# TEST AUTOMATION AND TOOL BASICS -SELENIUM

**Lesson 1: Introduction to Selenium**

## What is Automation Testing?

- The “Automation Testing” automates the job of testing a software
- In Automation Testing, a separate software is used to test the existing functional production software to be rolled out, based on the test cases identified
- Automation Testing reduces the overall efforts and time required in regression testing and speeds up testing life cycle

## Automation Testing – WHY and WHEN?

- Frequent regression testing
- Virtually unlimited execution of test cases is required
- Rapid feedback to developers
- Reduction in human efforts
- Test same application in multiple environment
- Finding defects missed in manual testing

# Manual Testing Vs Automation Testing

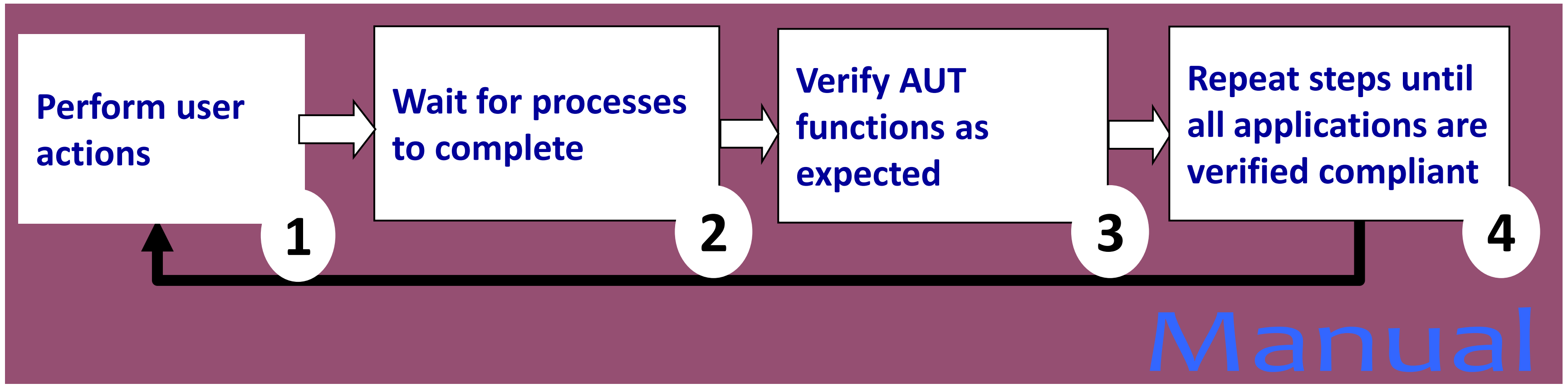
**Manual Testing**

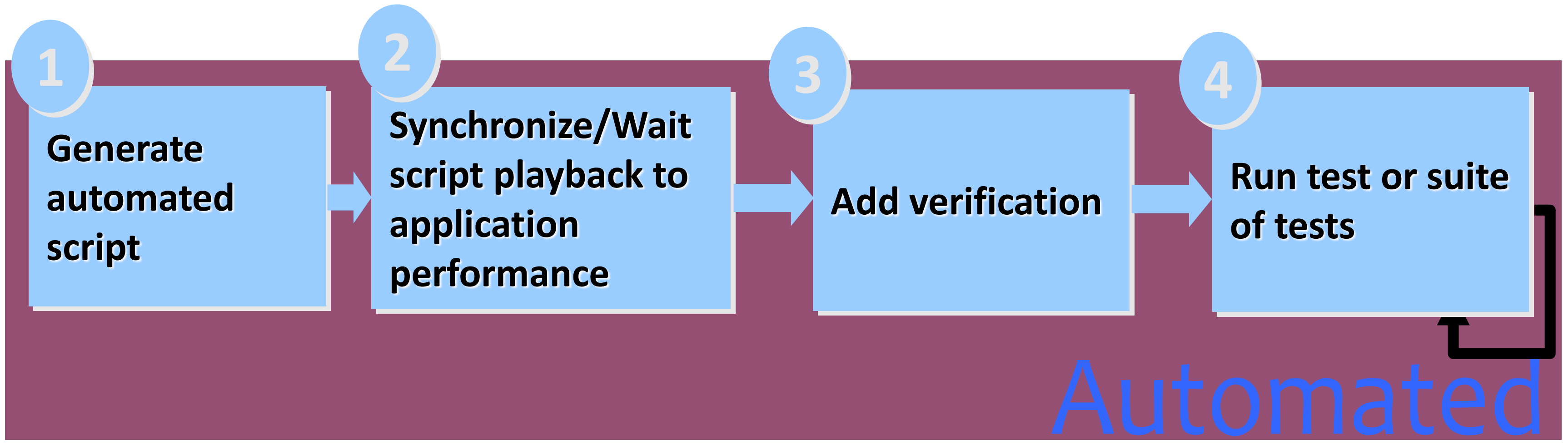
- Time consuming
- Low reliability
- Human resources
- Inconsistent

**Automation Testing**

- Speed
- Repeatability
- Programming capabilities
- Coverage
- Reliability
- Reusability

# Manual To Automated Testing


Perform user actions
1
Wait for processes to complete
2
Verify AUT functions as expected
3
Repeat steps until all applications are verified compliant
4
Manual


1
Generate automated script
2
Synchronize/Wait script playback to application performance
3
Add verification
4
Run test or suite of tests
Automated

## What Should Be Automated?

- Good candidates
  - Tests executed for each build
  - Business critical tests
  - Tests that are difficult/tedious to perform manually
- Bad candidates
  - Tests without predictable results
  - Test that require variable input/responses from the tester
  - Tests that perform operations in multiple environments

## Automation Testing - Disadvantages

- **High Initial Investment**
- **High Maintenance Cost**
- **Skill requirement**
- **Higher Timelines before use**
- **Long Payback Period**
- **Test Scripts Quality**
- **How to derive long term value**

## Introduction To Selenium

- Selenium is one of the most well known testing frameworks in the world that is in use
- It is an open source project that allows testers and developers alike to develop functional tests to drive the browser
- A functional testing tool for web applications
- It runs tests via a real browser that is driven by a JavaScript engine which is called "the BrowserBot"
- Works with any JavaScript-enabled browser ", since Selenium has been built using JavaScript
- It can be used to easily record and play tests

## Features of Selenium

- Allow Cross browser testing (Record in Firefox, Execute in IE)
- No dedicated machine required for test execution( user can work in parallel)
- Selenium uses JavaScript and IFrames to embed the BrowserBot in your browser
- The engine is tweaked to support wide range of browsers on Windows, Mac OS X and Linux

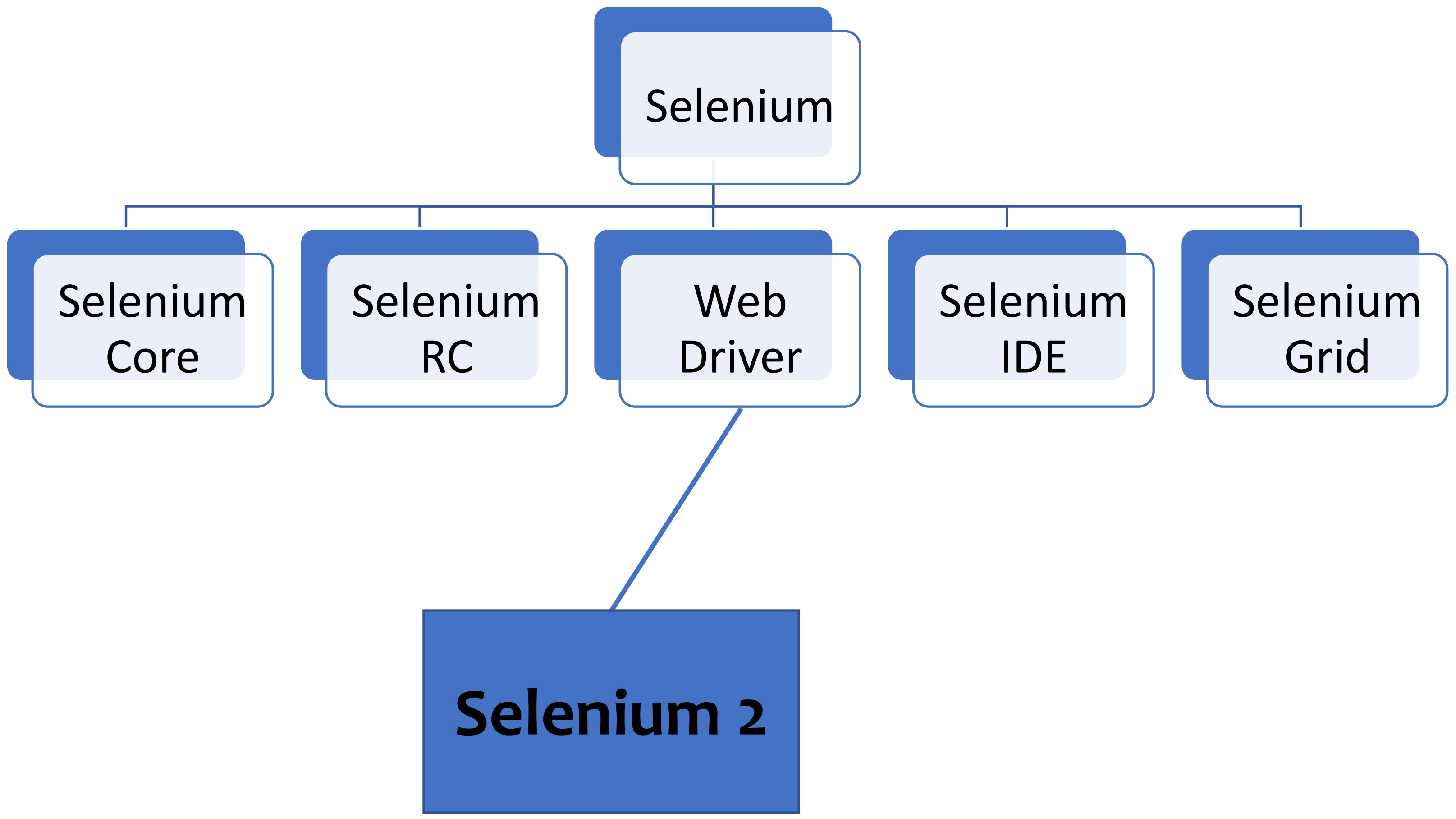
## Features of Selenium

- Languages Supported by Selenium – By Seleniumhq
  - Java
  - C#
  - Ruby
  - Python
  - JavaScript
- Third Party Language Bindings – NOT DEVELOPED by Seleniumhq
  - Perl
  - PHP
  - Haskell
  - Objective-C
- One should know at least one of these programming languages to dig deeper into Selenium

