Protractor Testing: Protractor plays an important role in the Testing of AngularJS applications and works as a Solution integrator combining powerful technologies like Selenium, Jasmine, Web driver, etc. It is intended not only to test AngularJS application but also for writing automating regression tests for normal Web Applications as well.

Need Of Protractor Framework: JavaScript is used in only almost all web applications. As the applications grows, JavaScript also increases in size and complexity. In such case, it becomes difficult task for Testers to test the web application for various scenarios.

Sometimes it is difficult to capture the web elements in AngularJS applications using Junit or Selenium WebDriver.

Protractor is a NodeJS program which is written in JavaScript and runs with Node to identinfy the web elements in AngularJS applications, and it also uses WebDriver to control the browser with user actions.

Why can't we find Angular JS web elements using Normal Selenium Web driver?

Angular JS applications have some extra HTML attributes like ng-repeater, ng-controller, ng-model.., etc. which are not included in Selenium locators. Selenium is not able to identify those web elements using Selenium code. So, Protractor on the top of Selenium can handle and controls those attributes in Web Applications.

The protractor is an end to end testing framework for Angular JS based applications. While most frameworks focus on conducting unit tests for Angular JS applications, Protractor focuses on testing the actual functionality of an application.

Protractor vs Selenium WebDriver:
What makes Protractor different from traditional Selenium WebDriver?

- Simple syntax to write test cases
- The ability to run multiple browsers at once using Selenium Grid
- Angular-specific locators
- Support for Behavior-driven development such as Jasmine/Mocha
- No need to add sleeps/waits

- Supported integration with Jenkins/Browser Stack/Grunt etc.
- Get rid of dealing with synchronization issue in Angular JS websites
- Multiple browser support (Firefox, Chrome, Safari, Internet explorer)
- Ability to run the same scripts in mobile browsers also without the need to change the code

Features:

1) WaitForAngular
From documentation:

"*Instruct WebDriver to wait until Angular has finished rendering and has no outstanding $http or $timeout calls before continuing. Note that Protractor automatically applies this command before every WebDriver action.*"

What this means is that there is no need to manually add waits to your script and Protractor will automatically wait for the web elements to load and only then executes the next steps.

2) It has the ability to export a global function *element*, which takes a locator and will return an ElementFinder. This ElementFinder has a set of action methods, such as click (), getText(), sendKeys() etc. This is the core of how to interact with the element and get information from it.
This global function helps to reduce the element locating syntax. Take a look at the following statement to locate the element in both Selenium WebDriver and Protractor:

Selenium Webdriver:
```
driver.findElement(By.cssSelector("css selector"));
```

Protractor:
```
element(by.css('some-css'));
```

3) Some new locator strategies and functions provided to help locate the Angular elements are: *By.binding*, *By.repeater*, *By.textarea*, *By.model*, *WebElement.all*, *WebElement.evaluate*, etc.

***4)* Assertions & annotations:**
Assertions are an important part of the automation scripts. Annotations are also very useful in effectively tagging certain methods in a class to have special meaning.

It provides a variety of assertions & annotations and in addition to that, also provides the ability to create your own assertions.

<u>Example:</u>

```
describe('Code to understand assertions/annotations', function() {
 beforeEach(function() {
 browser.get('http://juliemr.github.io/protractor-demo/');
 });
 afterEach(function() {
 browser.get('https://www.madewithangular.com/#/');
 });
 var multiplyNumbers = function(a, b) {
 element(by.model('first')).sendKeys(a);
 element(by.model('second')).sendKeys(b);
 element(by.model('operator')).click();
 element(by.id('gobutton')).click();
 };
 it('should multiply two integers', function() {
 multiplyNumbers(2, 2);
 expect(element.all(by.repeater('result in memory')).count()).toEqual(2);
 multiplyNumbers(3, 3);
 expect(element.all(by.repeater('result in memory')).count()).toEqual(2);
 });
 });
```

In the above example, we are using 2 annotations, *'beforeEach'* and *'afterEach'*. These annotations are also available in TestNG (traditional Selenium). These annotations make sure that a particular piece of code will be executed before/after respectively the execution of remaining code.

<u>So, here is how the execution of the code will take place:</u>

1. Protractor will reach inside the '*beforeEach*' block first and it will hit '*http://juliemr.github.io/protractor-demo/*' URL in the browser.

2. Now, the flow will move to the 'it' block and function '*multiplyNumbers'* will be called which will, in turn, perform the specified actions in it sending the control back to where the function was called.

3. At last, the assertion will do its job. Now, if we want to tackle multiple elements at the same time, you can use 'element.all' a feature of this tool. It will identify all of the available

elements with the specified locator (by.repeater in this case). It’s up to you what you wish to do with the identified elements. In this case, we are comparing the calculation history with a given number.

4. Since, in the first assertion, we are comparing the calculation history count with ‘2’ even though we performed calculation just once, the assertion will fail. The second assertion, however, will pass as after the second calculation, history count would be ‘2’.

There are many more available types of assertions. Some of which are given below:

a) Get text from a web element and compare it with a certain value:

```
  element(by.locator('someLocator')).getText(text) .then(function() {
  expect(text).toEqual('someData');
  expect(text).not.toEqual('someData');
  expect(text).toContain('someOtherData');
  });
```

b) Verify if a web element is displayed on the page or not:

```
  expect(browser.driver.findElement(by.locator(someLocator))
 .isDisplayed()).toBe(true);
```

5) Handling multiple browsers/windows/tabs:

There can be multiple cases when it comes to handling the browser. Some of these cases are highlighted below:

a) A new tab opens up by clicking on any link

Sometimes, when you click on any link, a new tab opens up and rest of the actions need to take place in the newly opened window. In this case, when you write the code to the point a new tab is opened, you need to implement Window Handler using the following way:

```
 //Get all of the opened windows and store the count in handles
 browser.getAllWindowHandles().then(function(handles) {
 //Now switch the control to the newly opened window
 browser.switchTo().window(handles[1]).then(function() {
 //Write the code which needs to be executed in the new tab
 });
 });
```

First get a count of all the available windows and then use indexing to switch control between the windows. The original window which initiated the new window will have index 0, whereas the subsequent windows will have increasing indexes.

b) Opening an entirely new browser with new session:

When you need to perform some actions on a browser and need to do further actions on a different session of the browser, we need to use the ***forkNewDriverInstance***. In this case, we create a new browser instance with a fresh browser name in the following way:

```
describe('Code to understand assertions/annotations', function() {
//Create a new browser instance
var newBrowser = browser.forkNewDriverInstance();
it('should should open multiple browsers instances', function() {
//Opens a URL in the 1st browser instance
browser.get('http://juliemr.github.io/protractor-demo/');
//Opens a URL in the 2nd browser instance
newBrowser.get('https://www.madewithangular.com/#/');
newBrowser.driver.quit();
 });
});
```

c) <u>Running your test case in multiple browsers:</u>

Running your test case in 2 browsers at a time is something configuration file can do for you. Just add the below code in your configuration file:

As soon as you run this configuration file, you will see tests running in both Firefox and Chrome simultaneously and run logs will be displayed in the command prompt separately.

```
// conf.js
exports.config = {
    framework: 'jasmine',
specs: ['SomeSpec.js'],
multiCapabilities: [{
 browserName: 'firefox'
}, {
browserName: 'chrome'
}]
}
```

<u>6) Use Page Objects to make your framework even better:</u>

This tool is good on its own but it becomes invincible when combined with Page Object Model(POM). Most of its shortcomings (if any) are overcome with the page object model. Moreover, POM also helps maintain your project in a more structured way.

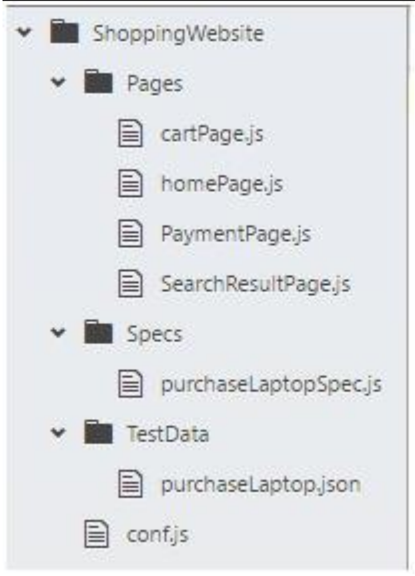
If you do not know what POM is, no worries. POM is a way to segregate your test case on the basis of the pages.

Take this example:
There is a shopping website. Your test case is to select a product, add it to cart and then purchase it.

Now, there are two ways to manage your test script code for this:

1. Write a plain test case with all of the locators on the same page your logic is written in,
2. Write all of your flow of test case, your logic in your spec file and segregate your locators and test data in different files. Each web page will have an equivalent .js page file. In this way, your code will be structured and if there is any other test case which requires the same locator, you do not need to write these locators again, just import this locator file and use it as per your need. Maintaining your test cases can be a real pain. If you use POM, your code will be in a much more structured manner.

Here is an example of using the page object model:



```
ShoppingWebsite
    Pages
        cartPage.js
        homePage.js
        PaymentPage.js
        SearchResultPage.js
    Specs
        purchaseLaptopSpec.js
    TestData
        purchaseLaptop.json
    conf.js
```

This is the flow is in the above snapshot:
1. There is a test case which purchases Laptop. The code for the flow and logic will be in the purchaseLaptopSpec.js.
2. All the pages which are encountered to purchase the Laptop will have 1 '.js' file with a proper title. All of the elements required to be used for purchasing the Laptop, their locators will be inside the respective page file.
3. The data required for this test case can be saved in the TestData folder either in a '.json' format or in the excel format.

4. Once done with the pages and locators, just import these files in your spec file to use the locator/test data and you are all set with your test case.

7) **Reporting**

NPM (Node Package Manager) provides various reporting packages, with which screenshot of each test step can be captured and also, once the test run completes, it will generate an HTML report for you.

7)**Integrate with other powerful tools such as Git/Jenkins/Browserstack/Grunt**

There are multiple tools available in the market to help make your test cases even better. Git/Jenkins/BrowserStack/Grunt are some such tools which add significant value to your normal Protractor test scripts. And the best part is that you don't have to touch your spec file to integrate your Protractor with any of these tools. It is your configuration file, which will take all of these things for you.

Git is a very powerful version control tool. It is always a best practice to keep your code in Git if there are multiple developers involved.

Jenkins is a continuous integration tool with which, you can schedule your test cases and run it as per your need. Protractor scripts can also be configured with Jenkins. Best use of running your test cases on Jenkins is that it is very fast and also you can run multiple test cases at a time.

BrowserStack is a cross-browser testing tool which can also be used to test your applications across different browsers. It can also be integrated with Protractor by adding the browserStack credentials in your config file.

Grunt is a JavaScript task runner. It provides you with the ability to perform several tasks for you. Its awesomeness is that there are more than 4000 tasks and you can create even more tasks as per your requirement. Following are few of the important daily use tasks we can use Grunt for:

1. List down all of the coding best practices and inform immediately whenever you violate any of these.
2. To create multiple spec files at run time. **For example**, if there is any test case that you wish to run several times (ranging from 1 to any number). This might seem unnecessary at this point of time but think of running any shopping website checkout flow to be run on every single available country. It would be tedious to create multiple specs manually. So, let Grunt do this for you.
3. The watch feature. You write a test case and every time as soon as save your code after making any change in it, you want your test case to run, Grunt has got it.
4. Concatenating multiple files.

Just give it a try and you will love it.

Page Object Model:

- **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
- Under this model, for each web page in the application, there should be corresponding page class.
- This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
- Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be waitForPaymentScreenDisplay().

Advantages of POM:

1. Page Object Patten says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. The Second benefit is the **object repository is independent of test cases**, so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.
3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. **Methods** get **more realistic names** which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

Example:

```
#test.feature


Feature: Angular Task List


  As a basic user


  I should be able to add and remove tasks from the task list


  So that I can keep track of my tasks

  Scenario: Protractor and Cucumber Test

    Given I go to "https://angularjs.org/"

    When I add "Be Awesome" in the task field

    And I click the add button

    Then I should see my new task in the list
```

```
//Page Object:

'use strict';

module.exports = {

  angularHomepage: {

    taskList: element(by.model('todoList.todoText')),

    taskButton: element(by.css('[value="add"]')),

    todoList: element.all(by.repeater('todo in todoList.todos'))

  }

});
```

```
// methods needed to interact with these elements.
```

```
'use strict';

module.exports = {



  angularHomepage: {

    taskList: element(by.model('todoList.todoText')),

    taskButton: element(by.css('[value="add"]')),

    todoList: element.all(by.repeater('todo in todoList.todos'))

  },



  go: function(site) {

    browser.get(site);

  },



  addTask: function(task) {

    var angular = this.angularHomepage;



    angular.taskList.sendKeys(task);

  },
  submitTask: function() {

    var angular = this.angularHomepage;

    angular.taskButton.click();

  }

};
```

```
// features/step_definitions/stepDefinitions.js



var angularPage = require('../pages/angularPage.js');

var chai = require('chai');

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



chai.use(chaiAsPromised);

var expect = chai.expect;



module.exports = function() {

  this.Given(/^I go to "([^"]*)"$/, function(site) {

    angularPage.go(site);
  });



  this.When(/^I add "([^"]*)" in the task field$/, function(task) {

    angularPage.addTask(task);

  });



  this.When(/^I click the add button$/, function() {

    angularPage.submitTask();

  });



  this.Then(/^I should see my new task in the list$/, function(callback) {

    var todoList = angularPage.angularHomepage.todoList;

    expect(todoList.count()).to.eventually.equal(3);

    expect(todoList.get(2).getText()).to.eventually.equal('Be Awesome')

      .and.notify(callback);

  });

};
```

<u>Organizing Page Objects:</u>

There are a few approaches you can consider: by component, by page, and/or by user workflow. It all depends on your team's needs, but these provide a good starting point for the structure of your page objects.

