Selenium Notes (2012)

Firefox Driver

Firefox driver is included in the selenium-server-stanalone.jar available in the downloads. The driver comes in the form of an xpi (firefox extension) which is added to the firefox profile when you start a new instance of FirefoxDriver.

Pros

- Runs in a real browser and supports Javascript
- Faster than the InternetExplorerDriver

Cons

- Slower than the HtmlUnitDriver

Important System Properties

The following system properties (read using System.getProperty() and set using System.setProperty() in Java code or the "-DpropertyName=value" command line flag) are used by the FirefoxDriver:

Property	What it means
webdriver.accept.untrusted.certs	?
webdriver.assume.untrusted.issuer	?
webdriver.development	**Never use in production** Indicates that we're in development mode.
webdriver.enable.native.events	?
webdriver.firefox.bin	The location of the binary used to control firefox.
webdriver.firefox.port	?
webdriver.firefox.profile	The name of the profile to

	use when starting firefox. This defaults to webdriver creating an anonymous profile
webdriver.firefox.useExisting	**Never use in production** Use a running instance of firefox if one is present
webdriver.log.file	Log file to dump javascript console logging to
webdriver.firefox.logfile	Log file to dump firefox stdout/stderr to
webdriver.reap_profile	Should be "true" if temporary files and profiles should not be deleted

Normally the Firefox binary is assumed to be in the default location for your particular operating system:

OS	**Expected Location of Firefox**
Linux	firefox (found using "which")
Mac	/Applications/Firefox.app/Contents/MacOS/firefox-bin
Windows	%PROGRAMFILES%\Mozilla Firefox\firefox.exe

By default, the Firefox driver creates an anonymous profile

Running with firebug

Download the firebug xpi file from mozilla and start the profile as follows:

```
  File file = new File("firebug-1.8.1.xpi");
  FirefoxProfile firefoxProfile = new FirefoxProfile();
  firefoxProfile.addExtension(file);
  firefoxProfile.setPreference("extensions.firebug.currentVersion", "1.8.1"); // Avoid startup screen

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

-Beta- load fast preference

There is beta feature to make firefox not wait for the full page to load after calling .get or .click. This may cause immediate find's to break, so please be sure to use an implicit or explicit wait too. This is only available for Firefox and not other browsers.

```
  FirefoxProfile fp = new FirefoxProfile();
  fp.setPreference("webdriver.load.strategy", "unstable"); // As of 2.19. from 2.9 - 2.18 use 'fast'
  WebDriver driver = new FirefoxDriver(fp);
```

Chrome Driver

Developed in collaboration with the Chromium team, the ChromeDriver is a standalone server which implements WebDriver's wire protocol.
The ChromeDriver consists of three separate pieces. There is the browser itself ("chrome"), the language bindings provided by the Selenium project ("the driver") and an executable downloaded from the Chromium project which acts as a bridge between "chrome" and the "driver". This executable is called "chromedriver", but we'll try and refer to it as the "server" in this page to reduce confusion.

Requirements

The ChromeDriver controls the browser using Chrome's automation proxy framework.
The server expects you to have Chrome installed in the default location for each system:

OS	Expected Location of Chrome
Linux	/usr/bin/google-chrome1
Mac	/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome
Windows XP	%HOMEPATH%\Local Settings\Application Data\Google\Chrome\Application\chrome.exe
Windows Vista	C:\Users\%USERNAME%\AppData\Local\Google\Chrome\Application\chrome.exe

1 For Linux systems, the ChromeDriver expects /usr/bin/google-chrome to be a symlink to the actual Chrome binary. See also the section on overriding the Chrome binary location .

Getting Started

To get set up, first download the appropriate prebuilt server. Make sure the server can be located on your PATH or specify its location via the webdriver.chrome.driver system property. Finally, all you need to do is create a new ChromeDriver instance:
WebDriver driver = new ChromeDriver();
driver.get("http://www.google.com");

Running the server in a child process

You may notice that the ChromeDriver class is merely a convenience class that starts the server upon creation and shuts it down when you call quit. While the server is light weight, starting and stopping it multiple times will add a noticeable delay to a larger test suite. To compensate for this, you can directly control the life and death of the server using the ChromeDriverService:
import static org.junit.Assert.assertEquals;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.runner.RunWith;

```
import org.junit.runners.BlockJUnit4ClassRunner
import org.openqa.selenium.chrome.ChromeDriverService;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

@RunWith(BlockJUnit4ClassRunner.class)}
public class ChromeTest extends TestCase {

  private static ChromeDriverService service;
  private WebDriver driver;

  @BeforeClass
  public static void createAndStartService() {
    service = new ChromeDriverService.Builder()
        .usingChromeDriverExecutable(new File("path/to/my/chromedriver"))
        .usingAnyFreePort()
        .build();
    service.start();
  }

  @AfterClass
  public static void createAndStopService() {
    service.stop();
  }

  @Before
  public void createDriver() {
    driver = new RemoteWebDriver(service.getUrl(),
        DesiredCapabilities.chrome());
  }

