**Selenium Content**

**1. Why WebDriver?What is the difference between Selenium RC and WD?**

WebDriver is a web automation framework that allows you to execute your tests against different browsers, not just Firefox, Chrome (unlike Selenium IDE).

WebDriver also enables you to use a programming language in creating your test scripts (not possible in Selenium IDE).

You can now use conditional operations like if-then-else or switch-case. You can also perform looping like do-while.

## **Difference between Selenium RC and Webdriver**

Before the advent of WebDriver in  2006, there was another, automation tool called Selenium Remote Control. Both WebDriver and Selenium RC have following features:

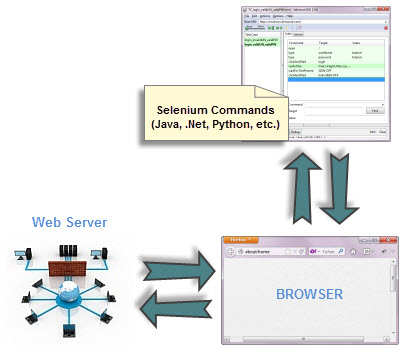
* They both allow you to use a programming language in designing your test scripts.
* They both allow you to run your tests against different browsers.

So how do they differ? Let us discuss the answers.

## 1. Architecture

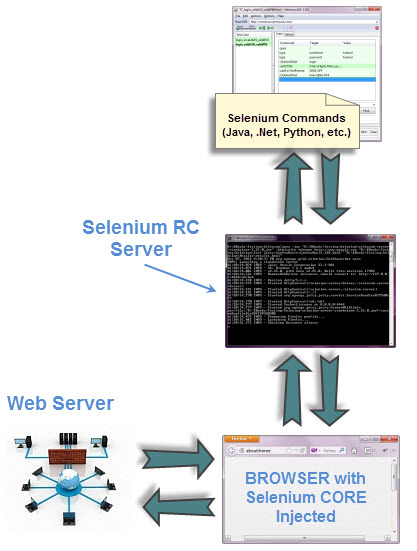
WebDriver's architecture is simpler than Selenium RC's.

* It controls the browser from the OS level
* All you need are your programming language's IDE (which contains your Selenium commands) and a browser.



##### Selenium RC's architecture is way more complicated.

* You first need to launch a separate application called Selenium Remote Control (RC) Server before you can start testing
* The Selenium RC Server acts as a "middleman" between your Selenium commands and your browser
* When you begin testing, Selenium RC Server "injects" a Javascript program called Selenium Core into the browser.
* Once injected, Selenium Core will start receiving instructions relayed by the RC Server from your test program.
* When the instructions are received, Selenium Core will execute them as Javascript commands.
* The browser will obey the instructions of Selenium Core and will relay its response to the RC Server.
* The RC Server will receive the response of the browser and then display the results to you.
* RC Server will fetch the next instruction from your test script to repeat the whole cycle.



## 2. Speed

WebDriver is faster than Selenium RC since it speaks directly to the browser uses the browser's own engine to control it.

Selenium RC is slower since it uses a Javascript program called Selenium Core. This Selenium Core is the one that directly controls the browser, not you.

## 3. Real-life Interaction

WebDriver interacts with page elements in a more realistic way. For example, if you have a disabled text box on a page you were testing, WebDriver really cannot enter any value in it just as how a real person cannot.

Selenium Core, just like other JavaScript codes, can access disabled elements. In the past, Selenium testers complain that Selenium Core was able to enter values to a disabled text box in their tests. Differences in API

**Sample programs in web driver**

**WebElement is a selenium webdriver class using which we opreate on html elements such as described below.**

**1.** Handel Text box

**WebElement empName = driver.findElement(By.id("name"));//the ‘name’ locator is used to control the text box.**

**empName**.sendKeys("selenium@gmail.com"); //here sendKeys used to fill the text

2.Handel HyperLink

**WebElement anchr1=driver.findElement(By.linkText("click here"));//anchor links**

**with text ‘click here’**

**is controlled here**

**System.out.println(anchr1.getText())// displays the content of the link**

**3.**Handel Button and Check box

**WebElement** saveEmp = driver.findElement(**By**.id("submitEmp"));//botton is controlled using webdriver

WebElement chk1 = driver.findElement(By.id("chkId")); //check box is controled

**chk1.click()**;//here check box is checked automated way

4.Handel Radio Button and DropDown Controls

**Select st = new Select(driver.findElement(By.id("state")));//get hold on drop down**

**st.selectByVisibleText("Alabama");//getting the option value from the visibleText**

**st.selectByIndex(2);//getting the value from index of the option**

**WebElement rdId = driver.findElement(By.id("rdId")); //radio button**

**rdId.click();**

**5.Diplaying drop down values :--**

|  |
| --- |
| import org.openqa.selenium.support.ui.Select; |

|  |  |
| --- | --- |
|  | Select dropdown = new Select(driver.findElement(By.id("identifier"))); |

