Implicit wait- applies for every driver statement. Hence it is used in “Before” or setup method

Explicit wait

WebDriverWait wait = new WebDriverWait(driver, 30);

Wait.until(Expectedconditions.elementToBeClickable(element));

Wait.until(ExpectedConditions.visibilityOfElementLocated(element));

‘driver.manage.timeouts.implicitlywait(10, Timeunit.seconds)

Fluent wait

## Why Do We Need Waits In Selenium?

Most of the web applications are developed using [Ajax](https://www.guru99.com/php-ajax.html) and [Javascript](https://www.guru99.com/interactive-javascript-tutorials.html). 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.

### **Implicit Wait in Selenium**

Implicit Wait directs the Selenium WebDriver to wait for a certain measure of time before throwing an exception. Once this time is set, WebDriver will wait for the element before the exception occurs.

Once the command is in place, Implicit Wait stays in place for the entire duration for which the browser is open. It’s default setting is 0, and the specific wait time needs to be set by the following protocol.

To add implicit waits in test scripts, import the following package.

import java.util.concurrent.TimeUnit;

**Syntax**

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

However, implicit wait increases test script execution time. It makes each command wait for the defined time before resuming test execution. If the application responds normally, the implicit wait can slow down the execution of test

Example

#### **Example Of Implicit Wait Command**

Package waitExample;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

public class WaitTest {

private WebDriver driver;

private String baseUrl;

private WebElement element;

@BeforeMethod

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://www.google.com";

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

}

@Test

public void testUntitled() throws Exception {

driver.get(baseUrl);

element = driver.findElement(By.id("lst-ib"));

element.sendKeys("Selenium WebDriver Interview questions");

element.sendKeys(Keys.RETURN);

List<WebElement> list = driver.findElements(By.className("\_Rm"));

System.out.println(list.size());

}

@AfterMethod

public void tearDown() throws Exception {

driver.quit();

}

}

### **Explicit Wait in Selenium**

By using the Explicit Wait command, the WebDriver is directed to wait until a certain condition occurs before proceeding with executing the code.

Setting Explicit Wait is important in cases where there are certain elements that naturally take more time to load. If one sets an implicit wait command, then the browser will wait for the same time frame before loading every web element. This causes an unnecessary delay in executing the test script.

Explicit wait is more intelligent, but can only be applied for specified elements. However, it is an improvement on implicit wait since it allows the program to pause for dynamically loaded Ajax elements.

In order to declare explicit wait, one has to use “ExpectedConditions”. The following Expected Conditions can be used in Explicit Wait.

* alertIsPresent()
* elementSelectionStateToBe()
* elementToBeClickable()
* elementToBeSelected()
* frameToBeAvaliableAndSwitchToIt()
* invisibilityOfTheElementLocated()
* invisibilityOfElementWithText()
* presenceOfAllElementsLocatedBy()
* presenceOfElementLocated()
* textToBePresentInElement()
* textToBePresentInElementLocated()
* textToBePresentInElementValue()
* titleIs()
* titleContains()
* visibilityOf()
* visibilityOfAllElements()
* visibilityOfAllElementsLocatedBy()
* visibilityOfElementLocated()

To use Explicit Wait in test scripts, import the following packages into the script.

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

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

Then, Initialize A Wait Object using WebDriverWait Class.

WebDriverWait wait = new WebDriverWait(driver,30);

Here, the reference variable is named <wait> for the <WebDriverWait> class. It is instantiated using the WebDriver instance. The maximum wait time must be set for the execution to layoff. Note that the wait time is measured in seconds.

#### **Example of Explicit Wait Command**

In the following example, the test script is for logging into “gmail.com” with a username and password. After a successful login, the code waits for the “compose” button to be available on the home page. Finally, it clicks on the button.

package waitExample;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

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

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

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

public class ExpectedConditionExample {

// created reference variable for WebDriver

WebDriver driver;

@BeforeMethod

public void setup() throws InterruptedException {

// initializing driver variable using FirefoxDriver

driver=new FirefoxDriver();

// launching gmail.com on the browser

driver.get("https://gmail.com");

// maximized the browser window

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

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

}

@Test

public void test() throws InterruptedException {

// saving the GUI element reference into a "element" variable of WebElement type

WebElement element = driver.findElement(By.id("Email"));

// entering username

element.sendKeys("dummy@gmail.com");

element.sendKeys(Keys.RETURN);

// entering password

driver.findElement(By.id("Passwd")).sendKeys("password");

// clicking signin button

driver.findElement(By.id("signIn")).click();

// explicit wait - to wait for the compose button to be click-able

WebDriverWait wait = new WebDriverWait(driver,30);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[contains(text(),'COMPOSE')]")));

// click on the compose button as soon as the "compose" button is visible

driver.findElement(By.xpath("//div[contains(text(),'COMPOSE')]")).click();

}

@AfterMethod

public void teardown() {

// closes all the browser windows opened by web driver

driver.quit();

}

}

The above code instructs Selenium WebDriver to wait for 30 seconds before throwing a TimeoutException. If it finds the element before 30 seconds, then it will return immediately. After that, it will click on the “Compose” button. In this case, the program will not wait for the entire 30 seconds, thus saving time and executing the script faster.

