

selenium-jupiter
A JUnit 5 extension for Selenium WebDriver

Boni García

Version 1.2.0

Table of Contents

Quick reference	1
Motivation	3
Examples	5
Chrome	5
Firefox	6
Edge	7
Opera	8
Safari	9
PhantomJS	10
HtmlUnit	10
Advanced	11
Using options	12
Using capabilities	14
AppiumDriver	17
Tuning WebDriverManager	19
Debugging	21
Support	22
About	22

JUnit 5 is the next generation of the well-known testing framework JUnit. *Jupiter* is the name given to the new programming and extension model provided by JUnit 5. Regarding the extension model of JUnit 5, it allows to incorporate extra capabilities for Jupiter tests. On the other hand, **Selenium WebDriver** is a testing framework which allows to control browsers (e.g. Chrome, Firefox, and so on) programmatically to carry out automated testing of web applications. This documentation presents **selenium-jupiter**, a JUnit 5 extension aimed to provide seamless integration of Selenium WebDriver within Jupiter tests. *selenium-jupiter* is open source (Apache 2.0 license) and is hosted on [GitHub](#).

Quick reference

selenium-jupiter has been built using the [dependency injection](#) capability provided by the extension model of JUnit 5. Thank to this feature, different types objects can be injected in JUnit 5 in `@Test` methods as parameters. Concretely, *selenium-jupiter* allows to inject subtypes of the **WebDriver** interface (e.g. *ChromeDriver*, *FirefoxDriver*, and so on).

Using *selenium-jupiter* it's easy as pie. First, you need to import the dependency in your project (typically as *test* dependency). In Maven, it is done as follows:

```
<dependency>
  <groupId>io.github.bonigarcia</groupId>
  <artifactId>selenium-jupiter</artifactId>
  <version>1.2.0</version>
  <scope>test</scope>
</dependency>
```

NOTE

selenium-jupiter 1.2.0 depends on **selenium-java 3.8.1**, **webdrivermanager 2.0.1**, and **appium java-client 5.0.4**. Therefore, by using the *selenium-jupiter* dependency, those libraries (*selenium-java*, *webdrivermanager*, and *appium java-client*) will be added as transitive dependencies to your project.

Then, you need to declare *selenium-jupiter* extension in your JUnit 5 test, simply annotating your test with `@ExtendWith(SeleniumExtension.class)`. Finally, you need to include one or more parameters in your `@Test` methods whose types implements the **WebDriver** interface (e.g. *ChromeDriver* to use Chrome, *FirefoxDriver* for Firefox, and so for). That's it. *selenium-jupiter* control the lifecycle of the **WebDriver** object internally, and you just need to use the **WebDriver** object in your test to drive the browser(s) you want. For example:

```

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeAndFirefoxJupiterTest {

    @Test
    public void testWithOneChrome(ChromeDriver chromeDriver) {
        // using Chrome in this test
    }

    @Test
    public void testWithFirefox(FirefoxDriver firefoxDriver) {
        // using Firefox in this test
    }

    @Test
    public void testWithChromeAndFirefox(ChromeDriver chromeDriver,
        FirefoxDriver firefoxDriver) {
        // using Chrome and Firefox in this test
    }

}

```

The **WebDriver** subtypes supported by *selenium-jupiter* are the following:

- **ChromeDriver**: Used to control Google Chrome browser.
- **FirefoxDriver**: Used to control Firefox browser.
- **EdgeDriver**: Used to control Microsoft Edge browser.
- **OperaDriver**: Used to control Opera browser.
- **SafariDriver**: Used to control Apple Safari browser (only possible in OSX El Capitan or greater).
- **HtmlUnitDriver**: Used to control HtmlUnit (headless browser).
- **PhantomJSDriver**: Used to control PhantomJS (headless browser).
- **InternetExplorerDriver**: Used to control Microsoft Internet Explorer. Although this browser is supported, Internet Explorer is deprecated (in favor of Edge) and its use is highly discouraged.
- **RemoteWebDriver**: Used to control remote browsers (*Selenium Grid*).
- **AppiumDriver**: Used to control mobile devices (Android, iOS).

