Day 1: Introduction, Installation and Overview

# Introduction

Selenium is a powerful testing framework for web browser automation. It provides a robust, expansive API for writing functional, cross-browser tests through the usage of the WebDriver interface. Selenium’s ability to write abstract, reusable code for browser automation establishes it as an invaluable tool for functional browser test automation. This training course will provide the reader with a strong foundation in Selenium’s core concepts.

Development with Selenium can occur in multiple programming languages: Java, Python, C#, Ruby, and Javascript. This course will focus on development using the **Java programming language**.

# Installation and Setup

### Download Java SDK (JDK)

Ensure that your machine contains the latest JDK of your choosing (preferably for Java 7 or 8). To check if Java is installed on your machine, open the command prompt and type:

java –version

If Java is not installed, navigate to <http://www.oracle.com/technetwork/java/javase/downloads/index.html> and install the latest JDK for Java 7 or 8. This training course assumes you are using **Java 7**. You may need to include Java on your Windows system path. For instructions, see <http://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_jdk_install.html#BABGDJFH> for guidance.

### Download Eclipse & Create a New Project

**Eclipse** is a free, widely used integrated development environment (IDE) primarily used for developing Java-based applications. While Selenium automation can be developed in almost any editor of your choice, this course will focus on development using Eclipse.

1. Navigate to <https://www.eclipse.org/downloads/> and download the latest version of the *Eclipse IDE for Java Developers*. As of the current revision of this document, the latest version is **Luna**.
2. Extract the downloaded zip file to a location of your choosing, and then create a shortcut on your desktop to the *eclipse.exe* file contained within the main Eclipse directory.
3. Create a new, blank Java project in Eclipse by clicking *File > New > New Java Project*.

### Download Selenium & Include in the Project

#### Download & Include Language Bindings

To include Selenium in your project, you need download the latest Selenium language bindings for Java.

1. Navigate to <http://www.seleniumhq.org/download/> and download the latest *Selenium Client & WebDriver Language Bindings* for Java. As of the current revision of this document, the latest version is **2.45.0**.
2. Find the file path to the project you created in the previous section and extract the recently downloaded zip file to this location. As of the current revision of this document, you should see a folder called *selenium-2.45.0* appear in your project’s directory structure.
3. In Eclipse, you should see this folder appear under the current project’s node. If this folder does not appear, right-click on the project name and select *Refresh* to allow Eclipse to update the project’s contents.
4. Right-click the project name and select *Build Path > Configure Build Path…*
5. Click the *Libraries* tab and select *Add JARs…*
6. Under the folder *selenium-2.45.*0, add the *selenium-java-2.45.0.jar* file to the build path.
7. Additionally, add all the jar folders in the *libs* subdirectory to the build path. Your build path should now look like this:
8. A node called *Referenced Libraries* should now appear under your project containing all the JARs that you added to the build path.

Alternatively, if you are familiar with Maven, you can add the following entry to your *dependencies* in your project’s *pom.xml* file:

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>2.45.0</version>

</dependency>

This training course will not cover how to install/configure Maven. Including Selenium via Maven will be left as an optional exercise for the reader.

#### (Optional) Download IE and Chrome WebDriver servers

Automation in Selenium is driven by the aptly named WebDriver interface which communicates with a **browser driver server** in order to send commands to the browser. By default, Selenium includes a server for interacting with Firefox. In order to interact with other browsers, such as IE or Chrome, the appropriate driver servers need to be downloaded and included in the project.

**For IE:**

1. Navigate to <http://docs.seleniumhq.org/download/> and download the latest *Internet Explorer Driver Server*.
2. Create another folder in your project on the file system called *web\_drivers*.
3. Extract the zip file you just downloaded and place the *IEDriverServer.exe* file in the *web\_drivers* folder.

**For Chrome:**

1. Navigate to <https://sites.google.com/a/chromium.org/chromedriver/downloads> and download the latest *ChromeDriver*.
2. Place the *chromedriver.exe* file in the *web\_drivers* folder, as with the IE driver.

# Overview of Testing in Selenium

The focus here is on developing reusable, maintainable code base. In general, the coder needs to know (1) which pages/sections need automated interaction, (2) how do these pages interact with each other, and (3) how dynamic are the elements automation needs to interact with.