Organizing locators by page or view may be the simplest approach as you build out your suite. For example, you would create a page object for every page accessible within your application. All elements related to that page would go in their appropriate file.

- pages
    - homePage.js
    - aboutPage.js
    - contactPage.js

Another option is organizing locators by component. These can be modules that are used across your application such as a search feature. Of course, you can incorporate both by page and component.

- pages
    - searchComponent.js
    - homePage.js
    - aboutPage.js
    - contactPage.js

Finally, it may be beneficial to organize by a particular user workflow like creating a new user. These page objects would essentially house all methods needed to accomplish the workflow.

- pages
    - searchComponent.js
    - homePage.js
    - aboutPage.js
    - contactPage.js
    - createUserFlow.js

The level of complexity for page objects can vary based on the need. The point of using a POM is reducing code duplication and increasing efficiency and maintainability of your testing suite. Before creating a large suite of tests, first consider the approach your team should take in building your automation suite.

Installation:

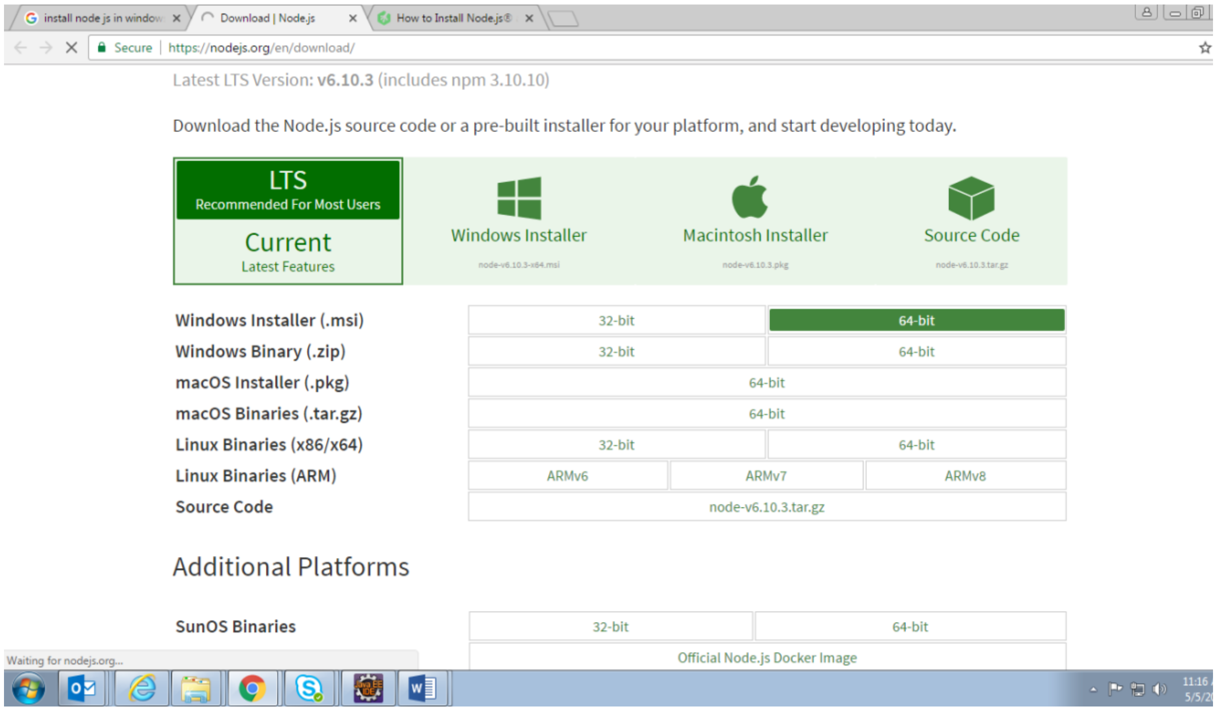
Install JDK & Node.JS

Download and install the **Java JDK** for your system and if it is already installed please ignore.

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Node.JS:

1. Go to the site https://nodejs.org/en/download/ and download necessary binary file or if installation file exists double click on that file and go to step-3.


install node js in window
Download | Node.js
How to Install Node.js®
Secure
https://nodejs.org/en/download/
Waiting for nodejs.org...

Latest LTS Version: **v6.10.3** (includes npm 3.10.10)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

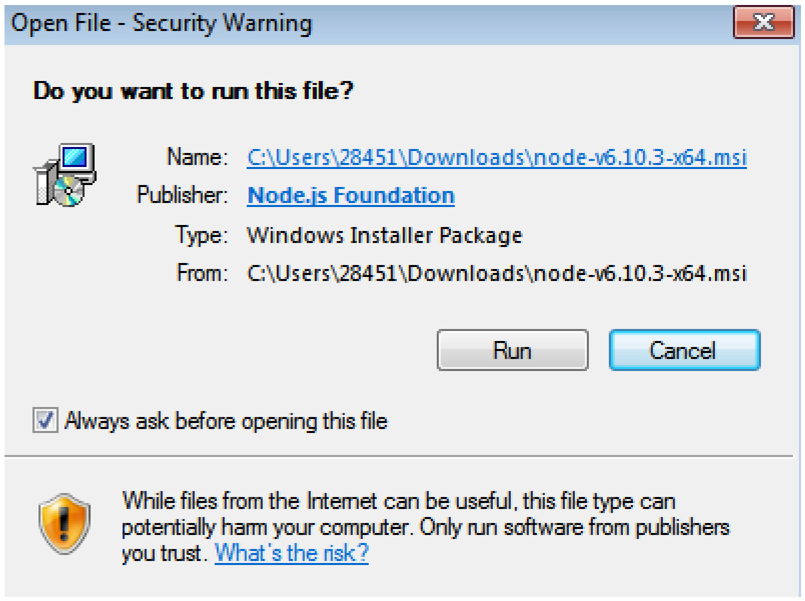
LTS Recommended For Most Users / Current Latest Features	Windows Installer	Macintosh Installer	Source Code
	node-v6.10.3-x64.msi	node-v6.10.3.pkg	node-v6.10.3.tar.gz

Windows Installer (.msi)	32-bit		64-bit
Windows Binary (.zip)	32-bit		64-bit
macOS Installer (.pkg)	64-bit		
macOS Binaries (.tar.gz)	64-bit		
Linux Binaries (x86/x64)	32-bit		64-bit
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v6.10.3.tar.gz		

Additional Platforms

SunOS Binaries	32-bit	64-bit
	Official Node.js Docker Image	

2. Double click on the downloaded .msi file to start the installation. Click "Run" button.



Open File - Security Warning

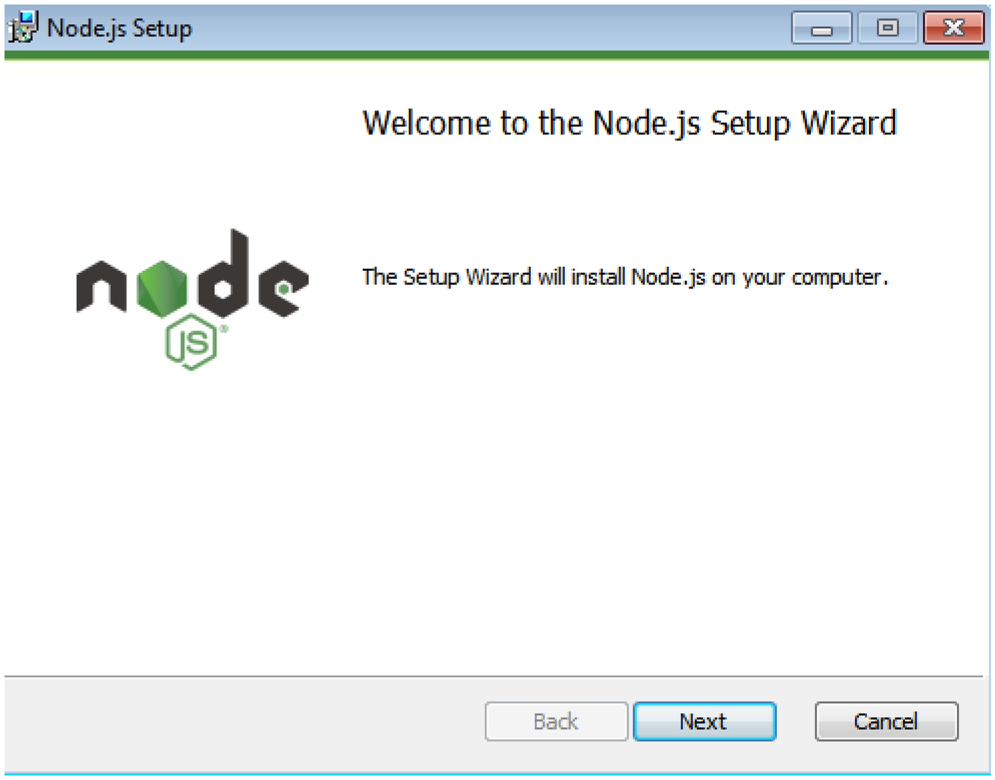
Do you want to run this file?

Name: C:\Users\28451\Downloads\node-v6.10.3-x64.msi
Publisher: Node.js Foundation
Type: Windows Installer Package
From: C:\Users\28451\Downloads\node-v6.10.3-x64.msi

