



FREE eBook

LEARNING protractor

Unaffiliated free eBook created from
Stack Overflow contributors.

#protractor

Table of Contents

About.....	1
Chapter 1: Getting started with protractor	2
Remarks.....	2
Versions.....	2
Examples.....	2
Installing and Setting up Protractor (On Windows).....	2
Write a Protractor test.....	3
First test using Protractor.....	4
Selective Running Tests.....	5
Pending Tests.....	6
Combinations.....	6
Protractor: E2E Testing for Enterprise Angular Applications.....	6
Chapter 2: Control Flow and Promises.....	9
Introduction.....	9
Examples.....	9
Understanding the Control Flow.....	9
Chapter 3: CSS Selectors.....	10
Syntax.....	10
Parameters.....	10
Remarks.....	10
Examples.....	11
\$ and \$\$ CSS selector locator shortcuts.....	11
Introduction to locators.....	12
Select element by an exact HTML attribute value.....	12
Select element by an HTML attribute that contains a specified value.....	12
Chapter 4: Explicit waits with browser.wait().....	13
Examples.....	13
browser.sleep() vs browser.wait().....	13
Chapter 5: Locating Elements.....	14
Introduction.....	14

Parameters.....	14
Examples.....	14
Protractor specific locators (for Angular-based applications).....	14
Binding locator.....	14
Example.....	14
Exact Binding locator.....	15
Example.....	15
Model locator.....	15
Example.....	15
Button text locator.....	15
Example.....	16
Partial button text locator.....	16
Repeater locator.....	16
Example.....	16
Exact repeater locator.....	17
Example.....	17
CSS and text locator.....	17
Example.....	18
Options locator.....	18
Example.....	18
Deep CSS locator.....	18
Example.....	18
Locator basics.....	19
Chapter 6: Page Objects.....	21
Introduction.....	21
Examples.....	21
First Page Object.....	21
Chapter 7: Protractor configuration file.....	23
Introduction.....	23
Examples.....	23
Simple Config file - Chrome.....	23

Config file with capabilities - Chrome.....	23
config file shardTestFiles - Chrome.....	23
config file multi-capabilities emulate - chrome.....	24
Chapter 8: Protractor Debugger.....	25
Syntax.....	25
Remarks.....	25
Examples.....	25
Using browser.pause().....	25
Using browser.debugger().....	26
Chapter 9: Testing non-angular apps with Protractor.....	28
Introduction.....	28
Examples.....	28
Changes needed to test non-angular app with Protractor.....	28
Chapter 10: XPath selectors in Protractor.....	29
Examples.....	29
Selecting a DOM element using protractor.....	29
Selecting elements with specific attributes.....	29
By Class.....	29
By id.....	30
Other attributes.....	30
Credits.....	32

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [protractor](#)

It is an unofficial and free protractor ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official protractor.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with protractor

Remarks

Protractor is an end-to-end test framework for AngularJS applications.

Protractor is a wrapper (built on the top) around Selenium WebDriver, so it contains every feature that is available in the Selenium WebDriver. Additionally, Protractor provides some new locator strategies and functions which are very helpful to automate the AngularJS application. Examples include things like: `waitForAngular`, `By.binding`, `By.repeater`, `By.textarea`, `By.model`, `WebElement.all`, `WebElement.evaluate`, etc.

Versions

Version	Release Data
0.0.1	2016-08-01

Examples

Installing and Setting up Protractor (On Windows)

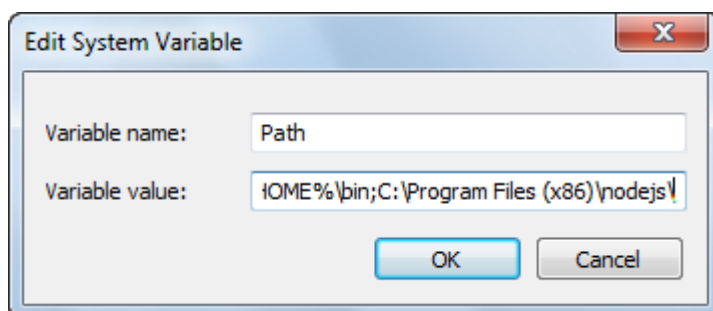
Requirements: Protractor requires the following dependencies to be installed prior to installation:

- Java JDK 1.7 or higher
- Node.js v4 or higher

Installation:

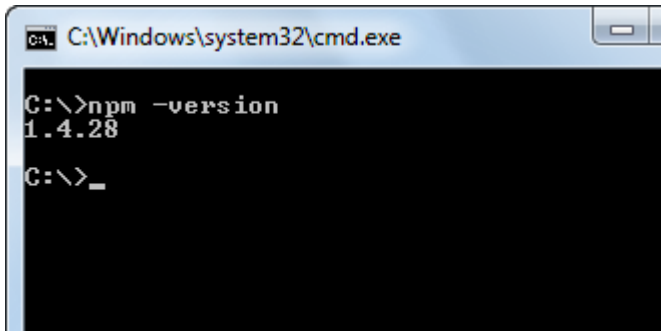
Download and install Node.js from this URL: <https://nodejs.org/en/>

To see if the Node.js installation is successful, you can go and check the Environment variables. The 'Path' under System Variables will be automatically updated.



You can also check the same by typing the command `npm -version` in command prompt which will

give you the installed version.



```
C:\Windows\system32\cmd.exe
C:\>npm -version
1.4.28
C:\>_
```

Now Protractor can be installed in two ways: Locally or Globally.

We can install protractor in a specified folder or project directory location. If we install in a project directory, every time we run, we should run from that location only.

To install locally in project directory, navigate to the project folder and type the command

```
npm install protractor
```

To install Protractor globally run the command:

```
$ npm install -g protractor
```

This will install two command line tools, `protractor` and `webdriver-manager`.

Run `protractor --version` to ensure protractor was successfully installed.

`webdriver-manager` is used to download the browser driver binaries and start the selenium server.

Download the browser driver binaries with:

```
$ webdriver-manager update
```

Start the selenium server with:

```
$ webdriver-manager start
```

To download internet explorer driver, run the command `webdriver-manager update --ie` in command prompt. This will download IEDriverServer.exe in your selenium folder

Write a Protractor test

Open a new command line or terminal window and create a clean folder for testing.

Protractor needs two files to run, a spec file and a configuration file.

Let's start with a simple test that navigates to the todo list example in the AngularJS website and adds a new todo item to the list.

Copy the following into `spec.js`

```
describe('angularjs homepage todo list', function() { it('should add a todo', function() { browser.get('https://angularjs.org');
```

```
    element(by.model('todoList.todoText')).sendKeys('write first protractor test');
    element(by.css('[value="add"]')).click();

    var todoList = element.all(by.repeater('todo in todoList.todos'));
    expect(todoList.count()).toEqual(3);
    expect(todoList.get(2).getText()).toEqual('write first protractor test');

    // You wrote your first test, cross it off the list
    todoList.get(2).element(by.css('input')).click();
    var completedAmount = element.all(by.css('.done-true'));
    expect(completedAmount.count()).toEqual(2);});});});
```

First test using Protractor

Protractor needs only two files to run the first test, `spec` (test code) file and configuration file. The `spec` file contains test code and the other one contains configuration details like `spec` file path, browser details, test url, framework parameters etc. To write first test we will be providing only selenium server address and `spec` file path. The other parameters like browser, timeout, framework will be picked up to default values.

The default browser for Protractor is Chrome.

conf.js - Configuration file

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js']
};
```

spec.js - Spec (test code) file

```
describe('first test in protractor', function() {
  it('should verify title', function() {
    browser.get('https://angularjs.org');

    expect(browser.getTitle()).toEqual('AngularJS - Superheroic JavaScript MVW Framework');
  });
});
```

seleniumAddress - Path to the server where webdriver server is running .

specs - An array element which contains path of test files. The multiple paths can be specified by comma separated values.

describe - Syntax from [Jasmine](#) framework. `describe` syntax sta

Selective Running Tests

Protractor can selectively run groups of tests using `fdescribe()` instead of `describe()`.

```
fdescribe('first group', ()=>{
  it('only this test will run', ()=>{
    //code that will run
  });
});
describe('second group', ()=>{
  it('this code will not run', ()=>{
    //code that won't run
  });
});
```

Protractor can selectively run tests within groups using `fit()` instead of `it()`.

```
describe('first group', ()=>{
  fit('only this test will run', ()=>{
    //code that will run
  });
  it('this code will not run', ()=>{
    //code that won't run
  });
});
```

If there is no `fit()` within an `fdescribe()`, then every `it()` will run. However, a `fit()` will block `it()` calls within the same `describe()` or `fdescribe()`.

