**Selenium RC (Remote Control)** is a test tool that allows you to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.

**Selenium RC** comes in two parts –

1. A server which automatically launches and kills browsers, and acts as a HTTP proxy for web requests from them.
2. Client libraries for your favorite computer language.

**Selenium Server**

1. Selenium Server receives Selenium commands from test program, interprets them, and reports back to program the results of running those tests.
2. The RC server bundles Selenium Core and automatically injects it into the browser. This occurs when test program opens the browser (using a client library API function). **Selenium-Core** is a JavaScript program, actually a set of JavaScript functions which interprets and executes **Selenese commands** using the browser’s built-in JavaScript interpreter.
3. The Server receives the **Selenese commands** from test program using simple **HTTP** GET/POST requests. This means you can use any programming language that can send HTTP requests to automate Selenium tests on the browser

**Selenium Grid**

**Selenium Grid** allows you to,

* Scale by distributing tests on several machines ( parallel execution )
* Manage multiple environments from a central point, making it easy to run the tests against a vast combination of browsers / OS.
* Minimize the maintenance time for the grid by allowing you to implement custom hooks to leverage virtual infrastructure for instance.

**Selenium Grid** allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, **Selenium Grid** support distributed test execution. It allows for running your tests in a distributed test execution environment.

* The default port used by the hub is 4444

**WebDriver:**

* The easiest way to set up a Selenium 2.0 Java project is to use **Maven**.
* using a maven pom.xml (project configuration) file
* <groupId>MySel20Proj</groupId>
* <artifactId>MySel20Proj</artifactId>
* A test suite is a collection of test cases which is displayed in the leftmost pane in the IDE.
* **Selenium commands**, often called **selenese,** **are the set of commands** that run tests. A sequence of these commands is a test script
* **POM** – Project Object Model

**Selenium commands** come in three “flavors”: **Actions, Accessors, and Assertions.**

* **Actions**are commands that generally **manipulate the state of the application**. They do things like “click this link” and “select that option”. If an Action fails, or has an error, the execution of the current test is stopped. Many Actions can be called with the “AndWait” suffix, e.g. “clickAndWait”. This suffix tells Selenium that the action will cause the browser to make a call to the server, and that Selenium should wait for a new page to load.
* **Accessors** examine the **state of the application and store the results in variables**, e.g. “**storeTitle**”. They are also used to automatically generate Assertions.
* **Assertions** are like Accessors, but they **verify that the state of the application conforms to what is expected**. Examples include “make sure the page title is X” and “verify that this checkbox is checked”.

All **Selenium Assertions** can be used in **3 modes**: “assert”, “verify”, and ” waitFor”. For example, you can “assertText”, “verifyText” and “waitForText”.

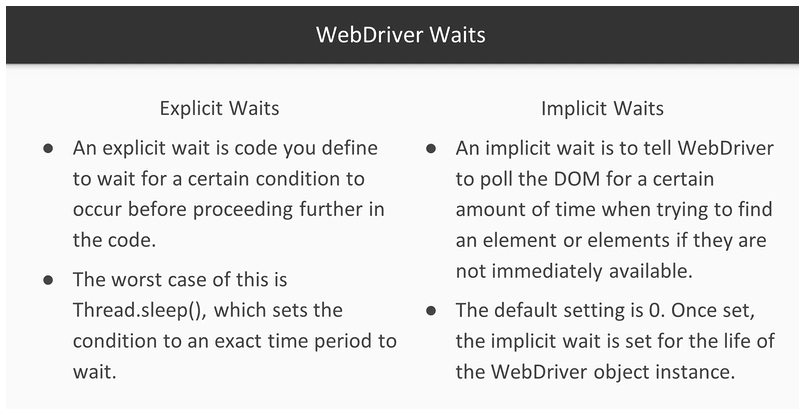
* **When an “assert” fails, the test is aborted.**
* **When a “verify” fails, the test will continue execution**, logging the failure. This allows a single “assert” to ensure that the application is on the correct page, followed by a bunch of “verify” assertions to test form field values, labels, etc.
* “**waitFor” commands wait for some condition to become true** (which can be useful for testing Ajax applications). They will succeed immediately if the condition is already true. However, they will fail and halt the test if the condition does not become true within the current timeout setting (see the setTimeout action below).
* **echo – The Selenese Print Command**
* There are three types of patterns: globbing, regular expressions, and exact.
* assertEquals**(actual,expected)**
* assertNotEquals**(actual,expected,Message);**
* assertTrue**(condition);**
* assertFalse**(condition);**
* assertNull**(object);**
* assertNotNull**(object);**

