**Task 4**

**Selenium Wait - types with descriptions:**

# Selenium Waits

You might have come across wait commands while writing your first Selenium program. In this article, you would be learning about what exactly Selenium Waits is. You would be covering various types and other necessary factors one needs to understand to get started with Selenium Waits.

## What is Selenium Waits?

Waits in Selenium is one of the important pieces of code that executes a ****test case.**** It runs on certain commands called scripts that make a page load through it. Selenium Waits makes the pages ****less vigorous**** and ****reliable.**** It provides various types of wait options adequate and suitable under favorable conditions. This ensures you don't mess up and get ended into failed scripts while performing automation testing with it.

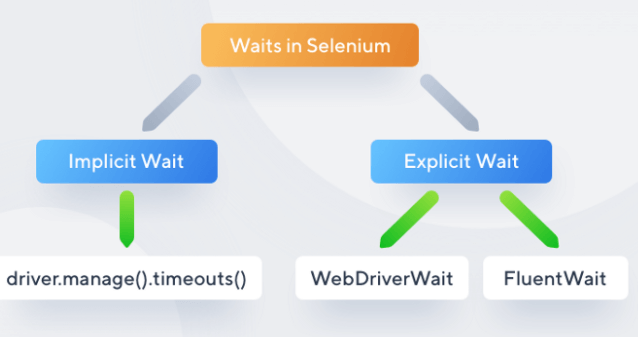
Elaborately, [Selenium](https://www.javatpoint.com/selenium-tutorial) Waits helps the user to troubleshoot various issues while page redirection across different web pages. It is achieved by refreshing the entire web page and reloading it with new elements. At times, there's a call from ****Ajax**** as well. Thus, some time lag might exist while reloading pages and reflecting elements present on the web pages after refreshing.

Another instance to understand Selenium Waits is navigating web pages back and forth with the ****navigate()**** command. This navigate() method comes from **[WebDriver,](https://www.javatpoint.com/selenium-webdriver)** whose main task is to simulate and manifest real-time scenarios like navigating between web pages concerning browsing history.

## Why Do You Need Waits In Selenium?

Today, most of the modern application's front-end is built on either [Ajax](https://www.javatpoint.com/ajax-tutorial) or [JavaScript](https://www.javatpoint.com/javascript-tutorial), followed by popular frameworks like Angular, React, or any other, which takes some time for loading elements on the web page. Hence, in such a case, Selenium throws an ****'ElementNotVisibleException'**** message when you tend to locate an element present in your script which is still not loaded on the web page.

## Types of Waits in Selenium:



### Implicit Waits:

The main function of implicit Wait is to tell the web driver to wait for some time before throwing a ****"No Such Element Exception".**** Its default setting is knocked at zero. Once the time is set, the driver automatically will wait for the amount of time defined by you before throwing the above-given exception.

****Syntax:****

driver.manage().timeouts().implicitlyWait(TimeOut,

TimeUnit.SECONDS);

**To understand how implicit wait works, let's consider an example.**

**package** JavaTpoint;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**public** **class** ImplicitWait{

**public** **static** **void** main(String[] args) **throws** InterruptedException

{

System.setProperty("webdriver.chrome.driver", "C:Selenium-java-

javaTpointchromedriver\_win32chromedriver.exe");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.manage().timeouts().pageLoadTimeout(40,

TimeUnit.SECONDS); // pageload timeout

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

// Implicit Wait for 20 seconds

driver.get("https://login.google.com/");

driver.findElement(By.xpath("//input[@id='login-

username']")).sendKeys("JavaTpoint.com"); //Finding element and

sending values

Thread.sleep(1000);

driver.findElement(By.xpath("//input[@id='login-signin']")).click(); //Clicking on the next button **if** element is located

}

}

In the code snippet given above, the Implicit Wait is defined for only ****20 seconds,**** implying that the output will load or arrive within the maximum waiting time of 20 seconds for the particular element.

**Note**: Implicit wait is globally applied. It is readily available for driver instance. It also means that if the driver is having interaction with one thousand elements in the meantime, the implicit wait will be applicable only for those 1000 elements. It cannot go beyond that.

### Explicit Waits

Explicit Waits also known as ****Dynamic Waits**** because it is highly specific conditioned. It is implemented by WebDriverWait class. To understand why you need Explicit Wait in Selenium, you must go through the basic knowledge of the wait statements in a program. In simple terms, you must know some conditions. Such conditions have been created to give you a gist of the Explicit Waits and why they are important.