```
fdescribe('first group', ()=>{
  fit('only this test will run', ()=>{
    //code that will run
  });
  it('this code will not run', ()=>{
    //code that won't run
  });
});
```

Even if a `fit()` is in a `describe()` instead of an `fdescribe()`, it will run. Also, any `it()` within an `fdescribe()` that does not contain a `fit()` will run.

```
fdescribe('first group', ()=>{
  it('this test will run', ()=>{
    //code that will run
  });
  it('this test will also run', ()=>{
    //code that will also run
  });
});
describe('second group', ()=>{
  it('this code will not run', ()=>{
    //code that won't run
  });
  fit('this code will run', ()={
    //code that will run
  });
});
```

```
});  
});
```

Pending Tests

Protractor allows tests to be set as pending. This means that protractor will not execute the test, but will instead output:

```
Pending:  
1) Test Name  
Temporarily disabled with xit
```

Or, if disabled with xdescribe():

```
Pending:  
1) Test Name  
No reason given
```

Combinations

- A xit() within an xdescribe() will output the xit() response.
- A xit() within an fdescribe() will still be treated as pending.
- A fit() within an xdescribe() will still run, and no pending tests will output anything.

Protractor: E2E Testing for Enterprise Angular Applications

Protractor Installation and Setup

Step 1: Download and install NodeJS from [here](#). Make sure you have latest version of node. Here, I am using node v7.8.0. You will need to have the Java Development Kit(JDK) installed to run selenium.

Step 2: Open your terminal and type in the following command to install protractor globally.

```
npm install -g protractor
```

This will install two tools such as protractor and webdriver manager. You can verify your Protractor Installation by following command: `protractor -version`. If Protractor is installed successfully then the system will display the installed version.(i.e. Version 5.1.1).Otherwise you will have to recheck the installation. Step 3: Update the webdriver manager to download the necessary binaries.

```
webdriver-manager update
```

Step 4: Following command will start up a Selenium Server. This step will run the web driver manager in the background and will listen to any tests which runs via protractor.

webdriver-manager start You can see information about the status of the server at

<http://localhost:4444/wd/hub/static/resource/hub.html>.

Writing First Test case using Protractor:

Before jump into the writing the test case, we have to prepare two files that is configuration file and spec file.

In configuration file :

```
//In conf.js
exports.config = {
  baseUrl: 'http://localhost:8800/adminapp',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['product/product_test.js'],
  directConnect : true,
  capabilities :{
    browserName: 'chrome'
  }
}
```

Basic Understanding of the Terminologies used in configuration file:

baseUrl – A base URL for your application under test.

seleniumAddress – To connect to a Selenium Server which is already running.

specs – Location of your spec file

directConnect : true – To connect directly to the browser Drivers.

capabilities – If you are testing on a single browser, use the capabilities option. If you are testing on multiple browsers, use the multiCapabilities array.

You can find more configuration option from [here](#). They have described all possible terminology with its definition.

In Spec file :

```
//In product_test.js

describe('Angular Enterprise Boilerplate', function() {
  it('should have a title', function() {
    browser.get('http://localhost:8800/adminapp');
    expect(browser.getTitle()).toEqual('Angular Enterprise Boilerplate');
  });
});
```

Basic Understanding of the Terminologies used in spec file:

By default, Protractor uses the jasmine framework for its testing interface. 'describe' and 'it' syntax is from jasmine framework. You can learn more from [here](#). Running First Test case:

Before run the test case make sure that your webdriver manager and your application running in different tabs of your terminal.

Now, Run the test with :

```
Protractor app/conf.js
```

You should see the chrome browser opens up with your application url and close itself. The test output should be 1 tests, 1 assertion, 0 failures.

Bravo! You successfully run your first test case.

Read **Getting started with protractor** online: <http://www.riptutorial.com/protractor/topic/933/getting-started-with-protractor>

Chapter 2: Control Flow and Promises

Introduction

Protractor/WebDriverJS has this mechanism called [Control Flow](#) - it is an internal queue of promises, it keeps the code execution organized.

Examples

Understanding the Control Flow

Consider the following test:

```
it('should test something', function() {  
    browser.get('/dashboard/');  
  