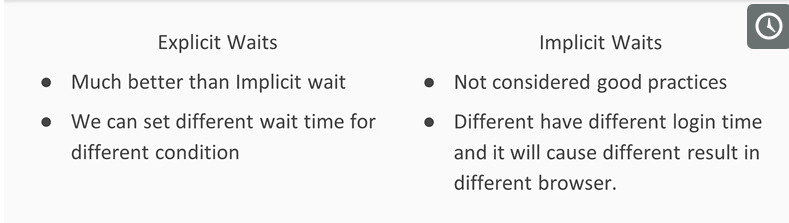
**Selenium IDE Locator**

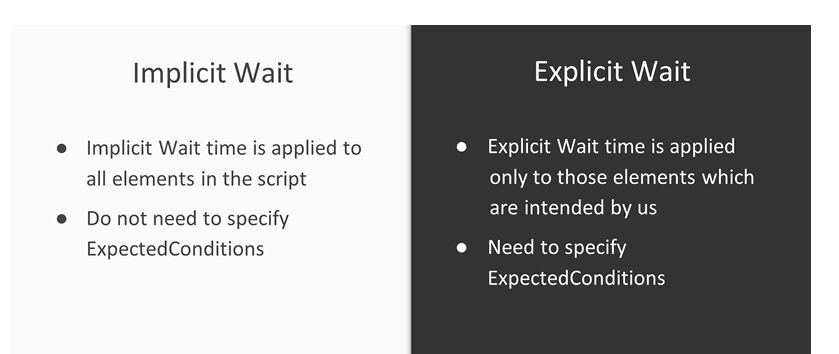
* **DOM -** Document Object Model represents an HTML document and can be accessed using JavaScript.
* **CSS:**CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents CSS uses Selectors for binding style properties to elements in the document
* **Locating Hyperlinks by Link Text**
* **Locating by XPath - XPath** is the language used for locating nodes in an XML document

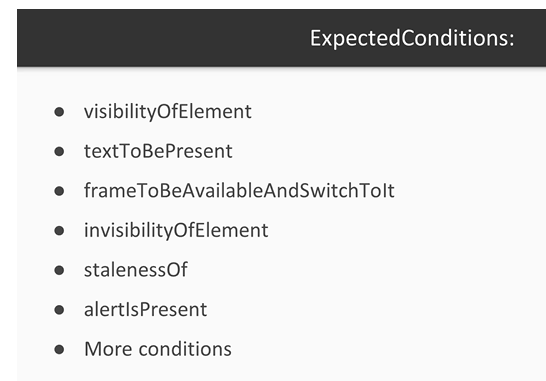
**WebDriver wait by default calls the ExpectedCodition every 500 milliseconds until it returns successfully.**

