Upgrade from Selenium 3 to Selenium 4

Java only

Ashis Raj

There is huge excitement within the testing community since the announcement of the release of Selenium 4 with major changes and W3C standardization to the Selenium suite (Selenium IDE, Selenium WebDriver and Selenium Grid).

One of the main reasons to release WebDriver as a major version (Selenium 4) is because of the complete W3C protocol adoption. The W3C protocol dialect has been available since the 3.8 version of Selenium WebDriver along with the JSON wire protocol. This change in protocol isn't going to impact the users in any way, as all major browser drivers (such as geckodriver and chromedriver), and many third party projects, have already fully adopted the W3C protocol.

Selenium 4 removes support for the legacy protocol and uses the W3C WebDriver standard by default under the hood.

This means that the tests will now run more consistently on each browser, as it will directly communicate, without the need of any encoding and decoding of API requests, through W3C Protocol.

Upgrade the Dependencies

The process of upgrading Selenium depends on which build tool is being used. The below example is for the most common one, which is Maven. The minimum Java version required is still 8.

Selenium 3

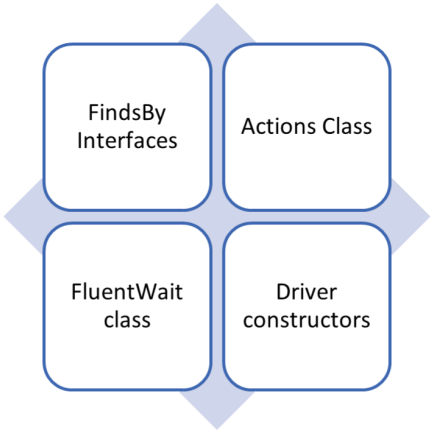
```
<dependencies>
  <!-- more dependencies ... -->
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
  </dependency>
  <!-- more dependencies ... -->
</dependencies>
```

Selenium 4

```
<dependencies>
    <!-- more dependencies ... -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>4.1.1</version>
    </dependency>
    <!-- more dependencies ... -->
</dependencies>
```

What's been deprecated/removed/replaced/added in Selenium 4?

Before planning a migration to Selenium 4, it is critical to understand what are all which have been deprecated/removed/replaced/added in this new version so that you can easily identify areas of your test automation framework that could possibly be affected by the changes.



FindsBy Interfaces *(removed)*

FindsBy interfaces were part of org.openqa.selenium.internal package having findElement(By) and findElements(By) utility methods, implemented by the RemoteWebDriver class. These are now removed. The By class can be used with findElement(By) and findElements(By) just like before.

Selenium 3

```
driver.findElementByClassName("className");
driver.findElementByCssSelector(".className");
driver.findElementById("elementId");
driver.findElementByLinkText("linkText");
driver.findElementByName("elementName");
driver.findElementByPartialLinkText("partialText");
driver.findElementByTagName("elementTagName");
driver.findElementByXPath("xPath");
```

Selenium 4

```
driver.findElement(By.className("className"));
driver.findElement(By.cssSelector(".className"));
driver.findElement(By.id("elementId"));
driver.findElement(By.linkText("linkText"));
driver.findElement(By.name("elementName"));
driver.findElement(By.partialLinkText("partialText"));
driver.findElement(By.tagName("elementTagName"));
driver.findElement(By.xpath("xPath"));
```

Actions Class *(few new methods added)*

The Actions class is a user-facing API for emulating complex user gestures like hovering, mouse movements, etc. A few new methods have been added to the Actions class as a replacement of the classes under package org.openqa.selenium.interactions. These methods can be replaced as shown in the below code snippet.

