

Test Automation & Advanced Selenium

Lesson 5: Testing Web
Applications Using Web Driver
API

Lesson Objectives

- Writing first Web Driver Test
- Locating UI Elements-Developers Tools
- Navigation API
 - get
 - Navigate
- Interrogation API
 - getTitle
 - getCurrentUrl
 - getPageSource



Lesson Objectives

- WebElement API
 - findElement & findElements
 - By
 - id
 - xpath
 - cssSelector
 - className
 - linkText
 - name
 - tagName
 - partialLinkText



Lesson Objectives (Contd.)

- WebElement API
 - click
 - clear
 - sendKeys
 - submit
 - Select – selectByVisibleText etc.
 - getText
 - getAttribute
- Handling Popup Dialogs and Alerts



Lesson Objectives (Contd.)

- Windows
 - getWindowHandle and getWindowHandles
 - switchTo
 - Manage
- Alerts
 - switchTo
 - dismiss
 - Accept
- Why synchronization is important



Lesson Objectives (Contd.)

- Using Explicit & Implicit Wait
 - Expected Condition & Expected Conditions
 - WebDriverWait
 - ImplicitlyWait
 - pageLoadTimeout
- JavaScript Executor



5.1: Testing Web Applications Using Web Driver API

Writing first Web Driver Test(Java)

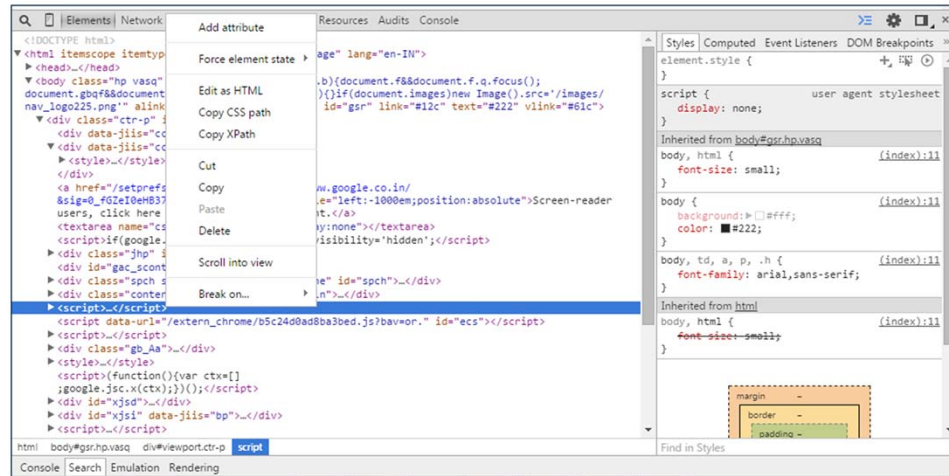
```
package org.openqa.selenium.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Selenium2Example {
    public static void main(String[] args) {
        // Create a new instance of the Firefox driver
        WebDriver driver = new FirefoxDriver();
        // And now use this to visit Google
        driver.get("http://www.google.com");
        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));
        // Enter something to search for
        element.sendKeys("Cheese!");
        // Now submit the form. WebDriver will find the form for us from the element
        element.submit();
        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());
        //Close the browser
        driver.quit();
    }
}
```

5.1: Testing Web Applications Using Web Driver API

Locating UI Elements-Developers Tools



5.1: Testing Web Applications Using Web Driver API

Navigation API

- `driver.get("URL")`
 - Required to navigate to a page
 - E.g.: `driver.get("http://www.google.com");`
 - WebDriver will wait until the page has fully loaded before returning control to your test or script
 - to ensure page is fully loaded then wait commands can be used
- `driver.navigate().to("URL")`
 - E.g.: `driver.navigate().to("http://www.google.com");`
 - Other Navigate commands
 - `driver.navigate().refresh();`
 - `driver.navigate().forward();`
 - `driver.navigate().back();`



Copyright © Capgemini 2015. All Rights Reserved 9

Single-Page Applications are an exception to this.

