**UNIT – 2**

**GETTING STARTED WITH NODE.JS:**

* [**Understanding Node.js**](#understanding_nodejs)
* [**Installing Node.js**](#installing_nodejs)
* [**Working with Node Packages**](#working_with_nodepackages)
* [**Creating a Node.js Application**](#nodejs_application)
* [**Writing Data to the Console**](#writing_data)

**UNDERSTANDING NODE.JS:**

* Node.js was developed in 2009 by Ryan Dahl as an answer to the frustration caused by concurrency issues, especially when dealing with web services.
* Google had just come out with the V8 JavaScript engine for the Chrome web browser, which was highly optimized for web traffic.
* Dahl created Node.js on top of V8 as a server-side environment that matched the client-side environment in the browser.
* The result is an extremely scalable server-side environment that allows developers to more easily bridge the gap between client and server.
* The fact that Node.js is written in JavaScript allows developers to easily navigate back and forth between client and server code and even reuse code between the two environments.
* Node.js has a great ecosystem with new extensions being written all the time.
* The Node.js environment is clean and easy to install, configure, and deploy. Literally in only an hour or two you can have a Node.js web server up and running.

**Who Uses Node.js?**

* Node.js quickly gained popularity among a wide variety of companies. These companies use Node.js first and foremost for scalability but also for ease of maintenance and faster development.
* The following are just a few of the companies using the Node.js technology:

■ Yahoo!

■ LinkedIn

■ eBay

■ New York Times

■ Dow Jones

■ Microsoft

**What Is Node.js Used For?**

* Node.js can be used for a wide variety of purposes. Because it is based on V8 and has highly

optimized code to handle HTTP traffic, the most common use is as a web server.

* However, Node.js can also be used for a variety of other web services such as:

■ Web services APIs such as REST

■ Real-time multiplayer games

■ Backend web services such as cross-domain, server-side requests

■ Web-based applications

■ Multiclient communication such as IM

**What Does Node.js Come With?**

Node.js comes with many built-in modules available right out of the box. This book covers

many but not all of these modules:

**Assertion testing:** Allows you to test functionality within your code.

**Buffer:** Enables interaction with TCP streams and file system operations.

**C/C++ add-ons:** Allows for C or C++ code to be used just like any other Node.js module.

**Child processes:** Allows you to create child processes.

**Cluster:** Enables the use of multicore systems.

**Command line options:** Gives you Node.js commands to use from a terminal.

**Console:** Gives the user a debugging console.

**Crypto:** Allows for the creation of custom encryption.

**Debugger:** Allows debugging of a Node.js file.

**DNS:** Allows connections to DNS servers.

**Errors:** Allows for the handling of errors.

**Events:** Enables the handling of asynchronous events.

**File system:** Allows for file I/O with both synchronous and asynchronous methods.

**Globals:** Makes frequently used modules available without having to include them first.

**HTTP:** Enables support for many HTTP features.

**HTTPS:** Enables HTTP over the TLS/SSL.

**Modules:** Provides the module loading system for Node.js.

**Net:** Allows the creation of servers and clients.

**OS:** Allows access to the operating system that Node.js is running on.

**Path:** Enables access to file and directory paths.

**Process:** Provides information and allows control over the current Node.js process.

**Query strings:** Allows for parsing and formatting URL queries.

**Readline:** Enables an interface to read from a data stream.

**REPL:** Allows developers to create a command shell.

**Stream:** Provides an API to build objects with the stream interface

**String decoder:** Provides an API to decode buffer objects into strings.

**Timers:** Allows for scheduling functions to be called in the future.

**TLS/SSL:** Implements TLS and SSL protocols.

**URL:** Enables URL resolution and parsing.

**Utilities:** Provides support for various apps and modules.

**V8:** Exposes APIs for the Node.js version of V8.

**VM:** Allows for a V8 virtual machine to run and compile code.

**ZLIB:** Enables compression using Gzip and Deflate/Inflate.

**INSTALLING NODE.JS**

To easily install Node.js, download an installer from the Node.js website at http://nodejs.org.

The Node.js installer installs the necessary files on your PC to get Node.js up and running. No

additional configuration is necessary to start creating Node.js applications.

**Looking at the Node.js Install Location**

If you look at the install location, you will see a couple of executable files and a node\_modules Folder. The node executable file starts the Node.js JavaScript VM. The following list describes the executables in the Node.js install location that you need to get started:

* **node:** This file starts a Node.js JavaScript VM. If you pass in a JavaScript file location, Node.js executes that script. If no target JavaScript file is specified, then a script prompt is shown that allows you to execute JavaScript code directly from the console.
* **npm:** This command is used to manage the Node.js packages discussed in the next section.
* **node\_modules:** This folder contains the installed Node.js packages. These packages act as

libraries that extend the capabilities of Node.js.

