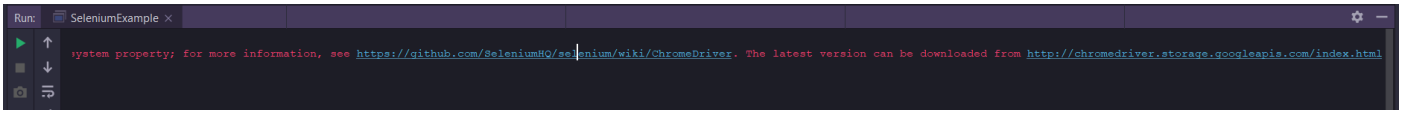
Selenium Example Class

```
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.