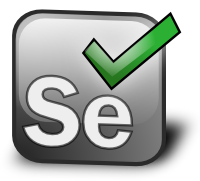
SELENIUM WEBDRIVER



**Vijay Kumar reddy**

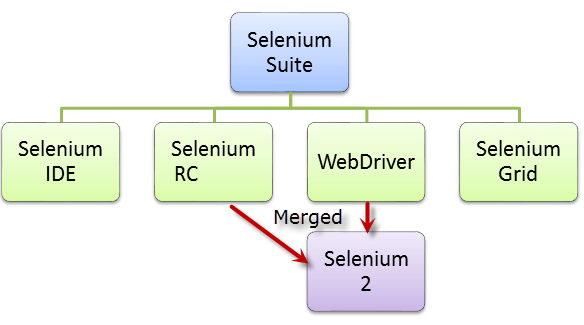
This Document Elaborately explains Starting from Scratch of Installation, Eclipse ,selenium IDE, Web Driver, TestNG.



|  |
| --- |
| INDEX  [1) Introduction](http://www.guru99.com/introduction-to-selenium.html) |
| [2) Install IDE & FireBug](http://www.guru99.com/install-selenuim-ide.html) |
| [3) Introduction IDE](http://www.guru99.com/introduction-selenuim-ide.html) |
| [4) First Script](http://www.guru99.com/first-selenium-test-script.html) |
| [5) Locators](http://www.guru99.com/locators-in-selenium-ide.html) |
| [6) Enhancements](http://www.guru99.com/enhancing-selenium-ide-script.html) |
| [7) Intro WebDriver](http://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html) |
| [8) Install Webdriver](http://www.guru99.com/installing-selenium-webdriver.html) |
| [9) First WebDriver Script](http://www.guru99.com/first-webdriver-script.html) |
| [10) Forms & Webdriver](http://www.guru99.com/accessing-forms-in-webdriver.html) |
| [11) Links & Tables](http://www.guru99.com/accessing-links-tables-selenium-webdriver.html) |
| [12) Keyboard Mouse Events](http://www.guru99.com/keyboard-mouse-events-files-webdriver.html) |
| [13) Selenium & TestNG](http://www.guru99.com/all-about-testng-and-selenium.html) |
| [14) Selenium Grid](http://www.guru99.com/introduction-to-selenium-grid.html) |

[1) Introduction](http://www.guru99.com/introduction-to-selenium.html)

|  |
| --- |
| **What is Selenium?**  **Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms.** It is quite similar to HP Quick Test Pro (QTP) only that Selenium focuses on automating web-based applications.  Selenium is not just a single tool but a suite of softwares, each catering to different testing needs of an organization. **It has four components.**   * Selenium Integrated Development Environment (IDE) * Selenium Remote Control (RC) * WebDriver * Selenium Grid |



At the moment, Selenium RC and WebDriver are merged into a single framework to form **Selenium 2**. Selenium 1, by the way, refers to Selenium RC.

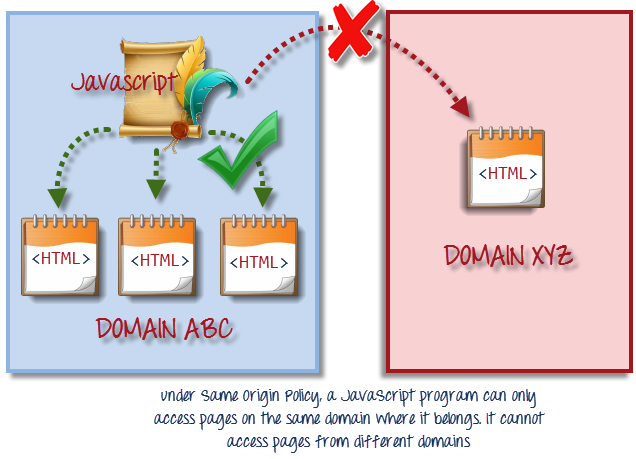
**Who developed Selenium?**

Since Selenium is a collection of different tools, it had different developers as well. Below are the key persons who made notable contributions to the Selenium Project

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/JasonHuggins.jpg | **Birth of Selenium Core**  Primarily, Selenium was **created by Jason Huggins in 2004**. An engineer at Thought Works, he was working on a web application that required frequent testing. Having realized that the repetitious manual testing of their application was becoming more and more inefficient, he created a JavaScript program that would automatically control the browser’s actions. He named this program as the “**JavaScriptTestRunner**.”  Seeing potential in this idea to help automate other web applications, he made JavaScript Runner open-source which was later re-named as **Selenium Core**. |
|  |  |

**The Same Origin Policy Issue**

**Same Origin policy prohibits JavaScript code from accessing elements from a domain that is different from where it was launched**. Example, the HTML code in www.google.com uses a JavaScript program “randomScript.js”. The same origin policy will only allow randomScript.js to access pages within google.com such as google.com/mail, google.com/login, or google.com/signup. However, it cannot access pages from different sites such as yahoo.com/search or guru99.com because they belong to different domains.



  This is the reason why prior to Selenium RC, testers needed to install local copies of both Selenium Core (a JavaScript program) and the web server containing the webapplication being tested so they would belong to the same domain

**Birth of Selenium Remote Control (Selenium RC)**



Unfortunately; testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the **same origin policy.** So another ThoughtWork’s engineer, **Paul Hammant**, decided to create a server that will act as an HTTP proxy to “trick” the browser into believing that Selenium Core and the web application being tested come from the same domain. This system became known as the **Selenium Remote Control** or **Selenium 1**.

**Birth of Selenium Grid**



Selenium Grid was developed by **Patrick Lightbody** to address the need of minimizing test execution times as much as possible. He initially called the system “**Hosted QA**.” It was capable of capturing browser screenshots during significant stages, and also of **sending out Selenium commands to different machines simultaneously.**

**Birth of Selenium IDE**

****

**Shinya Kasatani** of Japan created **Selenium IDE**, a Firefox extension that can automate the browser through a record-and-playback feature. He came up with this idea to further increase the speed in creating test cases. He donated Selenium IDE to the Selenium Project in **2006**.

**Birth of WebDriver**

****

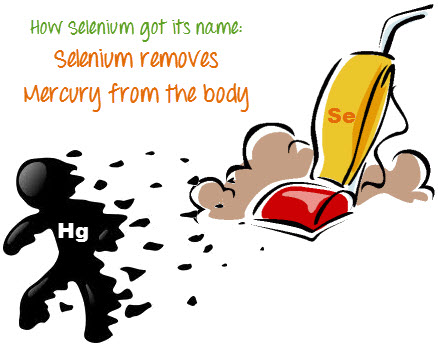
**Simon Stewart**created WebDriver circa **2006** when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core.**It was the first cross-platform testing framework that could control the browser from the OS level.**

**Birth of Selenium 2**

In **2008**, the whole Selenium Team decided to merge WebDriver and Selenium RC to form a more powerful tool  called **Selenium 2**, with **WebDriver being the core**. Currently, Selenium RC is still being developed but only in maintenance mode. Most of the Selenium Project’s efforts are now focused on Selenium 2.

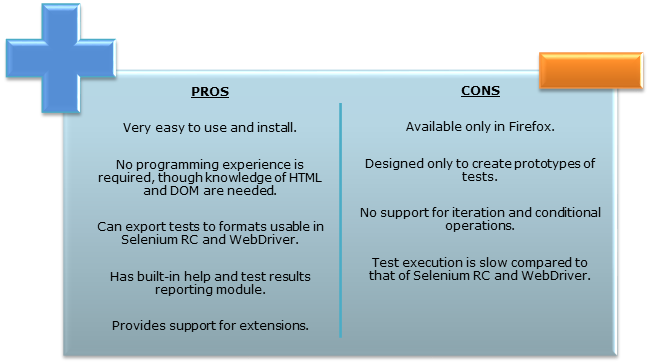
**So, Why the Name Selenium?**

It came from a joke which Jason cracked one time to his team. Another automated testing framework was popular during Selenium’s development, and it was by the company called **Mercury Interactive** (yes, the company who originally made QTP before it was acquired by HP). Since Selenium is a well-known antidote for Mercury poisoning, Jason suggested that name. His teammates took it, and so that is how we got to call this framework up to the present.



**Brief Introduction Selenium IDE**

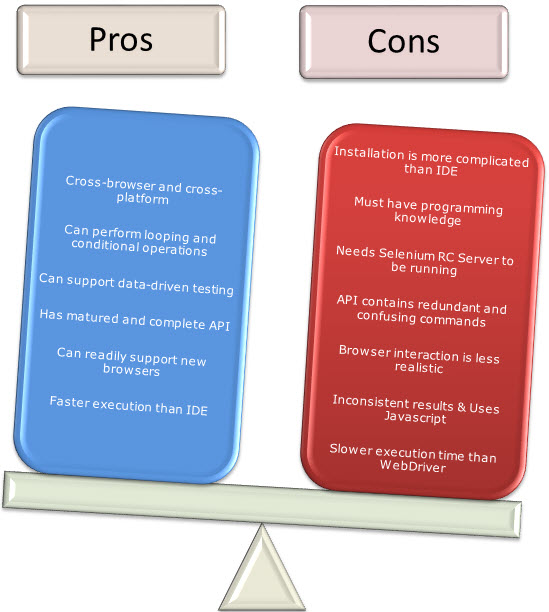
Selenium Integrated Development Environment (IDE) is the **simplest framework** in the Selenium suite and is **the easiest one to learn**. It is a **Firefox plugin** that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a **prototyping tool**. If you want to create more advanced test cases, you will need to use either Selenium RC or WebDriver.



**Brief Introduction Selenium Remote Control (Selenium RC)**

Selenium RC was the **flagship testing framework** of the whole Selenium project for a long time. This is the first automated web testing tool that **allowed users to use a programming language they prefer**.As of version 2.25.0, RC can support the following programming languages:

* Java
* C#
* PHP
* Python
* Perl
* Ruby

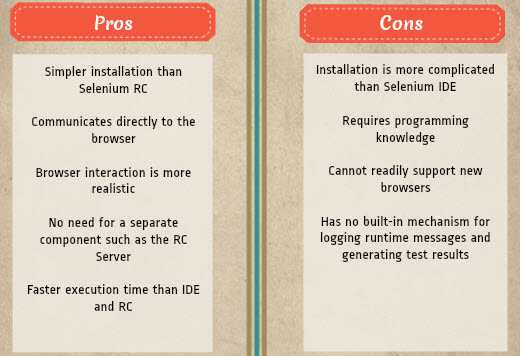


**Brief Introduction Web Driver**

The Web Driver proves itself to be **better than both Selenium IDE and Selenium RC** in many aspects. It implements a more modern and stable approach in automating the browser’s actions. Web Driver, unlike Selenium RC, does not rely on JavaScript for automation. **It controls the browser by directly communicating to it.**

The supported languages are the same as those in Selenium RC.

* Java
* C#
* PHP
* Python
* Perl
* Ruby



**Selenium Grid**

Selenium Grid is a tool **used together with Selenium RC to run parallel tests** across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

**Features:**

* Enables **simultaneous running of tests** in **multiple browsers and environments.**
* **Saves time** enormously.
* Utilizes the **hub-and-nodes** concept. The hub acts as a central source of Selenium commands to each node connected to it.

**Note on Browser and Environment Support**

Because of their architectural differences, Selenium IDE, Selenium RC, and WebDriver support different sets of browsers and operating environments.

|  | **Selenium IDE** | **Selenium RC** | **WebDriver** |
| --- | --- | --- | --- |
| **Browser**  **Support** | Mozilla Firefox | Mozilla Firefox  Internet Explorer  Google Chrome  Safari  Opera  Konqueror  Others | Internet Explorer versions 6 to 9, both 32 and 64-bit    Firefox 3.0, 3.5, 3.6, 4.0, 5.0, 6, 7 and above  (current version is 16.0.1)    Google Chrome 12.0.712.0 and above  (current version is 22.0.1229.94 m)    Opera 11.5 and above  (current version is 12.02)    Android – 2.3 and above for phones and tablets  (devices & emulators)    iOS 3+ for phones (devices & emulators) and 3.2+ for tablets (devices & emulators)    HtmlUnit 2.9 and above  (current version is 2.10) |
| **Operating System** | Windows  Mac OS X  Linux | Windows  Mac OS X  Linux  Solaris | All operating systems where the browsers above can run. |

**How to Choose the Right Selenium Tool for Your Need?**

| **Tool** | **Why Choose ?** |
| --- | --- |
| **Selenium IDE** | * To learn about concepts on automated testing and Selenium, including: * Selenese commands such as type, open, clickAndWait, assert, verify, etc. * Locators such as id, name, xpath, css selector, etc. * Executing customized JavaScript code using runScript * Exporting test cases in various formats. * To create tests with little or no prior knowledge in programming. * To create simple test cases and test suites that you can export later to RC or WebDriver. * To test a web application against Firefox only. |
| **Selenium RC** | * To design a test using a more expressive language than Selenese * To run your test against different browsers (except HtmlUnit) on different operating systems. * To deploy your tests across multiple environments using Selenium Grid. * To test your application against a new browser that supports JavaScript. * To test web applications with complex AJAX-based scenarios. |
| **WebDriver** | * To use a certain programming language in designing your test case. * To test applications that are rich in AJAX-based functionalities. * To execute tests on the HtmlUnit browser. * To create customized test results. |
| **Selenium Grid** | * To run your Selenium RC scripts in multiple browsers and operating systems simultaneously. * To run a huge test suite, that need to complete in soonest time possible. |

**A Comparison between Selenium and QTP**

**Quick Test Professional(QTP)** is a proprietary automated testing tool previously owned by the company **Mercury Interactive** before it was **acquired by Hewlett-Packard in 2006**. The Selenium Tool Suite has many advantages over  QTP (as of version 11) as detailed below –

**Advantages of Selenium over QTP**

|  |  |
| --- | --- |
| **Selenium** | **QTP** |
| **Open source**, **free to use**, and **free of charge.** | **Commercial**. |
| **Highly extensible** | Limited add-ons |
| Can run tests across **different browsers** | Can only run tests in **Firefox** , **Internet Explorer** and **Chrome** |
| Supports **various operating systems** | Can only be used in **Windows** |
| Supports **mobile devices** | Supports mobile devise using 3rd party software |
| Can execute tests **while** the **browser is minimized** | Needs to have the application under test to be visible on the desktop |
| Can execute tests **in parallel**. | Can only execute in parallel but using Quality Center which is again a paid product. |

**Advantages of QTP over Selenium**

|  |  |
| --- | --- |
| **QTP** | **Selenium** |
| Can test **both web and desktop applications** | Can only test web applications |
| Comes with a **built-in object repository** | Has no built-in object repository |
| **Automates faster than Selenium**because it is a fully featured IDE. | Automates at a slower rate because it does not have a native IDE and only third party IDE can be used for development |
| Data-driven testing is easier to perform because **it has built-in global and local data tables**. | Data-driven testing is more cumbersome since you have to rely on the programming language’s capabilities for setting values for your test data |
| **Can access controls within the browser**(such as the Favorites bar, Address bar, Back and Forward buttons, etc.) | Cannot access elements outside of the web application under test |
| Provides professional **customer support** | No official user support is being offered. |
| Has native capability to **export test data** into external formats | Has no native capability to export runtime data onto external formats |
| Parameterization Support is in built | Parameterization can be done via programming but is difficult to implement. |
| Test Reports are generated automatically | No native support to generate test /bug reports. |

Though clearly, QTP has more advanced capabilities, Selenium outweighs QTP in three main areas:

* **Cost**(because Selenium is completely free)
* **Flexibility**(because of a number of programming languages, browsers, and platforms it can support)
* **Parallel testing**(something that QTP is capable of but only with use of Quality Center)

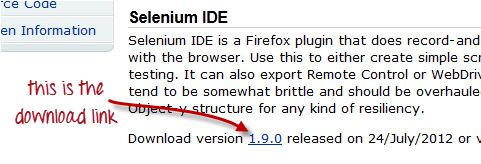
**Summary**

* The entire Selenium Tool Suite is comprised of four components:
* **Selenium IDE**, a Firefox add-on that you can only use in creating relatively simple test cases and test suites.
* **Selenium Remote Control**, also known as **Selenium 1**, which is the first Selenium tool that allowed users to use programming languages in creating complex tests.
* **WebDriver**, the newer breakthrough that allows your test scripts to communicate directly to the browser, thereby controlling it from the OS level.
* **Selenium Grid**is also a tool that is used with Selenium RC to execute parallel tests across different browsers and operating systems.
* Selenium RC and WebDriver was merged to form **Selenium 2**.
* Selenium is more advantageous than QTP in terms of **costs and flexibility**. It also allows you to **run tests in parallel**, unlike in QTP where you are only allowed to run tests sequentially.

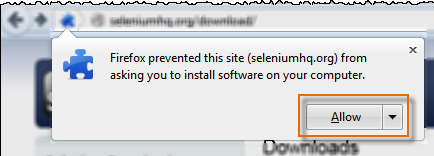
|  |
| --- |
| Installation of Selenium IDE What you need   * Mozilla Firefox * Active internet connection   If you do not have Mozilla Firefox yet, you can download it from <http://www.mozilla.org/en-US/firefox/new>. |

### Steps

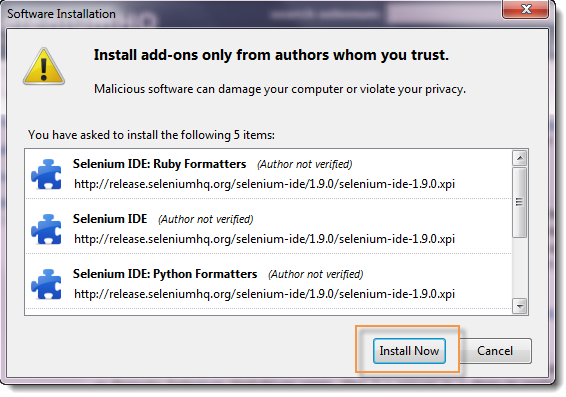
Launch Firefox and navigate to <http://seleniumhq.org/download/>. Under the **Selenium IDE** section, click on the link that shows the current version number.



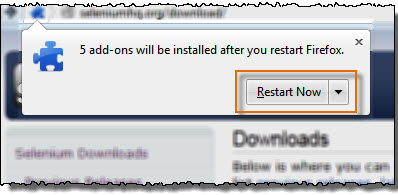
For security, a Firefox notification will pop up. Click on “**Allow**.”



Wait until Firefox completes the download and then click “**Install Now**.”

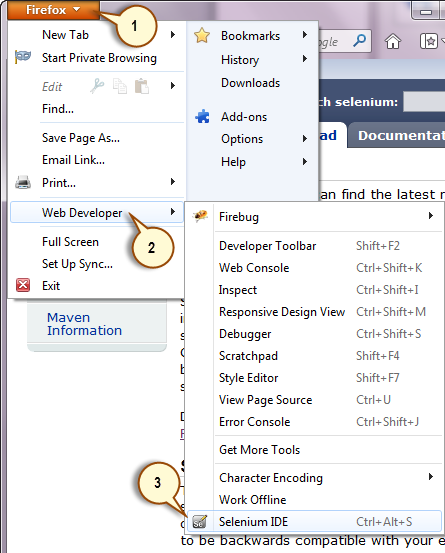


Wait until installation is completed. In the pop-up window, click “**Restart Now**.”

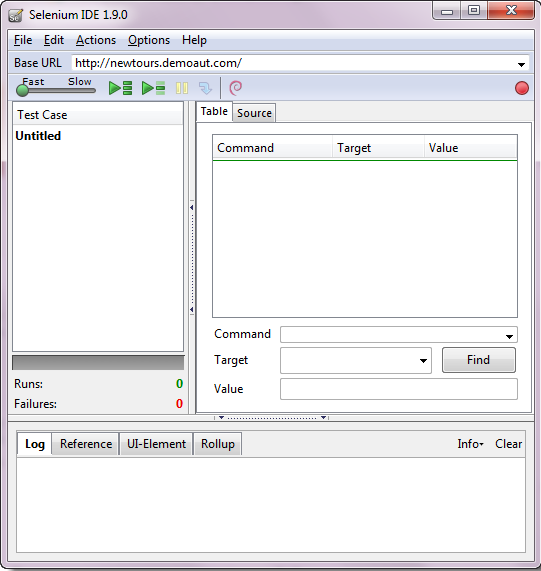


After Firefox has restarted, **launch Selenium IDE** using either of two ways:

* By pressing **Ctrl+Alt+S**
* By clicking on the **Firefox menu button** > **Web Developer**>  **Selenium IDE**



Selenium IDE should launch as shown below

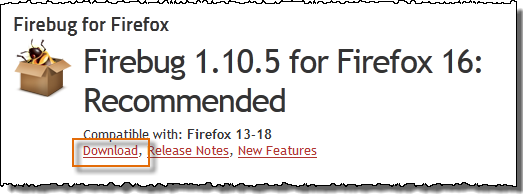


## Installation of Firebug

**Firebug**is a Firefox add-on that we will use to **inspect the HTML elements** of the web application under test. It will provide us the name of the element that our Selenese command would act upon.

**Step 1**

Use Firefox to navigate to Firebug’s download page (<https://getfirebug.com/downloads/>) and click on the download link.



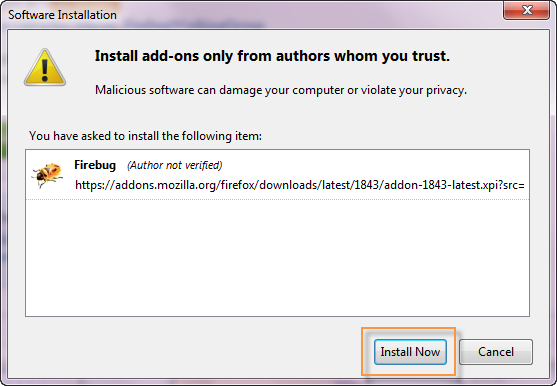
**Step 2**

Firefox will take you to its Firebug download section. Click the **“Add to Firefox”** button.



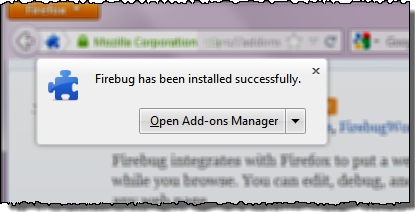
**Step 3**

Wait for Firefox to complete downloading this add-on. On the dialog box that comes after, click **“Install Now.”**



**Step 4**

Wait for installation to complete. A notification will pop-up saying, “Firebug has been installed successfully.” You can immediately close this pop-up.



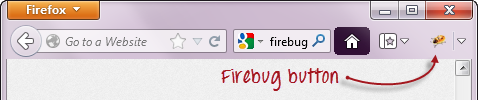
**Note**: In case you do not see above pop-up , no worries! This pop-up appears for a few seconds and disappears.

You do not need to restart Firefox after installing Firebug.

**Step 5**

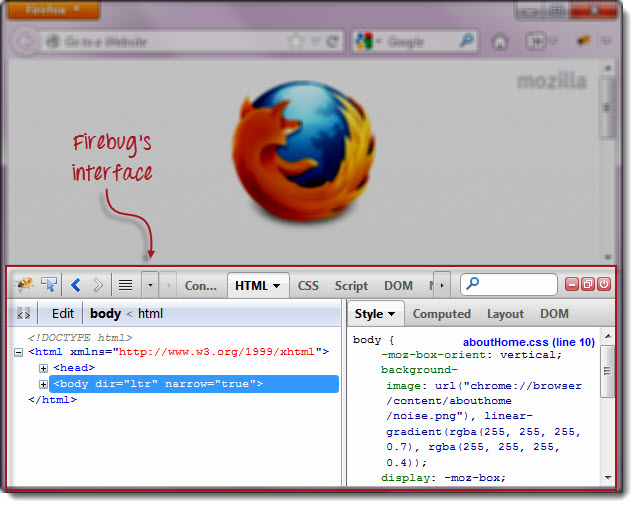
Launch Firebug by doing either of these two methods:

* Press **F12**
* Click on the **Firebug button** on the upper right corner of the Firefox window.



**Step 6**

Firebug should launch **at the bottom of Firefox** as shown below



## Plugins

**Selenium IDE can support additional Firefox add-ons or plugins created by other users**. You can visit [here](https://addons.mozilla.org/en-US/firefox/search/?q=selenium&appver=16.0&platform=windows)  for a list of Selenium add-ons available to date. Install them just as you do with other Firefox add-ons.

By default, Selenium IDE comes bundled with 4 plugins:

1.    Selenium IDE: C# Formatters

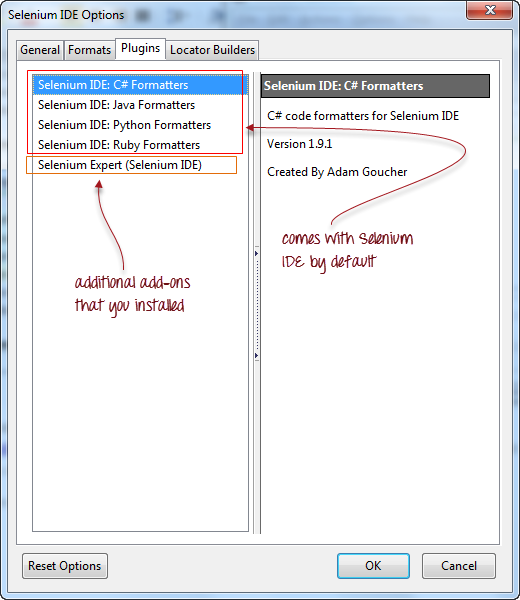
2.    Selenium IDE: Java Formatters

3.    Selenium IDE: Python Formatters

4.    Selenium IDE: Ruby Formatters

These four plugins are required by Selenium IDE to convert Selenese into different formats.

