Navigation

Selenium Documentation »

previous | next

Table Of Contents

Programming Language Preference: java | csharp | python | ruby | php | perl | javascript

Selenium WebDriver

NOTE: We're currently working on documenting these sections. We believe the information here is accurate, however be aware we are also still working on this chapter. Additional information will be provided as we go which should make this chapter more solid.

Introducing WebDriver

The primary new feature in Selenium 2.0 is the integration of the WebDriver API. WebDriver is designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API. Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

How Does WebDriver 'Drive' the Browser Compared to Selenium-RC?

Selenium-WebDriver makes direct calls to the browser using each browser's native support for automation. How these direct calls are made, and the features they support depends on the browser you are using. Information on each 'browser

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Previous topic

Selenium-IDE

Next topic

WebDriver: Advanced Usage

Donate to Selenium

with PayPal

Donate

driver' is provided later in this chapter.

For those familiar with Selenium-RC, this is quite different from what you are used to. Selenium-RC worked the same way for each supported browser. It 'injected' javascript functions into the browser when the browser was loaded and then used its javascript to drive the AUT within the browser. WebDriver does not use this technique. Again, it drives the browser directly using the browser's built in support for automation.

WebDriver and the Selenium-Server

You may, or may not, need the Selenium Server, depending on how you intend to use Selenium-WebDriver. If you will be only using the WebDriver API you do not need the Selenium-Server. If your browser and tests will all run on the same machine, and your tests only use the WebDriver API, then you do not need to run the Selenium-Server; WebDriver will run the browser directly.

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

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

Setting Up a Selenium-WebDriver Project

To install Selenium means to set up a project in a development so you can write a program using Selenium. How you do this depends on your programming language and your development environment.

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

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

through sponsorship

You can sponsor the Selenium project if you'd like some public recognition of your generous contribution.

Selenium Sponsors

See who supports the Selenium project.

BrowserStack

sauce LABS

experitest Selenium for Mobile

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 your preferred IDE, IntelliJ IDEA or 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. We won't teach the details of pom.xml files or for using Maven since there are already excellent references on this. 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
        <modelVersion>4.0.0</modelVersion>
        <groupId>MySel20Proj</groupId>
        <artifactId>MySel20Proj</artifactId>
        <version>1.0</version>
        <dependencies>
            <dependency>
                <groupId>org.seleniumhq.selenium</groupId>
                <artifactId>selenium-java</artifactId>
                <version>2.47.1</version>
            </dependency>
        </dependencies>
</project>
```

Be sure you specify the most current version. At the time of writing, the version listed above was the most current, however there were frequent releases immediately after the release of Selenium 2.0. 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
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

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

Finally, import the project into your preferred development environment. For those not familiar with this, we've provided an appendix which shows this.

Importing a maven project into IntelliJ IDEA. Importing a maven project into Eclipse.

C#

As of Selenium 2.2.0, the C# bindings are distributed as a set of signed dlls along with other dependency dlls. Prior to 2.2.0, all Selenium dll's were unsigned. To include Selenium in your project, simply download the latest selenium-dotnet zip file from http://selenium-release.storage.googleapis.com/index.html. If you are using Windows Vista or above, you should unblock the zip file before unzipping it: Right click on the zip file, click "Properties", click "Unblock" and click "OK".

Unzip the contents of the zip file, and add a reference to each of the unzipped dlls to your project in Visual Studio (or your IDE of choice).

Official NuGet Packages: RC WebDriver WebDriverBackedSelenium Support

Python

If you are using Python for test automation then you probably are already familiar with developing in Python. To add Selenium to your Python environment run the following command from a command-line.

```
pip install selenium
```

Pip requires pip to be installed, pip also has a dependency on setuptools.

Teaching Python development itself is beyond the scope of this document, however there are many resources on Python and likely developers in your organization can help you get up to speed.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Ruby

If you are using Ruby for test automation then you probably are already familiar with developing in Ruby. To add Selenium to your Ruby environment run the following command from a command-line.