# Features of Selenium

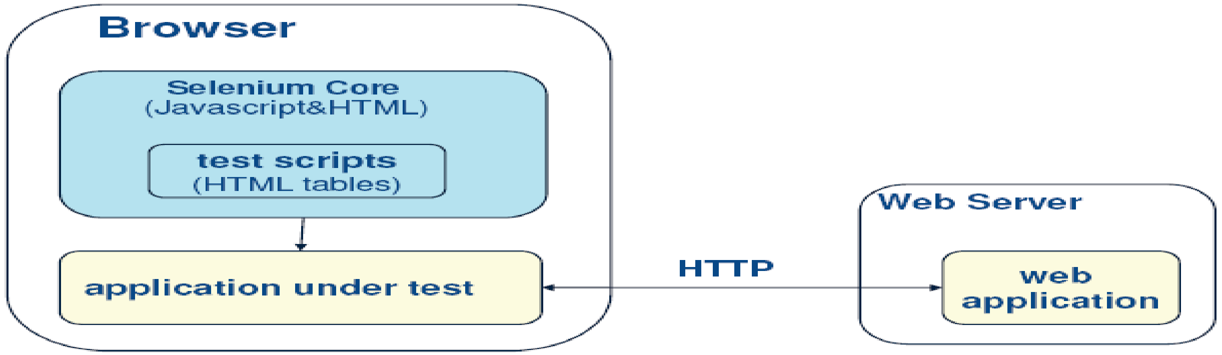
- Browsers Supported by Selenium
  - Mozilla Firefox
  - IE
  - Google Chrome
  - Opera
- Operating Systems supported by Selenium
  - Windows
  - Mac
  - Linux
  - Unix
  - Many more…..

# Flavors of Selenium



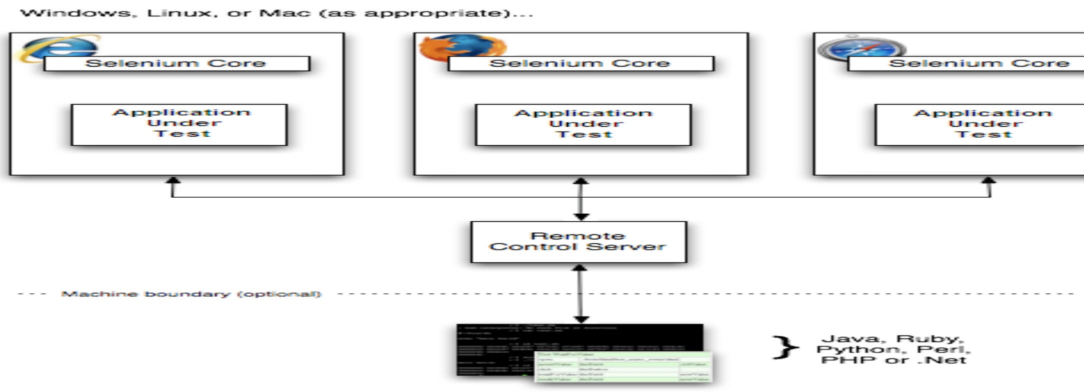
## Selenium Core

- Selenium Core is a JavaScript-based test tool for Web applications. Selenium Core tests run directly in a browser, just as real users do
  - Utility for running tests in web browser
  - Executes commands received from test script
  - Allows test scripts to run inside supported browsers
  - Works with Java script enabled browser
  - erating
  - rame



# Selenium RC (Remote Control)

- Selenium Remote Control (RC) is a test tool that allows you to write automated web application UI tests  against HTTP website using any mainstream JavaScript-enabled browser
- Selenium RC consists of two parts:
- Selenium Server: works as an http proxy for web request
- Client Libraries: Client library for selected language for automation



## Web Driver

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

## Selenium IDE

- Selenium IDE (Integrated Development Environment) to develop automation scripts using selenium
- Firefox extension
- Record and playback test in browser
- Intelligent field identification with IDs, names, XPaths etc.
- Record and walk through the test modes
- Import and export scripts in multiple formats e.g. HTML, Ruby, Java, C#, Perl and Python
- Allows script editing

## Selenium Grid

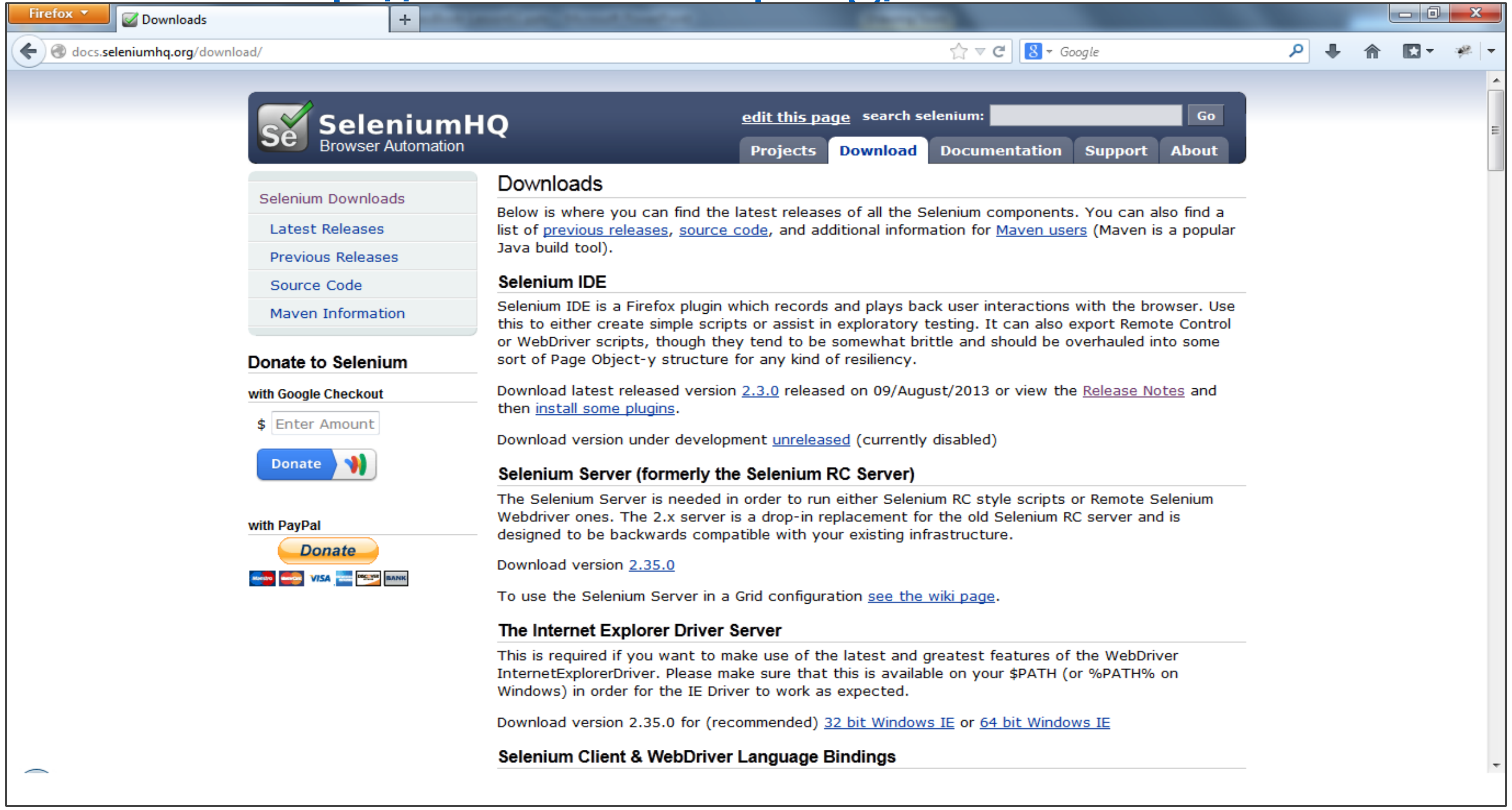
- Selenium Grid is basically a tool used along with Selenium RC to run test suits in multiple environments and to run them parallel
- Features of Selenium Grid
  - It enables concurrent running of test suits in multiple browsers and environments
  - It's a time effective technique of running tests
  - It works on the basis of hub and nodes concepts

## Selenium IDE – An Introduction

- Selenium IDE is an integrated development environment for Selenium tests
- It is implemented as a Firefox extension, and allows you to record, edit, and replay the web test in Firefox
- Using Selenium IDE is a great option available to testers to get started with writing test and group them together to build the Test Suit
- The recorded tests can be exported to many programming languages so that we can tweak them and put them in the testing framework
- The test cases and test suites can be replayed back to check the verifications and validations

# Installation of Selenium IDE – Step 1

- Open Mozilla Firefox Browser
- Visit – http://Seleniumhq.org/download



## Downloads

Below is where you can find the latest releases of all the Selenium components. You can also find a list of previous releases, source code, and additional information for Maven users (Maven is a popular Java build tool).

### Selenium IDE

Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.

Download latest released version 2.3.0 released on 09/August/2013 or view the Release Notes and then install some plugins.

Download version under development unreleased (currently disabled)

### Selenium Server (formerly the Selenium RC Server)

The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium Webdriver ones. The 2.x server is a drop-in replacement for the old Selenium RC server and is designed to be backwards compatible with your existing infrastructure.

Download version 2.35.0

To use the Selenium Server in a Grid configuration see the wiki page.