**Default polling interval is 500 ms**





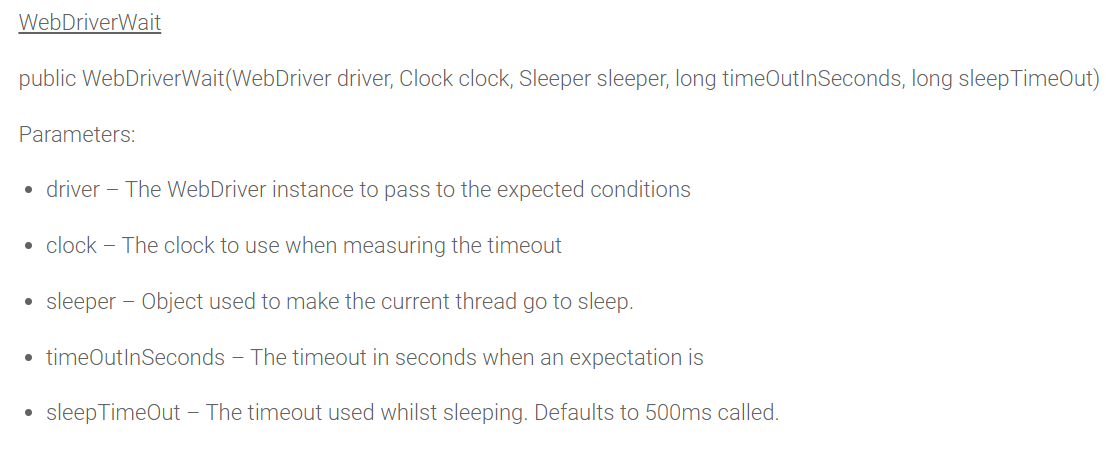


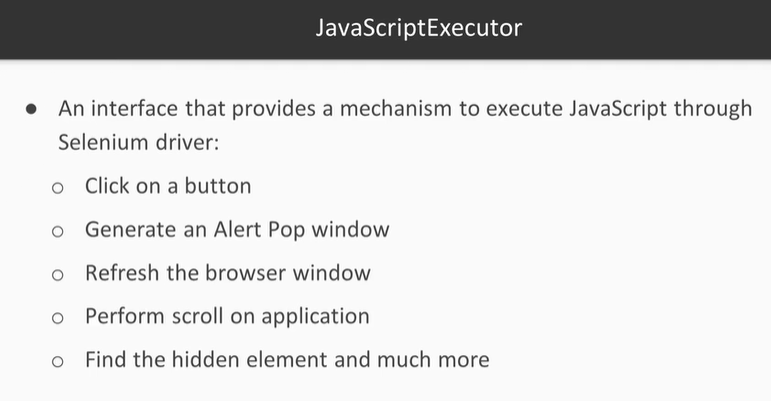


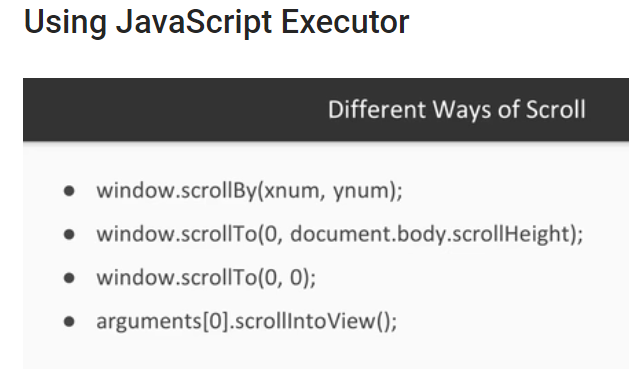
Explicit Wait Example:

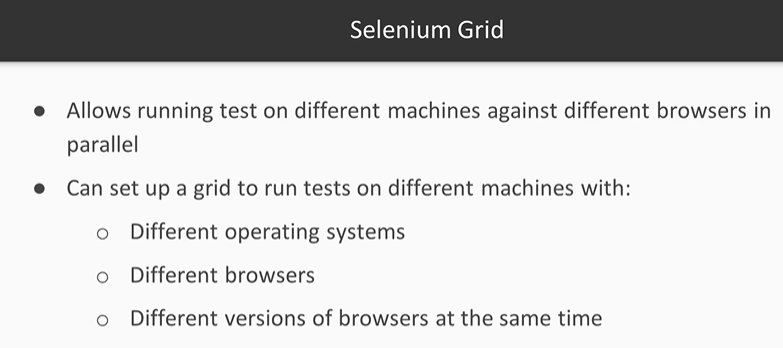


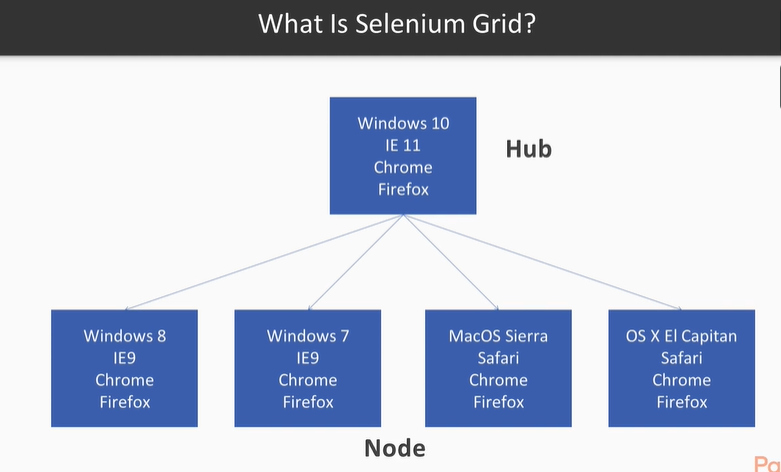
**WebDriver Wait**

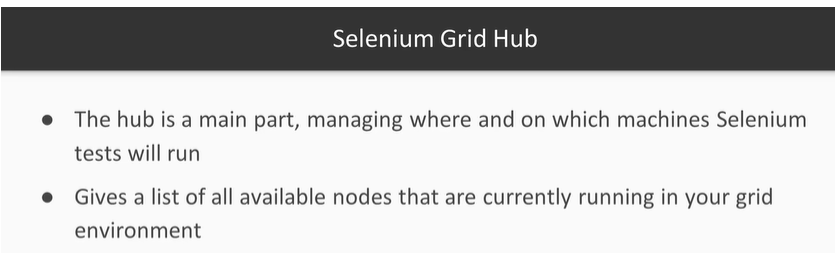


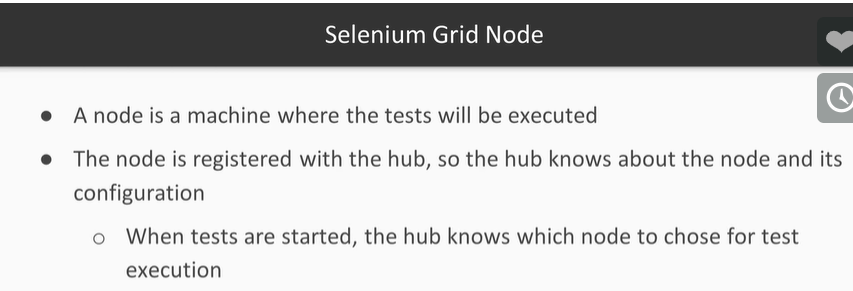


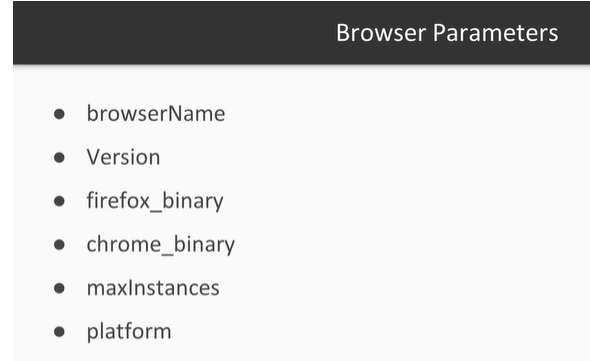












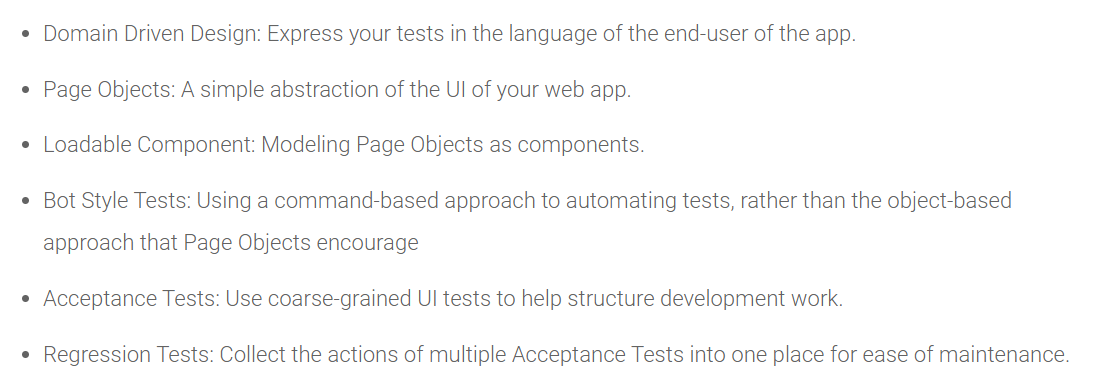
**If Browser parameter is not set then node will start 5 Firefox, 5 Chrome and 1 IE instance.**

Behavior vs. state testing

A test is a behavior test (also called interaction test) if it checks if certain methods were called with the correct input parameters. A behavior test does not validate the result of a method call.

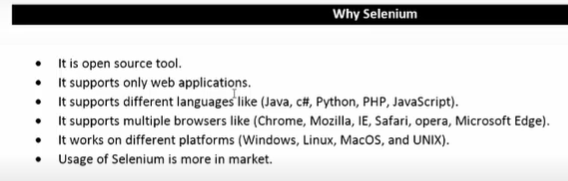
State testing is about validating the result. Behavior testing is about testing the behavior of the application under test.

If you are testing algorithms or system functionality, in most cases you may want to test state and not interactions. A typical test setup uses mocks or stubs of related classes to abstract the interactions with these other classes away afterwards you test the state or the behavior depending on your need.



Cascading Style Sheets (CSS) is a language specifically made for illustrating the appearance of various parts of a web page. CSS gives option to control the text color, fonts style, inter- paragraph spacing, columns sizing and layout, background images or colors and many other visual effects. CSS is used to style web pages, developed in HTML or XHTML