The difference between these two methods comes not from their behavior, but from the behavior in the way the application works and how browser deal with it. `navigate().to()` navigates to the page by changing the URL like doing forward/backward navigation.

Whereas, `get()` refreshes the page to changing the URL.

So, in cases where application domain changes, both the method behaves similarly. That is, page is refreshed in both the cases. But, in single-page applications, while `navigate().to()` do not refreshes the page, `get()` do.

Moreover, this is the reason browser history is getting lost when `get()` is used due to application being refreshed.

5.1: Testing Web Applications Using Web Driver API

Interrogation API

- `driver.getTitle()`
 - Get the title of the current page
- `driver.getCurrentUrl()`
 - Get the current URL of the browser
- `driver.getPageSource()`
 - Get the source code of the page

- Syntax:

```
public void testTitleReliability() {  
    driver.get("https://www.google.com");  
    boolean title = driver.getTitle().contains("Google");  
    if(title)  
        String currentURL = driver.getCurrentUrl();  
        (If you want to verify a particular text is present or not on the page, do as below)  
        boolean b = driver.getPageSource().contains("your text");  
        System.out.println("Expected title is present ");  
    else if(!title)  
        System.out.println(" Expected title is not present");  
}
```

5.1: Testing Web Applications Using Web Driver API

WebElement API

- **findElement:**
 - Used to locate single element and return WebElement object of first occurrences element on web page
 - If element not found, throw s exception NoSuchElementException
 - Syntax: findElement(By by)

- **Example:**

```
WebElement element = driver.findElement(By.id("Home"));  
element.click();
```

5.1: Testing Web Applications Using Web Driver API

WebElement API

- **findElements:**

- Used to find multiple element on webpage, e.g.: count total number of row in table
- Returns List of WebElement object of all occurrences of element
- If element not found, returns empty List of WebElement object
- Syntax: List element = findElements(By by)

- **Example:**

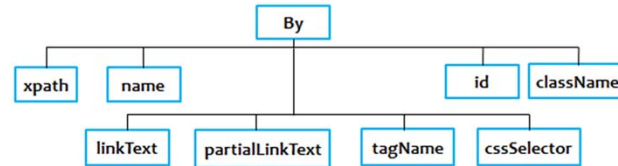
```
List {WebElement} element = driver.findElement(By.xpath("//table/tr"));
```

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

- By:

- A collection of factory functions for creating webdriver.Locator instances



- By id: Locates an element by its ID
 - Syntax: `driver.findElement(By.id("element id"))`
- By className: Locates elements that have a specific class name
 - Syntax : `driver.findElement(By.className("element class"))`
- By name: Locates elements whose name attribute has the given value
 - Syntax : `driver.findElement(By.name("element name"))`

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

- By XPath: Locates elements matching a XPath selector.
 - For example, given the selector "//div", WebDriver will search from the document root regardless of whether the locator was used with a WebElement
 - Syntax: `driver.findElement(By.xpath("xpath expression"))`
- By linkText :Locates link elements whose visible text matches the given string
 - Syntax : `driver.findElement(By.link("link text"))`
- By partialLinkText: Locates link elements whose visible text contains the given substring
 - Syntax : `driver.findElement(By.partialLinkText("link text"))`
- By tagName: Locates elements with a given tag name.
 - The returned locator is equivalent to using the `getElementsByTagName` DOM function
 - Syntax : `driver.findElement(By.tagName("element html tag name"))`
- By CSS Selector: Locates elements with a given tag name.
 - Syntax : `driver.findElement(By.cssSelector("css selector"))`



Copyright © Capgemini 2015. All Rights Reserved 14

CSS selectors for Selenium with example

When we don't have an option to choose Id or Name, we should prefer using CSS locators as the best alternative.

CSS is "Cascading Style Sheets" and it is defined to display HTML in structured and colorful styles are applied to webpage.

Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document. Visit to know more W3.Org/Css/selectors

CSS has more Advantage than Xpath

CSS is much more faster and simpler than the Xpath.