```
gem install selenium-webdriver
```

Teaching Ruby development itself is beyond the scope of this document, however there are many resources on Ruby and likely developers in your organization can help you get up to speed.

Perl

Perl bindings are provided by a third party, please refer to any of their documentation on how to install / get started. There is one known Perl binding as of this writing.

PHP

PHP bindings are provided by a third party, please refer to any of their documentation on how to install / get started. There are three known bindings at this time: By Chibimagic By Lukasz Kolczynski and By the Facebook

Migrating from Selenium 1.0

For those who already have test suites written using Selenium 1.0, we have provided tips on how to migrate your existing code to Selenium 2.0. Simon Stewart, the lead developer for Selenium 2.0, has written an article on migrating from Selenium 1.0. We've included this as an appendix.

Migrating From Selenium RC to Selenium WebDriver

Introducing the Selenium-WebDriver API by Example

WebDriver is a tool for automating web application testing, and in particular to verify that they work as expected. It aims to provide a friendly API that's easy to

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

explore and understand, easier to use than the Selenium-RC (1.0) API, which will help to make your tests easier to read and maintain. It's not tied to any particular test framework, so it can be used equally well in a unit testing or from a plain old "main" method. This section introduces WebDriver's API and helps get you started becoming familiar with it. Start by setting up a WebDriver project if you haven't already. This was described in the previous section, Setting Up a Selenium-WebDriver Project.

Once your project is set up, you can see that WebDriver acts just as any normal library: it is entirely self-contained, and you usually don't need to remember to start any additional processes or run any installers before using it, as opposed to the proxy server with Selenium-RC.

Note: additional steps are required to use ChromeDriver, Opera Driver, Android Driver and iOS Driver

You're now ready to write some code. An easy way to get started is this example, which searches for the term "Cheese" on Google and then outputs the result page's title to the console.


```java
package org.openqa.selenium.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Selenium2Example  {
    public static void main(String[] args) {
        // Create a new instance of the Firefox driver
        // Notice that the remainder of the code relies on the interface,
        // not the implementation.
        WebDriver driver = new FirefoxDriver();
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
        // And now use this to visit Google
        driver.get("http://www.google.com");
        // Alternatively the same thing can be done like this
        // driver.navigate().to("http://www.google.com");

        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));

        // Enter something to search for
        element.sendKeys("Cheese!");

        // Now submit the form. WebDriver will find the form for us from the e
        element.submit();

        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());

        // Google's search is rendered dynamically with JavaScript.
        // Wait for the page to load, timeout after 10 seconds
        (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>()
            public Boolean apply(WebDriver d) {
                return d.getTitle().toLowerCase().startsWith("cheese!");
            }
        });

        // Should see: "cheese! - Google Search"
        System.out.println("Page title is: " + driver.getTitle());

        //Close the browser
        driver.quit();
    }
}
```

csharp python ruby javascript perl

In upcoming sections, you will learn more about how to use WebDriver for things such as navigating forward and backward in your browser's history, and how to test web sites that use frames and windows. We also provide a more thorough

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

discussions and examples.

Selenium-WebDriver API Commands and Operations

Fetching a Page

The first thing you're likely to want to do with WebDriver is navigate to a page. The normal way to do this is by calling "get":

java

```
driver.get("http://www.google.com");
```

csharp ruby python perl

Dependent on several factors, including the OS/Browser combination, WebDriver may or may not wait for the page to load. In some circumstances, WebDriver may return control before the page has finished, or even started, loading. To ensure robustness, you need to wait for the element(s) to exist in the page using Explicit and Implicit Waits.

Locating UI Elements (WebElements)

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

The "Find" methods take a locator or query object called "By". "By" strategies are listed below.

By ID

This is the most efficient and preferred way to locate an element. Common pitfalls that UI developers make is having non-unique id's on a page or auto-generating the id, both should be avoided. A class on an html element is

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

more appropriate than an auto-generated id.

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

```
<div id="coolestWidgetEvah">...</div>
```

java

```
WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
```

csharp ruby python perl

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="cheese"><span>Cheddar</span></div><div class="cheese"><span>G
```

java

```
List<WebElement> cheeses = driver.findElements(By.className("cheese"));
```

csharp ruby python perl

By Tag Name

The DOM Tag Name of the element.

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

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
<iframe src="..."></iframe>
```

java

```
WebElement frame = driver.findElement(By.tagName("iframe"));
```

csharp ruby python perl

By Name

Find the input element with matching name attribute.

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

```
<input name="cheese" type="text"/>
```

java

```
WebElement cheese = driver.findElement(By.name("cheese"));
```

csharp ruby python perl

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=cheese">cheese</a>>
```

java

```
WebElement cheese = driver.findElement(By.linkText("cheese"));
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

csharp ruby python perl

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=cheese">search for cheese</a>>
```

java

```
WebElement cheese = driver.findElement(By.partialLinkText("cheese"));
```

csharp ruby python perl

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. IE 6,7 and FF3.0 currently use Sizzle as the css query engine.