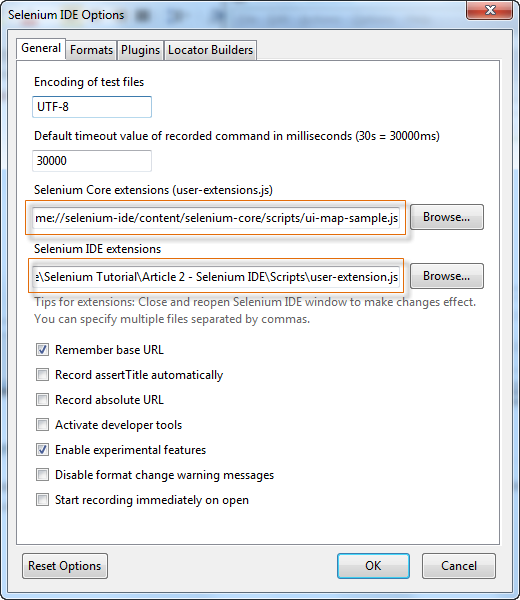
**The Plugins tab shows a list of all your installed add-ons, together with the version number and name of the creator of each.**



## User Extensions

Selenium IDE can support user extensions to provide advanced capabilities. User extensions are in the form of JavaScript files. You install them by specifying their absolute path in either of these two fields in the Options dialog box.

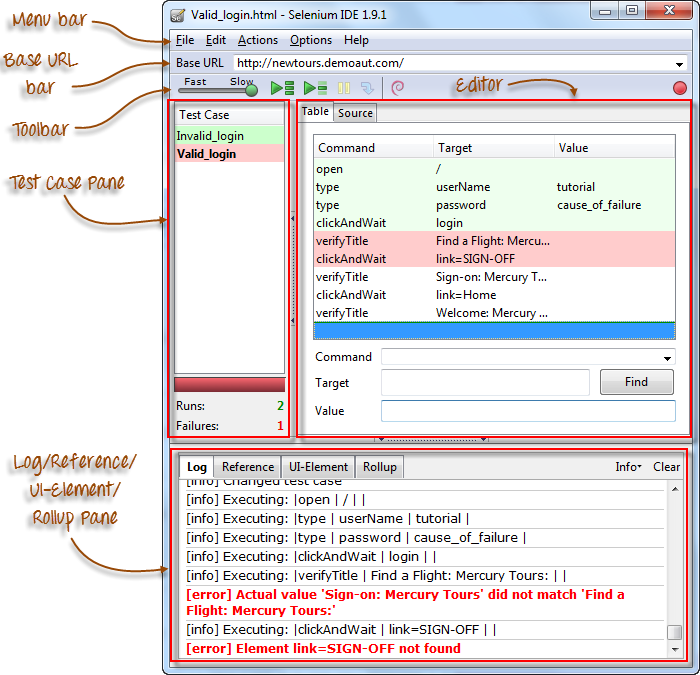
* Selenium Core extensions (user-extensions.js)
* Selenium IDE extensions



You will be able to find tons of user extensions [here](http://wiki.openqa.org/display/SEL/Contributed+User-Extensions).

|  |
| --- |
| **Selenium IDE (Integrated Development Environment) is the simplest tool in the Selenium Suite**. It is **a Firefox add-on that creates tests very quickly through its record-and-playback functionality**. This feature is similar to that of QTP. It is effortless to install and easy to learn.  Because of its simplicity, **Selenium IDE should only be used as a prototyping tool** – **not an overall solution for developing and maintaining complex test suites.**  Though you will be able to use Selenium IDE without prior knowledge in programming, **you should at least be familiar with HTML, JavaScript, and the DOM (Document Object Model)** to utilize this tool to its full potential. Knowledge of JavaScript will be required when we get to the section about the Selenese command “**runScript”**.  Selenium IDE supports autocomplete mode when creating tests. This feature serves two purposes:   * It helps the tester to enter commands more quickly. * It restricts the user from entering invalid commands. |

## Features of Selenium IDE

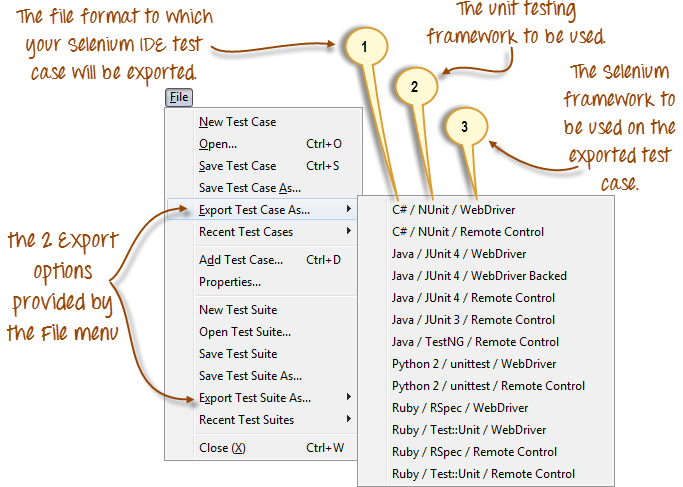


### Menu Bar

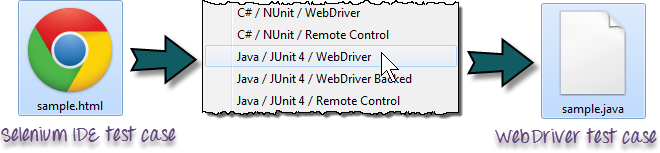
It is located at the **topmost portion** of the IDE. The most commonly used menus are the File, Edit, and Options menus.

**File menu**

* It contains options to create, open, save, and close tests.
* Tests are **saved in HTML format**.
* The most useful option is “**Export”** because **it allows you to turn your Selenium IDE test cases into file formats that can run on Selenium Remote Control and WebDriver**
* **“Export Test Case As…”** will export only the currently opened test case.
* **“Export Test Suite As…”** will export all the test cases in the currently opened test suite.



* As of **Selenium IDE v1.9.1**, test cases can be exported only to the following formats:
* .cs (C# source code)
* .java (Java source code)
* .py (Python source code)
* .rb (Ruby source code)

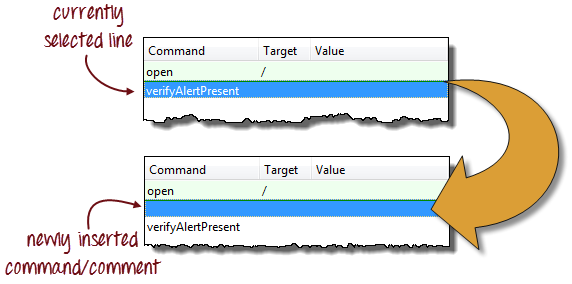


**Edit Menu**

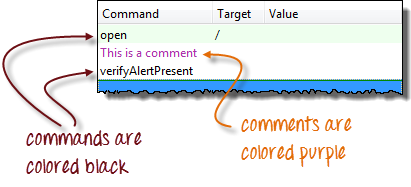
* It contains usual options like Undo, Redo, Cut, Copy, Paste, Delete, and Select All.
* The two most important options are the “**Insert New Command**” and “**Insert New Comment**”.



* The newly inserted command or comment **will be placed on top of the currently selected line**.



* **Commands**are colored **black**.
* **Comments**are colored **purple.**



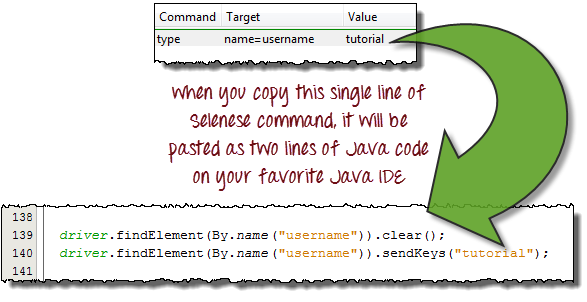
### ****Options menu****

It provides the **interface for configuring various settings** of Selenium IDE.

We shall concentrate on the **Options** and **Clipboard Format** options.

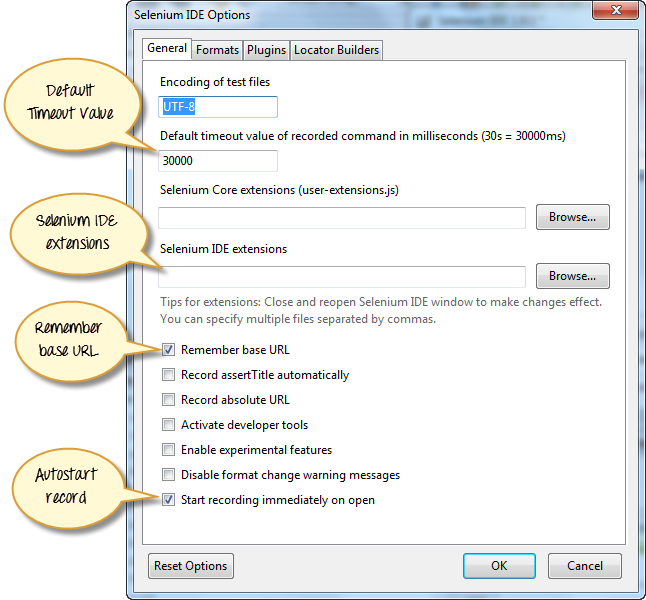
|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/clipboard_format_menu_list_complete.png | **Clipboard Format**     * **The Clipboard Format allows you to copy a Selenese command from the editor and paste it as a code snippet**. * The format of the code follows the option you selected here in Clipboard Format’s list. * **HTML is the default selection.** |

For example, when you choose **Java/JUnit 4/WebDriver** as your clipboard format, every Selenese command you copy from Selenium IDE’s editor will be pasted as **Java code**. See the illustration below.

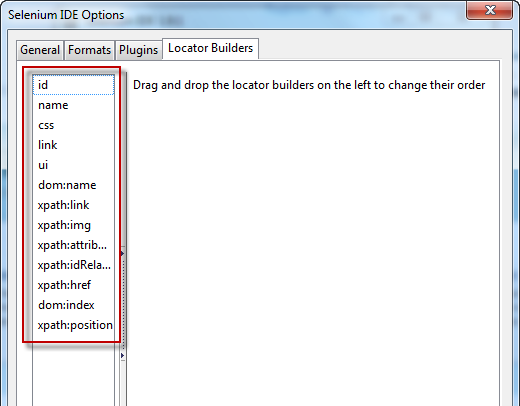


### Selenium IDE Options dialog box

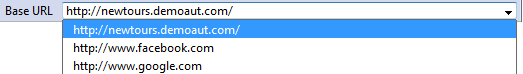
You can launch the Selenium IDE Options dialog box by clicking Options > Options… on the menu bar. Though there are many settings available, we will concentrate on the few important ones.



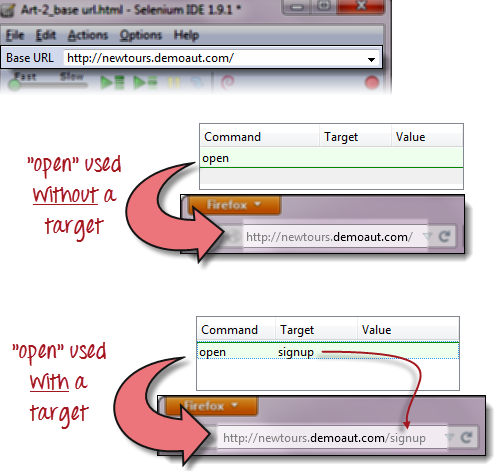
* **Default Timeout Value**. This refers to the time that Selenium has to wait for a certain element to appear or become accessible before it generates an error.  **Default timeout value is 30000ms**.
* **Selenium IDE extensions**. This is where you specify the extensions you want to use to extend Selenium IDE’s capabilities. You can visit <http://addons.mozilla.org/en-US/firefox/>and use “Selenium” as keyword to search for specific extensions.
* **Remember base URL.** Keep this checked if you want Selenium IDE to remember the Base URL every time you launch it. If you uncheck this, Selenium IDE will always launch with a blank value for the Base URL.
* **Autostart record.** If you check this, Selenium IDE will immediately record your browser actions  upon startup.
* **Locator builders.**This is where you specify the order by which locators are generated while recording. **Locators are ways to tell Selenium IDE which UI element should a Selenese command act upon**. In the setup below, when you click on an element with an ID attribute, that element’s ID will be used as the locator since “id” is the first one in the list. If that element does not have an ID attribute, Selenium will next look for the “name” attribute since it is second in the list. The list goes on and on until an appropriate one is found.



### Base URL Bar



* It has **a dropdown menu that** **remembers all previous values** for easy access.
* The Selenese command **“open” will take you to the URL that you specified in the Base URL**.
* In this tutorial series, we will be using [http://newtours.demoaut.com](http://newtours.demoaut.com/) as our Base URL. It is the site for Mercury Tours, a web application maintained by HP for web testing purposes. We shall be using this application because it contains a complete set of elements that we need for the succeeding topics.
* **The Base URL is very useful in accessing relative URLs**. Suppose that your Base URL is set to [http://newtours.demoaut.com](http://newtours.demoaut.com/). When you execute the command “open” with the target value “signup”, Selenium IDE will direct the browser to <http://newtours.demoaut.com/signup>. See the illustration below.



### Toolbar

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/fast-slow.png | **Playback Speed**. This controls the speed of your Test Script Execution. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/record.png | **Record.**This starts/ends your recording session.  Each browser action is entered as a Selenese command in the Editor. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/play_entire_test_suite.png | **Play entire test suite**. This will sequentially play all the test cases listed in the Test Case Pane. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/play_current_test_case.png | **Play current test case**. This will play only the currently selected test case in the Test Case Pane. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/pause_resume_2.png | **Pause/Resume**. This will pause or resume your playback. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/step.png | **Step**. This button will allow you to step into each command in your test script. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/rollup_button.png | **Apply rollup rules**. This is an advanced functionality. It allows you to group Selenese commands together and execute them as a single action. |

### Test Case Pane

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/test_case_pane_-_final.png | * In Selenium IDE, you can open **more than one test case at a time**. * **The test case pane shows you the list of currently opened test cases.** * When you open a test suite, the test case pane will **automatically list all the test cases** contained in it. * The test case written in **bold font** is the **currently selected test case** * After playback, **each test case is color-coded** to represent if it passed or failed.   + Green color means “Passed.”   + Red color means “Failed.” * At the bottom portion is a summary of the number of test cases that were run and failed. |

### Editor

You can think of the editor as **the place where all the action happens**. It is available in two views: Table and Source.

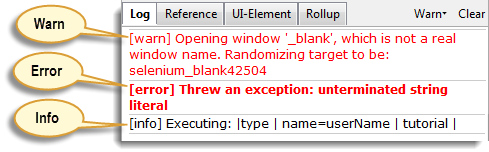
|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/Editor.png | **Table View**     * Most of the time, you will work on Selenium IDE using the **Table View**. * This is **where you create and modify Selenese commands.** * After playback, each step is color-coded. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/TableEditor.png | * To create steps, type the name of the command in the “Command” text box. * **It displays a dropdown list of commands**that match with the entry that you are currently typing. * Target is any parameter (like username , password) for a command and Value is the input value (like tom,123pass) for those Targets. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/editor_source_-_final.png | **Source View**     * **It displays the steps in HTML (default) format.** * **It also allows you to edit your script**just like in the Table View. |

### Log Pane

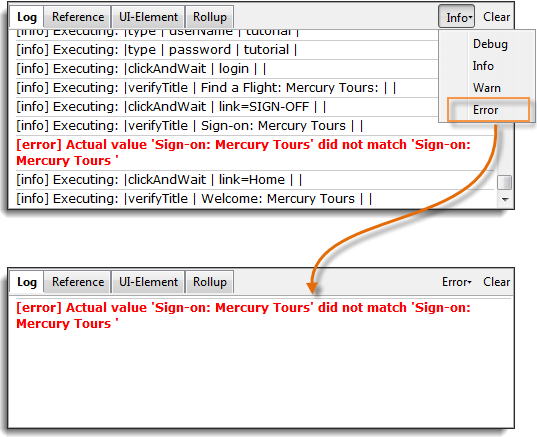
**The Log Pane displays runtime messages**during execution. It provides real-time updates as to what Selenium IDE is doing.

**Logs are categorized into four types:**

* Debug – By default, Debug messages are not displayed in the log panel. They show up only when you filter them. They provide technical information about what Selenium IDE is doing behind the scenes. It may display messages such as a specific module has done loading, a certain function is called, or an external JavaScript file was loaded as an extension.
* Info – It says which command Selenium IDE is currently executing.
* Warn – These are warning messages that are encountered in special situations.
* Error – These are error messages generated when Selenium IDE fails to execute a command, or if a condition specified by “verify” or “assert” command is not met.

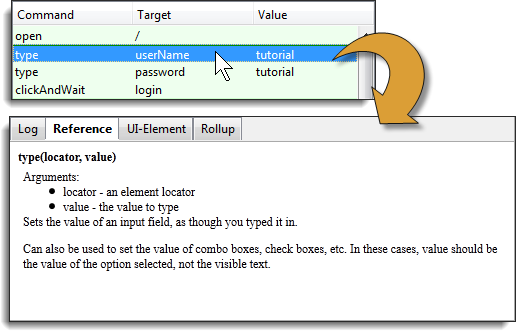


**Logs can be filtered by type**. For example, if you choose to select the “Error” option from the dropdown list, the Log Pane will show error messages only.



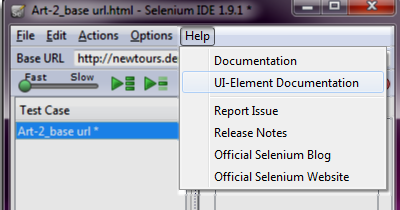
### Reference Pane

**The Reference Pane shows a concise description of the currently selected Selenese command in the Editor**. It also shows the **description about the locator and value** to be used on that command.

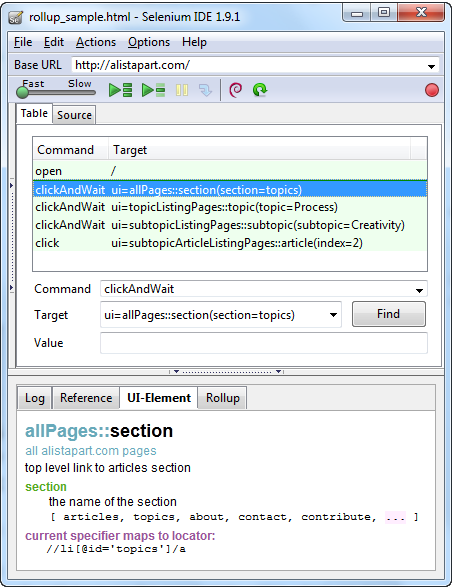


### UI-Element Pane

The UI-Element is for advanced Selenium users. **It uses JavaScript Object Notation (JSON) to define element mappings.** The documentation and resources are found in the “UI Element Documentation” option under the Help menu of Selenium IDE.



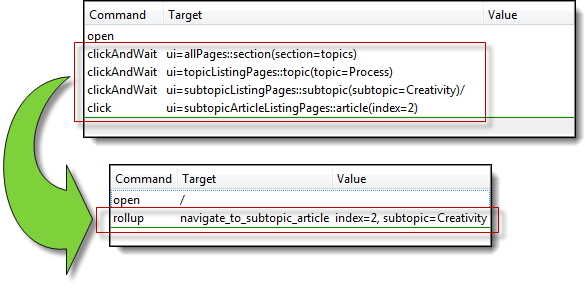
An example of a UI-element screen is shown below.



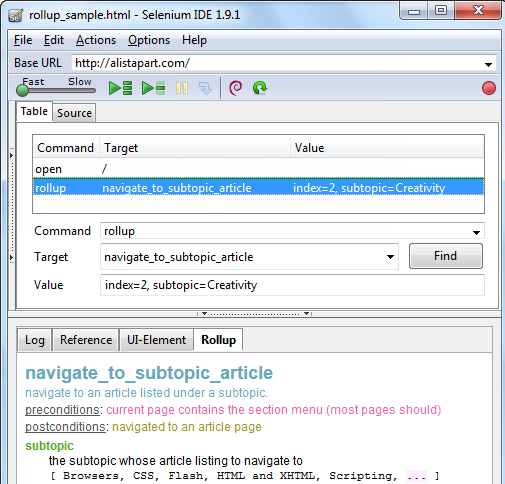
### Rollup Pane

**Rollup allows you to execute a group of commands in one step.**A group of commands is simply called as a “rollup.” It employs heavy use of JavaScript and UI-Element concepts to formulate a collection of commands that is similar to a “function” in programming languages.

**Rollups are reusable**; meaning, they can be used multiple times within the test case. Since rollups are groups of commands condensed into one, they contribute a lot in shortening your test script.



An example of how the contents of the rollup tab look like is shown below.



## Summary

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/summary_baby.png | * Selenium IDE (Integrated Development Environment) **is the simplest tool** in the Selenium Suite. * It must only be used as a **prototyping tool**. * **Knowledge of JavaScript and HTML is required for intermediate topics**such as executing the “runScript” and “rollup” commands.A **rollup** is a collection of commands that you can reuse to shorten your test scripts significantly.**Locators**are identifiers that tell Selenium IDE how to access an element. * **Firebug** (or any similar add-on) is used to obtain locator values**.** * The **menu bar** is used in creating, modifying, and exporting test cases into formats useable by Selenium RC and WebDriver. * The **default format for Selenese commands is HTML**. * The **“Options” menu provides access to various configurations** for Selenium IDE. * The **Base URL** is useful in accessing **relative URLs**. |

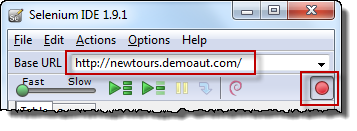
* The **Test Case Pane** shows the list of currently opened test cases and a concise summary of test runs.
* The **Editor** provides the **interface for your test scripts**.
* The **Table View** shows your script **in** **tabular format** with “Command”, “Target”, and “Value” as the columns.
* The **Source View** shows your script **in HTML format**.
* The **Log** and **Reference** tabs give feedback and other useful information when executing tests.
* The **UI-Element and Rollup** tabs are **for advanced Selenium IDE users only**. They both require considerable effort in coding JavaScript.
* **UI-Element**allows you to **conveniently map UI elements** using JavaScript Object Notation (JSON).

|  |
| --- |
| He will use the Mercury Tours website as our web application under test. It is an online flight reservation system that contains all the elements we need for this tutorial. Its URL is <http://newtours.demoaut.com/> and this will be our Base URL.  **NOTE**: The site http://newtours.demoaut.com/ at time is down and not available to work on. We have raised this issue with HP |

## Create a Script by Recording

Let us now create our first test script in Selenium IDE using the most common method – by recording. Afterwards, we shall execute our script using the playback feature.

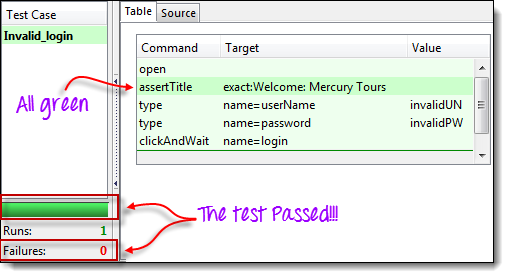
|  |
| --- |
| **Step 1**   * Launch Firefox and Selenium IDE. * Type the value for our Base URL: <http://newtours.demoaut.com/>. * Toggle the Record button on (if it is not yet toggled on by default). |



|  |  |
| --- | --- |
| **Step 2**  In Firefox, navigate to <http://newtours.demoaut.com/>. Firefox should take you to the page similar to the one shown below.  http://www.guru99.com/images/step_2%281%29.png | |
| **Step 3**   * Right-click on any blank space within the page, like on the Mercury Tours logo on the upper left corner. This will bring up the Selenium IDE context menu. Note: Do not click on any hyperlinked objects or images * Select the “Show Available Commands” option. * Then, select “assertTitle exact:Welcome: Mercury Tours”. This is a command that makes sure that the page title is correct. | |
| http://www.guru99.com/images/step_3%281%29.png  http://newguru99.revolutionventur.netdna-cdn.com/images/editor_-_after_assert_title_-_before_logging_in.png | |
| **Step 4**   * In the “User Name” text box of Mercury Tours, type an invalid username, “invalidUN”. * In the “Password” text box, type an invalid password, “invalidPW”. | |
| http://newguru99.revolutionventur.netdna-cdn.com/images/Step4a.png | http://newguru99.revolutionventur.netdna-cdn.com/images/step4b.png |
| **Step 5**   * Click on the “Sign-In” button. Firefox should take you to this page.     http://newguru99.revolutionventur.netdna-cdn.com/images/step_5a.png | |
| **Step 6**  Toggle the record button off to stop recording. Your script should now look like the one shown below.  http://newguru99.revolutionventur.netdna-cdn.com/images/step_6.png | |
| **Step 7**  Now that we are done with our test script, we shall save it in a test case. In the File menu, select “Save Test Case”. Alternatively, you can simply press Ctrl+S.  http://newguru99.revolutionventur.netdna-cdn.com/images/step_7.png | |
| **Step 8**   * Choose your desired location, and then name the test case as “Invalid\_login”. * Click the “Save” button.   http://newguru99.revolutionventur.netdna-cdn.com/images/step_8.png | |
| **Step 9.**  Notice that the file was saved as HTML.  http://newguru99.revolutionventur.netdna-cdn.com/images/step_9.png | |

**Step 10.**

Go back to Selenium IDE and click the Playback button to execute the whole script. Selenium IDE should be able to replicate everything flawlessly.



