**WEBDRIVER ADVANCED USAGE**

**WAITS:**In Selenium, "Waits" play an important role in executing tests. Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an "**ElementNotVisibleException**" exception. Using Selenium Waits, we can resolve this problem.

**TYPES OF WAITS:**Thread Wait  
Implicit Wait  
Explicit Wait  
Fluent Wait

**THREAD WAIT:**This is the simplest of wait that can be used in the script, It is always applied at **individual level** and is discouraged to be used in Automation script as this type of wait is **STATIC WAIT** (Execution will be halted for the specific time even if the element is found/located on the web page)

Syntax:  
Thread.sleep(“Specify the time to wait”)

**IMPLICIT WAIT:**The Implicit Wait in Selenium is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

This type of wait is **DYNAMIC WAIT-** If the element is located before the specified time then it will ignore the rest of the time and will move ahead with the execution.

To declare implicit wait:

**Syntax**:

driver.manage().timeouts().implicitlyWait(“Specify int value of time”, TimeUnit.SECONDS);

For using implicit waits in the test scripts, it’s mandatory to import the following package.

import java.util.concurrent.TimeUnit;

Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

**EXPLICIT WAIT:**The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception. It is an intelligent kind of wait, but it can be **applied only for specified elements**. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.

This type of wait is **DYNAMIC WAIT-** If the element is located before the specified time then it will ignore the rest of the time and will move ahead with the execution.

Once we declare explicit wait we have to use "**ExpectedConditions**".

WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));

**Import packages:**

import org.openqa.selenium.support.ui.ExpectedConditions  
import org.openqa.selenium.support.ui.WebDriverWait

The following are the Expected Conditions that can be used in Selenium Explicit Wait

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvaliableAndSwitchToIt()
6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()
13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. visibilityOfElementLocated()

|  |  |
| --- | --- |
| **Implicit Waits** | **Explicit Waits** |
| 1. Implicit Wait time is applied to all the elements in the script. You have predefines keyword for implicit wait. | 1. Explicit Wait time is applied only to those elements which are specified by the user. You do not have predefined keyword for explicit wait |
| 2. In Implicit Wait, we need **not** specify “ExpectedConditions” on the element to be located | 2. In Explicit Wait, we need to specify “ExpectedConditions” on the element to be located |
| 3. It is recommended to use when the elements are located with the time frame specified in implicit wait | 3. It is recommended to use when the elements are taking a long time to load and also for verifying the property of the element like (visibilityOfElementLocated, elementToBeClickable, elementToBeSelected) |

**FLUENT WAIT:**The **Fluent Wait in Selenium** is used to define maximum time for the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception. It checks for the web element at regular intervals until the object is found or timeout happens.

**Frequency:**Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

**Syntax:**Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(Duration.ofSeconds(SECONDS))

.pollingEvery(Duration.ofSeconds(SECONDS))

.ignoring(NoSuchElementException.class);

WebElement foo= wait.until(new Function<WebDriver, WebElement>(){

public WebElement apply(WebDriver driver) {

return driver.findElement(By.id("foo"));

}

}

**Page Load timeout :**

We can **set the amount of time to wait for a page load to complete** before throwing an error.

driver.manage().timeouts().pageLoadTimeout(20, TimeUnit.SECONDS);

Once added in the script, the WebDriver instance waits for 20 seconds for every page to get loaded before throwing an exception. If the page is not loaded in 20 seconds of time, then it throws TimedOutException at run time.

**Script timeout :**

We can set the amount of time **to wait for an asynchronous script to finish execution** before throwing any error.

Example :

driver.manage().timeouts().setScriptTimeout(20, TimeUnit.SECONDS);

Once added in the script, the WebDriver instance waits for 20 seconds for every asynchronous script to get executed on the web page before throwing an exception.

**ALERTS AND POP UP**

## **What is an Alert?**

When you test a web application using Selenium and try logging in to a website but, fail to add in the mandatory fields like mail ID or the password, the system throws an alert.



Alerts are basically an interface between the current web page and UI. It can also be defined as a small message box which displays an on-screen notification to give the user some kind of information or ask for permission to perform a certain kind of operation. It may be also used for warning purpose.

Now, let’s see why you should use this alert while testing an application.

## **Why is an Alert used in Selenium?**

Alert is basically used to display a warning message. It is a pop-up window that comes up on the screen. There are many user actions that can result in an alert on the screen. For example, if you click on a button that displays a message or maybe when you entered a form, HTML page asked you for some extra information. This is an alert.

Alerts are the unhandled window when you navigate for the first time to a webpage.

## **Types of Alerts in Selenium**

There are mainly 3 types of Alerts, namely:

1. Simple Alert
2. Prompt Alert
3. Confirmation Alert

Let me explain them in detail:

* **Simple Alert:**A simple alert just has an***OK*** button on them. They are mainly used to display some information to the user. The very first alert on the test page is a simple alert. The following code will read the text from the Alert and then accept the alert.

Alert simpleAlert = driver.switchTo().alert();

****

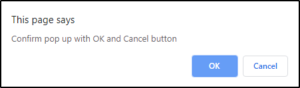
* **Prompt Alert**: In prompt alerts, you get an option to add a text field to the alert box. This is specifically used when some input is required from the user. You can use the ***sendKeys()***method to type some text in the Prompt alert box.

promptAlert.sendKeys("Accepting the alert");

****

* **Confirmation Alert**: This type of alert comes with an option to accept or dismiss the alert. In order to accept the alert, you can use the Alert.accept()and to dismiss, use Alert.dismiss()

confirmationAlert.dismiss();



Once you are clear with the different types of alerts, it will be easy to understand how to handle these alerts.

**PRACTICE URL:** <https://demoqa.com/alerts>

## **WINDOW AND WINDOW HANDLES PRACTICE URL: https://demoqa.com/browser-windows** **What is a window in Selenium?**

A window in any browser is the main webpage on which the user is landed after hitting a link/URL. Such a window in Selenium is referred to as the ***parent window*** also known as the ***main window*** which opens when the Selenium WebDriver session is created and has all the focus of the WebDriver.

### How do we identify parent window and child windows?

When a user hits a URL, a webpage opens. This main page is the ***parent window***  i.e. the main window on which the user has currently landed and will perform any operation. This is the same webpage that will open when our Selenium automation script will execute. All the windows which will open inside your main window will be termed as ***child windows***.

LET’S DO SOME PRACTICAL - **https://demoqa.com/browser-windows**

**Screenshot:**

Screenshot is an essential element for testing to have a look at what happened after the execution.

**public** **static** **void** takeScreenshotAtEndOfTest() **throws** IOException {

File scrFile = ((TakesScreenshot) *driver*).getScreenshotAs(OutputType.***FILE***);

String currentDir = System.*getProperty*("user.dir");

FileUtils.*copyFile*(scrFile, **new** File(currentDir + "/screenshots/" + System.*currentTimeMillis*() + ".png"));

}