FREE eBook

# LEARNING selenium-webdriver

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#selenium-webdriver

# Table of Contents

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: selenium-webdriver

It is an unofficial and free selenium-webdriver ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official selenium-webdriver.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapter 1: Getting started with selenium-webdriver

## Remarks

This section provides an overview of what selenium-webdriver is, and why a developer might want to use it.

It should also mention any large subjects within selenium-webdriver, and link out to the related topics. Since the Documentation for selenium-webdriver is new, you may need to create initial versions of those related topics.

## Versions

| Version | Release Date |
| --- | --- |
| 0.0.1 | 2016-08-03 |

## Examples

### Installation or Setup

To begin using WebDriver you will need to obtain the relevant Driver from the Selenium site: Selenium HQ Downloads. From here you need to download the driver relevant to the browser(s) and/or platform(s) you are trying to run WebDriver on, e.g. if you were testing in Chrome the Selenium site will direct you to:

https://sites.google.com/a/chromium.org/chromedriver/

In order to download `chromedriver.exe`.

Finally, before being able to use WebDriver you will need to download the relevant language bindings, e.g. if using C# you can access the download from Selenium HQ Downloads page to obtain the required .dll files or, alternatively, download them as packages in Visual Studio via NuGet package manager.

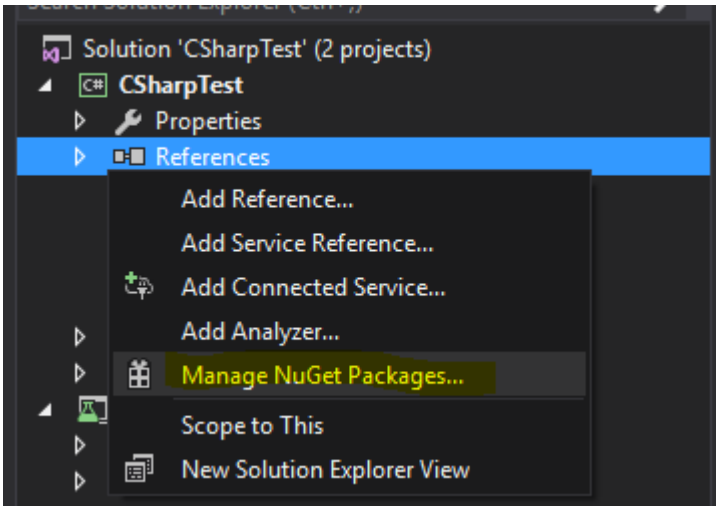
The required files should now be downloaded, for information on how to begin using WebDriver, refer to the other `selenium-webdriver` documentation.

---

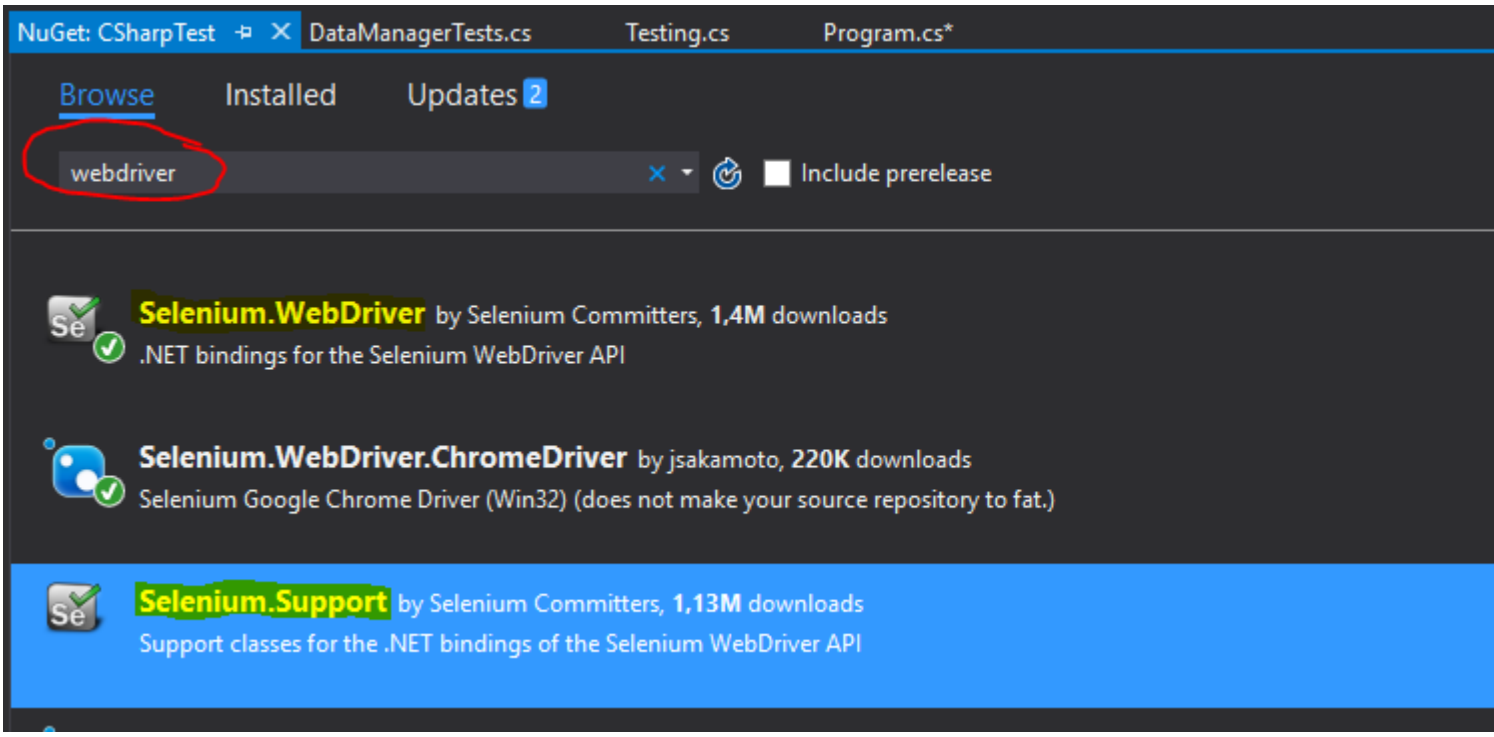
## For Visual Studio [NuGet]

The easiest way of installing Selenium WebDriver is by using a NuGet package manager.

In your project, right click "References", and click on "Manage NuGet Packages" as shown:




Then, type into the search box "*webdriver*". You should then see something like this:




Install "**Selenium.WebDriver**", and "**Selenium.Support**" (the Support package includes additional resources, such as Wait) by clicking on the Install button on the right side.

Then you can install your WebDrivers you wish to use, such as one of these:

- Selenium.WebDriver.ChromeDriver (Google Chrome)
- PhantomJS (headless)
-

## What is Selenium WebDriver ?

**Selenium** is a set of tools designed to automate browsers. It is commonly used for web

application tests across multiple platforms. There are a few tools available under the Selenium umbrella, such as Selenium WebDriver(ex-Selenium RC), Selenium IDE and Selenium Grid.

**WebDriver** is a remote control *interface* that enables you to manipulate DOM elements in web pages, as well as to command the behaviour of user agents. This interface provides a language-neutral wire protocol which has been implemented for various platforms such as:

- GeckoDriver (Mozilla Firefox)
- ChromeDriver (Google Chrome)
- SafariDriver (Apple Safari)
- InternetExplorerDriver (MS InternetExplorer)
- MicrosoftWebDriver, or EdgeDriver (MS Edge)
- OperaChromiumDriver (Opera browser)

as well as other implementations:

- EventFiringWebDriver
- HtmlUnitDriver
- PhantomJSDriver
- RemoteWebDriver

**Selenium WebDriver** is one of the Selenium tools which provides Object Oriented APIs in a variety of languages to allow for more control and the application of standard software development practices. To accurately simulate the way that a user will interact with a web application, it uses "Native OS Level Events" as oppose to "Synthesized JavaScript events".

**Links to refer:**

- http://www.seleniumhq.org/
- http://www.aosabook.org/en/selenium.html
- https://www.w3.org/TR/webdriver/

## Installation or setup for Java

In order to write tests using Selenium Webdriver and Java as programming language, you will need to download JAR files of Selenium Webdriver from the Selenium website.

There are multiple ways to setup a Java project for the Selenium webdriver, one of the easiest from all of them is using Maven. Maven downloads the required Java bindings for Selenium webdriver including all the dependencies. The other way is to download the JAR files and import them into your project.

**Steps to setup Selenium Webdriver project using Maven:**

1. Install maven on windows box following this document: https://maven.apache.org/install.html
2. Create a folder with name `selenium-learing`
3. Create a file into above folder using any text editor with name `pom.xml`
4. Copy below content to `pom.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>SeleniumLearning</groupId>
            <artifactId>SeleniumLearning</artifactId>
            <version>1.0</version>
            <dependencies>
                <dependency>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-learning</artifactId>
                    <version>3.0.0-beta1</version>
                </dependency>
            </dependencies>
    </project>
```

**Note**: Make sure that the version which you specified above is the latest one.You can check the latest version from here : http://docs.seleniumhq.org/download/maven.jsp

5. Using command line, run below command into the project directory.

   ```
   mvn clean install
   ```

Above command will download all the required dependencies and will add then into the project.

6. Write below command to generate an eclipse project which you can import to the Eclipse IDE.

   ```
   mvn eclipse:eclipse
   ```

7. To import the project into eclipse ide, you can follow below steps

   Open Elipse -> File -> Import -> General -> Existing Project into Workspace -> Next -> Browse -> Locate the folder contain pom.xml -> Ok -> Finish

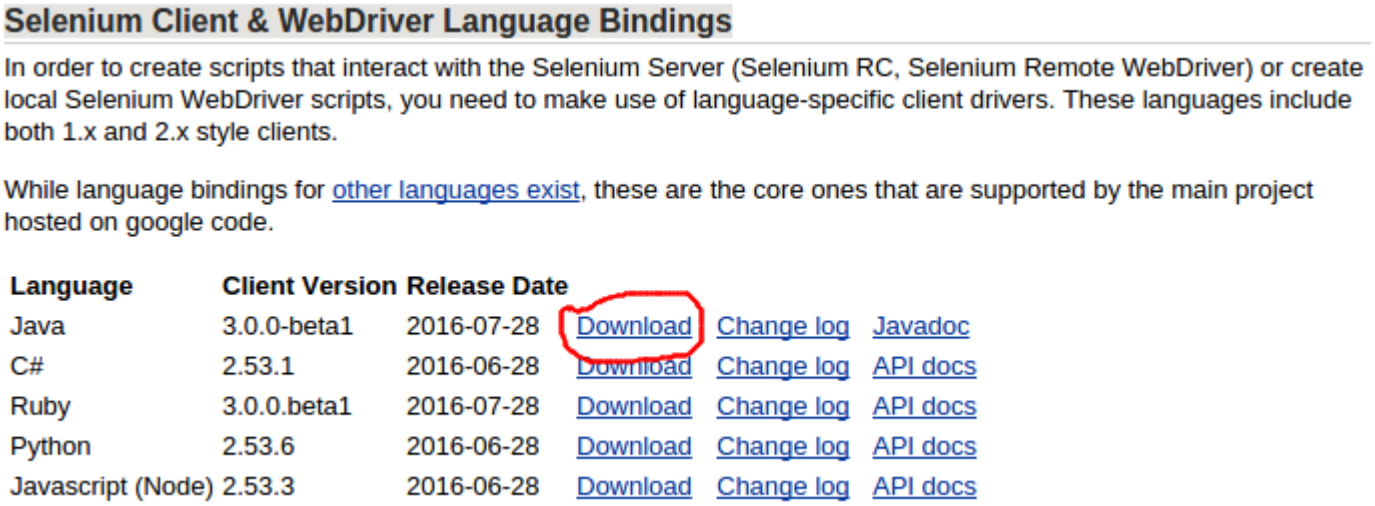
Install the m2eclipse plugin by right clicking on your project and select Maven -> Enable Dependency Management.

**Steps to setup Selenium Webdriver project using Jar files**

1. Create a new project in Eclipse following below steps.

   Open Elipse -> File -> New -> Java Project -> Provide a name (selenium-learning) -> Finish

2. Download jar files from http://www.seleniumhq.org/download/. You need to download both **Selenium Standalone Server** and **Selenium Client & WebDriver Language Bindings**. Since this document is talking about Java so you need to download only jar from Java section. Have a look in attached screenshot.



## Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for other languages exist, these are the core ones that are supported by the main project hosted on google code.

| Language | Client Version | Release Date | | | |
|---|---|---|---|---|---|
| Java | 3.0.0-beta1 | 2016-07-28 | Download | Change log | Javadoc |
| C# | 2.53.1 | 2016-06-28 | Download | Change log | API docs |
| Ruby | 3.0.0.beta1 | 2016-07-28 | Download | Change log | API docs |
| Python | 2.53.6 | 2016-06-28 | Download | Change log | API docs |
| Javascript (Node) | 2.53.3 | 2016-06-28 | Download | Change log | API docs |

Note: Selenium Standalone Server is only required if want to use remote server to run the tests. Since this document is all above setting up the project so its better to have everything at place.

3. The jars will get downloaded in zip file, unzip them. You should be able to see `.jar` directly.

4. In eclipse, right click on the project which you created in step-1 and follow below steps.

   Properties ->Java Build Path -> Select Libraries tab -> Click Add External Jars -> Locate the unzipped jar folder which you downloaded above -> Select all the jars from `lib` folder -> Click Ok -> Again click on Add External Jars -> Locate same unzipped folder -> Select the jar which is outside of lib folder (`client-combined-3.0.0-beta1-nodeps.jar`) -> Ok

Similarly add the `Selenium Standalone Server` following the above step.

5. Now you can start writing selenium code into your project.

**PS**: Above documentation is based on selenium-3.0.0 beta version so the names of jar files specified may change with version.

Read Getting started with selenium-webdriver online: https://riptutorial.com/selenium-webdriver/topic/878/getting-started-with-selenium-webdriver

# Chapter 2: Actions (Emulating complex user gestures)

## Introduction

The `Actions` class gives us a way of emulating precisely how a user would interact with a web page/elements. Using an instance of this class you can describe a series of actions, such as clicking, double-clicking, dragging, pressing keys, etc. Once these actions are described, in order to carry the actions out, you must call must build the actions (`.Build()`) and then instruct them to be performed (`.Perform()`). So we must describe, build, perform. The examples below will expand upon this.

## Syntax

- dragAndDrop(WebElement source, WebElement target)
- dragAndDropBy(WebElement source, int xOffset, int yOffset)
- perform()

## Parameters

| Parameters | Details |
|---|---|
| source | Element to emulate button down at. |
| target | Element to move to and release the mouse at. |
| xOffset | x co-ordinate to move to. |
| yOffset | y co-ordinate to move to. |

## Remarks

This section contains information of Actions class of Selenium WebDriver. The Actions class provides you convenient methods to perform complex user gestures like drag and drop, hold and click etc.

## Examples

### Drag and Drop

# C#

```
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Interactions;

namespace WebDriverActions
{
    class WebDriverTest
    {
        static void Main()
        {
            IWebDriver driver = new FirefoxDriver();

            driver.Navigate().GoToUrl("");
            IWebElement source = driver.FindElement(By.CssSelector(""));
            IWebElement target = driver.FindElement(By.CssSelector(""));
            Actions action = new Actions(driver);
            action.DragAndDrop(source, target).Perform();
        }
    }
}
```

The above will find an `IWebElement`, `source`, and drag it to, and drop it into the second `IWebElement`, `target`.

# Java

Drag and Drop using source and target webelement.

A convenience method that performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

/**
 * Drag and Drop test using source and target webelement
 */
public class DragAndDropClass {
    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        driver.get("");
        WebElement source = driver.findElement(By.cssSelector(""));
        WebElement target = driver.findElement(By.cssSelector(""));
        Actions action = new Actions(driver);
        action.build();
        action.dragAndDrop(source, target).perform();
    }
}
```

Drag an element and drop it at a given offset.

