Protractor is an open source end-to-end testing framework for Angular and AngularJS applications. It was built by Google on the top of WebDriver.

* [Home](https://www.protractortest.org/#/)
* [Quick Start](javascript:void(0))
* [Protractor Setup](javascript:void(0))
* [Protractor Tests](javascript:void(0))
* [Reference](javascript:void(0))

[[github logo](https://github.com/angular/protractor)View on GitHub](https://github.com/angular/protractor)

Protractor is an end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

Test Like a User

Protractor is built on top of WebDriverJS, which uses native events and browser-specific drivers to interact with your application as a user would.

For Angular Apps

Protractor supports Angular-specific locator strategies, which allows you to test Angular-specific elements without any setup effort on your part.

Automatic Waiting

You no longer need to add waits and sleeps to your test. Protractor can automatically execute the next step in your test the moment the webpage finishes pending tasks, so you don’t have to worry about waiting for your test and webpage to sync.

Setup

Use npm to install Protractor globally with:

npm install -g protractor

This will install two command line tools, protractor and webdriver-manager. Try running protractor --version to make sure it's working.

The webdriver-manager is a helper tool to easily get an instance of a Selenium Server running. Use it to download the necessary binaries with:

webdriver-manager update

Now start up a server with:

webdriver-manager start

This will start up a Selenium Server and will output a bunch of info logs. Your Protractor test will send requests to this server to control a local browser. You can see information about the status of the server at <http://localhost:4444/wd/hub>.

Write a 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 todo-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);

});

});

The describe and it syntax is from the Jasmine framework. browser is a global created by Protractor, which is used for browser-level commands such as navigation with browser.get.

Configuration

Now create the configuration file. Copy the following into conf.js:

exports.config = {

seleniumAddress: 'http://localhost:4444/wd/hub',

specs: ['todo-spec.js']

};

This configuration tells Protractor where your test files (specs) are, and where to talk to your Selenium Server (seleniumAddress). It will use the defaults for all other configuration. Chrome is the default browser.

Run the test

Now run the test with:

protractor conf.js

You should see a Chrome browser window open up and navigate to the todo list in the AngularJS page, then close itself (this should be very fast!). The test output should be 1 test, 3 assertions, 0 failures. Congratulations, you've run your first Protractor test!

Learn More

Learn more with the [Tutorial](https://www.protractortest.org/#/tutorial).

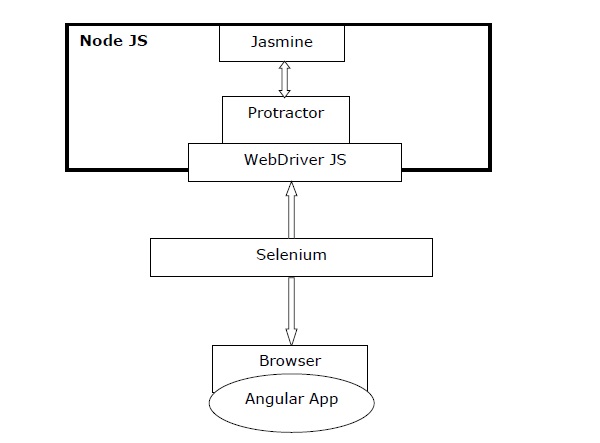
This chapter gives you an introduction to Protractor, where you will learn about the origin of this testing framework and why you have to choose this, working and limitations of this tool.

What is Protractor?

Protractor is an open source end-to-end testing framework for Angular and AngularJS applications. It was built by Google on the top of WebDriver. It also serves as a replacement for the existing AngularJS E2E testing framework called “Angular Scenario Runner”.

It also works as a solution integrator that combines powerful technologies such as NodeJS, Selenium, Jasmine, WebDriver, Cucumber, Mocha etc. Along with testing of AngularJS application, it also writes automated regression tests for normal web applications. It allows us to test our application just like a real user because it runs the test using an actual browser.

The following diagram will give a brief overview of Protractor −



Observe that in the above diagram, we have −

* **Protractor** − As discussed earlier, it is a wrapper over WebDriver JS especially designed for angular apps.
* **Jasmine** − It is basically a behavior-driven development framework for testing the JavaScript code. We can write the tests easily with Jasmine.
* **WebDriver JS** − It is a Node JS bindings implementation for selenium 2.0/WebDriver.
* **Selenium** − It simply automates the browser.