## Introduction to Selenium Commands – Selenese

* Selenese commands can have up to a maximum of two parameters: target and value.
* Parameters are not required all the time. It depends on how many the command will need.
* For a complete reference of Selenese commands, click [here](http://release.seleniumhq.org/selenium-core/1.0.1/reference.html)

### ****3 Types of Commands****

|  |  |
| --- | --- |
| **Actions** | These are commands that directly interact with page elements.  Example: the “click” command is an action because you directly interact with the element you are clicking at.  The “type” command is also an action because you are putting values into a text box, and the text box shows them to you in return. There is a two-way interaction between you and the text box. |
| **Accessors** | They are commands that allow you to store values to a variable.  Example: the “storeTitle” command is an accessor because it only “reads” the page title and saves it in a variable. It does not interact with any element on the page. |
| **Assertions** | They are commands that verify if a certain condition is met.  **3 Types of Assertions**   * **Assert**. When an “assert” command fails, the test is stopped immediately. * **Verify**. When a “verify” command fails, Selenium IDE logs this failure and continues with the test execution. * **WaitFor**. Before proceeding to the next command, “waitFor” commands will first wait for a certain condition to become true.   + If the condition becomes true within the waiting period, the step passes.   + If the condition does not become true, the step fails. Failure is logged, and test execution proceeds to the next command.   + By default, timeout value is set to 30 seconds. You can change this in the Selenium IDE Options dialog under the General tab. |

**Assert vs. Verify**

|  |
| --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/assert.png |
| http://newguru99.revolutionventur.netdna-cdn.com/images/verify.png |

### ****Common Commands****

|  |  |  |
| --- | --- | --- |
| **Command** | **Number of Parameters** | **Description** |
| open | 0 - 2 | Opens a page using a URL. |
| click/clickAndWait | 1 | Clicks on a specified element. |
| type/typeKeys | 2 | Types a sequence of characters. |
| verifyTitle/assertTitle | 1 | Compares the actual page title with an expected value. |
| verifyTextPresent | 1 | Checks if a certain text is found within the page. |
| verifyElementPresent | 1 | Checks the presence of a certain element. |
| verifyTable | 2 | Compares the contents of a table with expected values. |
| waitForPageToLoad | 1 | Pauses execution until the page is loaded completely. |
| waitForElementPresent | 1 | Pauses execution until the specified element becomes present. |

## Create a Script Manually with Firebug

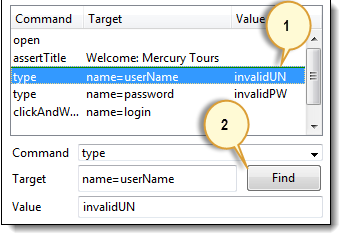
Now, we shall recreate the same test case manually, by typing in the commands. This time, we will need to use Firebug.

|  |
| --- |
| **Step 1**   * Open Firefox and Selenium IDE. * Type the base URL (<http://newtours.demoaut.com/>). * The record button should be OFF.   http://newguru99.revolutionventur.netdna-cdn.com/images/ToggleOff.png |
| **Step 2**  Click on the topmost blank line in the Editor.  http://newguru99.revolutionventur.netdna-cdn.com/images/editor_-_step_2a.png |
| Type “open” in the Command text box and press Enter.  http://newguru99.revolutionventur.netdna-cdn.com/images/ide_-_step_2.png |
| **Step 3**   * Navigate Firefox to our base URL and activate Firebug * In the Selenium IDE Editor pane, select the second line (the line below the “open” command) and create the second command by typing “assertTitle” on the Command box. * Feel free to use the autocomplete feature.   http://www.guru99.com/images/autocomplete%281%29.png |
| **Step 4**   * In Firebug, expand the <head> tag to display the <title> tag. * Click on the value of the <title> tag (which is “Welcome: Mercury Tours”) and paste it onto the Target field in the Editor.   http://newguru99.revolutionventur.netdna-cdn.com/images/Step4manual.png |
| **Step 5**   * To create the third command, click on the third blank line in the Editor and key-in “type” on the Command text box. * In Firebug, click on the “Inspect” button.   http://newguru99.revolutionventur.netdna-cdn.com/images/Step5mnual.png |
| Click on the User Name text box. Notice that Firebug automatically shows you the HTML code for that element.  http://newguru99.revolutionventur.netdna-cdn.com/images/ste5manual1.png |
| **Step 6**  Notice that the User Name text box does not have an ID, but it has a NAME attribute. We shall, therefore, use its NAME as the locator. Copy the NAME value and paste it onto the Target field in Selenium IDE.  http://newguru99.revolutionventur.netdna-cdn.com/images/step6a.png    Still in the Target text box, prefix “userName” with “name=”, indicating that Selenium IDE should target an element whose NAME attribute is “userName.”  http://newguru99.revolutionventur.netdna-cdn.com/images/step6b.png  Type “invalidUN” in the Value text box of Selenium IDE. Your test script should now look like the image below. We are done with the third command. Note: Instead of invalidUN , you may enter any other text string. But Selenium IDE is case sensitive and you type values/attributes exactly like in application.  http://newguru99.revolutionventur.netdna-cdn.com/images/Step6c.png |
| **Step 7**   * To create the fourth command, key-in “type” on the Command text box. * Again, use Firebug’s “Inspect” button to get the locator for the “Password” text box.    http://newguru99.revolutionventur.netdna-cdn.com/images/firebug_-_step_8.png  * Paste the NAME attribute (“password”) onto the Target field and prefix it with “name=” * Type “invalidPW” in the Value field in Selenium IDE. Your test script should now look like the image below.   http://newguru99.revolutionventur.netdna-cdn.com/images/ide_-_step_8.png |
| **Step 8**   * For the fifth command, type “clickAndWait” on the Command text box in Selenium IDE. * Use Firebug’s “Inspect” button to get the locator for the “Sign In” button.   http://newguru99.revolutionventur.netdna-cdn.com/images/firebug_-_step_9.png   * Paste the value of the NAME attribute (“login”) onto the Target text box and prefix it with “name=”. * Your test script should now look like the image below.   http://newguru99.revolutionventur.netdna-cdn.com/images/Step8a.png |
| **Step 9**  Save the test case in the same way as we did in the previous section. |

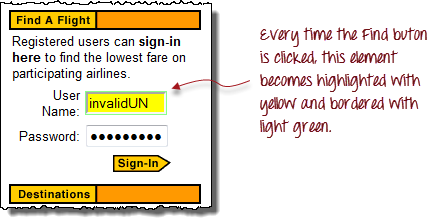
## Using the Find Button

**The Find button in Selenium IDE is used to verify if what we had put in the Target text box is indeed the correct UI element.**

Let us use the Invalid\_login test case that we created in the previous sections. Click on any command with a Target entry, say, the third command.



Click on the Find button. Notice that the User Name text box within the Mercury Tours page becomes highlighted for a second.



This indicates that Selenium IDE was able to detect and access the expected element correctly. If the Find button highlighted a different element or no element at all, then there must be something wrong with your script.

## Execute Command

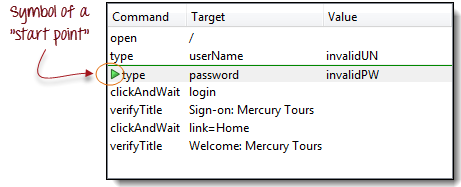
**This allows you to execute any single command without running the whole test case**. Just click on the line you wish to execute and then either click on “Actions > Execute this command” from the menu bar or simply press “X” on your keyboard.

|  |
| --- |
| **Step 1.** Make sure that your browser is on the Mercury Tours homepage. Click on the command you wish to execute. In this example, click on the “type | userName | invalidUN” line.  http://newguru99.revolutionventur.netdna-cdn.com/images/execute1.png |
| **Step 2.** Press “X” on your keyboard. |
| **Step 3.** Observe that the text box for username becomes populated with the text “invalidUN”  http://newguru99.revolutionventur.netdna-cdn.com/images/execute2.png |

**Executing commands this way is highly dependent on the page that Firefox is currently displaying**. This means that if you try the example above with the Google homepage displayed instead of Mercury Tours’, then your step will fail because there is no text box with a “userName” attribute within Google’s homepage.

## Start point

**A start point is an indicator that tells Selenium IDE which line the execution will start**. **Its shortcut key is “S”.**

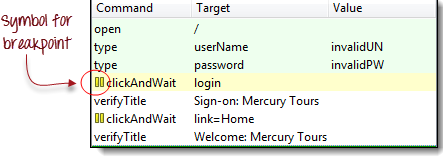


In the example above, playback will start on the third line (type | password | invalidPW). **You can only have one start point in a single test script.**

Start point is similar to Execute Command in such that they are dependent on the currently displayed page. The start point will fail if you are on the wrong page.

## Breakpoints

Breakpoints are indicators that tell Selenium IDE where to automatically pause the test. **The shortcut key is “B”.**



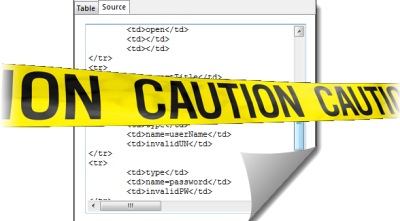
The yellow highlight means that the current step is pending. This proves that Selenium IDE has paused execution on that step. **You can have multiple breakpoints in one test case.**

## Step

It allows you to execute succeeding commands one at a time after pausing the test case. Let us use the scenario in the previous section “Breakpoints.”

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/Stepa.png | **Before clicking “Step.”**  The test case pauses at the line “clickAndWait | login”. |
| http://newguru99.revolutionventur.netdna-cdn.com/images/Stepb.png | **After clicking “Step.”**  The “clickAndWait | login” line is run and pauses to the next command (verifyTitle | Sign-on: Mercury Tours).    Notice that the next line is paused even though there is no breakpoint there. This is the main purpose of the Step feature – it executes the succeeding commands one at a time to give you more time to inspect the outcome after each step. |

## Important Things to Note When Using Other Formats in Source View



**Selenium IDE works well only with HTML – other formats are still in experimental mode**. It is **NOT advisable** to create or edit tests using other formats in Source View because there is still a lot of work needed to make it stable. Below are the known bugs as of version 1.9.1.

* You will not be able to perform playback nor switch back to Table View unless you revert to HTML.
* The only way to add commands safely on the source code is by recording them.
* When you modify the source code manually, all of it will be lost when you switch to another format.
* Though you can save your test case while in Source View, Selenium IDE will not be able to open it.

**The recommended way to convert Selenese tests is to use the “Export Test Case As…” option under the File menu, and not through the Source View.**.

## Summary

* Test scripts can be created either by recording or typing the commands and parameters manually.
* When creating scripts manually, Firebug is used to get the locator.
* The Find button is used to check that the command is able to access the correct element.
* Table View displays a test script in tabular form while Source View displays it in HTML format.
* Changing the Source View to a non-HTML format is still experimental.
* Do not use the Source View in creating tests in other formats. Use the Export features instead.
* Parameters are not required all the time. It depends upon the command.
* There are three types of commands:
* Actions – directly interacts with page elements
* Accessors – “reads” an element property and stores it in a variable
* Assertions – compares an actual value with an expected one
* Assertions have three types:
* Assert – upon failure, succeeding steps are no longer executed
* Verify – upon failure, succeeding steps are still executed.
* WaitFor – passes if the specified condition becomes true within the timeout period; otherwise, it will fail
* The most common commands are:
* open
* click/clickAndWait
* type/typeKeys
* verifyTitle/assertTitle
* verifyTextPresent
* verifyElementPresent
* verifyTable
* waitForPageToLoad
* waitForElementPresent

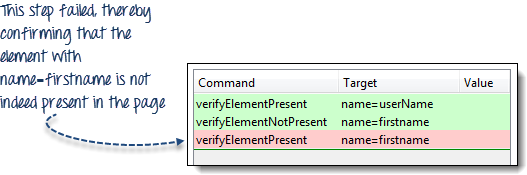
 Locators in Selenium IDE

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Locators tell Selenium IDE which GUI elements ( say Text Box , Buttons, Check Boxes etc) its needs to operate on**.  Identification of correct GUI elements is a prerequisite to  create an automation script.  But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all!  Hence, Selenium provides a number of Locators to precisely locate a GUI element |   The different types of locator are:   * 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 * dom:name * dom:index * XPath     There are commands that do not need a locator (such as the “open” command). However, most of them do need Locators.  **The choice of locator depends largely on your Application Under Test**. In this tutorial we will toggle between facebook , newtours.demoaut on basis of locators that these applications support. Likewise in your testing project you will select any of the above listed locators based on your application support. Locating by ID This is the most common way of locating elements since ID’s are supposed to be unique for each element.  **Target Format:** id=id of the element  For this example, we will use Facebook as our test app because Mercury Tours does not use ID attributes.  **Step 1.** Navigate to [http://www.facebook.com](http://www.facebook.com/). Inspect the “Email or Phone” text box using Firebug and take note of its ID. In this case, the ID is “email”.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator1.png    **Step 2.** Launch Selenium IDE and enter “id=email” in the Target box. Click the Find button and notice that the “Email or Phone” text box becomes highlighted with yellow and bordered with green, meaning, Selenium IDE was able to locate that element correctly.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator2.png Locating by Name Locating elements by name are very similar to locating by ID, except that we use the **“name=”** prefix instead.  **Target Format:** name=name of the element  In the following demonstration, we will now use Mercury Tours because all significant elements have names.  **Step 1.** Navigate to <http://newtours.demoaut.com/> and use Firebug to inspect the “User Name” text box. Take note of its name attribute.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator3.png  Here, we see that the element’s name is “username”.  **Step 2.** In Selenium IDE, enter “name=username” in the Target box and click the Find button. Selenium IDE should be able to locate the User Name text box by highlighting it.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator4.png Locating by Name using Filters Filters can be used when multiple elements have the same name. **Filters are additional attributes used to distinguish elements with the same name.**  **Target Format**: name=name\_of\_the\_element filter=value\_of\_filter  Let’s see an example -  **Step 1**. Log on to Mercury Tours using “tutorial” as the username and password. It should take you to the Flight Finder page shown below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator5.png  **Step 2.** Using Firebug, notice that the Round Trip and One Way radio buttons have the same name “tripType.” However, they have different VALUE attributes so we can use each of them as our filter.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator6.png  **Step 3.**   * We are going to access the One Way radio button first. Click the first line on the Editor. * In the Command box of Selenium IDE, enter the command “click”. * In the Target box, enter “name=tripType value=oneway”.  The “value=oneway” portion is our filter.   http://newguru99.revolutionventur.netdna-cdn.com/images/locator7.png  **Step 4**. Click the Find button and notice that Selenium IDE is able to highlight the One Way radio button with green – meaning that we are able to access the element successfully using its VALUE attribute.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator8.png  **Step 5.** Press the “X” key in your keyboard to execute this click command. Notice that the One Way radio button became selected.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator9.png  You can do the exact same thing with the Round Trip radio button, this time, using “name=tripType value=roundtrip” as your target. Locating by Link Text This type of locator applies only to hyperlink texts. We access the link by prefixing our target with “link=” and then followed by the hyperlink text.  **Target Format**: link=link\_text  In this example, we shall access the “REGISTER” link found in the Mercury Tours homepage.  **Step 1.**   * First, make sure that you are logged off from Mercury Tours. * Go to Mercury Tours homepage.   **Step 2**.   * Using Firebug, inspect the “REGISTER” link. The link text is found between and tags. * In this case, our link text is “REGISTER”. Copy the link text.   http://newguru99.revolutionventur.netdna-cdn.com/images/locator10.png  **Step 3**. Copy the link text in Firebug and paste it onto Selenium IDE’s Target box. Prefix it with “link=”.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator11.png  **Step 4.** Click on the Find button and notice that Selenium IDE was able to highlight the REGISTER link correctly.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator12.png  **Step 5.** To verify further, enter “clickAndWait” in the Command box and execute it. Selenium IDE should be able to click on that REGISTER link successfully and take you to the Registration page shown below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator13.png Locating by CSS Selector **CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes**.**Locating by CSS Selector is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.**  CSS Selectors have many formats, but we will only focus on the most common ones.   * Tag and ID * Tag and class * Tag and attribute * Tag, class, and attribute * Inner text   When using this strategy, we always prefix the Target box with “css=” as will be shown on the following examples. Locating by CSS Selector – Tag and ID Again, we will use Facebook’s Email text box in this example. As you can remember, it has an ID of “email” and we have already accessed it in the “Locating by ID” section. This time, we will use a CSS Selector with ID in accessing that very same element.   |  |  | | --- | --- | | **Syntax** | **Description** | | css=tag#id | * tag = the HTML tag of the element being accessed * # = the hash sign. This should always be present when using a CSS Selector with ID * id = the ID of the element being accessed |   **Keep in mind that the ID is always preceded by a hash sign (#).**  **Step 1.** Navigate to [www.facebook.com](http://www.facebook.com/). Using Firebug, examine the “Email or Phone” text box.  At this point, take note that the HTML tag is “input” and its ID is “email”. So our syntax will be “css=input#email”.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator14.png  **Step 2.** Enter “css=input#email” into the Target box of Selenium IDE and click the Find button. Selenium IDE should be able to highlight that element.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator15.png   Locating by CSS Selector – Tag and Class Locating by CSS Selector using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.   |  |  | | --- | --- | | **Syntax** | **Description** | | css=tag.class | * tag = the HTML tag of the element being accessed * . = the dot sign. This should always be present when using a CSS Selector with class * class = the class of the element being accessed |     **Step 1.** Navigate to [www.facebook.com](http://www.facebook.com/) and use Firebug to inspect the “Email or Phone” text box. Notice that its HTML tag is “input” and its class is “inputtext”.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator16.png  **Step 2.** In Selenium IDE, enter “css=input.inputtext” in the Target box and click Find. Selenium IDE should be able to recognize the Email or Phone text box.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator17.png  **Take note that when multiple elements have the same HTML tag and name, only the first element in source code will be recognized**. Using Firebug, inspect the Password text box in Facebook and notice that it has the same name as the Email or Phone text box.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator18.png  The reason why only the Email or Phone text box was highlighted in the previous illustration is that it comes first in Facebook’s page source.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator19.png Locating by CSS Selector – Tag and Attribute This strategy uses the HTML tag and a specific attribute of the element to be accessed.   |  |  | | --- | --- | | Syntax | Description | | css=tag[attribute=value] | * tag = the HTML tag of the element being accessed * [ and ] = square brackets within which a specific attribute and its corresponding value will be placed * attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. * value = the corresponding value of the chosen attribute. |     **Step 1.** Navigate to Mercury Tours’ Registration page (<http://newtours.demoaut.com/mercuryregister.php>) and inspect the “Last Name” text box. Take note of its HTML tag (“input” in this case) and its name (“lastName”).  http://newguru99.revolutionventur.netdna-cdn.com/images/locator20.png  **Step 2.** In Selenium IDE, enter “css=input[name=lastName]” in the Target box and click Find. Selenium IDE should be able to access the Last Name box successfully.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator21.png  **When multiple elements have the same HTML tag and attribute, only the first one will be recognized**. This behavior is similar to locating elements using CSS selectors with the same tag and class. Locating by CSS Selector - tag, class, and attribute    |  |  | | --- | --- | | Syntax | Description | | css=tag.class[attribute=value] | * tag = the HTML tag of the element being accessed * . = the dot sign. This should always be present when using a CSS Selector with class * class = the class of the element being accessed * [ and ] = square brackets within which a specific attribute and its corresponding value will be placed * attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. * value = the corresponding value of the chosen attribute. |     **Step 1.** Navigate to [www.facebook.com](http://www.facebook.com/) and use Firebug to inspect the ‘Email or Phone’ and ‘Password’ input boxes. Take note of their HTML tag, class, and attributes. For this example, we will select their ‘tabindex’ attributes.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator22.png  **Step 2.**  We will access the ‘Email or Phone’ text box first, thus, we will use a tabindex value of 1. Enter “css=input.inputtext[tabindex=1]” in Selenium IDE’s Target box and click Find. The ‘Email or Phone’ input box should be highlighted.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator23.png  **Step 3.** To access the Password input box, simply replace the value of the tabindex attribute. Enter “css=input.inputtext[tabindex=2]” in the Target box and click on the Find button. Selenium IDE must be able to identify the Password text box successfully.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator24.png Locating by CSS Selector – inner text As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**   |  |  | | --- | --- | | **Syntax** | **Description** | | css=tag:contains(“inner text”) | * tag = the HTML tag of the element being accessed * inner text = the inner text of the element |     **Step 1.** Navigate to Mercury Tours’ homepage (<http://newtours.demoaut.com/>) and use Firebug to investigate the “Password” label. Take note of its HTML tag (which is “font” in this case) and notice that it has no class, id, or name attributes.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator25.png  **Step 2.** Type **css=font:contains("Password:")** into Selenium IDE’s Target box and click Find. Selenium IDE should be able to access the Password label as shown on the image below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator26.png  **Step 3.** This time, replace the inner text with “Boston” so that your Target will now become “css=font:contains("Boston")”. Click Find. You should notice that the “Boston to San Francisco” label becomes highlighted. This shows you that Selenium IDE can access a long label even if you just indicated the first word of its inner text.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator27.png Locating by DOM (Document Object Model) The Document Object Model (DOM), in simple terms, is the way by which HTML elements are structured. Selenium IDE is able to use the DOM in accessing page elements. If we use this method, our Target box will always start with “dom=document...”; however, the “dom=” prefix is normally removed because Selenium IDE is able to automatically interpret anything that starts with the keyword “document” to be a path within the DOM anyway.  There are four basic ways to locate an element through DOM:   * getElementById * getElementsByName * dom:name (applies only to elements within a named form) * dom:index   Locating by DOM – getElementById  Let us focus on the first method – using the getElementById method. The syntax would be:   |  |  | | --- | --- | | **Syntax** | **Description** | | document.getElementById(“id of the element”) | 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 (“”). |     **Step 1.** Navigate to [www.facebook.com](http://www.facebook.com/) and use Firebug to inspect the “Keep me logged in” check box. Take note of its ID.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator28.png  We can see that the ID we should use is “persist\_box”.  **Step 2.** Open Selenium IDE and in the Target box, enter “document.getElementById(“persist\_box”)” and click Find. Selenium IDE should be able to locate the “Keep me logged in” check box. Though it cannot highlight the interior of the check box, Selenium IDE can still surround the element with a bright green border as shown below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator29.png Locating by DOM – getElementsByName The getElementById method can access only one element at a time, and that is the element with the ID that you specified. The getElementsByName method is different. It collects an array of elements that have the name that you specified. You access the individual elements using an index which starts at 0.   |  |  |  | | --- | --- | --- | | http://newguru99.revolutionventur.netdna-cdn.com/images/locator30.png | **getElementById**   * It will get only one element for you. * That element bears the ID that you specified inside the parentheses of getElementById(). | | | http://newguru99.revolutionventur.netdna-cdn.com/images/locator31.jpg | | **getElementsByName**   * It will get a collection of elements whose names are all the same. * Each element is indexed with a number starting from 0 just like an array * You specify which element you wish to access by putting its index number into the square brackets in getElementsByName’s syntax below. |      |  |  | | --- | --- | | **Syntax** | **Description** | | document.getElementsByName(“name”)[index] | * name = name of the element as defined by its ‘name’ attribute * index = an integer that indicates which element within getElementsByName’s array will be used. |     **Step 1.** Navigate to Mercury Tours’ Homepage and login using “tutorial” as the username and password. Firefox should take you to the Flight Finder screen.  **Step 2.** Using Firebug, inspect the three radio buttons at the bottom portion of the page (Economy class, Business class, and First class radio buttons). Notice that they all have the same name which is “servClass”.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator32.png  **Step 3.** Let us access the “Economy class” radio button first. Of all these three radio buttons, this element comes first so it has an index of 0. In Selenium IDE, type “document.getElementsByName(“servClass”)[0]” and click the Find button. Selenium IDE should be able to identify the Economy class radio button correctly.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator33.png  **Step 4.** Change the index number to 1 so that your Target will now become document.getElementsByName(“servClass”)[1]. Click the Find button and Selenium IDE should be able to highlight the “Business class” radio button, as shown below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator34.png Locating by DOM – dom:name As mentioned earlier, this method will only apply if the element you are accessing is contained within a named form.   |  |  | | --- | --- | | **Syntax** | **Description** | | document.forms[“name of the form”].elements[“name of the element”] | * name of the form = the value of the name attribute of the form tag that contains the element you want to access * name of the element = the value of the name attribute of the element you wish to access |     **Step 1.** Navigate to Mercury Tours homepage (<http://newtours.demoaut.com/>) and use Firebug to inspect the User Name text box. Notice that it is contained in a form named “home”.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator35.png  **Step 2.** In Selenium IDE, type “document.forms[“home”].elements[“userName”]” and click the Find button. Selenium IDE must be able to access the element successfully.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator36.png Locating by DOM – dom:index This method applies even when the element is not within a named form because it uses the form’s index and not its name.   |  |  | | --- | --- | | **Syntax** | **Description** | | document.forms[index of the form].elements[index of the element] | * index of the form = the index number (starting at 0) of the form with respect to the whole page * index of the element = the index number (starting at 0) of the element with respect to the whole form that contains it |     We shall access the “Phone” text box within Mercury Tours Registration page. The form in that page has no name and ID attribute so this will make a good example.  **Step 1.** Navigate to Mercury Tours Registration page and inspect the Phone text box. Notice that the form containing it has no ID and name attributes.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator37.png  **Step 2.** Enter “document.forms[0].elements[3]” in Selenium IDE’s Target box and click the Find button. Selenium IDE should be able to access the Phone text box correctly.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator38.png  **Step 3.** Alternatively, you can use the element’s name instead of its index and obtain the same result. Enter “document.forms[0].elements["phone"]” in Selenium IDE’s Target box. The Phone text box should still become highlighted.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator39.png Locating by XPath XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use XPath in locating HTML elements.  **Advantage:** It can access almost any element, even those without class, name, or id attributes.  **Disadvantage:** It is the most complicated method of identifying elements because of too many different rules and considerations.  Fortunately, Firebug can automatically generate XPath locators. In the following example, we will access an image that cannot possibly be accessed through the methods we discussed earlier.  **Step 1.** Navigate to Mercury Tours Homepage and use Firebug to inspect the orange rectangle to the right of the yellow “Links” box. Refer to the image below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator40.png  **Step 2**. Right click on the element’s HTML code and then select the “Copy XPath” option.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator41.png  **Step 3.** In Selenium IDE, type one forward slash “/” in the Target box then paste the XPath that we copied in the previous step. **The entry in your Target box should now begin with two forward slashes “//”.**  http://newguru99.revolutionventur.netdna-cdn.com/images/locator42.png  **Step 4**. Click on the Find button. Selenium IDE should be able to highlight the orange box as shown below.  http://newguru99.revolutionventur.netdna-cdn.com/images/locator43.png Summary Syntax for Locator Usage   |  |  |  | | --- | --- | --- | | **Method** | **Target Syntax** | **Example** | | By ID | id= *id\_of\_the\_element* | id=email | | By Name | name=*name\_of\_the\_element* | name=username | | By Name Using Filters | name=*name\_of\_the\_element* *filter*=*value\_of\_filter* | name=tripType value=oneway | | By Link Text | link=*link\_text* | link=REGISTER | | Tag and ID | css=*tag*#*id* | css=input#email | | Tag and Class | css=*tag*.*class* | css=input.inputtext | | Tag and Attribute | css=*tag*[*attribute*=*value*] | css=input[name=lastName] | | Tag, Class, and Attribute | css=*tag*.*class*[*attribute*=*value*] | css=input.inputtext[tabindex=1] | |