A convenience method that performs click-and-hold at the location of the source element, moves

by a given offset (x and y, both integers), then releases the mouse.

```
WebElement source = driver.findElement(By.cssSelector(""));
Actions action = new Actions(driver);
action.build()
action.dragAndDropBy(source, x, y).perform(); // x and y are integers value
```

### Move to Element

## C#

Suppose you want to test that when you hover over an element, a drop list is displayed. You may want to check the contents of this list, or perhaps select an option from the list.

First create an Action, to hover over the element *(e.g. my element has link text "Admin")*:

```
Actions mouseHover = new Actions(driver);
mouseHover.MoveToElement(driver.FindElement(By.LinkText("Admin"))).Perform();
```

In the example above:

- You have created the action `mouseHover`
- You have told `driver` to move to a specific element
- From here you can perform other `Actions` with the `mouseHover` object or continue testing with your `driver` object

**This approach is of particular use when clicking on an element performs a different function than hovering over it.**

A full example:

```
Actions mouseHover = new Actions(driver);
mouseHover.MoveToElement(driver.FindElement(By.LinkText("Admin"))).Perform();

Assert.IsTrue(driver.FindElement(By.LinkText("Edit Record")).Displayed);
Assert.IsTrue(driver.FindElement(By.LinkText("Delete Record")).Displayed);
```

Read Actions (Emulating complex user gestures) online: https://riptutorial.com/selenium-webdriver/topic/4849/actions--emulating-complex-user-gestures-

# Chapter 3: Basic Selenium Webdriver Program

## Introduction

This topic aims to show the basic web driver program in selenium supported languages like C#, Groovy, Java, Perl, PHP, Python and Ruby.

Journey includes opening browser driver --> Google Page --> shutdown the browser

## Examples

### C#

```
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;

namespace BasicWebdriver
{
    class WebDriverTest
    {
        static void Main()
        {
            using (var driver = new ChromeDriver())
            {
                driver.Navigate().GoToUrl("http://www.google.com");
            }
        }
    }
}
```

The above 'program' will navigate to the Google homepage, and then close down the browser after fully loading the page.

```
using (var driver = new ChromeDriver())
```

This instantiates a new WebDriver object using the `IWebdriver` interface and creates a new browser window instance. In this example we are using `ChromeDriver` (though this could be replaced by the appropriate driver for whichever browser we wanted to use). We are wrapping this with a `using` statement, because `IWebDriver` implements `IDisposable`, thus not needing to explicitly type in `driver.Quit();`.

In case you haven't downloaded your WebDriver using NuGet, then you will need to pass an argument in the form of a path to the directory where the driver itself "chromedriver.exe" is located.

---

## Navigating

```
driver.Navigate().GoToUrl("http://www.google.com");
```

and

```
driver.Url = "http://www.google.com";
```

Both of these lines do the same thing. They instruct the driver to navigate to a specific URL, and to wait until the page is loaded before it moves to the next statement.

There are other methods tied to navigation such as `Back()`, `Forward()` or `Refresh()`.

---

After that, the `using` block safely quits, and disposes the object.

### Python

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

def set_up_driver():
    path_to_chrome_driver = 'chromedriver'
    return webdriver.Chrome(executable_path=path_to_chrome_driver)

def get_google():
    driver = set_up_driver()
    driver.get('http://www.google.com')
    tear_down(driver)

def tear_down(driver):
    driver.quit()

if '__main__' == __name__:
    get_google()
```

The above 'program' will navigate to the Google homepage, and then close down the browser before completing.

```
if '__main__' == __name__:
    get_google()
```

First we have our main function, our point of entry into the program, that calls `get_google()`.

```
def get_google():
    driver = set_up_driver()
```

`get_google()` then starts by creating our `driver` instance via `set_up_driver()`:

```
def set_up_driver():
    path_to_chrome_driver = 'chromedriver'
    return webdriver.Chrome(executable_path=path_to_chrome_driver)
```

Whereby we state where `chromedriver.exe` is located, and instantiate our driver object with this

path. The remainder of `get_google()` navigates to Google:

```
driver.get('http://www.google.com')
```

And then calls `tear_down()` passing the driver object:

```
tear_down(driver)
```

`tear_down()` simply contains one line to shut down our driver object:

```
driver.quit()
```

This tells the driver to close all open browser windows and dispose of the browser object, as we have no other code after this call this effectively ends the program.

### Java

The code below is all about 3 steps.

1. Opening a chrome browser
2. Opening google page
3. Shutdown the browser

```
import org.openqa.selenium;
import org.openqa.selenium.chrome;

public class WebDriverTest {
    public static void main(String args[]) {
        System.setProperty("webdriver.chrome.driver", "C:\\path\\to\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        driver.get("http://www.google.com");
        driver.quit();
    }
}
```

The above 'program' will navigate to the Google homepage, and then close down the browser before completing.

```
System.setProperty("webdriver.chrome.driver", "C:\\path\\to\\chromedriver.exe");
WebDriver driver = new ChromeDriver();
```

The first line tells the system where to find the `ChromeDriver` (chromedriver.exe) executable. We then create our driver object by calling the `ChromeDriver()` constructor, again we could be calling our constructor here for any browser/platform.

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

This tells our driver to navigate to the specified url: http://www.google.com. The Java WebDriver API provides the `get()` method directly on the WebDriver interface, though further navigation

methods can be found via the `navigate()` method, e.g. `driver.navigate.back()`.

Once the page has finished loading we immediately call:

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

This tells the driver to close all open browser windows and dispose of the driver object, as we have no other code after this call this effectively ends the program.

```
driver.close();
```

Is an instruction (not shown here) to the driver to close only the active window, in this instance as we only have a single window the instructions would cause identical results to calling `quit()`.

## Java - Best practise with page classes

Usecase : Login to FB account

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class FaceBookLoginTest {
    private static WebDriver driver;
    HomePage homePage;
    LoginPage loginPage;
    @BeforeClass
    public void openFBPage(){
        driver = new FirefoxDriver();
        driver.get("https://www.facebook.com/");
        loginPage = new LoginPage(driver);
    }
    @Test
    public void loginToFB(){
       loginPage.enterUserName("username");
       loginPage.enterPassword("password");
       homePage = loginPage.clickLogin();
        System.out.println(homePage.getUserName());
    }

}
```

Page classes : Login Page & Home Page Login page class :

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class LoginPage {

    WebDriver driver;
    public LoginPage(WebDriver driver){
        this.driver = driver;
```

```
    }
    @FindBy(id="email")
    private WebElement loginTextBox;

    @FindBy(id="pass")
    private WebElement passwordTextBox;

    @FindBy(xpath = ".//input[@data-testid='royal_login_button']")
    private WebElement loginBtn;

    public void enterUserName(String userName){
        if(loginTextBox.isDisplayed()) {
            loginTextBox.clear();
            loginTextBox.sendKeys(userName);
        }
        else{
            System.out.println("Element is not loaded");
        }
    }
    public void enterPassword(String password){
        if(passwordTextBox.isDisplayed()) {
            passwordTextBox.clear();
            passwordTextBox.sendKeys(password);
        }
        else{
            System.out.println("Element is not loaded");
        }
    }
    public HomePage clickLogin(){
        if(loginBtn.isDisplayed()) {
            loginBtn.click();
        }
        return new HomePage(driver);
    }
}
```

Home page class:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class HomePage {

    WebDriver driver;
    public HomePage(WebDriver driver){
        this.driver = driver;
    }

    @FindBy(xpath=".//a[@data-testid='blue_bar_profile_link']/span")
    private WebElement userName;

    public String getUserName(){
        if(userName.isDisplayed()) {
            return userName.getText();
        }
        else {
            return "Username is not present";
        }
    }
```

```
}
```

Read Basic Selenium Webdriver Program online: https://riptutorial.com/selenium-webdriver/topic/3990/basic-selenium-webdriver-program

# Chapter 4: Error Handling in Automation using Selenium

## Examples

### Python

`WebDriverException` is a base `Selenium-WebDriver` exception that could be used to catch all other `Selenium-WebDriver` exceptions

To be able to catch exception it should be imported first:

```
from selenium.common.exceptions import WebDriverException as WDE
```

and then:

```
try:
    element = driver.find_element_by_id('ID')
except WDE:
    print("Not able to find element")
```

In the same way you can import other more specific exceptions:

```
from selenium.common.exceptions import ElementNotVisibleException
from selenium.common.exceptions import NoAlertPresentException
...
```

If you want to extract exception message only:

```
from selenium.common.exceptions import UnexpectedAlertPresentException

try:
    driver.find_element_by_tag_name('a').click()
except UnexpectedAlertPresentException as e:
    print(e.__dict__["msg"])
```

Read Error Handling in Automation using Selenium online: https://riptutorial.com/selenium-webdriver/topic/9548/error-handling-in-automation-using-selenium

# Chapter 5: Exceptions in Selenium-WebDriver

## Introduction

There are a number of exceptions that can be thrown while using a webdriver. The examples below are meant to give an idea of what they mean.

## Examples

### Python Exceptions

Selenium Exception Documentation

**ElementNotInteractableException:** Thrown when an element is present in the DOM but interactions with that element will hit another element due to paint order

- **ElementNotSelectableException:** Thrown when trying to select an unselectable element. Examples of unselectable elements:
  - script
- **ElementNotVisibleException:** Thrown when an element is present on the DOM, but it is not visible, and so is not able to be interacted with. Most commonly encountered when trying to click or read text of an element that is hidden from view.
- **ErrorInResponseException:** Thrown when an error has occurred on the server side. This may happen when communicating with the firefox extension or the remote driver server.
- **ImeActivationFailedException:** Thrown when activating an IME engine has failed.
- **ImeNotAvailableException:** Thrown when IME support is not available. This exception is thrown for every IME-related method call if IME support is not available on the machine.
- **InvalidArgumentException:** The arguments passed to a command are either invalid or malformed.
- **InvalidCookieDomainException:** Thrown when attempting to add a cookie under a different domain than the current URL.
- **InvalidElementStateException:** Thrown when an action would result in an invalid state for an element. Subclasses:
  - ElementNotInteractableException
  - ElementNotSelectableException
  - ElementNotVisibleException
- **InvalidSelectorException:** Thrown when the selector which is used to find an element does not return a WebElement. Currently this only happens when the selector is an xpath expression and it is either syntactically invalid (i.e. it is not a xpath expression) or the expression does not select WebElements (e.g. “count(//input)”).
- **InvalidSwitchToTargetException:** Thrown when frame or window target to be switched doesn’t exist.
- **MoveTargetOutOfBoundsException:** Thrown when the target provided to the ActionsChains move() method is invalid, i.e. out of document.

- **NoAlertPresentException:** Thrown when switching to no presented alert. This can be caused by calling an operation on the Alert() class when an alert is not yet on the screen.
- **NoSuchAttributeException:** Thrown when the attribute of element could not be found. You may want to check if the attribute exists in the particular browser you are testing against. Some browsers may have different property names for the same property. (IE8's .innerText vs. Firefox .textContent)
- **NoSuchElementException:** Thrown when element could not be found. If you encounter this exception, you may want to check the following:
    - Check your selector used in your find_by...
    - Element may not yet be on the screen at the time of the find operation, (webpage is still loading) see selenium.webdriver.support.wait.WebDriverWait() for how to write a wait wrapper to wait for an element to appear.
- **NoSuchFrameException:** Thrown when frame target to be switched doesn't exist.
- **NoSuchWindowException:** Thrown when window target to be switched doesn't exist. To find the current set of active window handles, you can get a list of the active window handles in the following way:
  ```
  print driver.window_handles
  ```
- **RemoteDriverServerException:**
- **StaleElementReferenceException:** Thrown when a reference to an element is now "stale". Stale means the element no longer appears on the DOM of the page. Possible causes of StaleElementReferenceException include, but not limited to:
    - You are no longer on the same page, or the page may have refreshed since the element was located.
    - The element may have been removed and re-added to the screen, since it was located. Such as an element being relocated. This can happen typically with a javascript framework when values are updated and the node is rebuilt.
    - Element may have been inside an iframe or another context which was refreshed.
- **TimeoutException:** Thrown when a command does not complete in enough time.
- **UnableToSetCookieException:** Thrown when a driver fails to set a cookie.
- **UnexpectedAlertPresentException:** Thrown when an unexpected alert is appeared. Usually raised when when an expected modal is blocking webdriver form executing any more commands.
- **UnexpectedTagNameException:** Thrown when a support class did not get an expected web element.
- **WebDriverException:** Base webdriver exception. All webdriver exceptions either use WebDriverException or InvalidStateException as the parent class.

Read Exceptions in Selenium-WebDriver online: https://riptutorial.com/selenium-webdriver/topic/10546/exceptions-in-selenium-webdriver

# Chapter 6: Executing Javascript in the page

## Syntax

- object ExecuteAsyncScript(string script, params object[] args);
- object ExecuteScript(string script, params object[] args);

## Examples

### C#

In order to execute JavaScript in a `IWebDriver` instance you need to cast the `IWebDriver` to a new interface, `IJavaScriptExecutor`

```
IWebDriver driver;
IJavaScriptExecutor jsDriver = driver as IJavaScriptExecutor;
```

You can now access all the methods available on the `IJavaScriptExecutor` instance which allow you to execute Javascript, for example:

```
jsDriver.ExecuteScript("alert('running javascript');");
```

### Python

To execute Javascript in python, use `execute_script("javascript script here")`. execute_script is called on a webdriver instance, and can be any valid javascript.

```
from selenium import webdriver
driver = webdriver.Chrome()
driver.execute_script("alert('running javascript');")
```

### Java

To execute Javascript in Java, create a new webdriver that supports Javascript. To use the `executeScript()` function, either the driver must be cast to a `JavascriptExecutor`, or a new variable can be set to the value of the casted driver: `((JavascriptExecutor)driver).` `driver.executeScript()` takes in a String that is valid Javascript.

```
WebDriver driver = new ChromeDriver();
JavascriptExecutor JavascriptExecutor = ((JavascriptExecutor)driver);
JavascriptExecutor.executeScript("alert('running javascript');");
```

### Ruby

```
require "selenium-webdriver"

driver = Selenium::WebDriver.for :chrome
driver.execute_script("alert('running javascript');")
```

Read Executing Javascript in the page online: https://riptutorial.com/selenium-webdriver/topic/6986/executing-javascript-in-the-page

# Chapter 7: Handle an alert

## Examples

### Selenium with Java

Here's how to Handle a popup alert in Java with Selenium:

There are 3 types of popups.

1. **Simple alert** : alert("This is a simple alert");
2. **Confirmation alert** : var popuResult = confirm("Confirm pop up with OK and Cancel button");
3. **Prompt alert** : var person = prompt("Do you like stackoverflow?", "Yes/No");

Its upto user which type of popup need to be handled in their test case.