### The Internet Explorer Driver Server

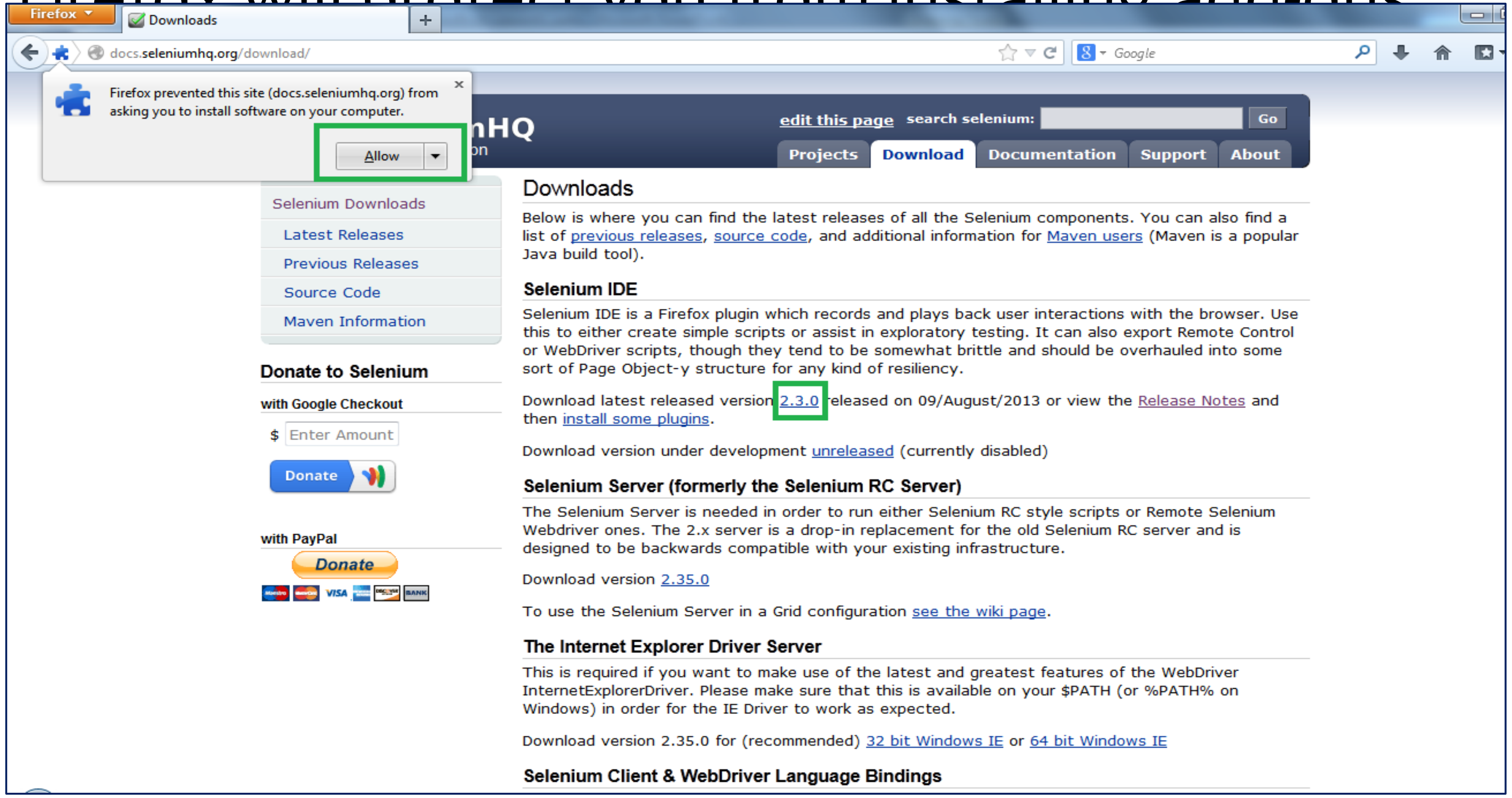
This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.35.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE

### Selenium Client & WebDriver Language Bindings

## Installation of Selenium IDE – Step 2

- Click on the Selenium IDE version as shown in the below screenshot
- Firefox will protect you from installing add-ons



Firefox prevented this site (docs.seleniumhq.org) from asking you to install software on your computer.

Donate to Selenium

with Google Checkout

$ Enter Amount

with PayPal

Downloads

Below is where you can find the latest releases of all the Selenium components. You can also find a list of previous releases, source code, and additional information for Maven users (Maven is a popular Java build tool).

Selenium IDE

Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.

Download latest released version 2.3.0 released on 09/August/2013 or view the Release Notes and then install some plugins.

Download version under development unreleased (currently disabled)

Selenium Server (formerly the Selenium RC Server)

The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium Webdriver ones. The 2.x server is a drop-in replacement for the old Selenium RC server and is designed to be backwards compatible with your existing infrastructure.

Download version 2.35.0

To use the Selenium Server in a Grid configuration see the wiki page.

The Internet Explorer Driver Server

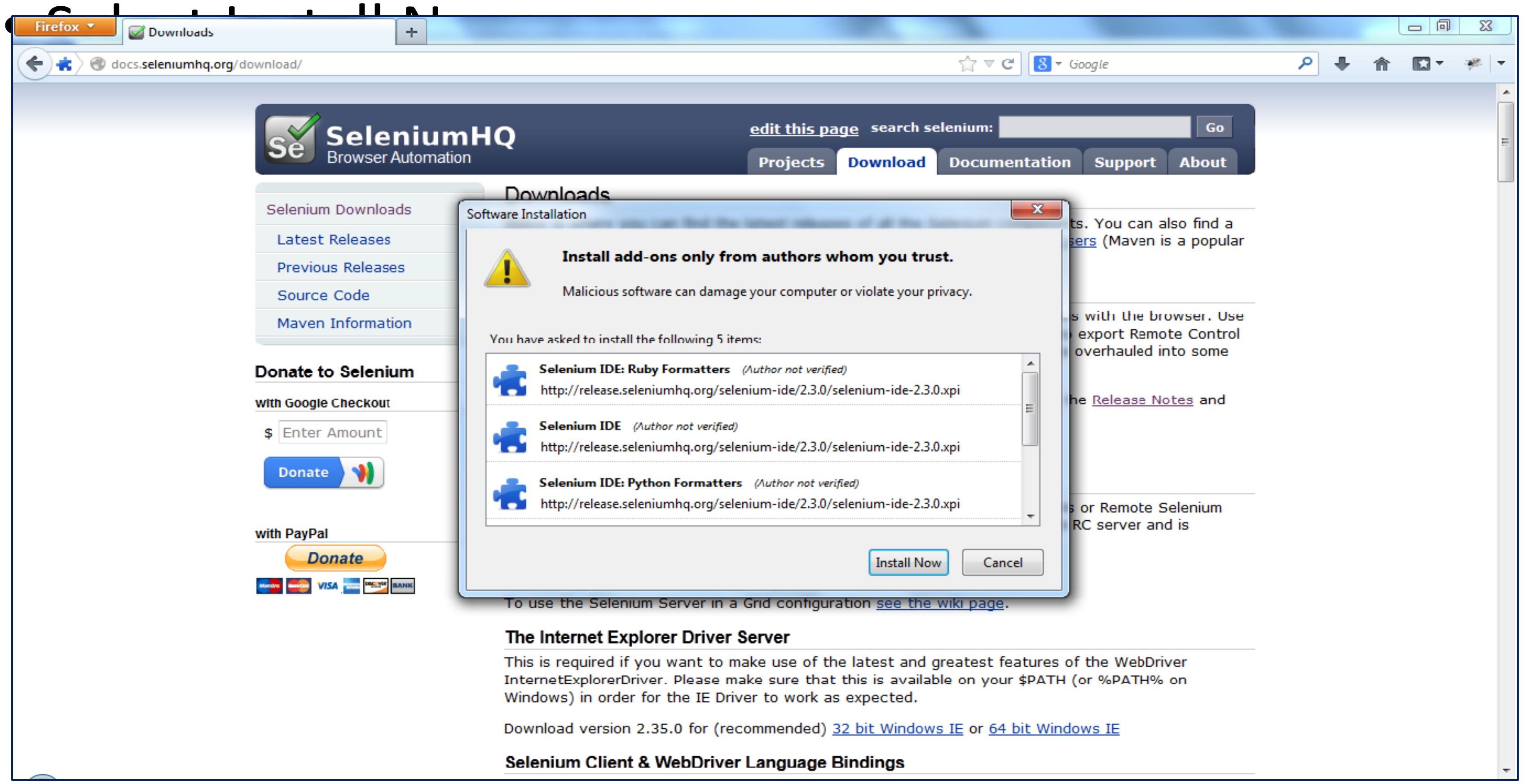
This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.35.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE

Selenium Client & WebDriver Language Bindings

# Installation of Selenium IDE – Step 3

- When downloading from Firefox, you'll be presented with the following window



Downloads

Software Installation

**Install add-ons only from authors whom you trust.**

Malicious software can damage your computer or violate your privacy.

You have asked to install the following 5 items:

**Selenium IDE: Ruby Formatters** *(Author not verified)*
http://release.seleniumhq.org/selenium-ide/2.3.0/selenium-ide-2.3.0.xpi

**Selenium IDE** *(Author not verified)*
http://release.seleniumhq.org/selenium-ide/2.3.0/selenium-ide-2.3.0.xpi

**Selenium IDE: Python Formatters** *(Author not verified)*
http://release.seleniumhq.org/selenium-ide/2.3.0/selenium-ide-2.3.0.xpi

Install Now    Cancel

ts. You can also find a sers (Maven is a popular

s with the browser. Use export Remote Control overhauled into some

he Release Notes and

s or Remote Selenium RC server and is

To use the Selenium Server in a Grid configuration see the wiki page.

**The Internet Explorer Driver Server**

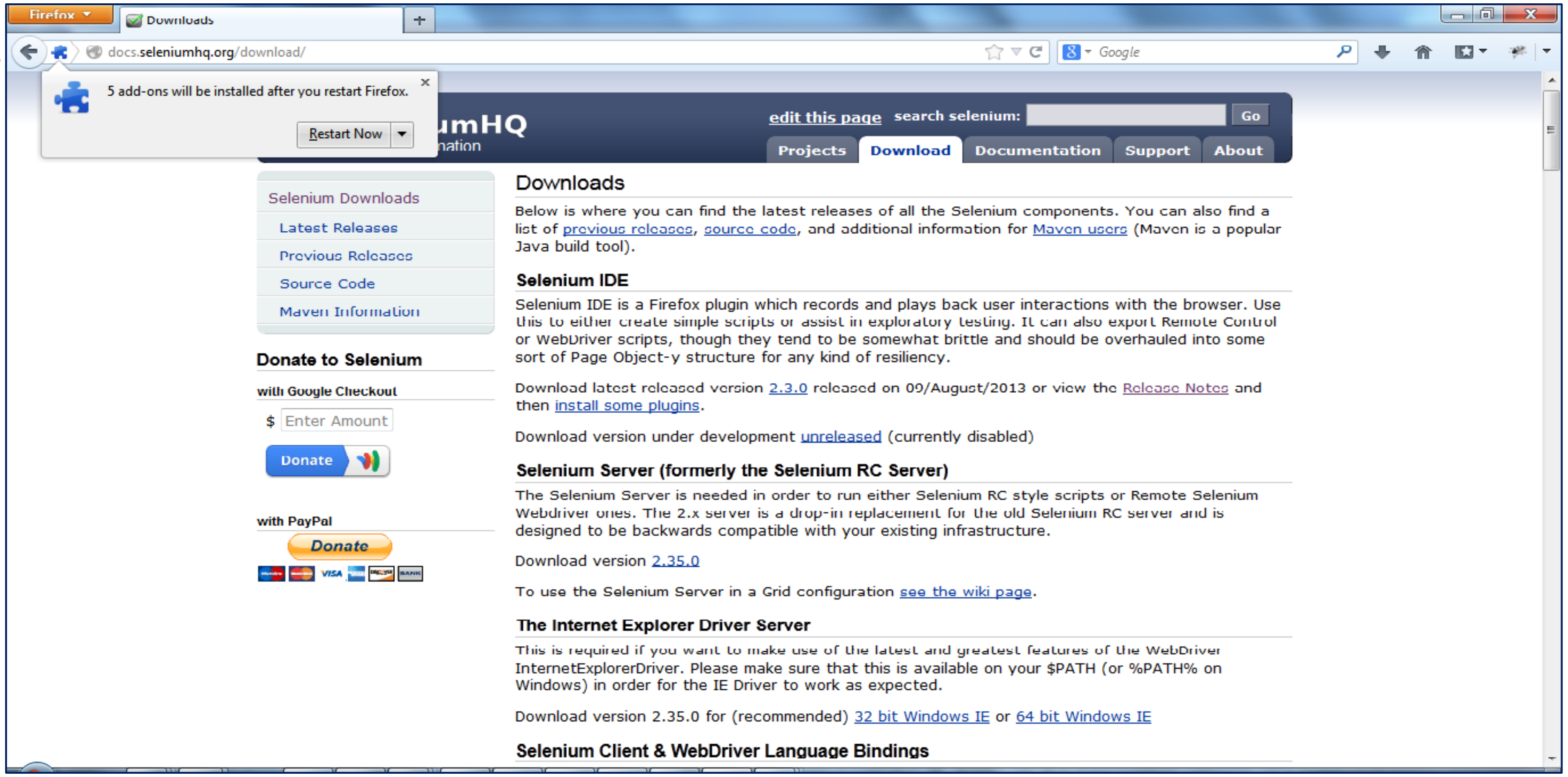
This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.35.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE

**Selenium Client & WebDriver Language Bindings**

## Installation of Selenium IDE – Step 4

- The Firefox Add-ons window pops up, first showing a progress bar, and when the download is complete, displays the following
-



5 add-ons will be installed after you restart Firefox.

Downloads

Below is where you can find the latest releases of all the Selenium components. You can also find a list of previous releases, source code, and additional information for Maven users (Maven is a popular Java build tool).

Selenium IDE

Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.

Download latest released version 2.3.0 released on 09/August/2013 or view the Release Notes and then install some plugins.

Download version under development unreleased (currently disabled)

Selenium Server (formerly the Selenium RC Server)

The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium Webdriver ones. The 2.x server is a drop-in replacement for the old Selenium RC server and is designed to be backwards compatible with your existing infrastructure.

Download version 2.35.0

To use the Selenium Server in a Grid configuration see the wiki page.

The Internet Explorer Driver Server

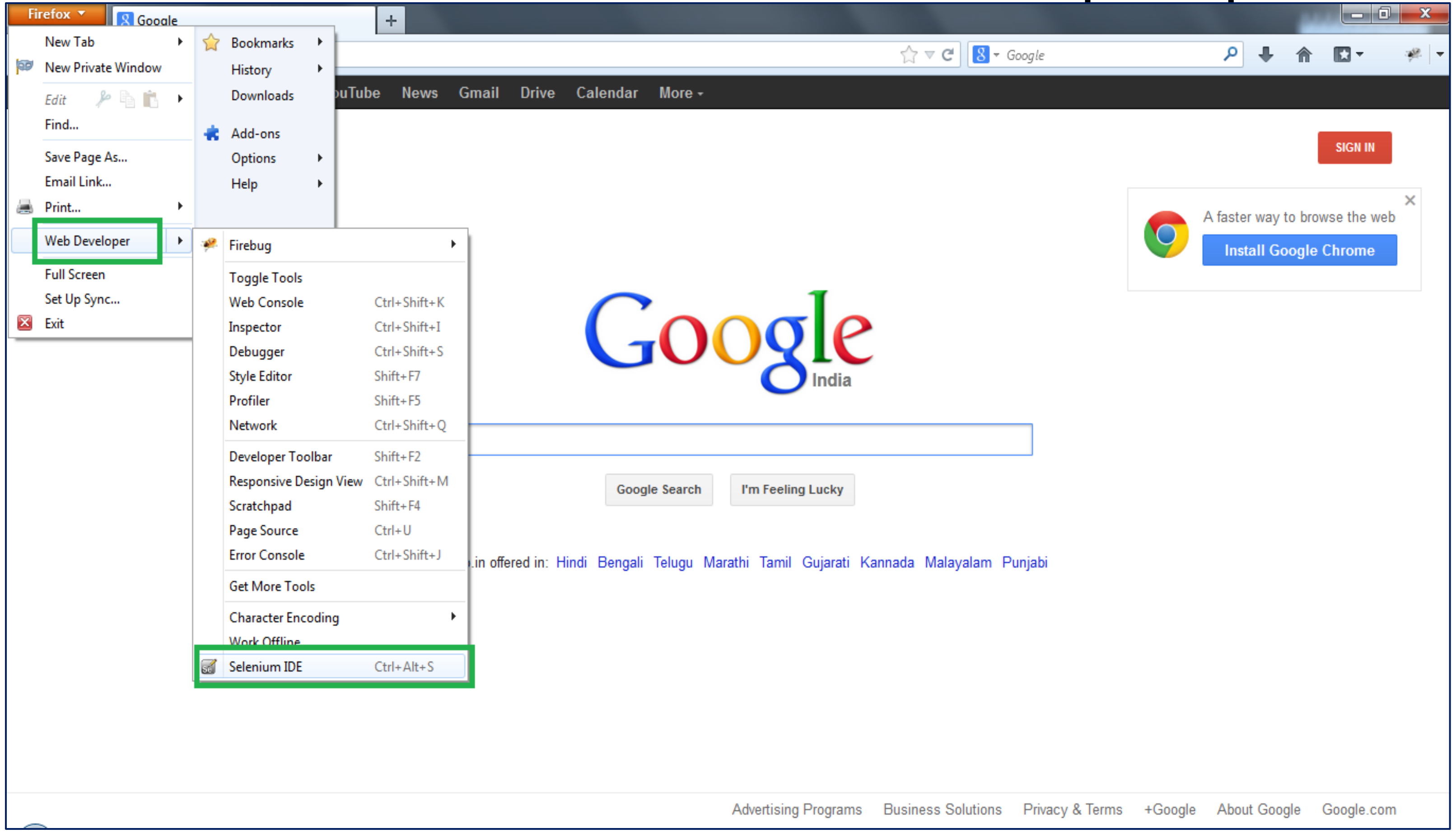
This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 2.35.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE

Selenium Client & WebDriver Language Bindings

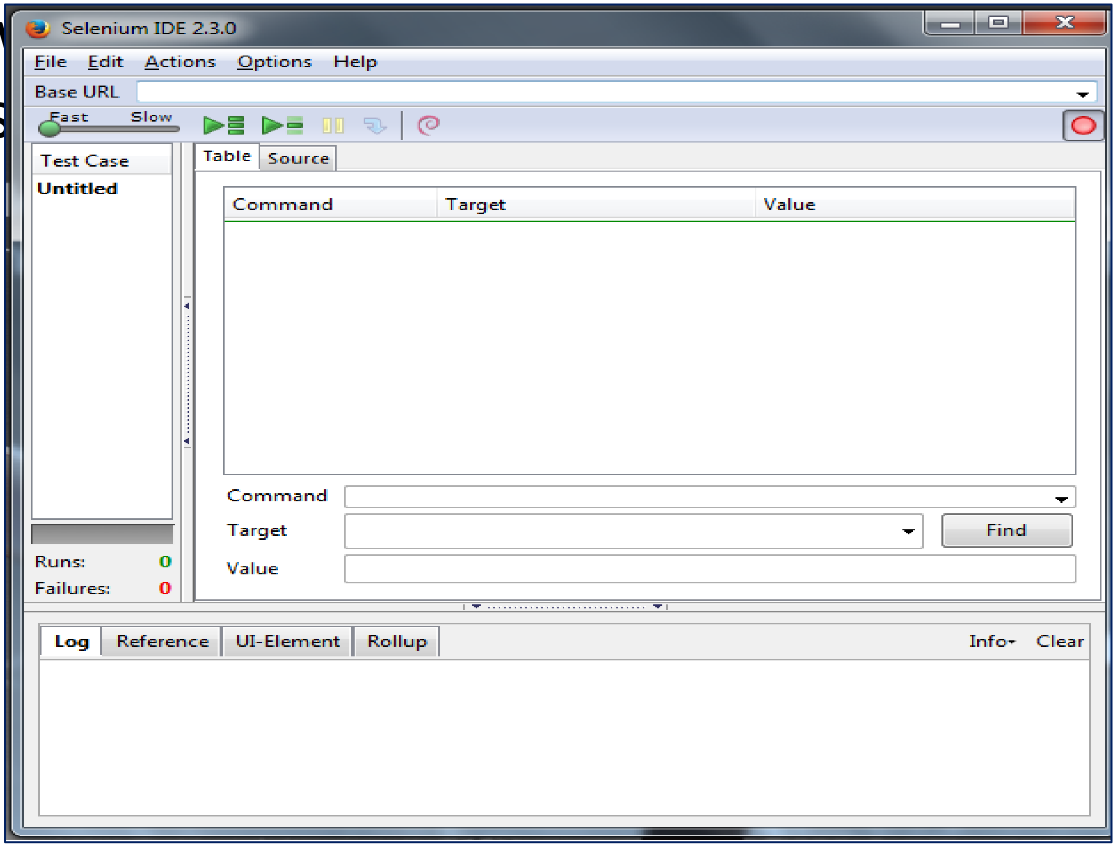
# Installation of Selenium IDE Completed

- After Firefox reboots you will find the Selenium-IDE listed under the Firefox Web Developer option

