# Setting Up the Browser

Protractor works with Selenium WebDriver, a browser automation framework. Selenium WebDriver supports several browser implementations or drivers which are discussed below.

## Browser Support

Protractor supports the two latest major versions of Chrome, Firefox, Safari, and IE.

Please see Browser Support for a full list of supported browsers and known issues.

## Configuring Browsers

In your Protractor config file (see config.ts), all browser setup is done within the `capabilities` object. This object is passed directly to the WebDriver builder (builder.js).

See DesiredCapabilities for full information on which properties are available.

## Using Mobile Browsers

Please see the Mobile Setup documentation for information on mobile browsers.

## Using Browsers Other Than Chrome

To use a browser other than Chrome, simply set a different browser name in the capabilities object.

```
capabilities: {
  'browserName': 'firefox'
}
```

You may need to install a separate binary to run another browser, such as IE or Android. For more information, see SeleniumHQ Downloads.

## Adding Chrome-Specific Options

Chrome options are nested in the `chromeOptions` object. A full list of options is at the ChromeDriver site. For example, to show an FPS counter in the upper right, your configuration would look like this:

```
capabilities: {
  'browserName': 'chrome',
  'chromeOptions': {
    'args': ['show-fps-counter=true']
  }
},
```

## Adding Firefox-Specific Options

Firefox options are nested in the `moz:firefoxOptions` object. A full list of options is at the GeckoDriver Github page. For example, to run in safe mode, your configuration would look like this:

```
capabilities: {
  'browserName': 'firefox',
  'moz:firefoxOptions': {
    'args': ['--safe-mode']
  }
},
```

## Testing Against Multiple Browsers

If you would like to test against multiple browsers, use the `multiCapabilities` configuration option.

```
multiCapabilities: [{
  'browserName': 'firefox'
}, {
  'browserName': 'chrome'
}]
```

Protractor will run tests in parallel against each set of capabilities. Please note that if `multiCapabilities` is defined, the runner will ignore the `capabilities` configuration.

## Using Multiple Browsers in the Same Test

If you are testing apps where two browsers need to interact with each other (e.g. chat systems), you can do that with protractor by dynamically creating browsers on the go in your test. Protractor exposes a function in the `browser` object to help you achieve this: `browser.forkNewDriverInstance(opt_useSameUrl, opt_copyMockModules)`. Calling this will return a new independent browser object. The first parameter in the function denotes whether you want the new browser to start with the same url as the browser you forked from. The second parameter denotes whether you want the new browser to copy the mock modules from the browser you forked from.

```
browser.get('http://www.angularjs.org');
browser.addMockModule('moduleA', "angular.module('moduleA', []).value('version', '3');");

// To create a new browser.
var browser2 = browser.forkNewDriverInstance();

// To create a new browser with url as 'http://www.angularjs.org':
var browser3 = browser.forkNewDriverInstance(true);

// To create a new browser with mock modules injected:
var browser4 = browser.forkNewDriverInstance(false, true);

// To create a new browser with url as 'http://www.angularjs.org' and mock modules injected:
var browser4 = browser.forkNewDriverInstance(true, true);
```

Now you can interact with the browsers. However, note that the globals `element`, `$`, `$$` and `browser` are all associated with the original browser. In order to interact with the new browsers, you must specifically tell protractor to do so like the following:

```
var element2 = browser2.element;
var $2 = browser2.$;
var $$2 = browser2.$$;
element2(by.model(...)).click();
$2('.css').click();
$$2('.css').click();
```

Protractor will ensure that commands will automatically run in sync. For example, in the following code, `element(by.model(...)).click()` will run before `browser2.$('.css').click()`:

```
browser.get('http://www.angularjs.org');
browser2.get('http://localhost:1234');
```

```
browser.sleep(5000);
element(by.model(...)).click();
browser2.$('.css').click();
```

## Setting up PhantomJS

PhantomJS is no longer officially supported. Instead, we recommend to use one of the following alternatives:

1. Chrome with headless mode. Available in Chrome 59+ on Linux/Mac OS X, and in Chrome 60+ on Windows.
2. Firefox with headless mode. Available in Firefox 55+ on Linux, and in Firefox 56+ on Windows/Mac OS X.
3. Chrome with Xvfb.

## Using headless Chrome

To start Chrome in headless mode, start Chrome with the `--headless` flag.

As of Chrome 58 you also need to set `--disable-gpu`, though this may change in future versions. Also, changing the window size during a test will not work in headless mode, but you can set it on the commandline like this `--window-size=800,600`.

```
capabilities: {
  browserName: 'chrome',

  chromeOptions: {
     args: [ "--headless", "--disable-gpu", "--window-size=800,600" ]
   }
}
```

## Using headless Firefox

To start Firefox in headless mode, start Firefox with the `--headless` flag.

```
capabilities: {
  browserName: 'firefox',

  'moz:firefoxOptions': {
     args: [ "--headless" ]
   }
}
```