Either you can

1. accept() To accept the alert
2. dismiss() To dismiss the alert
3. getText() To get the text of the alert
4. sendKeys() To write some text to the alert

***For simple alert:***

```
Alert simpleAlert = driver.switchTo().alert();
String alertText = simpleAlert.getText();
System.out.println("Alert text is " + alertText);
simpleAlert.accept();
```

***For Confirmation alert :***

```
Alert confirmationAlert = driver.switchTo().alert();
String alertText = confirmationAlert.getText();
System.out.println("Alert text is " + alertText);
confirmationAlert.dismiss();
```

***For Prompt alert :***

```
Alert promptAlert  = driver.switchTo().alert();
String alertText = promptAlert .getText();
System.out.println("Alert text is " + alertText);
//Send some text to the alert
promptAlert .sendKeys("Accepting the alert");
Thread.sleep(4000); //This sleep is not necessary, just for demonstration
promptAlert .accept();
```

according to your needs.

Another way you can do this, is wrap your code inside a try-catch:

```
try{
   // Your logic here.
} catch(UnhandledAlertException e){
  Alert alert = driver.switchTo().alert();
  alert.accept();
}
// Continue.
```

### C#

Here's how to close a popup alert in C# with Selenium:

```
IAlert alert = driver.SwitchTo().Alert();
// Prints text and closes alert
System.out.println(alert.Text);
alert.Accept();
or
alert.Dismiss();
```

according to your needs.

Another way you can do this, is wrap your code inside a try-catch:

```
try{
   // Your logic here.
} catch(UnhandledAlertException e){
  var alert = driver.SwitchTo().Alert();
  alert.Accept();
}
// Continue.
```

## Python

There are multiple ways to switch to alert pop-up in `Python`:

1. *Deprecated*:

```
alert = driver.switch_to_alert()
```

2. *Using* *`switch_to`*:

```
alert = driver.switch_to.alert
```

3. *Using* *`ExplicitWait`*:

```
 from selenium.webdriver.common.by import By
 from selenium.webdriver.support.ui import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as EC

alert = WebDriverWait(driver, TIMEOUT_IN_SECONDS).until(EC.alert_is_present())
```

4. *By declaring the instance of `Alert` class*:

```
from selenium.webdriver.common.alert import Alert

alert = Alert(driver)
```

To fill input field in pop-up triggered by `JavaScript prompt()`:

```
alert.send_keys('Some text to send')
```

To confirm dialog pop-up*:

```
alert.accept()
```

To dismiss:

```
alert.dismiss()
```

To get text from pop-up:

```
alert.text
```

**P.S.* *`alert.dismiss()` could be used to confirm pop-ups triggered by `JavaScript alert()` as well as `alert.confirm()`*

Read Handle an alert online: https://riptutorial.com/selenium-webdriver/topic/6048/handle-an-alert

# Chapter 8: Headless Browsers

## Examples

### PhantomJS [C#]

`PhantomJS` is a fully featured headless web browser **with** JavaScript support.

Before you start you will need to download a PhantomJS driver, and make sure to put this in the beginning of your code:

```
using OpenQA.Selenium;
using OpenQA.Selenium.PhantomJS;
```

Great, now onto the initialization:

```
var driver = new PhantomJSDriver();
```

This will simply create a new instance of the PhantomJSDriver class. You can then use it the same way as every WebDriver such as:

```
using (var driver = new PhantomJSDriver())
{
    driver.Navigate().GoToUrl("http://stackoverflow.com/");

    var questions = driver.FindElements(By.ClassName("question-hyperlink"));

    foreach (var question in questions)
    {
        // This will display every question header on StackOverflow homepage.
        Console.WriteLine(question.Text);
    }
}
```

This works fine. However, the problem you probably encountered is, when working with UI, `PhantomJS` opens a new console window, which is not really wanted in most cases. Luckily, we can hide the window, and even slightly improve performance using `PhantomJSOptions`, and `PhantomJSDriverService`. Full working example below:

```
// Options are used for setting "browser capabilities", such as setting a User-Agent
// property as shown below:
var options = new PhantomJSOptions();
options.AddAdditionalCapability("phantomjs.page.settings.userAgent",
"Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:25.0) Gecko/20100101 Firefox/25.0");

// Services are used for setting up the WebDriver to your likings, such as
// hiding the console window and restricting image loading as shown below:
var service = PhantomJSDriverService.CreateDefaultService();
service.HideCommandPromptWindow = true;
service.LoadImages = false;
```

```
// The same code as in the example above:
using (var driver = new PhantomJSDriver(service, options))
{
    driver.Navigate().GoToUrl("http://stackoverflow.com/");

    var questions = driver.FindElements(By.ClassName("question-hyperlink"));

    foreach (var question in questions)
    {
        // This will display every question header on StackOverflow homepage.
        Console.WriteLine(question.Text);
    }
}
```

*Pro tip: click on a class (e.g the `PhantomJSDriverService`), and press F12 to see exactly what they contain along with a brief description of what they do.*

### SimpleBrowser [C#]

`SimpleBrowser` is a lightweight WebDriver **without** JavaScript support.

It is considerably faster than an aforementioned `PhantomJS`, however when it comes to functionality, it is limited to simple tasks with no fancy features.

Firstly, you will need to download the SimpleBrowser.WebDriver package and then put this code at the beginning:

```
using OpenQA.Selenium;
using SimpleBrowser.WebDriver;
```

Now, here is a short example on how to use it:

```
using (var driver = new SimpleBrowserDriver())
{
    driver.Navigate().GoToUrl("http://stackoverflow.com/");

    var questions = driver.FindElements(By.ClassName("question-hyperlink"));

    foreach (var question in questions)
    {
        // This will display every question header on StackOverflow homepage.
        Console.WriteLine(question.Text);
    }
}
```

### Headless browser in Java

## HTMLUnitDriver

HTMLUnitDriver is the most lightweight implementation of headless(GUI-less) browser for Webdriver based on HtmlUnit. It models HTML documents and provides an API that allows you to

invoke pages, fill out forms, click links, etc. just like you do in your normal browser. It supports JavaScript and works with AJAX libraries. It is used for testing and retrieving data from website.

---

**Example:** Use of HTMLUnitDriver to fetch list of questions from `http://stackoverflow.com/`.

---

```
import java.util.List;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;

class testHeadlessDriver{
            private void getQuestions() {
                    WebDriver driver = new HtmlUnitDriver();
                    driver.get("http://stackoverflow.com/");
                    driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
                    List<WebElement> questions = driver.findElements(By.className("question-
hyperlink"));
                    questions.forEach((question) -> {
                        System.out.println(question.getText());
                    });
                   driver.close();
                }
    }
```

---

It is same as any other browser(Mozilla Firefox, Google Chrome, IE), but it does not have GUI, execution is not visible on screen.

Read Headless Browsers online: https://riptutorial.com/selenium-webdriver/topic/3931/headless-browsers

# Chapter 9: HTML Reports

## Introduction

This topic covers the creation of HTML reports for selenium tests. There are various types of plugins available for reporting and the widely used are Allure, ExtentReports and ReportNG.

## Examples

### ExtentReports

This example covers the implementation of ExtentReports in Selenium using TestNG, Java and Maven.

ExtentReports are available in two versions, community and commercial. For the ease and demonstration purpose, we will be using community version.

#### 1. Dependency

Add the dependency in your Maven pom.xml file for extent reports.

```
<dependency>
     <groupId>com.aventstack</groupId>
     <artifactId>extentreports</artifactId>
     <version>3.0.6</version>
 </dependency>
```

#### 2. Configure plugins

Configure the maven surefire plugin as below in pom.xml

```
 <build>
 <defaultGoal>clean test</defaultGoal>
 <plugins>
     <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.6.1</version>
         <configuration>
             <source>${jdk.level}</source>
             <target>${jdk.level}</target>
         </configuration>
     </plugin>
     <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-surefire-plugin</artifactId>
         <version>2.19.1</version>
         <configuration>
             <suiteXmlFiles>
                 <suiteXmlFile>testng.xml</suiteXmlFile>
             </suiteXmlFiles>
```

```
            </configuration>
        </plugin>
    </plugins>
</build>
```

### 3. Sample test with ExtentReports

Now, create a test with name test.java

```
public class TestBase {
    WebDriver driver;

    ExtentReports extent;
    ExtentTest logger;
    ExtentHtmlReporter htmlReporter;
    String htmlReportPath = "C:\\Screenshots/MyOwnReport.html"; //Path for the HTML report to
be saved

    @BeforeTest
    public void setup(){
        htmlReporter = new ExtentHtmlReporter(htmlReportPath);
        extent = new ExtentReports();
        extent.attachReporter(htmlReporter);

        System.setProperty("webdriver.chrome.driver", "pathto/chromedriver.exe");
        driver = new ChromeDriver();

    }

    @Test
    public void test1(){
        driver.get("http://www.google.com/");
        logger.log(Status.INFO, "Opened site google.com");
        assertEquals(driver.getTitle()), "Google");
        logger.log(Status.PASS, "Google site loaded");
    }

    @AfterMethod
    public void getResult(ITestResult result) throws Exception {
        if (result.getStatus() == ITestResult.FAILURE)
        {
            logger.log(Status.FAIL, MarkupHelper.createLabel(result.getName() + " Test case
FAILED due to below issues:", ExtentColor.RED));
            logger.fail(result.getThrowable());
        }
        else if (result.getStatus() == ITestResult.SUCCESS)
        {
            logger.log(Status.PASS, MarkupHelper.createLabel(result.getName() + " Test Case
PASSED", ExtentColor.GREEN));
        }
        else if (result.getStatus() == ITestResult.SKIP)
        {
            logger.log(Status.SKIP, MarkupHelper.createLabel(result.getName() + " Test Case
SKIPPED", ExtentColor.BLUE));
        }
     }

    @AfterTest
    public void testend() throws Exception {
        extent.flush();
```

```
    }

    @AfterClass
    public void tearDown() throws Exception {
        driver.close();
    }
```

### Allure Reports

This example covers the implementation of Allure Reports in Selenium using TestNG, Java and Maven.

# Maven Configuration

## Repository

Add following code to configure the jcenter repository

```
<repository>
            <id>jcenter</id>
            <name>bintray</name>
            <url>http://jcenter.bintray.com</url>
</repository>
```

## Dependency

Add following dependencies to your pom.xml

```
<dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>${aspectj.version}</version>
    </dependency>
    <dependency>
        <groupId>ru.yandex.qatools.allure</groupId>
        <artifactId>allure-testng-adaptor</artifactId>
        <version>1.5.4</version>
    </dependency>
```

## Surefire Plugin Configuration

```
<plugin>
        <groupId> org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.20</version>
        <configuration>
            <argLine>-
javaagent:${settings.localRepository}/org/aspectj/aspectjweaver/${aspectj.version}/aspectjweaver-
${aspectj.version}.jar
```

```
            </argLine>
            <properties>
                <property>
                    <name>listener</name>
                    <value>ru.yandex.qatools.allure.testng.AllureTestListener</value>
                </property>
            </properties>
            <suiteXmlFiles>testng.xml</suiteXmlFiles>
            <testFailureIgnore>false</testFailureIgnore>
        </configuration>
    </plugin>
```

# Sample test for Allure Report

Create a sample test with name test.java

```
public class test{
    WebDriver driver;
    WebDriverWait wait;

    @BeforeMethod
    public void setup() {
        System.setProperty("webdriver.chrome.driver", "path to/chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("https://www.google.com/");
        wait = new WebDriverWait(driver,50);
    }

    @Title("Title check")
    @Description("Checking the title of the loaded page.")
    @Test
    public void searchTest(){
        String title = driver.getTitle();
        LogUtil.log("Title Fetched: "+title);
        assertEquals(title,"Google");
        LogUtil.log("Test Passed. Expected: Google | Actual: "+title);
        System.out.println("Page Loaded");
    }

    @AfterMethod
    public void teardown(){
        driver.close();
    }
}
```

In the above class we have used LogUtiil class. This is simply done to log **Steps** in our test. Below is the code for the same

LogUtil.java

```
public final class LogUtil {

    private LogUtil() {
    }

    @Step("{0}")
```

```
    public static void log(final String message){
        //intentionally empty
    }
}
```

Here

**@Title("")** will add the title to your test in Allure Report

**@Description("")** will add the description to your test

**@Step("")** will add a step in the allure report for the test

On execution a xml file will be generated in the folder "target/allure-results/"

**Final Report with Jenkins**

If you are running in Jenkins with Allure Report plugin installed, then Jenkins will automatically render the report in your job.

**Final Report without Jenkins**

For those who dont have a Jenkins, use the following commandline to create the html report. Allure CLI is a Java application so it's available for all platforms. You have to manually install Java 1.7+ before using Allure CLI.

*Debian*

For Debian-based repositories we provide a PPA so the installation is straightforward: Install Allure CLI for debian

```
$ sudo apt-add-repository ppa:yandex-qatools/allure-framework
$ sudo apt-get update
$ sudo apt-get install allure-commandline
```

Supported distributions are: Trusty and Precise. After installation you will have allure command available.

*Mac OS*

You can install Allure CLI via Homebrew.

```
$ brew tap qatools/formulas
$ brew install allure-commandline
```

After installation you will have allure command available.

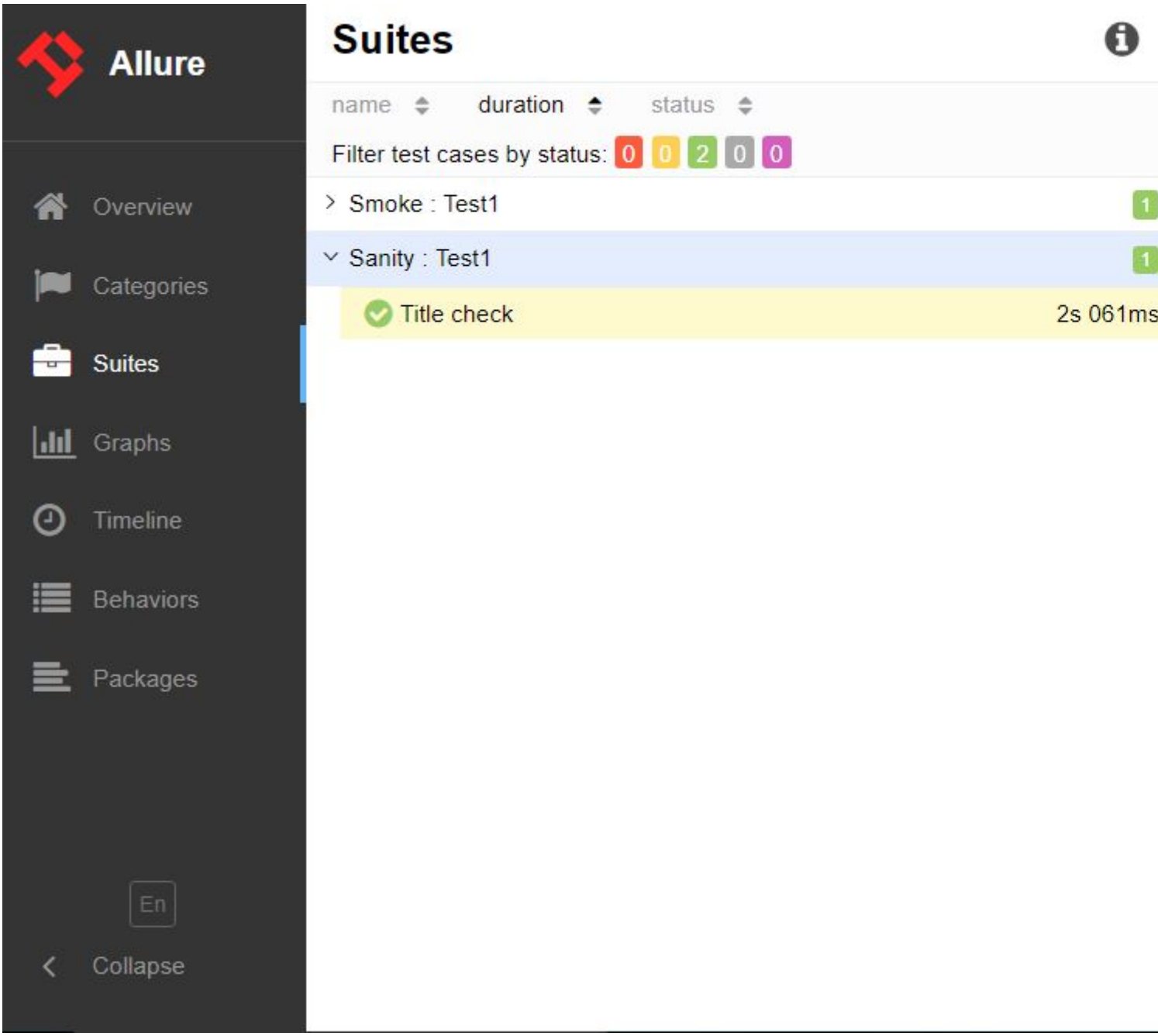
*Windows and other Unix*

1. Download the latest version as zip archive from https://github.com/allure-framework/allure-core/releases/latest.

2. Unpack the archive to allure-commandline directory. Navigate to bin directory.
3. Use allure.bat for Windows and allure for other Unix platforms.

