# PROTRACTOR E2E TESTING –

## TYPE OF WAITS-

## Implicit waits

This is special hidden automatic wait, on each driver.findElement(...). Original webdriver (js, python, java) throws NoSuchElementException if element cannot be found in page DOM structure. This kind of wait will be done before EVERY driver.findElement, no matter what kind of locator do you use. When implicit wait timed out, NoSuchElementException will be thrown outside findElement function.

**Enabling implicit wait**

By default implicit wait timeout is set to 0. browser.manage().timeouts().implicitlyWait(3000)makes webdriver automatically try/catch this exception, and retry to find this element. If 3 seconds (timeout) passed, and element is still not present in DOM - only then you are getting NoSuchElementException.

**When it is good:**

Your page modify DOM structure (99.999% of website) and some elements still not in the DOM, but appear within 1-3 seconds. To not make explicit waits, and reduce amount of code - you can try to set implicit wait timeout.

**When it is bad:** You want to test that element is not present in the DOM. This kind of wait is added before every .findElement call, so when you are trying to assert like this:

expect($('NON-EXIST-ELEMENT').isPresent()).toBeFalsy()

Your implicitWait still working here. First you will wait for 3 seconds to element to be present, then exception will be thrown, and caught by isPresent() function, that will return false in this case (what we actually asserting). So you are waiting for 3 extra seconds! It makes sense to set implicitWait(0) and then set it back after asserting element is not present (which might be really annoying).

**Conclusion** Implicit waits are good, when you understand how it is works. I recommend to not set implicit wait more than 1-5 seconds (you should define own value for each website). Also if you plan to assert lot of not-present elements - reset implicit wait to 0, and then set it back.

## Explicit waits

This kind of waiting that you should call by yourself, but it much more flexible than implicit waits. In protractorjs, when you need to wait for something, you must call browser.wait(). It accepts predicate function (function that will return only true/false, no exceptions). Webdriver will poll this function until timeout occurs (you specify it as second param). Also you can specify error message that you want to throw as third parameter.

Obviously, that in web automation you wait for some element conditions most of the time. For this guys have created collection of predicate functions. This functions calls ExpectedConditions, and will return true/false for element that was passed to them.

browser.wait(ExpectedConditions.visibilityOf($('NON-EXISTING-ELEMENT')), 3000, 'error message')

**When it is good:** When you have to wait for some tricky conditions of your elements. You can easily define own conditions that you want to wait, specify custom timeout and so on. Use before manipulating with elements that might not be ready yet.

**When it is bad:** When you try to help you by combining browser.sleep(), implicit waits and explicit waits together. browser.sleep() is bad by default, in 99% of cases you can replace it with browser.wait() with provided conditions, or write your own condition.

Much more fun happens when you have your implicit wait set, and you trying to call explicit wait. Imagine: browser.manage().timeouts().implicitlyWait(10000) browser.wait(EC.stalenessOf($('NON-EXIST-ELEMENT')), 5000) //waiting for 5 seconds for element to disappear

What happens here: Wait function calls stalenessOf() function for your element. Inside it, driver.findElement() got called. Implicit wait don't let this function to throw error immediately, and pools webpage for 10 seconds until implicit wait timeout happens, and we are getting NoSuchElementException. Exception happens, and execution returns to wait function, 10 seconds are passed already! Wait is terminated with TimeOutException, because it was scheduled only for 5 seconds. We are getting error with wait time much longer that expected.

Also keep in mind that JS is async, and cannot guarantee exact wait time because of Event Loop. Usually this makes waiting not exact - 5200 ms instead 5000 (as example). This is absolutely different story :)

What happens in your example

implicit timeout - 4000 milliseconds.

explicit timeout - 5000 milliseconds.

1. Wait started. First time calling predicate function - presenceOf()
2. Internally predicate calls original webdriverjs function - driver.findElement(By.xpath('//\*[@name='qwer']'))
3. Since implicit wait is set, we are waiting for it before throw error.
4. 4000 milliseconds of implicit element waiting passed. Only now we are returning error to predicate function.
5. Predicate function catch error, and returns false instead
6. Since we still have 1000 milliseconds before timeout of explicit wait - calling predicate function again.
7. Implicit wait started again. 4000 milliseconds later - throwing error back to predicate function
8. Predicate returns false
9. Wait function got false, and our explicit wait is out of time - in ideal case - it would be about 8000 milliseconds, but also be aware about async calls, so real time would be more
10. Wait throws error - jasminejs catch error, and fails test.

## Element is not visible-

Here are the common solutions:

make sure the element you want to click is actually visible. Sometimes you need to make extra actions on a page to make the element visible. For example, open up a dropdown for an option to appear or open menu for submenu to appear

* [wait](http://www.protractortest.org/#/api?view=webdriver.WebDriver.prototype.wait) for the [visibility](http://www.protractortest.org/#/api?view=ExpectedConditions.prototype.visibilityOf) of the element:

var EC = protractor.ExpectedConditions;

var mumbaiCity = element(by.id('mumbaiCity'));

browser.wait(EC.visibilityOf(mumbaiCity), 5000);

mumbaiCity.click();

there is an *another element* with the same id that is actually invisible. In this case, you need to improve your locator to match this specific element. For instance:

element(by.css(".city-checkbox #mumbaiCity")).click();

element(by.css(".city-checkbox input[ng-click\*=Mumbai]")).click();

