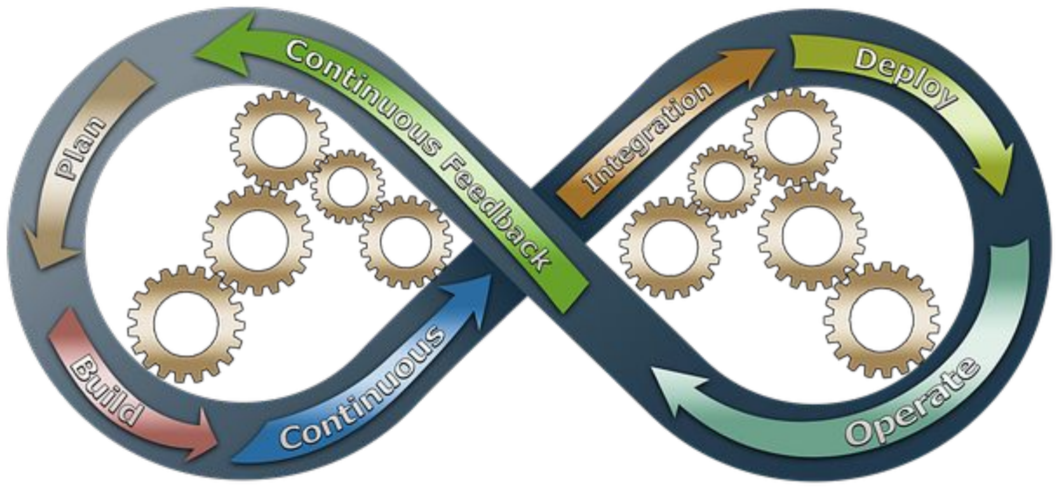


Session 4.8

Events and Others

AN INITIATIVE BY

UNICAL ACADEMY

Session 4.8

Introduction

UNICAL ACADEMY

Let’s go!!!

Handling Keyboard & Mouse Events

Method	Description
clickAndHold()	Clicks (without releasing) at the current mouse location.
contextClick()	Performs a context-click at the current mouse location.
doubleClick()	Performs a double-click at the current mouse location.
dragAndDrop(source, target)	Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse. **Parameters:** 1) source- element to emulate button down at. 2) target- element to move to and release the mouse at.
dragAndDropBy(source, x-offset, y-offset)	Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse. **Parameters:** 1) source- element to emulate button down at. 2) xOffset- horizontal move offset. 3) yOffset- vertical move offset.
keyDown(modifier_key)	Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed. **Parameters:** modifier_key - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL)
keyUp(modifier _key)	Performs a key release. **Parameters:** modifier_key - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL)

Method	Description
moveByOffset(x-offset, y-offset)	Moves the mouse from its current position (or 0,0) by the given offset. **Parameters:** x-offset- horizontal offset. A negative value means moving the mouse left. y-offset- vertical offset. A negative value means moving the mouse down.
moveToElement(toElement)	Moves the mouse to the middle of the element. **Parameters:** toElement- element to move to.
release()	Releases the depressed left mouse button at the current mouse location
sendKeys(onElement, charsequence)	Sends a series of keystrokes onto the element. **Parameters:** onElement - element that will receive the keystrokes, usually a text field charsequence - any string value representing the sequence of keystrokes to be sen

Page scroll

UNICAL ACADEMY

Page Scroll

The method **executeScript** is used to run Javascript commands in Selenium

Syntax:

```
WebElement elm = driver.findElement(By.name("name"));
((JavascriptExecutor)
driver).executeScript("arguments[0].scrollIntoView(true);",elm);
```

Example:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.JavascriptExecutor;
public class ScrollAction{
 public static void main(String[] args) {
System.setProperty("webdriver.chrome.driver",
"C:\\Users\\ghs6kor\\Desktop\\Java\\chromedriver.exe");
 WebDriver driver = new ChromeDriver();
driver.get("https://www.tutorialspoint.com/about/about_careers.htm ");
driver.manage().timeouts().implicitlyWait(4, TimeUnit.SECONDS);
// identify element
WebElement n=driver.findElement(By.xpath("//*[text()='Contact']"));
  // Javascript executor
((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView     (true);",
n);   }}
```

ScreenShot

Screenshot

- To take a screenshot in Selenium, we use an interface called TakesScreenshot, which enables the **Selenium *WebDriver*** to capture a screenshot and store it in different ways. It has a got a method "***getScreenshotAs()*** " which captures the screenshot and store it in the specified location.

Step-1:

```
File screenshotFile = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
```

- In the above code, we convert the WebDriver object (driver) to TakeScreenshot.
- And call getScreenshotAs() method to create an image file by providing the parameter OutputType.FILE.

Step-2: Save File in our desired location

```
FileUtils.copyFile(screenshotFile , new File("C:\\temp\\screenshot.png));
```

Screenshot of Full Page

- Selenium WebDriver doesn't provide the inherent capability to capture screenshot of the whole page.
- we have to use a third-party library named Ashot. It provides the ability to take a screenshot of a particular WebElement as well as a full-page screenshot.

Syntax:

```
Screenshot screenshot = new Ashot().takeScreenshot(driver);
```

- Capture the full page screenshot, which is more than the currently visible part on the screen.
- After creating the AShot object, we need to call the shootingStrategy() method before calling the takeScreenshot() method to set up the policy

Syntax:

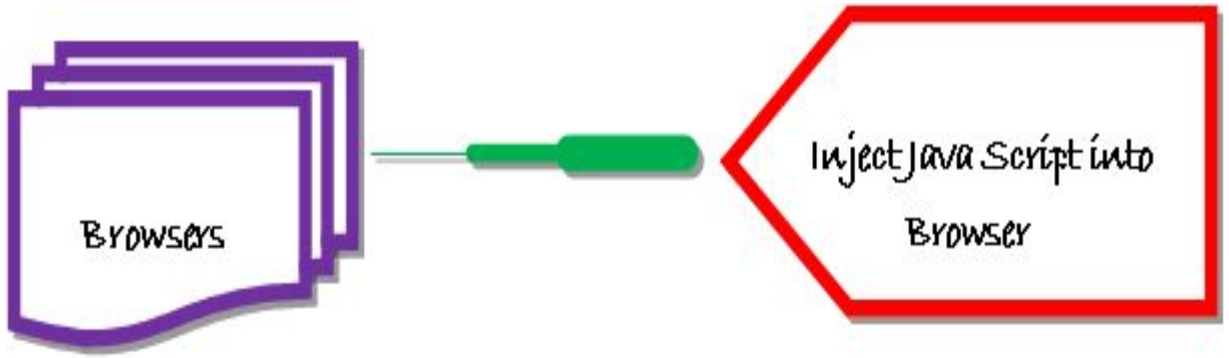
```
Screenshot s=new
AShot().shootingStrategy(ShootingStrategies.viewportPasting(1000)).
takeScreenshot(driver);

ImageIO.write(s.getImage(),"PNG",new File("<< file path>>"));
```

Execute JavaScript

JavaScriptExecutor

- JavaScriptExecutor is an Interface that helps to execute JavaScript through Selenium Webdriver.
- JavaScriptExecutor provides two methods "executescript" & "executeAsyncScript" to run javascript on the selected window or current page.



UNICAL ACADEMY

Methods of JavascriptExecutor

executeScript()

To run the specified JavaScript code in the current window or frame.

Syntax :

```
JavascriptExecutor jsExecutor = (JavascriptExecutor)driver;
jsExecutor.executeScript("JavaScriptCode");
```

executeAsyncScript()

To run specified asynchronous JavaScript code in the current window or frame. As the javaScript runs asynchronously, it requires an explicit callback indicating the finishing of script execution.

Syntax :