In the Commandline/Terminal now simply enter following syntax and report will be generated into allure-report folder

```
$ allure generate directory-with-results/
```




Read HTML Reports online: https://riptutorial.com/selenium-webdriver/topic/10721/html-reports

# Chapter 10: Interacting with the Browser Window(s)

## Examples

### Managing the active window

# C#

*Maximizing the window*

```
driver.Manage().Window.Maximize();
```

This is fairly straightforward, ensures that our currently active window is maximized.

*Position of the window*

```
driver.Manage().Window.Position = new System.Drawing.Point(1, 1);
```

Here we essentially move the currently active window to a new position. In the `Point` object we provide `x` and `y` co-ordinates; these are then used as offsets from the top-left corner of the screen to determine where the window should be placed. Note that you can also store the window position in a variable:

```
System.Drawing.Point windowPosition = driver.Manage().Window.Position;
```

*Size of the window*

Setting and getting the window size uses the same syntax as the position:

```
driver.Manage().Window.Size = new System.Drawing.Size(100, 200);
System.Drawing.Size windowSize = driver.Manage().Window.Size;
```

*URL of the window*

We can obtain the current URL of the active window:

```
string url = driver.Url;
```

We can also set the URL for the active window, which will make the driver navigate to the new value:

```
driver.Url = "http://stackoverflow.com/";
```

*Window handles*

We can obtain the handle for the current window:

```
string handle = driver.CurrentWindowHandle;
```

And we can obtain the handles for all open windows:

```
IList<String> handles = driver.WindowHandles;
```

# Python

*Maximizing the window*

```
driver.maximize_window()
```

*Get position of the window*

```
driver.get_window_position() # returns {'y', 'x'} coordinates
```

*Set position of the window*

```
driver.set_window_position(x, y) # pass 'x' and 'y' coordinates as arguments
```

*Get size of the window*

```
driver.get_window_size() # returns {'width', 'height'} values
```

*Set size of the window*

```
driver.set_window_size(width, height) # pass 'width' and 'height' values as arguments
```

*Current page title*

```
driver.title
```

*Current URL*

```
driver.current_url
```

*Window handles*

```
driver.current_window_handle
```

*List of currently opened windows*

```
driver.window_handles
```

### Closing the current browser window

Switch to the new opened tab. Close the current windows(In this case the new Tab). Switch back to first window.

**PROTRACTOR:**

```
browser.getAllWindowHandles().then(function (handles) {
    browser.driver.switchTo().window(handles[1]);
    browser.driver.close();
    browser.driver.switchTo().window(handles[0]);
});
```

**JAVA Selenium:**

```
        Set<String> handlesSet = driver.getWindowHandles();
        List<String> handlesList = new ArrayList<String>(handlesSet);
        driver.switchTo().window(handlesList.get(1));
        driver.close();
        driver.switchTo().window(handlesList.get(0));
```

### Handle multiple windows

# Python

Most commonly used scenario:

1. *open page in new window*
2. *switch to it*
3. *do something*
4. *close it*
5. *switch back to parent window*

```
# Open "Google" page in parent window
driver.get("https://google.com")

driver.title # 'Google'

# Get parent window
parent_window = driver.current_window_handle

# Open "Bing" page in child window
driver.execute_script("window.open('https://bing.com')")

# Get list of all windows currently opened (parent + child)
all_windows = driver.window_handles

# Get child window
child_window = [window for window in all_windows if window != parent_window][0]
```

```
# Switch to child window
driver.switch_to.window(child_window)

driver.title # 'Bing'

# Close child window
driver.close()

# Switch back to parent window
driver.switch_to.window(parent_window)

driver.title # 'Google'
```

Read Interacting with the Browser Window(s) online: https://riptutorial.com/selenium-webdriver/topic/5181/interacting-with-the-browser-window-s-

# Chapter 11: Interaction With Web Element

## Examples

### C#

Clearing the Content of element(Generally Text Box)

```
interactionWebElement.Clear();
```

Entering data to element (Generally Text Box)

```
interactionWebElement.SendKeys("Text");
```

Storing the value of the element.

```
string valueinTextBox = interactionWebElement.GetAttribute("value");
```

Storing Text of element.

```
string textOfElement = interactionWebElement.Text;
```

Clicking on an Element

```
interactionWebElement.Click();
```

Submitting a Form

```
interactionWebElement.Submit();
```

Identifing the visibility of an element on the page

```
bool isDisplayed=interactionWebElement.Displayed;
```

Identifing the state of an element on the page

```
bool isEnabled = interactionWebElement.Enabled;

bool isSelected=interactionWebElement.Selected;
```

Locating child element of interactionWebElement

```
IWebElement childElement = interactionWebElement.FindElement(By.Id("childElementId"));
```

Locating child elements of interactionWebElement

```
Ilist<IWebElement> childElements =
interactionWebElement.FindElements(By.TagName("childElementsTagName"));
```

### Java

Clearing the content of a web element: (note - when simulating user actions in tests, it's better to send backspace, see next action)

```
interactionWebElement.clear();
```

Entering data - simulating sending keystrokes:

```
interactionWebElement.sendKeys("Text");
interactionWebElement.sendKeys(Keys.CONTROL + "c"); // copy to clipboard.
```

Getting the value of an element's attribute:

```
interactionWebElement.getAttribute("value");
interactionWebElement.getAttribute("style");
```

Getting element's text:

```
String elementsText = interactionWebElement.getText();
```

Selecting from dropdown:

```
Select dropDown = new Select(webElement);
dropDown.selectByValue(value);
```

Self explanatory:

```
interactionWebElement.click();
interactionWebElement.submit(); //for forms
interactionWebElement.isDisplayed();
interactionWebElement.isEnabled(); // for exampale - is clickable.
interactionWebElement.isSelected(); // for radio buttons.
```

---

**Actions using** `org.openqa.selenium.interactions.Actions`:

Drag & Drop:

```
Action dragAndDrop = builder.clickAndHold(someElement)
   .moveToElement(otherElement)
   .release(otherElement)
   .build();

dragAndDrop.perform();
```

Select multiple:

```
Action selectMultiple = builder.keyDown(Keys.CONTROL)
   .click(someElement)
   .click(someOtherElement)
   .keyUp(Keys.CONTROL);

dragAndDrop.perform();
```

Self explanatory (using builder):

```
builder.doubleClick(webElement).perform();
builder.moveToElement(webElement).perform(); //hovering
```

See here for more examples of advanced actions and a complete list.

---

Using Javascript:

```
// Scroll to view element:
((JavascriptExecutor) driver).executeJavaScript("arguments[0].scrollIntoView(true);",
webElement);
```

Read Interaction With Web Element online: https://riptutorial.com/selenium-webdriver/topic/4280/interaction-with-web-element

# Chapter 12: Listeners

## Examples

### JUnit

If you are using JUnit to execute, you can extend the `TestWatcher` class:

```
public class TestRules extends TestWatcher {

    @Override
    protected void failed(Throwable e, Description description) {
        // This will be called whenever a test fails.
    }
```

So in your test class you can simply call it:

```
public class testClass{

@Rule
public TestRules testRules = new TestRules();

@Test
public void doTestSomething() throws Exception{
    // If the test fails for any reason, it will be caught be testrules.
}
```

### EventFiringWebDriver

Using the EventFiringWebDriver. You can attach WebDriverEventListener to it and override methodes, ie the onException method:

```
EventFiringWebDriver driver = new EventFiringWebDriver(new FirefoxDriver());
WebDriverEventListener listener = new AbstractWebDriverEventListener() {
    @Override
    public void onException(Throwable t, WebDriver driver) {
        // Take action
    }
};
driver.register(listener);
```

Read Listeners online: https://riptutorial.com/selenium-webdriver/topic/8226/listeners

# Chapter 13: Locating Web Elements

## Syntax

- ByChained(params By[] bys)

## Remarks

Items are found in Selenium through the use of *locators* and the `By` class. In order to make a robust automation project with Selenium, one should use locators for Web Elements smartly. The locators should be **descriptive, unique, and unlikely to change** so you won't get false positives in tests for example. The priority is to use:

1. **ID** - since it's unique and you'll get exactly the element you want.
2. **Class Name** - It's descriptive and can be unique in a given context.
3. **CSS** (better performance than xpath) - For more complicated selectors.
4. **XPATH** - Where CSS can't be used (XPATH Axis), e.g. `div::parent`.

The rest of the locators are prone to changes or rendering, and preferebly be avoided.

**Rule of thumb:** if your code cannot locate a particular element, one reason could be that your code hasn't waited for all the DOM elements to download. Consider telling your program to "wait" for a short period of time (try 3-5 seconds, and then slowly increase as needed) before searching for said element. Here is an example in Python, taken from this question:

```
from selenium import webdriver
import time

browser = webdriver.Firefox()
browser.get("https://app.website.com")

reports_element = browser.find_element_by_xpath("//button[contains(text(), 'Reports')]")

# Element not found! Try giving time for the browser to download all DOM elements:
time.sleep(10)

reports_element = browser.find_element_by_xpath("//button[contains(text(), 'Reports')]")
# This returns correct value!
```

---

## Examples

### Locating page elements using WebDriver

To interact with WebElements in a webpage, first we need to identify the location of the element.

***By*** is the keyword available in selenium.

You can locate the elements By..

1. **By** *ID*
2. **By** *Class Name*
3. **By** *TagName*
4. **By** *Name*
5. **By** *Link Text*
6. **By** *Partial Link Text*
7. **By** *CSS Selector*
8. **By** *XPath*
9. **Using** *JavaScript*

Consider the below script example

```
<form name="loginForm">
Login Username: <input id="username" name="login" type="text" />
Password: <input id="password" name="password" type="password" />
<input name="login" type="submit" value="Login" />
```

In the above code the username and password are set using id's. Now you are going to identify the elements with id.

```
driver.findElement(By.id(username));

driver.findElement(By.id(password));
```

As selenium support 7 different languages, this document gives you an idea to locate the elements in all the languages.

---

# By ID

Example of how to find an element using ID:

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

Java       -  WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
C#         -  IWebElement element = driver.FindElement(By.Id("coolestWidgetEvah"));
Python     -  element = driver.find_element_by_id("coolestWidgetEvah")
Ruby       -  element = driver.find_element(:id, "coolestWidgetEvah")
JavaScript/Protractor -  var elm = element(by.id("coolestWidgetEvah"));
```

---

# By Class Name

Example of how to find an element using class name:

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

```
Java       -  WebElement element = driver.findElement(By.className("cheese"));
C#         -  IWebElement element = driver.FindElement(By.ClassName("cheese"));
Python     -  element = driver.find_element_by_class_name("cheese")
Ruby       -  cheeses = driver.find_elements(:class, "cheese")
JavaScript/Protractor -  var elm = element(by.className("cheese"));
```

---

## By Tag Name

Example of how to find an element using tag name:

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

Java       -  WebElement element = driver.findElement(By.tagName("iframe"));
C#         -  IWebElement element = driver.FindElement(By.TagName("iframe"));
Python     -  element = driver.find_element_by_tag_name("iframe")
Ruby       -  frame = driver.find_element(:tag_name, "iframe")
JavaScript/Protractor -  var elm = element(by.tagName("iframe"));
```

---

## By Name

Example of how to find an element using name:

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

Java       -  WebElement element = driver.findElement(By.name("cheese"));
C#         -  IWebElement element = driver.FindElement(By.Name("cheese"));
Python     -  element = driver.find_element_by_name("cheese")
Ruby       -  cheese = driver.find_element(:name, "cheese")
JavaScript/Protractor -  var elm = element(by.name("cheese"));
```

---

## By Link Text

Example of how to find an element using link text:

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

Java       -  WebElement element = driver.findElement(By.linkText("cheese"));
C#         -  IWebElement element = driver.FindElement(By.LinkText("cheese"));
Python     -  element = driver.find_element_by_link_text("cheese")
Ruby       -  cheese = driver.find_element(:link, "cheese")
JavaScript/Protractor -  var elm = element(by.linkText("cheese"));
```

---

## By Partial Link Text

Example of how to find an element using partial link text:

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

```
Java       -  WebElement element = driver.findElement(By.partialLinkText("cheese"));
C#         -  IWebElement element = driver.FindElement(By.PartialLinkText("cheese"));
Python     -  element = driver.find_element_by_partial_link_text("cheese")
Ruby       -  cheese = driver.find_element(:partial_link_text, "cheese")
JavaScript/Protractor -  var elm = element(by.partialLinkText("cheese"));
```

---

## By CSS Selectors

Example of how to find an element using CSS Selectors:

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

Java       -  WebElement element = driver.findElement(By.cssSelector("#food.dairy")); //# is
used to indicate id and . is used for classname.
C#         -  IWebElement element = driver.FindElement(By.CssSelector("#food.dairy"));
Python     -  element = driver.find_element_by_css_selector("#food.dairy")
Ruby       -  cheese = driver.find_element(:css, "#food span.dairy.aged")
JavaScript/Protractor -  var elm = element(by.css("#food.dairy"));
```

Here's an article about creating CSS Selectors:
http://www.w3schools.com/cssref/css_selectors.asp

---

## By XPath

Example of how to find an element using XPath:

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

Java       -  WebElement element = driver.findElement(By.xpath("//input"));
C#         -  IWebElement element = driver.FindElement(By.XPath("//input"));
Python     -  element = driver.find_element_by_xpath("//input")
Ruby       -  inputs = driver.find_elements(:xpath, "//input")
JavaScript/Protractor -  var elm = element(by.xpath("//input"));
```

Here's an article about XPath: http://www.w3schools.com/xsl/xpath_intro.asp

---

## Using JavaScript

You can execute arbitrary javascript to find an element and as long as you return a DOM Element, it will be automatically converted to a WebElement object.