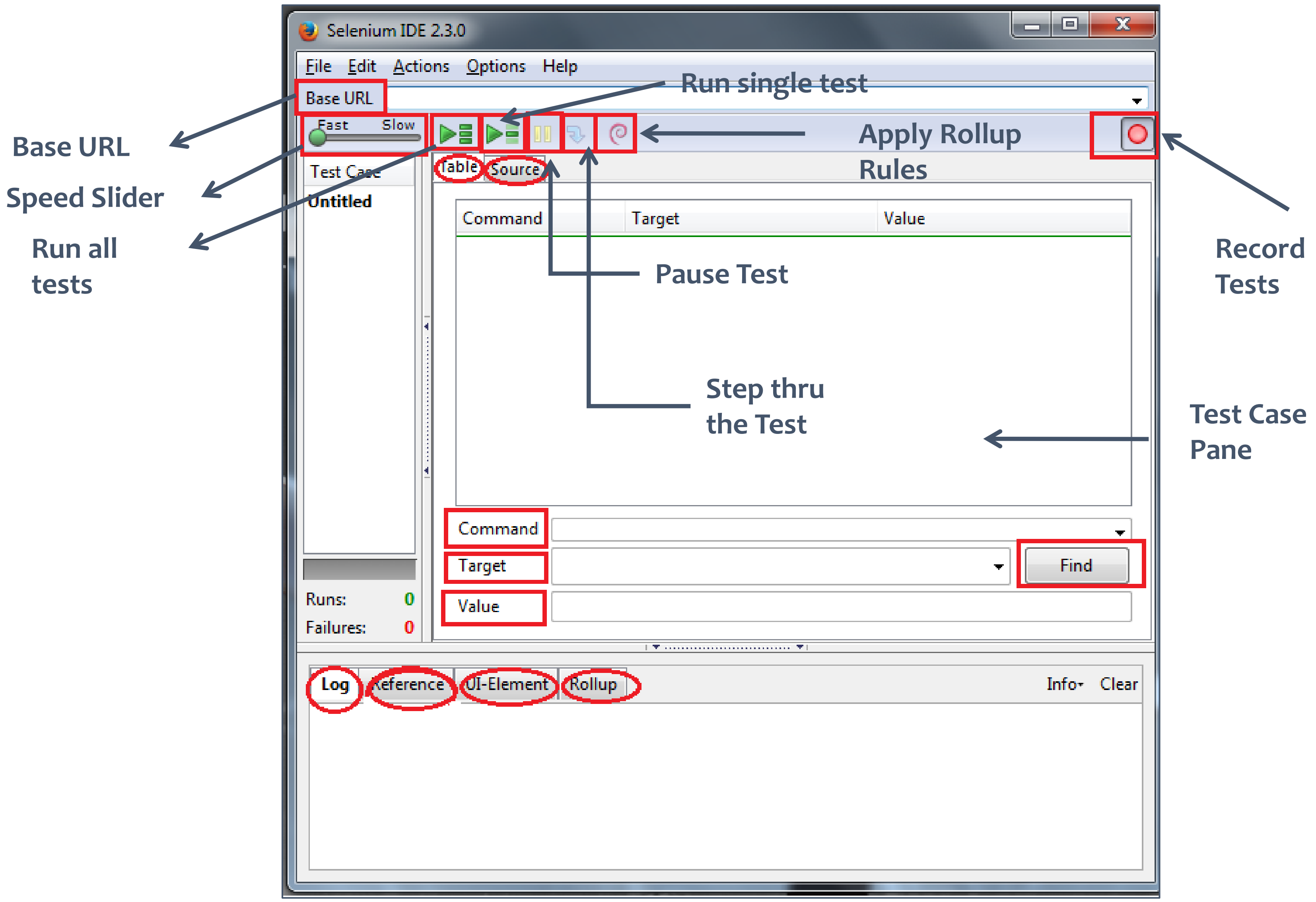


# Opening the Selenium IDE

- To run the Selenium-IDE, simply select it from the Firefox Web Developer option
- It opens as follows with an empty script-editing windo new test cas



# Components Of Selenium IDE



## Introduction to Selenium Commands – “Selenese”

- Selenium commands, often called as “Selenese”, are the set of commands that run your tests
- A sequence of these commands is a test script
- Selenium provides a rich set of commands for fully testing your web-app in virtually any way you can imagine
- These commands essentially create a testing language
- Selenese is essentially just a language which is nothing but the syntax and not dependent upon any language like C#, Java etc
- Selenese commands can have up to a maximum of two parameters: target and value

## Capabilities of “Selenese”

- With help of Selenese one can :
    - Test the existence of UI elements based on their HTML tags
    - Test for specific content
    - Test for broken links, input fields, selection list options, submitting forms, and table data among other things

- In addition Selenese supports testing of:
    - Window size
    - Mouse position
    - Alerts
    - Ajax functionality
    - Pop up windows
    - Event handling
    - And many other web-application features

## Types of Selenium Commands

| Type | Description |
|---|---|
| **Actions** | These are the commands that changes the state of the application by directly interacting with the page elements.<br><br>**Example**: Click the link, Select the option, Type text<br><br>If an Action fails, or has an error, the execution of the current test is stopped.<br>The **“AndWait”** suffix is used while calling the action. For example **“clickAndWait”**, this suffix instructs Selenium that it should wait for a new page to load. |
| **Accessors** | These are commands that allow you to examine the state of the application and store results in variables, e.g. “**storeTitle**”. |

# Types of Selenium Commands

| Type | Description |
|---|---|
| **Assertions** | They are like Accessors, but they verify that the state of the application conforms to what is expected.<br><br>**Examples:** "make sure the page title is something" and "verify that this radiobutton is selected".<br><br>**Three types of asertions**<br>**Assert:** When an "assert" fails, the test is aborted. For example **"assertText"**<br>**Verify:** When a "verify" fails, the test will continue execution, logging the failure. For example **"verifyText"**<br>**WaitFor:** Before proceeding to the next command, "waitFor" commands will first wait for a certain condition to become true.<br>**Step passes -** If the condition becomes true within the waiting period (30 Seconds).<br>**Step fails -** If the condition does not become true. Failure is logged, and test execution proceeds to the next command. |

# Selenium Commands

| **Command** | **Description** |
|---|---|
| **open** | It opens up the page using given URL |
| **click/clickAndWait** | It performs click operation and optionally waits for a new page to load |
| **verifyTitle/assertTitle** | It verifies an expected page title |
| **verifyTextPresent** | It verifies that the expected text is present on the page |
| **verifyElementPresent** | It verifies an expected UI element, as defined by its HTML tag, is present on the page |
| **verifyText** | It verifies that the expected text along with its HTML tag are present on the page |
| **verifyTable** | It can be used to verify the expected content on the table |
| **waitForPageToLoad** | It pauses execution until an expected new page loads. Called automatically when clickAndWait is used |
| **waitForElementPresent** | It pauses the execution until the expected UI is present on the web page |

## Understanding Element Locators in Selenium IDE

- The “Locators” informs Selenium IDE about which GUI elements it is supposed to operate on
- Identification of correct GUI elements is a prerequisite to  create an automation script
- Identifying the GUI element on a web page accurately is more difficult it sounds
- Sometimes we end up working on wrong GUI element or no elements at all
- Therefore, Selenium facilitates us with number of locators to precisely locate a GUI element on the web page

## Locators in Selenium

- The different types of Locators are given below :
  - ID
  - Name
  - Link Text
  - CSS Selector
    - Tag and ID
    - Tag and Class
    - Tag and Attribute
    - Tag, Class, and attribute
    - Inner Text
  - DOM (Document Object Model)
    - getElementById
    - getElementsByName

# Locators in Selenium

- ID - This is the most common technique of locating elements on the web page as ID’s are supposed to be unique for each element
- Name – Locating elements by their Name is very much similar to locating an element by its ID, except that we use “name=” instead
- Link Text – This type of locator is only used with the hyperlink element. We access the link by prefixing our target with “link=“ and then followed by the hyperlink text

## Finding elements by 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
- Selenium is compatible with CSS 1.0, CSS 2.0, and CSS 3.0 selectors
- CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes

## Finding elements by CSS - Examples

| **CSS Selector** | **Description** | **Syntax & Example** |
|---|---|---|
| **Tag and ID** | tag=HTML Tag<br>id=The ID of the element being accessed | Syntax - css=tag#id<br>Example –<br>css=input#Uname |
| **Tag and Class** | tag=HTML Tag<br>class=The class of the element being accessed | Syntax - css=tag.class<br>Example –<br>css=input.inputtext |
| **Tag and Attribute** | tag=HTML Tag<br>[attribute=value] | Syntax –<br>css=tag[attribute=value]<br>Example –<br>css=input[name=LName] |
| **Tag, Class and Attribute** | tag=HTML Tag<br>class=The class of the element being accessed<br>[attribute=value] | Syntax –<br>css=tag.class[attribute=value]<br>css=input.inputtext[name=LName] |
| **Inner Text** | tag=HTML Tag<br>Inner text=The inner text of the element | Syntax – css=tag:contains("inner text"<br>Example –<br>css=input.contains("Helllo") |

## Locating elements by DOM - Examples

| **DOM** | **Description** | **Syntax & Example** |
|---|---|---|
| **getElementById** | id of the element = this is the value of the ID attribute of the element to be accessed. This value should always be enclosed in a pair of parentheses (“”) | Syntax – document.getElementId(“id”)<br><br>Example – document.getElementId(“txtName” ) |
| **getElementsByName** | name = name of the element as defined by its ‘name’ attribute<br>index = an integer that indicates which element within getElementsByName’s array will be used | Syntax - document.getElementsByName(“name”)[index]<br>Example – document.getElementsByName(“rbGender”)[1] |

## Matching Text Patterns

- Using “Patterns” in selenium commands is one of the efficient  way of writing good tests
- They enable you to match various content types on a web page like Links, elements , text
- Examples of commands which require patterns are verifyTextPresent, verifyTitle, verifyAlert, assertConfirmation, verifyText, and verifyPrompt
- There are three types of patters those can be used along with Selenium Commands:
  - Globbing
  - Regular Expression
  - Exact

## Matching Text Patterns – Globbing Patterns

- “Globbing Patterns” is the one of the matching text patterns in selenium
- You can describe expected text pattern with command's target column and can use it with verify and assert commands
- We can use globbing pattern when expected text string is dynamic and can use with commands like verifyTitle, assertText, verifyTextPresent, assertTextPresent etc

## Matching Text Patterns – Globbing Patterns

- Globbing is fairly limited
- Only two special characters are supported in the Selenium implementation

| **Pattern** | **Description** | **Example** |
|---|---|---|
| * | Used to “match anything,” i.e., nothing, a single character, or many characters | Example – **glob:Film*Television Department** |
| [ ] **(character class)** | Used to “match any single character found inside the square brackets.” A dash (hyphen) can be used as a shorthand to specify a range of characters. | Example – <br><br>**[aeiou]** - matches any lowercase vowel<br>**[0-9]** - matches any digit<br>**[a-zA-Z0-9]** - matches any alphanumeric character |

## Matching Text Patterns – Regular Expression

- Regular Expression pattern is the most powerful of the three types of patterns that selenium command supports
- Regular expressions are also supported by most high-level programming languages
- In Selenese, regular expression patterns allow a user to perform many tasks that would be very difficult otherwise
- For example, if you need to create a test that needs to ensure that a textbox should contain nothing but a numeric value
- Selenese regular expression patterns offer the same wide array of special characters that exist in JavaScript