In IE Xpath works very slow, where as Css works faster when compared to Xpath.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

- Click():
 - For Example: Login button is available on login screen
 - Syntax:
 - `WebElement click = driver.findElement(By.xpath("//*[@id='btnLogOn']"));`
`click.click();`
- Scenarios where Click() is used:
 - "Check / Uncheck " a checkbox
 - Select a radio button



Copyright © Capgemini 2015. All Rights Reserved 15

Click():

Click a link / button:

To click on an object through webdriver first we need to find out which locator we are going to use. Is it ID, name, xpath, or css? For this purpose we can utilize firebug / xpath checker to find out is there any id / name exists for the object we are going to perform action upon. Then write the code as below:

```
driver.findElement(By.xpath("//a[@href='CustomerInfo.htm']")).click();
```

In the above line of code "driver" could be FirefoxDriver, InternetExplorerDriver, ChromeDriver, HtmlUnitDriver, etc. On one of these browsers we are going to find an element and then click as per the code.

findElement is an API provided by the webdriver which requires argument "By.xpath". The "xpath" can be replaced by one of the below methods if we need to identify the element with any other attributes such as css, name, classname, etc.

"Check / Uncheck " a checkbox

To "Check / Uncheck" a checkbox, the object needs to be identified using findElement method and then just click. To find out whether the checkbox is checked or not utilize the method – element.isSelected()

```
WebElement kanclick = driver.findElement(By.name("kannada"));  
kanclick.click();  
System.out.println(kanclick.isSelected());
```

Above code snippet will first click the checkbox named kannada and then verifies whether it is clicked or not.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

- **Clear():**
 - Function sets the value property of the element to an empty string ("")
- **Syntax:**

```
driver.findElement(By.xpath("//*[@id='textBox']")).clear();
```
- **SendKeys():**
 - Method is used to simulate typing into an element, which may set its value
- **Syntax:**

```
driver.findElement(By.id("NameTextBox")).sendKeys("Rahul");
```



Copyright © Capgemini 2015. All Rights Reserved 15

Click():

Click a link / button:

To click on an object through webdriver first we need to find out which locator we are going to use. Is it ID, name, xpath, or css? For this purpose we can utilize firebug / xpath checker to find out if there is any id / name exists for the object we are going to perform action upon. Then write the code as below:

```
driver.findElement(By.xpath("//a[@href='CustomerInfo.htm']")).click();
```

In the above line of code "driver" could be FirefoxDriver, InternetExplorerDriver, ChromeDriver, HtmlUnitDriver, etc. On one of these browsers we are going to find an element and then click as per the code.

findElement is an API provided by the webdriver which requires argument "By.xpath". The "xpath" can be replaced by one of the below methods if we need to identify the element with any other attributes such as css, name, classname, etc.

"Check / Uncheck" a checkbox

To "Check / Uncheck" a checkbox, the object needs to be identified using findElement method and then just click. To find out whether the checkbox is checked or not utilize the method – element.isSelected()

```
WebElement kanclick = driver.findElement(By.name("kannada"));
kanclick.click();
System.out.println(kanclick.isSelected());
```

Above code snippet will first click the checkbox named kannada and then verifies whether it is clicked or not.



Select a radio button

Follow the same steps which are used in Checkbox to select a radio button and then verify the status using `isSelected()` method.

```
WebElement gender =  
driver.findElement(By.xpath("//input[@name='male']"));  
gender.click();  
System.out.println(gender.isSelected());
```

`Clear();`

The `clear()` method executes an "Automation Atom", which is a JavaScript function intended to provide the smallest basic unit of automation functionality for a browser. In the case of `clear()`, that function sets the value property of the element to an empty string (""), then fires the `onchange` event on the element. The atoms function you're interested in is `bot.action.clear()`

`click`

`clear`

`sendKeys`

`submit`

`Select – selectByVisibleText` etc.