Simple example on a page that has jQuery loaded:

```
Java       -  WebElement element = (WebElement)
((JavascriptExecutor)driver).executeScript("return $('.cheese')[0]");

C#         -  IWebElement element = (IWebElement)
```

```
((IJavaScriptExecutor)driver).ExecuteScript("return $('.cheese')[0]");

Python     -  element = driver.execute_script("return $('.cheese')[0]");
Ruby       -  element = driver.execute_script("return $('.cheese')[0]")
JavaScript/Protractor -
```

*Please note: This method will not work if your particular WebDriver doesn't support JavaScript, such as SimpleBrowser.*

### Selecting by multiple criteria [C#]

It's also possible to use selectors together. This is done by using the `OpenQA.Selenium.Support.PageObjects.ByChained` object:

```
element  = driver.FindElement(new ByChained(By.TagName("input"), By.ClassName("class"));
```

Any number of `By`s can be chained and are used as an AND type selection (i.e. all `By`s are matched)

### Selecting elements before the page stops loading

When calling `driver.Navigate().GoToUrl(url);`, the code execution stops until the page is fully loaded. This is sometimes unnecessary when you just want to extract data.

*Note: The code samples below could be considered hacks. There is no "official" way of doing this.*

---

## Create a new thread

Create and launch a thread for loading a webpage, then use Wait.

*C#*

```
using (var driver = new ChromeDriver())
{
    new Thread(() =>
    {
        driver.Navigate().GoToUrl("http://stackoverflow.com");
    }).Start();

    new WebDriverWait(driver, TimeSpan.FromSeconds(10))
      .Until(ExpectedConditions.ElementIsVisible(By.XPath("//div[@class='summary']/h3/a")));
}
```

---

## Use Timeouts

Using a WebDriverTimeout, you can load a page, and after a certain period of time, it will throw an exception, which will make the page stop loading. In the catch block, you can use Wait.

*C#*

```
using (var driver = new ChromeDriver())
{
    driver.Manage().Timeouts().SetPageLoadTimeout(TimeSpan.FromSeconds(5));

    try
    {
        driver.Navigate().GoToUrl("http://stackoverflow.com");
    }
    catch (WebDriverTimeoutException)
    {
        new WebDriverWait(driver, TimeSpan.FromSeconds(10))
          .Until(ExpectedConditions.ElementIsVisible
            (By.XPath("//div[@class='summary']/h3/a")));
    }
}
```

**The problem**: When you set the timeout for too short, the page will stop loading regardless of whether is your desired element present. When you set the timeout for too long, you're going to negate the performance benefit.

Read Locating Web Elements online: https://riptutorial.com/selenium-webdriver/topic/3991/locating-web-elements

# Chapter 14: Navigate between multiple frames

## Introduction

In web pages contain number of frame, selenium consider Frame is seprate window so access the content present into frame need to switch into frame. Many time we need Web structure where we have frame with in frame to navigate within frame windows Selenium provide swithTo() method.

## Examples

### Frame example

```
<iframe "id="iframe_Login1">

    <iframe "id="iframe_Login2">

        <iframe "id="iframe_Login3">

        </iframe>

    </iframe>

</iframe>
```

To switch into frame in selenium use swithTo() and frame()method.

driver.switchTo().frame(iframe_Login1); driver.switchTo().frame(iframe_Login2); driver.switchTo().frame(iframe_Login3);

To switch back we can use parentFrame() and defaultContest();

parentFrame() : Change focus to the parent context. If the current context is the top level browsing context, the context remains unchanged.

```
driver.switchTo().parentFrame();
```

defaultContent() : Selects either the first frame on the page, or the main document when a page contains iframes.

```
 driver.switchTo().defaultContent();
```

Read Navigate between multiple frames online: https://riptutorial.com/selenium-webdriver/topic/9803/navigate-between-multiple-frames

# Chapter 15: Navigation

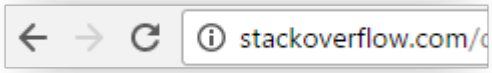
## Syntax

- **C#**
- void Back()
- void Forward()
- void GotToUrl(string url)
- void Refresh()
- **Python**
- driver.back()
- driver.forward()
- driver.get("URL")
- driver.refresh()
- **Java**
- driver.navigate().back();
- driver.navigate().forward();
- driver.navigate().to("URL");
- driver.navigate().refresh();

## Examples

### Navigate() [C#]

It's possible to navigate the browser directly, like using the standard toolbar commands available on all browsers:




You can create a navigation object by calling `Navigate()` on the driver:

```
IWebDriver driver
INavigation navigation = driver.Navigate();
```

A navigation object allows you to perform numerous actions that navigate the browser around the web:

```
//like pressing the back button
navigation.Back();
//like pressing the forward button on a browser
navigation.Forward();
//navigate to a new url in the current window
navigation.GoToUrl("www.stackoverflow.com");
//Like pressing the reload button
navigation.Refresh();
```

## Navigate () [Java]

For Navigate to any url :

```
driver.navigate().to("http://www.example.com");
```

For move backwards :

```
driver.navigate().back();
```

For move forwards :

```
driver.navigate().forward();
```

For refresh the page :

```
driver.navigate().refresh();
```

## Browser methods in WebDriver

WebDriver, The main interface to use for testing, which represents an idealised web browser. The methods in this class fall into three categories:

- Control of the browser itself
- Selection of WebElements
- Debugging aids

Key methods are get(String), which is used to load a new web page, and the various methods similar to findElement(By), which is used to find WebElements. In this post we are going to learn browser controlling methods. get

```
void get(java.lang.String url)
```

Load a new web page in the current browser window. This is done using an HTTP GET operation, and the method will block until the load is complete. it is best to wait until this timeout is over, since should the underlying page change whilst your test is executing the results of future calls against this interface will be against the freshly loaded page. **Usage**

```
//Initialising driver
 WebDriver driver = new FirefoxDriver();

 //setting timeout for page load
 driver.manage().timeouts().pageLoadTimeout(20, TimeUnit.SECONDS);

 //Call Url in get method
 driver.get("https://www.google.com");
 //or
 driver.get("https://seleniumhq.org");
```

### getCurrentUrl

```
java.lang.String getCurrentUrl()
```

Get a string representing the current URL that the browser is looking at. It returns the URL of the page currently loaded in the browser.

*Usage*

```
//Getting current url loaded in browser & comparing with expected url
 String pageURL = driver.getCurrentUrl();
 Assert.assertEquals(pageURL, "https://www.google.com");
```

### getTitle

```
java.lang.String getTitle()
```

It returns the title of the current page, with leading and trailing whitespace stripped, or null if one is not already set.

*Usage*

```
//Getting current page title loaded in browser & comparing with expected title
 String pageTitle = driver.getTitle();
 Assert.assertEquals(pageTitle, "Google");

getPageSource

java.lang.String getPageSource()
```

Get the source of the last loaded page. If the page has been modified after loading (for example, by Javascript) there is no guarantee that the returned text is that of the modified page.

*Usage*

```
//get the current page source
 String pageSource = driver.getPageSource();
```

### close

```
void close()
```

Close the current window, quitting the browser if it's the last window currently open. If there are more than one window opened with that driver instance this method will close the window which is having current focus on it.

*Usage*

```
//Close the current window
     driver.close();
```

**quit**

```
void quit()
```

Quits this driver, closing every associated window. After calling this method we can not use any other method using same driver instance.

*Usage*

```
//Quit the current driver session / close all windows associated with driver
     driver.quit();
```

These are all very useful methods available in Selenium 2.0 to control browser as required.

Read Navigation online: https://riptutorial.com/selenium-webdriver/topic/7272/navigation

# Chapter 16: Page Object Model

## Introduction

A significant role in automating web sites and web applications involves identifying items on the screen and interacting with them. Items are found in Selenium through the use of locators and the `By` class. These locators and interactions are put inside Page Objects as a best practice to avoid duplicate code and make maintenance easier. It encapsulates `WebElements` and suppose to contain behavior and return info on the page (or part of a page in a web app).

## Remarks

Page object model is a pattern where we write object oriented classes that serve as an interface to a particular view of web page. We use the methods of that page class to perform the required action. Few years back, we were manipulating the HTML code of webpage in test classes directly which was very difficult to maintain along with brittle to changes in UI.

However, having your code organized in a way of Page Object Pattern provides an application specific API, allowing you to manipulate the page elements without digging around the HTML. The basic Rue of thumb says, your page object should have everything which a human can do on that webpage. For example, to access the text field on a webpage you should a method there to get the text and return string after doing all the modifications.

Few important points you should keep in mind while designing the page objects:

1. Page object usually should not build only for pages, but you should prefer to build it for significant elements of page. For example, a page having multiple tabs to show different charts of your academics should have same number of pages as the count of tabs.

2. Navigating from one view to other should return the instance of page classes.

3. The utility methods which are required to be there only for a specific view or webpage should belong to that page class only.

4. The assertion methods shouldn't be taken care by page classes, you can have methods to return boolean but don't verify them there. For example, to verify user full name you can have method to get boolean value:

```
public boolean hasDisplayedUserFullName (String userFullName) {
    return driver.findElement(By.xpath("xpathExpressionUsingFullName")).isDisplayed();
}
```

5. If your webpage is based on iframe, prefer to have page classes for iframes too.

Advantages of Page Object Pattern:

1. Clean separation between test code and page code

2. In case of any change in UI of webpage, no need to change your code at multiple places. Change only in page classes.
3. No scattered element locators.
4. Makes code easier to understand
5. Easy maintenance

# Examples

## Introduction (Using Java)

An example to perform login test based on Page object pattern:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

/**
 * Class which models the view of Sign-In page
 */
public class SignInPage {

    @FindBy(id="username")
    private usernameInput;

    @FindBy(id="password")
    private passwordInput;

    @FindBy(id="signin")
    private signInButton;

    private WebDriver driver;

    public SignInPage(WebDriver driver) {
        this.driver = driver;
    }

    /**
     * Method to perform login
     */
    public HomePage performLogin(String username, String password) {
        usernameInput.sendKeys(username);
        passwordInput.sendKeys(password);
        signInButton.click();
        return PageFactory.initElements(driver, HomePage.class);
    }
}


import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
/**
 * Class which models the view of home page
 */
public class HomePage {
```

```
    @FindBy(id="logout")
    private logoutLink;

    private WebDriver driver;

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }

    /**
     * Method to log out
     */
    public SignInPage logout() {
        logoutLink.click();
        wait.ForPageToLoad();
        return PageFactory.initElements(driver, SignInPage.class);
    }
}

/**
 * Login test class
 */
public class LoginTest {
    public void testLogin() {
        SignInPage signInPage = new SignInPage(driver);
        HomePage homePage = signInPage.login(username, password);
        signInPage = homePage.logout();
    }
}
```

### C#

Page Objects should contain behavior, return info for assertions and possibly a method for page ready state method on initialization. Selenium supports Page Objects using annotations. In C# it's as follows:

```
using OpenQA.Selenium;
using OpenQA.Selenium.Support.PageObjects;
using OpenQA.Selenium.Support.UI;
using System;
using System.Collections.Generic;

public class WikipediaHomePage
{
    private IWebDriver driver;
    private int timeout = 10;
    private By pageLoadedElement = By.ClassName("central-featured-logo");

    [FindsBy(How = How.Id, Using = "searchInput")]
    [CacheLookup]
    private IWebElement searchInput;

    [FindsBy(How = How.CssSelector, Using = ".pure-button.pure-button-primary-progressive")]
    [CacheLookup]
    private IWebElement searchButton;

    public ResultsPage Search(string query)
    {
```

```
        searchInput.SendKeys(query);
        searchButton.Click();
    }

    public WikipediaHomePage VerifyPageLoaded()
    {
        new WebDriverWait(driver, TimeSpan.FromSeconds(timeout)).Until<bool>((drv) =>  return
drv.ExpectedConditions.ElementExists(pageLoadedElement));

        return this;
    }
}
```

notes:

- `CacheLookup` saves the element in the cache and saves returning a new element each call. This improves performance but isn't good for dynamically changing elements.
- `searchButton` has 2 class names and no ID, that's why I can't use class name or id.
- I verified that my locators will return me the element I want using Developer Tools (for Chrome), in other browsers you can use FireBug or similar.
- `Search()` method returns another page object (`ResultsPage`) as the search click redirects you to another page.

## Best Practices Page Object Model

- Create separate files for header and footer(as they are common for all the pages and it does not make sense to make them a part of a single page)
- Keep common elements(Like Search/Back/Next etc) in separate file(The idea is to remove any kind of duplication and keeping the segregation logical)
- For Driver, its a good idea to create a separate Driver class and keep Driver as static so that it can be accessed across all pages! (I have all my webpages extend DriverClass)
- The functions used in PageObjects are broken down into smallest possible chunk keeping in mind the frequency and the way in which they will be called(The way you have done for login- although login can be broken down into enterUsername and enterPassword functions but still keeping it as Login function is more logical because in majority of the cases, the Login function would be called rather than separate calls to enterUsername and enterPassword functions)
- Using PageObjects itself segregates Test script from the elementLocators
- Have utility functions in separate utils folder(like DateUtil, excelUtils etc)
- Have configurations in separate conf folder(like setting the environment on which the tests need to be run, configuring the output and input folders)
- Incorporate screenCapture on failure
- Have a static wait variable in the DriverClass with some implicit wait time as you have done Always try to have conditional waits rather than static waits like: wait.until(ExpectedConditions). This ensures that the wait is not slowing down the execution unnecessarily.

Read Page Object Model online: https://riptutorial.com/selenium-webdriver/topic/4853/page-object-model

# Chapter 17: Robot In Selenium

## Syntax

- delay(int ms)
- keyPress(int keycode)
- keyRelease(int keycode)
- mouseMove(int x, int y)
- mousePress(int buttons)
- mouseRelease(int buttons)
- mouseWheel(int wheelAmt)

## Parameters

| Parameter | Details |
|---|---|
| ms | Time to sleep in milliseconds |
| keycode | Constant to press the specified key for example to press `A` code is `VK_A`. Please refer for more details : https://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyEvent.html |
| x,y | Screen coordintates |
| buttons | The Button mask; a combination of one or more mouse button masks |
| wheelAmt | Number of notches to move mouse wheel, negative value to move up/away from user positive value to move down/towards user |

## Remarks

This section contains details about implementation of Robot API with Selenium Webdriver. The Robot class is used to generate native system input when selenium is not capable to do that for example pressing right key of mouse, pressing F1 key etc.

## Examples

### Keypress event using Robot API (JAVA)

```
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;

public class KeyBoardExample {
```

```
    public static void main(String[] args) {
        try {
            Robot robot = new Robot();
            robot.delay(3000);
            robot.keyPress(KeyEvent.VK_Q); //VK_Q for Q
        } catch (AWTException e) {
            e.printStackTrace();
        }
    }
}
```

**With Selenium**

Sometimes we need to press any key in order to test the key press event on web application. For an instance to test the ENTER key on login form we can write something like below with Selenium WebDriver