  @After
  public void quitDriver() {
    driver.quit();
  }

  @Test
  public void testGoogleSearch() {
    driver.get("http://www.google.com");
    WebElement searchBox = driver.findElement(By.name("q"));
    searchBox.sendKeys("webdriver");
    searchBox.quit();
    assertEquals("webdriver - Google Search", driver.getTitle());
  }
}
```

Running the server as a standalone process

Since the ChromeDriver implements the wire protocol, it is fully compatible with any RemoteWebDriver client. Simply start up your server, create a client, and away you go:

```
WebDriver driver = new RemoteWebDriver("http://localhost:9515",
DesiredCapabilities.chrome());
driver.get("http://www.google.com");
```

Advanced Usage

Starting Chromium with Specific Flags

The ChromeDriver can be made to start the browser with specific command line flags using the chrome.switches capability key; this key should define a list of command line flags that should be passed to the browser on start-up. For example, to start Chrome as a maximized window:

```
  DesiredCapabilities capabilities = DesiredCapabilities.chrome();
  capabilities.setCapability("chrome.switches", Arrays.asList("--start-maximized"));
  WebDriver driver = new ChromeDriver(capabilities);
```

Similarly, to load an extension when Chrome starts:

```
  DesiredCapabilities capabilities = DesiredCapabilities.chrome();
  capabilities.setCapability("chrome.switches", Arrays.asList("--load-
extension=/path/to/extension/directory"));
  WebDriver driver = new ChromeDriver(capabilities);
```

Or to load with a specific profile (note that the default profile directories can be found here):

```
  DesiredCapabilities capabilities = DesiredCapabilities.chrome();
  capabilities.setCapability("chrome.switches", Arrays.asList("--user-data-
dir=/path/to/profile/directory"));
  WebDriver driver = new ChromeDriver(capabilities);
```

The full list of flags can be found here.

Setting Chrome's proxy configuration

Since ChromeDriver 18.0.995.0, WebDriver's Proxy Desired Capability can be used. If you're using a previous version. The following can be used:

```
  DesiredCapabilities capabilities = DesiredCapabilities.chrome();
  capabilities.setCapability("chrome.switches", Arrays.asList("--proxy-server=http://your-proxy-
domain:4443"));
  WebDriver driver = new ChromeDriver(capabilities);
```

Overriding the Chrome binary location

You can specify the location of the Chrome binary by passing the "chrome.binary" capability, e.g. with a typical Chromium install on Debian:

```
  DesiredCapabilities capabilities = DesiredCapabilities.chrome();
  capabilities.setCapability("chrome.binary", "/usr/lib/chromium-browser/chromium-browser");
  WebDriver driver = new ChromeDriver(capabilities);
```

Or in Ruby:

```
  Selenium::WebDriver::Chrome.path = "/usr/lib/chromium-browser/chromium-browser"
  driver = Selenium::WebDriver.for :chrome
```

Chrome Extensions

The ‘chrome.extensions’ capability can be used to specify a list of extensions, each in the form of a base64 encoded crx file, to install on startup.

Troubleshooting

If you are using the Remote WebDriver and you get the *The path to the chromedriver executable must be set by the webdriver.chrome.driver system property* error message you likely need to check that one of these conditions is met:

- The chromedriver binary is in the system path, or
- The Selenium Server was started with -Dwebdriver.chrome.driver=c:\path\to\your\chromedriver.exe

Known Issues

There are a handful of known issues with ChromeDriver, listed below:

1 Can only retrieve the name and value of set cookies (no domain, path, etc.)
2 Typing does not work with rich-text enabled documents
3 Cannot specify a custom profile
4 HTML 5 API not implemented
5 If running Selenium as a Windows Service, be sure to use the alternate 'standalone' chrome install. This will install chrome for all users, not just the current user. The Selenium service will try and start Chrome as the SYSTEM user and will not be able to find the binary, as noted in Issue #46. The alternate install is an easy way to correct that problem. The alternate install can be found here.

Think you've found a bug?

Check if the bug has been reported yet. If it hasn't, please open a new issue and be sure to include the following:

- What platform are you running on?
- What version of the chromedriver are you using?
- What version of Chrome are you using?
- The failure stacktrace, if available.
- The contents of chromedriver's log file (chromedriver.log).

Of course, if your bug has already been reported, you can update the issue with the information above. Having more information to work on makes it easier for us to track down the cause of the bug.

Testing earlier versions of Chrome

As previously mentioned, the ChromeDriver is only compatible with Chrome version 12.0.712.0 or newer. If you need to test an older version of Chrome, use Selenium RC and a Selenium-backed WebDriver instance:

```
URL seleniumServerUrl = new URL("http://localhost:4444");
URL serverUnderTest = new URL("http://www.google.com");
CommandExecutor executor = new SeleneseCommandExecutor(seleniumServerUrl,
serverUnderTest, DesiredCapabilities.chrome());
WebDriver driver = new RemoteWebDriver(executor);
```

Building From Source

The ChromeDriver is available as part of the Chromium project. See this page if you'd like to build the server yourself.