****Condition 1:****

Suppose you have a web page consisting of a login form that takes input and loads the Home or Main page content. This page is dynamic because of the time constraints and network frequency, sometimes taking 10 seconds or maybe 15 seconds to load completely. Explicit Wait comes in handy in such cases and allows you to wait until the page is not present to display.

****Condition 2:****

Consider that you are working on an application that is travel themed and users fill the web form and submit it using submit button. Now, you might need to wait until and unless the specific data is not displayed. In such a case, Explicit Wait becomes helpful by waiting until a specific period for the set of elements that are not displayed yet.

****Syntax:****

WebDriverWait wait=**new**

ebDriverWait(WebDriveReference,TimeOut);

The above syntax justifies an object of WebDriver Wait and is passed to the driver's preference, and the timeout is taken as a parameter. To understand this more broadly, consider the below sample application.

**package** JavaTpoint;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**public** **class** Locators {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

System.setProperty("webdriver.chrome.driver", "C:Selenium-java-

javatpointchromedriver\_win32chromedriver.exe");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.manage().timeouts().pageLoadTimeout(40,

TimeUnit.SECONDS);

driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

driver.get("https://www.facebook.com/");

WebElement firstname= driver.findElement(By.name("firstname"));

WebElement lastname= driver.findElement(By.name("lastname"));

sendKeys(driver, firstname, 10, "Edureka");

sendKeys(driver, lastname, 20, "Edureka");

WebElement forgotAccount=

driver.findElement(By.linkText("Forgotten account?"));

clickOn(driver,forgotAccount, 10);

driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);

}

**public** **static** **void** sendKeys(WebDriver driver1, WebElement element,

**int** timeout, String value){

**new** WebDriverWait(driver1,

timeout).until(ExpectedConditions.visibilityOf(element));

element.sendKeys(value);

}

**public** **static** **void** clickOn(WebDriver driver1, WebElement element,

**int** timeout)

{

**new** WebDriverWait(driver1,

timeout).until(ExpectedConditions.elementToBeClickable(element));

element.click();

}

}

In the sample snippet given above, you can see Facebook sign-up credentials using locators have been created. Further, a generic utility function is created to make all the elements available for the Explicit Wait. Also, sendKeys() method is defined for the particular text field that would internally provide explicit Wait. Inside the sendKeys() method, there are some expected conditions for the element. This means that the driver is being asked to wait for 20 seconds until and unless the expected condition of the element is visible. Furthermore, the main purpose of applying the sendKeys() method is to take the first name and the last name, and that will be passed to the driver. The timeout is defined to be 10 seconds for the first name and the last name.

When the above program is executed, the Chrome driver will launch Chrome, and it will navigate through facebook.com to take the mentioned values. It is, although not mandatory, to explicitly set the timeout for a particular value since it is changeable. Another advantage to adding up here is that once you define the timeout for 10 seconds, it becomes applicable for all the elements present on the web page, and then it cannot be modified. The same concept works with the onClick() method defined in the above program, but this method is constrained only to links. This is the best way to execute, understand and execute Explicit Wait.

**Note:** Dynamic weights are consider dynamic because that if you take implicit, Explicit, or Fluent waits, and define the timeout method to be 20 seconds, and if the elements takes only 5 seconds, the remaining 15 seconds are not considered. It will not wait for the whole 20 seconds.

## Difference between Implicit and Explicit:

|  |  |
| --- | --- |
| **IMPLICIT WAIT** | **EXPLICIT WAIT** |
| It is by default applied to all the elements in the script. | It is applicable to only a certain element which is specific to a certain condition. |
| We cannot wait based on a specified condition like element selectable/clickable unlike explicit. | In explicit, we can specify the wait based on a specific condition. |
| It is usually used when you are sure the element may be visible in a certain time | It is usually used, when you are not aware of the time of the element visibility. It is subjected to dynamic nature. |

### Fluent Wait

Fluent Wait is quite similar to explicit Wait. It is similar in terms of management and functioning. In Fluent Wait, you can perform wait for action for an element only when you are unaware of the time it might take to be clickable or visible. Few differential factors that Fluent offers are as follows:

****The pooling frequency****

The pooling frequency in the case of Explicit is 500 milliseconds. But, using Fluent Wait, this pooling frequency can be changed to any value based upon your need. This usually means telling the script to keep an eye on the element after every 'x' seconds.