* The web page element that the browser formats also called as the selector.
* The actual formatting instructions also called as the declaration block or rules block.



**Selenium IDE default file format .HTML**

**Test Framework Advantage:**

will increase a team’s test speed and efficiency,

improve test accuracy, and

will reduce test maintenance costs as well as lower risks.

* Improved test efficiency
* Lower maintenance costs
* Minimal manual intervention
* Maximum test coverage
* Reusability of code

**Framework Types:**

* Linear Automation Framework or record-and-playback framework
* Modular Based Testing Framework
* Library Architecture Testing Framework
* Data-Driven Framework
* Keyword-Driven Framework
* Hybrid Testing Framework

The announcement for Selenium 3.0 official release was made on 13, Oct, 2016.

Major changes in Selenium 3.0 are

* Firefox is only fully supported at version 47.0.1 or earlier. Support for later versions of Firefox is provided by geckodriver, which is   based on the evolving W3C WebDriver spec, and uses the wire protocol   in that spec, which is liable to change without notice.
* You may wish to choose an ESR release such as 45.4.0esr or earlier.
* Firefox 47.0.0 is not supported at all.
* Stability fixes in Grid.
* All Grid nodes can now offer help.
* Updated to the latest version of HtmlUnitDriver.
* Re-enabled log gathering for the standalone server.
* Firefox profile is passed to both the legacy FirefoxDriver and geckodriver.