**Verify Node.js Executables**

Take a minute and verify that Node.js is installed and working before moving on. To do so, open a console prompt and execute the following command to bring up a Node.js VM:

node

Next, at the Node.js prompt execute the following to write "Hello World" to the screen.

>console.log("Hello World");

You should see "Hello World" output to the console screen. Now exit the console using

Ctrl+C in Windows or Cmd+C on a Mac.

Next, verify that the npm command is working by executing the following command in the

OS console prompt:

npm version

You should see output similar to the following:

{ npm: '3.10.5',

ares: '1.10.1-DEV',

http\_parser: '2.7.0',

icu: '57.1',

modules: '48',

node: '6.5.0',

openssl: '1.0.2h',

uv: '1.9.1',

v8: '5.1.281.81',

zlib: '1.2.8'}

**Selecting a Node.js IDE**

* If you are planning on using an Integrated Development Environment (IDE) for your Node.js projects, you should take a minute and configure that now as well.
* Most developers are particular about the IDE that they like to use, and there will likely be a way to configure at least for JavaScript if not Node.js directly.
* For example, Eclipse has some great Node.js plugins, and the WebStorm IDE by IntelliJ has some good features for Node.js built in.
* If you are unsure of where to start, we use Visual Studio Code for the built-in TypeScript functionality required later in this book. That said, you can use any editor you want to generate your Node.js web applications. In reality, all you need is a decent text editor.
* Almost all the code you will generate will be .js, .json, .html, and .css. So pick the editor in which you feel the most comfortable writing those types of files.

**WORKING WITH NODE PACKAGES**

One of the most powerful features of the Node.js framework is the ability to easily extend

it with additional Node Packaged Modules (NPMs) using the Node Package Manager (NPM).

That’s right, in the Node.js world, NPM stands for two things. This book refers to the Node

Packaged Modules as modules to make it easier to follow.

**What Are Node Packaged Modules?**

* A Node Packaged Module is a packaged library that can easily be shared, reused, and installed in different projects. Many different modules are available for a variety of purposes. For example, the Mongoose module provides an ODM (Operational Data Model) for MongoDB, Express extends Node’s HTTP capabilities, and so on.
* Node.js modules are created by various third-party organizations to provide the needed features that Node.js lacks out of the box. This community of contributors is active in adding and updating modules.
* Node Packaged Modules include a package.json file that defines the packages. The package.json file includes informational metadata, such as the name, version author, and contributors, as well as control metadata, such as dependencies and other requirements that the Node Package Manager uses when performing actions such as installation and publishing.

**Understanding the Node Package Registry**

The Node modules have a managed location called the Node Package Registry where packages are registered. This allows you to publish your own packages in a location where others can use them as well as download packages that others have created.

The Node Package Registry is located at https://npmjs.com. From this location you can view the newest and most popular modules as well as search for specific packages, as shown in Figure 3.1.

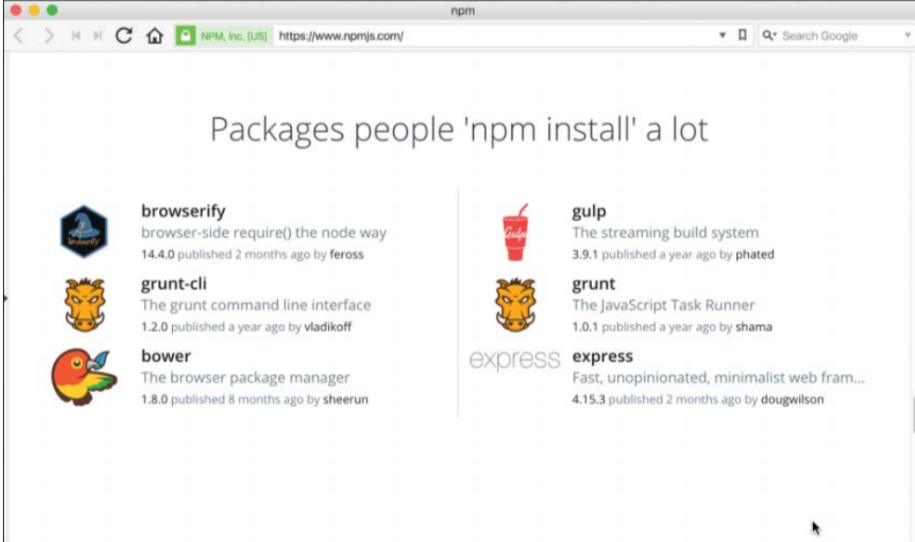


Figure 3.1 The official Node Package Modules website

**Using the Node Package Manager**

The Node Package Manager you have already seen is a command-line utility. It allows you

to find, install, remove, publish, and do everything else related to Node Package Modules.