Origin

As said earlier, Protractor is a replacement for the existing AngularJS E2E testing framework called “Angular Scenario Runner”. Basically, the origin of Protractor starts with the end of Scenario Runner. A question that arises here is why do we need to build Protractor? To understand this, we first need to check about its predecessor - Scenario Runner.

Protractor’s Inception

Julie Ralph, the prime contributor to the development of Protractor, had the following experience with Angular Scenario Runner on other project within Google. This further became the motivation to build Protractor, specially to fill the gaps −

*“We tried using Scenario Runner and we found that it really just couldn’t do the things that we needed to test. We needed to test things like logging in. Your login page is not an Angular page, and the Scenario Runner couldn’t deal with that. And it couldn’t deal with things like popups and multiple windows, navigating the browser history, stuff like that.”*

The biggest advantage to the Protractor was the maturity of Selenium project and it wraps up its methods so that it can be easily used for Angular projects. The design of Protractor is built in such a way that it tests all layers such that web UI, backend services, persistence layer and so on of an application.

Why Protractor?

As we know that almost all the applications are using JavaScript for development. The task of testers becomes difficult when JavaScript increases in size and becomes complex for applications due to the increasing number of the applications itself. Most of the times it becomes very difficult to capture the web elements in AngularJS applications, uses extended HTML syntax to express web application components, by using JUnit or Selenium WebDriver.

The question here is that why Selenium Web Driver is not able to find AngularJS web elements? The reason is because AngularJS applications are having some extended HTML attributes like ng-repeater, ng-controller and ng-model etc. which are not included in Selenium locators.

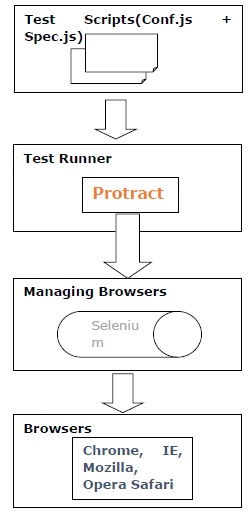
Here, the importance of Protractor comes into existence because Protractor on the top of Selenium can handle and control those extended HTML elements in AngularJS web applications. That is why we can say that most of the frameworks focus on conducting unit tests for AngularJS applications, Protractor used to do testing of the actual functionality of an application.

Working of Protractor

Protractor, the testing framework, works in conjunction with Selenium to provide an automated test infrastructure for simulating a user’s interaction with an AngularJS application that is running in browser or mobile device.

The working of Protractor can be understood with the help of following steps −

* **Step 1** − In the first step, we need to write the tests. It can be done with the help of Jasmine or Mocha or Cucumber.
* **Step 2** − Now, we need to run the test which can be done with the help of Protractor. It is also called test runner.
* **Step 3** − In this step, Selenium server will help to manage the browsers.
* **Step 4** − At last, the browser APIs are invoked with the help of Selenium WebDriver.



Advantages

This open source end-to-end testing framework offers the following advantages −

* An open source tool, Protractor is very easy to install and setup.
* Works well with Jasmine framework to create the test.
* Supports test driven development (TDD).
* Contains automatic waits which means we do not need to explicitly add waits and sleeps to our test.
* Offers all the advantages of Selenium WebDriver.
* Supports parallel testing through multiple browsers.
* Provides the benefit of auto-synchronization.
* Has excellent testing speed.

Limitations

This open source end-to-end testing framework possesses the following limitations −

* Does not uncover any verticals in browser automation because it is a wrapper for WebDriver JS.
* Knowledge of JavaScript is essential for the user, because it is available only for JavaScript.
* Only provides front-end testing because it is a UI driven testing tool.

# **Choosing a Framework**

Protractor supports two behavior driven development (BDD) test frameworks out of the box: Jasmine and Mocha. These frameworks are based on JavaScript and Node.js and provide the syntax, scaffolding, and reporting tools you will use to write and manage your tests.

## Using Jasmine

Currently, Jasmine Version 2.x is supported and the default test framework when you install Protractor. For more information about Jasmine, see the [Jasmine GitHub site](http://jasmine.github.io/). For more information regarding how to upgrade to Jasmine 2.x from 1.3, see the [Jasmine upgrade guide](https://www.protractortest.org/#/jasmine-upgrade).

## Using Mocha

Note: Limited support for Mocha is available as of December 2013. For more information, see the [*Mocha documentation site*](http://mochajs.org/).

If you would like to use the Mocha test framework, you'll need to use the BDD interface and Chai assertions with [Chai As Promised](http://chaijs.com/plugins/chai-as-promised).