Some of the important features of Selenium 3 are

* Minimum version of Java required to run Selenium 3 is Java 8+
* You would need to use GeckoDriver if you want to run your scripts in Firefox versions greater than 47.0.1. Firefox 47.0.1 and before would not need GeckoDriver
* Apple has come up with its own SafariDriver to let you run your tests in Safari on Mac
* If you use selenium IDE to write code and export it in table format for execution purposes, you need to use a new test runner. Selenium Html Runner can be used for this purpose and is available in the download section of Selenium official website.

Java

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

we recommend starting directly with Selenium::WebDriver, and focusing on the two main classes, **Selenium::WebDriver::Driver** and **Selenium::WebDriver::Element**. This is the entry point to the whole WebDriver API.

To retrieve the innerText value contained within an element.

**element.getText();**

**To Delete Cookies in 3 ways:**

// By name

driver.manage().deleteCookieNamed(“CookieName”);

// By Cookie

driver.manage().deleteCookie(loadedCookie);

// Or all of them

driver.manage().deleteAllCookies();

Get Current URL Command

getCurrentUrl() : String – This method fetches the string representing the Current URL which is opened in the browser. Accepts nothing as a parameter and returns a String value.

Get Page Source Command

getPageSource() : String – This method returns the Source Code of the page. Accepts nothing as a parameter and returns a String value.

Close Command

close() : void – This method Close only the current window the WebDriver is currently controlling. Accepts nothing as a parameter and returns nothing.

Quit Command

quit() : void – This method Closes all windows opened by the WebDriver. Accepts nothing as a parameter and returns nothing.

Navigation commands

Navigate To Command

to(String arg0) : void – This method Loads a new web page in the current browser window. It accepts a String parameter and returns nothing.

Forward Command

forward() : void – This method does the same operation as clicking on the Forward Button of any browser. It neither accepts nor returns anything.

Back Command

back() : void – This method does the same operation as clicking on the Back Button of any browser. It neither accepts nor returns anything.

Refresh Command

refresh() : void – This method Refresh the current page. It neither accepts nor returns anything.

Void sendKeys(CharSequence… keysToSend) – Similar to the existing sendKeys() method in Webdriver

Void pressKey- Sends a key press only, without releasing it. Should only be implemented for modifier keys (Control, Alt and Shift)

Void releaseKey(Keys keyToRelease) – Releases a modifier key

DeSelect Methods

The way we select different values of DropDown & Multi Select, the same way we can also deselect the values. But the only challenge in these methods are they do not work for DropDown and only work for Multi Select elements.

* deselectAll( ) : void – Clear all selected entries. This is only valid when the SELECT supports multiple selections.
* deselectByIndex(int arg0) : void –Deselect the option at the given index.
* deselectByValue(String arg0) : void –Deselect all options that have a value matching the argument.
* deselectByVisibleText(String arg0) : void – Deselect all options that display text matching the argument.

# HtmlUnit Driver

This is currently the fastest and most lightweight implementation of WebDriver. As the name suggests, this is based on HtmlUnit. HtmlUnit is a java based implementation of a WebBrowser without a GUI. For any language binding (other than java) the Selenium Server is required to use this driver.

Usage, in Java language

WebDriver driver = new HtmlUnitDriver();

Pros

* Fastest implementation of WebDriver
* A pure Java solution and so it is platform independent.
* Supports JavaScript

Cons

* Emulates other browsers’ JavaScript behaviour

# Firefox Driver

In Java language

WebDriver driver = new FirefoxDriver();

Pros

* Runs in a real browser and supports JavaScript
* Faster than the Internet Explorer Driver

Cons

* Slower than the HtmlUnit Driver

# Gecko Driver

Selenium users must update to version 3.5 or later to use geckodriver

Selenium Support is best in Firefox 55 and greater

Windows XP support in Firefox was dropped with Firefox 53, we do not support this platform.

log object

| Name | Type | Description |
| --- | --- | --- |
| level | string | Set the level of verbosity of geckodriver and Firefox. Available levels are trace,  debug, config, info, warn, error, and fatal.  If left undefined the default is info. |

Port to use for the WebDriver server. Defaults to 4444.