### **Difference between Implicit and Explicit Wait Commands in Selenium**

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| Applies to all elements in a test script. | Applies only to specific elements as intended by the user. |
| No need to specify “ExpectedConditions” on the element to be located | Must always specify “ExpectedConditions” on the element to be located |
| Most effective when used in a test case in which the elements are located with the time frame specified in implicit wait | Most effective when used when the elements are taking a long time to load. Also useful for verifying property of the element such as visibilityOfElementLocated, elementToBeClickable,elementToBeSelected |

### **Fluent Wait in Selenium**

Fluent Wait in Selenium marks the maximum amount of time for Selenium WebDriver to wait for a certain condition (web element) becomes visible. It also defines how frequently WebDriver will check if the condition appears before throwing the “**ElementNotVisibleException**”.

To put it simply, Fluent Wait looks for a web element repeatedly at regular intervals until timeout happens or until the object is found.

Fluent Wait commands are most useful when interacting with web elements that can take longer durations to load. This is something that often occurs in Ajax applications.

While using Fluent Wait, it is possible to set a default polling period as needed. The user can configure the wait to ignore any exceptions during the polling period.

Fluent waits are also sometimes called smart waits because they don’t wait out the entire duration defined in the code. Instead, the test continues to execute as soon as the element is detected – as soon as the condition specified in .**until(YourCondition)** method becomes true.

**Syntax**

Wait wait = new FluentWait(WebDriver reference)

.withTimeout(timeout, SECONDS)

.pollingEvery(timeout, SECONDS)

.ignoring(Exception.class);

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

public WebElement applyy(WebDriver driver) {

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

}

});

#### **Example of Fluent Wait Command**

//Declare and initialise a fluent wait

FluentWait wait = new FluentWait(driver);

//Specify the timout of the wait

wait.withTimeout(5000, TimeUnit.MILLISECONDS);

//Sepcify polling time

wait.pollingEvery(50, TimeUnit.MILLISECONDS);

//Specify what exceptions to ignore

wait.ignoring(NoSuchElementException.class)

//This is how we specify the condition to wait on.

//This is what we will explore more in this chapter

wait.until(ExpectedConditions.alertIsPresent());

[Source](https://www.toolsqa.com/selenium-webdriver/advance-webdriver-waits/)

This command operates with two primary parameters: timeout value and polling frequency. The above code defines the time out value as 5 seconds and polling frequency as 0.25 seconds. It directs WebDriver to wait for no more than 5 seconds to verify a specific condition. If the condition occurs during those 5 seconds, it will execute the next step in the test script. If not, it will return “**ElementNotVisibleException**”.

Note: -Suppose we are trying to find an element which has some **“ExpectedConditions** “(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an “**ElementNotVisibleException**“.

**FluentWait Command**

***Purpose:*** Each ***FluentWait*** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as ***NoSuchElementExceptions*** when searching for an element on the page.

// Waiting 30 seconds for an element to be present on the page, checking

// for its presence once every 5 seconds.

Wait wait = new FluentWait(driver)

.withTimeout(30, SECONDS)

.pollingEvery(5, SECONDS)

.ignoring(NoSuchElementException.class);

WebElement foo = wait.until(new Function() {

public WebElement apply(WebDriver driver) {

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

}

});

**A few other associated commands are:**

* **PageLoadTimeout Command**

This command establishes the time WebDriver must wait for a page to completely load before triggering an error. In case the timeout set is negative, the page load time can be indefinite.

This timeout is applicable only to **driver.manage()** and **driver.navigate.to()** methods.

**Syntax:**

driver.manage().timeouts().pageLoadTimeout(100, SECONDS);

* **SetScriptTimeout Command**

This command establishes the time WebDriver will wait for an asynchronous script to finish executing before triggering an error. Like the previous command, the script will run indefinitely if the timeout is set to a negative value.

**Syntax:**

driver.manage().timeouts().setScriptTimeout(100,SECONDS);

* **Sleep Command**

The Sleep command is rarely used because it is quite ineffective. It causes WebDriver to wait for a specific time (and does not let it run faster even if the specified condition is met). In fact, Selenium wait commands are considered the smarter, more effective alternative to the Sleep command.

**Syntax:**

thread.sleep(1000);

<https://www.browserstack.com/guide/wait-commands-in-selenium-webdriver>

additional theory

Implicit wait

This is always applied globally for all the elements wherever ‘driver’ keyword is used. So in general, it is used in setup method.

Dynamic in nature. Waits until the element found or timeout

Time can be changed anywhere at any point in your code

Explicity wait

Graphical user interface, text, application

Description automatically generated

If you have ajax elements , then explicit wait is best option

Below methods are useful to avoid extra lines of code whenever you use explicit wait for an element

Graphical user interface, text, application, email

Description automatically generated

Explicit wait is better compared to implicit wait. Because we do not want selenium to wait for every element unnecessarily. So we avoid implicit wait.

Implicit wait only waits for presence of element. It waits till the time element is not loaded into dom