****Ignore Exception****

While pooling, if an element is not found, you can ignore some expectations like 'NoSuchElement'. Apart from this factor, similar to Explicit and Implicit Wait, you can define the amount of time for the element to be actionable or visible.

****Syntax:****

 Wait**<WebDriver>** fluentWait = new FluentWait**<WebDriver>**(driver)

       .withTimeout(60, SECONDS) // this defines the total amount of

time to wait for

       .pollingEvery(2, SECONDS) // this defines the polling frequency

       .ignoring(NoSuchElementException.class); // this defines the

exception to ignore

      WebElement foo = fluentWait.until(new Function**<WebDriver**,

WebElement**>**()

 {

     public WebElement apply(WebDriver driver)  //in this method

defined your own subjected conditions for which we need to wait for

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

}});

Although the syntax appears to be very complex, it comes in handy when you start using it. This is why probably all the software testers prefer using Explicit and Implicit Wait because of the simple syntax. Fluent waits appear complex because it doesn't provide predefined conditions that need to apply to the elements. Rather, Fluent Wait defines its condition within the apply method.

**package** javaTpoint.test;

**import** org.testng.annotations.Test;

**import** java.util.NoSuchElementException;

**import** java.util.concurrent.TimeUnit;

**import** java.util.function.Function;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.FluentWait;

**import** org.openqa.selenium.support.ui.Wait;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**import** org.testng.annotations.Test;

**public** **class** AppTest3 {

**protected** WebDriver driver;

    @Test

**public** **void** javaTpointTutorial() **throws** InterruptedException

    {

    System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );

    String eTitle = "Demo javaTpoint Page";

    String aTitle = "" ;

    driver = **new** ChromeDriver();

    // launch Chrome and redirect it to the Base URL

    driver.get("https://javatpoint.com/home" );

    //Maximizes the browser window

    driver.manage().window().maximize() ;

    //get the actual value of the title

    aTitle = driver.getTitle();

    //compare the actual title with the expected title

**if** (aTitle.contentEquals(eTitle))

    {

    System.out.println( "Test Passed") ;

    }

**else** {

    System.out.println( "Test Failed" );

        }

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

            .withTimeout(30, TimeUnit.SECONDS)

            .pollingEvery(5, TimeUnit.SECONDS)

            .ignoring(NoSuchElementException.**class**);

    WebElement clickseleniumlink = wait.until(**new**

Function<WebDriver, WebElement>(){

**public** WebElement apply(WebDriver driver ) {

**return**

driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/di

v[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/div/div[1]/div/div/a/i"));

        }

    });

    //click on the selenium link

    clickseleniumlink.click();

    //close~ browser

    driver.close() ;

    }

}

In the above code snippet, you can observe that the fluent Wait has been set for the timeout of 30 seconds, and the frequency is kept at 5 seconds by "NoSuchElementException". In another instance, a new function to identify the web element on the former page is created. If the element to found to fall in the same time frame, it may perform all the operations. If for some reason, any error shoots up, the function returns "ElementNotVisibleException".

## Cloud applications of Selenium Waits

Most testers prefer performing automation testing for the projects they are working on using some cloud-based service providers designed to suit Selenium. One such popular cloud-based testing method is LamdbaTest. LambdaTest is a powerful cross-platform browser testing tool.

Consider that you are running a cloud-based Selenium Grid like LambdaTest, and this is where the Selenium complex test suit comes up with a rapid timeout issue. You also have the facility of pausing the WebDriver for 90 seconds which can also be chosen as a default time limit, thereby avoiding timeout errors as the automation script runs successfully.

## Final Thoughts

Selenium Waits help the users to design scripts with reliability and fewer dependencies. You might choose whichever wait you want to proceed with by ensuring the business unit achieves the result and the purpose behind automation testing using Selenium. Another important aspect to note is ensuring that you are not keeping Selenium Waits unnecessarily in your application. Initially, there was the option of threads in the previous versions of Selenium which is now replaced with Waits. Selenium waits offer dynamic testing methods and an optimal environment for testing using platforms like JUnit or other testing mechanisms. In this article, you saw how the difference between the former waits like implicit, explicit, and fluent. You also learned why Fluent Wait is not preferred though highly sophisticated but syntactically tough for developers to carry out unit testing methods using Selenium. Later you learned through example code snippets how the aspects of Selenium Wait work in close integration with each other.