**Selenium Class**

# Chapter#16 - Window Operations

1. Create New Tab
2. Create New Window
3. Full Screen Window
4. Window Handle
5. Window Handles

## Create New Tab

**driver.switchTo().newWindow(WindowType.TAB);**

Opens a new browser tab. Switches the WebDriver's focus to the newly created tab, allowing subsequent commands to interact with elements within that tab.

**This method was introduced in Selenium 4.**

**Browser compatibility:** Not all browsers may fully support the WindowType distinction between tabs and windows. The driver might create a new window instead of a tab if a tab isn't supported.

**Subsequent interactions:** After opening the new tab, you can use WebDriver commands to navigate to a URL, interact with elements, and perform actions within that tab.

## Create New Window

**driver.switchTo().newWindow(WindowType.WINDOW);**

Open a new browser window: This code snippet instructs the WebDriver to create a new, separate browser window. Unlike newWindow(WindowType.TAB), which opens a new tab within the same browser window, this creates a completely new window with its own set of controls and history.

**Switch focus to the new window:** Additionally, the code automatically switches the WebDriver's focus to the newly created window. This means subsequent WebDriver commands like get(), findElement(), and click() will operate on elements within the new window, not the original one.

Like newWindow(WindowType.TAB), this feature was introduced in **Selenium 4.**

**Browser compatibility:** While the code aims to open a window, some browsers might create a tab instead if they don't fully support distinct window types.

**Subsequent interactions:** After opening the new window, you can use WebDriver commands to navigate to URLs, interact with elements, and perform actions within that separate window.

## Full Screen Window

**driver.manage().window().fullscreen();**

The code driver.manage().window().fullscreen(); in Selenium serves two main purposes:

1. **Maximize the browser window:** In most browsers, using this command will expand the browser window to fill the entire screen, excluding the taskbar or menu bar. This can be helpful for maximizing the available space when automating web interactions.

2. **Enter full-screen mode (browser-specific):** However, in some browsers like Chrome, using this command actually triggers full-screen mode, which hides the menu bar and taskbar completely, offering a truly immersive experience.

Browser differences: The exact behavior of this command varies across browsers. In some, it only maximizes, while in others, it enters full-screen mode.

**Temporary effect:** This command affects the current window of the driver. If you open new windows or tabs, they won't automatically be in full screen.

**Potential usability issues:** Full-screen mode can sometimes reduce accessibility and user experience, so use it judiciously in automation scripts.

## Window Handle

**driver.getWindowHandle();**

The driver.getWindowHandle(); method in Selenium serves a crucial purpose in managing multiple browser windows and tabs:

1. **Obtaining the unique identifier of the current window:** This method returns a string representing the unique identifier of the window that the Selenium WebDriver is currently focused on. Think of it as a license plate number for each window.

2. **Switching control between windows:** By storing these window handles, you can use methods like driver.switchTo().window(handle) to switch focus to a specific window based on its handle. This enables interacting with different windows in your automation script.

3. **Identifying windows when multiple exist:** When working with multiple windows or tabs, getWindowHandle() helps you distinguish between them. You can store handles in a data structure like a set or list and iterate through them to perform actions in each window.

## Window Handles

**driver.getWindowHandles();**

The driver.getWindowHandles(); method in Selenium serves a key role in managing multiple windows and tabs within your automation scripts:

1. **Retrieving all window handles:** This method returns a Set containing unique strings representing the identifiers of all windows currently open with the Selenium WebDriver. Think of them as individual "license plate numbers" for each window.

2. **Identifying and switching windows:** By understanding these handles, you can use methods like driver.switchTo().window(handle) to switch focus to specific windows. This allows you to interact with different windows sequentially or perform parallel actions in different contexts.

3. **Understanding window relationships:** In scenarios with multiple windows or tabs, getWindowHandles() helps you analyze and potentially track their relationships. You can store the handles in a data structure like a list or set and manipulate them based on your automation logic.