|  |
| --- |
|  |

**Enhancing Selenium IDE Script**

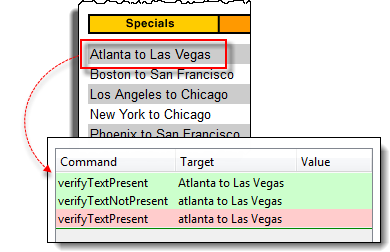
|  |
| --- |
| In this tutorial, we look at commands that will make your automation script more intelligent and complete. Verify Presence of an Element We can use following two commands to verify the presence of an element:   * **verifyElementPresent** – returns TRUE if the specified element was FOUND in the page; FALSE if otherwise * **verifyElementNotPresent** – returns TRUE if the specified element was NOT FOUND anywhere in the page; FALSE if it is present.   The test script below verifies that the User Name text box is present within the Mercury Tours homepage while the First Name text box is not. The First Name text box is actually an element present in the Registration page of Mercury Tours, not in the homepage. strong>Verify Presence of a Certain Text |



### Verify Presenc of a Certain Text

* **verifyTextPresent** – returns TRUE if the specified text string was FOUND somewhere in the page; FALSE if otherwise
* **verifyTextNotPresent** – returns TRUE if the specified text string was NOT FOUND anywhere in the page; FALSE if it was found

Remember that these commands are case-sensitive.

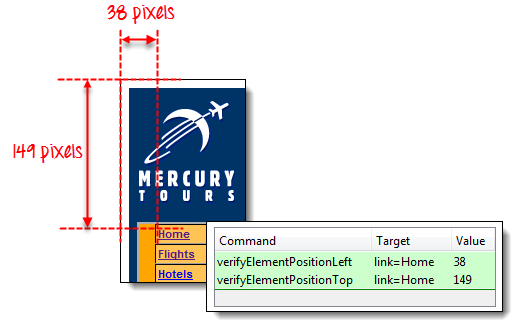


In the scenario above, "Atlanta to Las Vegas" was treated differently from "atlanta to Las Vegas" because the letter "A"of "Atlanta" was in uppercase on the first one while lowercase on the other. When the verifyTextPresent command was used on each of them, the other one passed and the other failed.

### Verify Specific Position of an Element

Selenium IDE indicates the position of an element by measuring (in pixels) how far it is from the left or top edge of the browser window.

* **verifyElementPositionLeft** – verifies if the specified number of pixels match the distance of the element from the left edge of the page. This will return FALSE if the value specified does not match the distance from the left edge.
* **verifyElementPositionTop** – verifies if the specified number of pixels match the distance of the element from the top edge of the page. This will return FALSE if the value specified does not match the distance from the top edge.



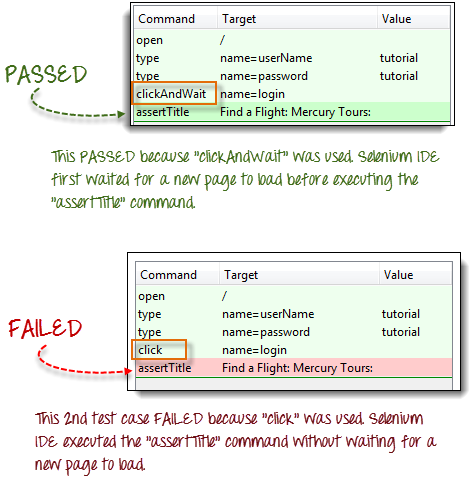
## Wait commands

### andWait commands

These are commands that will wait for a new page to load before moving onto the next command.

Examples are

* clickAndWait
* typeAndWait
* selectAndWait



### waitFor commands

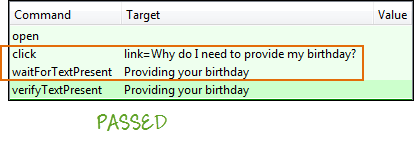
These are commands that wait for a specified condition to become true before proceeding to the next command (irrespective of loading of a new page). These commands are more appropriate to be used on AJAX-based dynamic websites that change values and elements without reloading the whole page. Examples include:

* waitForTitle
* waitForTextPresent
* waitForAlert

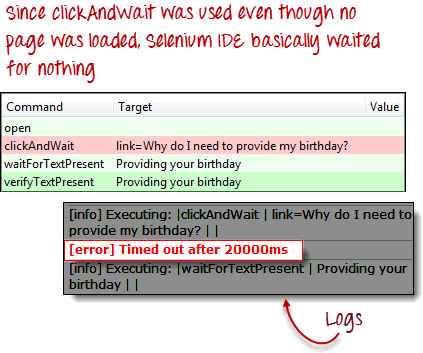
Consider the Facebook scenario below.



We can use a combination of "click" and "waitForTextPresent" to verify the presence of the text "Providing your birthday".



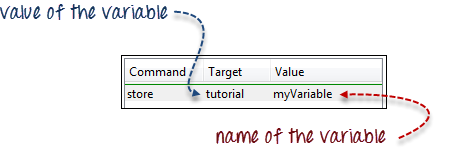
We cannot use clickAndWait because no page was loaded upon clicking on the "Why do I need to provide my birthday?" link.If we do, the test will fail



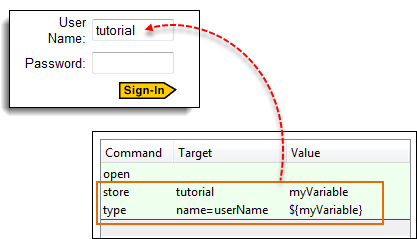
## Storing Variables and the Echo command

### Store

To store variables in Selenium IDE, we use the "store" command. The illustration below stores the value "tutorial" to a variable named "myVariable".



To access the variable, simply enclose it in a ${ … } symbol. For example, to enter the value of "myVariable" onto the "userName" textbox of Mercury Tours, enter ${myVariable} in the Value field.



### StoreElementPresent

This command stores either "true" or "false" depending on the presence of the specified element. The script below stores the Boolean value "true" to "var1" and "false" to "var2". To verify, we will use the "echo" command to display the values of var1 and var2. The Base URL for the illustration below was set to Mercury Tours homepage.

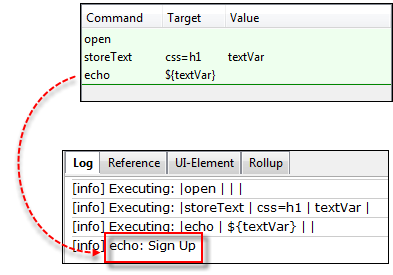


### StoreText

This command is used to store the inner text of an element onto a variable. The illustration below stores the inner text of the tag in Facebook onto a variable named 'textVar'.



Since it is the only element in the page, it is safe to use 'css=h1' as our target. The image below shows that Selenium IDE was able to save the string "Sign Up" in the 'textVar' variable by printing its value correctly.



## Alerts, Pop-up and Multiple Windows

Alerts Alerts are probably the simplest form of pop-up windows. The most common Selenium IDE commands used in handling alerts are the following:

|  |  |
| --- | --- |
| assertAlert  assertNotAlert | retrieves the message of the alert and asserts it to a string value that you specified |
| assertAlertPresent  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  verifyNotAlert | retrieves the message of the alert and verifies if it is equal to the string value that you specified |
| verifyAlertPresent  verifyAlertNotPresent | verifies if an Alert is present or not |

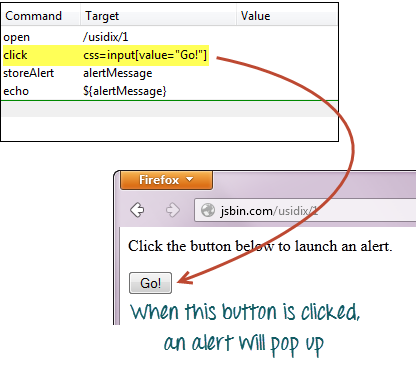
  Remember these two things when working with alerts:

* Selenium IDE will automatically click on the OK button of the alert window and so you will not be able to see the actual alert.
* Selenium IDE will not be able to handle alerts that are within the page's onload() function. It will only be able to handle alerts that are generated after the page has completely loaded.

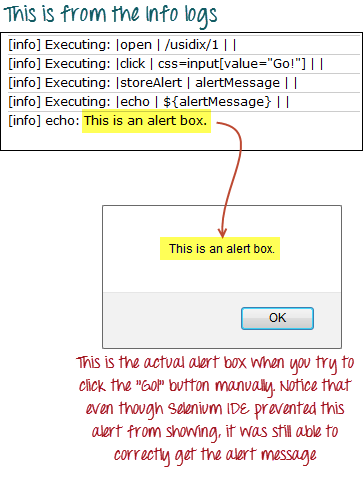
In this example, we will use the storeAlert command to show that even though Selenium IDE did not show the actual alert, it was still able to retrieve its message.

**Step 1.** In Selenium IDE, set the Base URL to http://jsbin.com.

**Step 2.** Create the script as shown below.



**Step 3.** Execute the script and do not expect that you will be able to see the actual alert.

****

## Confirmations

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

  However, these are the additional commands that you need to use to instruct Selenium which option to choose, whether the OK or the CANCEL button.

* chooseOkOnNextConfirmation/chooseOkOnNextConfirmationAndWait
* chooseCancelOnNextConfirmation

**You should use these commands before a command that triggers the confirmation box so that Selenium IDE will know beforehand which option to choose.** Again, you will not be able to see the actual confirmation box during script execution.

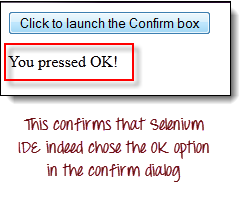
Let us test a webpage that has a button that was coded to show whether the user had pressed the OK or the CANCEL button.

**Step 1.** In Selenium IDE, set the Base URL to http://jsbin.com

**Step 2.** Create the script as shown below. This time, we will press the OK button first.



**Step 3.** Execute the script and notice that you do not see the actual confirmation, but the webpage was able to indicate which button Selenium IDE had pressed.



**Step 4.** Replace the "chooseOkOnNextConfirmation" command with "chooseCancelOnNextConfirmation" and execute the script again.



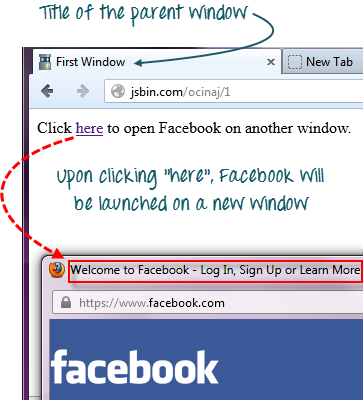
## Multiple Windows

If you happen to click on a link that launches a separate window, **you must first instruct Selenium IDE to select that window first before you could access the elements within it**. To do this, you will **use the window's title as its locator**.

**We use the selectWindow command in switching between windows.**

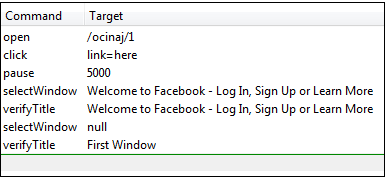
We will use a link http://jsbin.com/ocinaj/1 whose title is "First Window". The "here" hyperlink found in that page will open Facebook on a new window, after which we shall instruct Selenium IDE to do the following:

* Transfer control from the parent window to the newly launched Facebook window using the "selectWindow" command and its title as the locator
* Verify the title of the new window
* Select back the original window using the "selectWindow" command and "null" as its target.
* Verify the title of the currently selected window



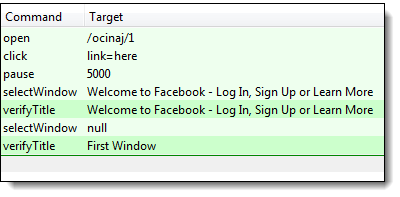
**Step 1.** Set the Base URL to http://jsbin.com.

**Step 2.** Create the script as shown below.



We need the "pause" command to wait for the newly launched window to load before we could access its title.

**Step 3.** Execute the script. Notice that the test case passed, meaning that we were able to switch between windows and verify their titles successfully.



Always remember that setting selectWindow's target to "null" will automatically select the parent window (in this case, the window where the element "link=here" is found)

## Summary

* The three most commonly used commands in verifying page elements are:
  + verifyElementPresent/ verifyElementNotPresent
  + verifyTextPresent/ verifyTextNotPresent
  + verifyElementPositionLeft/ verifyElementPositionTop
* Wait commands are classified into two:
  + andWait commands – used when a page is expected to be loaded
  + waitFor commands – used when no new page is expected to be loaded
* The "store" command (and all its variants) are used to store variables in Selenium IDE
* The "echo" command is used to print a string value or a variable
* Variables are enclosed within a ${…} when being printed or used on elements
* Selenium IDE automatically presses the OK button when handling alerts
* When handling confirmation dialogs, you may instruct Selenium IDE which option to use:
  + chooseOkOnNextConfirmation/chooseOkOnNextConfirmationAndWait
  + chooseCancelOnNextConfirmation
* Window titles are used as locators when switching between browser windows.
* When using the "selectWindow" command, setting the Target to "null" will automatically direct Selenium IDE to select the parent window.

**Intro WebDriver Comparision with Selenium RC**

|  |
| --- |
| Now that you have learned to create simple tests in Selenium IDE, we shall now create more powerful scripts using an advanced tool called **WebDriver**.   What is WebDriver? WebDriver is a web automation framework that allows you to **execute your tests against different browsers**, not just Firefox (unlike Selenium IDE). |



Web Driver 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.

Following programming languages are supported by WebDriver

* Java
* .Net
* PHP
* Python
* Perl
* Ruby

**You do not have to know all of them. You just need to be knowledgeable in one.**However, in this tutorial, we will be using Java with Eclipse as our IDE.

## WebDriver Vs Selenium RC

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

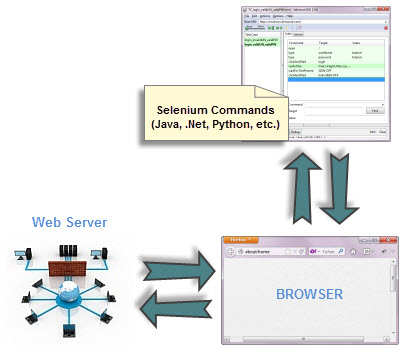
* They both allow you to **use a programming language** in designing your test scripts.
* They both allow you to **run your tests against different browsers.**

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

### Architecture

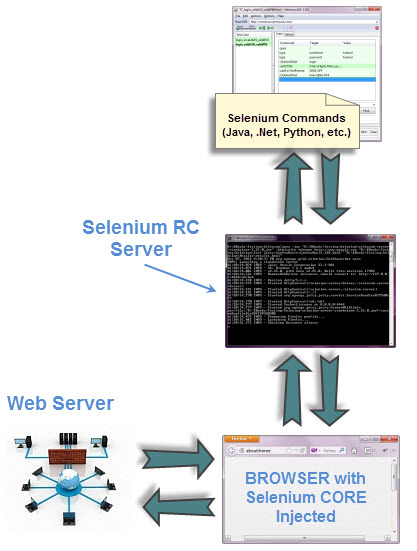
**WebDriver’s architecture is simpler than Selenium RC’s**.

* 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.



### Speed



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

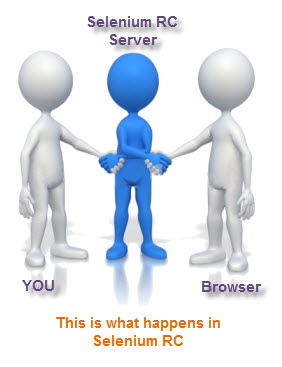


**Selenium RC is slower  sinceit uses a Javascript program called Selenium Core.**This Selenium Core is the one that directly controls the browser, not you.

### Real-life Interaction

##### http://newguru99.revolutionventur.netdna-cdn.com/images/2-way_handshake.jpg

**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

### API



**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.

## Browser Support

##### http://newguru99.revolutionventur.netdna-cdn.com/images/htmlunit.jpg

**WebDriver can support the headless HtmlUnit browser**.

HtmlUnit is termed as “headless” because it is an invisible browser – it is GUI-less.

It is a very fast browser because no time is spent in waiting for page elements to load. This accelerates your test execution cycles.

Since it is invisible to the user, it can only be controlled through automated means.

**Selenium RC cannot support the headless HtmlUnit browser.** It needs a real, visible browser to operate on

## Limitations of WebDriver

### ****WebDriver Cannot Readily Support New Browsers****

Remember that WebDriver operates on the OS level. Also remember that different browsers communicate with the OS in different ways. If a new browser comes out, it may have a different process of communicating with the OS as compared to other browsers. So, **you have to give the WebDriver team quite some time to figure that new process out** before they can implement it on the next WebDriver release.

However, it is up to the WebDriver’s team of developers to decide if they should support the new browser or not.

##### 

### Selenium RC Has Built-In Test Result Generator



**Selenium RC automatically generates an HTML file of test results**. The format of the report was pre-set by RC itself. Take a look at an example of this report below.

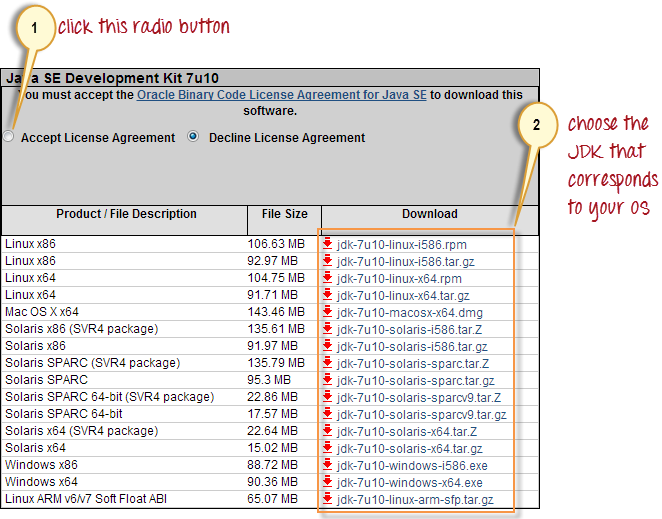
**WebDriver has no built-in command that automatically generates a Test Results File**. You would have to rely on your IDE’s output window, or design the report yourself using the capabilities of your programming language and store it as text, html, etc.

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/Bean_ok_summary.jpg | Summary  * WebDriver is a tool for testing web applications **across different browsers** using different programming languages. * You are now able to make powerful tests because WebDriver **allows you to use a programming language** of your choice in designing your tests. * WebDriver is **faster than Selenium RC** because of its simpler architecture. * WebDriver **directly talks to the browser** while Selenium RC needs the help of the RC Server in order to do so. * WebDriver’s APIis**more concise** than Selenium RC’s. * WebDriver **can support HtmlUnit** while Selenium RC cannot. * The only drawbacks of WebDriver are: * It **cannot readily support new browsers**, but Selenium RC can. * It **does not have a built-in command** for automatic generation of test results. |

|  |
| --- |
| **Install Webdriver**  In this tutorial we will install Webdriver (Java only) and Configure Eclipse   Step 1 – Install Java on your computer Download and install the **Java Software Development Kit (JDK)** [here](http://www.oracle.com/technetwork/java/javase/downloads/index.html). |



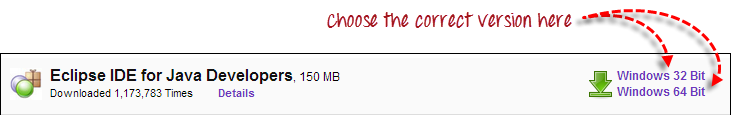
Next -



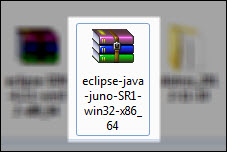
This JDK version comes bundled with Java Runtime Environment (JRE) so you do not need to download and install the JRE separately.

## Step 2 – Install Eclipse IDE

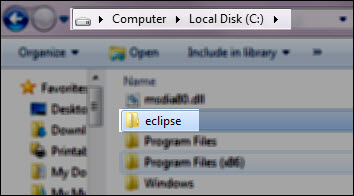
Download “**Eclipse IDE for Java Developers**” [here](http://www.eclipse.org/downloads/). Be sure to choose correctly between Windows 32 Bit and 64 Bit versions.



You should be able to download a ZIP file named “eclipse-java-juno-SR1-win32-x86\_64.zip” (the version number “SR1” may change over time).



Inside that ZIP file, there is an “eclipse” folder which contains all the application files. You can extract the “eclipse” folder anywhere you want in your PC; but for this tutorial, extract it to your C drive.



Unlike  other popular software , no installation is required to use eclipse.

## Step 3 – Download the Selenium Java Client Driver

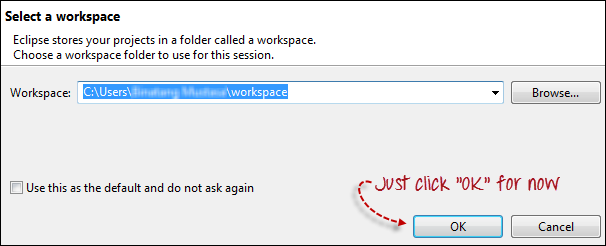
You can download the **Selenium Java Client Driver** [here](http://seleniumhq.org/download/). You will find client drivers for other languages there, but only choose the one for Java.



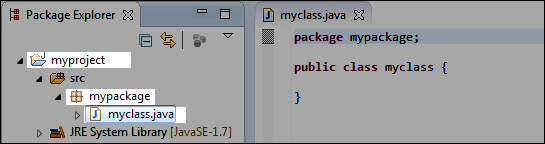
This download comes as a ZIP file named “selenium-2.25.0.zip”. For simplicity, extract the contents of this ZIP file on your C drive so that you would have the directory “C:\selenium-2.25.0\”. This directory contains all the JAR files that we would later import on Eclipse.