`getText`

`getAttribute`

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

■ SendKeys():

- Scenarios where sendKeys() is used:
 - Sending special characters (Enter , F5, Ctrl, Alt etc..)
 - Key events to WebDriver
 - Uploading a file

Syntax:

```
//Sending Ctrl+A
driver.findElement(By.xpath("//body")).sendKeys(Keys.chord(Keys.CONTROL, "a"));

//Sending pagedown key from keyboard
driver.findElement(By.id("name")).sendKeys(Keys.PAGE_DOWN);

//Sending space key
driver.findElement(By.id("name")).sendKeys(Keys.SPACE);
```

■ Submit():

- If form has submit button which has type = "submit" instead of type = "button" then .submit() method will work `<input type="submit" value="Submit">`
- If button is not inside <form> tag then .submit() method will not work.

Syntax:

```
driver.findElement(By.xpath("//input[@name='Company']")).submit();
```



Copyright © Capgemini 2015. All Rights Reserved 18

When to use .click() method

You can use .click() method to click on any button of software web application. Means element's type = "button" or type = "submit", .click() method will work for both.

When to use .submit() method

If you will look at firebug view for any form's submit button then always its type will be "submit". In this case, .submit() method is a very good alternative of .click() method.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)■ **Select:**

- WebDriver's support classes
- Used to work with Dropdowns

- Method Name: `selectByIndex`
- Syntax: `select.selectByIndex(Index);`

- Method Name: `selectByValue`
- Syntax: `select.selectByValue(Value);`

- Method Name: `selectByVisibleText`
- Syntax: `select.selectByVisibleText(Text);`

```
<html>
<head>
<title>Select Example by Index value</title>
</head>
<body>
<select name="Mobiles"><option value="0" selected> Please select</option>
<option value="1">iPhone</option>
<option value="2">Nokia</option>
<option value="3">Samsung</option>
<option value="4">HTC</option>
<option value="5">BlackBerry</option>
</select>
</body>
</html>
```

```
<html>
<head>
<title>Select Example by Value</title>
</head>
<body>
<p>Which mobile device do you like most?</p>
<select name="Mobiles"><option selected> Please select</option>
<option value="iphone">iPhone</option>
<option value="nokia">Nokia</option>
<option value="samsung">Samsung</option>
<option value="htc">HTC</option>
<option value="blackberry">BlackBerry</option>
</select>
</body>
</html>
```



Copyright © Capgemini 2015. All Rights Reserved 19

Method Name: `selectByIndex`

Purpose: To Select the option based on the index given by the user.

There is an attribute called "values" which will have the index values.

Example:

```
WebElement element=driver.findElement(By.name("Mobiles"));
Select se=new Select(element);
se.selectByIndex(1);
```

Method Name: `selectByValue`

Purpose: To Select the options that have a value matching with the given argument by the user.

Example:

```
WebElement element=driver.findElement(By.name("Mobiles"));
Select se=new Select(element);
se.selectByValue("nokia");
```

Method Name: `selectByVisibleText`

Purpose: To Select all options that display text matching the given argument. It will not look for any index or value, it will try to match the VisibleText (which will display in dropdown)

Example:

```
WebElement element=driver.findElement(By.name("Mobiles"));
Select se=new Select(element);
se.selectByVisibleText("HTC");
```



Method Name: `deselectByIndex`

Purpose: To Deselect the option at the given index. The user has to provide the value of index.

Example:

```
se.deselectByIndex(1);
```

Method Name: `deselectByValue`

Purpose: To Deselect all options that have a value matching the given argument.

Example:

```
se.deselectByValue("nokia");
```

Method Name: `deselectByVisibleText`

Purpose: To Deselect all options that display text matching the given argument.