Download the dependencies with npm. Mocha should be installed in the same place as Protractor - so if protractor was installed globally, install Mocha with -g.

npm install -g mocha

npm install chai

npm install chai-as-promised

You will need to require and set up Chai inside your test files:

var chai = require('chai');

var chaiAsPromised = require('chai-as-promised');

chai.use(chaiAsPromised);

var expect = chai.expect;

You can then use Chai As Promised as such:

expect(myElement.getText()).to.eventually.equal('some text');

Finally, set the 'framework' property to 'mocha', either by adding framework: 'mocha' to the config file or by adding --framework=mocha to the command line.

Options for Mocha such as 'reporter' and 'slow' can be given in the [config file](https://github.com/angular/protractor/blob/5.4.1/spec/mochaConf.js) with mochaOpts:

mochaOpts: {

reporter: "spec",

slow: 3000

}

For a full example, see Protractor’s own test: [/spec/mocha/lib\_spec.js](https://github.com/angular/protractor/blob/5.4.1/spec/mocha/lib_spec.js).

## Using Cucumber

Note: Cucumber is no longer included by default as of version *3.0*. You can integrate Cucumber with Protractor with the *custom* framework option. For more information, see the [*Protractor Cucumber Framework site*](https://github.com/mattfritz/protractor-cucumber-framework) or the [*Cucumber GitHub site*](https://github.com/cucumber/cucumber-js).

If you would like to use the Cucumber test framework, download the dependencies with npm. Cucumber should be installed in the same place as Protractor - so if protractor was installed globally, install Cucumber with -g.

npm install -g cucumber

npm install --save-dev protractor-cucumber-framework

Set the 'framework' property to custom by adding framework: 'custom' and frameworkPath: 'protractor-cucumber-framework' to the config file(cucumberConf.js)

Options for Cucumber such as 'format' can be given in the config file with cucumberOpts, A basic cucumberConf.js file has been provided below:

/\*

Basic configuration to run your cucumber

feature files and step definitions with protractor.

\*\*/

exports.config = {

seleniumAddress: 'http://localhost:4444/wd/hub',

baseUrl: 'https://angularjs.org/',

capabilities: {

browserName:'chrome'

},

framework: 'custom', // set to "custom" instead of cucumber.

frameworkPath: require.resolve('protractor-cucumber-framework'), // path relative to the current config file

specs: [

'./cucumber/\*.feature' // Specs here are the cucumber feature files

],

// cucumber command line options

cucumberOpts: {

require: ['./cucumber/\*.js'], // require step definition files before executing features

tags: [], // <string[]> (expression) only execute the features or scenarios with tags matching the expression

strict: true, // <boolean> fail if there are any undefined or pending steps

format: ["pretty"], // <string[]> (type[:path]) specify the output format, optionally supply PATH to redirect formatter output (repeatable)

'dry-run': false, // <boolean> invoke formatters without executing steps

compiler: [] // <string[]> ("extension:module") require files with the given EXTENSION after requiring MODULE (repeatable)

},

onPrepare: function () {

browser.manage().window().maximize(); // maximize the browser before executing the feature files

}

};

## Using Serenity/JS

[Serenity/JS](http://serenity-js.org/) is an acceptance testing library which can be integrated as a drop-in replacement of [Mocha](http://serenity-js.org/mocha/readme.html) or [Cucumber](http://serenity-js.org/cucumber/readme.html) framework adapters to provide advanced [scalability and reporting capabilities](http://serenity-js.org/overview/readme.html).

To use it, [install](http://serenity-js.org/overview/installation.html) and [configure](http://serenity-js.org/overview/configuration.html) your test framework of choice.

Next, [install Serenity/JS](http://serenity-js.org/overview/installation.html):

npm install serenity-js --save-dev

and instruct Protractor to use the Serenity/JS adapter:

exports.config = {

framework: 'custom',

frameworkPath: require.resolve('serenity-js')

// ...

};

Specifying either cucumberOpts or mochaOpts in your Protractor configuration will make Serenity/JS infer that it should delegate the execution to [Cucumber](https://github.com/cucumber/cucumber-js) or [Mocha](https://mochajs.org/), respectively.

To learn more, [visit the project website](http://serenity-js.org/) or [follow the tutorial](http://serenity-js.org/from-scripts-to-serenity/readme.html).

## Using a Custom Framework

Check section [Framework Adapters for Protractor](https://github.com/angular/protractor/blob/5.4.1/lib/frameworks/README.md) specifically [Custom Frameworks](https://github.com/angular/protractor/blob/5.4.1/lib/frameworks/README.md#custom-frameworks)