    $("#myid").click();  
    expect(element(by.model('username')).getText()).toEqual('Test');  
  
    console.log("HERE");  
});
```

In the following test, when the `console.log()` is executed and you see `HERE` on the console, none of the Protractor commands from previous lines have been executed. This is an entirely *asynchronous* behavior. The commands are represented as promises and were put on the Control Flow which would execute and resolve the promises sequentially, one by one.

See more at [Promises and the Control Flow](#).

Read [Control Flow and Promises](#) online: <http://www.riptutorial.com/protractor/topic/8580/control-flow-and-promises>

Chapter 3: CSS Selectors

Syntax

- `by.css('css-selector')`
- `by.id('id')`
- `by.model('model')`
- `by.binding('binding')`

Parameters

Parameter	Details
css-selector	A css selector like <code>'.class-name'</code> to select the element on the base of class name
id	Id of the dom element
model	Model used for dom element
binding	Name of the binding which is used to bound to certain element

Remarks

How to write css selectors?

The most important attributes to write css selectors are class and id of dom. For an instance if a html dom looks like below example:

```
<form class="form-signin">
  <input type="text" id="email" class="form-control" placeholder="Email">
  <input type="password" id="password" class="form-control" placeholder="Password">
  <button class="btn btn-block" id="signin-button" type="submit">Sign in</button>
</form>
```

Then to select the email input field, you can write css selector in following way:

1. **Using class name:** The class name in css selector starts with special character `.(dot)`. The css selector for that will be like this `.form-control`.

```
by.css('.form-control')
```

Since the `form-control` class is shared by both input elements so it raises a concern of duplicity in locators. So in such situation if id is available then you should always prefer to use id instead of class name.

2. Using ID: The id in css selector starts with special character # (hash). So the css selector using id for email input element will be written like below:

```
by.css('#email')
```

3. Using multiple class names: If dom element has multiple classes then you can with combination of classes as css selector. For example if dom element is like this:

```
<input class="username-class form-control">
// css selector using multiple classes
by.css('.username-class.form-control')
```

4. Using tag name with other attributes : The general expression to write css selector using tag name and other attributes is `tagname[attribute-type='attribute-value']`. So following the expression the css locator for sign-in button can be formed like this:

```
by.css("button[type='submit']") //or
by.css("button[id='signin-button']")
```

Examples

\$ and \$\$ CSS selector locator shortcuts

The Protractor API allows CSS element locators to use the jQuery-like [shortcut notation](#) `$()`.

Normal CSS Element Locator:

```
element(by.css('h1.documentation-text[ng-bind="title"]'));
element(by.css('[ng-click="submit"]'));
```

Shortcut `$()` CSS Element Locator:

```
$('h1.documentation-text[ng-bind="title"]');
$('[ng-click="submit"]');
```

For finding multiple elements under a locator use the [shortcut notation](#) `$$()`.

Normal CSS Element Locator:

```
element.all(by.css('h1.documentation-text[ng-bind="title"]'));
element.all(by.css('[ng-click="submit"]'));
```

Shortcut `$$()` CSS Element Locator:

```
$$('h1.documentation-text[ng-bind="title"]');
$$('[ng-click="submit"]');
```

Introduction to locators

A locator in Protractor is used to perform action on HTML dom elements. The most common and best locators used in Protractor are css, id, model and binding. For example commonly used locators are:

```
by.css('css-selector')
by.id('id')
```

Select element by an exact HTML attribute value

To select an element by an exact HTML attribute use the css locator pattern [\[attribute=value\]](#)

```
//selects the first element with href value '/contact'
element(by.css('[href="/contact"]'));

//selects the first element with tag option and value 'foo'
element(by.css('option[value="foo"]'));

//selects all input elements nested under the form tag with name attribute 'email'
element.all(by.css('form input[name="email"]'));
```

Select element by an HTML attribute that contains a specified value

To select an element by an HTML attribute that contains a specified value use the css locator pattern [\[attribute*=value\]](#)

```
//selects the first element with href value that contains 'cont'
element(by.css('[href*="cont"]'));

//selects the first element with tag h1 and class attribute that contains 'fo'
element(by.css('h1[class*="fo"]'));

