**How to wait for an AJAX call to complete in WebDriver**

Many web applications built on Web 2.0 technology, contain [AJAX](https://en.wikipedia.org/wiki/Ajax_(programming)) calls. AJAX calls don’t refresh the whole page, only a certain part of a page is refreshed. When an AJAX call is made, while the page is waiting for a response from the server, a waiting icon appears on the page to inform the user, the page is waiting for information, this could be either a rotating circle, or a loading horizontal bar, etc…

WebDriver is clever enough to wait for the whole page to load before doing any action and without the user having to specify a wait for page to load. However, in case of AJAX calls, because the whole page is not refreshed, WebDriver has no way of knowing something is happening.

In normal cases, the waiting icon has a css style attribute such as

<div class="rotating-icon"; style="display: none"></div>

for when the AJAX call is not started or already completed, and

<div class="rotating-icon"; style="display: block"></div>

when the section is waiting for response from the server. **When the display is block the user sees the waiting icon on the page.**

The css attribute changes dynamically when the user sends a request and when the request-response is completed. Luckily in WebDriver we can get the value of a tag attribute and check against an expected value. e.g.

driver.findElement(By.cssSelector(".rotating-icon")).getAttribute();

This would either output “block” or “none” depending on the state of the request.

The code snippets below shows how to wait while the element contains a certain attribute value.

**public void waitWhileElementHasAttributeValue(String locator,**

**String attribute, String value) {**

**While (driver.findElement(LocatorStrategy.getLocatorMethod(locator)).getAttribute(attribute).contains (value)) {**

**int timeout = 10;**

**if(timeout > 0) {**

**Timeout--;**

**Try {**

**System.out.println(attribute + "t" + value);**

**Thread.sleep(1000);**

**} catch(Exception e) {**

**}**

**}**

**}**

**}**

**What is testNG listeners?**

Listeners “listen” to the event defined in the selenium script and behave accordingly.

**Benefits of Automation?**

1. Provides quick feedback whenever new feature is developed.
2. Release the burden on manual QA’s to run the regression tasks and make them focus on other important QA activities.
3. Can be run inattentive e.g. during nights.
4. To execute the regression test repeatedly on every new release.
5. The primary reason for test automation is to free up QA time for interesting exploratory testing and to give confidence to the team that the application is still in good order as new changes are delivered.
6. Quick feedback is one of the objectives of automated tests because developers are keen to know if what they have developed works and hasn’t broken current functionality.
7. Faster releases, increased test coverage, frequent test execution, faster feedback to development team.

**What is the difference between automation and manual testing?**

It is like Testing (manual or investigation exercise or enforce human’s judgement) vs Checking facts (Automation).

**Things to consider before doing the automation.**

Don’t Automate Unstable Functionality

**Which test cases to automate?**

Use a Risk-based approach to determine which tests should be automated. To get the most value out of automation, only automate the most important business cases and scenarios.

Also, a high number of automated tests adds maintenance cost and difficult to maintain.

Another note to bear in mind is that not all tests can be automated. Some tests are very complex in nature and require many downstream system checking and can be inconsistent. In these cases, it is best to leave these checks for manual testing.

**Automation Objectives:**

Automated checks are a great way of confirming that the application still functions properly after changes made to it.

##### Repeatability:

The automated tests should be repeatable and the output should be consistent in each run so that developers can rely on the outcome of the tests. This also means that we would not normally automate a test if it’s going to be run only once; the only exception to this is if you are running a test against a very large number of data, such as checking a link redirection script with many links.

##### **Reliability:**

The automated tests should really be checking verifications correctly and be able to determine actual results against valid expected results. This also means that if the results cannot be determined easily or automated tests are subject to environment issues which can cause false positives in the test results, then the tests cannot be reliable.

##### **Time:**

The automated tests should also save us time. If a simple test takes 10 minutes to complete but the same result can be determined in 1 minute manually, then it is best to not automate such tests.

**Difference between implicit wait and explicit wait?**

Using waits, we are telling WebDriver to wait for a certain amount of time before going to next step.

**Implicit wait:** When an implicit wait is implemented in tests, if WebDriver cannot find an element in the Document Object Model (DOM), it will wait for a defined amount of time for the element to appear in the DOM. In other terms, an implicit wait polls the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available.

***If you write implicit wait statement in you webdriver software testing script then it will be applied automatically to all elements of your test case.***

Implicit waits can slow down your tests because once set, the implicit wait is set for the life of the WebDriver object’s instance. This means that when an application responds normally, it will still wait for each element to appear in the DOM which increases the overall execution time.

Another downside is if for example you set the waiting limit to be 5 seconds and the elements appears in the DOM in 6 seconds, your tests will fail because you told it to wait a maximum of 5 seconds.

**Explicit wait:**

Explicit waits are better than implicit wait. Unlike an implicit wait, you can write custom code or conditions for a wait before proceeding further in the code.