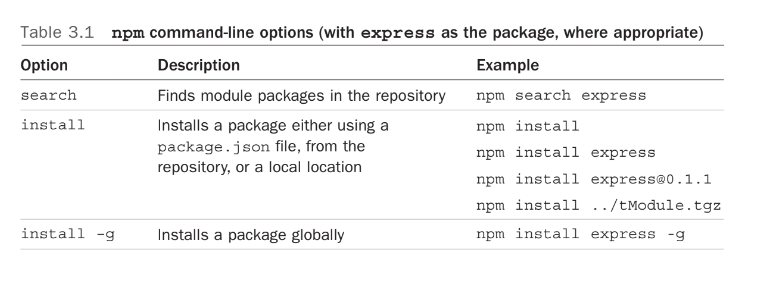
The Node Package Manager provides the link between the Node Package Registry and

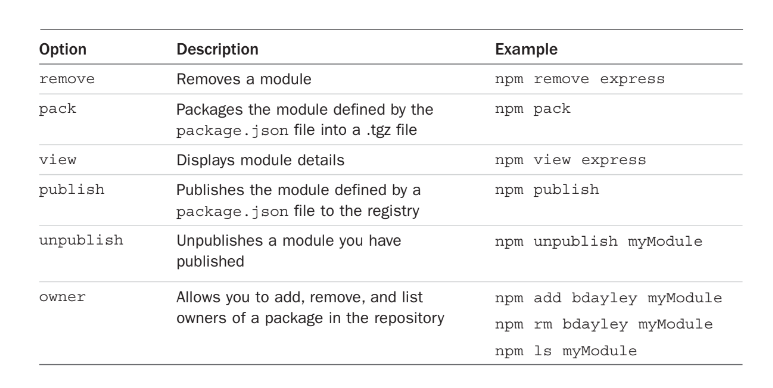
your development environment.

The simplest way to really explain the Node Package Manager is to list some of the command-

line options and what they do. You use many of these options in the rest of this chapter and

through out the book. Table 3.1 lists the Node Package Manager commands.

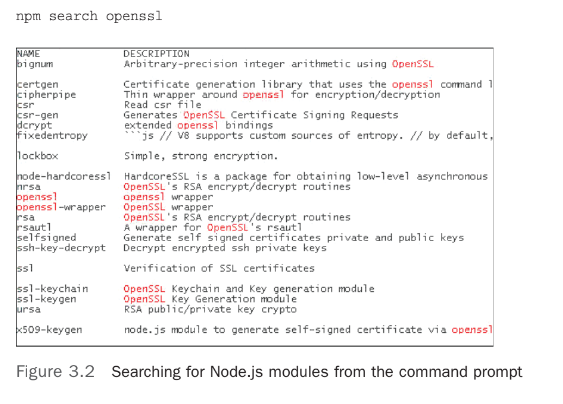




**Searching for Node Package Modules**

* You can also search for modules in the Node Package Registry directly from the command prompt using the npm search <search\_string> command.

For example, the following command searches for modules related to openssl and displays the results.



**Installing Node Packaged Modules**

* To use a Node module in your applications, it must first be installed where Node can find it.
* To install a Node module, use the npm install <module\_name> command.
* This downloads the Node module to your development environment and places it into the node\_modules folder where the install command is run.
* For example, the following command installs the express module:

**‘**npm install express**’**

**Using package.json**

* All Node modules must include a package.json file in their root directory.
* The package.json file is a simple JSON text file that defines the module including dependencies.
* The package.json file can contain a number of different directives to tell the Node Package Manager how to handle the module.
* The following is an example of a package.json file with a name, version, description, and

dependencies:

{

"name": "my\_module",

"version": "0.1.0",

"description": "a simple node.js module",

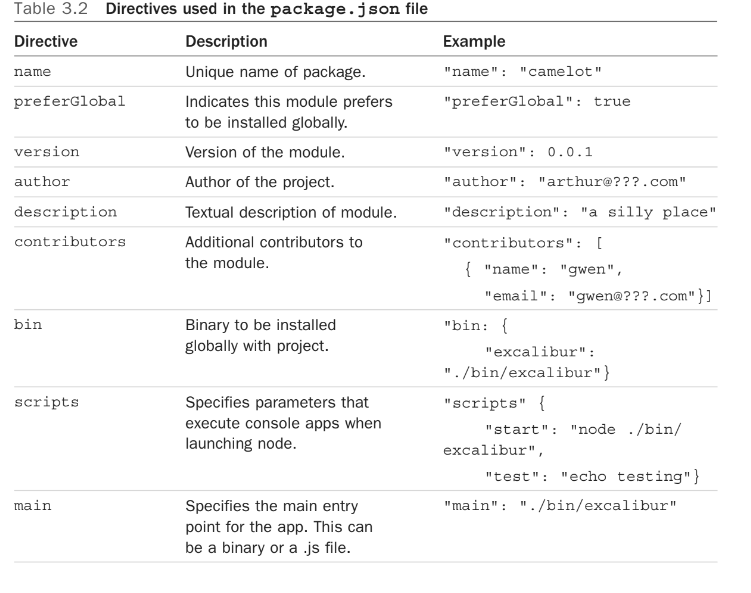
"dependencies" : {

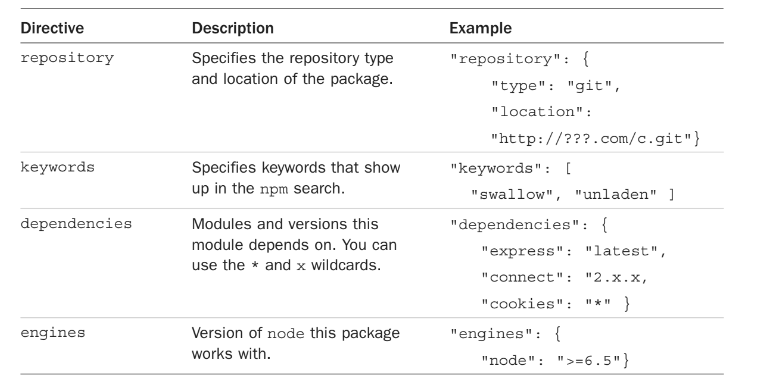
"express" : "latest"

}

}

* The only required directives in the package.json file are name and version. The rest depend on what you want to include.





**Creating a Node.js Application**

* Now you have enough information to jump into a Node.js project and get your feet wet.
* In this section, you create your own Node Packaged Module and then use that module as a library in a Node.js application.

**CREATING A NODE.JS PACKAGED MODULE**

* To create a Node.js Packaged Module you need to create the functionality in JavaScript, define the package using a package.json file, and then either publish it to the registry or package it for local use.
* Node.js modules are a type of [package](https://docs.npmjs.com/about-packages-and-modules) that can be published to npm.
* There are three steps you need to follow:

1. [Create a ‘package.json’ file](https://docs.npmjs.com/creating-node-js-modules#create-a-package-json-file)
2. [Create the file that will be loaded when your module is required by another application](https://docs.npmjs.com/creating-node-js-modules#create-the-file-that-will-be-loaded-when-your-module-is-required-by-another-application)
3. [Test your module](https://docs.npmjs.com/creating-node-js-modules#test-your-module)

* [**Create a ‘package.json’ file**](https://docs.npmjs.com/creating-node-js-modules#create-a-package-json-file)**:**
* To create a package.json file, on the command line, in the root directory of your Node.js module, run “npm init”.
  + For [scoped modules](https://docs.npmjs.com/about-scopes), run npm init --scope=@scope-name
  + For [unscoped modules](https://docs.npmjs.com/creating-and-publishing-unscoped-public-packages), run npm init
* To create a default package.json using information extracted from the current directory, use the npm init command with the --yes or –y flag.
* Provide responses for the required fields (name and version), as well as the main field:

name: The name of your module.

version: The initial module version.(recommended – 1.0.0)

* **Default values extracted from the current directory**

name: the current directory name

version: always 1.0.0

description: info from the README, or an empty string ""

scripts: by default creates an empty test script

keywords: empty

author: empty

license: [ISC](https://opensource.org/licenses/ISC)