Let’s consider the following dropdown element for our example:

<select id="mySelect">

<option value="option1">France</option>

<option value="option2">Italy</option>

<option value="option3">Spain</option>

</select>

### **1. Identify the select HTML element:**

|  |  |
| --- | --- |
| 1  2 | WebElement mySelectElement = driver.findElement(By.id("mySelect"));  Select dropdown= new Select(mySelectElement); |

or pass it directly to the Select element:

|  |  |
| --- | --- |
| 1 | Select dropdown = new Select(driver.findElement(By.id("mySelect"))); |

### 2. To select an option you can do:

All select/deselect methods will throw NoSuchElementException if no matching option elements are found.  
Select by Visible Text (select all options that display text matching the argument):

|  |  |
| --- | --- |
|  | dropdown.selectByVisibleText("Italy"); |

or

Select by Index (select the option at the given index. This is done by examining the “index” attribute of an element, and not merely by counting):

|  |  |
| --- | --- |
|  | dropdown.selectByIndex(2); |

6.How to capture Screenshot in Selenium webdriver

For taking  screenshots Selenium has provided TakesScreenShot interface in this interface you can use getScreenshotAs method which will capture the entire screenshot in form of file then using FileUtils we can copy screenshots from one location to another location

Scenario – Open Google  and take screenshot

**Let’s implement the same:--**

**package com.ayantsoft.Selenium.webpage;**

**import java.io.File;**

**import java.io.IOException;**

**import org.apache.commons.io.FileUtils;**

**import org.openqa.selenium.OutputType;**

**import org.openqa.selenium.TakesScreenshot;**

**import org.openqa.selenium.WebDriver;**

**import org.openqa.selenium.firefox.FirefoxDriver;**

**import org.testng.annotations.Test;**

**public class ScreenshootGoogle {**

**public void TestJavaS1()**

**{**

**// Open Firefox**

**WebDriver driver=new FirefoxDriver();**

**// Maximize the window**

**driver.manage().window().maximize();**

**// Pass the url**

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

**// Take screenshot and store as a file format**

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

**try {**

**// now copy the screenshot to desired location using copyFile //method**

**FileUtils.copyFile(src, new File("/home/somnath/Desktop/selenium/error.png"));**

**}**

**catch (IOException e)**

**{**

**System.out.println(e.getMessage());**

**}**

**}**

**}**

**To Be able to run this program get the jar file of appache commons.io**

**7.Synchronization:--**

**Synchronization can be classified into two categories:**

1. Unconditional   
2. Conditional Synchronization

**Unconditional :**  
In this we just specify timeout value only. We will make the tool to wait until certain amount of time and then proceed further.

*Examples: Wait() and Thread.Sleep();*

The main disadvantage for the above statements are, there is a chance of unnecessary waiting time even though the application is ready.

The advantages are like in a situation where we interact for third party systems like interfaces, it is not possible to write a condition or check for a condition. Here in this situations, we have to make the application to wait for certain amount of time by specifying the timeout value.

**Conditional Synchronization:**

We specify a condition along with timeout value, so that tool waits to check for the condition and then come out if nothing happens.

It is very important to set the timeout value in conditional synchronization, because the tool should proceed further instead of making the tool to wait for a particular condition to satisfy.

1. 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.

The default setting is 0. Once when we define the implicit wait, it will set for the life of the WebDriver object instance.

It is a mechanism which will be written once and applied for entire session automatically. It should be applied immediately once we initiate the Webdriver.

Implicit wait will not work all the commands/statements in the application. It will work only for "FindElement" and "FindElements" statements.

If we set implicit wait, find element will not throw an exception if the element is not found in first instance, instead it will poll for the element until the timeout and then proceeds further. We should always remember to add the below syntax immediately below the Webdriver statement.

Syntax:

driver.manage.TimeOuts.implicitwait(6,Timeunit.SECONDS);

**Example using implicit timeout**

WebDriver driver = new FirefoxDriver();

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

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

Explicit Wait:

We need to define a wait statement for certain condition to be satisfied until the specified timeout period. If the Webdriver finds the element within the timeout period the code will get executed.

Explicit wait is mostly used when we need to Wait for a specific content/attribute change after performing any action, like when application gives AJAX call to system and get dynamic data and render on UI.

Example: Like there are drop-downs Country and State, based on the country value selected, the values in the state drop-down will change, which will take few seconds of time to get the data based on user selection.

Example:

/\*Explicit wait for state dropdown field\*/

WebDriverWait wait = new WebDriverWait(driver, 10);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("statedropdown")));

The above statement waits up to 10 seconds before throwing Exception (TimeoutException - Timed out after 10 seconds waiting for visibility of element) or if it finds the element, it will return in 0 - 10 seconds.