Selenium WebDriver with C# Cheatsheet

A comprehensive cheat sheet for using Selenium WebDriver with C#, covering setup, basic operations, element interactions, waits, and best practices. Ideal for beginners and intermediate users.

Setup and Basic Operations

Installation via NuGet

Install Selenium WebDriver and browser-specific drivers (e.g., ChromeDriver) using NuGet Package Manager in Visual Studio.

```
Install-Package Selenium.WebDriver
Install-Package Selenium.WebDriver.ChromeDriver
```

Ensure the correct version of the ChromeDriver is installed to match your Chrome browser version.

Project Setup

1. Create a new C# project in Visual Studio (e.g., Console App or Test Project).
2. Add references to the installed Selenium WebDriver packages.

Basic Browser Operations

Opening a browser:	`using OpenQA.Selenium;`
	`using OpenQA.Selenium.Chrome;`
	
	`IWebDriver driver = new ChromeDriver();`

Navigating to a URL:	`driver.Navigate().GoToUrl("https://www.example.com");`
Closing the browser:	`driver.Quit();`
Maximize window:	`driver.Manage().Window.Maximize();`
Refresh page:	`driver.Navigate().Refresh();`

Finding and Interacting with Elements

Finding Elements

By ID:	`IWebElement element = driver.FindElement(By.Id("elementId"));`
By Name:	`IWebElement element = driver.FindElement(By.Name("elementName"));`
By XPath:	`IWebElement element = driver.FindElement(By.XPath("//div[@class='example']"));`
By CSS Selector:	`IWebElement element = driver.FindElement(By.CssSelector(".exampleClass"));`
By Tag Name:	`IWebElement element = driver.FindElement(By.TagName("h1"));`
By Class Name:	`IWebElement element = driver.FindElement(By.ClassName("example"));`
Finding Multiple Elements:	`IList<IWebElement> elements = driver.FindElements(By.XPath("//div"));`

Interacting with Elements

Clicking an element:	`element.Click();`
Sending keys (typing text):	`element.SendKeys("Hello, Selenium!");`
Clearing text:	`element.Clear();`
Getting an attribute value:	`string attributeValue = element.GetAttribute("value");`
Getting text of an element:	`string elementText = element.Text;`

Waits and Handling Popups

Implicit Waits

Implicit waits tell WebDriver to wait for a certain amount of time when trying to find an element if it is not immediately available.

```
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
```

Note: Avoid mixing implicit and explicit waits.

Explicit Waits

Explicit waits allow you to wait for a certain condition to be met before proceeding.

```
using OpenQA.Selenium.Support.UI;

WebDriverWait wait = new WebDriverWait(driver,
TimeSpan.FromSeconds(10));
IWebElement element =
wait.Until(ExpectedConditions.ElementIsVisible(By.Id("elementId"
)));
```

Common `ExpectedConditions`:

- `ElementIsVisible`
- `ElementToBeClickable`
- `PresenceOfAllElementsLocatedBy`
- `TitleContains`
- `AlertIsPresent`

Handling Alerts

Switching to an alert:	`IAlert alert = driver.SwitchTo().Alert();`
Accepting an alert:	`alert.Accept();`
Dismissing an alert:	`alert.Dismiss();`
Getting alert text:	`string alertText = alert.Text;`

Handling Iframes

Switching to an iframe by index:	`driver.SwitchTo().Frame(0);`
Switching to an iframe by name or ID:	`driver.SwitchTo().Frame("iframeName");`
Switching back to the main document:	`driver.SwitchTo().DefaultContent();`

Advanced Interactions, Screenshots, and Test Frameworks

Advanced Interactions

Hovering over an element:	`Actions actions = new Actions(driver); actions.MoveToElement(element).Perform();`
Right-clicking an element:	`Actions actions = new Actions(driver); actions.ContextClick(element).Perform();`
Drag and drop:	`Actions actions = new Actions(driver); actions.DragAndDrop(sourceElement, targetElement).Perform();`
Sending keyboard actions:	`element.SendKeys(Keys.Enter);`

Screenshots

Taking a screenshot:

```
ITakesScreenshot screenshotDriver = driver as ITakesScreenshot;
Screenshot screenshot = screenshotDriver.GetScreenshot();
screenshot.SaveAsFile("screenshot.png",
ScreenshotImageFormat.Png);
```

Test Framework Integration (NUnit)

Basic NUnit setup:

```
using NUnit.Framework;

[TestFixture]
public class MyTestClass
{
    IWebDriver driver;

    [SetUp]
    public void Setup()
    {
        driver = new ChromeDriver();
    }

    [Test]
    public void MyTestMethod()
    {
        driver.Navigate().GoToUrl("https://www.example.com");
        Assert.AreEqual("Example Domain", driver.Title);
    }

    [TearDown]
    public void TearDown()
    {
        driver.Quit();
    }
}
```

- `[SetUp]` runs before each test.
- `[Test]` marks a method as a test method.
- `[TearDown]` runs after each test.

Best Practices

Page Object Model (POM)

The Page Object Model (POM) is a design pattern that creates an object repository for web elements. Each page in the application has its own class, which defines the elements and methods to interact with them.

Benefits:

- Improves code reusability.
- Reduces code duplication.
- Increases maintainability.

Example:

```
public class LoginPage
{
    private IWebDriver driver;
    private IWebElement usernameField =>
driver.FindElement(By.Id("username"));
    private IWebElement passwordField =>
driver.FindElement(By.Id("password"));
    private IWebElement loginButton =>
driver.FindElement(By.Id("login"));

    public LoginPage(IWebDriver driver)
    {
        this.driver = driver;
    }

    public void EnterUsername(string username)
    {
        usernameField.SendKeys(username);
    }

    public void EnterPassword(string password)
    {
        passwordField.SendKeys(password);
    }

    public void ClickLogin()
    {
        loginButton.Click();
    }
}
```

Configuration Management

Store configuration settings (e.g., browser type, base URL) in a configuration file (e.g., `appsettings.json`) or environment variables.

```
// Example using ConfigurationManager
string browserType =
ConfigurationManager.AppSettings["BrowserType"];
```

Reusability and Maintainability Tips

- Write reusable methods for common actions (e.g., login, navigation).
- Use descriptive names for variables and methods.
- Add comments to explain complex logic.
- Keep test methods short and focused.
- Regularly review and refactor code.