An explicit wait can be used where synchronization is needed, for example, the page is loaded but we are still waiting for a call to complete and an element to appear.

Selenium WebDriver provides WebDriverWait and ExpectedCondition classes for implementing an explicit wait. The ExpectedCondition class provides a set of predefined conditions to wait before proceeding further in the code.

**To summarize:**

Implicit wait time is applied to all elements in your script and Explicit wait time is applied only for a particular specified element.

**How to Handle Dynamic Changing IDs In XPath?**

Use starts-with function. In this xpath's ID attribute, "post-body-" part remain same every time. So you can use xpath as bellow.

//div[starts-with(@id,'post-body-')]/div[1]/form[1]/input[1]

Or we can contains method

**Exceptions in selenium**

1. *No such element :*

Reasons: Wrong element locator

Element taking too much to load

Dynamic generation of element

1. Time out exception: When command does not complete operation within given time.
2. NoAlertPresentException
3. ElementNotSelectableException
4. ElementNotVisibleException
5. StaleElementReferenceException

The two reasons for Stale element reference are:

* The element has been deleted entirely.
* The element is no longer attached to the DOM.

**Solution 1: Refresh the page and try to find the element again**

driver.navigate().refersh();

driver.findElement (By.xpath("xpath here")).click();

**Solution 2:**

If an element is not attached to DOM then you could try using ‘try-catch block’ within ‘for loop’

For (int I=0; i<=2;i++){

   try {

     driver.findElement(By.xpath("xpath here")).click();

     break;

   }

  catch(Exception e){

      Sysout(e.getMessage());

   }

**Solution 3:**

Wait for the element till it gets available

wait.until(ExpectedConditions.presenceOfElementLocated(By.id("tabl e")));

**Difference between get () and navigate () methods?**

**Get ():** Will open the website and wait till the whole page is loaded

**Navigate:** Used to navigate (back, forth) and refresh the page and will not wait the loading of entire web page

**Google search page, I wants to search for some words without clicking on Google Search button. Is it possible In WebDriver? How?**

**Answer:** Yes we can do it using WebDriver sendKeys method where we do not need to use Google Search button. Syntax is as bellow.

driver.findElement(By.xpath("//input[@id='gbqfq']")).sendKeys("Search Syntax",Keys.ENTER)

**Drawback of Xpath**

* It is slower than cssSelector locator.
* xPath which works In one browser may not work In other browser for same page of software web application because some browsers (Ex. IE) reads only Lower-cased tag name and Attribute Name. So if used It In upper case then it will work In Firefox browser but will not work In IE browser. Every browser reads xPath in different way. In sort, do not use xPath locators in your test cases of software web application if you have to perform cross browser testing using selenium WebDriver software testing tool.

**Alert handling**

Alert alt= driver.switchto().alert();

**//To locate alert.**

Alert A1 = driver.switchTo().alert();

**//To read the text from alert popup.**

String Alert1 = A1.getText();

**//To accept/Click Ok on alert popup.**

A1.accept();

**//Confirmation Pop up Handling.**

driver.findElement(By.xpath("//button[@onclick='myFunction()']")).click();

Alert A2 = driver.switchTo().alert();

String Alert2 = A2.getText();

**//To click On cancel button of confirmation box.**

A2.dismiss();

**//Prompt Pop up Handling.**

driver.findElement(By.xpath("//button[contains(.,'Show Me Prompt')]")).click();

Alert A3 = driver.switchTo().alert();

String Alert3 = A3.getText();

**//To type text In text box of prompt pop up.**

A3.sendKeys("This Is John");

A3.accept();

**Screen shot capturing in selenium**

We can use selenium web driver TakesScreenshot method to capture screenshot. Java File class will be used to store screenshot in your system's local drive.

File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshot, new File ("D:\\screenshot.jpg"));

**Scroll to bottom of page:**

JavascriptExecutor javascript = (JavascriptExecutor) driver;

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

**Scroll to element in selenium WebDriver**

JavascriptExecutor javascript = (JavascriptExecutor) driver;

WebElement element = driver.findElement (By.xpath ("//div [@id='dragdiv']"));

je.executeScript ("arguments [0].scrollIntoView (true);” element);

**Java Script to check the loading of web page**

JavascriptExecutor javascript = (JavascriptExecutor) driver;

If (js.executeScript ("return document.readyState").toString ().equals ("complete")) {

System.out.println ("Page Is loaded.");

return;

**Switching between windows tabs**

//Switching between tabs using CTRL + tab keys.

driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL +"\t");

//Switch to current selected tab's content

driver.switchTo().defaultContent();

**How to switch between windows:**

/ Store the current window handle

String winHandleBefore = driver.getWindowHandle();

// perform the click operation that opens new window

// Switch to new window opened

for(String winHandle : driver.getWindowHandles()){

driver.switchTo().window(winHandle);

}

// Perform the actions on new window

// Close the new window, if that window no more required

driver.close();

// Switch back to original browser (first window)

driver.switchTo().window(winHandleBefore);