```
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;

public class LoginTest {

    @Test
    public void testEnterKey() throws InterruptedException
    {
        WebDriver driver=new FirefoxDriver();
        Robot robot=null;
        driver.get("test-url");
        driver.manage().window().maximize();
        driver.findElement(By.xpath("xpath-expression")).click();
        driver.findElement(By.xpath("xpath-expression")).sendKeys("username");
        driver.findElement(By.xpath("xpath-expression")).sendKeys("password");
        try {
            robot=new Robot();
        } catch (AWTException e) {
            e.printStackTrace();
        }
        //Keyboard Activity Using Robot Class
        robot.keyPress(KeyEvent.VK_ENTER);
    }
}
```

## Mouse Event using Robot API (JAVA)

**Mouse movement:**

```
import java.awt.Robot;

public class MouseClass {
 public static void main(String[] args) throws Exception {
     Robot robot = new Robot();
```

```
     // SET THE MOUSE X Y POSITION
     robot.mouseMove(300, 550);
     }
}
```

### Press left/right button of mouse:

```
import java.awt.Robot;
import java.awt.event.InputEvent;

public class MouseEvent {
 public static void main(String[] args) throws Exception {
     Robot robot = new Robot();

     // LEFT CLICK
     robot.mousePress(InputEvent.BUTTON1_MASK);
     robot.mouseRelease(InputEvent.BUTTON1_MASK);

     // RIGHT CLICK
     robot.mousePress(InputEvent.BUTTON3_MASK);
     robot.mouseRelease(InputEvent.BUTTON3_MASK);
     }
}
```

### Click and scroll the wheel:

```
import java.awt.Robot;
import java.awt.event.InputEvent;

public class MouseClass {
 public static void main(String[] args) throws Exception {
     Robot robot = new Robot();

     // MIDDLE WHEEL CLICK
     robot.mousePress(InputEvent.BUTTON3_DOWN_MASK);
     robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK);

     // SCROLL THE MOUSE WHEEL
     robot.mouseWheel(-100);
     }
}
```

Read Robot In Selenium online: https://riptutorial.com/selenium-webdriver/topic/4877/robot-in-selenium

# Chapter 18: Scrolling

## Introduction

This Topic will provide several approaches of how to perform scrolling with `selenium`

## Examples

### Scrolling using Python

*1. Scrolling to target element ("BROWSE TEMPLATES" button at the bottom of page) with `Actions`*

```
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Chrome()
driver.get('http://www.w3schools.com/')
target = driver.find_element_by_link_text('BROWSE TEMPLATES')
actions = ActionChains(driver)
actions.move_to_element(target)
actions.perform()
```

*2. Scrolling to target element ("BROWSE TEMPLATES" button at the bottom of page) with `JavaScript`*

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get('http://www.w3schools.com/')
target = driver.find_element_by_link_text('BROWSE TEMPLATES')
driver.execute_script('arguments[0].scrollIntoView(true);', target)
```

*3. Scrolling to target element ("BROWSE TEMPLATES" button at the bottom of page) with built-in method*

```
from selenium import webdriver

    driver = webdriver.Chrome()
    driver.get('http://www.w3schools.com/')
    target = driver.find_element_by_link_text('BROWSE TEMPLATES')
    target.location_once_scrolled_into_view
```

*Note that `location_once_scrolled_into_view` also returns `x`, `y` coordinates of element after scrolling*

*4. Scrolling to the bottom of page with `Keys`*

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()
```

```
driver.get('http://www.w3schools.com/')
driver.find_element_by_tag_name('body').send_keys(Keys.END) # Use send_keys(Keys.HOME) to
scroll up to the top of page
```

Note that `send_keys(Keys.DOWN)`/`send_keys(Keys.UP)` and `send_keys(Keys.PAGE_DOWN)`/`send_keys(Keys.PAGE_UP)` also could be used for scrolling

## Different Scrolling using java with different ways

Below give solution can be also use in another supported programming languages with some syntax changes

---

1. To do **Scroll down** page/section/division in webpage while there is custom scroll bar(Not browser scroll). Click Here For demo and check scroll bar has its independent element.

   In below given code pass your scroll bar element and require scroll points.

```
    public static boolean scroll_Page(WebElement webelement, int scrollPoints)
    {
    try
    {
        System.out.println("---------------- Started - scroll_Page ----------------");
        driver = ExecutionSetup.getDriver();
        dragger = new Actions(driver);

        // drag downwards
        int numberOfPixelsToDragTheScrollbarDown = 10;
        for (int i = 10; i < scrollPoints; i = i + numberOfPixelsToDragTheScrollbarDown)
        {
            dragger.moveToElement(webelement).clickAndHold().moveByOffset(0,
numberOfPixelsToDragTheScrollbarDown).release(webelement).build().perform();
        }
        Thread.sleep(500);
        System.out.println("---------------- Ending - scroll_Page ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- scroll is unsucessfully done in scroll_Page -----
-----------");
        e.printStackTrace();
        return false;
    }
  }
```

---

2. To do **Scroll Up** page/section/division in webpage while there is custom scroll bar(Not browser scroll). Click Here For demo and check scroll bar has its independent element.

   In below given code pass your scroll bar element and require scroll points.

```
public static boolean scroll_Page_Up(WebElement webelement, int scrollPoints)
{
    try
```

```
    {
        System.out.println("---------------- Started - scroll_Page_Up ----------------");
        driver = ExecutionSetup.getDriver();
        dragger = new Actions(driver);
        // drag upwards
        int numberOfPixelsToDragTheScrollbarUp = -10;
        for (int i = scrollPoints; i > 10; i = i + numberOfPixelsToDragTheScrollbarUp)
        {
            dragger.moveToElement(webelement).clickAndHold().moveByOffset(0,
numberOfPixelsToDragTheScrollbarUp).release(webelement).build().perform();
        }
        System.out.println("---------------- Ending - scroll_Page_Up ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- scroll is unsucessfully done in scroll_Page_Up---
-------------");
        e.printStackTrace();
        return false;
    }
}
```

---

3. To do scroll down when **multiple browser scroll** (In-Built browser) and you want to scroll down with **Page Down Key**. Click Here for demo

   In below given code pass your scroll area element like `<div>` and require page down key.

```
 public static boolean pageDown_New(WebElement webeScrollArea, int iLoopCount)
 {
    try
    {
        System.out.println("---------------- Started - pageDown_New ----------------");
        driver = ExecutionSetup.getDriver();
        dragger = new Actions(driver);

        for (int i = 0; i <= iLoopCount; i++)
        {

dragger.moveToElement(webeScrollArea).click().sendKeys(Keys.PAGE_DOWN).build().perform();
        }
        System.out.println"---------------- Ending - pageDown_New ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- Not able to do page down ----------------");
        return false;
    }
}
```

---

4. To do scroll UP when **multiple browser scroll** (In-Built browser) and you want to scroll Up with **Page UP Key**. Click Here for demo

   In below given code pass your scroll area element like `<div>` and require page up key.

```
public static boolean pageUp_New(WebElement webeScrollArea, int iLoopCount)
{
    try
    {
        System.out.println("---------------- Started - pageUp_New ----------------");
        driver = ExecutionSetup.getDriver();
        dragger = new Actions(driver);

        for (int i = 0; i <= iLoopCount; i++)
        {

dragger.moveToElement(webeScrollArea).click().sendKeys(Keys.PAGE_UP).build().perform();
        }
        System.out.println("---------------- Ending - pageUp_New ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- Not able to do page up ----------------");
        return false;
    }
}
```

---

5. To do scroll Down when **multiple browser scroll** (In-Built browser) and you want to scroll down with **Only Down arrow Key**. Click Here for demo

   In below given code pass your scroll area element like `<div>` and require down key.

```
public static boolean scrollDown_Keys(WebElement webeScrollArea, int iLoopCount)
{
    try
    {
        System.out.println("---------------- Started - scrollDown_Keys ----------------");
        driver = ExecutionSetup.getDriver();
        dragger = new Actions(driver);

        for (int i = 0; i <= iLoopCount; i++)
        {

dragger.moveToElement(webeScrollArea).click().sendKeys(Keys.DOWN).build().perform();
        }
        System.out.println("---------------- Ending - scrollDown_Keys ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- Not able to do scroll down with keys-------------
---");
        return false;
    }
}
```

---

6. To do scroll Up when **multiple browser scroll** (In-Built browser) and you want to scroll Up with **Only Up arrow Key**. Click Here for demo

   In below given code pass your scroll area element like `<div>` and require up key.

```
public static boolean scrollUp_Keys(WebElement webeScrollArea, int iLoopCount)
{
    try
    {
        System.out.println("---------------- Started - scrollUp_Keys ----------------");
        driver = ExecutionSetup.getDriver();
        dragger = new Actions(driver);

        for (int i = 0; i <= iLoopCount; i++)
        {
            dragger.moveToElement(webeScrollArea).click().sendKeys(Keys.UP).build().perform();
        }
        System.out.println("---------------- Ending - scrollUp_Keys ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- Not able to do scroll up with keys---------------
-");
        return false;
    }
}
```

---

7. To do scroll Up/Down when **browser scroll** (In-Built browser) and you want to scroll Up/Down with **Only fixed point**. Click Here for demo

   In below given code pass your scroll point. Positive means down and negative means scroll up.

```
public static boolean scroll_without_WebE(int scrollPoint)
{
    JavascriptExecutor jse;
    try
    {
        System.out.println("---------------- Started - scroll_without_WebE ----------------");

        driver = ExecutionSetup.getDriver();
        jse = (JavascriptExecutor) driver;
        jse.executeScript("window.scrollBy(0," + scrollPoint + ")", "");

        System.out.println("---------------- Ending - scroll_without_WebE ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- scroll is unsucessful in scroll_without_WebE ----
------------");
        e.printStackTrace();
        return false;
    }
}
```

---

8. To do scroll Up/Down when **browser scroll** (In-Built browser) and you want to scroll Up/Down to **For make element in visible area or dynamic scroll**. Click Here for demo

   In below given code pass your element.

```
public static boolean scroll_to_WebE(WebElement webe)
{
    try
    {
        System.out.println("---------------- Started - scroll_to_WebE ----------------");

        driver = ExecutionSetup.getDriver();
        ((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView();", webe);

        System.out.println("---------------- Ending - scroll_to_WebE ----------------");
        return true;
    }
    catch (Exception e)
    {
        System.out.println("---------------- scroll is unsucessful in scroll_to_WebE ---------
-------");
        e.printStackTrace();
        return false;
    }
}
```

---

***Note : Please verify your case and use methods. If any case is missing then let me know.***

Read Scrolling online: https://riptutorial.com/selenium-webdriver/topic/9063/scrolling

# Chapter 19: Select Class

## Syntax

- **Java**
- deselectAll()
- deselectByIndex(int index)
- deselectByValue(java.lang.String value)
- deselectByVisibleText(java.lang.String text)
- getAllSelectedOptions()
- getFirstSelectedOption()
- getOptions()
- isMultiple()
- selectByIndex(int index)
- selectByValue(java.lang.String value)
- selectByVisibleText(java.lang.String text)

## Parameters

| Parameters | Details |
|---|---|
| index | The option at this index will be selected |
| value | The value to match against |
| text | The visible text to match against |

## Remarks

`Select` class of Selenium WebDriver provides useful methods for interacting with `select` options. User can perform operations on a select dropdown and also de-select operation using the below methods.

In **C#** the Select class is actually `SelectElement`

## Examples

### Different ways to Select from DropDown list

Below is and HTML page

```
<html>
<head>
<title>Select Example by Index value</title>
```

```
</head>
<body>
<select name="Travel"><option value="0" selected> Please select</option>
<option value="1">Car</option>
<option value="2">Bike</option>
<option value="3">Cycle</option>
<option value="4">Walk</option>
</select>
</body>
</html>
```

# JAVA

## Select By Index

To select the option by Index using Java

```
public class selectByIndexExample {
    WebDriver driver;
    @Test
    public void selectSamples()
    {
        driver = new FirefoxDriver();
        driver.get("URL GOES HERE");
        WebElement element=driver.findElement(By.name("Travel")); //This is the 'Select'
element locator
        Select sel=new Select(element);
        sel.selectByIndex(1); //This will select the first 'Option' from 'Select' List i.e.
Car
    }
}
```

## Select By Value

```
public class selectByValueExample {
    WebDriver driver;
    @Test
    public void selectSamples()
    {
        driver = new FirefoxDriver();
        driver.get("URL GOES HERE");
        WebElement element=driver.findElement(By.name("Travel")); //This is the 'Select'
element locator
        Select sel=new Select(element);
        sel.selectByValue("Bike"); //This will select the 'Option' from 'Select' List which
has value as "Bike".
        //NOTE: This will be case sensitive
    }
}
```

## Select By Visibility Text

```
public class selectByVisibilityTextExample {
    WebDriver driver;
    @Test
    public void selectSamples()
    {
        driver = new FirefoxDriver();
        driver.get("URL GOES HERE");
        WebElement element=driver.findElement(By.name("Travel")); //This is the 'Select'
element locator
        Select sel=new Select(element);
        sel.selectByVisibleText("Cycle"); //This will select the 'Option' from 'Select' List
who's visibility text is "Cycle".
        //NOTE: This will be case sensitive
    }
}
```

# C#

All examples below are based on the generic `IWebDriver` interface

## Select By Index

```
IWebElement element=driver.FindElement(By.name("Travel"));
SelectElement selectElement = new SelectElement(title);
selectElement.SelectByIndex(0);
```

## Select By Value

```
IWebElement element=driver.FindElement(By.name("Travel"));
SelectElement selectElement = new SelectElement(title);
selectElement.SelectByIndex("1");
//NOTE: This will be case sensitive
```

## Select By Text

```
IWebElement element=driver.FindElement(By.name("Travel"));
SelectElement selectElement = new SelectElement(title);
selectElement.SelectByText("Walk");
```

Read Select Class online: https://riptutorial.com/selenium-webdriver/topic/6426/select-class

# Chapter 20: Selenium e2e setup

## Introduction

This topic covers the end to end setup of Selenium i.e. Selenium Webdriver + TestNG + Maven + Jenkins.

For report addition, please refer to topic HTML Reports

## Examples

### TestNG Setup

TestNG is your updated test framework for junit. We are going to utilize **testng.xml** for invoking test suites. This is helpful when we are going to use CI ahead.

# testng.xml

In the root folder of your project create an xml file with the name testng.xml. Note that name can be different as well, but for convenience its used as "testng" everywhere.

Below is the simple code for testng.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Smoke"> //name of the suite
    <test name="Test1"> //name of the test
        <classes>
            <class name="test.SearchTest">
            <methods>
                <include name="searchTest"/>
            </methods>
            </class>
        </classes>
    </test>
</suite>
```

### Maven Setup

TBD. How to setup pom.xml for calling testng.xml

### Jenkins Setup

TBD. Will cover the Jenkins setup for pulling code from git/bitbucket etc.

Read Selenium e2e setup online: https://riptutorial.com/selenium-webdriver/topic/10724/selenium-e2e-setup

# Chapter 21: Selenium Grid

## Examples

### Node configuration

Selenium Grid Node configuration resides on the Node itself and holds the information about network configuration and Node capabilities. The configuration can be applied in various ways:

- Default Configuration
- JSON Configuration
- Command line Configuration

**JSON Configuration**

The node configuration in JSON file is split into 2 sections:

- Capabilities
- Configuration

***Capabilities*** defines areas like what Browser types and versions are supported, locations of browser binaries, number of maximum instances of each browser type.