## Step 4 – Configure Eclipse IDE with WebDriver

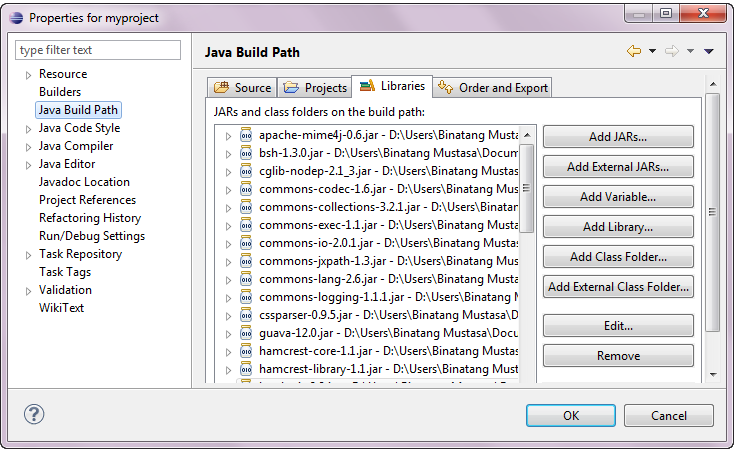
1. Launch the “eclipse.exe” file inside the “eclipse” folder that we extracted in step 2. If you followed step 2 correctly, the executable should be located on C:\eclipse\eclipse.exe.
2. When asked to select for a workspace, just accept the default location.



1. Create a new project through File > New > Java Project. Name the project as “myproject”.
2. Right-click on the newly created project and select New > Package, and name that package as “mypackage”.
3. Create a new Java class under mypackage by right-clicking on it and then selecting New > Class, and then name it as “myclass”. Your Eclipse IDE should look like the image below.



1. Right-click on myproject and select **Properties**.
2. On the Properties dialog, click on “Java Build Path”.
3. Click on the **Libraries** tab, and then click “Add External JARs..”
4. Navigate to C:\selenium-2.25.0\ (or any other location where you saved the extracted contents of “selenium-2.25.0.zip” in step 3).
5. Add all the JAR files inside and outside the “libs” folder. Your Properties dialog should now look similar to the image below.



1. Finally, click OK and we are done importing Selenium libraries into our project.

## Different Drivers

**HTMLUnit and Firefox are two browsers that WebDriver can directly automate** – meaning that no other separate component is needed to install or run while the test is being executed. For other browsers, a separate program is needed. That program is called as the **Driver Server**.

A driver server is different for each browser. For example, Internet Explorer has its own driver server which you cannot use on other browsers. Below is the list of driver servers and the corresponding browsers that use them.

|  |  |  |
| --- | --- | --- |
| **Browser** | **Name of Driver Server** | **Remarks** |
| HTMLUnit | (none) | WebDriver can drive HTMLUnit without the need of a driver server |
| Firefox | (none) | WebDriver can drive Firefox without the need of a driver server |
| Internet Explorer | Internet Explorer Driver Server | Available in 32 and 64-bit versions. Use the version that corresponds to the architecture of your IE |
| Chrome | ChromeDriver | Though its name is just “ChromeDriver”, it is in fact a Driver Server, not just a driver. The current version can support versions higher than Chrome v.21 |
| Opera | OperaDriver | Though its name is just “OperaDriver”, it is in fact a Driver Server, not just a driver. |
| PhantomJS | GhostDriver | PhantomJS is another headless browser just like HTMLUnit. |
| Safari | SafariDriver | Though its name is just “SafariDriver”, it is in fact a Driver Server, not just a driver. |

You can download these driver servers [here](http://seleniumhq.org/download/)

## Summary

* Aside from a browser, you will need the following to start using WebDriver
* **Java Development Kit (JDK).** <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
* **Eclipse IDE** - <http://www.eclipse.org/downloads/>
* **Java Client Driver** - <http://seleniumhq.org/download/>
* When starting a WebDriver project in Eclipse, do not forget to import the Java Client Driver files onto your project. These files will constitute your Selenium Library.
* HTMLUnit and Firefox are the only browsers that you can automate without the use of a Driver Server.
* Each other browser has its own driver server.

|  |
| --- |
| **First webDriver Script**  Using the Java class “myclass”  that we created in the previous tutorial, let us try to create a WebDriver script that would:     1. fetch Mercury Tours’ homepage 2. verify its title 3. print out the result of the comparison 4. close it before ending the entire program. |

### WebDriver Code

Below is the actual WebDriver code for the logic presented by the scenario above

01.package mypackage;

02.

03.

04.

05.import org.openqa.selenium.WebDriver;

06.

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

08.

09.

10.

11.public class myclass {

12.

13.

14.

15.public static void main(String[] args) {

16.

17.// declaration and instantiation of objects/variables

18.

19.WebDriver driver = new FirefoxDriver();

20.

21.String baseUrl = "<http://newtours.demoaut.com>";

22.

23.String expectedTitle = "Welcome: Mercury Tours";

24.

25.String actualTitle = "";

26.

27.

28.

29.// launch Firefox and direct it to the Base URL

30.

31.driver.get(baseUrl);

32.

33.

34.

35.// get the actual value of the title

36.

37.actualTitle = driver.getTitle();

38.

39.

40.

41./\*

42.

43.\* compare the actual title of the page witht the expected one and print

44.

45.\* the result as "Passed" or "Failed"

46.

47.\*/

48.

49.if (actualTitle.contentEquals(expectedTitle)){

50.

51.System.out.println("Test Passed!");

52.

53.} else {

54.

55.System.out.println("Test Failed");

56.

57.}

58.

59.

60.

61.//close Firefox

62.

63.driver.close();

64.

65.

66.

67.// exit the program explicitly

68.

69.System.exit(0);

70.

71.}

72.

73.

74.

75.}

## Explaining the code

### Importing Packages

To get started, you need to import following two packages:

1. **org.openqa.selenium.\***- contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver** – contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class

If your test needs more complicated actions such as accessing another class, taking browser screenshots, or manipulating external files, definitely you will need to import more packages.

### Instantiating objects and variables

Normally, this is how a driver object is instantiated.

http://www.guru99.com/images/image004%282%29.png

A FirefoxDriver class with no parameters means that the default Firefox profile will be launched by our Java program. The default Firefox profile is similar to launching Firefox in safe mode (no extensions are loaded).

For convenience, we saved the Base URL and the expected title as variables.

### Launching a Browser Session

WebDriver’s **get()** method is used to launch a new browser session and directs it to the URL that you specify as its parameter.

http://www.guru99.com/images/image005%282%29.png

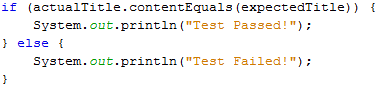
### Get the Actual Page Title

The WebDriver class has the **getTitle()** method that is always used to obtain the page title of the currently loaded page.

http://www.guru99.com/images/image006%282%29.png

### Compare the Expected and Actual Values

This portion of the code simply uses a basic Java if-else structure to compare the actual title with the expected one.



### Terminating a Browser Session

The “**close()**” method is used to close the browser window.

http://www.guru99.com/images/image008%283%29.png

### Terminating the Entire Program

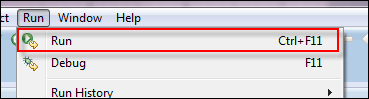
If you use this command without closing all browser windows first, your whole Java program will end, while leaving browser window open.

http://www.guru99.com/images/image009%283%29.png

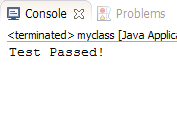
### Running the Test

There are two ways to execute code in Eclipse IDE.

1. On Eclipse’s menu bar, click **Run > Run.**
2. Press **Ctrl+F11** to run the entire code.



 If you did everything correctly, Eclipse would output “Test Passed!”



## Locating GUI Elements

Locating elements in web Driver is done by using the “**findElement(By.locator())**” method. The “locator” part of the code is same as any of the locators previously discussed in the Selenium IDE chapters of this tutoriasl.Infact, it is recommended you locate GUI elements using IDE and once successfully identified export the code to web driver.

Here is a sample code that locates an element by its id. Facebook is used as the Base URL.

01.package mypackage;

02.

03.import org.openqa.selenium.By;

04.

05.import org.openqa.selenium.WebDriver;

06.

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

08.

09.public class myclass {

10.

11.

12.

13.public static void main(String[] args) {

14.

15.WebDriver driver = new FirefoxDriver();

16.

17.String baseUrl = "<http://www.facebook.com>";

18.

19.String tagName = "";

20.

21.

22.

23.driver.get(baseUrl);

24.

25.tagName = driver.findElement(By.id("email")).getTagName();

26.

27.System.out.println(tagName);

28.

29.driver.close();

30.

31.System.exit(0);

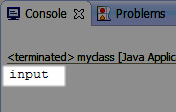
32.

33.}

34.

35.}

We used the **getTagName()** method to extract the tag name of that particular element whose id is “email”. When run, this code should be able to correctly identify the tag name “input” and will print it out on Eclipse’s Console window.

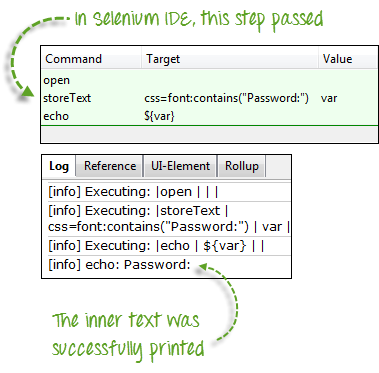


Summary for locating elements

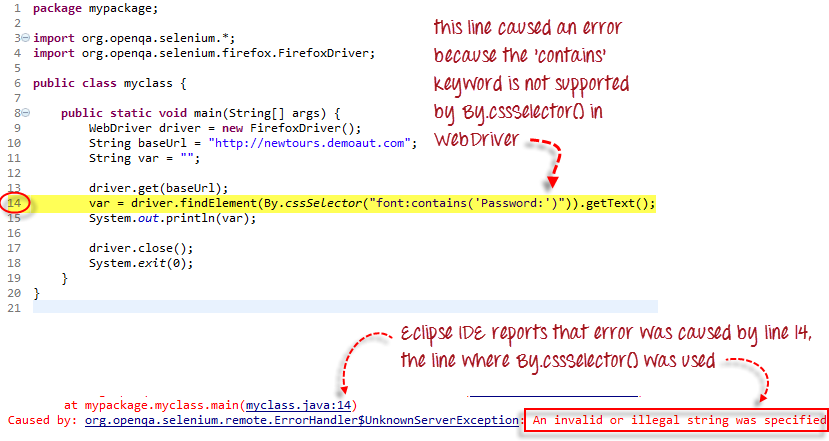
|  |  |  |
| --- | --- | --- |
| **Variation** | **Description** | **Sample** |
| By.**className** | finds elements based on the value of the “class” attribute | findElement(By.className(“someClassName”)) |
| By.**cssSelector** | finds elements based on the driver’s underlying CSS Selector engine | findElement(By.cssSelector(“input#email”)) |
| By.**id** | locates elements by the value of their “id” attribute | findElement(By.id(“someId”)) |
| By.**linkText** | finds a link element by the exact text it displays | findElement(By.linkText(“REGISTRATION”)) |
| By.**name** | locates elements by the value of the “name” attribute | findElement(By.name(“someName”)) |
| By.**partialLinkText** | locates elements that contain the given link text | findElement(By.partialLinkText(“REG”)) |
| By.**tagName** | locates elements by their tag name | 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]”)) |

### Note on Using findElement(By.cssSelector())

**By.cssSelector() does not support the “contains” feature**. Consider the Selenium IDE code below -



In Selenium IDE above, the entire test passed. However in the WebDriver script below, the same test generated an error because WebDriver does not support the “contains” keyword when used in the By.cssSelector() method.



## Common Commands

### Instantiating Web Elements

Instead of using the long “driver.findElement(By.locator())” syntax every time you will access a particular element, we can instantiate a WebElement object for it. The WebElement class is contained in the “org.openqa.selenium.\*” package.

http://www.guru99.com/images/image016%282%29.png

### Clicking on an Element

Clicking is perhaps the most common way of interacting with web elements**. The click() method is used to simulate the clicking of any element.**  The following example shows how click() was used to click on Mercury Tours’  “Sign-In” button.

http://www.guru99.com/images/image017%282%29.png

Following things must be noted when using the click() method.

* **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** (height and width must not be equal to zero).

### Get Commands

Get commands fetch various important information about the page/element. Here are some important “get” commands you must be familiar with.

|  |  |
| --- | --- |
| **get()**  Sample usage: | * It automatically opens a new browser window and fetches the page that you specify inside its parentheses. * It is the counterpart of Selenium IDE’s “open” command. * The parameter must be a **String** object. |
| **getTitle()**  Sample usage: | * Needs no parameters * Fetches the title of the current page * Leading and trailing white spaces are trimmed * Returns a null string if the page has no title |
| **getPageSource()**  Sample usage: | * Needs no parameters * Returns the **source code of the page** as a String value |
| **getCurrentUrl()**  Sample usage: | * Needs no parameters * Fetches the string representing the **current URL** that the browser is looking at |
| **getText()**  Sample usage: | * Fetches the **inner text** of the element that you specify |

### Navigate commands

These commands allow you to  refresh,go-into and switch back and forth between different web pages.

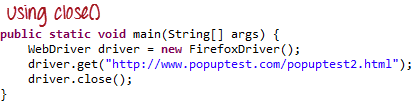
|  |  |
| --- | --- |
| **navigate().to()**  Sample usage: | * It automatically **opens a new browser window and fetches the page** that you specify inside its parentheses. * **It does exactly the same thing as the get() method.** |
| **navigate().refresh()**  Sample usage: | * Needs no parameters. * It **refreshes** the current page. |
| **navigate().back()**  Sample usage: | * Needs no parameters * Takes you **back by one page** on the browser’s history. |
| **navigate().forward()**  Sample usage: | * Needs no parameters * Takes you **forward by one page** on the browser’s history. |

### Closing and Quitting Browser Windows

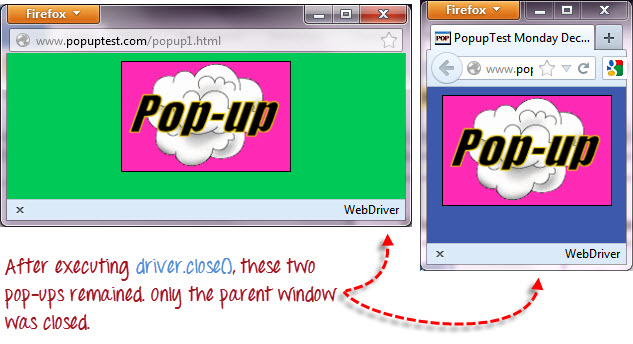
|  |  |
| --- | --- |
| **close()**  Sample usage: | * Needs no parameters * **It closes only the browser window that WebDriver is currently controlling**. |
| **quit()**  Sample usage: | * Needs no parameters * **It closes all windows that WebDriver has opened.** |



To clearly illustrate the difference between close() and quit(), try to execute the code below. It uses a webpage that automatically pops up a window upon page load and opens up another after exiting.



Notice that only the parent browser window was closed and not the two pop-up windows.



But if you use quit(), all windows will be closed – not just the parent one. Try running the code below and you will notice that the two pop-ups above will automatically be closed as well.

01.package mypackage;

02.

03.

04.

05.import org.openqa.selenium.WebDriver;

06.

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

08.

09.

10.

11.public class myclass {

12.

13.

14.

15.public static void main(String[] args) {

16.

17.WebDriver driver = new FirefoxDriver();

18.

19.

20.

21.driver.get("<http://www.popuptest.com/popuptest2.html>");

22.

23.driver.quit();  // using QUIT all windows will close

24.

25.

26.

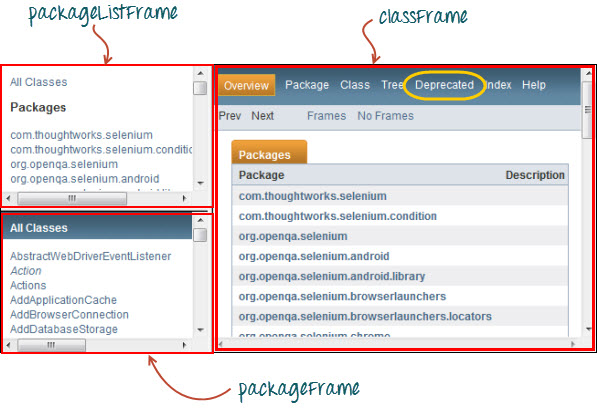
27.}

28.

29.}

## Switching Between Frames

To access GUI elements in a Frame, we should first direct WebDriver to focus on the frame or pop-up window first before we can access elements within them. Let us take, for example, the web page <http://selenium.googlecode.com/svn/trunk/docs/api/java/index.html>



This page has 3 frames whose “name” attributes are indicated above. We wish to access the “Deprecated” link encircled above in yellow. In order to do that, we must first instruct WebDriver to switch to the “classFrame” frame using the **“switchTo().frame()”** method. We will use the name attribute of the frame as the parameter for the “frame()” part.

01.package mypackage;

02.

03.

04.

05.import org.openqa.selenium.By;

06.

07.import org.openqa.selenium.WebDriver;

08.

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

10.

11.

12.

13.public class myclass {

14.

15.

16.

17.public static void main(String[] args) {

18.

19.WebDriver driver = new FirefoxDriver();

20.

21.

22.

23.driver.get("<http://selenium.googlecode.com/svn/trunk/docs/api/java/index.html>");

24.

25.driver.switchTo().frame("classFrame");

26.

27.driver.findElement(By.linkText("Deprecated")).click();

28.

29.driver.quit();

30.

31.

32.

33.}

34.

35.}

After executing this code, you will see that the “classFrame” frame is taken to the “Deprecated API” page, meaning that our code was successfully able to access the “Deprecated” link.

## Switching Between Pop-up Windows

WebDriver allows pop-up windows like alerts to be displayed, unlike in Selenium IDE. To access the elements within the alert (such as the message it contains), we must use the **“switchTo().alert()”** method. In the code below, we will use this method to access the alert box and then retrieve its message using the **“getText()”** method, and then automatically close the alert box using the **“switchTo().alert().accept()”** method.

First,  head over to <http://jsbin.com/usidix/1> and manually click the “Go!” button there and see for yourself the message text.



Lets see the WebDriver code to do this-

01.package mypackage;

02.

03.

04.

05.import org.openqa.selenium.By;

06.

07.import org.openqa.selenium.WebDriver;

08.

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

10.

11.

12.

13.public class myclass {

14.

15.

16.

17.public static void main(String[] args) {

18.

19.WebDriver driver = new FirefoxDriver();

20.

21.String alertMessage = "";

22.

23.

24.

25.driver.get("<http://jsbin.com/usidix/1>");

26.

27.driver.findElement(By.cssSelector("input[value=\"Go!\"]")).click();

28.

29.alertMessage = driver.switchTo().alert().getText();

30.

31.driver.switchTo().alert().accept();

32.

33.

34.

35.System.out.println(alertMessage);

36.

37.driver.quit();

38.

39.

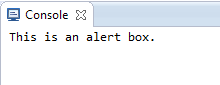
40.

41.}

42.

43.}

On the Eclipse console, notice that the printed alert message is:



## Waits

There are two kinds of waits.

1. Implicit wait – used to set the default waiting time throughout the program
2. Explicit wait – used to set the waiting time for a particular instance only

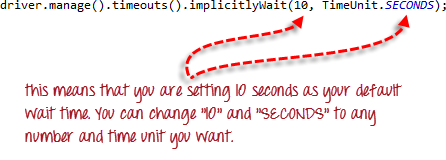
### Implicit Wait

* It is simpler to code than Explicit Waits.
* It is usually declared in the instantiation part of the code.
* You will only need one additional package to import.

To start using an implicit wait, you would have to import this package into your code.

http://www.guru99.com/images/image038%281%29.png

Then on the instantiation part of your code, add this.



### Explicit Wait

**Explicit waits are done using the WebDriverWait and ExpectedCondition classes**. For the following 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.

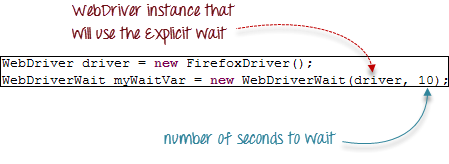
**Step 1**

Import these two packages:

**http://www.guru99.com/images/image040%281%29.png**

**Step 2**

Declare a WebDriverWait variable. In this example, we will use “myWaitVar” as the name of the variable.



**Step 3**

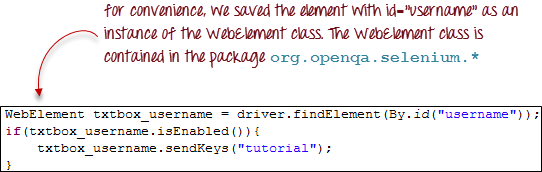
Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur. In this case, we will use explicit wait on the “username” (Mercury Tours HomePage) input before we type the text “tutorial” onto it.

http://www.guru99.com/images/image042%281%29.png

## Conditions

Following  methods are used  in conditional and looping operations --

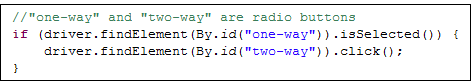
* **isEnabled()** is used when you want to verify whether a certain element is enabled or not before executing a command.



* **isDisplayed()** is used when you want to verify whether a certain element is displayed or not before executing a command.

http://www.guru99.com/images/image044%281%29.png

* **isSelected()** is used when you want to verify whether a certain **check box, radio button, or option in a drop-down box** is selected. It does not work on other elements.



## Using ExpectedConditions

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

Below are some of the most common ExpectedConditions methods.

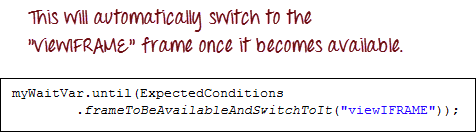
* **alertIsPresent()** – waits until an alert box is displayed.

http://www.guru99.com/images/image046%281%29.png

* **elementToBeClickable()** – waits until an element is visible and, at the same time, enabled. The sample code below will wait until the element with id=”username” to become visible and enabled first before assigning that element as a WebElement variable named “txtUserName”.

http://www.guru99.com/images/image047%281%29.png

* **frameToBeAvailableAndSwitchToIt()** – waits until the given frame is already available, and then automatically switches to it.



## Catching Exceptions

When using isEnabled(), isDisplayed(), and isSelected(), WebDriver assumes that the element already exists on the page. Otherwise, it will throw a **NoSuchElementException**. To avoid this, we should use a try-catch block so that the program will not be interrupted.

01.WebElement txtbox\_username = driver.findElement(By.id("username"));

02.

03.try{

04.

05.if(txtbox\_username.isEnabled()){

06.

07.txtbox\_username.sendKeys("tutorial");

08.

09.}

10.

11.}

12.

13.catch(NoSuchElementException nsee){

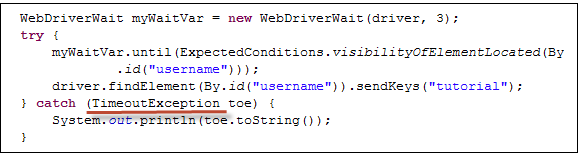
14.

15.System.out.println(nsee.toString());

16.

17.}

If you use explicit waits, the type of exception that you should catch is the “TimeoutException”.



## Summary

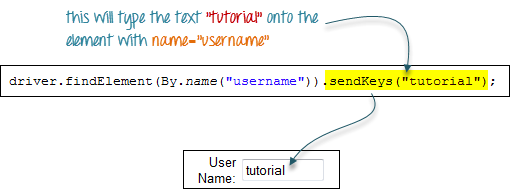
* To start using the WebDriver API, you must import at least these two packages.
* **org.openqa.selenium.\***
* **org.openqa.selenium.firefox.FirefoxDriver**
* The **get()** method is the equivalent of Selenium IDE’s “open” command.
* Locating elements in WebDriver is done by using the **findElement(By())** method.
* The following are the available options for locating elements in WebDriver:
* By.**className**
* By.**cssSelector**
* By.**id**
* By.**linkText**
* By.**name**
* By.**partialLinkText**
* By.**tagName**
* By.**xpath**
* TheBy.cssSelector() **does not** support the **“contains”** feature.
* You can instantiate an element using the **WebElement** class.
* Clicking on an element is done by using the **click()** method.
* WebDriver provides these useful **get commands**:
* get()
* getTitle()
* getPageSource()
* getCurrentUrl()
* getText()
* WebDriver provides these useful **navigation commands**
* navigate().forward()
* navigate().back()
* navigate().to()
* navigate().refresh()
* The close() and quit() methods are used to close browser windows. **Close()** is used to close a single window; while **quit()** is used to close all windows associated to the parent window that the WebDriver object was controlling.
* The **switchTo().frame()** and **switchTo().alert()** methods are used to direct WebDriver’s focus onto a frame or alert, respectively.
* **Implicit waits** are used to set the waiting time throughout the program, while **explicit waits** are used only on specific portions.
* You can use the **isEnabled(), isDisplayed(),isSelected(),** and a combination of **WebDriverWait** and **ExpectedConditions** methods when verifying the state of an element. However, they do not verify if the element exists.
* When isEnabled(), isDisplayed(),or isSelected() was called while the element was not existing, WebDriver will throw a **NoSuchElementException**.
* When WebDriverWait and ExpectedConditions methods were called while the element was not existing, WebDriver would throw a **TimeoutException**.