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

3. Click "Next" button

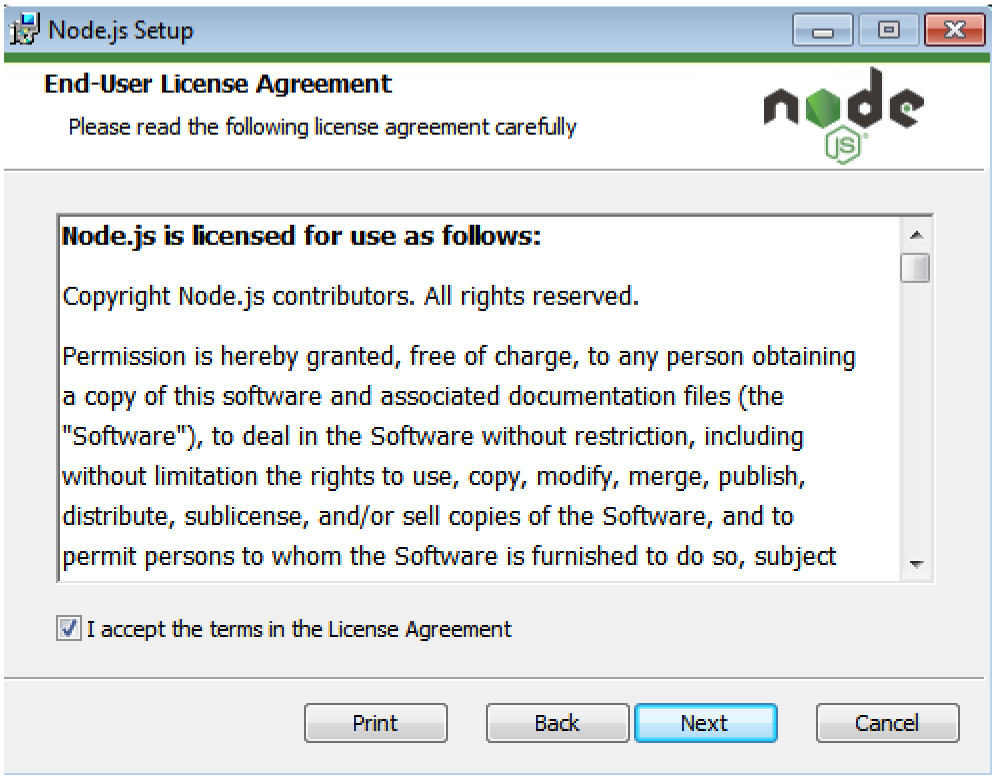


Node.js Setup

Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

4. Accept the license agreement check box and click on "Next" button


Node.js Setup

End-User License Agreement

Please read the following license agreement carefully

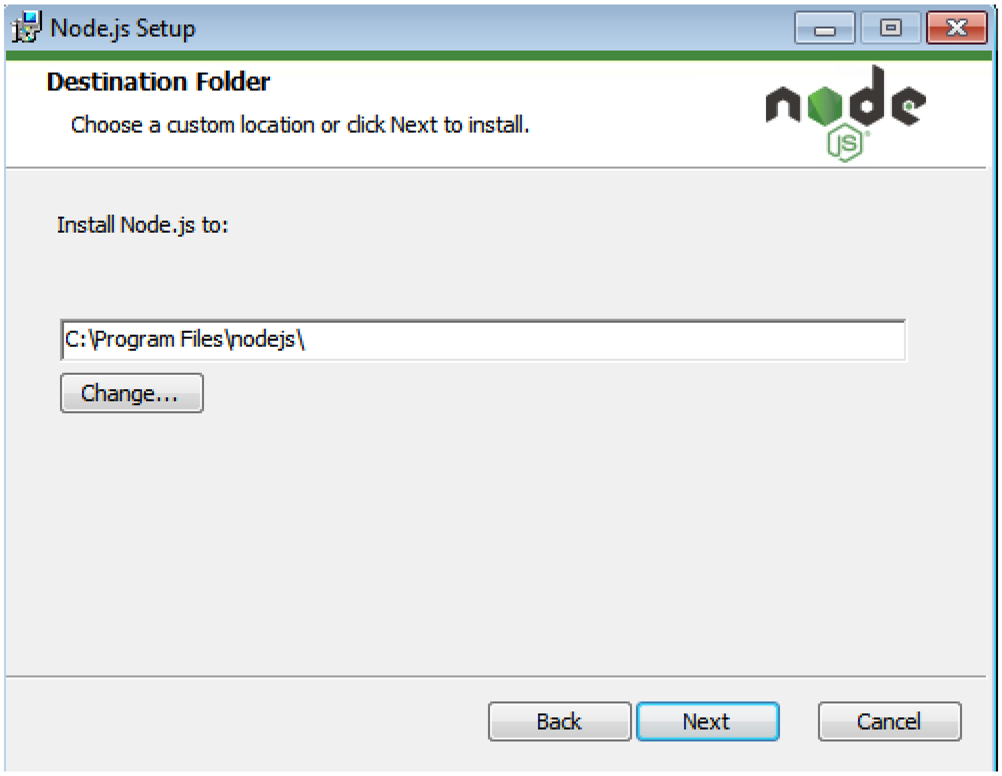
Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

[✓] I accept the terms in the License Agreement

5. Choose the file location for the installation of Node.JS and click “Next” button


Node.js Setup

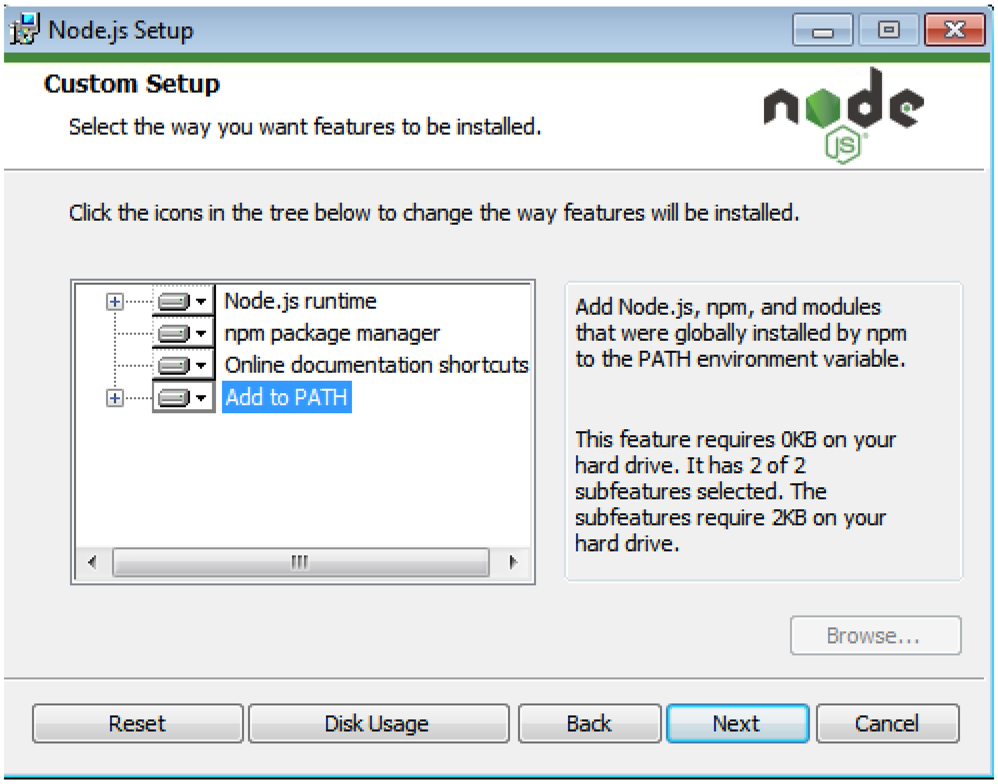
Destination Folder

Choose a custom location or click Next to install.

Install Node.js to:

C:\Program Files\nodejs\

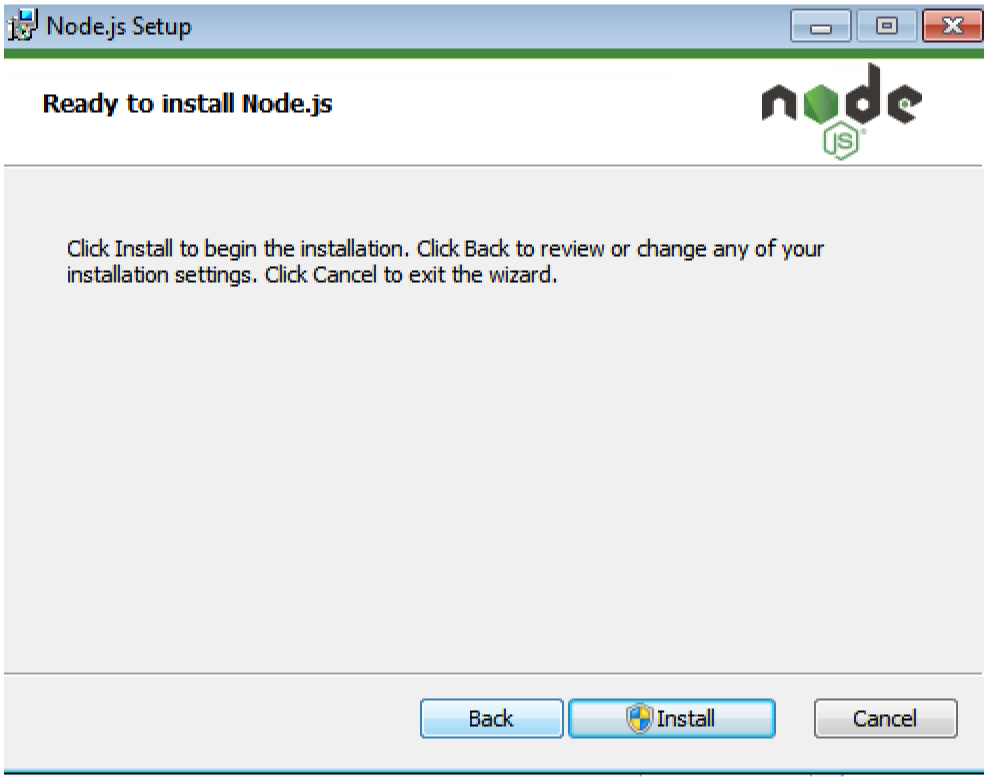
6. Select the feature and click on “Next” button


Node.js Setup
Custom Setup
Select the way you want features to be installed.
node JS
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH
Browse...
Reset
Disk Usage
Back
Next
Cancel

Add Node.js, npm, and modules that were globally installed by npm to the PATH environment variable.

This feature requires 0KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 2KB on your hard drive.

7. Click on “Install” button

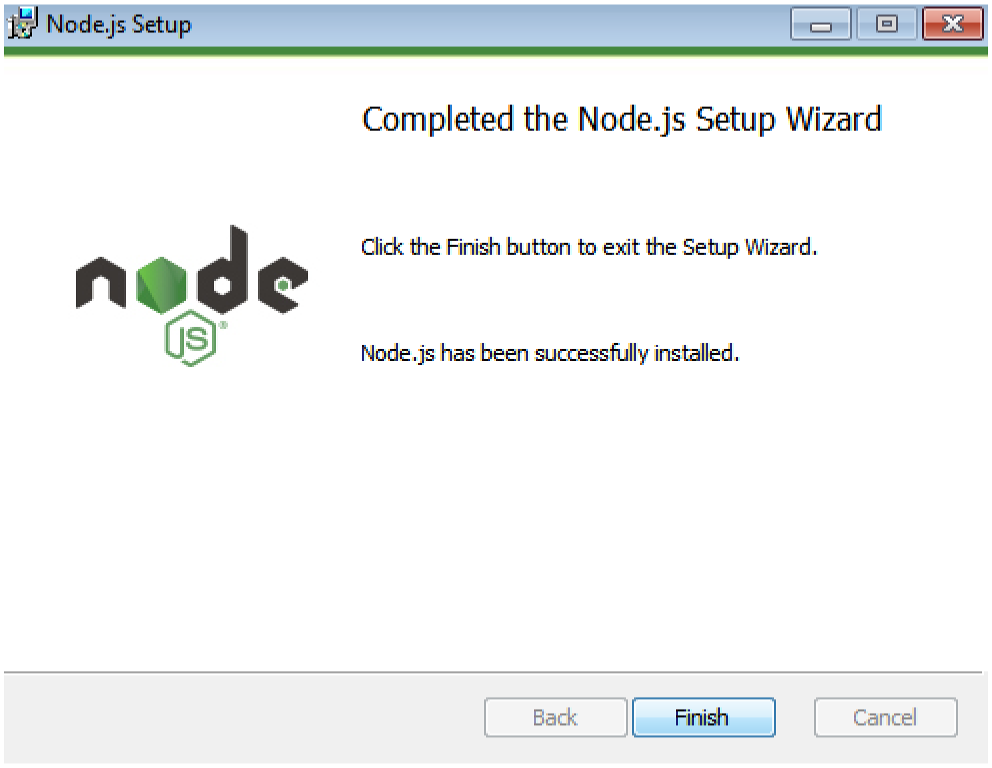


Node.js Setup

Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

8. Click on Finish button



Node.js Setup

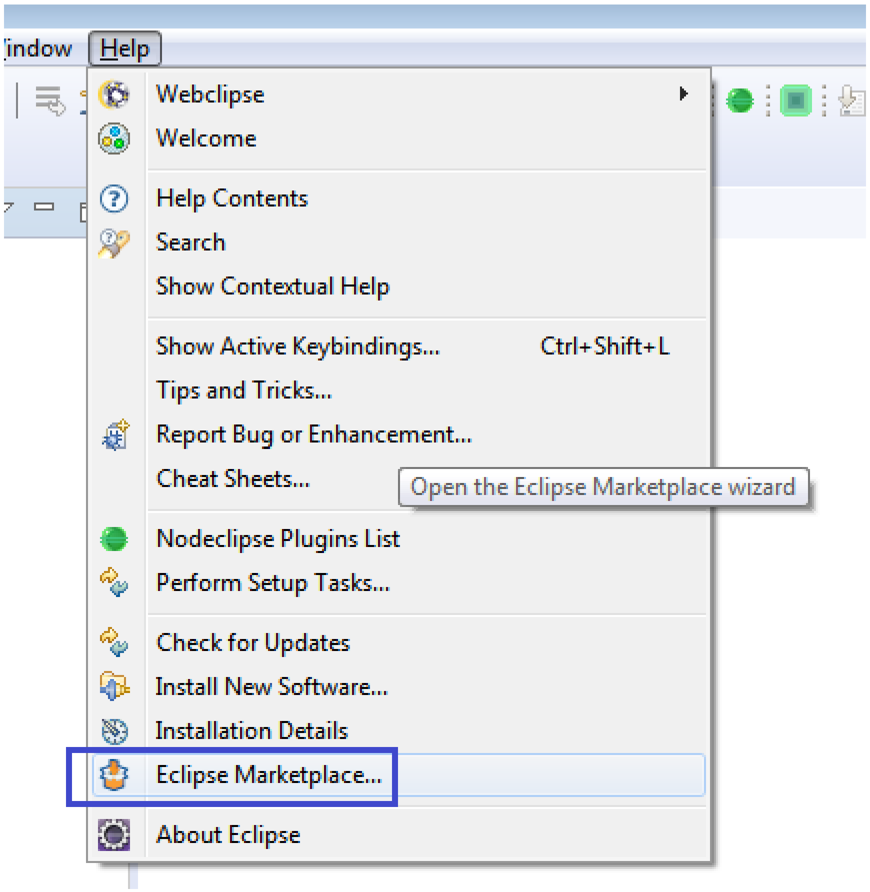
Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