## Matching Text Patterns – Regular Expression

| **Pattern** | **Match** |
|---|---|
| [ ] (character class) | character class: any single character that appears inside the brackets |
| * | quantifier: 0 or more of the preceding character (or group) |
| + | quantifier: 1 or more of the preceding character (or group) |
| . | Any single character |
| ? | quantifier: 0 or 1 of the preceding character (or group) |
| {1,5} | quantifier: 1 through 5 of the preceding character (or group) |
| \| | alternation: the character/group on the left or the character/group on the right |
| ( ) | grouping: often used with alternation and/or quantifier |

## Matching Text Patterns – Exact Pattern

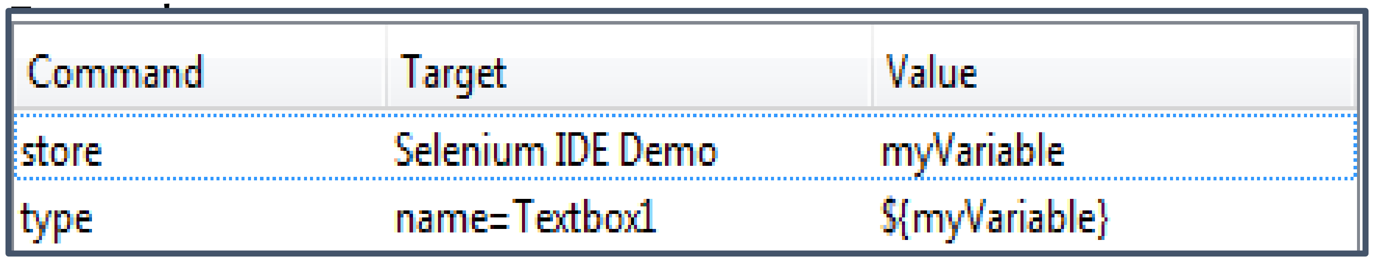
- The exact type of Selenium pattern is of marginal usefulness
- It uses no special characters at all
- If you needed to look for an actual asterisk character which is special for both globbing and regular expression patterns, the exact pattern would be one way to do that
- For example, if you wanted to verify the text present on the web page like “* Conditions apply” then the code “glob:* Conditions apply” might not work
- In order to ensure that the “* Conditions apply” text is verified on the web page, the “exact” prefix can be used
- Valid pattern – <u>exact: * Conditions apply</u>

## Storing information from the page in the test

- Sometimes there is a need to store elements that are on the page to be used later in a test
- This could be that your test needs to pick a date that is on the page and use it later so that you do not need to hardcode values into your test
- You can also use Selenium variables to store constants at the beginning of a script
- Selenium variables can be used to store values passed to your test program from the command-line, from another program, or from a file
- Once the element has been stored you will be able to use it again by requesting it from a JavaScript dictionary that Selenium keeps track of
- To use the variable it will take one of the following two formats: it can look like ${variableName}

## Store Commands

- store
  - The plain store command is the most basic of the many store commands and can be used to simply store a constant value in a selenium variable
  - It takes two parameters, the text value to be stored and a selenium variable
  -

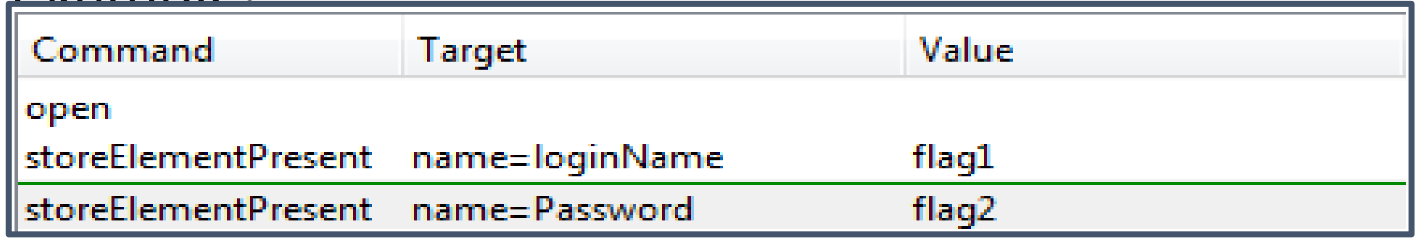


| Command | Target | Value |
|---|---|---|
| store | Selenium IDE Demo | myVariable |
| type | name=Textbox1 | ${myVariable} |

  - The above test stores the value “Selenium IDE Demo” in the variable “myVariable”
  - You can read the value of the variable in the texbox on your web page named “Textbox1” by setting the value for the type command as ${myVariable}
  - Upon execution of above script will store the value “Selenium IDE Demo” in the textbox “Textbox1”

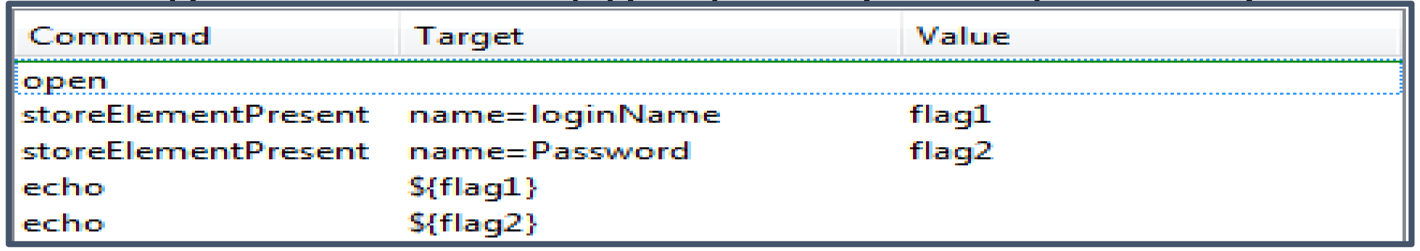
# Store Commands

- storeElementPresent
  - This command stores either "true" or "false" depending on the presence of the specified element
  - Example:



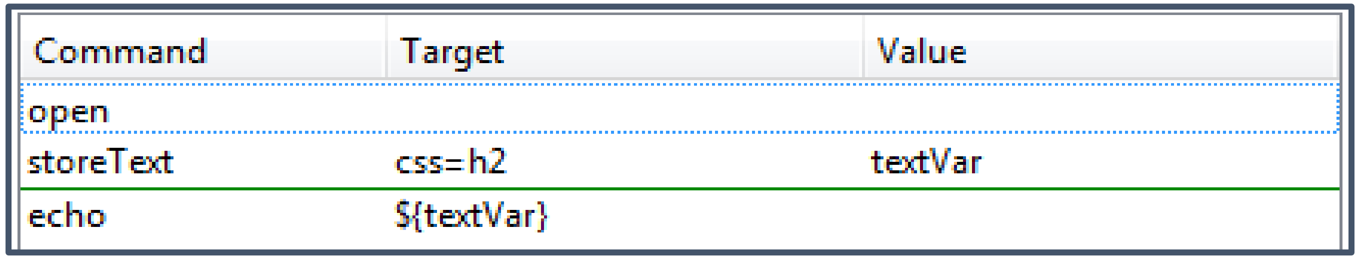
| Command | Target | Value |
|---|---|---|
| open | | |
| storeElementPresent | name=loginName | flag1 |
| storeElementPresent | name=Password | flag2 |

  - In the above test script, the variables falg1 & flag2 will store the values either true or false depending
  -



| Command | Target | Value |
|---|---|---|
| open | | |
| storeElementPresent | name=loginName | flag1 |
| storeElementPresent | name=Password | flag2 |
| echo | ${flag1} | |
| echo | ${flag2} | |

## Store Commands

- storeText
  - This command is used to store the inner text of an element onto a variable
  - 

| Command | Target | Value |
|---|---|---|
| open | | |
| storeText | css=h2 | textVar |
| echo | ${textVar} | |

  - The above script will save the inner text in the variable “textVar” of the element having satisfied the condition i.e. “css=h2”

# Working with Alerts

- Alerts are probably the simplest form of pop-up windows

| **Command** | **Description** |
|---|---|
| **assertAlert**<br>**assertNotAlert** | Retrieves the message of the alert and asserts it to a string value that you specified. |
| **assertAlertPresent**<br>**assertAlertNotPresent** | Asserts if an Alert is present or not |
| **storeAlert** | Retrieves the alert message and stores it in a variable that you will specify. |
| **storeAlertPresent** | Returns TRUE if an alert is present; FALSE if otherwise. |
| **verifyAlert**<br>**verifyNotAlert** | Retrieves the message of the alert and verifies if it is equal to the string value that you specified. |
| **verifyAlertPresent**<br>**verifyAlertNotPresent** | verifies if an Alert is present or not |

## Working with Confirmation

- Confirmations are pop-ups that give you an OK and a CANCEL button, as opposed to alerts which give you only the OK button
- The commands you can use in handling confirmations are similar to those in handling alerts
  - assertConfirmation/assertNotConfirmation
  - assertConfirmationPresent/assertConfirmationNotPresent
  - storeConfirmation
  - storeConfirmationPresent
  - verifyConfirmation/verifyNotConfirmation
  - verifyConfirmationPresent/verifyConfirmationNotPresent
  - chooseOkOnNextConfirmation/chooseOkOnNextConfirmationAndWait
  - chooseCancelOnNextConfirmation

## Introduction to Debugging in Selenium IDE

- Debugging means finding and fixing errors in your test case
- This is a normal part of test case development
- Sometimes, as a test automator, you will need to debug your tests to see what is wrong
- There are various commonly used techniques are available in Selenium IDE which can be used to identify an error in the test case
- The tester can optionally break or start the execution of a test case to debug and figure out the existing error in the test case