bugs: information from the current directory, if present

homepage: information from the current directory, if present

* **Example:**

****

## Create the file that will be loaded when your module is required by another application:

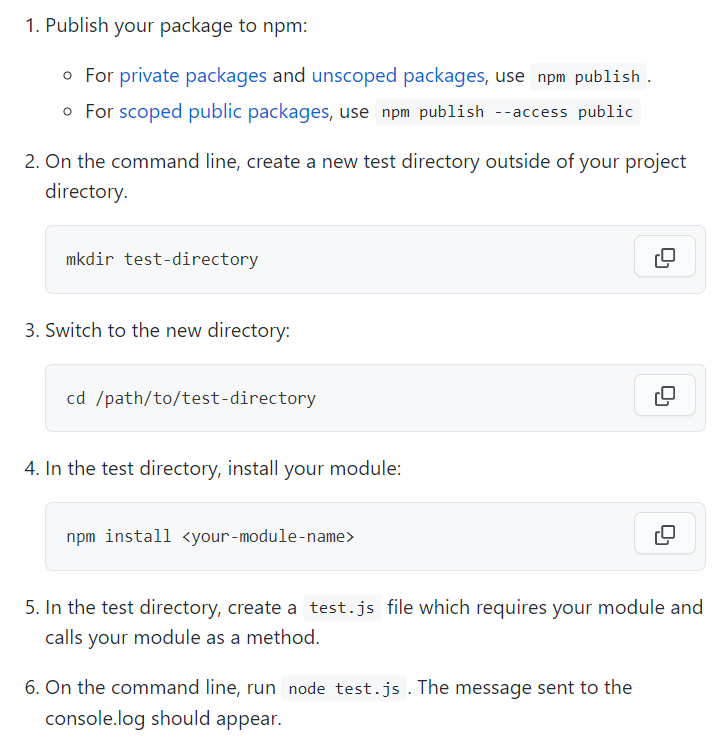
* In the file, add a function as a property of the exports object. This will make the function available to other code:

exports.printMsg = function() {

console.log("This is a message from the demo package");

}

## Test your module



**WRITING DATA TO THE CONSOLE**

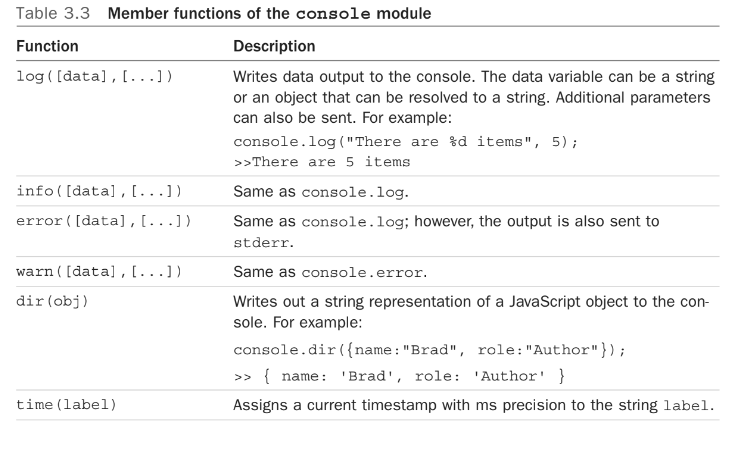
* One of the most useful modules in Node.js during the development process is the console

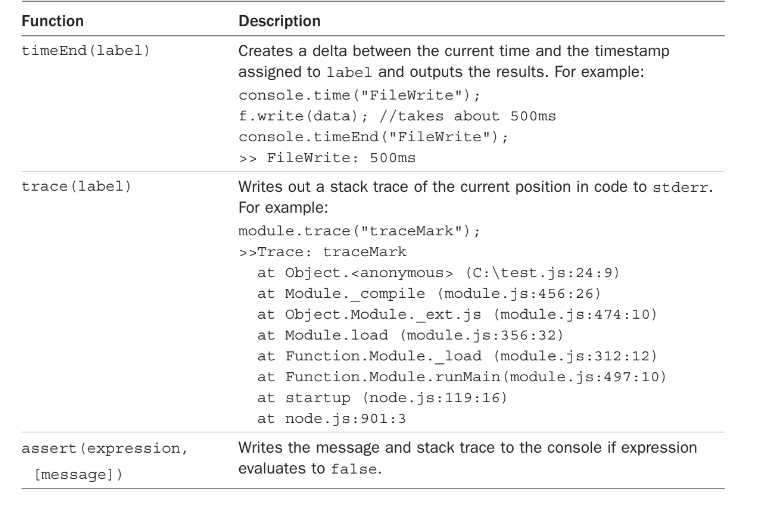
module.

* This module provides a lot of functionality when writing debug and information statements to the console.
* The console module allows you to control output to the console, implement time delta output, and write tracebacks and assertions to the console.
* Because the console module is so widely used, you do not need to load it into your modules

using a require() statement.

* You simply call the console function using console.<function>(<parameters>).

****

****