WARNING

The browser to be used must be installed in the machine running the test beforehand (except in the case of `RemoteWebDriver`, in which the requirement is to know a Selenium Server URL). In the case of mobile devices (`AppiumDriver`), the emulator should be up and running in local or available in a Appium Server identified by an URL.

Motivation

`Selenium WebDriver` allows to control different types of browsers (such as Chrome, Firefox, Edge, and so on) programmatically using different programming languages. This is very useful to implement automated tests for web applications. Nevertheless, in order to use WebDriver, we need to pay a prize. For security reasons, the automated manipulation of a browser can only be done using native features of the browser. In practical terms, it means that a binary file must be placed in between the test using the WebDriver API and the actual browser. On the one hand, the communication between the WebDriver object and that binary is done using the ([W3C WebDriver specification](#), formerly called *JSON Wire Protocol*). It consists basically on a REST service using JSON for requests and responses. On the other hand, the communication between the binary and the browser is done using native capabilities of the browser. Therefore, the general schema of Selenium WebDriver can be illustrated as follows:

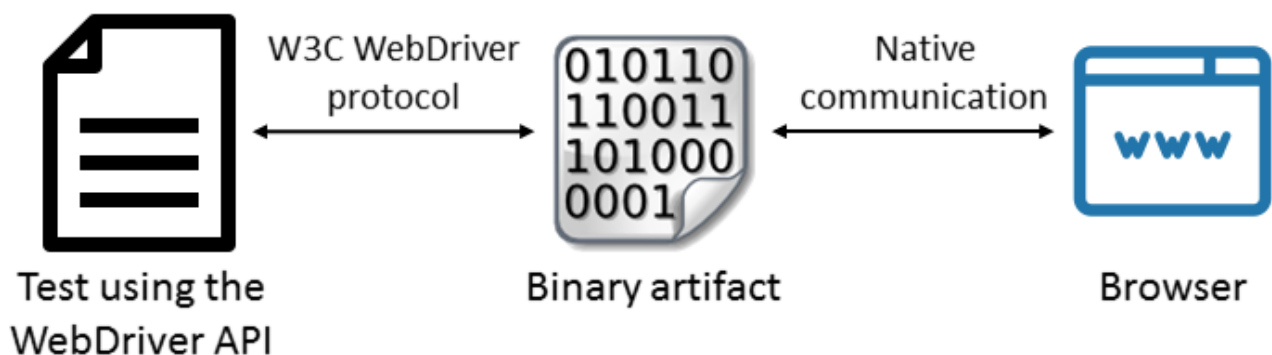


Figure 1. *WebDriver general scenario*

From a tester point of view, the need of this binary component is a pain in the neck, since it should be downloaded manually for the proper platform running the test (i.e. Windows, Linux, Mac). Moreover, the binary version should be constantly updated. The majority of browsers evolve quite fast, and the corresponding binary file required by WebDriver needs to be also updated. The following picture shows a fine-grained diagram of the different flavor of WebDriver binaries and browsers:

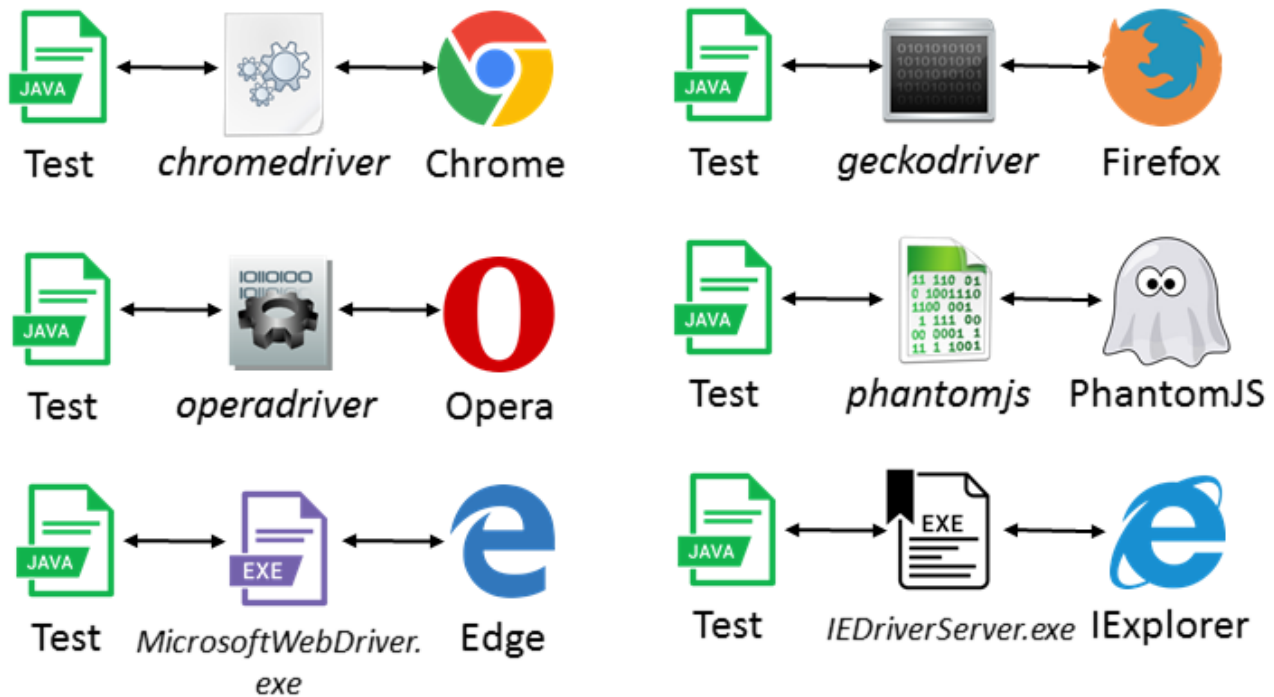


Figure 2. WebDriver scenario for Chrome, Firefox, Opera, PhantomJS, Edge, and Internet Explorer

Concerning Java, in order to locate these drivers, the absolute path of the binary controlling the browser should be exported in a given environment variable before creating a WebDriver instance, as follows:

```
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
System.setProperty("webdriver.opera.driver", "/path/to/operadriver");
System.setProperty("webdriver.ie.driver", "C:/path/to/IEDriverServer.exe");
System.setProperty("webdriver.edge.driver", "C:/path/to/MicrosoftWebDriver.exe");
System.setProperty("phantomjs.binary.path", "/path/to/phantomjs");
System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");
```

In order to simplify the life of Java WebDriver users, in March 2015 the utility [WebDriverManager](#) was first released. WebDriverManager is a library which automates all this process (download the proper binary and export the proper variable) for Java in runtime. The WebDriverManager API is quite simple, providing a singleton object for each of the above mentioned browsers:

```
ChromeDriverManager.getInstance().setup();
FirefoxDriverManager.getInstance().setup();
OperaDriverManager.getInstance().setup();
PhantomJsDriverManager.getInstance().setup();
EdgeDriverManager.getInstance().setup();
InternetExplorerDriverManager.getInstance().setup();
```

The solution implemented by WebDriverManager is today supported by similar tools for other languages, such as [webdriver-manager](#) for **Node.js** or [WebDriverManager.Net](#) for **.NET**.

On September 2017, a new major version of the well-know testing JUnit framework was released.

This leads to ***selenium-jupiter***, which can be seen as the natural evolution of *WebDriverManager* for **JUnit 5** tests. Internally, *selenium-jupiter* is built using two foundations:

1. It uses *WebDriverManager* to manage the binaries requires by WebDriver.
2. It uses the *dependency injection* feature of the extension model of JUnit 5 to inject WebDriver objects within `@Test` methods.