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

Example of to find the cheese below:

```
<div id="food"><span class="dairy">milk</span><span class="dairy aged">ch
```

java

```
WebElement cheese = driver.findElement(By.cssSelector("#food span.dairy.a
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

csharp ruby python perl

By XPATH

At a high level, WebDriver uses a browser's native XPath capabilities wherever possible. On those browsers that don't have native XPath support, we have provided our own implementation. This can lead to some unexpected behaviour unless you are aware of the differences in the various xpath engines.

Driver	Tag and Attribute Name	Attribute Values	Native XPath Support
HtmlUnit Driver	Lower-cased	As they appear in the HTML	Yes
Internet Explorer Driver	Lower-cased	As they appear in the HTML	No
Firefox Driver	Case insensitive	As they appear in the HTML	Yes

This is a little abstract, so for the following piece of HTML:

```
<input type="text" name="example" />
<INPUT type="text" name="other" />
```

java

```
List<WebElement> inputs = driver.findElements(By.xpath("//input"));
```

csharp ruby python perl

The following number of matches will be found

XPath expression	HtmlUnit Driver	Firefox Driver	Internet Explorer Driver

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

//input	1 (“example”)	2	2
//INPUT	0	2	0

Sometimes HTML elements do not need attributes to be explicitly declared because they will default to known values. For example, the “input” tag does not require the “type” attribute because it defaults to “text”. The rule of thumb when using xpath in WebDriver is that you **should not** expect to be able to match against these implicit attributes.

Using JavaScript

You can execute arbitrary javascript to find an element and as long as you return a DOM Element, it will be automatically converted to a WebElement object.

Simple example on a page that has jQuery loaded:

java

```
WebElement element = (WebElement) ((JavascriptExecutor)driver).executeScr
```

csharp ruby python perl

Finding all the input elements to the every label on a page:

java

```
List<WebElement> labels = driver.findElements(By.tagName("label"));
List<WebElement> inputs = (List<WebElement>) ((JavascriptExecutor)driver)
    "var labels = arguments[0], inputs = []; for (var i=0; i < labels.len
    "inputs.push(document.getElementById(labels[i].getAttribute('for')));
```

csharp ruby python perl

User Input - Filling In Forms

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

We've already seen how to enter text into a textarea or text field, but what about the other elements? You can "toggle" the state of checkboxes, and you can use "click" to set something like an OPTION tag selected. Dealing with SELECT tags isn't too bad:

java

```
WebElement select = driver.findElement(By.tagName("select"));
List<WebElement> allOptions = select.findElements(By.tagName("option"));
for (WebElement option : allOptions) {
    System.out.println(String.format("Value is: %s", option.getAttribute("va
    option.click();
}
```

csharp ruby python perl

This will find the first "SELECT" element on the page, and cycle through each of its OPTIONs in turn, printing out their values, and selecting each in turn. As you will notice, this isn't the most efficient way of dealing with SELECT elements. WebDriver's support classes include one called "Select", which provides useful methods for interacting with these.

java

```
Select select = new Select(driver.findElement(By.tagName("select")));
select.deselectAll();
select.selectByVisibleText("Edam");
```

csharp ruby python

This will deselect all OPTIONs from the first SELECT on the page, and then select the OPTION with the displayed text of "Edam".

Once you've finished filling out the form, you probably want to submit it. One way to do this would be to find the "submit" button and click it:

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

java

```
driver.findElement(By.id("submit")).click();
```

ruby python perl

Alternatively, WebDriver has the convenience method "submit" on every element. If you call this on an element within a form, WebDriver will walk up the DOM until it finds the enclosing form and then calls submit on that. If the element isn't in a form, then the `NoSuchElementException` will be thrown:

java

