AUTOMATE THE PLANET

Most Complete Selenium WebDriver Kotlin Cheat Sheet

Initialize

```
// Maven: selenium-chrome-driver
import org.openqa.selenium.chrome.ChromeDriver
val driver: WebDriver = ChromeDriver()
// Maven: selenium-firefox-driver
import org.openqa.selenium.firefox.FirefoxDriver
val driver: WebDriver = FirefoxDriver()
// Maven: selenium-edge-driver
import org.openqa.selenium.firefox.EdgeDriver
val driver: WebDriver = EdgeDriver()
// Maven: selenium-ie-driver
import org.openqa.selenium.ie.InternetExplorerDriver
val driver: WebDriver = InternetExplorerDriver()
// Maven: selenium-safari-driver
import org.openqa.selenium.safari.SafariDriver
val driver: WebDriver = SafariDriver()
```

Locators

```
driver.findElement(By.className("className"))
driver.findElement(By.cssSelector("css"))
driver.findElement(By.id("id"))
driver.findElement(By.linkText("text"))
driver.findElement(By.name("name"))
driver.findElement(By.partialLinkText("pText"))
driver.findElement(By.tagName("input"))
driver.findElement(By.xpath("//*[@id='editor']"))
// Find multiple elements
val anchors = driver.findElements(By.tagName("a"))
// Search for an element inside another
val div =
driver.findElement(By.tagName("div")).findElement(By.tagName("a"))
```

Basic Browser Operations

```
// Navigate to a page
driver.navigate().to("http://google.com")
// Get the title of the page
val title = driver.title
// Get the current URL
val url = driver.currentUrl
// Get the current page HTML source
val html = driver.pageSource
```

Basic Elements Operations

```
val element = driver.findElement(By.id("id"))
element.click()
element.sendKeys("someText")
element.clear()
element.submit()
val innerText = element.text
val isEnabled = element.isEnabled
val isDisplayed = element.isDisplayed
val isSelected = element.isSelected
val element = driver.findElement(By.id("id"))
val select = Select(element)
select.selectByIndex(1)
select.selectByVisibleText("Ford")
select.selectByValue("ford")
select.deselectAll()
select.deselectByIndex(1)
select.deselectByVisibleText("Ford")
select.deselectByValue("ford")
val allSelected: List<WebElement> = select.getAllSelectedOptions()
val isMultipleSelect: Boolean = select.isMultiple()
```

Advanced Elements Operations

```
// Drag and Drop
val element: WebElement = driver.findElement(By.xpath(
"//*[@id='project']/p[1]/div/div[2]"))
Actions(driver).dragAndDropBy(element, 30, 0).build().perform()
// Check if an element is visible
Assert.assertTrue(driver.findElement(By.xpath(
"//*[@id='tve_editor']/div")).isDisplayed)
// Upload a file
val element = driver.findElement(By.id("RadUpload1file0"))
val filePath = "D://WebDriver.Series.Tests//WebDriver.xml"
element.sendKeys(filePath)
// Scroll focus to control
val link = driver.findElement(By.partialLinkText("Previous post"))
(driver as JavascriptExecutor).executeScript(
"window.scroll(0, ${link.location.getY()});")
// Taking an element screenshot
val element = driver.findElement(By.xpath(
"//*[@id='tve_editor']/div"))
val screenshotFile =
(driver as TakesScreenshot).getScreenshotAs(OutputType.FILE)
val fullImg = ImageIO.read(screenshotFile)
val point = element.location
val elementWidth = element.size.getWidth()
val elementHeight = element.size.getHeight()
val eleScreenshot = fullImg.getSubimage(point.getX(), point.getY(),
elementWidth, elementHeight)
ImageIO.write(eleScreenshot, "png", screenshotFile)
val tempDir: String = getProperty("java.io.tmpdir")
val destFile = File(Paths.get(tempDir, "$fileName.png").toString())
FileUtils.getFileUtils().copyFile(screenshotFile, destFile)
// Focus on a control
val link = driver.findElement(By.partialLinkText("Previous post"))
Actions(driver).moveToElement(link).build().perform()
// Wait for visibility of an element
WebDriverWait(driver, 30).until(
ExpectedConditions.visibilityOfAllElementsLocatedBy(
By.xpath("//*[@id='tve_editor']/div[2]/div[2]/div/div")))
```