All in all, using Selenium WebDriver to control browsers using Java was never that easy. Using JUnit 5 and *selenium-jupiter*, you simply need to declare the flavor of browser you want to use in your test method and use it.

Examples

This section contains a comprehensive collection of examples demonstrating the basic usage of *selenium-jupiter* in JUnit 5 tests using different types of browsers. All these examples are part of the test suite of *selenium-jupiter* and are executed on [Travis CI](#).

Chrome

The following example contains a simple usage of Chrome in JUnit 5. The complete source code of this test is hosted on [GitHub](#). Notice that this class contains two tests (methods annotated with `@Test`). The first one (`testWithOneChrome`) declares just one `ChromeDriver` parameter, and therefore this test controls a single Chrome browser. On the other hand, the second `@Test` (`testWithTwoChromes`) declares two different `ChromeDriver` parameters, and so, it controls two Chrome browsers.

```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.startsWith;
import static org.hamcrest.MatcherAssert.assertThat;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeJupiterTest {

    @Test
    public void testWithOneChrome(ChromeDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }

    @Test
    public void testWithTwoChromes(ChromeDriver driver1, ChromeDriver driver2) {
        driver1.get("http://www.seleniumhq.org/");
        driver2.get("http://junit.org/junit5/");
        assertThat(driver1.getTitle(), startsWith("Selenium"));
        assertThat(driver2.getTitle(), equalTo("JUnit 5"));
    }
}

```

Firefox

The following `test` uses Firefox as browser(s). To that aim, `@Test` methods simply need to include `FirefoxDriver` parameters.


```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.startsWith;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.firefox.FirefoxDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class FirefoxJupiterTest {

    @Test
    public void testWithOneFirefox(FirefoxDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }

    @Test
    public void testWithTwoFirefoxs(FirefoxDriver driver1,
        FirefoxDriver driver2) {
        driver1.get("http://www.seleniumhq.org/");
        driver2.get("http://junit.org/junit5/");
        assertThat(driver1.getTitle(), startsWith("Selenium"));
        assertThat(driver2.getTitle(), equalTo("JUnit 5"));
    }
}

```

Edge

The following [example](#) uses one Edge browser. This test should be executed on a Windows machine with Edge.

```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.edge.EdgeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class EdgeJupiterTest {

    @Test
    void edgeTest(EdgeDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

TIP

The required version of *MicrosoftWebDriver.exe* depends on the version on Edge to be used (more info [here](#)). By default, WebDriverManager downloads and uses the latest version of the binaries. Nevertheless, a concrete version can be fixed. Take a look to the [advance examples section](#) to find out how to setup the different options of WebDriverManager.

Opera

Are you one of the few using Opera? No problem, you can still make automated [tests](#) with JUnit 5, WebDriver, and *selenium-jupiter*, as follows:

```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.opera.OperaDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class OperaJupiterTest {

    @Test
    public void test(OperaDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

Safari

You can also use [Safari](#) in conjunction with *selenium-jupiter*. Take into account that *SafariDriver* requires Safari 10 running on OSX El Capitan or greater.

```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.safari.SafariDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class SafariJupiterTest {

    @Test
    public void test(SafariDriver driver) {
        driver.get("http://www.seleniumhq.org/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

PhantomJS

PhantomJS is a headless browser (i.e. a browser without GUI), and it can be convenient for different types of tests. The following [example](#) demonstrates how to use PhantomJS with *selenium-jupiter*.