```
element.submit();
```

ruby python perl

Moving Between Windows and Frames

Some web applications have many frames or multiple windows. WebDriver supports moving between named windows using the "switchTo" method:

java

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

ruby python perl

All calls to `driver` will now be interpreted as being directed to the particular window. But how do you know the window's name? Take a look at the javascript or link that opened it:

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

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Alternatively, you can pass a "window handle" to the "switchTo().window()" method. Knowing this, it's possible to iterate over every open window like so:

java

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

ruby python perl

You can also switch from frame to frame (or into iframes):

java

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

ruby python perl

Popup Dialogs

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

java

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

csharp ruby python perl

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

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

RubyDocs for more information.

Navigation: History and Location

Earlier, we covered navigating to a page using the "get" command (`driver.get("http://www.example.com")`) As you've seen, WebDriver has a number of smaller, task-focused interfaces, and navigation is a useful task. Because loading a page is such a fundamental requirement, the method to do this lives on the main WebDriver interface, but it's simply a synonym to:

java

```
driver.navigate().to("http://www.example.com");
```

ruby python perl

To reiterate: "`navigate().to()`" and "`get()`" do exactly the same thing. One's just a lot easier to type than the other!

The "navigate" interface also exposes the ability to move backwards and forwards in your browser's history:

java

```
driver.navigate().forward();
driver.navigate().back();
```

ruby python

Please be aware that this functionality depends entirely on the underlying browser. It's just possible that something unexpected may happen when you call these methods if you're used to the behaviour of one browser over another.

Cookies

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Before we leave these next steps, you may be interested in understanding how to use cookies. First of all, you need to be on the domain that the cookie will be valid for. If you are trying to preset cookies before you start interacting with a site and your homepage is large / takes a while to load an alternative is to find a smaller page on the site, typically the 404 page is small (http://example.com/some404page)

java

```
// Go to the correct domain
driver.get("http://www.example.com");

// Now set the cookie. This one's valid for the entire domain
Cookie cookie = new Cookie("key", "value");
driver.manage().addCookie(cookie);

// And now output all the available cookies for the current URL
Set<Cookie> allCookies = driver.manage().getCookies();
for (Cookie loadedCookie : allCookies) {
    System.out.println(String.format("%s -> %s", loadedCookie.getName(), lo
}

// You can delete cookies in 3 ways
// By name
driver.manage().deleteCookieNamed("CookieName");
// By Cookie
driver.manage().deleteCookie(loadedCookie);
// Or all of them
driver.manage().deleteAllCookies();
```

python ruby perl

Changing the User Agent

This is easy with the Firefox Driver:

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

java

```
FirefoxProfile profile = new FirefoxProfile();
profile.addAdditionalPreference("general.useragent.override", "some UA stri
WebDriver driver = new FirefoxDriver(profile);
```

ruby python perl

Drag And Drop

Here's an example of using the Actions class to perform a drag and drop. Native events are required to be enabled.

java

```
WebElement element = driver.findElement(By.name("source"));
WebElement target = driver.findElement(By.name("target"));

(new Actions(driver)).dragAndDrop(element, target).perform();
```

ruby python perl

Driver Specifics and Tradeoffs

Selenium-WebDriver's Drivers

WebDriver is the name of the key interface against which tests should be written, but there are several implementations. These include:

HtmlUnit Driver

This is currently the fastest and most lightweight implementation of WebDriver. As the name suggests, this is based on HtmlUnit. HtmlUnit is a java based implementation of a WebBrowser without a GUI. For any language binding

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

(other than java) the Selenium Server is required to use this driver.

Usage

java

```
WebDriver driver = new HtmlUnitDriver();
```

csharp python ruby perl

Pros

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

Cons

- Emulates other browsers' JavaScript behaviour (see below)

JavaScript in the HtmlUnit Driver

None of the popular browsers uses the JavaScript engine used by HtmlUnit (Rhino). If you test JavaScript using HtmlUnit the results may differ significantly from those browsers.

When we say "JavaScript" we actually mean "JavaScript and the DOM". Although the DOM is defined by the W3C each browser has its own quirks and differences in their implementation of the DOM and in how JavaScript interacts with it. HtmlUnit has an impressively complete implementation of the DOM and has good support for using JavaScript, but it is no different from any other browser: it has its own quirks and differences from both the W3C standard and the DOM implementations of the major browsers, despite its ability to mimic other browsers.

With WebDriver, we had to make a choice; do we enable HtmlUnit's

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

JavaScript capabilities and run the risk of teams running into problems that only manifest themselves there, or do we leave JavaScript disabled, knowing that there are more and more sites that rely on JavaScript? We took the conservative approach, and by default have disabled support when we use HtmlUnit. With each release of both WebDriver and HtmlUnit, we reassess this decision: we hope to enable JavaScript by default on the HtmlUnit at some point.

Enabling JavaScript

If you can't wait, enabling JavaScript support is very easy:

java