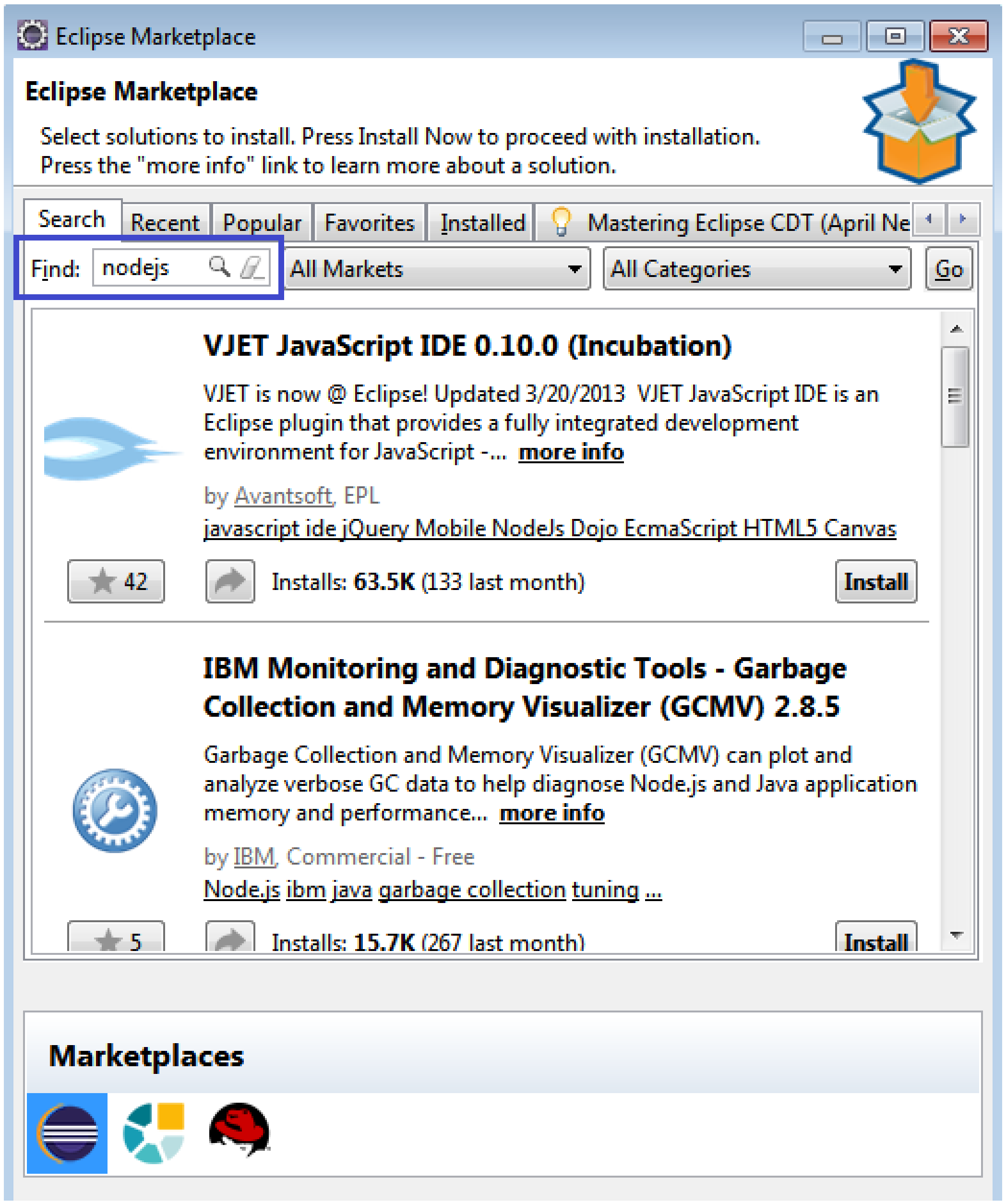
Node.js has been successfully installed.

Install the NodeJS plug-in in Eclipse and create a new Node.JS project

1. Open Eclipse IDE (Version: Neon)
2. Click on Help → Eclipse Marketplace…



3. Enter search text as "nodejs" and click on search icon


Eclipse Marketplace
Eclipse Marketplace
Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.
Search
Recent
Popular
Favorites
Installed
Mastering Eclipse CDT (April Ne
Find: nodejs
All Markets
All Categories
Go

VJET JavaScript IDE 0.10.0 (Incubation)

VJET is now @ Eclipse! Updated 3/20/2013 VJET JavaScript IDE is an Eclipse plugin that provides a fully integrated development environment for JavaScript -... more info

by Avantsoft, EPL

javascript ide jQuery Mobile NodeJs Dojo EcmaScript HTML5 Canvas

42 Installs: **63.5K** (133 last month) Install

IBM Monitoring and Diagnostic Tools - Garbage Collection and Memory Visualizer (GCMV) 2.8.5

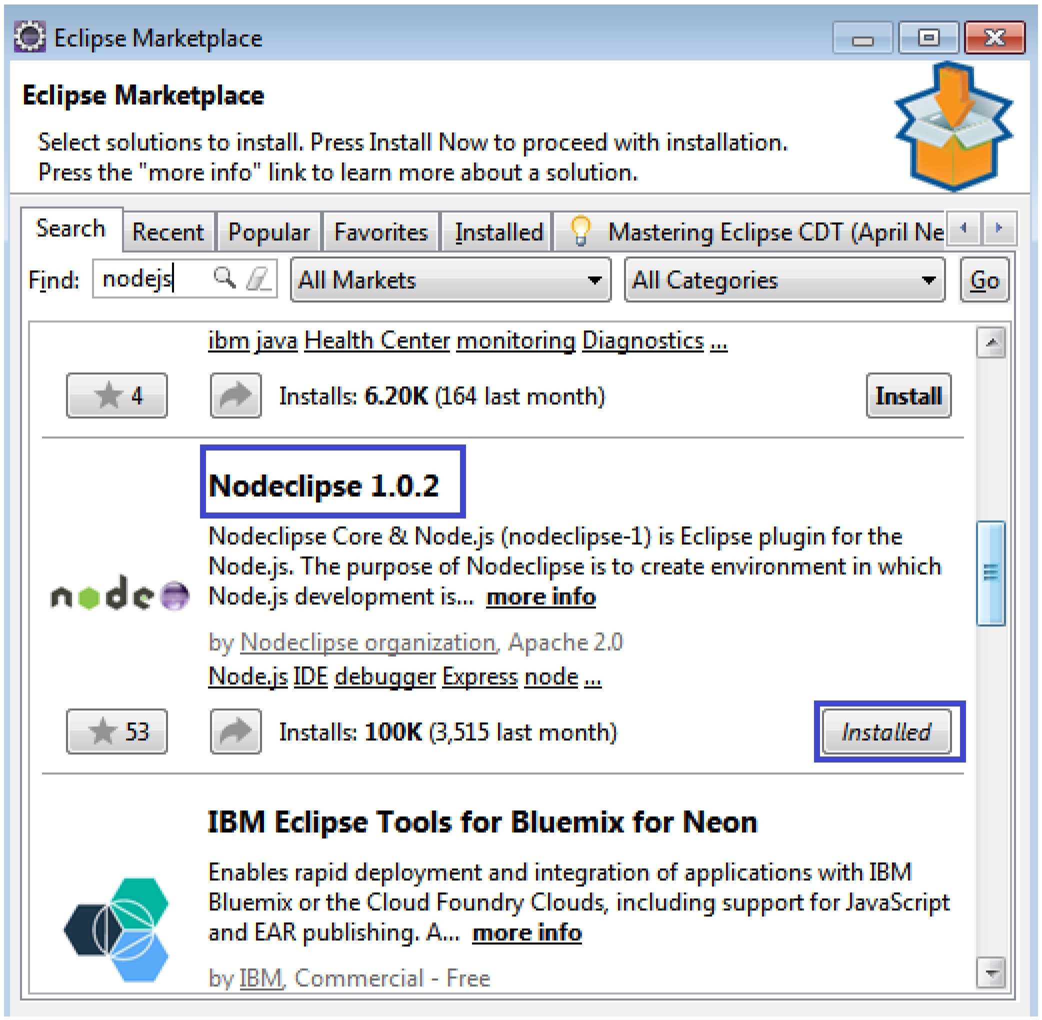
Garbage Collection and Memory Visualizer (GCMV) can plot and analyze verbose GC data to help diagnose Node.js and Java application memory and performance... more info

by IBM, Commercial - Free

Node.js ibm java garbage collection tuning ...

5 Installs: **15.7K** (267 last month) Install

4. Verify the “Nodeclipse 1.0.2” plug-in and if it is installed by default, close the window. If it not installed click on “Install” button.


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Search | Recent | Popular | Favorites | Installed | Mastering Eclipse CDT (April Ne

Find: nodejs | All Markets | All Categories | Go

ibm java Health Center monitoring Diagnostics ...

★ 4 | Installs: **6.20K** (164 last month) | **Install**

Nodeclipse 1.0.2

Nodeclipse Core & Node.js (nodeclipse-1) is Eclipse plugin for the Node.js. The purpose of Nodeclipse is to create environment in which Node.js development is... **more info**

by Nodeclipse organization, Apache 2.0

Node.js IDE debugger Express node ...

★ 53 | Installs: **100K** (3,515 last month) | *Installed*

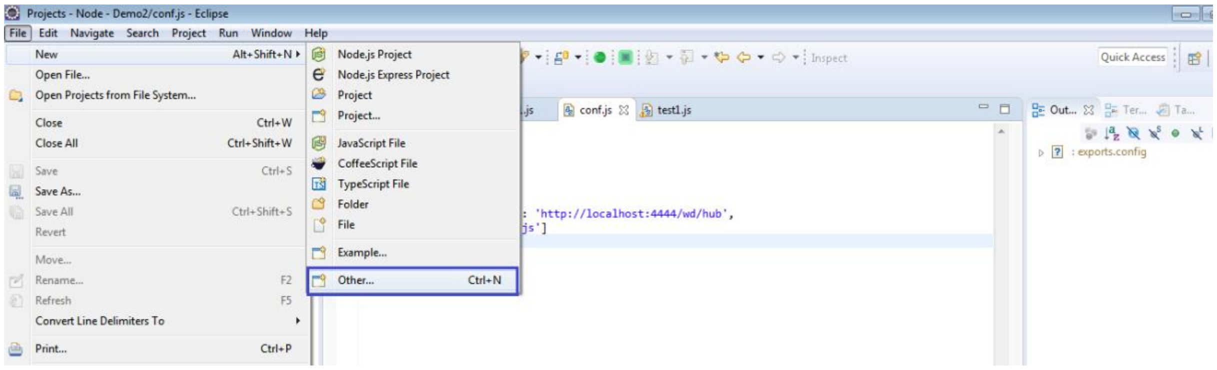
IBM Eclipse Tools for Bluemix for Neon

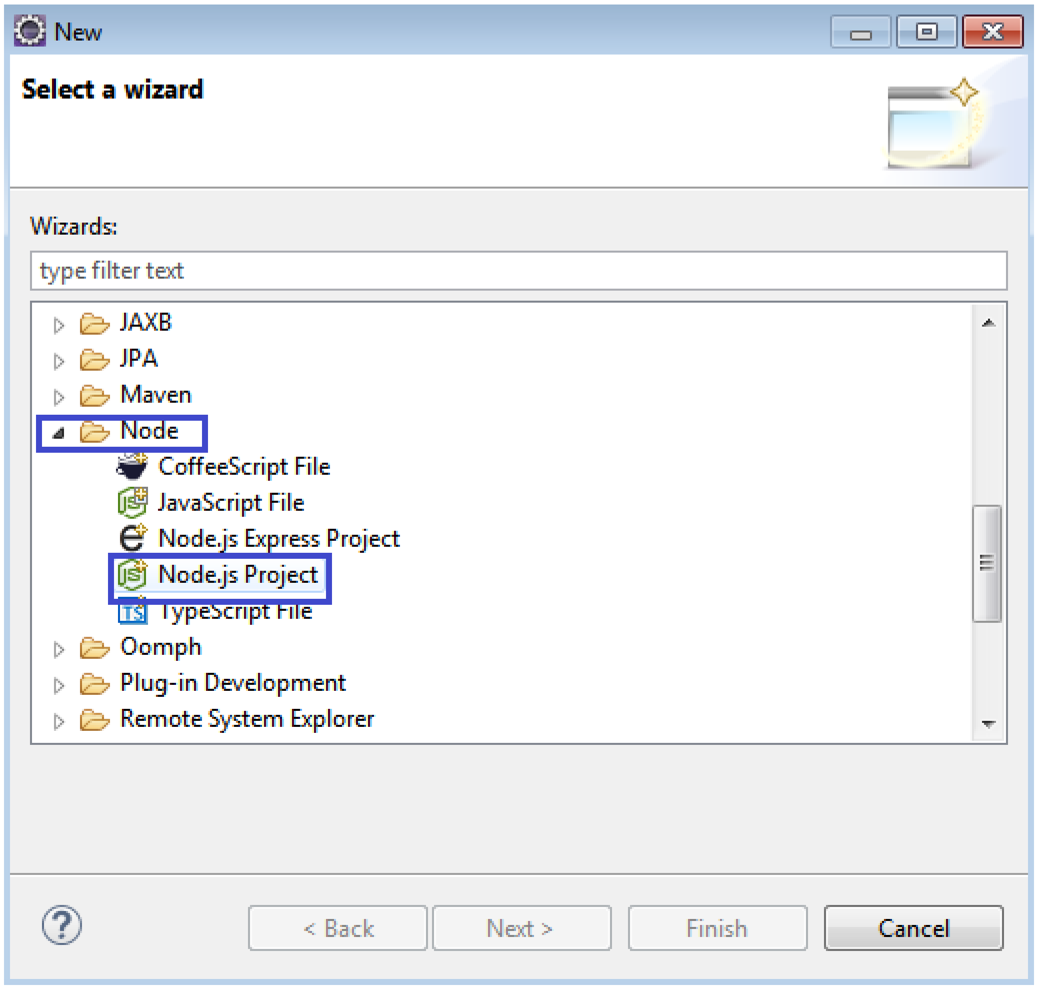
Enables rapid deployment and integration of applications with IBM Bluemix or the Cloud Foundry Clouds, including support for JavaScript and EAR publishing. A... **more info**