***Configuration*** deals with settings like hub and node addresses and ports.

Below is a an example of a JSON configuration file:

```
{
  "capabilities": [
    {
      "browserName": "firefox",
      "acceptSslCerts": true,
      "javascriptEnabled": true,
      "takesScreenshot": false,
      "firefox_profile": "",
      "browser-version": "27",
      "platform": "WINDOWS",
      "maxInstances": 5,
      "firefox_binary": "",
      "cleanSession": true
    },
    {
      "browserName": "chrome",
      "maxInstances": 5,
      "platform": "WINDOWS",
      "webdriver.chrome.driver": "C:/Program Files (x86)/Google/Chrome/Application/chrome.exe"
    },
    {
      "browserName": "internet explorer",
      "maxInstances": 1,
      "platform": "WINDOWS",
      "webdriver.ie.driver": "C:/Program Files (x86)/Internet Explorer/iexplore.exe"
```

```
    }
  ],
  "configuration": {
    "_comment" : "Configuration for Node",
    "cleanUpCycle": 2000,
    "timeout": 30000,
    "proxy": "org.openqa.grid.selenium.proxy.WebDriverRemoteProxy",
    "port": 5555,
    "host": ip,
    "register": true,
    "hubPort": 4444,
    "maxSessions": 5
  }
}
```

## How to create a node

To create a node, you first need to have a hub. If you don't have a hub, you can create it like this:

```
java -jar selenium-server-standalone-<version>.jar -role hub
```

Then you're able to create a node:

```
java -jar selenium-server-standalone-<version>.jar -role node -hub
http://localhost:4444/grid/register // default port is 4444
```

More info here: https://github.com/SeleniumHQ/selenium/wiki/Grid2

Read Selenium Grid online: https://riptutorial.com/selenium-webdriver/topic/1359/selenium-grid

# Chapter 22: Selenium Grid Configuration

## Introduction

Selenium Grid is a framework to run test distributed over a range of test devices. It's used for testing web applications. Its possible to write tests in different popular programming languages, including C#, Groovy, Java, Perl, PHP, Python and Ruby. The tests can be run against a range of webbrowsers on platforms like Windows, Linux, and OS X.

It is open-source software, released under the Apache 2.0 license: web developers can download and use it without charge.

## Syntax

- for to run the jar file the following is the syntax for every jar file
- `java -jar <jar-file-full-name>.jar -<your parameters if any>`

## Parameters

| Parameters | Details |
|---|---|
| role | Is what tells the selenium which it was `hub` or `node` |
| port | This is to specify which port the `hub` or `node` should be listening. |
| hub | This parameter is used in `node` to specify the hub url |
| browserName | Its been used in `node` to specify the browser name like firefox, chrome, internet explorer |
| maxInstances | Its where the instance of the browser is being specified eg. 5 means there will be 5 instance of the browser which user specified will be present. |
| nodeConfig | A Json configuration file for the node. You can specify the role, port etc. in here |
| hubConfig | A Json configuration file for the node. You can specify the role, port, max instances etc. in here |

## Examples

### Java code for Selenium Grid

```
String hubUrl = "http://localhost:4444/wd/hub"
```

```
DesiredCapabilities capability = DesiredCapabilities.firefox(); //or which browser you want
RemoteWebDriver driver = new RemoteWebDriver(hubUrl, capability);
```

### Creating a Selenium Grid hub and node

# Creating a hub

A quick configuration for a hub and node setup in selenium grid. For more information see: Grid 2 docs

## Requirements

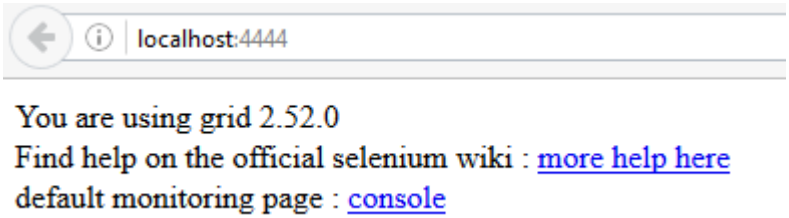
To set up a grid hub you need the flowing:

- Selenium-server-standalone-.jar

## Creating the hub

To Create a Hub you need to run the selenium server.

1. Download Selenium-server-standalone-.jar
2. Open your terminal and navigate to the folder where Selenium-server-standalone-.jar is
3. Execute the flowing command:
    1. For default configuration `java -jar selenium-server-standalone-<Version>.jar -role hub`
    2. For Json configuration `java -jar selenium-server-standalone-<Version>.jar -role hub -hubConfig hubConfig.json`
4. Open http://localhost:4444/ you will see a message a follows




On clicking `console` -> `View config` for to view the Configuration for the hub details.

# Creating a Node

## Requirements

To set up a grid hub you need the flowing:

- Selenium-server-standalone-.jar
- Webdrivers

    - Chrome driver
    - FireFox driver
    - Microsoft Edge driver
- Browsers
    - Chrome
    - FireFox
    - Microsoft Edge (Windows 10)

# Creating the Node

Now To create Nodes for the Hub

1. Download Selenium-server-standalone-.jar
2. Download the browsers you want to test in
3. Download the drivers for the browsers you want to test in
4. Open new terminal and navigate to the selenium server jar file location
5. Execute the folowing command:
    1. for default configuration `java -jar selenium-server-standalone-<VERSION NUMBER>.jar -role node`
    2. For Json configuration `java -jar selenium-server-standalone-<Version>.jar -role node -nodeConfig nodeConfig.json`
6. Now go to http://localhost:4444/grid/console to view the node details

## Configuragtion via Json

An example configuration for a hub:

```
java -jar selenium-server-standalone-<version>.jar -role hub -hubConfig hubConfig.json
```

```
{
    "_comment" : "Configuration for Hub - hubConfig.json",
    "host": ip,
    "maxSessions": 5,
    "port": 4444,
    "cleanupCycle": 5000,
    "timeout": 300000,
    "newSessionWaitTimeout": -1,
    "servlets": [],
    "prioritizer": null,
    "capabilityMatcher": "org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
    "throwOnCapabilityNotPresent": true,
    "nodePolling": 180000,
    "platform": "WINDOWS"
}
```

An example configuration for a node

```
java -jar selenium-server-standalone-<version>.jar -role node -nodeConfig nodeConfig.json
```

```
{
  "capabilities":
  [
```

```
    {
      "browserName": "opera",
      "platform": "WINDOWS",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver",
      "webdriver.opera.driver": "C:/Selenium/drivers/operadriver.exe",
      "binary":"C:/Program Files/Opera/44.0.2510.1159/opera.exe"
    },
    {
      "browserName": "chrome",
      "platform": "WINDOWS",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver",
      "webdriver.chrome.driver": "C:/Selenium/drivers/chromedriver.exe",
      "binary":"C:/Program Files/Google/Chrome/Application/chrome.exe"
    },
    {
      "browserName": "firefox",
      "platform": "WINDOWS",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver",
      "webdriver.gecko.driver": "C:/Selenium/drivers/geckodriver.exe",
      "binary":"C:/Program Files/Mozilla Firefox/firefox.exe"
    }
  ],
  "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
  "maxSession": 5,
  "port": 5555,
  "register": true,
  "registerCycle": 5000,
  "hub": "http://localhost:4444",
  "nodeStatusCheckTimeout": 5000,
  "nodePolling": 5000,
  "role": "node",
  "unregisterIfStillDownAfter": 60000,
  "downPollingLimit": 2,
  "debug": false,
  "servlets" : [],
  "withoutServlets": [],
  "custom": {}
}
```

Read Selenium Grid Configuration online: https://riptutorial.com/selenium-webdriver/topic/2504/selenium-grid-configuration

# Chapter 23: Selenium-webdriver with Python, Ruby and Javascript along with CI tool

## Introduction

This is one way of running selenium tests with CircleCI

## Examples

### CircleCI integration with Selenium Python and Unittest2

Circle.yml

```
machine:
  python:
    # Python version to use - Selenium requires python 3.0 and above
    version: pypy-3.6.0
dependencies:
    pre:
        # Install pip packages
        - pip install selenium
        - pip install unittest
test:
  override:
    # Bash command to run main.py
    - python main.py
```

main.py

```
import unittest2

# Load and run all tests in testsuite matching regex provided
loader = unittest2.TestLoader()
# Finds all the tests in the same directory that have a filename that ends in test.py
testcases = loader.discover('.', pattern="*test.py")
test_runner = unittest2.runner.TextTestRunner()
# Checks that all tests ran
success = test_runner.run(testcases).wasSuccessful()
```

example_test.py

```
class example_test(unittest.TestCase):
    def test_something(self):
        # Make a new webdriver instance
        self.driver = webdriver.Chrome()
        # Goes to www.goooogle.com
        self.driver.get("https://www.google.com")
```

Read Selenium-webdriver with Python, Ruby and Javascript along with CI tool online:

https://riptutorial.com/selenium-webdriver/topic/10091/selenium-webdriver-with-python--ruby-and-javascript-along-with-ci-tool

# Chapter 24: Setting / Getting Browser window size

## Introduction

Setting or getting window size of any browser during automation

## Syntax

- driver.manage().window().maximize();
- driver.manage().window().setSize(*DimensionObject*);
- driver.manage().window().getSize()

## Examples

### JAVA

Set to maximum Browser window size :

```
//Initialize Browser
System.setProperty("webdriver.gecko.driver", "E:\\path\\to\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("https://www.google.com/");

//Set Browser window size
driver.manage().window().maximize();
```

Set specific window size :

```
//Initialize Browser
System.setProperty("webdriver.gecko.driver", "E:\\path\\to\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("https://www.google.com/");

//Initialize Dimension class object and set Browser window size
org.openqa.selenium.Dimension d = new org.openqa.selenium.Dimension(400, 500);
driver.manage().window().setSize(d);
```

Get Browser window size :

```
//Initialize Browser
System.setProperty("webdriver.gecko.driver", "E:\\path\\to\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("https://www.google.com/");

//Get Browser window size and print on console
System.out.println(driver.manage().window().getSize());
```

Read Setting / Getting Browser window size online: https://riptutorial.com/selenium-webdriver/topic/10093/setting---getting-browser-window-size

# Chapter 25: Switching Frames

## Syntax

- **Java**
- driver.switchTo().frame(String name);
- driver.switchTo().frame(String id);
- driver.switchTo().frame(int index);
- driver.switchTo().frame(WebElement frameElement);
- driver.switchTo().defaultContent();
- **C#**
- driver.SwitchTo().Frame(int frameIndex);
- driver.SwitchTo().Frame(IWebElement frameElement);
- driver.SwitchTo().Frame(string frameName);
- driver.SwitchTo().DefaultContent();
- **Python**
- driver.switch_to_frame(nameOrId)
- driver.switch_to.frame(nameOrId)
- driver.switch_to_frame(index)
- driver.switch_to.frame(index)
- driver.switch_to_frame(frameElement)
- driver.switch_to.frame(frameElement)
- driver.switch_to_default_content()
- driver.switch_to.default_content()
- **JavaScript**
- driver.switchTo().frame(nameOrId)
- driver.switchTo().frame(index)
- driver.switchTo().defaultContent()

## Parameters

| parameter | details |
|---|---|
| nameOrId | Select a frame by its name of id. |
| index | Select a frame by its zero-based index. |
| frameElement | Select a frame using its previously located WebElement |

## Examples

### To switch to a frame using Java

For an instance, if the html source code of an html view or element is wrapped by an iframe like this:

```
<iframe src="../images/eightball.gif" name="imgboxName" id="imgboxId">
   <p>iframes example</p>
   <a href="../images/redball.gif" target="imgbox">Red Ball</a>
</iframe><br />
```

... then to perform any action on the web-elements of the iframe, you have to switch the focus to the iframe first, using any one of the below methods:

**Using frame Id** (should be used only if you know the id of the iframe).

```
driver.switchTo().frame("imgboxId"); //imgboxId - Id of the frame
```

**Using frame name** (should be used only if you know the name of the iframe).

```
driver.switchTo().frame("imgboxName"); //imgboxName - Name of the frame
```

**Using frame index** (should be used only if you do not have the id or name of the iframe), where the index defines the position of the iframe amongst all frames.

```
driver.switchTo().frame(0); //0 - Index of the frame
```

Note: If you have three frames in the page, then the first frame will be at index 0, the second at index 1, and the third at index 2.

**Using previously located webelement** (should be used only if you have already located the frame and have returned it as a `WebElement`).

```
driver.switchTo().frame(frameElement); //frameElement - webelement that is the frame
```

So, to click on the `Red Ball` anchor:

```
driver.switchTo().frame("imgboxId");
driver.findElement(By.linkText("Red Ball")).Click();
```

## To get out of a frame using Java

To switch focus to either main document or first frame of the page. You have to use below syntax.

```
driver.switchTo().defaultContent();
```

## Switch to a frame using C#

### 1. Switch to a frame by Index.

Here we are switching to index 1. Index refers to the order of frames on the page. This should be

used as a last resort, as frame id or names are much more reliable.

```
driver.SwitchTo().Frame(1);
```

**2. Switch to a frame by Name**

```
driver.SwitchTo().Frame("Name_Of_Frame");
```

**3. Switch to a frame by Title, Id, or others by passing IWebElement**

If you want to switch to a frame by id or title you have to pass in a web element as a parameter:

```
driver.SwitchTo().Frame(driver.FindElement(By.Id("ID_OF_FRAME")));
driver.SwitchTo().Frame(driver.FindElement(By.CssSelector("iframe[title='Title_of_Frame']")));
```

Also note that if your frame takes a few seconds to come up, you may have to use a wait:

```
new WebDriverWait(driver, TimeSpan.FromSeconds(10))
    .Until(ExpectedConditions.ElementIsVisible(By.Id("Id_Of_Frame")));
```

**Get out of a frame:**

```
driver.SwitchTo().DefaultContent()
```

## To get out of a frame using C#

To switch focus to either main document or first frame of the page. You have to use below syntax.

webDriver.SwitchTo().DefaultContent();

## Switch among Child Frames of a Parent Frame.

Consider you have a Parent Frame(Frame-Parent). and 2 child frames(Frame_Son,Frame_Daughter). Lets see various conditions and how to handle.

1.From Parent to Son or Daughter:

```
 driver.switchTo().frame("Frame_Son");
 driver.switchTo().frame("Frame_Daughter");
```

2.From Son to Parent: If parent is default frame, switch to default frame, else from default frame switch to parent frame. But you cannot switch directly from son to Parent.

```
driver.switchTo().defaultContent();
driver.switchTo().frame("Frame_Parent");
```

3.From Son to Daughter: If your sister does some mistake don't yell at her, just reach out to your Parent. Similarly, you give control to parent frame and then to daughter frame.

```
driver.switchTo().defaultContent();
driver.switchTo().frame("Frame_Parent");
driver.switchTo().frame("Frame_Daughter");
```

### Wait for your frames to load

In quite a few cases your frame might not show up immediately and you probably have to wait till it is loaded to switch. Or else you will have NoSuchFrameException.