Selenium 3

```
Actions act = new Actions(driver);
WebElement toDoList= driver.findElement(By.id("toDoListBtn"));

// click method - to click on a webElement
act.moveToElement(toDoList).click();

// double click - double click on a webElement
act.moveToElement(toDoList).doubleClick();
```

```
// Context click - Right click on a webElement
act.moveToElement(toDoList).contextClick();

//clickAndHold method - click and hold on a webElement without releasing
act.moveToElement(toDoList).clickAndHold();

// release -release the hold on a webElement
act.moveToElement(toDoList).release();
```

Selenium 4

```
Actions act = new Actions(driver);
WebElement toDoList= driver.findElement(By.id("toDoListBtn"));

// click method - to click on a webElement
act.click(toDoList);

//clickAndHold method - click and hold on a webElement without releasing
act.clickAndHold(toDoList);

// Context click - Right click on a webElement
act.contextClick(toDoList);

// double click - double click on a webElement
act.doubleClick(toDoList);

// release -release the hold on a webElement
act.release(toDoList);
```

FluentWait Class *(Waits and Timeouts)*

The parameters received in Timeout have switched from expecting (long time, TimeUnit unit) to expect (Duration duration).

Selenium 3

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.manage().timeouts().setScriptTimeout(2, TimeUnit.MINUTES);
driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);
```

Selenium 4

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
driver.manage().timeouts().scriptTimeout(Duration.ofMinutes(2));
```

```
driver.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(10));
```

Waits are also expecting different parameters now. WebDriverWait is now expecting a Duration instead of a long for timeout in seconds and milliseconds. The withTimeout and pollingEvery utility methods from FluentWait have switched from expecting (long time, TimeUnit unit) to expect (Duration duration).

Selenium 3

```
new WebDriverWait(driver, 3)
    .until(ExpectedConditions.elementToBeClickable(By.cssSelector("#id")));

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
    .withTimeout(30, TimeUnit.SECONDS)
    .pollingEvery(5, TimeUnit.SECONDS)
    .ignoring(NoSuchElementException.class);
```

Selenium 4

```
new WebDriverWait(driver, Duration.ofSeconds(3))
  .until(ExpectedConditions.elementToBeClickable(By.cssSelector("#id")));

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
  .withTimeout(Duration.ofSeconds(30))
  .pollingEvery(Duration.ofSeconds(5))
  .ignoring(NoSuchElementException.class);
```

Driver Constructors and Capabilities *(replaced)*

A couple of driver constructors have been replaced. Namely, the ones that accepted Capabilities objects have been replaced with ones that accept Options. This means you will need to create a specific Options object for whichever Driver class you're using, set your requirements and pass this object to the Driver constructor.

If the test capabilities are not structured to be W3C compliant, they may cause a session not to be started. Here is the list of W3C WebDriver standard capabilities:

- browserName
- browserVersion (replaces version)
- platformName (replaces platform)
- acceptInsecureCerts
- pageLoadStrategy
- proxy
- timeouts
- unhandledPromptBehavior

An up-to-date list of standard capabilities can be found at W3C WebDriver.

Any capability that is not contained in the list above, needs to include a vendor prefix. This applies to browser specific capabilities as well as cloud vendor specific capabilities. For example, if your cloud vendor uses build and name capabilities for your tests, you need to wrap them in a cloud:options block (check with your cloud vendor for the appropriate prefix).

Selenium 3

```
DesiredCapabilities caps = DesiredCapabilities.chrome();
caps.setCapability("platform", "Windows 10");
caps.setCapability("version", "92");
caps.setCapability("build", myTestBuild);
caps.setCapability("name", myTestName);
WebDriver driver = new RemoteWebDriver(new URL(cloudUrl), caps);
```

Selenium 4

```
ChromeOptions browserOptions = new ChromeOptions();
browserOptions.setPlatformName("Windows 10");
browserOptions.setBrowserVersion("92");
Map<String, Object> cloudOptions = new HashMap<>();
cloudOptions.put("build", myTestBuild);
cloudOptions.put("name", myTestName);
browserOptions.setCapability("cloud:options", cloudOptions);
WebDriver driver = new RemoteWebDriver(new URL(cloudUrl), browserOptions);
```

BrowseType Interface *(deprecated)*

The BrowserType interface has been around for a long time, however it is getting deprecated in favour of the new Browser interface.

Selenium 3

```
MutableCapabilities capabilities = new MutableCapabilities();
capabilities.setCapability("browserVersion", "92");
capabilities.setCapability("browserName", BrowserType.FIREFOX);
```

Selenium 4

```
MutableCapabilities capabilities = new MutableCapabilities();
capabilities.setCapability("browserVersion", "92");
capabilities.setCapability("browserName", Browser.FIREFOX);
```