```
HtmlUnitDriver driver = new HtmlUnitDriver(true);
```

csharp ruby python perl

This will cause the HtmlUnit Driver to emulate Firefox 3.6's JavaScript handling by default.

Firefox Driver

Controls the Firefox browser using a Firefox plugin. The Firefox Profile that is used is stripped down from what is installed on the machine to only include the Selenium WebDriver.xpi (plugin). A few settings are also changed by default (see the source to see which ones) Firefox Driver is capable of being run and is tested on Windows, Mac, Linux. Currently on versions 3.6, 10, latest - 1, latest

Usage

java

```
WebDriver driver = new FirefoxDriver();
```

csharp python ruby perl

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Pros

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

Cons

- Slower than the HtmlUnit Driver

Modifying the Firefox Profile

Suppose that you wanted to modify the user agent string (as above), but you've got a tricked out Firefox profile that contains dozens of useful extensions. There are two ways to obtain this profile. Assuming that the profile has been created using Firefox's profile manager (`firefox -ProfileManager`):

java

```
ProfilesIni allProfiles = new ProfilesIni();
FirefoxProfile profile = allProfiles.getProfile("WebDriver");
profile.setPreferences("foo.bar", 23);
WebDriver driver = new FirefoxDriver(profile);
```

Alternatively, if the profile isn't already registered with Firefox:

java

```
File profileDir = new File("path/to/top/level/of/profile");
FirefoxProfile profile = new FirefoxProfile(profileDir);
profile.addAdditionalPreferences(extraPrefs);
WebDriver driver = new FirefoxDriver(profile);
```

As we develop features in the Firefox Driver, we expose the ability to use them. For example, until we feel native events are stable on Firefox for Linux, they are disabled by default. To enable them:

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

java

```
FirefoxProfile profile = new FirefoxProfile();
profile.setEnableNativeEvents(true);
WebDriver driver = new FirefoxDriver(profile);
```

python ruby

Info

See the Firefox section in the wiki page for the most up to date info.

Internet Explorer Driver

This driver is controlled by a .dll and is thus only available on Windows OS. Each Selenium release has its core functionality tested against versions 6, 7 and 8 on XP, and 9 on Windows7.

Usage

java

```
WebDriver driver = new InternetExplorerDriver();
```

csharp python ruby perl

Pros

- Runs in a real browser and supports JavaScript with all the quirks your end users see.

Cons

- Obviously the Internet Explorer Driver will only work on Windows!
- Comparatively slow (though still pretty snappy)

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

- XPath is not natively supported in most versions. Sizzle is injected automatically which is significantly slower than other browsers and slower when comparing to CSS selectors in the same browser.
- CSS is not natively supported in versions 6 and 7. Sizzle is injected instead.
- CSS selectors in IE 8 and 9 are native, but those browsers don't fully support CSS3

Info

See the Internet Explorer section of the wiki page for the most up to date info. Please take special note of the Required Configuration section.

ChromeDriver

ChromeDriver is maintained / supported by the Chromium project iteslf. WebDriver works with Chrome through the chromedriver binary (found on the chromium project's download page). You need to have both chromedriver and a version of chrome browser installed. chromedriver needs to be placed somewhere on your system's path in order for WebDriver to automatically discover it. The Chrome browser itself is discovered by chromedriver in the default installation path. These both can be overridden by environment variables. Please refer to the wiki for more information.

Usage

java

```
WebDriver driver = new ChromeDriver();
```

csharp python ruby perl

Pros

- Runs in a real browser and supports JavaScript
- Because Chrome is a Webkit-based browser, the ChromeDriver may

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

allow you to verify that your site works in Safari. Note that since Chrome uses its own V8 JavaScript engine rather than Safari's Nitro engine, JavaScript execution may differ.

Cons

- Slower than the HtmlUnit Driver

Info

See our wiki for the most up to date info. More info can also be found on the downloads page

Getting running with ChromeDriver

Download the ChromeDriver executable and follow the other instructions on the wiki page

Opera Driver

See the Opera Driver wiki article in the Selenium Wiki for information on using the Opera Driver.

iOS Driver

See either the ios-driver or appium projects.

Android Driver

See the Selendroid project

Alternative Back-Ends: Mixing WebDriver and RC Technologies

WebDriver-Backed Selenium-RC

The Java version of WebDriver provides an implementation of the Selenium-RC API. These means that you can use the underlying WebDriver technology using the Selenium-RC API. This is primarily provided for backwards compatibility. It

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

allows those who have existing test suites using the Selenium-RC API to use WebDriver under the covers. It's provided to help ease the migration path to Selenium-WebDriver. Also, this allows one to use both APIs, side-by-side, in the same test code.

Selenium-WebDriver is used like this:


```
// You may use any WebDriver implementation. Firefox is used here as an exa
WebDriver driver = new FirefoxDriver();

// A "base url", used by selenium to resolve relative URLs
 String baseUrl = "http://www.google.com";

// Create the Selenium implementation
Selenium selenium = new WebDriverBackedSelenium(driver, baseUrl);

// Perform actions with selenium

selenium.open("http://www.google.com");
selenium.type("name=q", "cheese");
selenium.click("name=btnG");

// Get the underlying WebDriver implementation back. This will refer to the
// same WebDriver instance as the "driver" variable above.
WebDriver driverInstance = ((WebDriverBackedSelenium) selenium).getWrappedD

//Finally, close the browser. Call stop on the WebDriverBackedSelenium inst
//instead of calling driver.quit(). Otherwise, the JVM will continue runnin
//the browser has been closed.
selenium.stop();
```

Pros

- Allows for the WebDriver and Selenium APIs to live side-by-side
- Provides a simple mechanism for a managed migration from the

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Selenium RC API to WebDriver's
- Does not require the standalone Selenium RC server to be run

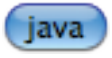
Cons

- Does not implement every method
- More advanced Selenium usage (using "browserbot" or other built-in JavaScript methods from Selenium Core) may not work
- Some methods may be slower due to underlying implementation differences

Backing WebDriver with Selenium

WebDriver doesn't support as many browsers as Selenium RC does, so in order to provide that support while still using the WebDriver API, you can make use of the `SeleneseCommandExecutor`

Safari is supported in this way with the following code (be sure to disable pop-up blocking):




```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setBrowserName("safari");
CommandExecutor executor = new SeleneseCommandExecutor(new URL("http://loca
WebDriver driver = new RemoteWebDriver(executor, capabilities);
```

There are currently some major limitations with this approach, notably that findElements doesn't work as expected. Also, because we're using Selenium Core for the heavy lifting of driving the browser, you are limited by the JavaScript sandbox.

Running Standalone Selenium Server for use with RemoteDrivers

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

From Selenium's Download page download selenium-server-standalone-<version>.jar and optionally IEDriverServer. If you plan to work with Chrome, download it from Google Code.

Unpack IEDriverServer and/or chromedriver and put them in a directory which is on the $PATH / %PATH% - the Selenium Server should then be able to handle requests for IE / Chrome without additional modifications.

Start the server on the command line with

```
java -jar <path_to>/selenium-server-standalone-<version>.jar
```

If you want to use native events functionality, indicate this on the command line with the option

```
-Dwebdriver.enable.native.events=1
```

For other command line options, execute

```
java -jar <path_to>/selenium-server-standalone-<version>.jar -help
```

In order to function properly, the following ports should be allowed incoming TCP connections: 4444, 7054-5 (or twice as many ports as the number of concurrent instances you plan to run). Under Windows, you may need to unblock the applications as well.

Additional Resources

You can find further resources for WebDriver in WebDriver's wiki

Of course, don't hesitate to do an internet search on any Selenium topic, including Selenium-WebDriver's drivers. There are quite a few blogs on Selenium along with numerous posts on various user forums. Additionally the Selenium User's Group is a great resource. http://groups.google.com/group/selenium-users

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Next Steps

This chapter has simply been a high level walkthrough of WebDriver and some of its key capabilities. Once getting familiar with the Selenium-WebDriver API you will then want to learn how to build test suites for maintainability, extensibility, and reduced fragility when features of the AUT frequently change. The approach most Selenium experts are now recommending is to design your test code using the Page Object Design Pattern along with possibly a Page Factory. Selenium-WebDriver provides support for this by supplying a PageFactory class in Java and C#. This is presented, along with other advanced topics, in the next chapter. Also, for high-level description of this technique, you may want to look at the Test Design Considerations chapter. Both of these chapters present techniques for writing more maintainable tests by making your test code more modular.

Navigation

Selenium Documentation »

previous | next

© Copyright 2008-2012, Selenium Project. Last updated on Aug 17, 2015.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com