//selects all li elements with a title attribute that contains 'users'
element.all(by.css('li[title*="users"]'));
```

Read CSS Selectors online: <http://www.riptutorial.com/protractor/topic/1524/css-selectors>

Chapter 4: Explicit waits with `browser.wait()`

Examples

`browser.sleep()` vs `browser.wait()`

When it comes to dealing with timing issue, it is tempting and easy to put a "quick"

`browser.sleep(<timeout_in_milliseconds>)` and move on.

The problem is, it would some day fail. There is no golden/generic rule on what sleep timeout to set and, hence, at some point due to network or performance or other issues, it might take more time for a page to load or element to become visible etc. Plus, most of the time, you would end up waiting more than you actually should.

`browser.wait()` on the other hand works differently. You provide an [Expected Condition function](#) for Protractor/WebDriverJS to execute and wait for the result of the function to evaluate to true. *Protractor would continuously execute the function and stop once the result of the function evaluates to true or a configurable timeout has been reached.*

There are multiple built-in Expected Conditions, but you can also create and use a custom one (sample [here](#)).

Read Explicit waits with `browser.wait()` online:

<http://www.riptutorial.com/protractor/topic/8297/explicit-waits-with-browser-wait-->

Chapter 5: Locating Elements

Introduction

To be able to interact with a page, you need to tell Protractor exactly which element to look for. The basis used for selecting elements are locators. Protractor, as well as including the generic Selenium selectors, also has Angular-specific locators which are more robust and persistent to changes. However, sometimes, even in an Angular application, regular locators must be used.

Parameters

Parameter	Detail
selector	A string which specifies the value of the selector (depends on the locator)

Examples

Protractor specific locators (for Angular-based applications)

These locators should be used as a priority when possible, because they are more persistent to changes in an application then locators based on css or xpath, which can easily break.

Binding locator

Syntax

```
by.binding('bind value')
```

Example

View

```
<span>{{user.password}}</span>  
<span ng-bind="user.email"></span>
```

Locator

```
by.binding('user.password')  
by.binding('user.email')
```

Also supports partial matches

```
by.binding('email')
```

Exact Binding locator

Similar to `binding`, except partial matches are not allowed.

Syntax

```
by.exactBinding('exact bind value')
```

Example

View

```
<span>{{user.password}}</span>
```

Locator

```
by.exactBinding('user.password')  
by.exactBinding('password') // Will not work
```

Model locator

Selects an element with an [Angular model directive](#)

Syntax

```
by.model('model value')
```

Example

View

```
<input ng-model="user.username">
```

Locator

```
by.model('user.username')
```

Button text locator

Selects a button based on its text. Should be used only if button text not expected to change often.

Syntax

```
by.buttonText('button text')
```

Example

View

```
<button>Sign In</button>
```

Locator

```
by.buttonText('Sign In')
```

Partial button text locator

Similar to `buttonText`, but allows partial matches. Should be used only if button text not expected to change often.

Syntax

```
by.partialButtonText('partial button text')
```

Example

View

```
<button>Register an account</button>
```

Locator

```
by.partialButtonText('Register')
```

Repeater locator

Selects an element with an [Angular repeater directive](#)

Syntax

```
by.repeater('repeater value')
```

Example

View

```
<tbody ng-repeat="review in reviews">
  <tr>Movie was good</tr>
  <tr>Movie was ok</tr>
  <tr>Movie was bad</tr>
</tbody>
```

Locator

```
by.repeater('review in reviews')
```

Also supports partial matches

```
by.repeater('reviews')
```

Exact repeater locator

Similar to `repeater`, but does not allow partial matches

Syntax

```
by.exactRepeater('exact repeater value')
```

Example

View

```
<tbody ng-repeat="review in reviews">
  <tr>Movie was good</tr>
  <tr>Movie was ok</tr>
  <tr>Movie was bad</tr>
</tbody>
```

Locator

```
by.exactRepeater('review in reviews')
by.exactRepeater('reviews') // Won't work
```

CSS and text locator

An extended CSS locator where you can also specify the text content of the element.

Syntax

```
by.cssContainingText('css selector', 'text of css element')
```

Example

View

```
<ul>
  <li class="users">Mike</li>
  <li class="users">Rebecca</li>
</ul>
```

Locator

```
by.cssContainingText('.users', 'Rebecca') // Will return the second li only
```

Options locator

Selects an element with an [Angular options directive](#)

Syntax

```
by.options('options value')
```

Example

View

```
<select ng-options="country.name for c in countries">
  <option>Canada</option>
  <option>United States</option>
  <option>Mexico</option>
</select>
```

Locator

```
by.options('country.name for c in countries')
```

Deep CSS locator

CSS locator that extends into the [shadow DOM](#)

Syntax