```
import static org.hamcrest.CoreMatchers.notNullValue;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.phantomjs.PhantomJSDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class PhantomjsJupiterTest {

    @Test
    public void test(PhantomJSDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getPageSource(), notNullValue());
    }
}
```

HtmlUnit

HtmlUnit is another headless browser that can be used easily in a Jupiter test, for [example](#) like this:

```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class HtmlUnitJupiterTest {

    @Test
    public void test(HtmlUnitDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

Advanced

So far, we have discovered how to use different browsers (Chrome, Firefox, Edge, Opera, Safari, PhantomJS, HtmlUnit) with its default options and supposing that the browser to be used is installed on the machine running the test. Nevertheless, if you have used intensively Selenium WebDriver, different questions might come to your mind:

- What if I need to specify options (e.g. `ChromeOptions`, `FirefoxOptions`, etc) to my WebDriver object?
- What if need to specify desired capabilities (e.g. browser type, version, platform)?
- And what about remote browsers (`Selenium Grid`)? How is `RemoteWebDriver` supported by *selenium-jupiter*?

In order to support the advance features of Selenium WebDriver, *selenium-jupiter* provides several annotations aimed to allow a fine-grained control of the WebDriver object instantiation. These annotations are:

- **DriverOptions**: Annotation to configure *options* (e.g. `ChromeOptions` for Chrome, `FirefoOptions` for Firefox, `EdgeOptions` for Edge, `OperaOptions` for Opera, and `SafariOptions` for Safari).
- **DriverCapabilities**: Annotation to configure the desired *capabilities* (WebDriver's object `DesiredCapabilities`).
- **DriverUrl**: Annotation used to identify the URL value needed to instantiate a `RemoteWebDriver` object.

All these 3 annotation can be used both at *parameter-level* (applied to a single WebDriver

parameter), and also at *field-level* (applied globally in a test class). Keep reading to find out several examples about that.

Using options

The following [example](#) shows how to specify options for Chrome. To that aim, the annotation `@DriverOptions` is used to set a collection of options. This array of options is configured using the annotation `@Option` declaring the parameters `name` and `value`). In the example, the annotation `@DriverOptions` is used at *parameter-level*. In the first test (called `headlessTest`), we are setting the argument `--headless`, used in Chrome to work as a headless browser. In the second test (`webrtcTest`), we are using two different arguments: `--use-fake-device-for-media-stream` and `--use-fake-ui-for-media-stream`, used to fake user media (i.e. camera and microphone) in [WebRTC](#) applications. In the third test (`extensionTest`), we are adding an extension to Chrome using the `@Option` named `EXTENSION`. The value of this field is an extension file that will be searched: i) using value as its relative/absolute path; ii) using value as a file name in the project classpath.

```

import static io.github.bonigarcia.SeleniumJupiter.ARGS;
import static io.github.bonigarcia.SeleniumJupiter.EXTENSION;
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.DriverOptions;
import io.github.bonigarcia.Option;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeWithOptionsJupiterTest {

    @Test
    void headlessTest(@DriverOptions(options = {
        @Option(name = ARGS, value = "--headless") }) ChromeDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }

    @Test
    void webrtcTest(@DriverOptions(options = {
        @Option(name = ARGS, value = "--use-fake-device-for-media-stream"),
        @Option(name = ARGS, value = "--use-fake-ui-for-media-stream") })
        ChromeDriver driver) {
        driver.get(
            "https://webrtc.github.io/samples/src/content/devices/input-output/");
        assertThat(driver.findElement(By.id("video")).getTagName(),
            equalTo("video"));
    }

    @Test
    void extensionTest(@DriverOptions(options = {
        @Option(name = EXTENSION, value = "hello_world.crx") }) ChromeDriver
        driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

As introduced before, this annotation `@DriverOptions` can be used also at *field-level*, as shown in this other [example](#). This test is setting to `true` the Firefox preferences `media.navigator.streams.fake` and

`media.navigator.permission.disabled`, used also for WebRTC.

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;

import io.github.bonigarcia.DriverOptions;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class FirefoxWithGlobalOptionsJupiterTest {

    @DriverOptions
    FirefoxOptions firefoxOptions = new FirefoxOptions();
    {
        // Flag to use fake media for WebRTC user media
        firefoxOptions.addPreference("media.navigator.streams.fake", true);

        // Flag to avoid granting access to user media
        firefoxOptions.addPreference("media.navigator.permission.disabled",
            true);
    }

    @Test
    public void webrtcTest(FirefoxDriver driver) {
        driver.get(
            "https://webrtc.github.io/samples/src/content/devices/input-output/");
        assertEquals(driver.findElement(By.id("video")).getTagName(),
            "video");
    }
}
```