```
JavascriptExecutor jsExecutor = (JavascriptExecutor)driver;
jsExecutor.executeAsyncScript("Async JavaScript Code");
```

Headless Browser

Headless Browser

- A headless browser is a web-browser **without a graphical user interface**.
- These programs execute like any other browser but do not display any *UI*.
- In headless browsers, when *Selenium* tests run, they execute in the background.

Benefits

- Useful in CI pipeline
- Beneficial in web scraping
- Support for multiple browser versions
- Faster automation test execution
- Multi-Tasking

Different Types of Headless Browser

The following below are different kinds of implementation approach of Headless driver

1. HtmlUnit
2. Ghost
3. PhantomJS
4. ZombieJS
5. Watir-webdriver

Limitations:

- Debugging will not be feasible, as the only way to check what's running on the browser is to grab the screenshots and validate the output.
- Headless browsers don't mimic the exact user behavior, as the page doesn't render precisely with all the dependencies that it will render in an actual browser.
- Cosmetic bugs like the location of a web element, the color of a web element may get missed while running the tests in headless mode.

HTMLUnit Driver

UNICAL ACADEMY

setup of HTMLUnit Driver

Step 1:

To download the HtmlUnitDriver dependencies, follow the steps as mentioned below:

1. Navigate to **https://github.com/SeleniumHQ/htmlunit-driver/releases**.

2. Click on the latest version of the HTML unit driver as shown in the image below:

Step 2:

After downloading the jar file, follow the below steps to add the jar to the Eclipse project.

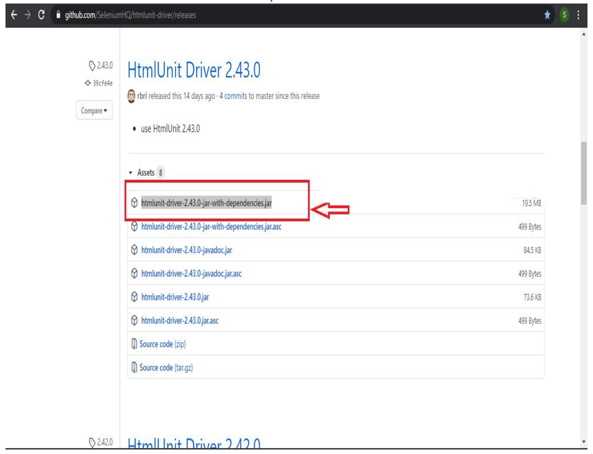
1. Right-click on your project.
2. Select Build Path and Click on Configure Build Path

Step 3:

- Click on the Libraries tab and select Add External JARs.
- Select the downloaded jar file from the folder.
- Click and Apply and Ok. The jar will add as below –

Step 4:

- Once you have the jar added to the Eclipse project, you can import the class.
- “import org.openqa.selenium.htmlunit.HtmlUnitDriver;”
- Add above line into the code.
- You can create a
- HtmlUnitWebDriver instance as
- HtmlUnitDriver unitDriver = new HtmlUnitDriver();



github.com/SeleniumHQ/htmlunit-driver/releases

2.43.0
39cfe4e
Compare ▾

HtmlUnit Driver 2.43.0

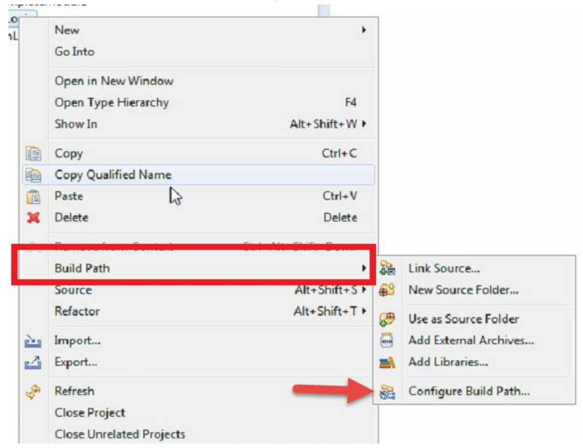
rbri released this 14 days ago · 4 commits to master since this release

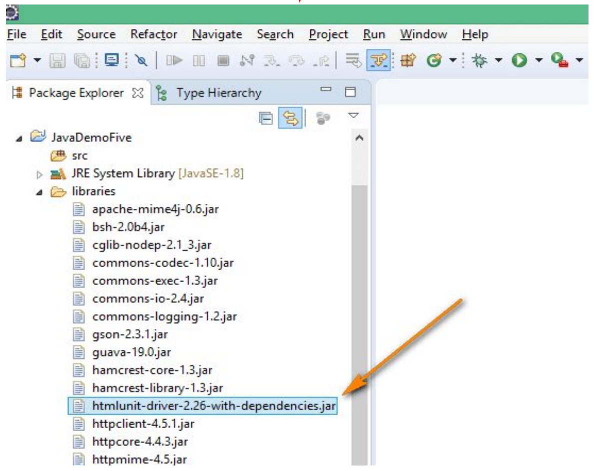
- use HtmlUnit 2.43.0

Assets 8

htmlunit-driver-2.43.0-jar-with-dependencies.jar	19.5 MB
htmlunit-driver-2.43.0-jar-with-dependencies.jar.asc	499 Bytes
htmlunit-driver-2.43.0-javadoc.jar	84.5 KB
htmlunit-driver-2.43.0-javadoc.jar.asc	499 Bytes
htmlunit-driver-2.43.0.jar	73.6 KB
htmlunit-driver-2.43.0.jar.asc	499 Bytes
Source code (zip)	
Source code (tar.gz)	

2.42.0
HtmlUnit Driver 2.42.0




File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Type Hierarchy

```
JavaDemoFive
    src
    JRE System Library [JavaSE-1.8]
    libraries
        apache-mime4j-0.6.jar
        bsh-2.0b4.jar
        cglib-nodep-2.1_3.jar
        commons-codec-1.10.jar
        commons-exec-1.3.jar
        commons-io-2.4.jar
        commons-logging-1.2.jar
        gson-2.3.1.jar
        guava-19.0.jar
        hamcrest-core-1.3.jar
        hamcrest-library-1.3.jar
        htmlunit-driver-2.26-with-dependencies.jar
        httpclient-4.5.1.jar
        httpcore-4.4.3.jar
        httpmime-4.5.jar
```

Handling Notifications

UNICAL ACADEMY

Different Types of Notifications

- Push Notifications Popups.
- Geo Location Popup.
- Automation Info Bar.

Implementation Steps

Step 1:

- Create a instance of ChromeOptions class.
- ChromeOptions options = new ChromeOptions();

Step 2:

- Add chrome switch to disable notification – "–disable-notifications"
- options.addArguments("--disable-notifications");

Step 3:

- Set path for the chrome driver.
- System.setProperty("webdriver.chrome.driver", "/home/users/garima.pathak/Desktop/softwares/chromedriver");

Step 4:

- Pass ChromeOptions instance to ChromeDriver Constructor.
- WebDriver driver =new ChromeDriver(options);
- Give the navigation of the page in which we want to handle the notifications.
- driver.get("http://wordpressdemo.webkul.com/wordpress-latest-tweets/");

UNICAL ACADEMY

Session Recap