So its always a good choice to wait before you switch. Following is a ideal way to wait till a frame is loaded.

```
try{
        new WebDriverWait(driver, 300).ignoring(StaleElementReferenceException.class).
                        ignoring(WebDriverException.class).

until(ExpectedConditions.visibilityOf((driver.findElement(By.id("cpmInteractionDivFrame")))));}
    catch{
```

// throws exception only if your frame is not visible with in your wait time 300 seconds }

Read Switching Frames online: https://riptutorial.com/selenium-webdriver/topic/4589/switching-frames

# Chapter 26: Taking Screenshots

## Introduction

Taking Screenshots and saving in a particular path

## Syntax

- File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
- FileUtils.copyFile(src, new File("D:\screenshot.png"));

## Examples

### JAVA

Code to take and save screenshot :

```
public class Sample
{
    public static void main (String[] args)
    {
        *//Initialize Browser*
        System.setProperty("webdriver.gecko.driver", "**E:\\path\\to\\geckodriver.exe**");
        WebDriver driver = new FirefoxDriver();
        driver.manage().window().maximize();
        driver.get("https://www.google.com/");

        //Take Screesnshot
        File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
        try {
            //Save Screenshot in destination file
            FileUtils.copyFile(src, new File("D:\\screenshot.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Takes Screenshot :

```
File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
```

Stores Screenshot from source to destination :

```
FileUtils.copyFile(src, new File("D:\\screenshot.png"));
```

Read Taking Screenshots online: https://riptutorial.com/selenium-webdriver/topic/10094/taking-screenshots

# Chapter 27: Using @FindBy annotations in Java

## Syntax

- CLASS_NAME: @FindBy(className = "classname")
- CSS: @FindBy(css = "css")
- ID: @FindBy(id = "id")
- ID_OR_NAME: @FindBy(how = How.ID_OR_NAME, using ="idOrName")
- LINK_TEXT: @FindBy(linkText= "text")
- NAME: @FindBy(name= "name")
- PARTIAL_LINK_TEXT: @FindBy(partialLinkText= "text")
- TAG_NAME: @FindBy(tagName="tagname")
- XPATH: @FindBy(xpath="xpath")

## Remarks

Note that there are two ways of using the annotation. Examples:

```
@FindBy(id = "id")
```

and

```
@FindBy(how = How.ID, using  ="id")
```

are equal and both look for element by its ID. In case of `ID_OR_NAME` you can only use

```
@FindBy(how = How.ID_OR_NAME, using ="idOrName")
```

`PageFactory.initElements()` must be used after page object instantiation to find elements marked with `@FindBy` annotation.

## Examples

### Basic example

Assume that we are using Page object model. Page Object class:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
```

```
public class LoginPage {

    @FindBy(id="loginInput") //method used to find WebElement, in that case Id
    WebElement loginTextbox;

    @FindBy(id="passwordInput")
    WebElement passwordTextBox;

    //xpath example:
    @FindBy(xpath="//form[@id='loginForm']/button(contains(., 'Login')")
    WebElement loginButton;

    public void login(String username, String password){
        // login method prepared according to Page Object Pattern
        loginTextbox.sendKeys(username);
        passwordTextBox.sendKeys(password);
        loginButton.click();
        // because WebElements were declared with @FindBy, we can use them without
        // driver.find() method
    }
}
```

And Tests class:

```
class Tests{
    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        LoginPage loginPage = new LoginPage();

        //PageFactory is used to find elements with @FindBy specified
        PageFactory.initElements(driver, loginPage);
        loginPage.login("user", "pass");
    }
}
```

There are few methods to find WebElements using @FindBy - check Syntax section.

Read Using @FindBy annotations in Java online: https://riptutorial.com/selenium-webdriver/topic/6777/using--findby-annotations-in-java

# Chapter 28: Using Selenium Webdriver with Java

## Introduction

Selenium webdriver is web automation framework which allows you to test your web application against different web browsers. Unlike Selenium IDE, webdriver allows you to develop your own test cases in programming language of your choice. It supports Java, .Net, PHP, Python, Perl, Ruby.

## Examples

### Opening browser window with specific URL using Selenium Webdriver in Java

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

class test_webdriver{
    public static void main(String[] args) {
       WebDriver driver = new FirefoxDriver();
       driver.get("http://stackoverflow.com/");
       driver.close();
    }
}
```

### Opening a browser window with to() method

Opening a browser with to() method.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
class navigateWithTo{
    public static void main(String[] args) {
       WebDriver driver = new FirefoxDriver();
       driver.navigate().to("http://www.example.com");
       driver.close();
    }
}
```

Read Using Selenium Webdriver with Java online: https://riptutorial.com/selenium-webdriver/topic/9158/using-selenium-webdriver-with-java

# Chapter 29: Wait

## Examples

### Types of Wait in Selenium WebDriver

While running any web application it's necessary to take loading time into consideration. If your code tries to access any element that is not yet loaded, WebDriver will throw an exception and your script will stop.

There are three types of Waits -

- **Implicit Waits**
- **Explicit Waits**
- **Fluent Waits**

Implicit waits are used to set the waiting time throughout the program, while explicit waits are used only on specific portions.

---

## Implicit Wait

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. Implicit waits are basically your way of telling WebDriver the latency that you want to see if specified web element is not present that WebDriver is looking for. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance. Implicit wait is declared in the instantiation part of the code using the following snippet.

Example in **Java**:

```
driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
// You need to import the following class - import java.util.concurrent.TimeUnit;
```

Example in **C#**:

```
driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(15));
```

So in this case, you are telling WebDriver that it should wait 15 seconds in cases of specified element is not available on the UI (DOM).

---

## Explicit wait

You may encounter instances when some element takes more time to load. Setting implicit wait for

such cases doesn't make sense as browser will wait unnecessarily for the same time for every element, increasing the automation time. Explicit wait helps here by bypassing implicit wait altogether for some specific elements.

Explicit waits are intelligent waits that are confined to a particular web element. Using explicit waits you are basically telling WebDriver at the max it is to wait for X units of time before it gives up.

Explicit waits are done using the WebDriverWait and ExpectedConditions classes. In the below example, we shall wait up to 10 seconds for an element whose id is username to become visible before proceeding to the next command. Here are the steps.

Example in **Java**:

```
//Import these two packages:
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

//Declare a WebDriverWait variable. In this example, we will use myWaitVar as the name of the
variable.
WebDriverWait myWaitVar = new WebDriverWait(driver, 30);

//Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur.
In this case, we will use explicit wait on the username input before we type the text tutorial
onto it.
myWaitVar.until(ExpectedConditions.visibilityOfElementLocated(By.id(“username”)));
driver.findElement(By.id(“username”)).sendKeys(“tutorial”);
```

ExpectedConditions class has some predefined common conditions to wait for an element. Click here to see list of these conditions in Java binding.

Example in **C#**:

```
using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;
using OpenQA.Selenium.PhantomJS;

// You can use any other WebDriver you want, such as ChromeDriver.
using (var driver = new PhantomJSDriver())
{
    driver.Navigate().GoToUrl("http://somedomain/url_that_delays_loading");

    // We aren't going to use it more than once, so no need to declare this a variable.
    new WebDriverWait(driver, TimeSpan.FromSeconds(10))
        .Until(ExpectedConditions.ElementIsVisible(By.Id("element-id")));

    // After the element is detected by the previous Wait,
    // it will display the element's text
    Console.WriteLine(driver.FindElement(By.Id("element-id")).Text);
}
```

In this example, system will wait for 10 seconds until the element is visible. If the element will not be visible after the timeout, the WebDriver will throw a `WebDriverTimeoutException`.

*Please note: If the element is visible before the 10 second timeout, system will immediately*

*proceed for further process.*

---

# Fluent wait

Unlike implicit and explicit wait, fluent wait uses two parameters. Timeout value and polling frequency. Let's say we have timeout value as 30 seconds and polling frequency as 2 seconds. WebDriver will check for element after every 2 seconds until timeout value (30 seconds). After timeout value is exceeded without any result, exception is thrown. Below is a sample code which shows implementation of fluent wait.

Example in **Java**:

```
Wait wait = new FluentWait(driver).withTimeout(30, SECONDS).pollingEvery(2,
SECONDS).ignoring(NoSuchElementException.class);


WebElement testElement = wait.until(new Function() {
    public WebElement apply(WebDriver driver) {
        return driver.findElement(By.id("testId"));
    }
});
```

Another advantage of using fluent wait is, we can ignore specific types of exceptions (Eg. NoSuchElementExceptions) while waiting. Due to all these provisions, fluent wait is helpful in AJAX applications as well as in scenarios when element load time fluctuates often. Strategic use of fluent wait significantly improves automation efforts.

---

### Different types of explicit wait conditions

In explicit wait, you expect for a condition to happen. For example you want to wait until an element is clickable.

Here is a demonstration of a few common problems.

*Please note: In all of these examples you can use any* `By` *as a locator, such as* `classname`, `xpath`, `link text`, `tag name` *or* `cssSelector`

---

# Wait until element is visible

For example, if your website takes some time to load, you can wait until the page completes loading, and your element is visible to the WebDriver.

*C#*

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.ElementIsVisible(By.Id("element-id")));
```

*Java*

```
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("element-id")));
```

---

## Wait until element is not visible anymore

Same as before, but reversed.

*C#*

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.InvisibilityOfElementLocated(By.Id("element-id")));
```

*Java*

```
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.invisibilityOfElementLocated(By.id("element-id")));
```

---

## Wait until text is present in the specified element

*C#*

```
IWebElement element = driver.FindElement(By.Id("element-id"));

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.TextToBePresentInElement(element, "text"));
```

*Java*

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

WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.textToBePresentInElement(element, "text"));
```

If you go to the given link above, you will see all the wait condition there.

The difference between the usage of these wait conditions are in their input parameter.

That means you need to pass the WebElement if its input parameter is WebElement, you need to pass the element locator if it takes the By locator as its input parameter.

Choose wisely what kind of wait condition you want to use.

### Waiting For Ajax Requests to Complete

# C#

```
using OpenQA.Selenium
using OpenQA.Selenium.Chrome;
using System.Threading;

namespace WebDriver Tests
{
    class WebDriverWaits
    {
        static void Main()
        {
            IWebDriver driver = new ChromeDriver(@"C:\WebDriver");

            driver.Navigate().GoToUrl("page with ajax requests");
            CheckPageIsLoaded(driver);

            // Now the page is fully loaded, you can continue with further tests.
        }

        private void CheckPageIsLoaded(IWebDriver driver)
        {
            while (true)
            {
                bool ajaxIsComplete = (bool)(driver as
IJavaScriptExecutor).ExecuteScript("return jQuery.active == 0");
                if (ajaxIsComplete)
                    return;
                Thread.Sleep(100);
            }
        }
    }
}
```

This example is useful for pages where ajax requests are made, here we use the `IJavaScriptExecutor` to run our own JavaScript code. As it is within a `while` loop it will continue to run until `ajaxIsComplete == true` and so the return statement is executed.

We check that all ajax requests are complete by confirming that `jQuery.active` is equal to `0`. This works because each time a new ajax request is made `jQuery.active` is incremented and each time a request complements it is decremented, from this we can deduce that when `jQuery.active == 0` all ajax requests must be complete.

## Fluent Wait

Fluent wait is a superclass of **explicit** wait (`WebDriverWait`) that is more configurable since it can accept an argument to the wait function. I'll pass on **implicit** wait, since it's a best practice to avoid it.

Usage (Java):

```
Wait wait = new FluentWait<>(this.driver)
        .withTimeout(driverTimeoutSeconds, TimeUnit.SECONDS)
```

```
        .pollingEvery(500, TimeUnit.MILLISECONDS)
        .ignoring(StaleElementReferenceException.class)
        .ignoring(NoSuchElementException.class)
        .ignoring(ElementNotVisibleException.class);

WebElement foo = wait.until(ExpectedConditions.presenceOfElementLocated(By.yourBy));

// or use your own predicate:
WebElement foo = wait.until(new Function() {
  public WebElement apply(WebDriver driver) {
    return element.getText().length() > 0;
  }
});
```

When you use Explicit wait with it's defaults it's simply a `FluentWait<WebDriver>` with defaults of: `DEFAULT_SLEEP_TIMEOUT = 500;` and ignoring `NotFoundException`.

## Fluent wait

Each FluentWait instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as NoSuchElementExceptions when searching for an element on the page.It is associated with the driver.

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
.withTimeout(30, SECONDS) //actuall wait for the element to pe present
.pollingEvery(5, SECONDS) //selenium will keep looking for the element after every 5seconds
.ignoring(NoSuchElementException.class); //while ignoring this condition
wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("username"));
```

Read Wait online: https://riptutorial.com/selenium-webdriver/topic/4435/wait

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with selenium-webdriver | Abhilash Gupta, Alice, Community, Eugene S, iamdanchiv, Jakub Lokša, Josh, Kishor, Michal, Mohit Tater, Pan, Priyanshu Shekhar, rg702, Santoshsarma, Tomislav Nakic-Alfirevic, vikingben |
| 2 | Actions (Emulating complex user gestures) | Josh, Kenil Fadia, Liam, Priyanshu Shekhar, Tom Mc |
| 3 | Basic Selenium Webdriver Program | Jakub Lokša, Josh, Liam, Priya, Sudha Velan, Thomas, vikingben |
| 4 | Error Handling in Automation using Selenium | Andersson |
| 5 | Exceptions in Selenium-WebDriver | Brydenr |
| 6 | Executing Javascript in the page | Brydenr, Liam |
| 7 | Handle an alert | Andersson, Aurasphere, Priya, SlightlyKosumi |
| 8 | Headless Browsers | Abhilash Gupta, Jakub Lokša, Liam, r_D, Tomislav Nakic-Alfirevic |
| 9 | HTML Reports | Ashish Deshmukh |
| 10 | Interacting with the Browser Window(s) | Andersson, Josh, Sakshi Singla |
| 11 | Interaction With Web Element | Jakub Lokša, Liam, Moshisho, Siva, Sudha Velan |
| 12 | Listeners | Erki M. |
| 13 | Locating Web Elements | alecxe, daOnlyBG, Jakub Lokša, Josh, Liam, Łukasz Piaszczyk, Moshisho, NarendraR, noor, Priya, Sakshi Singla, Siva |
| 14 | Navigate between multiple frames | Pavan T, Raghvendra |

| | | |
|---|---|---|
| 15 | Navigation | Andersson, Liam, Santoshsarma, viralpatel |
| 16 | Page Object Model | JeffC, Josh, Moshisho, Priyanshu Shekhar, Sakshi Singla |
| 17 | Robot In Selenium | Priyanshu Shekhar |
| 18 | Scrolling | Andersson, Sagar007 |
| 19 | Select Class | Gaurav Lad, Liam |
| 20 | Selenium e2e setup | Ashish Deshmukh |
| 21 | Selenium Grid | Eugene S, Y-B Cause |
| 22 | Selenium Grid Configuration | mnoronha, Prasanna Selvaraj, selva, Thomas |
| 23 | Selenium-webdriver with Python, Ruby and Javascript along with CI tool | Brydenr |
| 24 | Setting / Getting Browser window size | Abhilash Gupta |
| 25 | Switching Frames | Andersson, dreamwork801, Jakub Lokša, Java_deep, Jim Ashworth, Karthik Taduvai, Kyle Fairns, Liam, lloyd, Priyanshu Shekhar, SlightlyKosumi |
| 26 | Taking Screenshots | Abhilash Gupta, Sanchit |
| 27 | Using @FindBy annotations in Java | Alex Wittig, Łukasz Piaszczyk |
| 28 | Using Selenium Webdriver with Java | r_D, the_coder |
| 29 | Wait | Jakub Lokša, Josh, Kenil Fadia, Liam, Moshisho, noor, Sajal Singh, Saurav |