by IBM, Commercial - Free

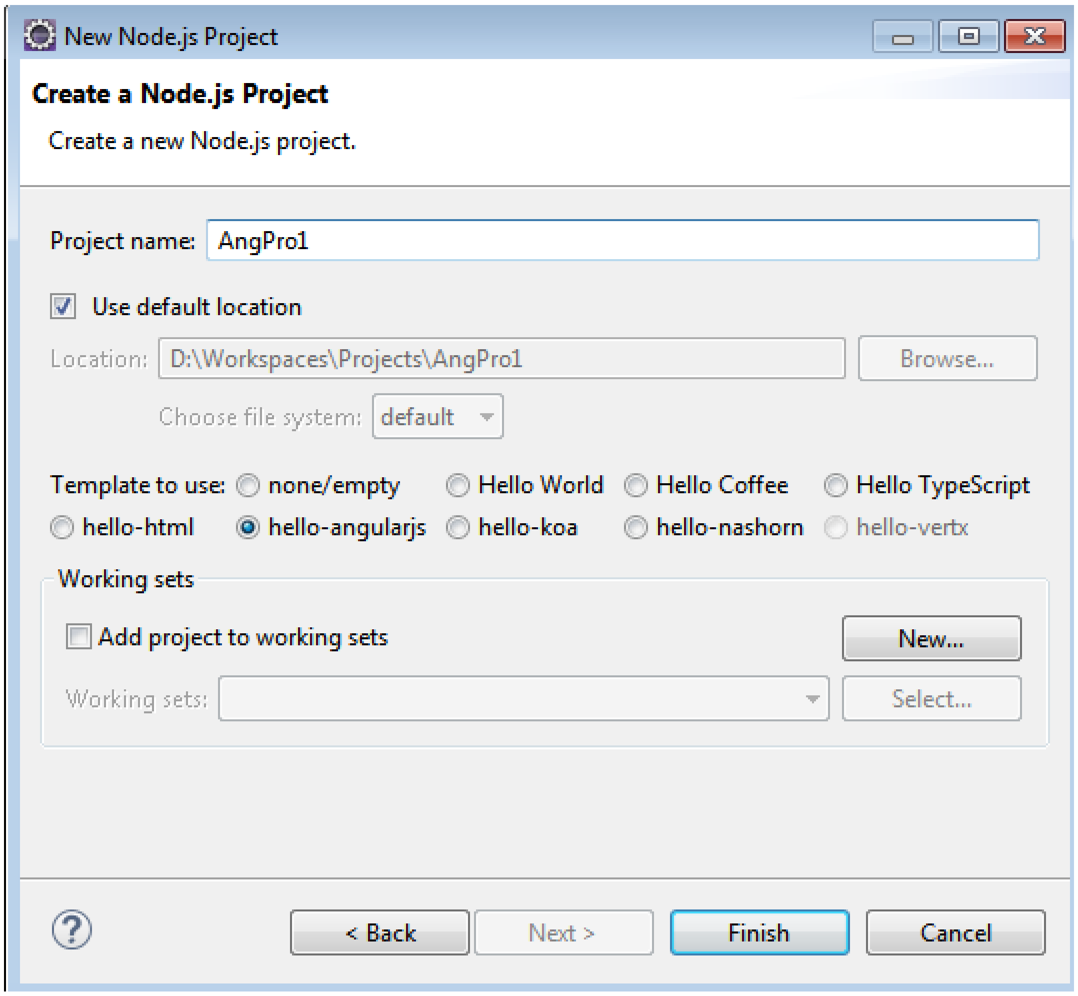
Creating new Node.js project in Eclipse

1. Create a new Node.js project from File > New > Other > Node > Node.js Project





2. Click on Next button
3. Enter the "Project Name" in text box and select any default template


New Node.js Project

Create a Node.js Project

Create a new Node.js project.

Project name: AngPro1

[✓] Use default location

Location: D:\Workspaces\Projects\AngPro1

Choose file system: default

Template to use: none/empty, Hello World, Hello Coffee, Hello TypeScript, hello-html, [✓ hello-angularjs], hello-koa, hello-nashorn, hello-vertx

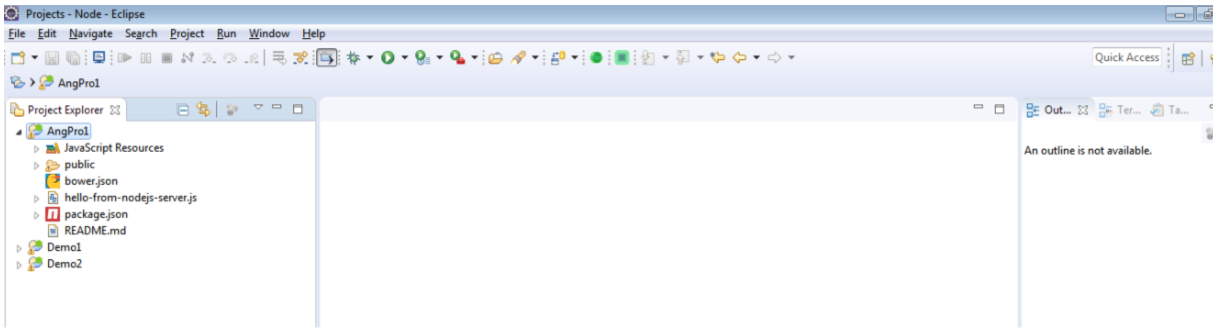
Working sets

[ ] Add project to working sets

Working sets:

4. Click on Finish button

New project is created and this project will display in eclipse project explorer.


Projects - Node - Eclipse

```
Project Explorer
◢ AngPro1
    ▷ JavaScript Resources
    ▷ public
      bower.json
    ▷ hello-from-nodejs-server.js
    ▷ package.json
      README.md
▷ Demo1
▷ Demo2
```

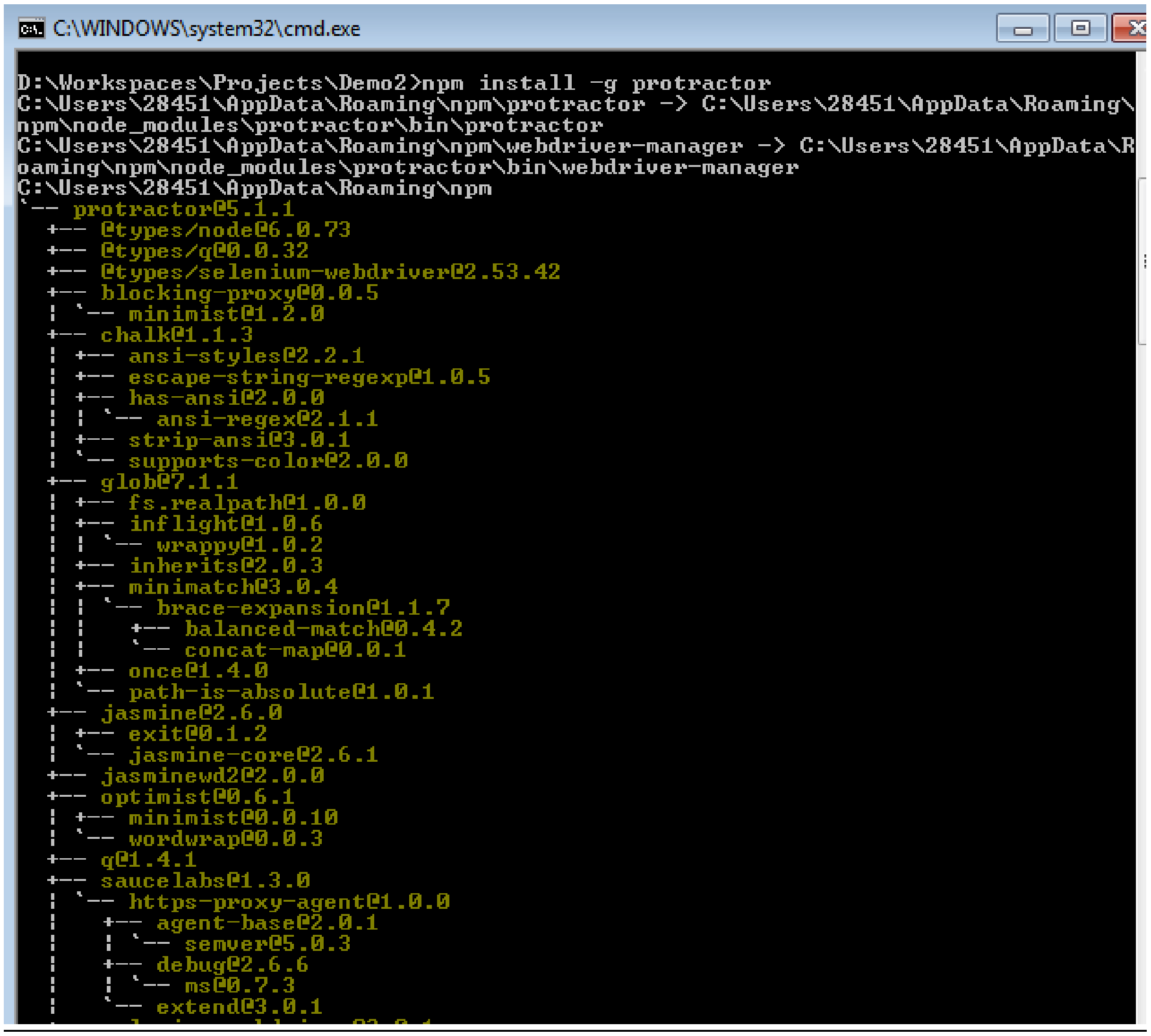
An outline is not available.