```
by.deepCss('css selector')
```

Example

View

```
<div>
  <span id="outerspan">
    <"shadow tree">
      <span id="span1"></span>
      <"shadow tree">
        <span id="span2"></span>
      </>
    </>
  </div>
```

Locator

```
by.deepCss('span') // Will select every span element
```

Locator basics

Locators by themselves do not return an element which can be interacted with in Protractor, they are simply instructions that indicate Protractor how to find the element.

To access the element itself, use this syntax:

```
element(locator);
element.all(locator);
```

Note: the element(s) is not actually accessed until an action is performed on it - that is, Protractor will only actually go retrieve the element when an action such as `getText()` is called on the element.

If you want to select only one element using a locator, use `element`. If your locator points to multiple elements, `element` will return the first one found. `element` returns an `ElementFinder`.

If you want to select multiple elements using a locator, `element.all` will return all elements found. `element.all` returns an `ElementArrayFinder`, and every element in the array can be accessed using different methods - for example, the `map` function.

```
element.all(locator).map(function(singleElement) {
  return singleElement.getText();
});
```

Chaining locators

You can chain multiple locators to select an element in a complex application. You can't directly chain `locator` objects, you must chain `ElementFinders`:

```
element(by.repeater('movie in movies')).element(by.linkText('Watch Frozen on Netflix'))
```

There is no limit to how many chains you can use; in the end, you will still receive a single `ElementFinder` or `ElementArrayFinder`, depending on your locators.

Read Locating Elements online: <http://www.riptutorial.com/protractor/topic/10825/locating-elements>

Chapter 6: Page Objects

Introduction

Page objects is a design pattern which results in less code duplicates, easy maintenance and more readability.

Examples

First Page Object

```
/* save the file in 'pages/loginPage'
var LoginPage = function(){

};

/*Application object properties*/
LoginPage.prototype = Object.create({}, {
  userName: {
    get: function() {
      return browser.driver.findElement(By.id('userid'));
    }
  },
  userPass: {
    get: function() {
      return browser.driver.findElement(By.id('password'));
    }
  },
  submitBtn: {
    get: function() {
      return browser.driver.findElement(By.id('btnSubmit'));
    }
  }
});

/* Adding functions */
LoginPage.prototype.login = function(strUser, strPass) {
  browser.driver.get(browser.baseUrl);
  this.userName.sendKeys(strUser);
  this.userPass.sendKeys(strPass);
  this.submitBtn.click();
};

module.exports = LoginPage;
```

Let's use our first page object file in our test.

```
var LoginPage = require('../pages/loginPage');
describe('User Login to Application', function() {
  var loginPage = new LoginPage();

  beforeAll(function() {
    loginPage.login(browser.params.userName, browser.params.userPass);
```

```
});  
  
it('and see a success message in title', function() {  
    expect(browser.getTitle()).toEqual('Success');  
});  
  
});
```

Read Page Objects online: <http://www.riptutorial.com/protractor/topic/9747/page-objects>

Chapter 7: Protractor configuration file

Introduction

The configuration file contains information which Protractor uses to run your test script. Here I'll try to give a few different variations.

Examples

Simple Config file - Chrome

```
var config = {};  
var timeout = 120000;  
  
config.framework = 'jasmine2';  
config.allScriptsTimeout = timeout;  
config.getPageTimeout = timeout;  
config.jasmineNodeOpts.isVerbose = true;  
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;  
config.specs = ['qa/**/*.Spec.js'];  
config.browserName = 'chrome';  
  
exports.config = config;
```

Config file with capabilities - Chrome

```
var config = {};  
var timeout = 120000;  
  
config.framework = 'jasmine2';  
config.allScriptsTimeout = timeout;  
config.getPageTimeout = timeout;  
config.jasmineNodeOpts.isVerbose = true;  
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;  
config.specs = ['qa/**/*.Spec.js'];  
config.capabilities = {  
  browserName: 'chrome',  
  'chromeOptions': {  
    'args': ['start-minimized', 'window-size=1920,1080']  
  }  
};  
  
exports.config = config;
```

config file shardTestFiles - Chrome

This configuration lets' you run your total spec files in two browser instances in parallel. It helps reduce the overall test execution time. Change the maxInstances based on your need.

Note: *Make sure your tests are independent.*

