You have reached the beginning of time!

NEXT POST
N|Solid v4.8.4 is now available ⟩

The Basics of Package.json in Node.js and npm

FEATURED ARTICLES

N|Solid v4.8.4 is now available
In **Product** on Nov 04 2022

N|Solid v4.8.3 is now available
In **Product** on Oct 27 2022

npm Node.js

SHARE

CATEGORIES

Community

How To

Node.js

NodeSource

Product

The `package.json` file is core to the Node.js ecosystem and is a basic part of understanding and working with Node.js, npm, and even modern JavaScript. The `package.json` is used as what equates to a manifest about applications, modules, packages, and more - it's a tool to that's used to make modern development streamlined, modular, and efficient.

As a developer in the Node.js ecosystem, understanding the basics of `package.json` is one of the first steps to really kicking off your development experience with Node.js.

Because of how **essential** understanding the basics of `package.json` is to development with Node.js, I've gone through and outlined some of the most common and important properties of a `package.json` file that you'll need to use `package.json` effectively.

Identifying Metadata Inside `package.json`

The `name` property

The `name` property in a `package.json` file is one of the fundamental components of the `package.json` structure. At its core, `name` is a string that is *exactly* what you would expect - the name of the module that the `package.json` is describing.

Inside your `package.json`, the `name` property as a string would look something like this:

```
"name": "metaverse"
```

Despite having only a few material restrictions (a max length of 214 characters, can't begin with `.` or `<i>`, no uppercase letters, and no characters that aren't URL-friendly), one interesting aspect of the `name` property is that, there have been software ecosystems that have developed standard naming conventions

A few examples of this kind of namespacing are `babel-plugin-` for Babel and the the webpack `-loader` tooling.

The `version` property

The `version` property is a key part of a `package.json`, as it denotes the current version of the module that the `package.json` file is describing.

While the `version` property isn't _required to follow semver, semver is the standard used by the vast majority of modules and projects in the Node.js ecosystem - and the module version, according to semver, is what you'll typically find in the `version` property of a `package.json` file.

Inside your `package.json`, the `version` property as a string using semver could look like this:

```
"version": "5.12.4"
```

The `license` property

ways you can use the `license` property of a `package.json` file (to do things like dual-licensing or defining your own license), the most typical usage of it is to use a SPDX License identifier - some examples that you may recognize are `MIT`, `ISC`, and `GPL-3.0`.

Inside your `package.json`, the `license` property with an `MIT` license look like this:

```
"license": "MIT"
```

Looking for more info on npm? Check out our complete guide:

READ NOW: THE ULTIMATE GUIDE TO NPM ⟩

The `description` property

The `description` property of a `package.json` file is a string that contains a human-readable description about the module - basically, it's the module

`description` property is frequently indexed by search tools like npm search and the npm CLI search tool to help find relevant packages based on a search query.

Inside your `package.json`, the `description` property would look like this:

```
    "description": "The Metaverse virtual reality. The final
outcome of all virtual worlds, augmented reality, and the
Internet."
```

The `keywords` property

The `keywords` property inside a `package.json` file is, as you may have guessed, a collection of keywords about a module. Keywords can help identify a package, related modules and software, and concepts.

The `keywords` property is always going to be an array, with one or more strings as the array's values - each one of these strings will, in turn, be one of the project's keywords.

Inside your `package.json`, the `keywords` array would look something like this:

```
    "augmented reality",
    "snow crash"
]
```

Functional Metadata Inside `package.json`

The `main` property

The `main` property of a `package.json` is a direction to the entry point to the module that the `package.json` is describing. In a Node.js application, when the module is called via a require statement, the module's exports from the file named in the `main` property will be what's returned to the Node.js application.

Inside your `package.json`, the `main` property, with an entry point of `app.js`, would look like this:

```
"main": "app.js",
```

The `repository` property of a `package.json` is an array that defines *where* the source code for the module lives. Typically, for open source projects, this would be a public GitHub repo, with the `repository` array noting that the type of version control is `git`, and the URL of the repo itself. One thing to note about this is that it's not just a URL the repo can be accessed from, but the full URL that the *version control* can be accessed from.

Inside your package.json, the `repository` property would look like this:

```
"repository": {
    "type": "git",
    "url": "https://github.com/bnb/metaverse.git"
}
```

The `scripts` property

The `scripts` property of a `package.json` file is simple conceptually, but is complex functionally to the point that it's used as a build tool by many.

At its simplest, the `scripts` property takes an object with as many key/value pairs as desired. Each one of the keys in these key/value pairs is the name of a

Inside your package.json, the `scripts` property with a `build` command to execute `node app.js` (presumably to build your application) and a `test` command using Standard would look like this:

```
"scripts": {
    "build": "node app.js",
    "test": "standard"
}
```

The `dependencies` property

The `dependencies` property of a module's `package.json` is where dependencies - the *other* modules that *this* module uses - are defined. The `dependencies` property takes an object that has the name and version at which each dependency should be used. Tying things back to the `version` property defined earlier, the version that a module needs is defined. Do note that you'll frequently find carets (`^`) and tildes (`~`) included with package versions. These are the notation for version range - taking a deep-dive into these is outside the scope of this article, but you can learn more in our primer on semver.

Inside your package.json, the `dependencies` property of your module may look

```
"dependencies": {
  "async": "^0.2.10",
  "npm2es": "~0.4.2",
  "optimist": "~0.6.0",
  "request": "~2.30.0",
  "skateboard": "^1.5.1",
  "split": "^0.3.0",
  "weld": "^0.2.2"
},
```

The `devDependencies` property

The `devDependencies` property of a `package.json` is almost identical to the `dependencies` property in terms of structure, with a key difference. The `dependencies` property is used to define the dependencies that a module needs to run in *production*. The `devDependencies` property is *usually* used to define the dependencies the module needs to run in *development*.

Inside your package.json, the `devDependencies` property would look something like this:

Want to keep going?

If you want to keep learning about Node.js, npm, package.json, and development with the Node.js stack, I've got some **awesome** articles for you.

We *also* have a guide on some great utilities for Node.js developers - if you want to kick your developer experience to 11, be sure to check it out to find some tools to help you get there.

The goal with this guide was to help kickstart you with `package.json` for development with Node.js and npm. If you want to take the leap and ensure that you're *always* on solid footing with Node.js and npm modules, you should check out NodeSource Certified Modules - an awesome tool to help ensure that you spend more time building applications and less time worrying about modules.

NODESOURCE

The NodeSource platform offers a high-definition view of the performance, security and behavior of Node.js applications and functions.

START FOR FREE