Example:

```
se.deselectByVisibleText("HTC");
```

Method Name: `deselectAll`

Purpose: To Clear all selected entries. This works only when the SELECT supports multiple selections. It throws `NotImplemented eError` if the "SELECT" does not support multiple selections. In select it mandatory to have an attribute `multiple="multiple"`

Example:

```
se.deselectAll();
```

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

■ getText():

- Get the text content from a DOM-element found by given selector
- Make sure the element you want to request the text from is interact able otherwise empty string is returned

Syntax:

```
WebElement TxtBoxContent = driver.findElement(By.id("WebElementID"));
TxtBoxContent.getText();
```

■ getAttribute():

- getText() will only get the inner text of an element
- To get the value, you need to use getAttribute("attribute name")
- Attribute name can be class, id, name, status etc

```
<button name="btnK" id="gbqfba" aria-label="Google Search" class="gbqfba"><span id="gbqfsa">Google Search</span></button>
```

Syntax:

```
WebElement TxtBoxContent = driver.findElement(By.id(WebElementID));
System.out.println("Printing " + TxtBoxContent.getAttribute("class"));
```



Copyright © Capgemini 2015. All Rights Reserved 21

When to use .click() method

You can use .click() method to click on any button of software web application. Means element's type = "button" or type = "submit", .click() method will work for both.

When to use .submit() method

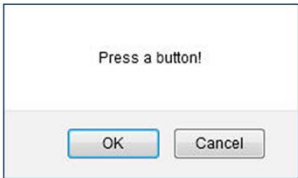
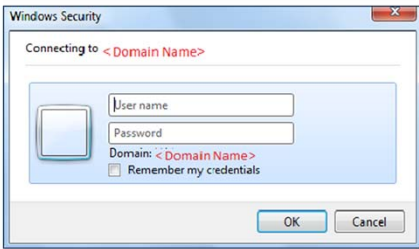
If you will look at firebug view for any form's submit button then always its type will be "submit". In this case, .submit() method is a very good alternative of .click() method.


5.1: Testing Web Applications Using Web Driver API

Handling Popup Dialogs and Alerts

Two types of alerts:

- Windows based alert pop ups
 - Selenium will not be able to recognize it, since it is an OS-level dialog
- Web based alert pop ups
 - Can be Alert box/ Pop up box/ confirmation Box/ Prompt/ Authentication Box
 - Alert interface gives us following methods to deal with the alert
 - `accept()` : To accept the alert
 - `dismiss()` : To dismiss the alert
 - `getText()` : To get the text of the alert
 - `sendKeys()` : To write some text to the alert



 CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

Simple alert

Simple alerts just have a OK button on them. They are mainly used to display some information to the user.

Confirmation Alert

This alert comes with an option to accept or dismiss the alert. To accept the alert you can use `Alert.accept()` and to dismiss you can use the `Alert.dismiss()`.

Prompt Alerts

In prompt alerts you get an option to add text to the alert box. This is specifically used when some input is required from the user. We will use the `sendKeys()` method to type something in the Prompt alert box.

5.1: Testing Web Applications Using Web Driver API

Windows

- Multiple windows are handled by switching the focus from one window to another

- Syntax:**

- // Opening site**

```
driver.findElement(By.xpath("//img[@alt='SeleniumMasterLogo']")).click();
```

- // Storing parent window reference into a String Variable**

```
String Parent_Window = driver.getWindowHandle();
```

- // Switching from parent window to child window**

```
for (String Child_Window : driver.getWindowHandles())
{
```

```
    driver.switchTo().window(Child_Window);
```

- // Performing actions on child window**

```
    driver.findElement(By.id("dropdown_txt")).click();
```

```
    driver.findElement(By.xpath("//*[@id='anotherItemDiv']")).click();
```

```
}
```

- //Switching back to Parent Window**

```
driver.switchTo().window(Parent_Window);
```

- //Performing some actions on Parent Window**

```
driver.findElement(By.className("btn_style")).click();
```



Copyright © Capgemini 2015. All Rights Reserved 23

Steps for understanding window handling:

Window A has a link "Link1" and we need to click on the link (click event).
Window B displays and we perform some actions.

The entire process can be fundamentally segregated into following steps:

Step 1 : Clicking on Link1 on Window A

A new Window B is opened.