**Accessing Forms in webDriver**

|  |  |
| --- | --- |
| In this tutorial , we will learn how to access forms and its  elements using Webdriver Accessing Form Elements **Input Box**  Input boxes refer to either of these two types:   1. **Text Fields**– text boxes that accept typed values and show them as they are. 2. **Password Fields**– text boxes that accept typed values but mask them as a series of special characters (commonly dots and asterisks) to avoid sensitive values to be displayed.   http://www.guru99.com/images/image002%281%29.png |  |

### Entering Values in Input Boxes

The **sendKeys()** method is used to enter values into input boxes.



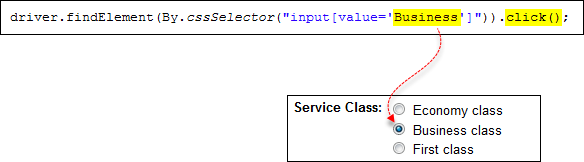
### Deleting Values in Input Boxes

The **clear()** method is used to delete the text in an input box. **This method does not need any parameter**. The code snippet below will clear out the text “tutorial” in the User Name text box.

http://www.guru99.com/images/image004%283%29.png

### Radio Button

Toggling a radio button on is done using the **click()** method.

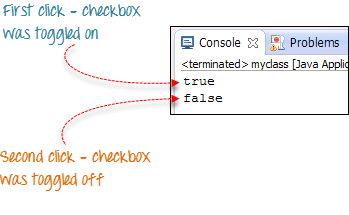


### Check Box

Toggling a check box on/off is also done using the **click()** method.

The code below will click on Facebook’s “Keep me logged in” check box twice and then output the result as TRUE when it is toggled on, and FALSE if it is toggled off.

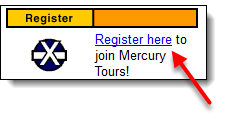




### Links

Links also are accessed by using the **click()** method.

Consider the below link found in Mercury Tours’ homepage.



You can access this link using linkText() or partialLinkText() together with click(). Either of the two lines below will be able to access the “Register here” link shown above.

http://www.guru99.com/images/image009%284%29.png

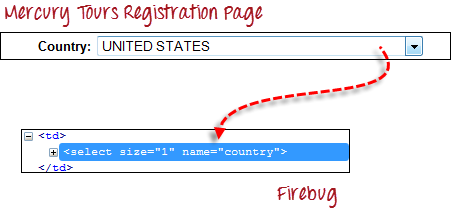
http://www.guru99.com/images/image010%283%29.png

### Drop-Down Box

Before we can control drop-down boxes, we must do following two things :

1. Import the package **org.openqa.selenium.support.ui.Select**
2. Instantiate the drop-down box as a “Select” object in WebDriver

As an example, go to Mercury Tours’ Registration page (<http://newtours.demoaut.com/mercuryregister.php>) and notice the “Country” drop-down box there.



**Step 1**

Import the “Select” package.

http://www.guru99.com/images/image012%281%29.png

**Step 2**

Declare the drop-down element as an instance of the Select class. In the example below, we named this instance as “drpCountry”.

http://www.guru99.com/images/image013%283%29.png

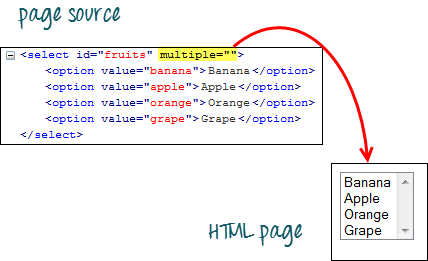
**Step 3**

We can now start controlling “drpCountry” by using any of the available Select methods. The sample code below will select the option “ANTARCTICA”.

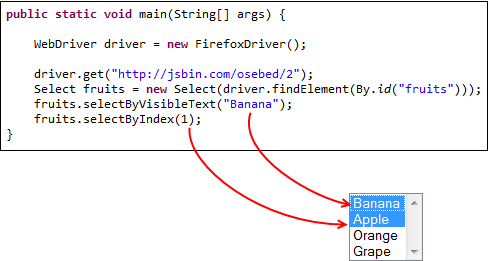
http://www.guru99.com/images/image014%283%29.png

### Selecting Items in a Multiple SELECT element

We can also use the **selectByVisibleText()** method in selecting multiple options in a multi SELECT element. As an example, we will take <http://jsbin.com/osebed/2> as the base URL. It contains a drop-down box that allows multiple selections at a time.



The code below will select the first two options using the selectByVisibleText() method.



### Select Methods

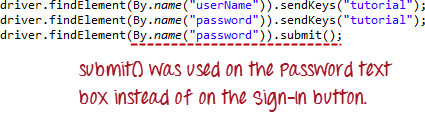
The following are the most common methods used on drop-down elements.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **selectByVisibleText()** and  **deselectByVisibleText()**  Example:  http://www.guru99.com/images/image017%283%29.png | * Selects/deselects the option that displays the text matching the parameter. * **Parameter**: The exactly displayed text of a particular option |
| **selectByValue()** and  **deselectByValue()**  Example:  http://www.guru99.com/images/image018%283%29.png | * Selects/deselects the option whose “value” attribute matches the specified parameter. * **Parameter**: value of the “value” attribute * Remember that not all drop-down options have the same text and “value”, like in the example below.   http://www.guru99.com/images/image019%282%29.png |
| **selectByIndex()** and  **deselectByValue()**  Example:  *http://www.guru99.com/images/image020%282%29.png* | * Selects/deselects the option at the given index. * **Parameter**: the index of the option to be selected. |
| **isMultiple()**  Example:  *http://www.guru99.com/images/image021%282%29.png* | * Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise. * **Needs parameters needed** |
| **deselectAll()**  Example:  *http://www.guru99.com/images/image022%282%29.png* | * Clears all selected entries. This is only valid when the drop-down element supports multiple selections. * **No parameters needed** |

### Submitting a Form

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.



**When submit() is used, WebDriver will look up the DOM to know which form the element belongs to, and then trigger its submit function.**

## Summary

* The table below summarizes the commands to access each type of element discussed above.

|  |  |  |
| --- | --- | --- |
| **Element** | **Command** | **Description** |
| **Input Box** | sendKeys() | used to enter values onto text boxes |
| clear() | used to clear text boxes of its current value |
| **Check Box,**  **Radio Button,** | click() | used to toggle the element on/off |
| **Links** | click() | used to click on the link and wait for page load to complete before proceeding to the next command. |
| **Drop-Down Box** | selectByVisibleText()/  deselectByVisibleText() | selects/deselects an option by its displayed text |
| selectByValue()/  deselectByValue() | selects/deselects an option by the value of its “value” attribute |
| selectByIndex()/  deselectByIndex() | selects/deselects an option by its index |
| isMultiple() | returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise |
| deselectAll() | deselects all previously selected options |
| **Submit Button** | submit() |  |

* WebDriver allows selection of more than one option in a multiple SELECT element.
* To control drop-down boxes, you must first import the org.openqa.selenium.support.ui.Select package and then create a Select instance.
* You can use the submit() method on any element within the form. WebDriver will automatically trigger the submit function of the form where that element belongs to.

**Accessing Links & Tables - Selenium Webdriver**

|  |
| --- |
| In this tutorial, we are going to learn about accessing links & Tables using Webdriver |

## Accessing Links

### Links Matching a Criterion

Links can be accessed using an exact or partial match of their link text. The examples below provide scenarios where multiple matches would exist, and would explain how WebDriver would deal with them.

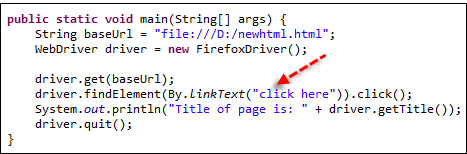
### Exact Match

**Accessing links using their exact link text is done through the By.linkText() method**. However, if there are two links that have the very same link text, this method will only access the first one. Consider the HTML code below

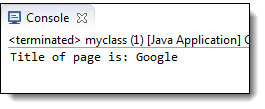




When you try to run the WebDriver code below, you will be accessing the first “click here” link

.

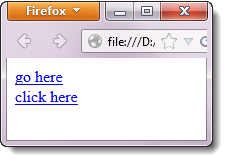
As a result, you will automatically be taken to Google.



### Partial Match

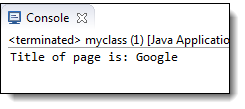
Accessing links using a portion of their link text is done using the **By.partialLinkText()** method. If you specify a partial link text that has multiple matches, only the first match will be accessed. Consider the HTML code below.





When you execute the WebDriver code below, you will still be taken to Google.



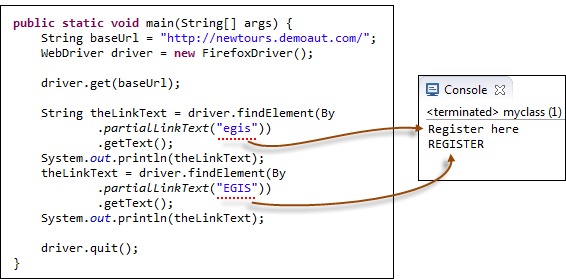


### Case-sensitivity

The parameters for **By.linkText()** and **By.partialLinkText()** are both case-sensitive, meaning that capitalization matters. For example, in Mercury Tours’ homepage, there are two links that contain the text “egis” – one is the “REGISTER” link found at the top menu, and the other is the “Register here” link found at the lower right portion of the page.



Though both links contain the character sequence “egis”, the "By.partialLinkText()" method will access these two links separately depending on capitilization of the characters. See the sample code below.



### All Links

One of the common procedures in web testing is to test if all the links present within the page are working. This can be conveniently done using a combination of the **Java for-each loop** and the **By.tagName(“a”)** method. The WebDriver code below checks each link from the Mercury Tours homepage to determine those that are working and those that are still under construction.

01.package practice\_webdriver;

02.

03.import java.util.List;

04.

05.

06.

07.import java.util.concurrent.TimeUnit;

08.

09.import org.openqa.selenium.\*;

10.

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

12.

13.import org.openqa.selenium.support.ui.ExpectedConditions;

14.

15.import org.openqa.selenium.support.ui.WebDriverWait;

16.

17.

18.

19.public class AllLinks {

20.

21.

22.

23.public static void main(String[] args) {

24.

25.String baseUrl = "<http://newtours.demoaut.com/>";

26.

27.WebDriver driver = new FirefoxDriver();

28.

29.String underConsTitle = "Under Construction: Mercury Tours";

30.

31.driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

32.

33.

34.

35.driver.get(baseUrl);

36.

37.List<WebElement> linkElements = driver.findElements(By.tagName("a"));

38.

39.String[] linkTexts = new String[linkElements.size()];

40.

41.int i = 0;

42.

43.

44.

45.//extract the link texts of each link element

46.

47.for (WebElement e : linkElements) {

48.

49.linkTexts[i] = e.getText();

50.

51.i++;

52.

53.}

54.

55.

56.

57.//test each link

58.

59.for (String t : linkTexts) {

60.

61.driver.findElement(By.linkText(t)).click();

62.

63.if (driver.getTitle().equals(underConsTitle)) {

64.

65.System.out.println("\"" + t + "\""

66.

67.+ " is under construction.");

68.

69.} else {

70.

71.System.out.println("\"" + t + "\""

72.

73.+ " is working.");

74.

75.}

76.

77.driver.navigate().back();

78.

79.}

80.

81.driver.quit();

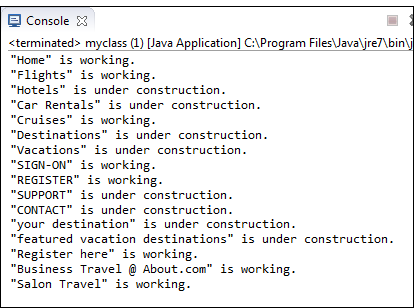
82.

83.}

84.

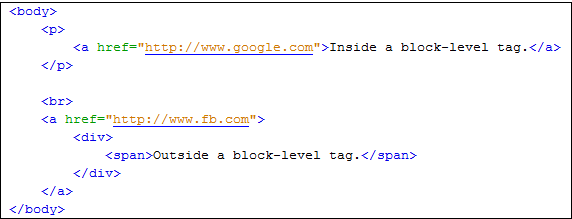
85.}

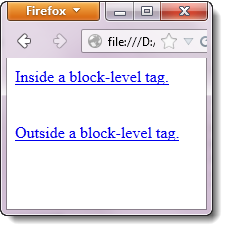
The output should be similar to the one indicated below.



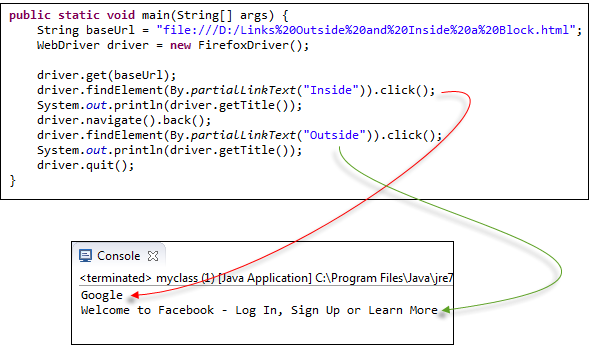
### Links Outside and Inside a Block

The latest HTML5 standard allows the <a> tags to be placed inside and outside of block-level tags like <div>, <p>, or <h1>. The "By.linkText()" and "By.partialLinkText()" methods can access a link located outside and inside these block-level elements. Consider the HTML code below.





The WebDriver code below accesses both of these links using By.partialLinkText() method.



The output above confirms that both links were accessed successfully because their respective page titles were retrieved correctly.

### Accessing Image Links

Image links are images that act as references to other sites or sections within the same page. Since they are images, we cannot use the By.linkText() and By.partialLinkText() methods because image links basically have no link texts at all. In this case, we should resort to using either By.cssSelector or By.xpath. The first method is more preferred because of its simplicity.

In the example below, we will access the “Facebook” logo on the upper left portion of Facebook’s Password Recovery page.



We will use By.cssSelector and the element’s “title” attribute to access the image link. And then we will verify if we are taken to Facebook’s homepage.

01.package practice\_webdriver;

02.

03.

04.

05.import org.openqa.selenium.\*;

06.

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

08.

09.

10.

11.public class ImageLink {

12.

13.

14.

15.public static void main(String[] args) {

16.

17.String baseUrl = "<https://www.facebook.com/login/identify?ctx=recover>";

18.

19.WebDriver driver = new FirefoxDriver();

20.

21.

22.

23.driver.get(baseUrl);

24.

25.//click on the "Facebook" logo on the upper left portion

26.

27.driver.findElement(By.cssSelector("a[title=\"Go to Facebook Home\"]")).click();

28.

29.

30.

31.//verify that we are now back on Facebook's homepage

32.

33.if (driver.getTitle().equals("Welcome to Facebook - Log In, Sign Up or Learn More")) {

34.

35.System.out.println("We are back at Facebook's homepage");

36.

37.} else {

38.

39.System.out.println("We are NOT in Facebook's homepage");

40.

41.}

42.

43.driver.quit();

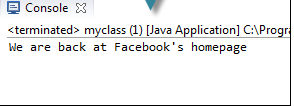
44.

45.}

46.

47.}

Result

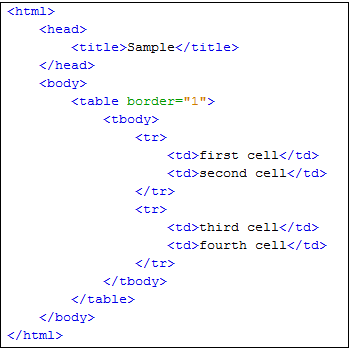


## Reading a Table

There are times when we need to access elements (usually texts) that are within HTML tables. However, it is very seldom for a web designer to provide an id or name attribute to a certain cell in the table. Therefore, we cannot use the usual methods such as “By.id()”, “By.name()”, or “By.cssSelector()”. In this case, the most reliable option is to access them using the “By.xpath()” method.

### XPath Syntax

Consider the HTML code below.



We will use XPath to get the inner text of the cell containing the text “fourth cell”.



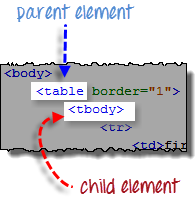
**Step 1 – Set the Parent Element (table)**

**XPath locators in WebDriver always start with a double forward slash “//” and then followed by the parent element**. Since we are dealing with tables, the parent element should always be the <table> tag. The first portion of our XPath locator should therefore start with “//table”.

http://www.guru99.com/images/image021%281%29.png

**Step 2 – Add the child elements**

The element immediately under <table> is <tbody> so we can say that <tbody> is the “child” of <table>. And also, <table> is the “parent” of <tbody>. All child elements in XPath are placed to the right of their parent element, separated with one forward slash “/” like the code shown below.

**Step 3 – Add Predicates**

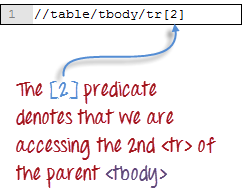
The <tbody> element contains two <tr> tags. We can now say that these two <tr> tags are “children” of <tbody>. Consequently, we can say that <tbody> is the parent of both the <tr> elements.

Another thing we can conclude is that the two <tr> elements are siblings. **Siblings refer to child elements having the same parent**.

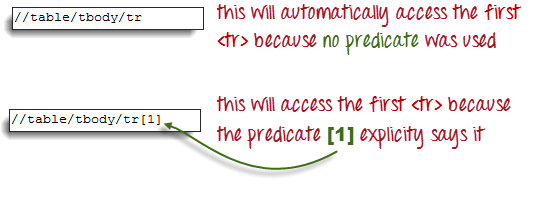
To get to the <td> we wish to access (the one with the text “fourth cell”), we must first access the **second** <tr> and not the first. If we simply write “//table/tbody/tr”, then we will be accessing the first <tr> tag.

So, how do we access the second <tr> then? The answer to this is to use **Predicates**.

**Predicates are numbers or HTML attributes enclosed in a pair of square brackets “[ ]” that distinguish a child element from its siblings**. Since the <tr> we need to access is the second one, we shall use “[2]” as the predicate.



If we won’t use any predicate, XPath will access the first sibling. Therefore, we can access the first <tr> using either of these XPath codes.

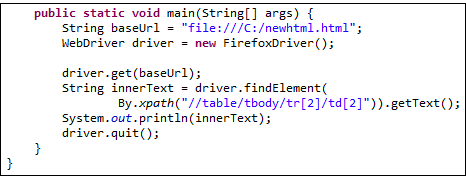


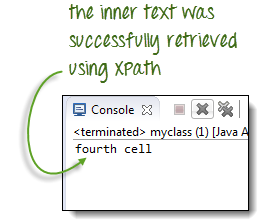
**Step 4 – Add the Succeeding Child Elements Using the Appropriate Predicates**

The next element we need to access is the second <td>. Applying the principles we have learned from steps 2 and 3, we will finalize our XPath code to be like the one shown below.

http://newguru99.revolutionventur.netdna-cdn.com/images/image026.png

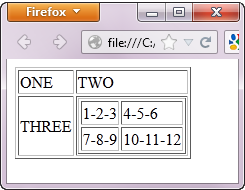
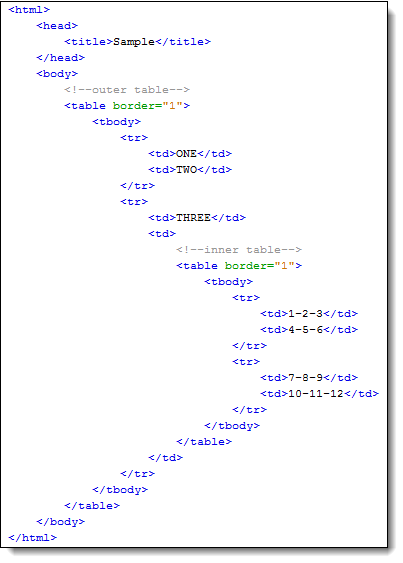
Now that we have the correct XPath locator, we can already access the cell that we wanted to and obtain its inner text using the code below. It assumes that you have saved the HTML code above as “newhtml.html” within your C Drive.



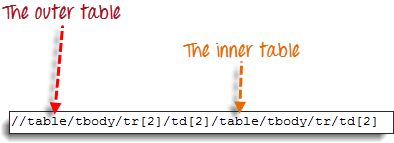


## Accessing Nested Tables

The same principles discussed above applies to nested tables. **Nested tables are tables located within another table**. An example is shown below.



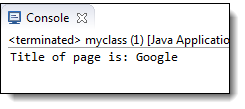
To access the cell with the text “4-5-6” using the “//parent/child” and predicate concepts from the previous section, we should be able to come up with the XPath code below.



The WebDriver code below should be able to retrieve the inner text of the cell which we are accessing.



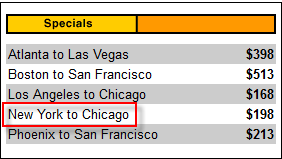
The output below confirms that the inner table was successfully accessed.

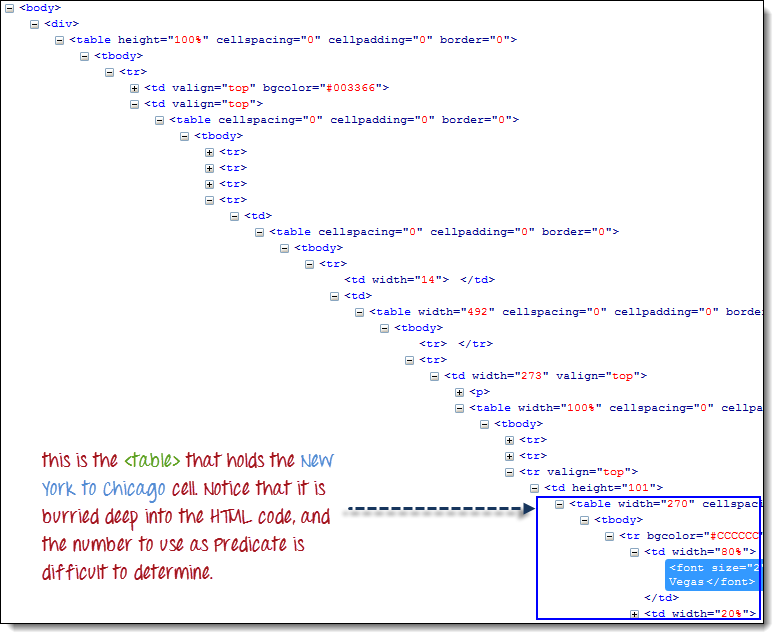


### Using Attributes as Predicates

If the element is written deep within the HTML code such that the number to use for the predicate is very difficult to determine, we can use that element’s unique attribute instead.

In the example below, the “New York to Chicago” cell is located deep into Mercury Tours homepage’s HTML code.

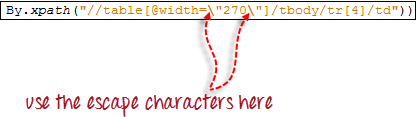




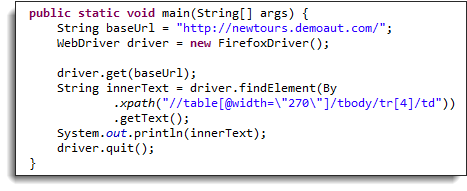
In this case, we can use the table’s unique attribute (width=”270”) as the predicate. **Attributes are used as predicates by prefixing them with the @ symbol**. In the example above, the “New York to Chicago” cell is located in the first <td> of the fourth <tr>, and so our XPath should be as shown below.

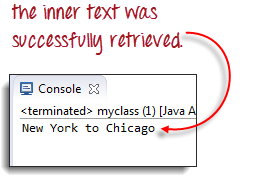
http://newguru99.revolutionventur.netdna-cdn.com/images/image036.png

Remember that when we put the XPath code in Java, we should use the escape character backward slash “\” for the double quotation marks on both sides of “270” so that the string argument of By.xpath() will not be terminated prematurely.



We are now ready to access that cell using the code below.





## Shortcut: Use Firebug

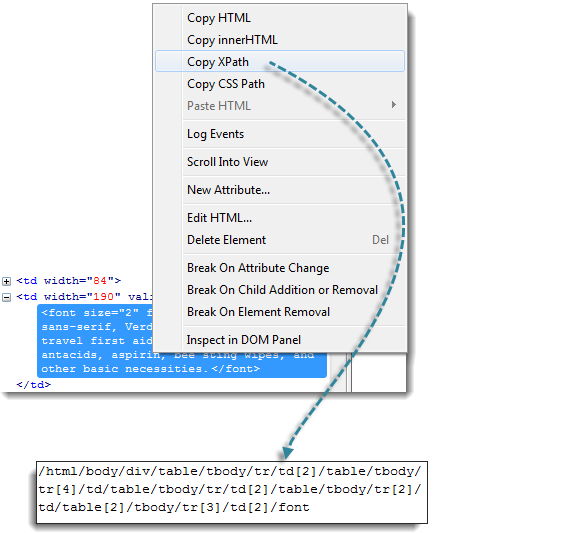
If the number or attribute of an element is extremely difficult or impossible to obtain, the quickest way to generate the XPath code is thru Firebug.