* Or, if you got multiple elements matching the same locator - you can ["filter"](http://www.protractortest.org/#/api?view=ElementArrayFinder.prototype.filter) out a *visible element*:

var mumbaiCity = element.all(by.id('mumbaiCity')).filter(function (elm) {

return elm.isDisplayed().then(function (isDisplayed) {

return isDisplayed;

});

}).first();

mumbaiCity.click();

* move to element and then click via [browser.actions()](http://www.protractortest.org/#/api?view=webdriver.WebDriver.prototype.actions):

var mumbaiCity = element(by.id('mumbaiCity'));

browser.actions().mouseMove(mumbaiCity).click().perform();

* [scroll into view](https://developer.mozilla.org/en-US/docs/Web/API/Element/scrollIntoView) of the element and then click:

var mumbaiCity = element(by.id('mumbaiCity'));

browser.executeScript("arguments[0].scrollIntoView();", mumbaiCity.getWebElement());

mumbaiCity.click();

* click *via javascript* (beware of the [differences](https://stackoverflow.com/questions/34562061/webdriver-click-vs-javascript-click) though):

var mumbaiCity = element(by.id('mumbaiCity'));

browser.executeScript("arguments[0].click();", mumbaiCity.getWebElement());

* sometimes, you just need to *maximize the browser window*:

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

## For Checking Selected Dropdown-

Example HTML-

<select ng-options="opions.c as options.n for option in options" ng-model="model">

<option value="0">Option 1</option>

<option value="1">Option 2</option>

<option label="Other" value="string:Other" selected="selected">Other</option>

</select>

Way 1:

element(by.model('model')).element(by.css('option:checked')).getText();

Way 2:

**var** contactRole = **element**.all(by.*xpath*(**'//select/option[@label="Other"]'**)).first();  
contactRole.getAttribute(**'selected'**).then(**function** (value) {  
 ***console***.log(**'RoleValue'**, value); // return true. If Other is selected.  
})

Way 3:

var all\_options = element.all(

by.options("opions.c as options.n for option in options") //use whatever string is assigned to the ng-options attribute in the html

);

var first\_option = all\_options.get(0); //or all\_options.first()

var second\_option = all\_options.get(1); //or all\_options.last()

expect(

first\_option.getAttribute('value')

).toEqual("Africa");

## Wait for some attribute Value

<div class="zoomButton no-print">

<button class="exportButtton no-print">

<i class="fa fa-download"> </i>

</button>

<a class="oc-download-btn" download="Regression Testing.png"></a></div>

// AFTER BUTTON CLICK ‘href’ attribute is added like this

<a class="oc-download-btn" download="Regression Testing.png" href=”google.com”></a></div>

Way 1:

**var** downloadButton = **element**(by.*xpath*(**'//div[@class="zoomButton no-print"]'**));  
*//browser.wait(EC.stalenessOf(downloadButton), 5000,'Download button is not attatched to page');*browser.sleep(1000);  
downloadButton.click().then(**function** () {  
 **var** loadingElement = **element**(by.*xpath*(**'//div[@class="mask"]'**));  
 browser.wait(**EC**.not(**EC**.presenceOf(loadingElement)), 60000, **"Loading Data"**);  
 **var** imageLink = **element**(by.*css*(**'[class="oc-download-btn"]'**));  
 browser.wait(**function** () {  
 **return** imageLink.getAttribute(**'href'**).then(**function** (value) {  
 **return** value !== **null**;  
 });  
 }, 60000);

## Way to close current Tab When onClick event, a new Tab is opened for it.

Switch to the new opened tab. Close the current windows (in this case, the new tab). Switch back to the first window.

browser.getAllWindowHandles().then(function (handles) {

browser.switchTo().window(handles[1]);

browser.close();

browser.switchTo().window(handles[0]);

});

1. When You Expect for element to be Present by EC.presenceOf(element), But gives error like “Cannot bind property of undefined”.

# [**Can protractor be made to run slowly?**](https://stackoverflow.com/questions/24960290/can-protractor-be-made-to-run-slowly)

This needs to be run before any tests are invoked (outside describe block)

var origFn = browser.driver.controlFlow().execute;

browser.driver.controlFlow().execute = function() {

var args = arguments;

// queue 100ms wait

origFn.call(browser.driver.controlFlow(), function() {

return protractor.promise.delayed(100);

});

return origFn.apply(browser.driver.controlFlow(), args);

};

## How to create Shortcut For element locators like $ and $$-

To create a shortcut, add the custom locator on the global namespace and on the prototype of ElementFinder and ElementArrayFinder:

global.$r = function(selector) {

return protractor.element(protractor.by.repeater(selector));

};

global.$$r = function(selector) {

return protractor.element.all(protractor.by.repeater(selector));

};

ElementFinder.prototype.$$r = function(selector) {

return this.all(protractor.by.repeater(selector));

};

ElementFinder.prototype.$r = function(selector) {

return this.element(protractor.by.repeater(selector));

};

ElementArrayFinder.prototype.$$r = function(selector) {

return this.all(protractor.by.repeater(selector));

};

## Usage:

$r("item in items")

$$r("item in items")

$("#id").$r("item in items")

$("#id").$$r("item in items")