Step 2 : Save reference for Window A

Step 3 : Create reference for Window B

Step 4 : Move Focus from Window A to Window B

Window B is active now

Step 5 : Perform Actions on Window B

Complete the entire set of Actions

Step 6 : Move Focus from Window B to Window A

Window A is active now

5.1: Testing Web Applications Using Web Driver API

Alerts

- Present in the **org.openqa.selenium.Alert** package
- Syntax:

```
Alert simpleAlert = driver.switchTo().alert(); //switch from main window to an alert
String alertText = simpleAlert.getText(); //To get the text present on alert
System.out.println("Alert text is " + alertText);
```

//Simple alert

```
simpleAlert.accept(); //To click on 'Ok'/'Yes' on Alert
```

OR

//Confirmation Alert

```
simpleAlert.dismiss(); //To click on 'Cancel'/'No' on Alert
```

OR

//Prompt Alerts

```
simpleAlert.sendKeys("Accepting the alert"); //Send some text to the alert
```



Copyright © Capgemini 2015. All Rights Reserved 24

Alerts are different from regular windows. The main difference is that alerts are blocking in nature. They will not allow any action on the underlying webpage if they are present. So if an alert is present on the webpage and you try to access any of the element in the underlying page you will get following exception:
UnhandledAlertException: Modal dialog present

5.1: Testing Web Applications Using Web Driver API

Why synchronization is important

- “Mechanism which involves more than one components to work parallel with each other”
- Every time user performs an operation on the browser, one of the following happens:
 - The request goes all the way to server and entire DOM is refreshed when response comes back
 - The request hits the server and only partial DOM gets refreshed (Ajax requests or asynchronous JavaScript calls)
 - The request is processed on the client side itself by JavaScript functions
- So if we think about the overall workflow, there is a need of certain synchronization that happens between the client(aka. browser) and the server (the url)

5.1: Testing Web Applications Using Web Driver API

Using Explicit & Implicit Wait

- Implicit Wait
- Element Synchronization
 - Default element existence timeout can be set
 - Below statement will set the default object synchronization timeout as 20
 - Means that selenium script will wait for maximum 20 seconds for element to exist
 - If Web element does not exist within 20 seconds, it will throw an exception
- `driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);`



Copyright © Capgemini 2015. All Rights Reserved 25

Synchronization can be classified into two categories:

1. Unconditional
2. Conditional Synchronization

Unconditional :

In this we just specify timeout value only. We will make the tool to wait until certain amount of time and then proceed further.

Examples: `Wait()` and `Thread.Sleep()`;

The main disadvantage for the above statements are, there is a chance of unnecessary waiting time even though the application is ready.

The advantages are like in a situation where we interact for third party systems like interfaces, it is not possible to write a condition or check for a condition. Here in this situations, we have to make the application to wait for certain amount of time by specifying the timeout value.

Conditional Synchronization:

We specify a condition along with timeout value, so that tool waits to check for the condition and then come out if nothing happens.

It is very important to set the timeout value in conditional synchronization, because the tool should proceed further instead of making the tool to wait for a particular condition to satisfy.

In Selenium we have implicit Wait and Explicit Wait conditional statements

5.1: Testing Web Applications Using Web Driver API

Using Explicit & Implicit Wait

- Explicit Wait
 - Specific condition synchronization
- Instruct selenium to wait until element is in expected condition
 - Syntax:

```
WebDriverWait w = new WebDriverWait(driver,20);
w.ignoring(NoSuchElementException.class);
WebElement P = null;
//below statement will wait until element becomes visible
P=w.until(ExpectedConditions.visibilityOfElementLocated(By.id("x")));
//below statement will wait until element becomes clickable.
P= w.until(ExpectedConditions.elementToBeClickable(By.id("ss")));
```



Copyright © Capgemini 2015. All Rights Reserved 27

Page Load Synchronization:

Default page navigation timeout can be set.