Consider the example below from Mercury Tours homepage.



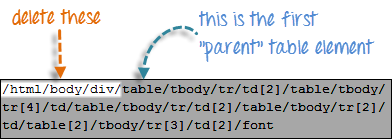
**Step 1**

Use Firebug to obtain the XPath code.



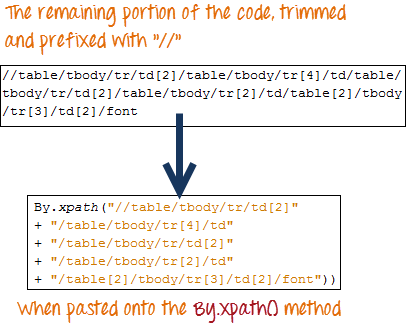
**Step 2**

Look for the first “table” parent element and delete everything to the left of it.

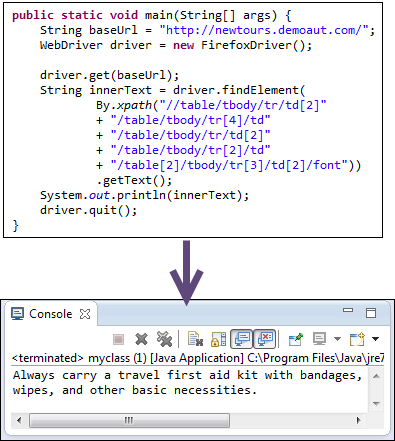


**Step 3**

Prefix the remaining portion of the code with double forward slash “//” and copy it over to your WebDriver code.



The WebDriver code below will be able to successfully retrieve the inner text of the element we are accessing.



## Summary

* Accessing links using their exact match is done using By.linkText() method.
* Accessing links using their partial match is done using By.partialLinkText() method.
* If there are multiple matches, By.linkText() and By.partialLinkText() will only select the first match.
* Pattern matching using By.linkText() and By.partialLinkText() is case-sensitive.
* The By.tagName("a") method is used to fetch all links within a page.
* Links can be accessed by the By.linkText() and By.partialLinkText() whether they are inside or outside block-level elements.
* Accessing image links are done using By.cssSelector() and By.xpath() methods.
* By.xpath() is commonly used to access table elements.

**Key Boards & Mouse Events**

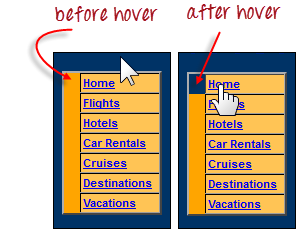
## Handling Keyboard & Mouse Events



**Handling special keyboard and mouse events are done using the Advanced User Interactions API**. It contains the **Actions** and the **Action** classes that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  source- element to emulate button down at.  target- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  source- element to emulate button down at.  xOffset- horizontal move offset.  yOffset- vertical move offset. |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  x-offset- horizontal offset. A negative value means moving the mouse left.  y-offset- vertical offset. A negative value means moving the mouse up. |
| **moveToElement(toElement)** | Moves the mouse to the middle of the element. **Parameters**:  toElement- element to move to. |
| **release()** | Releases the depressed left mouse button at the current mouse location |
| **sendKeys(onElement, charsequence)** | Sends a series of keystrokes onto the element. **Parameters**:  onElement - element that will receive the keystrokes, usually a text field  charsequence – any string value representing the sequence of keystrokes to be sent |

In the following example, we shall use the moveToElement() method to mouse-over on one Mercury Tours’ table rows. See the example below.



The cell shown above is a portion of a <TR> element. If not hovered, its color is #FFC455 (orange). After hovering, the cell’s color becomes transparent. It becomes the same color as the blue background of the whole orange table.

**Step 1**

Import the **Actions** and **Action** classes.

http://newguru99.revolutionventur.netdna-cdn.com/images/image047.png

**Step 2**

Instantiate a new Actions object.

http://newguru99.revolutionventur.netdna-cdn.com/images/image048.png

**Step 3**

Instantiate an Action using the Actions object in step 2.

http://newguru99.revolutionventur.netdna-cdn.com/images/image049.png

In this case, we are going to use the moveToElement() method because we are simply going to mouse-over the “Home” link. The build() method is always the final method used so that all the listed actions will be compiled into a single step.

**Step 4**

Use the perform() method when executing the Action object we designed in Step 3.

http://newguru99.revolutionventur.netdna-cdn.com/images/image050.png

Below is the whole WebDriver code to check the background color of the <TR> element before and after the mouse-over.

01.package mypackage;

02.

03.

04.

05.import org.openqa.selenium.\*;

06.

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

08.

09.

10.

11.import org.openqa.selenium.interactions.Action;

12.

13.import org.openqa.selenium.interactions.Actions;

14.

15.

16.

17.public class myclass {

18.

19.

20.

21.public static void main(String[] args) {

22.

23.String baseUrl = "<http://newtours.demoaut.com/>";

24.

25.WebDriver driver = new FirefoxDriver();

26.

27.

28.

29.driver.get(baseUrl);

30.

31.WebElement link\_Home = driver.findElement(By.linkText("Home"));

32.

33.WebElement td\_Home = driver

34.

35..findElement(By

36.

37..xpath("//html/body/div"

38.

39.+ "/table/tbody/tr/td"

40.

41.+ "/table/tbody/tr/td"

42.

43.+ "/table/tbody/tr/td"

44.

45.+ "/table/tbody/tr"));

46.

47.

48.

49.Actions builder = new Actions(driver);

50.

51.Action mouseOverHome = builder

52.

53..moveToElement(link\_Home)

54.

55..build();

56.

57.

58.

59.String bgColor = td\_Home.getCssValue("background-color");

60.

61.System.out.println("Before hover: " + bgColor);

62.

63.mouseOverHome.perform();

64.

65.bgColor = td\_Home.getCssValue("background-color");

66.

67.System.out.println("After hover: " + bgColor);

68.

69.driver.quit();

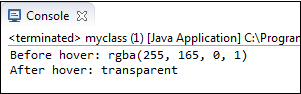
70.

71.}

72.

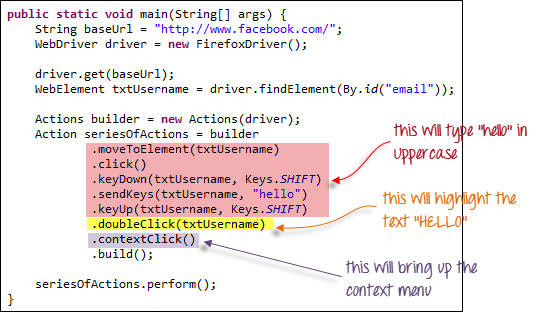
73.}

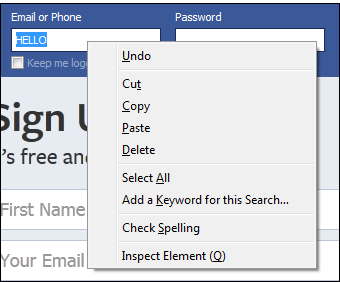
The output below clearly states that the background color became transparent after the mouse-over.



### Building a Series of Multiple Actions

**You can build a series of actions using the Action and Actions classes**. Just remember to close the series with the build() method. Consider the sample code below.

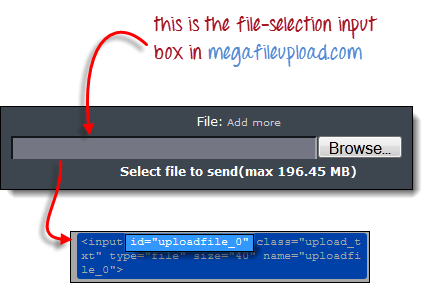




## Uploading Files

For this section, we will use <http://www.megafileupload.com/> as our test application. This site easily allows any visitor to upload and download files without requiring them to sign up.

**Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.**



Let’s say we wish to upload the file “C:\newhtml.html”. Our WebDriver code should be like the one shown below.

01.package mypackage;

02.

03.

04.

05.

06.

07.import org.openqa.selenium.\*;

08.

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

10.

11.public class myclass {

12.

13.public static void main(String[] args) {

14.

15.String baseUrl = "<http://www.megafileupload.com/>";

16.

17.WebDriver driver = new FirefoxDriver();

18.

19.

20.

21.driver.get(baseUrl);

22.

23.WebElement uploadElement = driver.findElement(By.id("uploadfile\_0"));

24.

25.

26.

27.// enter the file path onto the file-selection input field

28.

29.uploadElement.sendKeys("C:\\newhtml.html");

30.

31.

32.

33.// check the "I accept the terms of service" check box

34.

35.driver.findElement(By.id("terms")).click();

36.

37.

38.

39.// click the "UploadFile" button

40.

41.driver.findElement(By.name("send")).click();

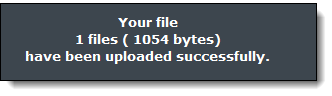
42.

43.}

44.

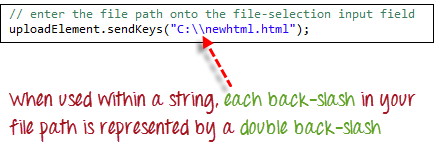
45.}

After running this script, you should be able to upload the file successfully and you should get a message similar to this.



Remember following two things when uploading files in WebDriver

1. There is no need to simulate the clicking of the “Browse” button.  WebDriver automatically enters the file path onto the file-selection text box of the <input type=”file”> element
2. When setting the file path in your Java IDE, use the proper escape character for the back-slash.



## Downloading Files

**WebDriver has no capability to access the Download dialog boxes** presented by browsers when you click on a download link or button. However, we can bypass these dialog boxes using a separate program called “wget”.

### What is Wget?

**Wget is a small and easy-to-use command-line program used to automate downloads**. Basically, we will access Wget from our WebDriver script to perform the download process.

### ****Setting up Wget****

**Step 1**

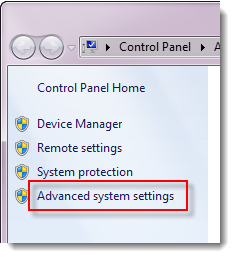
In your C Drive, create a new folder and name it as “Wget”.

Download wget.exe [he­re](http://users.ugent.be/%7Ebpuype/cgi-bin/fetch.pl?dl=wget/wget.exe) and place it in the Wget folder you created from the step above.

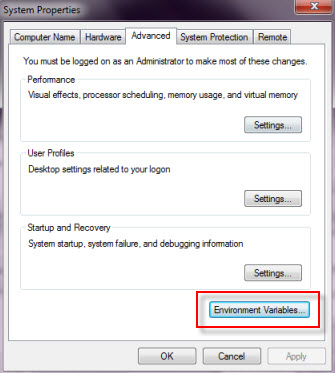
**Step 2**

Bring up the System Properties window by pressing Win + Pause on your keyboard.

Click on “Advanced System Settings”.

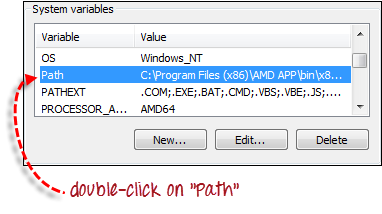


Click on the “Environment Variables”.



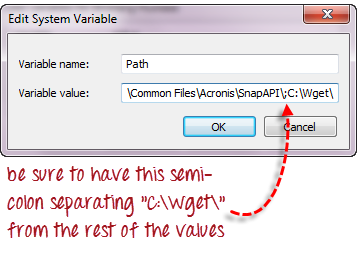
**Step 3**

In the Environment Variables dialog box, in the “System variables” section, scroll down the list until you find “Path” and then double-click it.



**Step 4**

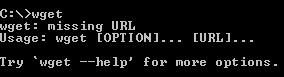
In the “Variable value” text box, add the new path “C:\Wget\”. Just be sure that there is a semi-colon separating this new entry from the preexisting values.



**Step 5**

Click OK on all dialog boxes.

Launch the command prompt and type the command “wget”. You should get the following response.



### Using WebDriver and Wget

In the following example, we will use WebDriver and wget to download a popular chat software called Yahoo Messenger. Our base URL shall be <http://messenger.yahoo.com/>.



**Step 1**

Import the “java.io.IOException” package because we will have to catch an IOException later in Step 4.

http://newguru99.revolutionventur.netdna-cdn.com/images/image065.png

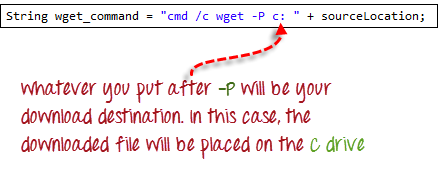
**Step 2**

Use getCssAttribute() to obtain the “href” value of the download link and save it as a String variable. In this case, we named the variable as “sourceLocation”.

http://newguru99.revolutionventur.netdna-cdn.com/images/image066.png

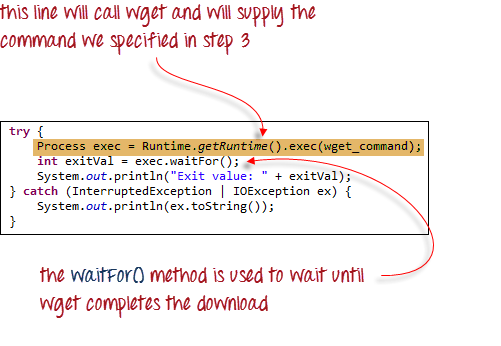
**Step 3**

Set-up the syntax for wget using the following command.



**Step 4**

Initiate the download process by calling wget from our WebDriver code.



To sum it all up, your WebDriver code could look like the one shown below.

01.package mypackage;

02.

03.import java.io.IOException;

04.

05.import org.openqa.selenium.\*;

06.

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

08.

09.public class myclass {

10.

11.

12.

13.public static void main(String[] args) {

14.

15.String baseUrl = "<http://messenger.yahoo.com/>";

16.

17.WebDriver driver = new FirefoxDriver();

18.

19.

20.

21.driver.get(baseUrl);

22.

23.WebElement downloadButton = driver.findElement(By

24.

25..id("messenger-download"));

26.

27.String sourceLocation = downloadButton.getAttribute("href");

28.

29.String wget\_command = "cmd /c wget -P c: " + sourceLocation;

30.

31.

32.

33.try {

34.

35.Process exec = Runtime.getRuntime().exec(wget\_command);

36.

37.int exitVal = exec.waitFor();

38.

39.System.out.println("Exit value: " + exitVal);

40.

41.} catch (InterruptedException | IOException ex) {

42.

43.System.out.println(ex.toString());

44.

45.}

46.

47.driver.quit();

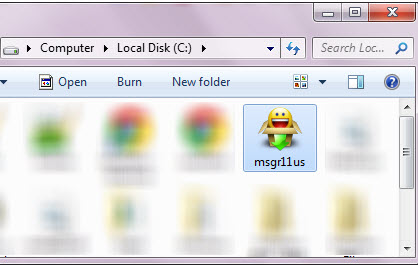
48.

49.}

50.

51.}

After executing this code, check your C drive and verify that the Yahoo Messenger installer was successfully downloaded there.



## Summary

* Handling special keyboard and mouse events are done using the AdvancedUserInteractions API.
* Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.
* WebDriver cannot automate downloading of files on its own.
* The eaasiest way to download files using WebDriver is to use Wget.

**All about TestNG and Selenium**

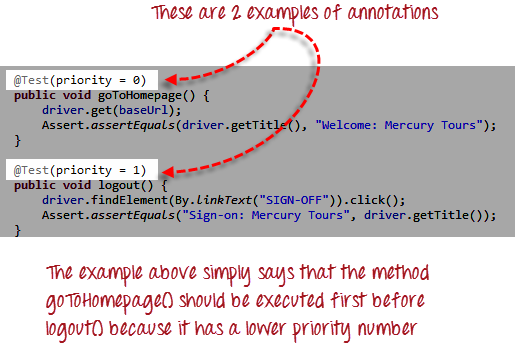
|  |
| --- |
| What is TestNG? So far we had been doing Selenium tests without generating a proper format for the test results. From this point on, we shall tackle how to make these reports using a test framework called TestNG.  TestNG is a testing framework that overcomes the limitations of another popular testing framework called JUnit. The “NG” means “Next Generation”. Most Selenium users use this more than JUnit because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium. |

## Advantages of TestNG over JUnit

There are three major advantages of TestNG over JUnit:

* Annotations are easier to understand
* Test cases can be grouped more easily
* Parallel testing is possible

**Annotations in TestNG are lines of code that can control how the method below them will be executed**. They are always preceded by the @ symbol. A very early and quick example is the one shown below.



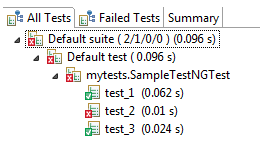
Annotations will be discussed later in the section named “Annotations used in TestNG”, so it is perfectly ok if you do not understand the above example just yet. It is just important to note for now that annotations in TestNG are easier to code and understand than in JUnit.

The ability to run tests in parallel is available in TestNG but not in JUnit, so it is the more preferred framework of testers using Selenium Grid.

## Why do we need TestNG in Selenium?

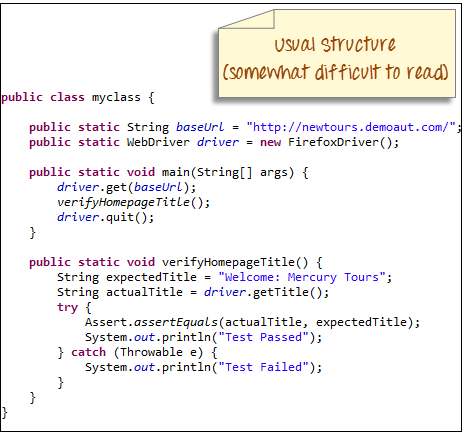
TestNG can generate reports based on our Selenium test results.

* WebDriver has no native mechanism for generating reports.
* TestNG can generate the report in a readable format like the one shown below.



TestNG simplifies the way the tests are coded

* There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.



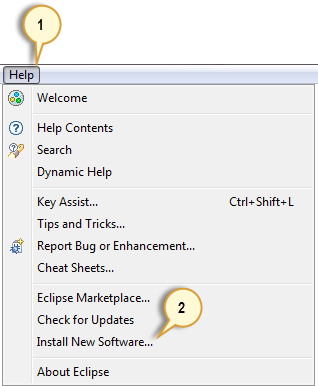


* Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.

## Installing TestNG in Eclipse

**Step 1**

* Launch Eclipse.
* On the menu bar, click Help.
* Choose the “Install New Software…” option.



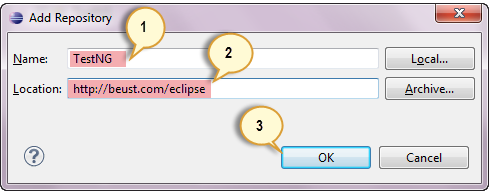
**Step 2**

In the Install dialog box, click the Add button

http://newguru99.revolutionventur.netdna-cdn.com/images/step_2-0047.png

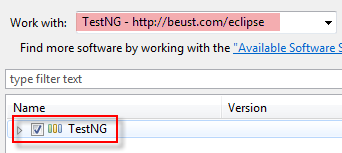
**Step 3**

1. In “Name”, type TestNG.
2. In “Location”, type <http://beust.com/eclipse>.
3. Click OK



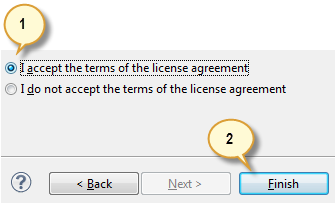
**Step 4**

* Notice that “TestNG - <http://beust.com/eclipse>” was populated onto the “Work with:” textbox.
* Check the “TestNG” check box as shown below, then click Next.



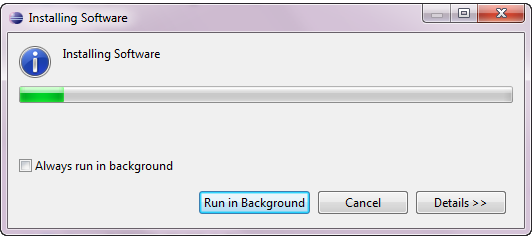
**Step 5**

* Click Next again on the succeeding dialog box until you reach the License Agreement dialog.
* Click “I accept the terms of the license agreement” then click Finish.

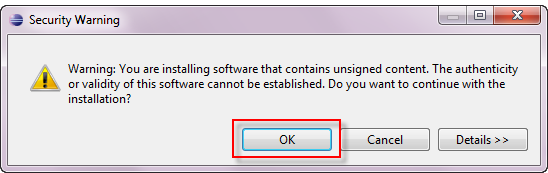


**Step 6**

Wait for the installation to finish

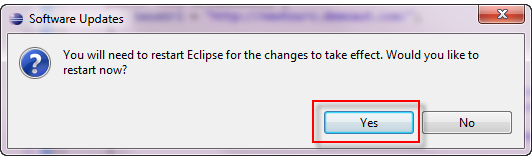


If you encounter a Security warning, just click OK



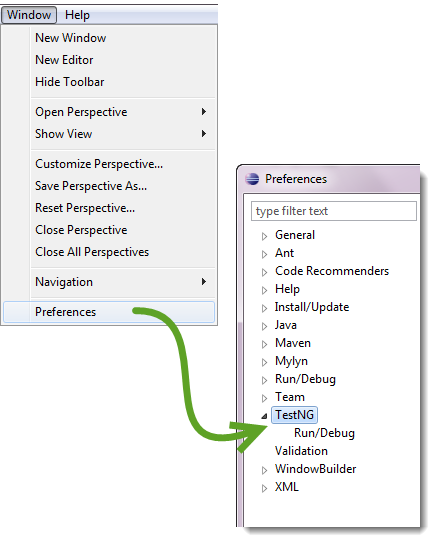
**Step 7**

When Eclipse prompts you for a restart, just click Yes.



**Step 8**

After restart, verify if TestNG was indeed successfully installed. Click Window > Preferences and see if TestNG is included on the Preferences list.



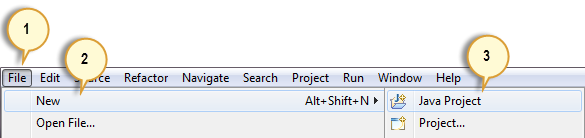
## First test case using annotations

Before we create a test case, we should first setup a new TestNG Project in Eclipse and name it as “FirstTestNGProject”.

### Setting up a new TestNG Project

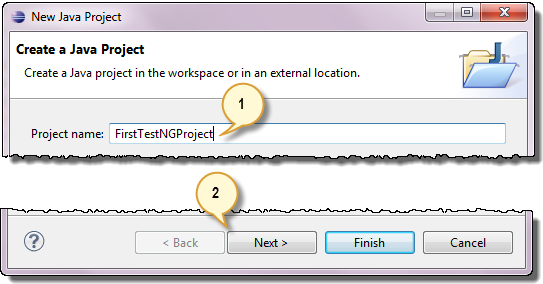
**Step 1**

Click File > New > Java Project



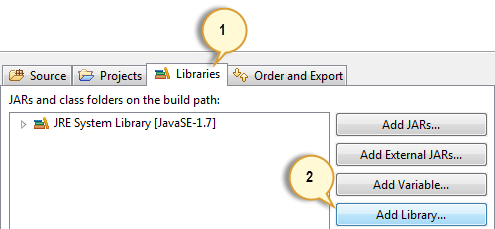
**Step 2**

Type “FirstTestNGProject” as the Project Name then click Next.



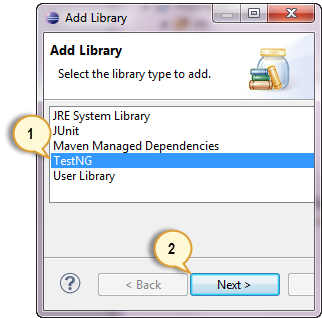
**Step 3**

We will now start to import the TestNG Libraries onto our project. Click yon the “Libraries” tab, and then “Add Library…”



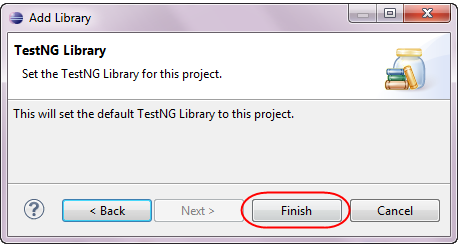
**Step 4**

On the Add Library dialog, choose “TestNG” and click Next.

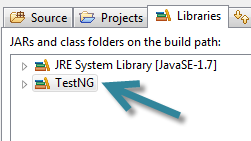


**Step 5**

Click Finish.

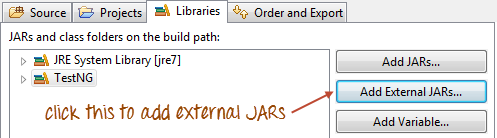


You should notice that TestNG is included on the Libraries list.