<u>Starting with Protractor</u>

Open the command prompt and go to created project location path

1. Use “npm” to install Protractor globally with below command

npm install –g protractor



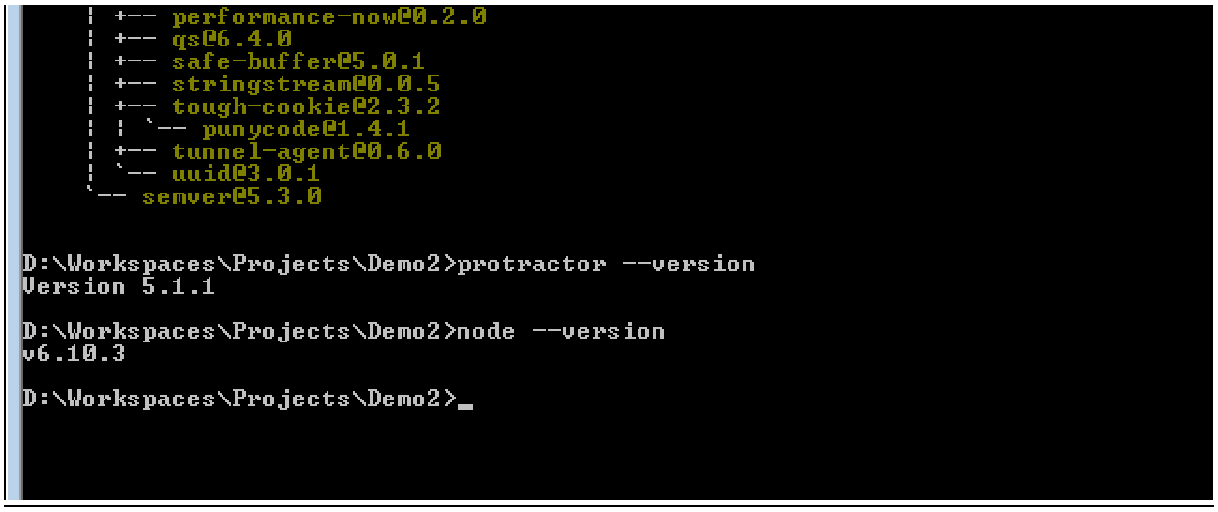
C:\WINDOWS\system32\cmd.exe

```
D:\Workspaces\Projects\Demo2>npm install -g protractor
C:\Users\28451\AppData\Roaming\npm\protractor -> C:\Users\28451\AppData\Roaming\
npm\node_modules\protractor\bin\protractor
C:\Users\28451\AppData\Roaming\npm\webdriver-manager -> C:\Users\28451\AppData\R
oaming\npm\node_modules\protractor\bin\webdriver-manager
C:\Users\28451\AppData\Roaming\npm
`-- protractor@5.1.1
  +-- @types/node@6.0.73
  +-- @types/q@0.0.32
  +-- @types/selenium-webdriver@2.53.42
  +-- blocking-proxy@0.0.5
  ¦ `-- minimist@1.2.0
  +-- chalk@1.1.3
  ¦ +-- ansi-styles@2.2.1
  ¦ +-- escape-string-regexp@1.0.5
  ¦ +-- has-ansi@2.0.0
  ¦ ¦ `-- ansi-regex@2.1.1
  ¦ +-- strip-ansi@3.0.1
  ¦ `-- supports-color@2.0.0
  +-- glob@7.1.1
  ¦ +-- fs.realpath@1.0.0
  ¦ +-- inflight@1.0.6
  ¦ ¦ `-- wrappy@1.0.2
  ¦ +-- inherits@2.0.3
  ¦ +-- minimatch@3.0.4
  ¦ ¦ `-- brace-expansion@1.1.7
  ¦ ¦   +-- balanced-match@0.4.2
  ¦ ¦   `-- concat-map@0.0.1
  ¦ +-- once@1.4.0
  ¦ `-- path-is-absolute@1.0.1
  +-- jasmine@2.6.0
  ¦ +-- exit@0.1.2
  ¦ `-- jasmine-core@2.6.1
  +-- jasminewd2@2.0.0
  +-- optimist@0.6.1
  ¦ +-- minimist@0.0.10
  ¦ `-- wordwrap@0.0.3
  +-- q@1.4.1
  +-- saucelabs@1.3.0
  ¦ `-- https-proxy-agent@1.0.0
  ¦   +-- agent-base@2.0.1
  ¦   ¦ `-- semver@5.0.3
  ¦   +-- debug@2.6.6
  ¦   ¦ `-- ms@0.7.3
  ¦   `-- extend@3.0.1
```

2. Check the Protractor and node versions with below commands

Protractor –version

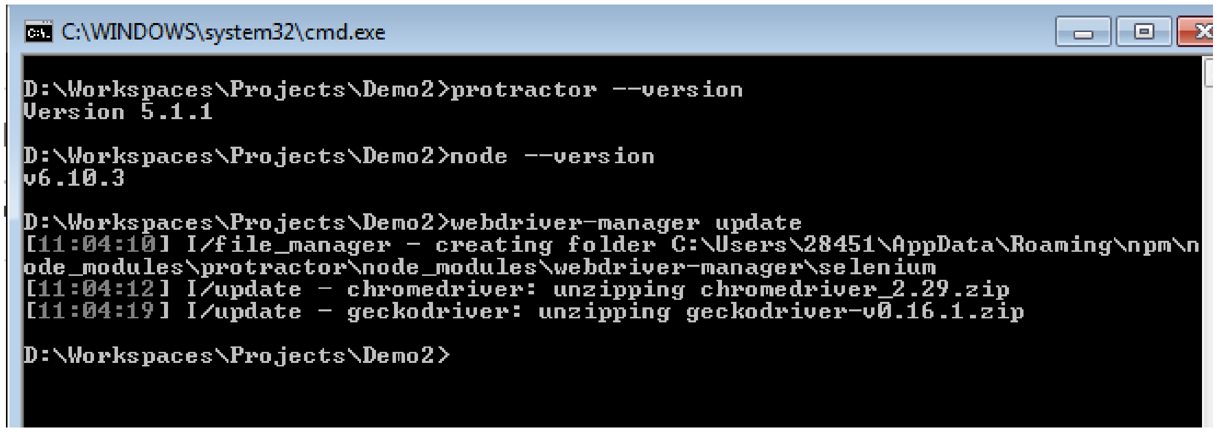
Node –version



```
    ¦ +-- performance-now@0.2.0
    ¦ +-- qs@6.4.0
    ¦ +-- safe-buffer@5.0.1
    ¦ +-- stringstream@0.0.5
    ¦ +-- tough-cookie@2.3.2
    ¦ ¦ `-- punycode@1.4.1
    ¦ +-- tunnel-agent@0.6.0
    ¦ `-- uuid@3.0.1
    `-- semver@5.3.0

D:\Workspaces\Projects\Demo2>protractor --version
Version 5.1.1

D:\Workspaces\Projects\Demo2>node --version
v6.10.3

D:\Workspaces\Projects\Demo2>_
```

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

webdriver-manager update



```
D:\Workspaces\Projects\Demo2>protractor --version
Version 5.1.1

D:\Workspaces\Projects\Demo2>node --version
v6.10.3

D:\Workspaces\Projects\Demo2>webdriver-manager update
[11:04:10] I/file_manager - creating folder C:\Users\28451\AppData\Roaming\npm\n
ode_modules\protractor\node_modules\webdriver-manager\selenium
[11:04:12] I/update - chromedriver: unzipping chromedriver_2.29.zip
[11:04:19] I/update - geckodriver: unzipping geckodriver-v0.16.1.zip

D:\Workspaces\Projects\Demo2>
```

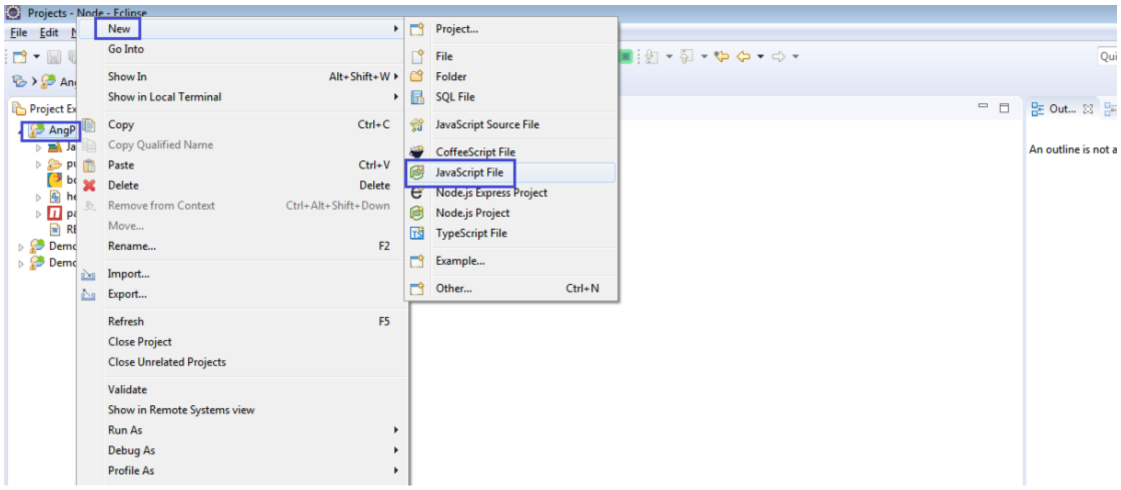
4. Now start up a server with below command

webdriver-manager start

<u>Write sample test cases</u>

Below are the examples of basic configuration and spec files and Protractor need these two files to run.

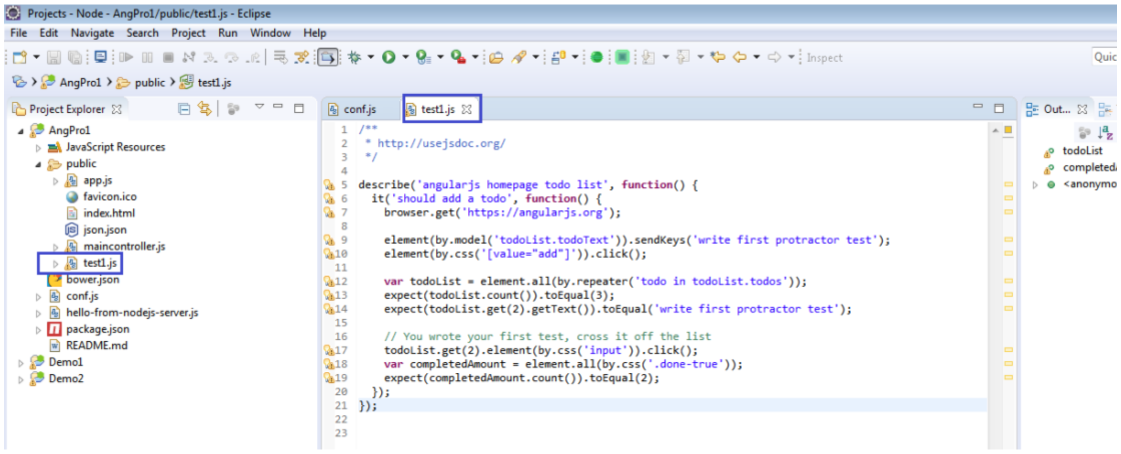
1. Create new java script file from Right click on created Project name from explorer > New > Java script File.



2. Enter the spec file name as “test1.js” and click on finish button. Enter the below sample code in that file.

```
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**.

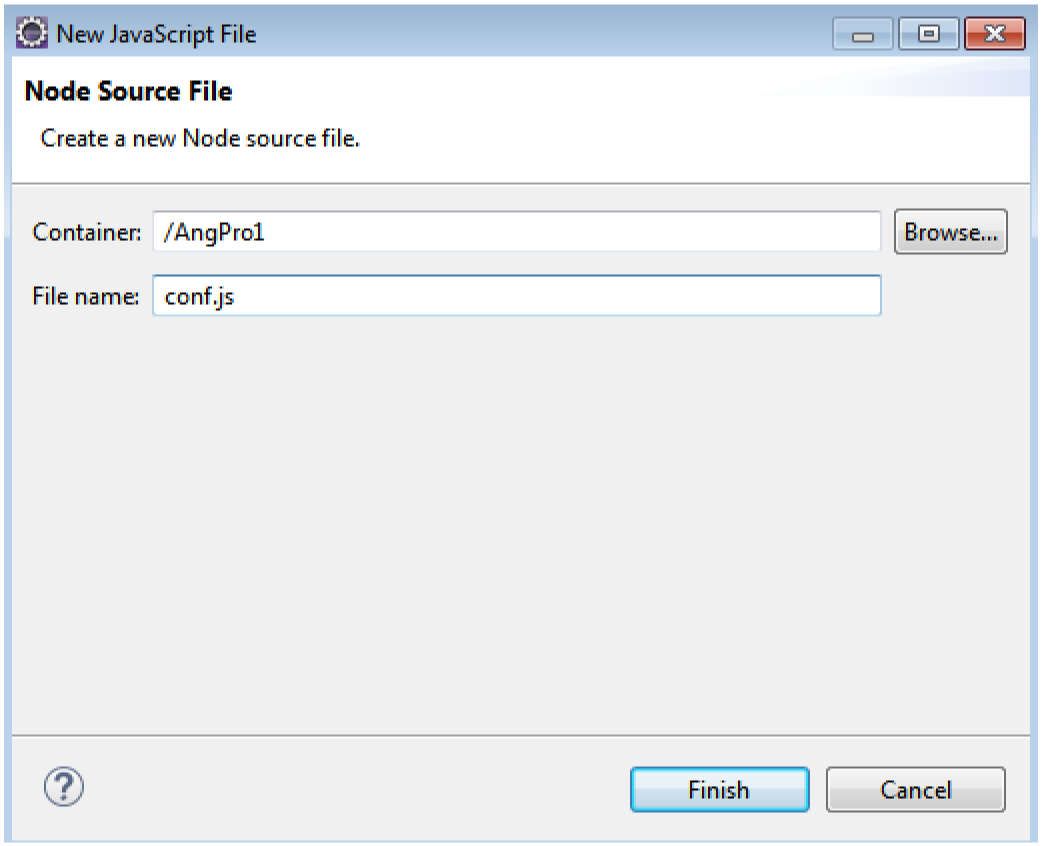