Below statement will set the navigation timeout as 50

Means that selenium script will wait for maximum 50 seconds for page to load

If page does not load within 50 seconds, it will throw an exception

Syntax

```
driver.manage().timeouts().pageLoadTimeout(50, TimeUnit.SECONDS);
```

5.1: Testing Web Applications Using Web Driver API

JavaScript Executor

- An interface which provides mechanism to execute Javascript through selenium driver
- Provides "executeScript" & "executeAsyncScript" methods, to run JavaScript in the context of the currently selected frame or window
- Used to enhance the capabilities of the existing scripts by performing Javascript injection into our application under test
- Package
`import org.openqa.selenium.JavascriptExecutor;`

Syntax

```
JavascriptExecutor js = (JavascriptExecutor) driver;  
js.executeScript(Script,Arguments);
```

script - The JavaScript to execute

Arguments - The arguments to the script.(Optional)

5.1: Testing Web Applications Using Web Driver API

JavaScript Executor(Scenarios)

- How to generate Alert Pop window in selenium?

- Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver;  
Js.executeScript("alert('hello world');");
```

- How to click a button in Selenium WebDriver using JavaScript?

Code: JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("arguments[0].click();", element);

- How to refresh browser window using Javascript ?

- Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver;  
driver.executeScript("history.go(0)");
```



5.1: Testing Web Applications Using Web Driver API

JavaScript Executor(Scenarios)

- How to get innertext of the entire webpage in Selenium?

Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver;  
string sText = js.executeScript("return  
document.documentElement.innerText;").toString();
```

- How to get the Title of our webpage ?

Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver;  
string sText = js.executeScript("return document.title;").toString();
```

5.1: Testing Web Applications Using Web Driver API

JavaScript Executor(Scenarios)

- How to perform Scroll on application using Selenium?

Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver; //Vertical scroll -  
down by 50 pixels js.executeScript("window.scrollTo(0,50)");
```

- Note: for scrolling till the bottom of the page we can use the code:

```
js.executeScript("window.scrollTo(0,document.body.scrollHeight)");
```

5.1: Testing Web Applications Using Web Driver API

JavaScript Executor(Scenarios)

- How to click on a Sub Menu which is only visible on mouse hover on Menu?

Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver;  
//Hover on Automation Menu on the Menu Bar  
js.executeScript("$('.ul.menus.menu-secondary.sf-js-enabled.sub-menu li').hover()");
```

- How to navigate to different page using Javascript?

Code:

```
JavascriptExecutor js = (JavascriptExecutor)driver; //Navigate to  
new Page js.executeScript("window.location =  
'https://www.facebook.com/uftHelp'");
```


Summary

- In this lesson, you have learnt
 - Multiple windows are handled by switching the focus from one window to another.
 - By is a collection of factory functions for creating webdriver.Locator instances.
 - Alert contains methods for dismissing, accepting, inputting, and getting text from alert prompts.
 - Explicit synchronization points are inserted in the script using WebDriverWait class.
 - Each and every time when there is need to match speed of the application and speed of test execution we have to use thread.sleep().
 - The implicit wait will not wait for the entire time that is specified, rather it will only wait, until the entire page is loaded.



Add the notes here.

Summary

- In this lesson, you have learnt
 - An interface which provides mechanism to execute Javascript through selenium driver
 - Used to click on a Sub Menu which is only visible on mouse hover on Menu
 - Used to to get innertext of the entire webpage in Selenium
 - Used to navigate to different page using Javascript
 - Used to click a button in Selenium WebDriver using JavaScript



Add the notes here.

Review Question

■ Question 1

- Select which is NOT an Explicit Wait
 - VisibilityOfElementLocated
 - ElementToBeClickable
 - PageLoadTimeout
 - None of the above



■ Question 2: True/False

- The syntax is correct:
- Syntax : `driver.findElement(By. PartialLinkText("link text"));`

■ Question 3: Fill in the Blanks

- `findElements` is used to find _____ element on webpage

Review Question

- Question 4: True/False

- The syntax is correct:

Syntax :

JavascriptExecutor js = (JavascriptExecutor)driver;

- Question 5: Fill in the Blanks

- An interface which provides mechanism to execute _____ through selenium driver



Answer 1:

True

Answer 2:

Javascript