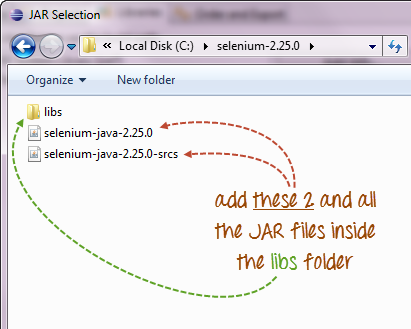


**Step 6**

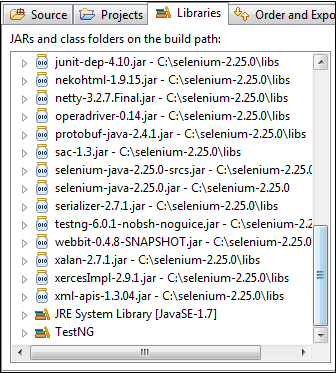
We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from <http://docs.seleniumhq.org/download/> when we were installing Selenium and Eclipse in the previous chapters.



Then, navigate to where you have placed the Selenium JAR files.

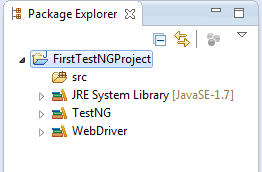


After adding the external JARs, your screen should look like this.



**Step 7**

Click Finish and verify that our FirstTestNGProject is visible on Eclipse’s Package Explorer window.



## Creating a New TestNG Test File

Now that we are done setting up our project, let us create a new TestNG file.

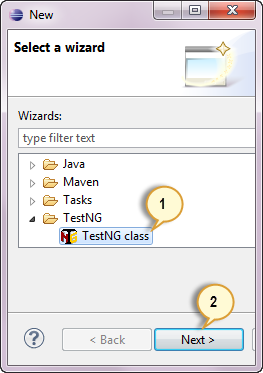
**Step 1**

Right-click on the “src” package folder then choose New > Other…



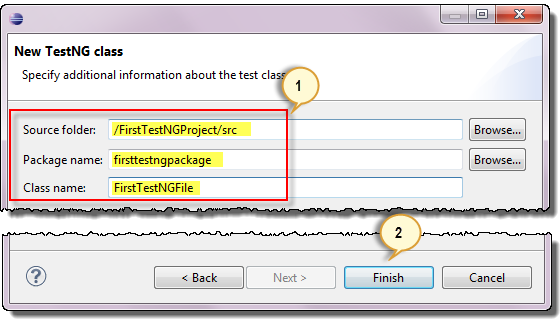
**Step 2**

Click on the TestNG folder and select the “TestNG class” option. Click Next.

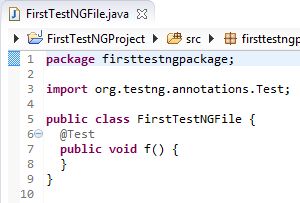


**Step 3**

Type the values indicated below on the appropriate input boxes and click Finish. Notice that we have named our Java file as “FirstTestNGFile”.

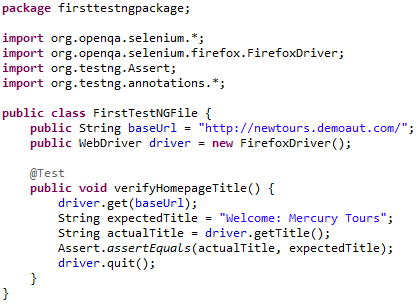


Eclipse should automatically create the template for our TestNG file shown below.



### Coding Our First Test Case

Let us now create our first test case that will check if Mercury Tours’ homepage is correct. Type your code as shown below.



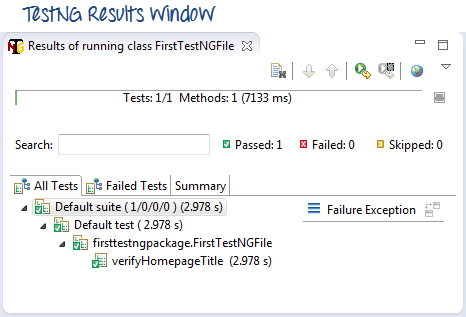
Notice the following.

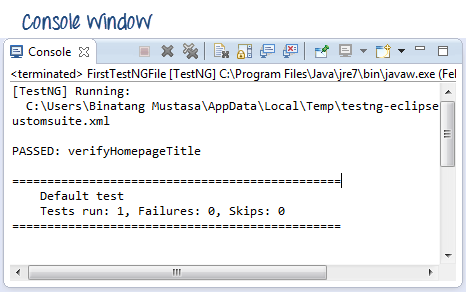
* TestNG does not require you to have a main() method.
* Methods need not be static.
* We used the @Test annotation. **@Test is used to tell that the method under it is a test case**. In this case, we have set the verifyHomepageTitle() method to be our test case so we placed an ‘@Test’ annotation above it.
* Since we use annotations in TestNG, we needed to import the package org.testng.annotations.\*.
* We used the Assert class. **The Assert class is used to conduct verification operations in TestNG**. To use it, we need to import the org.testng.Assert package.

You may have multiple test cases (therefore, multiple @Test annotations) in a single TestNG file. This will be tackled in more detail later in the section “Annotations used in TestNG”.

### Running the Test

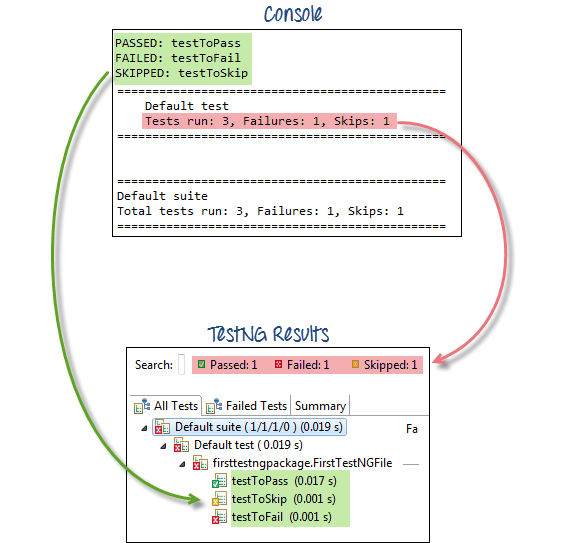
To run the test, simply run the file in Eclipse as you normally do. Eclipse will provide two outputs – one in the Console window and the other on the TestNG Results window.





## Checking reports created by TestNG

The Console window in Eclipse gives a text-based report of our test case results while the TestNG Results window gives us a graphical one.

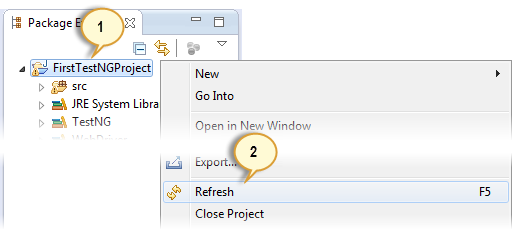


### Generating HTML Reports

TestNG has the ability to generate reports in HTML format.

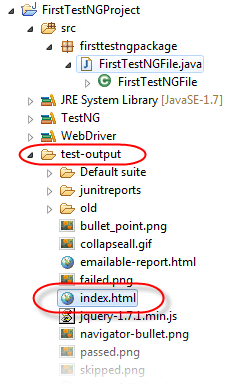
**Step 1**

After running our FirstTestNGFile that we created in the previous section, right-click the project name (FirstTestNGProject) in the Project Explorer window then click on the “Refresh” option.



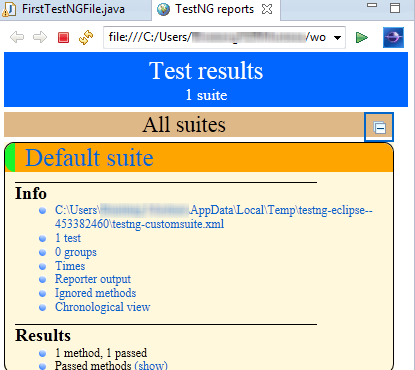
**Step 2**

Notice that a “test-output” folder was created. Expand it and look for an index.html file. This HTML file is a report of the results of the most recent test run.



**Step 3**

Double-click on that index.html file to open it within Eclipse’s built-in web browser. You can refresh this page any time after you rerun your test by simply pressing F5 just like in ordinary web browsers.

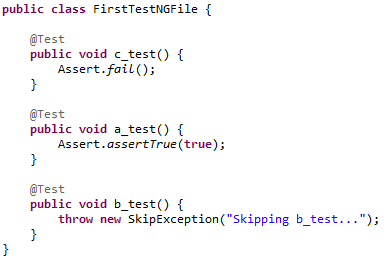


## Annotations used in TestNG

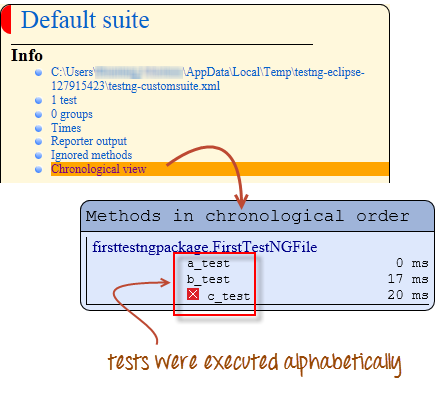
In the previous section, you have been introduced to the @Test annotation. Now, we shall be studying more advanced annotations and their usages.

### Multiple Test Cases

We can use multiple @Test annotations in a single TestNG file. By default, methods annotated by @Test are executed alphabetically. See the code below. Though the methods c\_test, a\_test, and b\_test are not arranged alphabetically in the code, they will be executed as such.



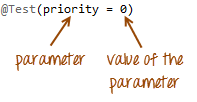
Run this code and on the generated index.html page, click “Chronological view”.



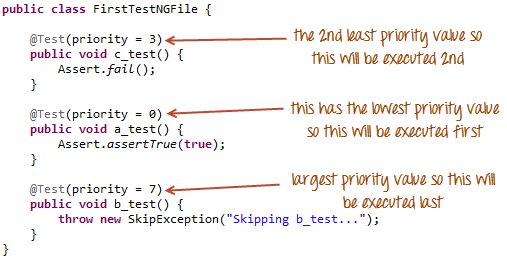
### Parameters

If you want the methods to be executed in a different order, use the parameter “priority”. **Parameters are keywords that modify the annotation’s function**.

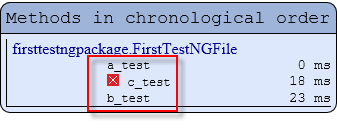
* Parameters require you to assign a value to them. You do.this by placing a “=” next to them, and then followed by the value.
* Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.



TestNG will execute the @Test annotation with the lowest priority value up to the largest. There is no need for your priority values to be consecutive.



The TestNG HTML report will confirm that the methods were executed based on the ascending value of priority.



### Multiple Parameters

Aside from “priority”, @Test has another parameter called “alwaysRun” which can only be set to either “true” or “false”. **To use two or more parameters in a single annotation, separate them with a comma** such as the one shown below.

### http://newguru99.revolutionventur.netdna-cdn.com/images/multiple_parameters-0010.png

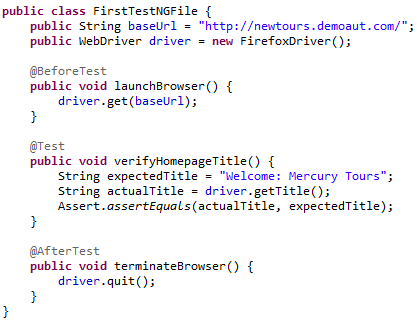


### 

@BeforeTest and @AfterTest

|  |  |
| --- | --- |
| **@BeforeTest** | methods under this annotation will be executed **prior to the first test case in the TestNG file**. |
| **@AfterTest** | methods under this annotation will be executed **after all test cases in the TestNG file are executed**. |

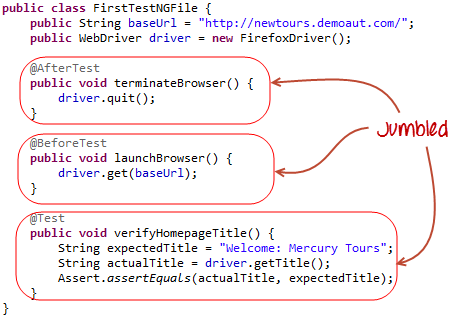
Consider the code below.



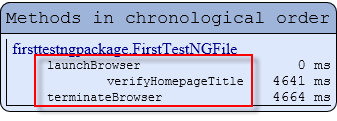
Applying the logic presented by the table and the code above, we can predict that the sequence by which methods will be executed is:

* 1st – launchBrowser()
* 2nd – verifyHomepageTitle()
* 3rd – terminateBrowser()

**The placement of the annotation blocks can be interchanged without affecting the chronological order by which they will be executed**. For example, try to rearrange the annotation blocks such that your code would look similar to the one below.



Run the code above and notice that



@BeforeMethod and @AfterMethod

|  |  |
| --- | --- |
| **@BeforeMethod** | methods under this annotation will be executed **prior to each method in each test case**. |
| **@AfterMethod** | methods under this annotation will be executed **after each method in each test case.** |

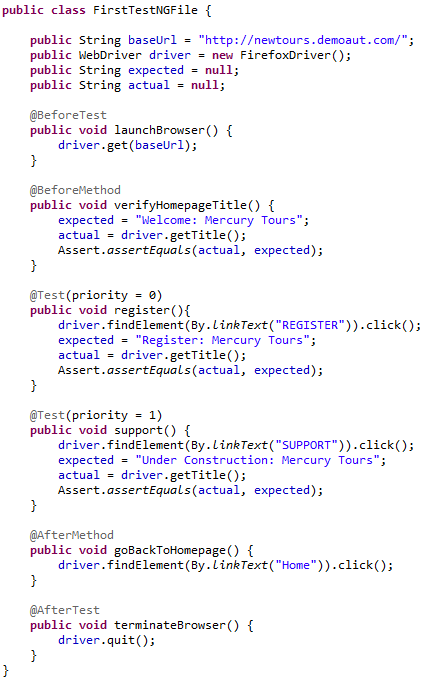
In Mercury Tours, suppose we like to verify the titles of the target pages of the two links below.



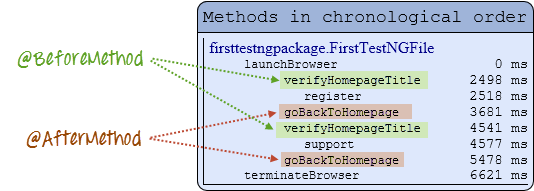
The flow of our test would be:

* Go to the homepage and verify its title.
* Click REGISTER and verify the title of its target page.
* Go back to the homepage and verify if it still has the correct title.
* Click SUPPORT and verify the title of its target page.
* Go back to the homepage and verify if it still has the correct title.

The code below illustrates how @BeforeMethod and @AfterMethod are used to efficiently execute the scenario mentioned above.



After executing this test, your TestNG should report the following sequence.



Simply put, @BeforeMethod should contain methods that you need to run **before** each test case while @AfterMethod should contain methods that you need to run **after** each test case.

### Summary of TestNG Annotations

**@BeforeSuite**: The annotated method will be run before all tests in this suite have run.

**@AfterSuite**: The annotated method will be run after all tests in this suite have run.

**@BeforeTest**: The annotated method will be run before any test method belonging to the classes inside the tag is run.

**@AfterTest**: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

**@BeforeGroups**: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

**@AfterGroups**: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

**@BeforeClass**: The annotated method will be run before the first test method in the current class is invoked.

**@AfterClass**: The annotated method will be run after all the test methods in the current class have been run.

**@BeforeMethod**: The annotated method will be run before each test method.

**@AfterMethod**: The annotated method will be run after each test method.

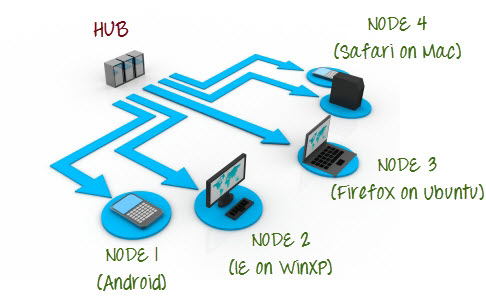
**@Test**: The annotated method is a part of a test case

## Conclusion

* TestNG is a testing framework that is capable of making Selenium tests easier to understand and of generating reports that are easy to understand.
* The main advantages of TestNG over JUnit are the following.
* Annotations are easier to use and understand.
* Test cases can be grouped more easily.
* TestNG allows us to create parallel tests.
* The Console window in Eclipse generates a text-based result while the TestNG window is more useful because it gives us a graphical output of the test result plus other meaningful details such as:
  + Runtimes of each method.
  + The chronological order by which methods were executed.
* TestNG is capable of generating HTML-based reports.
* Annotations can use parameters just like the usual Java methods.

**Introduction to Selenium Grid**

|  |
| --- |
| What is Selenium Grid?   **Selenium Grid is a part of the Selenium Suite that specializes on running multiple tests across different browsers, operating systems, and machines in parallel**.  Selenium Grid has 2 versions – the older Grid 1 and the newer Grid 2. We will only focus on Grid 2 because Grid 1 is gradually being deprecated by the Selenium Team. |



Selenium Grid uses a hub-node concept where you only run the test on a single machine called a **hub**, but the execution will be done by different machines called **nodes**.

## When to Use Selenium Grid?

You should use Selenium Grid when you want to do either one or both of following :

* **Run your tests against different browsers, operating systems, and machines all at the same time.**This will ensure that the application you are testing is fully compatible with a wide range of browser-OS combinations.
* **Save time in execution of your test suites**. If you set up Selenium Grid to run, say, 4 tests at a time, then you would be able to finish the whole suite around 4 times faster.

## Grid 1.0 Vs Grid 2.0

Following are the main differences between Selenium Grid 1 and 2.

|  |  |
| --- | --- |
| **Grid 1** | **Grid 2** |
| Selenium Grid 1 has its own remote control that is different from the Selenium RC server. They are two different programs. | Selenium Grid 2 is now bundled with the Selenium Server jar file |
| You need to install and configure Apache Ant first before you can use Grid 1. | You do not need to install Apache Ant in Grid 2. |
| Can only support Selenium RC commands/scripts. | Can support both Selenium RC and WebDriver scripts. |
| You can only automate one browser per remote control. | One remote control can automate up to 5 browsers. |

## What is a Hub and Node?

### The Hub

* The hub is the central point where you load your tests into.
* There should only be one hub in a grid.
* The hub is launched only on a single machine, say, a computer whose OS is Windows 7 and whose browser is IE.
* The machine containing the hub is where the tests will be run, but you will see the browser being automated on the node.

### The Nodes

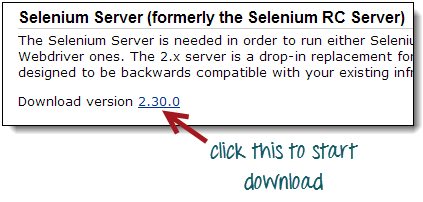
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* There can be one or more nodes in a grid.
* Nodes can be launched on multiple machines with different platforms and browsers.
* The machines running the nodes need not be the same platform as that of the hub.

## How to Install and Use Grid 2.0?

In this section, you will use 2 machines. The first machine will be the system that will run the hub, while the other machine will run a node. For simplicity, let us call the machine where the hub runs as “Machine A” while the machine where the node runs will be “Machine B”. It is also important to note their IP addresses. Let us say that Machine A has an IP address of 192.168.1.3 while Machine B has an IP of 192.168.1.4.

**Step 1**

Download the Selenium Server by [here](http://docs.seleniumhq.org/download/).



**Step 2**

You can place the Selenium Server .jar file anywhere in your HardDrive.But for the purpose of this tutorial, place it on the C drive of both Machine A and Machine B. After doing this, you are now done installing Selenium Grid. The following steps will launch the hub and the node.

**Step 3**

* We are now going to launch a hub. Go to Machine A. Using the command prompt, navigate to the root of Machine A’s - C drive ,because that is the directory where we placed the Selenium Server.
* On the command prompt, type **java –jar selenium-server-standalone-2.30.0.jar –role hub**
* The hub should successfully be launched. Your command prompt should look similar to the image below



**Step 4**

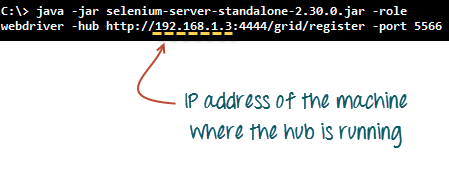
Another way to verify whether the hub is running is by using a browser. Selenium Grid, by default, uses Machine A’s port 4444 for its web interface. Simply open up a browser and go to <http://localhost:4444/grid/console>



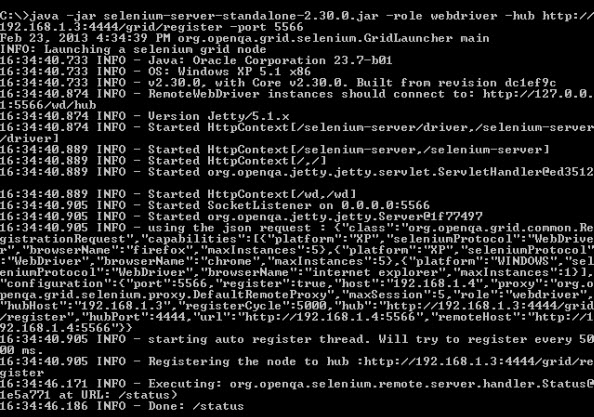
Also, you can check if Machine B can access the hub’s web interface by launching a browser there and going to [http://iporhostnameofmachineA:4444/grid/console](http://iporhostnameofmachinea:4444/grid/console) where “iporhostnameofmachineA” should be the IP address or the hostname of the machine where the hub is running. Since Machine A’s IP address is 192.168.1.3, then on the browser on Machine B you should type [http://192.168.1.3:4444/grid/console](http://192.168.1.11:4444/grid/console)

**Step 5**

* Now that the hub is already set up, we are going to launch a node. Go to Machine B and launch a command prompt there.
* Navigate to the root of Drive C and type the code below. We used the IP address 192.168.1.3 because that is where the hub is running. We also used port 5566 though you may choose any free port number you desire.

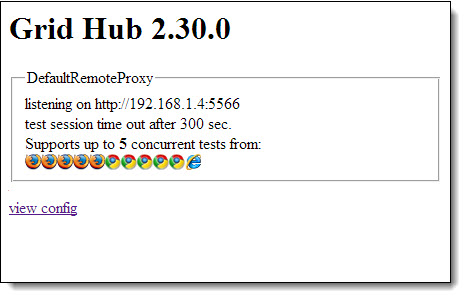


* When you press Enter, your command prompt should be similar to the image below.



**Step 6**

Go to the Selenium Grid web interface and refresh the page. You should see something like this.



At this point, you have already configured a simple grid. You are now ready to run a test remotely on Machine B.

## Designing Test Scripts That Can Run on the Grid

To design test scripts that will run on the grid, we need to use **DesiredCapabilites** and the **RemoteWebDriver** objects.

* **DesiredCapabilites** is used to set the type of **browser** and **OS** that we will automate
* **RemoteWebDriver** is used to set which node (or machine) that our test will run against.

To use the **DesiredCapabilites** object, you must first import this package

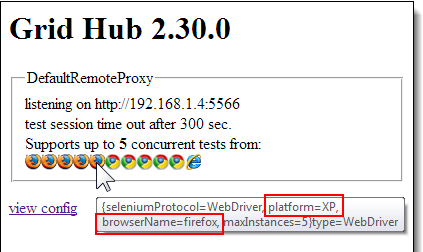
http://newguru99.revolutionventur.netdna-cdn.com/images/import_desired_capabilites.png

To use the **RemoteWebDriver** object, you must import these packages.

http://newguru99.revolutionventur.netdna-cdn.com/images/import_RemoteWebDriver.png

### Using the DesiredCapabilites Object

Go to the Grid’s web interface and hover on an image of the browser that you want to automate. Take note of the **platform** and the **browserName** shown by the tooltip.



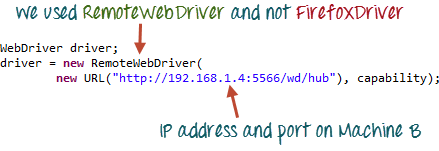
In this case, the platform is “XP” and the browserName is “firefox”.

We will use the platform and the browserName in our WebDriver as shown below (of course you need to import the necessary packages first).

http://newguru99.revolutionventur.netdna-cdn.com/images/DesiredCapabilities_code.png

### Using the RemoteWebDriver Object

Import the necessary packages for RemoteWebDriver and then pass the DesiredCapabilities object that we created above as a parameter for the RemoteWebDriver object.



## Running a Sample Test Case on the Grid

Below is a simple WebDriver TestNG code that you can create in Eclipse on Machine A. Once you run it, automation will be performed on Machine B.



 The test should pass.

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/result_1.png | http://newguru99.revolutionventur.netdna-cdn.com/images/result_2.png |

### Summary

* Selenium Grid is used to run multiple tests simultaneously in different browsers and platforms.
* Grid uses the hub-node concept.
* The hub is the central point wherein you load your tests.
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* To install Selenium Grid, you only need to download the Selenium Server jar file – the same file used in running Selenium RC tests.
* There are 2 ways to verify if the hub is running: one was through the command prompt, and the other was through a browser
* To run test scripts on the Grid, you should use the DesiredCapabilities and the RemoteWebDriver objects.
* DesiredCapabilites is used to set the type of browser and OS that we will automate
* RemoteWebDriver is used to set which node (or machine) that our test will run against