```
/**
 * http://usejsdoc.org/
 */

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);
  });
});
```

3. Create a new config java script file and named as “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.

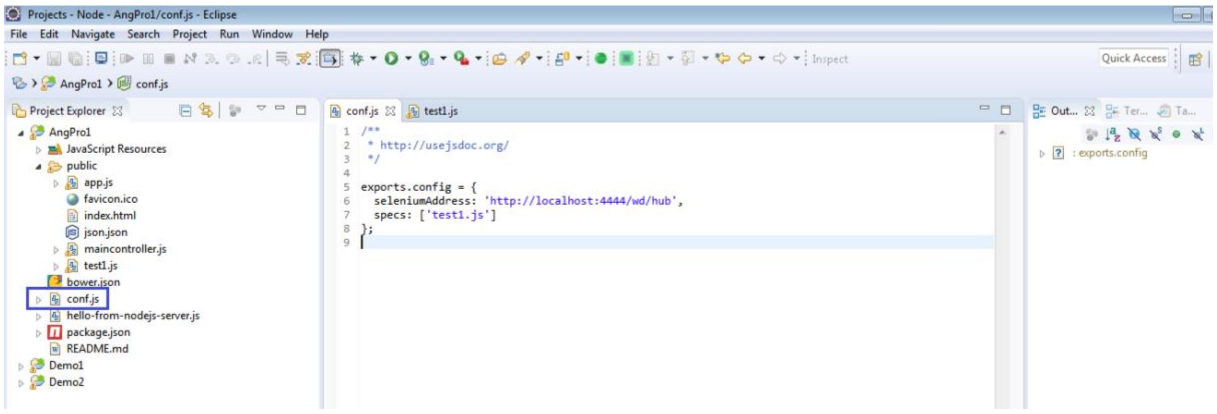


Node Source File

Create a new Node source file.

Container: /AngPro1

File name: conf.js



```
/**
 * http://usejsdoc.org/
 */

exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['test1.js']
};
```

4. Run test by below command in command prompt

protractor conf.js

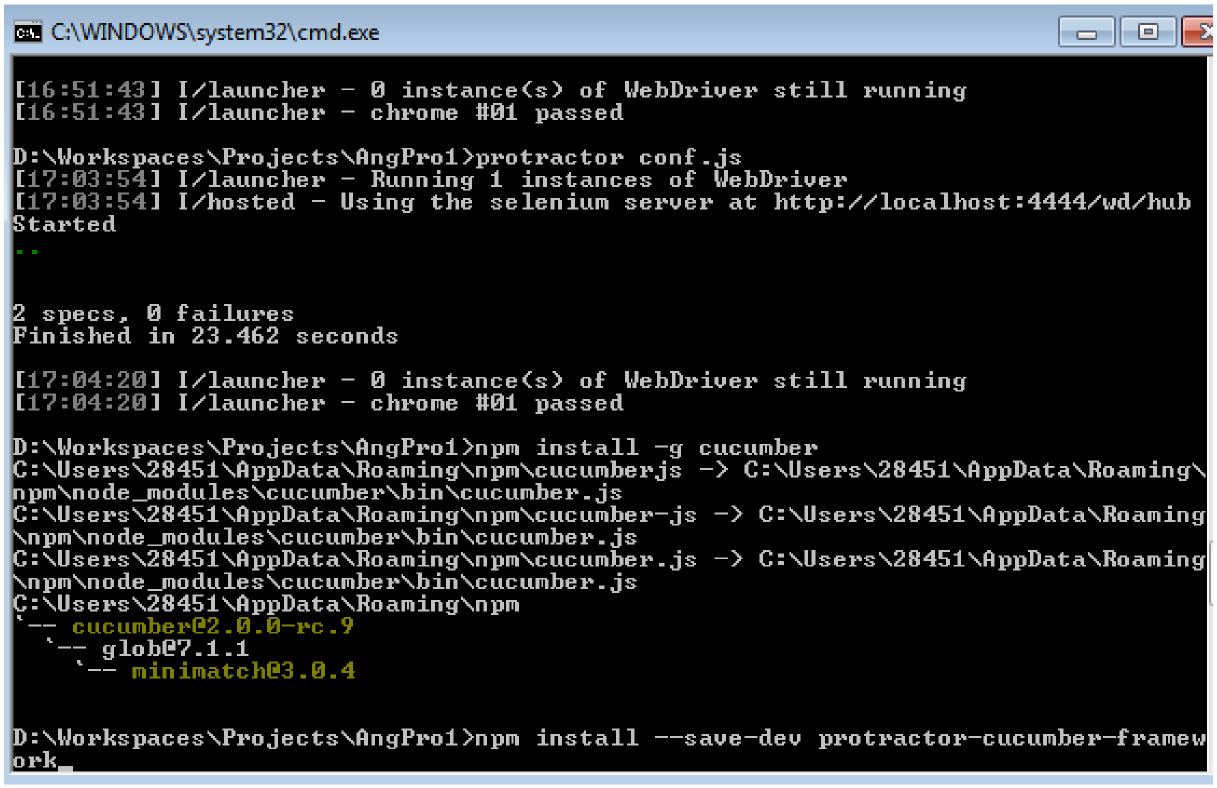
You should see a Chrome browser window open up and navigate to the todo list in the AngularJS page, then close itself (this should be very fast!).

CUCUMBER SETUP

Cucumber needs to be installed at the root of the project and make sure it is installed in the same place as Protractor.

Install Cucumber globally with below command

npm install -g cucumber

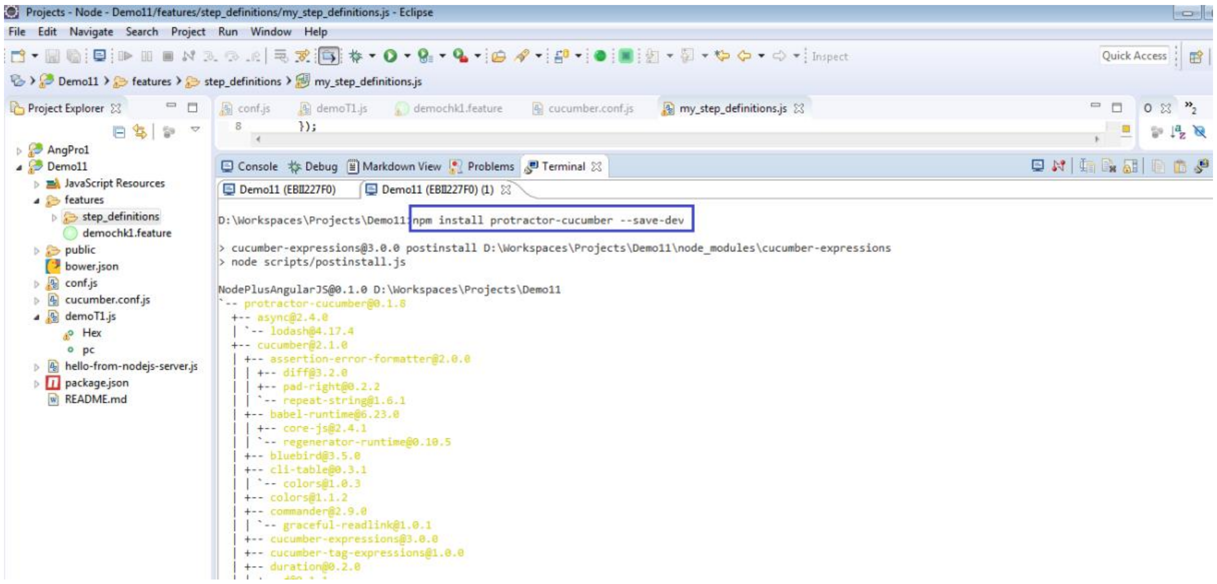


C:\WINDOWS\system32\cmd.exe

```
[16:51:43] I/launcher - 0 instance(s) of WebDriver still running
[16:51:43] I/launcher - chrome #01 passed

D:\Workspaces\Projects\AngPro1>protractor conf.js
[17:03:54] I/launcher - Running 1 instances of WebDriver
[17:03:54] I/hosted - Using the selenium server at http://localhost:4444/wd/hub
Started
..


2 specs, 0 failures
Finished in 23.462 seconds

[17:04:20] I/launcher - 0 instance(s) of WebDriver still running
[17:04:20] I/launcher - chrome #01 passed

D:\Workspaces\Projects\AngPro1>npm install -g cucumber
C:\Users\28451\AppData\Roaming\npm\cucumberjs -> C:\Users\28451\AppData\Roaming\
npm\node_modules\cucumber\bin\cucumber.js
C:\Users\28451\AppData\Roaming\npm\cucumber-js -> C:\Users\28451\AppData\Roaming
\npm\node_modules\cucumber\bin\cucumber.js
C:\Users\28451\AppData\Roaming\npm\cucumber.js -> C:\Users\28451\AppData\Roaming
\npm\node_modules\cucumber\bin\cucumber.js
C:\Users\28451\AppData\Roaming\npm
`-- cucumber@2.0.0-rc.9
  `-- glob@7.1.1
    `-- minimatch@3.0.4


D:\Workspaces\Projects\AngPro1>npm install --save-dev protractor-cucumber-framew
ork_
```

Install Protractor-cucumber in project location locally with the below command.

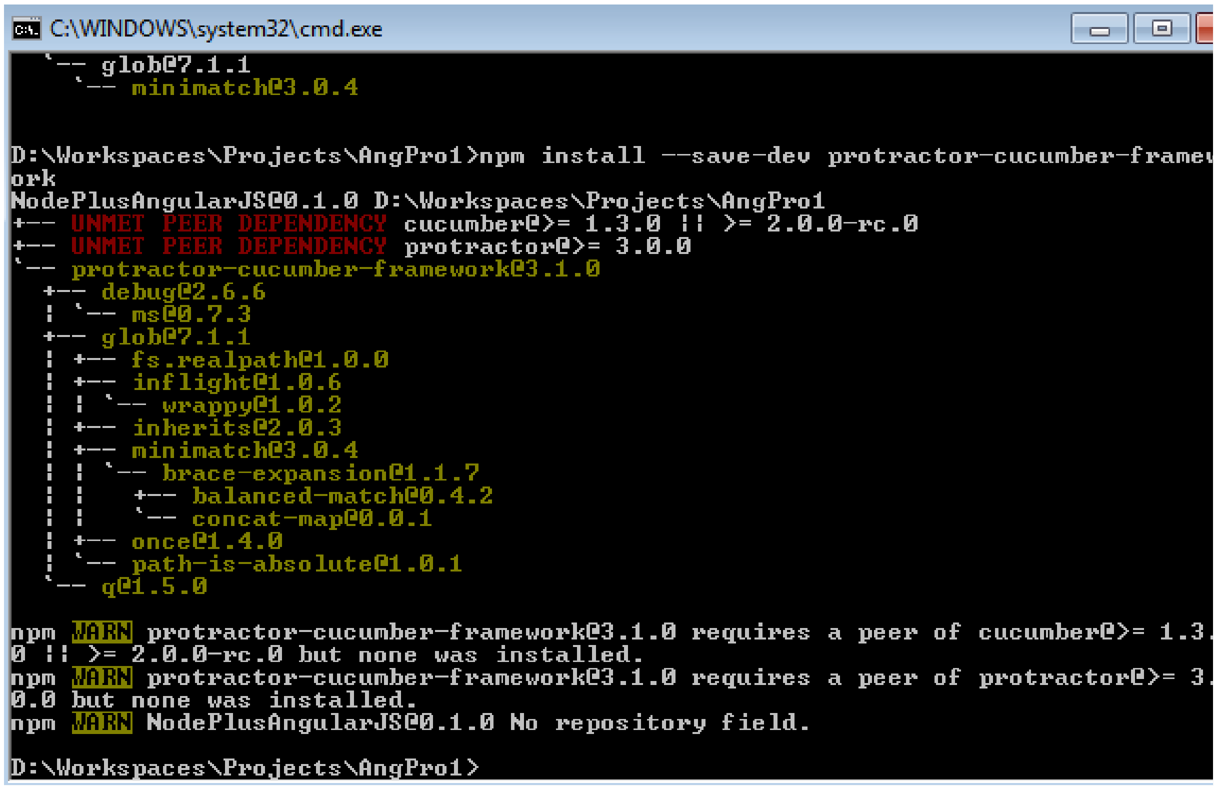
npm install protractor-cucumber --save-dev


Projects - Node - Demo11/features/step_definitions/my_step_definitions.js - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
AngPro1
Demo11
JavaScript Resources
features
step_definitions
demochk1.feature
public
bower.json
conf.js
cucumber.conf.js
demoT1.js
hello-from-nodejs-server.js
package.json
README.md
Console Debug Markdown View Problems Terminal
Demo11 (EBI227F0)
Demo11 (EBI227F0) (1)

```
D:\Workspaces\Projects\Demo11>npm install protractor-cucumber --save-dev

> cucumber-expressions@3.0.0 postinstall D:\Workspaces\Projects\Demo11\node_modules\cucumber-expressions
> node scripts/postinstall.js

