Selenium 2.0 WebDriver

Selenium WebDriver was developed to better support dynamic web pages where elements of a page may change without having to reload the page. WebDriver’s goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

Setting Up a Selenium-WebDriver Project in Java

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

First, create a folder to contain your Selenium project files. Then, to use Maven, you need a pom.xml file. This can be created with a text editor. Your pom.xml file will look something like this. Create this file in the folder you created for your project.

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>MySel20Proj</groupId>

<artifactId>MySel20Proj</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

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

<artifactId>selenium-server</artifactId>

<version>3.0.1</version>

</dependency>

</dependencies>

</project>

Be sure you specify the most current version. Check the Maven download page for the current release and edit the above dependency accordingly.

Now, from a command-line, CD into the project directory and run maven as follows:

mvn clean install

This will download Selenium and all its dependencies and will add them to the project.

Finally, import the project into your preferred development environment.

Selenium-WebDriver API Commands and Operations  
Fetching a Page

Ex.: driver.get(“[http://www.google.com](http://www.google.com/)”);

Locating of Elements (WebElements)

By ID  
This is the most efficient and preferred way to locate an element. Common pitfalls that UI developers make is having non-unique IDs on a page or auto-generating the ID, both should be avoided. A class on an html element is more appropriate than an auto-generated ID.

Example of how to find an element that looks like this:

<div id=“coolestWidgetEver”>...</div>

WebElement element = driver.findElement(By.id(“coolestWidgetEver”));

By Class Name

“Class” in this case refers to the attribute on the DOM element. Often in practical use there are many DOM elements with the same class name, thus finding multiple elements becomes the more practical option over finding the first element.

Example of how to find an element that looks like this:

<div class=“vegetable”><span>Carrot</span></div><div class=“vegetable”><span>Celery</span></div>

List<WebElement> veggies = driver.findElements(By.className(“vegetable”));

By Tag Name

The DOM Tag Name of the element.

Example of how to find an element that looks like this:

<iframe src=“...”></iframe>

WebElement frame = driver.findElement(By.tagName(“iframe”));

By Name

Find the input element with matching name attribute.

Example of how to find an element that looks like this:

<input name=“tofu” type=“text”/>

WebElement tofu = driver.findElement(By.name(“tofu”));

By Link Text

Find the link element with matching visible text.

Example of how to find an element that looks like this:

<a href=“http://www.google.com/search?q=tofu”>tofu</a>>

WebElement tofu = driver.findElement(By.linkText(“tofu”));

By Partial Link Text

Find the link element with partial matching visible text.

Example of how to find an element that looks like this:

<a href=“http://www.google.com/search?q=tofu”>search for tofu</a>>

WebElement tofu = driver.findElement(By.partialLinkText(“tofu”));

By CSS

Like the name implies it is a locator strategy by css. Native browser support is used by default, so please refer to w3c css selectors for a list of generally available css selectors. If a browser does not have native support for css queries, then Sizzle is used.

Beware that not all browsers were created equal, some css the might work in one version may not work in another.

Example of how to find the tofu below:

<div id=“food”><span class=“soy”>soybeans</span><span class=“soy processed”>tofu</span></div>

WebElement tofu = driver.findElement(By.cssSelector(“#food span.soy.processed”));