Host to use for the WebDriver server. Defaults to 127.0.0.1.

# Internet Explorer Driver

WebDriver driver = new InternetExplorerDriver();

| Switch | Meaning |
| --- | --- |
| –port=<portNumber> | Specifies the port on which the HTTP server of the IE driver will listen for commands from language bindings. **Defaults to 5555.** |
| –host=<hostAdapterIPAddress> | Specifies the IP address of the host adapter on which the HTTP server of the IE driver will listen for commands from language bindings. **Defaults to 127.0.0.1.** |
| –log-level=<logLevel> | Specifies the level at which logging messages are output. Valid values are FATAL, ERROR, WARN, INFO, DEBUG, and TRACE. **Defaults to FATAL.** |
| –log-file=<logFile> | Specifies the full path and file name of the log file. Defaults to stdout. |
| –extract-path=<path> | Specifies the full path to the directory used to extract supporting files used by the server. Defaults to the TEMP directory if not specified. |
| –silent | Suppresses diagnostic output when the server is started. |

# Chrome Driver

The ChromeDriver consists of three separate pieces. There is the browser itself (“chrome”), the language bindings provided by the Selenium project (“the driver”) and an executable downloaded from the Chromium project which acts as a bridge between “chrome” and the “driver”.

In Java language

WebDriver driver = new ChromeDriver();

# Opera Driver

OperaDriver is used to test application on the Opera browser using WebDriver. OperaDriver is being developed by the Opera software. It uses the scope transport protocol to communicate between OperaDriver and the Opera browser.

WebDriver driver = new OperaDriver();

OperaChromiumDriver can be used without extra setup on Chromium-based versions of Opera starting from version 26.

# iOS Driver

It automates any IOS native, hybrid, or mobile web application using the Selenium / WebDriver API. ios-driver is fully compatible with the Selenium / Webdriver API. IOS automation is therefore as easy as automation for a browser. ios-driver fully integrates with Selenium Grid so you can reuse your existing web automation

* Implements the JSON wire protocol
* Runs on emulators and devices
* Can be run as a regular node in a Selenium Grid

**Inspector**

The inspector allows you to examine the elements of a native app in the same style as you would with i.e. Firebug or the Chrome inspector. To access the inspector, set a breakpoint in the test code and execute it in Debug mode.

**Native tests**

IOS automation is very content based. There are no IDs like for web pages. This makes testing of localized pages a bit trickier. Fortunately the localization data is embedded into the app, so the server can access it if you know the key for the content.

# Android Driver

Appium can also be used for the purpose and it is an open source, cross-platform test automation tool. Supporting Android,

The Selendroid project

Selendroid is a test automation framework which drives off the UI of Android native and hybrid applications (apps) and the mobile web. Tests are written using the Selenium 2 client API.

# RemoteWebDriver

RemoteWebDriver is an implementation class of the WebDriver interface that a test script developer can use to execute their test scripts via the RemoteWebDriver server on a remote machine.

There are two parts to RemoteWebDriver: a server and a client. The RemoteWebDriver server is a component that listens on a port for various requests from a RemoteWebDriver client. Once it receives the requests, it forwards them to any of the following: Firefox Driver, IE Driver, or Chrome Driver, whichever is asked.

**RemoteWebDriver Modes**

The remote webdriver comes in two flavours:

* Client mode: where the language bindings connect to the remote instance. This is the way that the FirefoxDriver, OperaDriver and the RemoteWebDriver client normally work.
* Server mode: where the language bindings are responsible for setting up the server, which the driver running in the browser can connect to. The ChromeDriver works in this way.

# WebDriver and the Selenium-Server

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

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

**Send Key Combination:**