```

var config = {};
var timeout = 120000;

config.framework = 'jasmine2';
config.allScriptsTimeout = timeout;
config.getPageTimeout = timeout;
config.jasmineNodeOpts.isVerbose = true;
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;
config.specs = ['qa/**/*.Spec.js'];
config.capabilities = {
  browserName: 'chrome',
  shardTestFiles: true,
  maxInstances: 2,
  'chromeOptions': {
    'args': ['start-minimized', 'window-size=1920,1080']
  }
};

exports.config = config;

```

config file multi-capabilities emulate - chrome

```

var config = {};
var timeout = 120000;

config.framework = 'jasmine2';
config.allScriptsTimeout = timeout;
config.getPageTimeout = timeout;
config.jasmineNodeOpts.isVerbose = true;
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;
config.specs = ['qa/**/*.Spec.js'];
config.multiCapabilities = [{
  browserName: 'chrome',
  shardTestFiles: true,
  maxInstances: 2,
  'chromeOptions': {
    'args': ['start-minimized', 'window-size=1920,1080']
  }
},
{
  browserName: 'chrome',
  shardTestFiles: true,
  maxInstances: 1,
  'chromeOptions': {
    'args': ['show-fps-counter=true'],
    'mobileEmulation': {
      'deviceName': 'Apple iPhone 6'
    }
  }
}
];

exports.config = config;

```

Read Protractor configuration file online:

<http://www.riptutorial.com/protractor/topic/9745/protractor-configuration-file>

Chapter 8: Protractor Debugger

Syntax

- `browser.pause()`
- `browser.debugger()`

Remarks

This section explains how we can debug protractor tests.

Examples

Using `browser.pause()`

The `pause()` method is one of the easiest solution Protractor provides you to debug the code, in order to use it you have to add it in your code where you want to pause the execution. Once the execution is in paused state:

1. You can use `c` (type `C`) to move forward. Be careful while using it, you have to write this command without any delay as you might get timeout error from your assertion library if you delayed to press `c`.
2. Type `repl` to enter interactive mode. The interactive mode is used to send browser commands directly to open instance of browser. For example in interactive mode you can issue command like this:

```
> element(by.css('#username')).getText()  
> NoSuchElementError: No element found using locator: by.username("#username")
```

Notice output of above command appears directly over there, which lets you know correctness of your command.

Note: If you have opened the Chrome Dev Tools, you must close them before continuing the test because ChromeDriver cannot operate when the Dev Tools are open.

3. Exit debug mode using `CTRL+C`, you can take yourself out from debug mode using classical `CTRL+C` command.

```
it('should pause when we use pause method', function () {  
    browser.get('/index.html');  
  
    var username = element(by.model('username'));  
    username.sendKeys('username');  
    browser.pause();  
  
    var password = element(by.model('password'));
```

```
password.sendKeys('password');
browser.pause();
});
```

4. Press d to continue to the next debugger statement

Using browser.debugger()

You can use `browser.debugger()` to stop the execution. You can insert it any place in your code and it will stop the execution after that line until you don't command to continue.

Note: To run the tests in debugger mode you have to issue command like this:

```
`protractor debug <configuration.file.js>`
```

Enter `c` to start execution and continue after the breakpoint or enter `next` command. The next command steps to the next line in control flow.

The debugger used in Protractor uses [node debugger](#) and it pause the execution in asynchronous way. For example, in below code the `browser.debugger()` will get called when `username.sendKeys('username')` has been executed.

Note: Since these are asynchronous tasks, you would have to increase the default timeout of your specs else default timeout exception would be thrown!

```
it('should pause when we use pause method', function () {
  browser.get('/index.html');

  var username = element(by.model('username'));
  username.sendKeys('username');
  browser.debugger();

  var password = element(by.model('password'));
  password.sendKeys('password');
});
```

One can enter the `repl` mode by entering the command-

```
debug > repl
> element(by.model('abc')).sendKeys('xyz');
```

This will run the `sendKeys` command as the next task, then re-enter the debugger.

One can change the `Port no.` they want to debug their scripts by just passing the port to the debugger method-

```
browser.debugger(4545); //will start the debugger in port 4545
```

The `debugger()` method injects a client side from Protractor to browser and you can run few commands in browser console in order to fetch the elements. One of the example to use client

side script is:

```
window.clientSideScripts.findInputs('username');
```

Read Protractor Debugger online: <http://www.riptutorial.com/protractor/topic/3910/protractor-debugger>

Chapter 9: Testing non-angular apps with Protractor

Introduction

Protractor is made for testing Angular applications. However, it is still possible to test non-angular applications with Protractor if needed.

Examples

Changes needed to test non-angular app with Protractor

Use `browser.driver` instead of `driver`

Use `browser.driver.ignoreSynchronization = true`

Reason: Protractor waits for angular components to load completely on a web-page before it begins any execution. However, since our pages are non-angular, Protractor keeps waiting for 'angular' to load till the test fails with timeout. So, we need to explicitly tell the Protractor to not to wait for 'angular'

Read Testing non-angular apps with Protractor online:

<http://www.riptutorial.com/protractor/topic/8830/testing-non-angular-apps-with-protractor>

Chapter 10: XPath selectors in Protractor

Examples

Selecting a DOM element using protractor

Apart from CSS, model, and binding selectors, protractor can also locate elements using xpath View

```
<ul>
<li><a href='http://www.google.com'>Go to google</a></li>
</ul>
```

Code

```
var googleLink= element(by.xpath('//ul/li/a'));
expect(element.getText()).to.eventually.equal('Go to google','The text you mention was not found');
```

Selecting elements with specific attributes

XPath selectors can be used to select elements with specific attributes, such as class, id, title etc.

By Class

View:

```
<div class="HakunaMatata"> Hakuna Matata </div>
```

Code:

```
var theLionKing= element(by.xpath('//div[@class="HakunaMatata"]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

However, an element can have multiple classes. In such cases, the 'contains' workaround can be used

View:

```
<div class="Hakuna Matata"> Hakuna Matata </div>
```

Code:

```
var theLionKing= element(by.xpath('//div[contains(@class,"Hakuna")]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

The above piece of code will return elements containing both 'class="HakunaMatata"' and 'class="Hakuna Matata"'. If your search text is a part of a space-separated list, then the following workaround may be used:

```
var theLionKing= element(by.xpath('//div[contains(concat(' ',normalize-space(@class),' '),  
"Hakuna")]'));  
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

By id

ID remains the easiest and the most precise locator which can be used to select an element.

View:

```
<div id="HakunaMatata">Hakuna Matata</div>
```

Code:

```
var theLionKing= element(by.xpath('//div[@id="HakunaMatata"]'));  
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

As with classes, the contains function can be used to find an element containing the given text.

Other attributes

Finding an element with a given **title** attribute

View

```
<div title="Hakuna Matata">Hakuna Matata</div>
```

Code

```
var theLionKing= element(by.xpath('//div[@title="Hakuna Matata"]'));  
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Selecting an element with a specific text

View

```
<div class="Run Simba Run">Run Simba</div>
```

Code

```
var runSimba= element(by.xpath('//div[text()='Run Simba']'));  

```

As with other text based searches, the contains function can be used to select elements with text() containing the required match.

View

```
<div class="Run Simba Run">Run Simba,run</div>
```

Code

```
var runSimba= element(by.xpath('//div[contains(text(),"Run Simba")]'));  
expect(runSimba.getText()).to.eventually.equal('Run Simba, run', "Text not found"); //true
```

Selecting an element with a specific name attribute

View

```
<input type="text" name="FullName"></input>
```

Code

```
var fullNameInput= element(by.xpath('//input[@name="FullName"]'));  
fullNameInput.sendKeys("John Doe");
```

Read XPath selectors in Protractor online: <http://www.riptutorial.com/protractor/topic/7205/xpath-selectors-in-protractor>

Credits

S. No	Chapters	Contributors
1	Getting started with protractor	Bhoomi Bhalani , Community , Devmati Wadikar , Manuli Piyalka , olyv , Peter Stegnar , Praveen , Priyanshu Shekhar , SilentLupin , sonhu , Stephen Leppik
2	Control Flow and Promises	alecxe
3	CSS Selectors	alecxe , Droogans , leon , Priyanshu Shekhar , sonhu
4	Explicit waits with browser.wait()	alecxe
5	Locating Elements	Sébastien Dufour-Beauséjour
6	Page Objects	Barney , Suresh Salloju
7	Protractor configuration file	Barney
8	Protractor Debugger	Devmati Wadikar , Priyanshu Shekhar , Ram Pasala , Sakshi Singla , Stephen Leppik
9	Testing non-angular apps with Protractor	Sakshi Singla
10	XPath selectors in Protractor	Shubhang