Using capabilities

The annotation `@DriverCapabilities` is used to specify WebDriver capabilities (i.e. type browser, version, platform, etc.). These capabilities are typically used for Selenium Grid tests (i.e. tests using remote browsers). To that aim, an Selenium Hub (also known as *Selenium Server*) should be up and running, and its URL should be known. This URL will be specified using the *selenium-jupiter* annotation `@DriverUrl`.

The following example provides a complete [example](#) about this. As you can see, in the test setup (`@BeforeAll`) a Selenium Grid is implemented, first starting a Hub (a.k.a. *Selenium Server*), and then a couple of nodes (Chrome and Firefox) are registered in the Hub. Therefore, remote test using `RemoteWebDriver` can be executed, simply pointing to the Hub (whose URL in this case is <http://localhost:4444/wd/hub> in this example) and selecting the browser to be used using the `Capabilities`.


```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.openqa.selenium.remote.DesiredCapabilities.firefox;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.grid.selenium.GridLauncherV3;
import org.openqa.selenium.Capabilities;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.RemoteWebDriver;

import io.github.bonigarcia.Capability;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.DriverUrl;
import io.github.bonigarcia.SeleniumExtension;
import io.github.bonigarcia.wdm.ChromeDriverManager;
import io.github.bonigarcia.wdm.FirefoxDriverManager;

@ExtendWith(SeleniumExtension.class)
public class RemoteWebDriverJupiterTest {

    @DriverUrl
    String url = "http://localhost:4444/wd/hub";

    @DriverCapabilities
    Capabilities capabilities = firefox();

    @BeforeAll
    static void setup() throws Exception {
        // Start hub
        GridLauncherV3.main(new String[] { "-role", "hub", "-port", "4444" });

        // Register Chrome in hub
        ChromeDriverManager.getInstance().setup();
        GridLauncherV3.main(new String[] { "-role", "node", "-hub",
            "http://localhost:4444/grid/register", "-browser",
            "browserName=chrome", "-port", "5555" });

        // Register Firefox in hub
        FirefoxDriverManager.getInstance().setup();
        GridLauncherV3.main(new String[] { "-role", "node", "-hub",
            "http://localhost:4444/grid/register", "-browser",
            "browserName=firefox", "-port", "5556" });
    }

    @Test
    void testWithRemoteChrome(
        @DriverUrl("http://localhost:4444/wd/hub") @DriverCapabilities(capability
    = {

```

```

        @Capability(name = "browserName", value = "chrome") })
RemoteWebDriver driver)
    throws InterruptedException {
    exercise(driver);
}

@Test
void testWithRemoteFirefox(RemoteWebDriver driver)
    throws InterruptedException {
    exercise(driver);
}

void exercise(WebDriver driver) {
    driver.get("https://bonigarcia.github.io/selenium-jupiter/");
    assertThat(driver.getTitle(),
        containsString("A JUnit 5 extension for Selenium WebDriver"));
}
}