NodePlusAngularJS@0.1.0 D:\Workspaces\Projects\Demo11
`-- protractor-cucumber@0.1.8
  +-- async@2.4.0
  | `-- lodash@4.17.4
  +-- cucumber@2.1.0
  | +-- assertion-error-formatter@2.0.0
  | | +-- diff@3.2.0
  | | +-- pad-right@0.2.2
  | | `-- repeat-string@1.6.1
  | +-- babel-runtime@6.23.0
  | | +-- core-js@2.4.1
  | | `-- regenerator-runtime@0.10.5
  | +-- bluebird@3.5.0
  | +-- cli-table@0.3.1
  | | `-- colors@1.0.3
  | +-- colors@1.1.2
  | +-- commander@2.9.0
  | | `-- graceful-readlink@1.0.1
  | +-- cucumber-expressions@3.0.0
  | +-- cucumber-tag-expressions@1.0.0
  | +-- duration@0.2.0
```

Install the Protractor-Cucumber framework locally with below command.

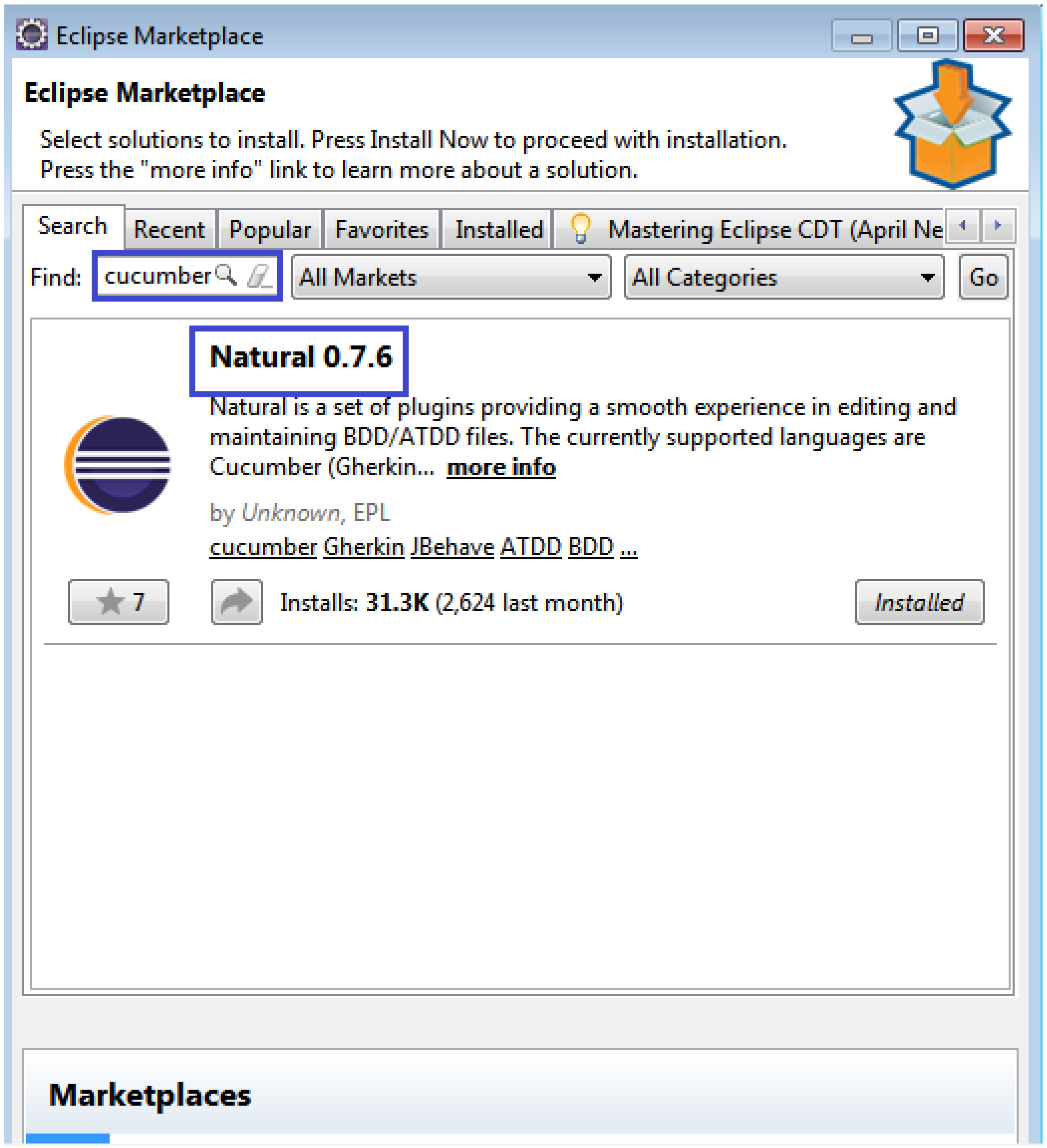
npm install --save-dev protractor-cucumber-framework



```
  `-- glob@7.1.1
    `-- minimatch@3.0.4


D:\Workspaces\Projects\AngPro1>npm install --save-dev protractor-cucumber-framew
ork
NodePlusAngularJS@0.1.0 D:\Workspaces\Projects\AngPro1
+-- UNMET PEER DEPENDENCY cucumber@>= 1.3.0 || >= 2.0.0-rc.0
+-- UNMET PEER DEPENDENCY protractor@>= 3.0.0
`-- protractor-cucumber-framework@3.1.0
  +-- debug@2.6.6
  | `-- ms@0.7.3
  +-- glob@7.1.1
  | +-- fs.realpath@1.0.0
  | +-- inflight@1.0.6
  | | `-- wrappy@1.0.2
  | +-- inherits@2.0.3
  | +-- minimatch@3.0.4
  | | `-- brace-expansion@1.1.7
  | |   +-- balanced-match@0.4.2
  | |   `-- concat-map@0.0.1
  | +-- once@1.4.0
  | `-- path-is-absolute@1.0.1
  `-- q@1.5.0

npm WARN protractor-cucumber-framework@3.1.0 requires a peer of cucumber@>= 1.3.
0 || >= 2.0.0-rc.0 but none was installed.
npm WARN protractor-cucumber-framework@3.1.0 requires a peer of protractor@>= 3.
0.0 but none was installed.
npm WARN NodePlusAngularJS@0.1.0 No repository field.

D:\Workspaces\Projects\AngPro1>
```

The Cucumber installation is completed and now we can start writing feature files. Before that we can add cucumber plug-in in eclipse.

Add the Cucumber plug-in eclipse market place



Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Natural 0.7.6

Natural is a set of plugins providing a smooth experience in editing and maintaining BDD/ATDD files. The currently supported languages are Cucumber (Gherkin... more info

by *Unknown*, EPL

cucumber Gherkin JBehave ATDD BDD ...

Installs: **31.3K** (2,624 last month)

Now we can start writing the cucumber feature file. In the project, create a "features" folder with a "test.feature" feature file. All of your features will be housed in this folder.

Below is an example of a cucumber feature file:

```
#features/test.feature
Feature: Running Cucumber with Protractor
  As a user of Protractor
  I should be able to use Cucumber
  In order to run my E2E tests

  Scenario: Protractor and Cucumber Test
    Given I go to "https://angularjs.org/"
    When I add "Be Awesome" in the task field
    And I click the add button
    Then I should see my new task in the list
```

Cucumber needs step definitions files which contains necessary steps to run the feature file. Save these step definition files in a “step_definitions” folder within the “features” folder. From there, you can write out your steps using Protractor functions and locators. The basic structure of a step definition is the keyword (i.e. *Given*, *When*, or *Then*) followed by a regular expression. The regular expression is used to identify the desired step implementation. The step text in scenarios is matched against the regular expressions of appropriate (i.e. *given*, *when*, or *then*) step implementations to find the one to use. When a match is found, the substrings matching any groups in the regular expression are used as arguments to the block which is then evaluated. The result of that evaluation is discarded. Note that these arguments are **always** strings. If you need them converted to other types, it’s up to you to do that conversion in the block.

Please find the below step definition file for reference.

```
module.exports = function() {
  this.Given(/^I go to "([^"]*)"$/, function(site) {
    browser.get(site);
  });

  this.When(/^I add "([^"]*)" in the task field$/, function(task) {
    element(by.model('todoList.todoText')).sendKeys(task);
  });

  this.When(/^I click the add button$/, function() {
    var el = element(by.css('[value="add"]'));
    el.click();
  });

  this.Then(/^I should see my new task in the list$/, function(callback) {
    var todoList = element.all(by.repeater('todo in todoList.todos'));
    expect(todoList.count()).to.eventually.equal(3);
    expect(todoList.get(2).getText()).to.eventually.equal('Do not Be Awesome')
      .and.notify(callback);
  });
};
```

Now, in order to run these, you will need to make a few adjustments to the *conf.js* file. Cucumber is no longer included by default for Protractor 3.x so you will pass in the *custom* option for your framework plus a few extras for the Cucumber framework itself.

Please find below updated configuration file as “cucumber.conf.js” to run the cucumber feature file.

```
//cucumber.conf.js
exports.config = {
  seleniumAddress: 'http://127.0.0.1:4444/wd/hub',
  getPageTimeout: 60000,
  allScriptsTimeout: 500000,
  framework: 'custom',
  // path relative to the current config file
  frameworkPath: require.resolve('protractor-cucumber-framework'),
  capabilities: {
    'browserName': 'chrome'
  },

  // Spec patterns are relative to this directory.
  specs: [
    'features/*.feature'
  ],

  baseURL: 'http://localhost:8080/',

  cucumberOpts: {
    require: 'features/step_definitions/stepDefinitions.js',
    tags: false,
    format: 'pretty',
    profile: false,
    'no-source': true
  }
};
```

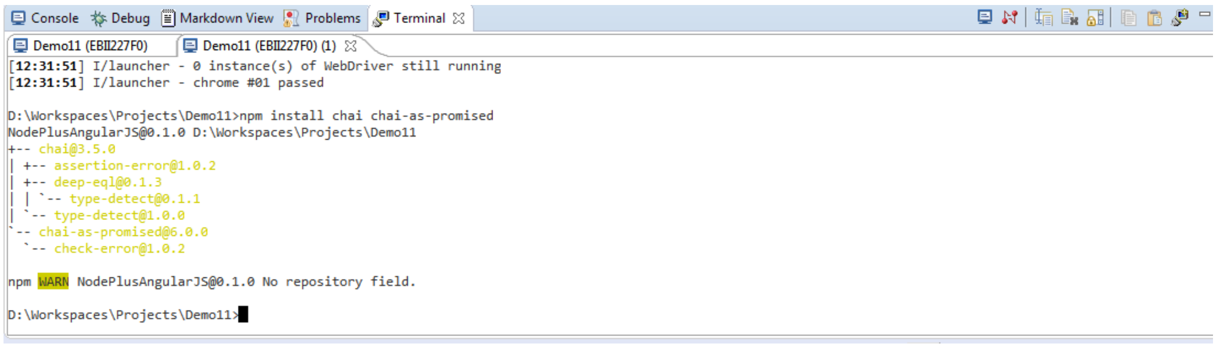
<u>Install Assertions: chai and Chais-As-Promised</u>

Since you're using the custom framework option with Protractor you'll need to add an assertion library like Chai — a popular choice. Chai allows us to write assertions in a simple, readable style — such as the familiar *expect* syntax in

expect(element.getText()).to.eventually.equal('Name');

To install, we should run below command

npm install chai chai-as-promised



```
[12:31:51] I/launcher - 0 instance(s) of WebDriver still running
[12:31:51] I/launcher - chrome #01 passed

D:\Workspaces\Projects\Demo11>npm install chai chai-as-promised
NodePlusAngularJS@0.1.0 D:\Workspaces\Projects\Demo11
+-- chai@3.5.0
| +-- assertion-error@1.0.2
| +-- deep-eql@0.1.3
| | `-- type-detect@0.1.1
| `-- type-detect@1.0.0
`-- chai-as-promised@6.0.0
  `-- check-error@1.0.2

npm WARN NodePlusAngularJS@0.1.0 No repository field.

D:\Workspaces\Projects\Demo11>
```

<u>Reports:</u>

<u>To setup GruntJS build here is the steps:</u>

1. Make sure you have setup your `package.json` or setup new one:
2. `npm init`
3. Install Grunt CLI as global:
4. `npm install -g grunt-cli`
5. Install Grunt in your local project:
6. `npm install grunt --save-dev`
7. Install any Grunt Module you may need in your build process. Just for sake of this sample I will add Concat module for combining files together:
8. `npm install grunt-contrib-concat --save-dev`

<u>Grunt</u>

You can use grunt-init for creating Gruntfile.js if you want wizard-based creation instead of raw coding for step 5.

To do so, please follow these steps:

```
npm install -g grunt-init
git clone https://github.com/gruntjs/grunt-init-gruntfile.git ~/.grunt-
init/gruntfile
grunt-init gruntfile
```

`If runner doesn’t work`

npm install grunt-protractor-runner --save-dev