## Using Breakpoints in Test Case

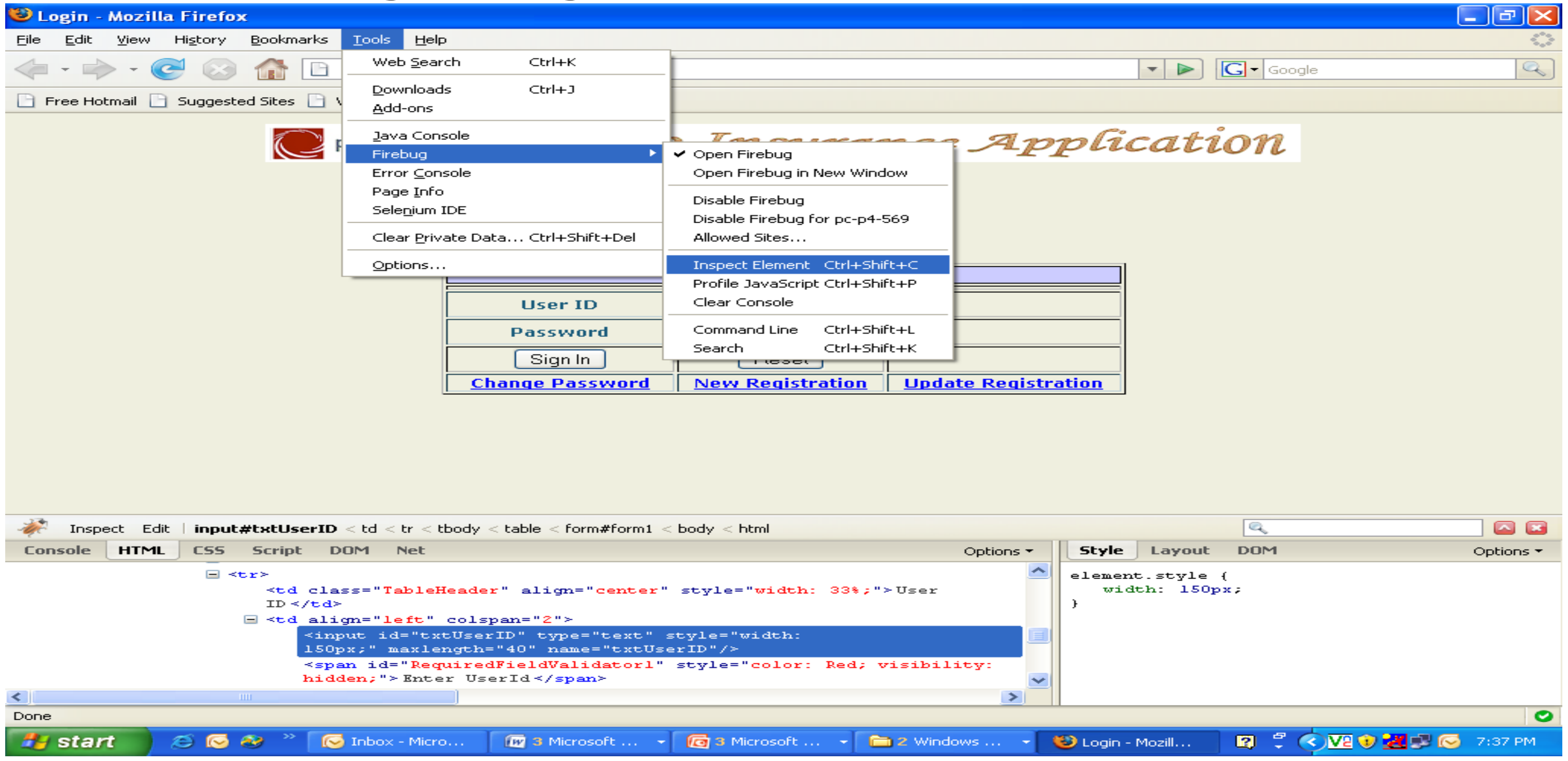
- One can run up to a specific command in the middle of the test case and inspect how the test case behaves at that point
- To do this, set a breakpoint on the command just before the one to be examined
- Steps to be followed
  - Select a command
  - Right-click, and from the context menu select Toggle Breakpoint
  - Then click the Run button to run your test case from the beginning up to the breakpoint
  - Click on Step button to execute the test case which has halted as it has reached the breakpoint
  - Observer the test execution

# Using Startpoint in Test Case

- If you have a really long test and it’s failing towards the end, then you can set a custom start point so that you don’t have to run the entire test when you’re investigating the failure
- For example, your test might register a new user, log in, and then fail on the Home Page
- You could simply navigate to the home page yourself and set your test to start from there
- To set a start point simply right click on the first command you want Selenium IDE to execute and click ‘Set / Clear Start Point’
- You will see a small play icon appear to the left of your command

# Object identification using firebug

- Firebug is add on to Firefox
- It helps in getting object properties, DOM structure,



```
<tr>
    <td class="TableHeader" align="center" style="width: 33%;">User
    ID</td>
  <td align="left" colspan="2">
      <input id="txtUserID" type="text" style="width:
      150px;" maxlength="40" name="txtUserID"/>
      <span id="RequiredFieldValidator1" style="color: Red; visibility:
      hidden;">Enter UserId</span>
```

```
element.style {
    width: 150px;
}
```

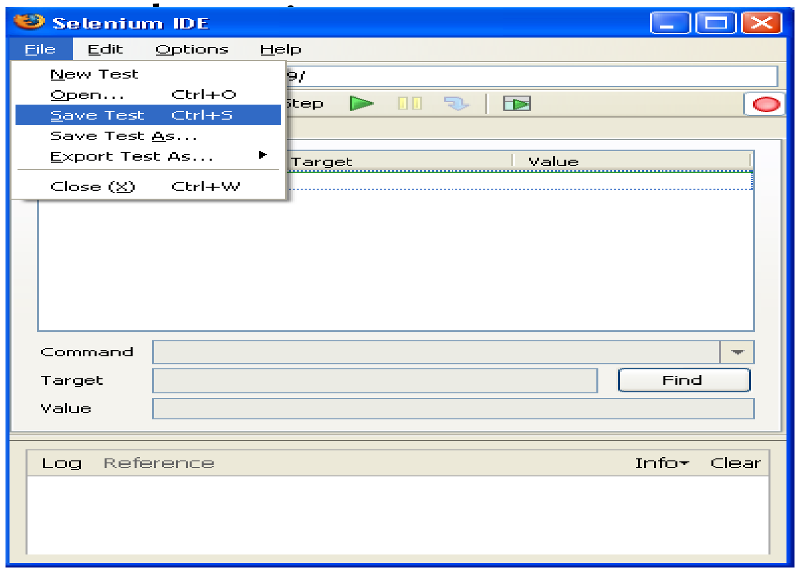
## Create scripts using IDE

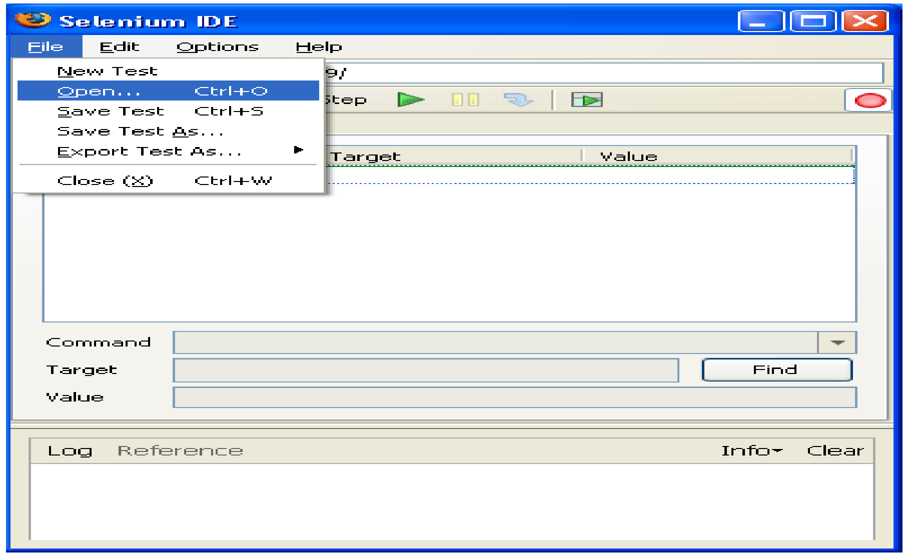
- Perform following steps to create script:
  - Perform following steps to create script:
  - Launch Mozilla Firefox
  - Open application in Firefox
  - Invoke Selenium Tools ->Selenium IDE
  - Invoke firebug Tools -> firebug -> Open Firebug
  - Enter command in Selenium IDE
  - Inspect element using firebug and specify element locator
  - Specify value if required
  - Repeat above steps as required.

# Save and load scripts in IDE

- To save test go to
  - File->Save Test
  - Give the test name and save it in desired

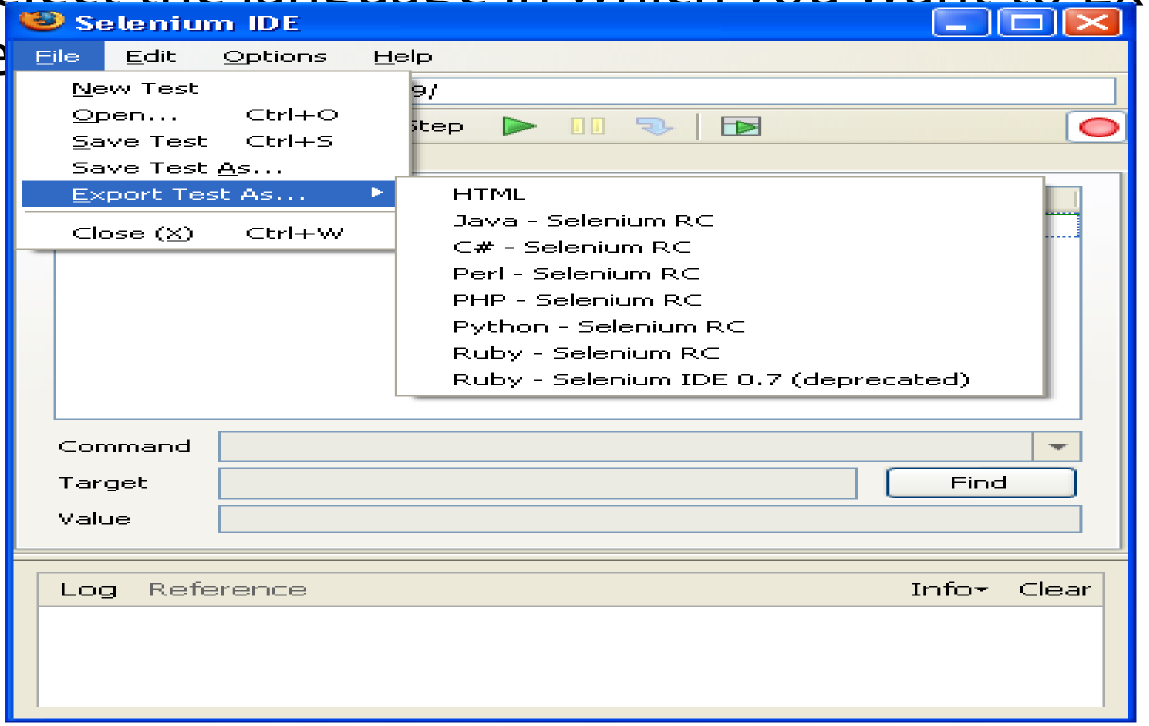
- To load a test go to
  - File->Open
  - And open a particular test





# Export scripts to multiple language

- To export a test in particular language perform the following steps
  - File->Export Test As
  - Select the language in which you want to Export the te