```

The following class contains an example which uses Chrome as browser and capabilities defined using `@DriverCapabilities`. Concretely, this example uses the mobile emulation feature provided out of the box by Chrome (i.e. render the web page using small screen resolutions to emulate smartphones).

```

import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import java.util.HashMap;
import java.util.Map;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.remote.DesiredCapabilities;

import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ChromeWithGlobalCapabilitiesJupiterTest {

    @DriverCapabilities
    DesiredCapabilities capabilities = DesiredCapabilities.chrome();
    {
        Map<String, String> mobileEmulation = new HashMap<String, String>();
        mobileEmulation.put("deviceName", "Nexus 5");
        Map<String, Object> chromeOptions = new HashMap<String, Object>();
        chromeOptions.put("mobileEmulation", mobileEmulation);
        capabilities.setCapability(ChromeOptions.CAPABILITY, chromeOptions);
    }

    @Test
    void chromeTest(ChromeDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

AppiumDriver

The annotation `@DriverCapabilities` can be also used to specify the desired capabilities to create an instances of `AppiumDriver` to drive mobile devices (Android or iOS). If not `@DriverUrl` is specified, *selenium-jupiter* will start automatically an instance of Appium Server (by default in port 4723) in the localhost after each test execution (this server is shutdown before each test). For example:

```

import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;

import io.appium.java_client.AppiumDriver;
import io.github.bonigarcia.Capability;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class AppiumChromeJupiterTest {

    @Test
    void testWithAndroid(@DriverCapabilities(capability = {
        @Capability(name = "browserName", value = "chrome"),
        @Capability(name = "deviceName", value = "Android") }) AppiumDriver
        <WebElement> driver)
        throws InterruptedException {

        String context = driver.getContext();
        driver.context("NATIVE_APP");
        driver.findElement(By.id("com.android.chrome:id/terms_accept")).click();
        driver.findElement(By.id("com.android.chrome:id/negative_button"))
            .click();
        driver.context(context);

        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertTrue(driver.getTitle().contains("JUnit 5 extension"));
    }
}

```

We can also specify a custom Appium Server URL changing the value of `@DriverUrl`, at field-level or parameter-level:

```

import java.io.File;
import java.net.URISyntaxException;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;

import io.appium.java_client.AppiumDriver;
import io.appium.java_client.MobileElement;
import io.github.bonigarcia.DriverCapabilities;
import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class AppiumApkJupiterTest {

    @DriverCapabilities
    DesiredCapabilities capabilities = new DesiredCapabilities();
    {
        try {
            File apk = new File(this.getClass()
                .getResource("/selendroid-test-app.apk").toURI());
            capabilities.setCapability("app", apk.getAbsolutePath());
            capabilities.setCapability("deviceName", "Android");

        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    @Test
    void testWithAndroid(AppiumDriver<MobileElement> driver)
        throws InterruptedException {
        WebElement button = driver.findElement(By.id("buttonStartWebview"));
        button.click();

        WebElement inputField = driver.findElement(By.id("name_input"));
        inputField.clear();
        inputField.sendKeys("Custom name");
    }
}

```

Tuning WebDriverManager

As introduced before, *selenium-jupiter* internally uses [WebDriverManager](#) to manage the required binary to control browsers. This tool can be configured in several ways, for example to force using a given version of the binary (by default it tries to use the latest version), or force to use the cache

(instead of connecting to the online repository to download the binary artifact). For further information about this configuration capabilities, please take a look to the [WebDriverManager documentation](#).

In this section we are going to present a couple of simple examples tuning somehow WebDriverManager. The following example shows how to force a version number for a binary, concretely for Edge:

```
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.edge.EdgeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class EdgeSettingVersionJupiterTest {

    @BeforeAll
    static void setup() {
        System.setProperty("wdm.edgeVersion", "3.14393");
    }

    @Test
    void webRTCTest(EdgeDriver driver) {
        driver.get("http://www.seleniumhq.org/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}
```

This other example shows how to force cache (i.e. binaries previously downloaded by WebDriverManager) to avoid the connection with online repository to check the latest version:

```

import static java.lang.System.setProperty;
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.SeleniumExtension;

@ExtendWith(SeleniumExtension.class)
public class ForceCacheJupiterTest {

    @BeforeAll
    static void setup() {
        setProperty("wdm.forceCache", "true");
    }