For consideration (1), determine the components of the page(s) that require interaction in order to complete test cases. This will involve a manual overview of the page in section, and any test case documentation available. Often, web content reuses certain elements across many pages; a good example of this is interaction with a navigation menu. If the test cases require navigation through a common menu, for example, then factoring out this menu as a separate piece of code is a good idea.

For consideration (2), it is expected that certain parts of a web application will move the browser to a different page. Alternatively, some web applications interact with a single page, reactively updating the onscreen content based on user interaction. Understanding how the content changes dependent on previous UI actions allows for the planning of better code. For example, if a page reloads its content after the submission of a form, it is likely that references to onscreen objects will become stale and require updating.

For consideration (3), much web technology is driven by dynamically generated content. From fancy UI animations, to AJAX calls in the background, the content presented can and will probably change. Before digging into any automation, the coder should perform an analysis to determine which portions of the web application are prone to change.

# Introduction to Locators

Interaction with elements on screen in Selenium is a two-step process: (1) locating the element on the page, and (2) interacting with the located element. Interactions could vary from actual UI actions such as clicking on the element to discovering information about the element. The first step in this process can be accomplished through a concept in Selenium called **locators**.

Locators are objects that contain information to, unsurprisingly, locate elements on a page. Locators provide Selenium with the context necessary to search the page for an element that you wish to interact with in some manner.

Selenium provides the following locators: Class, CSS, Link Text, Partial Link Text, Tag Name, Name, ID, and XPath. While each locator has its appropriate use cases, this course will start with the most basic and frequently used: ID, Name, and XPath.

This section of the course will not delve into the nitty-gritty details on the usage and appropriate usage scenarios for each of these locators. For now, the reader should just be aware that there are multiple options available to them when the problem of locating elements on a page presents itself.

## Finding locators

How do we determine what the value for a particular locator should be? If I am searching for a particular search box on a page, how do I know which XPath to use? Which ID? Etc. Most modern browsers provide the functionality to inspect elements and retrieve a basic view of this information.

When viewing the page, right-click on the element (or near the element) you want to interact with. This should open another window in the web browser pertaining to its version of developer tools. You should see an HTML representation of the current document, with the element you inspected highlighted. Right-clicking on this piece of HTML should produce a popup menu that allows you to extract XPath/CSS selector/etc. You can then paste this information into a text-editing program to view the data you just extracted.

Alternatively, pressing F12 typically launches a browser’s version of developer tools, where you can navigate for a particular element manually and then extract its locator.

## Finding quality elements

The general principle is to find elements that are descriptive, stable, and unique.

Web content is inherently prone to change; web admins frequently add new content to web pages, whether in the form of bug fixes, enhancements, etc. Additionally, certain elements may be subject to change due to the dynamic nature of pages on the modern Web. Elements can have their *class* and other attributes change over the lifetime of a page visit. Thus, we want to target elements that are unlikely to change; this minimizes the maintenance cost for your automation.

Descriptive and uniqueness in determining elements also plays a major role in selecting quality elements; web content frequently reuses identifiers for styling/programming purposes and it is possible to accidentally select too many elements, or the wrong element altogether. Concise, but unique locators will provide the most accurate way to grab the elements you need for your code to function. Prime candidates for this are the **id** and **class** attributes. The **id**attribute is particularly attractive because web standards demand that an id be unique on the page in which it resides. Nevertheless, not all elements will have an id associated with them; nor will id’s necessarily be stable – many third-party frameworks generate id’s on the fly with dynamic identifiers that may change from release to release.

## Document Object Model (DOM)

The Document Object Model (DOM) represents web pages as a tree of elements. In Selenium, a basic understanding of the DOM will help us in selecting our locators. Due to dynamic content, elements in the DOM may not appear in the original markup. This allows us to write code that conditionally selects elements based on event-based preconditions.

# Additional Resources

### Selenium Documentation

<http://selenium.googlecode.com/svn@7068/trunk/docs/api/java/index.html>

### More on DOM

<https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction>

### More on Locators

<http://www.softwaretestinghelp.com/using-selenium-xpath-and-other-locators-selenium-tutorial-5/>

<https://blog.mozilla.org/webqa/2013/09/26/writing-reliable-locators-for-selenium-and-webdriver-tests/>

### Introduction to CSS selectors

<http://www.sitepoint.com/web-foundations/introduction-css-selectors/>

### Introduction to XPath

<http://www.data2type.de/en/xml-xslt-xslfo/xpath/xpath-introduction/>