## An Introduction to Web Driver

- “Web Driver” is a Web Automation Framework which is also knows as “Selenium 2”
- It allows you to create and execute tests against different browsers, unlike Selenium IDE which works only with Firefox
- WebDriver is designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API
- Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded
- WebDriver’s goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems

# Web Driver Vs Selenium RC

- WebDriver is implemented through a browser-specific browser driver, which sends commands to a browser, and retrieves results
- Most browser drivers actually launch and access a browser application, there is also a HtmlUnit browser driver, which simulates a browser using HtmlUnit
- Selenium RC is written in JavaScript which causes a significant weakness
- Browsers impose a pretty strict security model on any JavaScript that they execute in order to protect a user from malicious scripts
- Rather than being a JavaScript application running within the browser, it uses whichever mechanism is most appropriate to control the browser
- For Firefox, this means that WebDriver is implemented as an extension. For IE, WebDriver makes use of IE's Automation controls

## Benefits of Web Driver over Selenium RC

- Web Deriver is much faster than Selenium RC as it uses browsers own engine to control the behavior
- Selenium RC is slower as it uses Selenium Core, the JavaScript program to control the browser
- Though Selenium RC's API is more matured but contains redundancies and often confusing commands
- For example, most of the time, testers are confused whether to use type or typeKeys, or whether to use click, mouseDown, or mouseDownAt
- Worse, different browsers interpret each of these commands in different ways too
- WebDriver's API is simpler than Selenium RC's & it does not contain redundant and confusing commands
- Web Driver can use HtmlUnit, the headless or invisible browser, Selenium RC needs real or visible browsers to operate on

## Limitation of Web Driver

- Web Driver cannot support new web browsers out of the box
  - Web Driver controls browser from OS level
  - Different web browsers communicate with the OS in a different way
  - New browsers may have different way of communicating with OS in a different way
- No built-in test result generator support
  - Selenium RC automatically generates the test result in an HTML format
  - Web Driver has no built-in provision that can help tester in generating Test Results File
  - The Tester would have to rely on your IDE's output window, or design the report yourself using the capabilities of your programming language and store it as text, html, etc

## Writing first Web Driver Test Script

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

public class myFirstTestScript
{
    public static void main(String[] args)
    {
     // declaration and instantiation of objects/variables
     WebDriver driver = new FirefoxDriver();
     String baseUrl = "http://http://docs.seleniumhq.org/";
     String expectedTitle = "Selenium - Web Browser Automation";
     String actualTitle = "";

     // launch Firefox and direct it to the Base URL
     driver.get(baseUrl);
     // get the actual value of the title
     actualTitle = driver.getTitle();
```

# Writing first Web Driver Test Script

```
if (actualTitle.contentEquals(expectedTitle))
    {
        System.out.println("Test Passed!");
     }
     else
    {
        System.out.println("Test Failed");
     }

     //Close browser window
     driver.close();

  }
}
```

## Locating UI Elements

- Locating elements in WebDriver can be done on the WebDriver instance itself or on a WebElement
- Each of the language bindings expose a “Find Element” and “Find Elements” method
- The first returns a WebElement object otherwise it throws an exception
- The latter returns a list of WebElements, it can return an empty list if no DOM elements match the query.
- The “Find” methods take a locator or query object called “By”

# Locating UI Elements

| Locator | Description | Usage |
|---|---|---|
| **ByID** | Locates element using value of their “ID” attribute | HTML - <div id=“div1”>...</div><br>Java - WebElement element = driver.findElement(By.id(“div1"));  |
| **By.ClassName** | Locates element using value of their “Class” attribute | HTML<br><div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></div><br><br>Java - List<WebElement> cheeses = driver.findElements(By.className("cheese")); |
| **By.Name** | Locates element using value of their “Name” attribute | HTML<br><input name=“txtUName" type="text"/><br>Java<br>WebElement cheese = driver.findElement(By.name(“txtUName")); |

# Locating UI Elements

| Locator | Description | Usage |
|---|---|---|
| **ByLinkText** | Finds a link element by the exact text it displays | HTML – <br> <a href="http://www.google.com/search?q=cheese">cheese</a> <br> Java -WebElement cheese = driver.findElement(By.linkText("cheese")); |
| **By.PartialLinkText** | Find the link element with partial matching visible text. | HTML <br> <a href="http://www.google.com/search?q=cheese">search for cheese</a> <br> Java - WebElement cheese = driver.findElement(By.partialLinkText("cheese")); |
| **By.CSS** | Finds elements based on the driver's underlying CSS Selector engine | findElement(By.cssSelector("input#email")) |

# Locating UI Elements

| Locator | Description | Usage |
|---|---|---|
| **By.tagName** | locates elements by their tag name | HTML - <div id="div1">...</div> Java - findElement(By.tagName("div")) |
| **By.xpath** | locates elements via Xpath | findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]/table/tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/form/table/tbody/tr[5]")) |

# Using sendKeys() and click()

- Example of sendKeys()

```
WebElement myElement = driver.findElement(By.id(“Username”));
myElement.sendKeys(“SeleniumUsers”);
```

- Clicking on an Element

```
driver.findElement(By.name(“Click Me”)).click();
```

- It does not take any parameter/argument
- The method automatically waits for a new page to load if applicable
- The element to be clicked-on, must be visible

## Using Get Commands API

| Command | Description |
|---|---|
| **Get()** | 1. It automatically opens a new browser window and fetches the page that you specify inside its parentheses<br>2. The parameter must be a string |
| **getTitle()** | 1. Fetches the title of the current page<br>2. Return null if the current page has no title<br>3. Needs no parameters |
| **getPageSource()** | 1. Return the source code of the page as a string value<br>2. Needs no parameters |
| **getCurrentUrl()** | 1. Gets the url of the current page loaded in the browser<br>2. Needs no parameters |
| **getText()** | 1. Fetches the inner text of the element that you specify |

## Using Navigate Commands API

| Command | Description |
|---|---|
| **navigate().to()** | 1. Behaves exactly same as get() method<br>2. It opens a new browser window and loads the page that you specify inside its parentheses |
| **navigate().refresh()** | 1. Refreshes current loaded page in the browser<br>2. Needs no parameters |
| **navigate().back()** | 1. Takes you back by one page on the browsers history<br>2. Needs no parameters |
| **navigate().forward()** | 1. Takes you forward by one page on the browsers history<br>2. Needs no parameters |

## Closing & Quitting Browser Window

| Command | Description |
|---|---|
| **close()** | 1. It closes the browser window which is being opened currently<br>2. Needs no parameters |
| **quit()** | 1. It closes all windows that web drive has opened<br>2. Needs no parameters |

# Moving between Windows and Frames

**HTML Code**

```
<a href="somewhere.html" target="windowName">Click here to open a
new window</a>
```

**Java Code**

```
driver.switchTo().window("windowName");
```

•

**Java Code**

```
for (String handle : driver.getWindowHandles()) {
driver.switchTo().window(handle); }
```

**Java Code**

```
driver.switchTo().frame("frameName");
```

# Handling Popup Dialogs

- Starting with Selenium 2.0 beta 1, there is built in support for handling popup dialog boxes
- After you've triggered an action that opens a popup, you can access the alert with the following:

**Java Code**

```
Alert alert = driver.switchTo().alert();
```

- This will return the currently open alert object
- With this object you can now accept, dismiss, read its contents or even type into a prompt
- This interface works equally well on alerts, confirms, and prompts

# Using Explicit & Implicit Wait

- Waiting is having the automated task execution elapse a certain amount of time before continuing with the next step
- Explicit Waits
  - An explicit waits is code you define to wait for a certain condition to occur before proceeding further in the code
  - There are some convenience methods provided that help you write code that will wait only as long as required
  - WebDriverWait in combination with ExpectedCondition is one way this can be accomplished
  - Import following two packages
    - import org.openqa.selenium.support.ui.ExpectedConditions;
    - import org.openqa.selenium.support.ui.WebDriverWait;

# Using Explicit along with Expected Condition

- The ExpectedConditions class offers a wider set of conditions that you can use in conjunction with WebDriverWait's until() method

```
WebDriver driver = new FirefoxDriver();
WebDriverWait myWait = new WebDriverWait(driver,10);
```

```
myWait.until(ExpectedConditions.visibilityOfElementLocated(By.id("userna
me")));
drive.findElement(By.id("username")).sendKeys("SeleniumUser");
```

- The above code will put an explicit wait on the "username" element before we type the text "SeleniumUser" into it

## Using Explicit along with Expected Condition

- alertIsPresent – Waits until an alert box is visible

```
If(myWait.until(ExpectedConditions.alertIsPresent()) != null)
{
        System.out.println(“Alert box is available”);
}
```

```
WebElement txtQualification =
myWait.until(ExpectedConditions.elementToBeClickable(By.id(“Qualification
”)));
```

- visible and, at the same time, enabled

## Using Explicit & Implicit Wait

- Implicit Waits
  - It is easy to code the Implicit wait compare to coding the explicit wait
  - The right place for declaring implicit wait for the test is in the instantiation part of the code
  - Import following package to declare implicit wait in the test
    - import java.util.concurrent.TimeUnit;

➢

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

## Working with Forms using Web Driver

| Element | Command | Example |
|---|---|---|
| **InputBox** | sendKeys()<br>clear() | driver.findElement(By.name(“username”)).sendKeys(“SeleniumUse r”); |
| **RadioButton, CheckBox** | click() | driver.findElement(By.cssSelector(“input[value=‘Male’]”)).<br>click();<br>WebElement chkHobbies =<br>driver.findElement(By.id(“chkMusic”));<br>chkHobbies.click(); |
| **Links** | click() | Driver.findElement(By.linkText(“Click Me”)).click(); |
| **Drop-Down Box** | Select | select drpCountry = new<br>Select(driver.findElement(By.name(“Country”))); |
| **Submit Form** | submit() | The submit() method is used to submit a form. This is an alternative to clicking the form’s submit button. You can use submit() on any element within the form, not just on the submit button itself.<br>driver.findElement(By.name(“password”)).submit(); |

# Working with Forms using Web Driver – DropDown Box

| Command | Description | Example |
|---|---|---|
| **selectByVisibleText() and deselectByVisibleText()** | Selects/deselects the option that displays the text matching the parameter. | drpFruit.selectByVisibleText(“Mango”); |
| **selectByValue() and deselectByValue()** | Selects/deselects the option whose “value” attribute matches the specified parameter. | drpFruit.selectByValue(“123”); |
| **selectByIndex()** | Selects/deselects the option at the given index. | drpFruit.selectByIndex(0); |
| **isMultiple()** | Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise. | If(drpCountry.isMultiple()) { //Do something } |
| **deselectAll()** | Clears all selected entries. This is only valid when the drop-down element supports multiple selections. | drpContry.deselectAll(); |