    @Test
    public void test(ChromeDriver driver) {
        driver.get("https://bonigarcia.github.io/selenium-jupiter/");
        assertThat(driver.getTitle(),
            containsString("A JUnit 5 extension for Selenium WebDriver"));
    }
}

```

Debugging

As of version 1.2.0, *selenium-jupiter* provides a built-in feature for debugging purposes. When a JUnit 5 test using this extension fails, *selenium-jupiter* will make an **screenshot** for each of the browser sessions at the end of the test. These screenshots, encoded in **Base 64**, are logged using the **debug** level of ([Simple Logging Facade for Java \(SLF4J\)](#)). You can copy&paste the resulting Base 64 string in the URL bar of any browser and watch the screenshot. All in all, this capability allows to watch what was the status of the web under test just after the test is declared as failure, which can help to identify the cause of the test failure.

TESTS

Running `io.github.bonigarcia.test.basic.ChromeJupiterTest`

...

2017-12-13 02:41:53 [main] DEBUG i.g.bonigarcia.SeleniumExtension - Screenshot (in Base64) at the end of session 5712cce700bb76d8f5f5d65a00e2c7bc (copy&paste this string as URL in browser to watch it)

data:image/png;base64,iVBORw0KGgoAAAANSUHEUgAAAYkAAANaCAIAAAACvpRSAAAgAE1EQVR4nOy9e3xV1Zn//1lr384t5+RyCDmBhJBwSRACBAqhXFQsY1GLZeygrWVGba1TtVU72gv9fgfnK3Yq36mdqTq0LTL1R8fyq2Wk3kCGSqEMLxKQiyQICZAAJySHnJyT5Nz2ZX3/ODGEk71DDuSCuN4vXrySnXX2WWfvtc7670d51vMQxhg4HA6Hw+Fw0OKTDofTfQkdiH5wOBw0h8PhcEzh2ovD4XA4HA5n80Dai8PhcDgcDmfW4NqLw+FwOBwOZ/Dg2ovD4XA4HA5n80Dai8PhcDgcDmfW4NqLw+FwOBwOp5/5wx/+YPUrr04HA6Hw+Fw+pOk8LKsX1x7cTgcDofD4fQb3SWXqfzi2o

...

nr2SgWY2TMdyRZMNENknWYTIo75WxuUWNCij9k178gCKO3BHLtkeQjiMYEgXy66/8f5nl+x57/98mYAvFysJUmcKrGBLMv/dC0SLJSUjrXpwn3Ba6MgAlILbL+//8/5fArudfRS108WQBr2oDm4uNtmZOPKRm7GSgptWXBfJPrAAAL01EQVS4/biJEm74sxVmrVz///+U468MXDMRZc7zY5Aj1uDndWEG2gCWk+R5ARMgl5aZrga3N7XBT9vCbXrBWPjx/3/96tsQz8LL4foDPzSV1H4crofHwrFXxyDZx1xHaWN7EjwAiT/f69Gf/8G105SUDCBnj2EW8huvfnTLaFNSMsD043+89QUmwCz/yUhp9AFYT/DCDwBA00ZPpbBVqWAAAABJRu5ErkJggg==

Tests run: 2, Failures: 1, Errors: 0, Skipped: 1, Time elapsed: 7.219 sec <<< FAILURE!

- in `io.github.bonigarcia.test.basic.ChromeJupiterTest`

`testWithOneChrome(ChromeDriver)` Time elapsed: 6.594 sec <<< FAILURE!

Support

There are several ways to get in touch with *selenium-jupiter*:

- Questions about *selenium-jupiter* are supposed to be discussed in [StackOverflow](#), using the tag *selenium-jupiter*.
- Comments, suggestions and bug-reporting should be done using the [GitHub issues](#).
- If you think *selenium-jupiter* can be enhanced, consider contribute to the project by means of a [pull request](#).

About

selenium-jupiter (Copyright © 2017) is a project created by [Boni Garcia \(@boni_gg\)](#) licensed under [Apache 2.0 License](#). This documentation is released under the terms of [CC BY 3.0](#) (also available in [PDF](#)).