Node.js Packages

The Node platform provides a way to structure an application under the form of a package.

We'll cover the following

- Anatomy of a Package
- The `package.json` File
- Semantic Versioning
- Dependencies

Anatomy of a Package

Technically, a package is a folder containing the following elements:

- A `package.json` file which describes the application and its dependencies.
- A entry point into the application, defaulting to the `index.js` file.
- A `node_modules/` subfolder, which is the default place where Node looks for modules to be loaded into the application.
- All the other files forming the source code of the application.

The `package.json` File

This JSON file describes the application and its dependencies: you can think of it as the app's ID document. It has a well-defined format consisting of many fields, most of them optional. The two mandatory fields are:

- `name` (all lowercase letters without dots, underscores and any non-URL safe character in it)
- `version` (following the semantic versioning format - more on that later)

Below is an example of a typical `package.json` file.

```json
{
  "name": "thejsway-node-example",
  "version": "1.0.0",
  "description": "Node example for the book \"The JavaScript Way\"",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "moment": "^2.18.1",
    "semver": "^5.3.0"
  },
  "keywords": [
    "javascript",
    "node",
    "thejsway"
  ],
  "author": "Baptiste Pesquet"
}
```

Semantic Versioning

Node packages are versioned using a format called *semantic versioning*. A version number is a three-digit string of the form `MAJOR.MINOR.PATCH` (example : `2.18.1`).

Here are the rules for defining a version number:

- The very first version should be `1.0.0`. Bug fixes and minor changes should increment the `PATCH` digit.
- New features added in a backwards-compatible way should increment the `MINOR` digit.
- Breaking changes should increment the `MAJOR` digit. These strict rules exist to facilitate the management of *dependencies* between packages.

Dependencies

In the `package.json` file definition, the `dependencies` field is used to declared the external packages needed by the current package. Each dependency is created with the package name followed by a version range. This *version range* specifies the package versions that are acceptable to use.

There are many ways to define a version range. The most commonly used ones are:

- Targeting a very specific version. Example: `2.18.1`.
- Using the `~` operator to allow patch-level changes. For example, the `~2.18.1` version range accepts version `2.18.7`, but not `2.19.0` nor `3.0.0`.
- Using the `^` operator to allow changes that do not modify the left-most non-zero digit in the version. Examples:
    - The `^2.18.1` version range accepts versions `2.18.7` and `2.19.0`, but not `3.0.0`.
    - The `^0.2.3` version range accepts version `0.2.5` but not `0.3.0` nor `1.0.0`.

Fine-tuning the targeted versions of external packages though version ranges helps limiting the risk of breaking the application apart when updating its dependencies.