Advanced Browser Operations

```
// Handle JavaScript pop-ups
val alert = driver.switchTo().alert()
alert.accept()
alert.dismiss()
// Switch between browser windows or tabs
val windowHandles = driver.windowHandles
val firstTab = windowHandles.toTypedArray().first()
driver.switchTo().window(windowHandles.toTypedArray()[2])
// Navigation history
driver.navigate().back()
driver.navigate().refresh()
driver.navigate().forward()
// Maximize window
driver.manage().window().maximize()
// Add a new cookie
val newCookie = Cookie("customName", "customValue")
driver.manage().addCookie(newCookie)
// Get all cookies
val cookies = driver.manage().cookies
// Delete a cookie by name
driver.manage().deleteCookieNamed("CookieName")
// Delete all cookies
driver.manage().deleteAllCookies()
//Taking a full-screen screenshot
val screenshotFile = (driver as
TakesScreenshot).getScreenshotAs(OutputType.FILE)
val tempDir = getProperty("java.io.tmpdir")
val destFile = File(Paths.get(tempDir, "$fileName.png").toString())
FileUtils.getFileUtils().copyFile(screenshotFile, destFile)
// Wait until a page is fully loaded via JavaScript
WebDriverWait(driver, 30).until<Any>
{ (it as JavascriptExecutor).executeScript("return d
ocument.readyState" ) as String == "complete" }
// Switch to frames
driver.switchTo().frame(1)
driver.switchTo().frame("frameName")
val element = driver.findElement(By.id("id"))
driver.switchTo().frame(element)
// Switch to the default document
driver.switchTo().defaultContent()
```

Advanced Browser Configurations

```
// Use a specific Firefox profile
val profile = ProfilesIni()
val firefoxProfile = profile.getProfile("ProfileName")
val firefoxOptions = FirefoxOptions()
firefoxOptions.setProfile(firefoxProfile)
driver = FirefoxDriver(firefoxOptions)
// Set a HTTP proxy Firefox
val profile = ProfilesIni()
val firefoxProfile = FirefoxProfile()
firefoxProfile.setPreference("network.proxy.type", 1)
firefoxProfile.setPreference("network.proxy.http", "myproxy.com")
firefoxProfile.setPreference("network.proxy.http_port", 3239)
val firefoxOptions = FirefoxOptions()
firefoxOptions.setProfile(firefoxProfile)
driver = FirefoxDriver(firefoxOptions)
// Set a HTTP proxy Chrome
val proxy = Proxy()
proxy.setProxyType(Proxy.ProxyType.MANUAL)
proxy.setAutodetect(false)
proxy.setSslProxy("127.0.0.1:3239")
val chromeOptions = ChromeOptions()
chromeOptions.setProxy(proxy)
driver = ChromeDriver(chromeOptions)
// Accept all certificates Firefox
val firefoxProfile = FirefoxProfile()
firefoxProfile.setAcceptUntrustedCertificates(true)
firefoxProfile.setAssumeUntrustedCertificateIssuer(false)
val firefoxOptions = FirefoxOptions()
firefoxOptions.setProfile(firefoxProfile)
driver = FirefoxDriver(firefoxOptions)
// Accept all certificates Chrome
val chromeOptions = ChromeOptions()
chromeOptions.addArguments("--ignore-certificate-errors")
driver = ChromeDriver(chromeOptions)
// Set Chrome options
val chromeOptions = ChromeOptions()
chromeOptions.addArguments("user-data-dir=C:PathToUserData")
driver = ChromeDriver(chromeOptions)
// Turn off the JavaScript Firefox
val profile = ProfilesIni()
val firefoxProfile = profile.getProfile("ProfileName")
firefoxProfile.setPreference("javascript.enabled", false)
val firefoxOptions = FirefoxOptions()
firefoxOptions.setProfile(firefoxProfile)
driver = FirefoxDriver(firefoxOptions)
// Set the default page load timeout
driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS)
// Start Firefox with plugins
val profile = FirefoxProfile()
firefoxProfile.addExtension(File("C:extensionsLocationextension.xpi"))
val firefoxOptions = FirefoxOptions()
firefoxOptions.setProfile(firefoxProfile)
driver = FirefoxDriver(firefoxOptions)
// Start Chrome with an unpacked extension
val chromeOptions = ChromeOptions()
chromeOptions.addArguments("load-extension=/path/to/extension")
driver = ChromeDriver(chromeOptions)
// Start Chrome with a packed extension
val chromeOptions = ChromeOptions()
chromeOptions.addExtensions(File("local/path/to/extension.crx"))
driver = ChromeDriver(chromeOptions)
// Change the default files' save location
val firefoxProfile = FirefoxProfile()
val downloadFilepath = "c:temp"
firefoxProfile.setPreference("browser.download.folderList", 2)
firefoxProfile.setPreference("browser.download.dir", downloadFilepath)
firefoxProfile.setPreference("browser.download.manager.alertOnEXEOpen",
false)
firefoxProfile.setPreference("browser.helperApps.neverAsk.saveToDisk",
"application/msword, application/binary, application/ris, text/csv,
image/png, application/pdf, text/html, text/plain, application/zip,
application/x-zip, application/x-zip-compressed, application/download,
application/octet-stream")
val firefoxOptions = FirefoxOptions()
firefoxOptions.setProfile(firefoxProfile)
driver = FirefoxDriver(firefoxOptions)
```