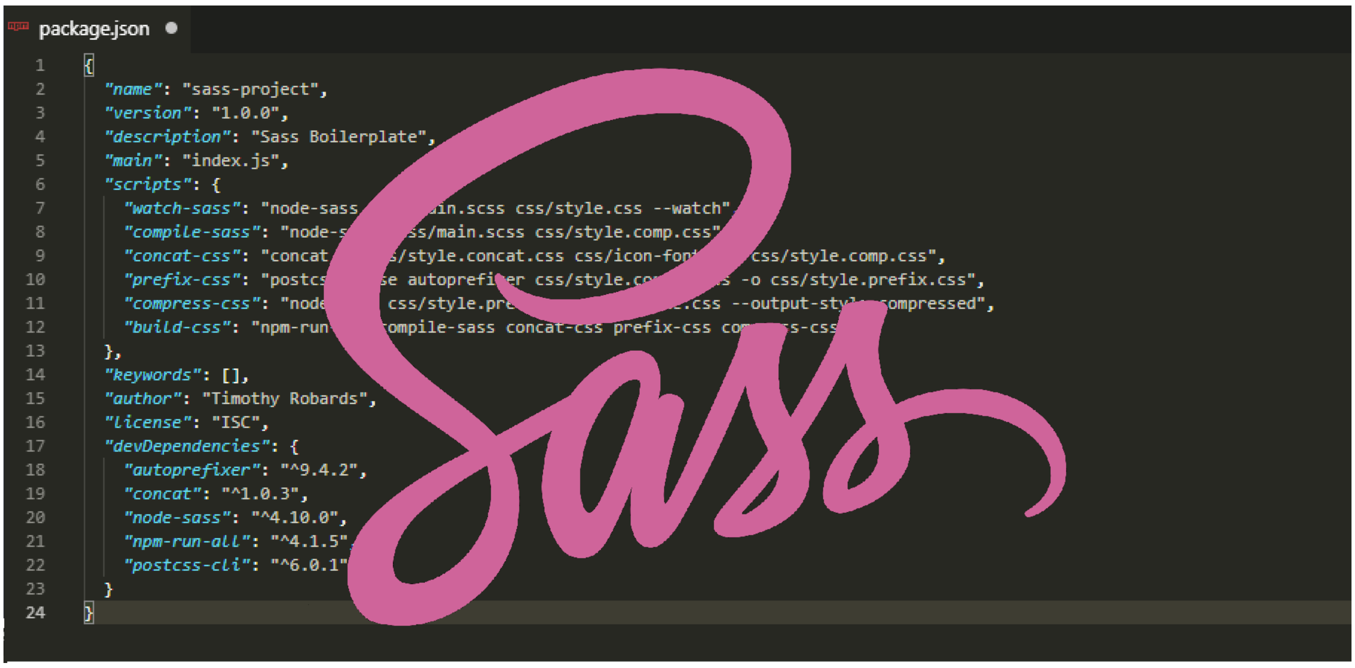
Creating a Build Process

In this section, we'll learn how to create a SASS build process, starting with a brief overview.



```
{
  "name": "sass-project",
  "version": "1.0.0",
  "description": "Sass Boilerplate",
  "main": "index.js",
  "scripts": {
    "watch-sass": "node-sass [illegible]in.scss css/style.css --watch",
    "compile-sass": "node-s[illegible]ss/main.scss css/style.comp.css",
    "concat-css": "concat [illegible]s/style.concat.css css/icon-fon[illegible] css/style.comp.css",
    "prefix-css": "postcs[illegible]se autoprefi[illegible]er css/style.co[illegible]s -o css/style.prefix.css",
    "compress-css": "node[illegible] css/style.pr[illegible].css --output-sty[illegible]compressed",
    "build-css": "npm-run[illegible]ompile-sass concat-css prefix-css co[illegible]s-css[illegible]
  },
  "keywords": [],
  "author": "Timothy Robards",
  "license": "ISC",
  "devDependencies": {
    "autoprefixer": "^9.4.2",
    "concat": "^1.0.3",
    "node-sass": "^4.10.0",
    "npm-run-all": "^4.1.5",
    "postcss-cli": "^6.0.1"
  }
}
```

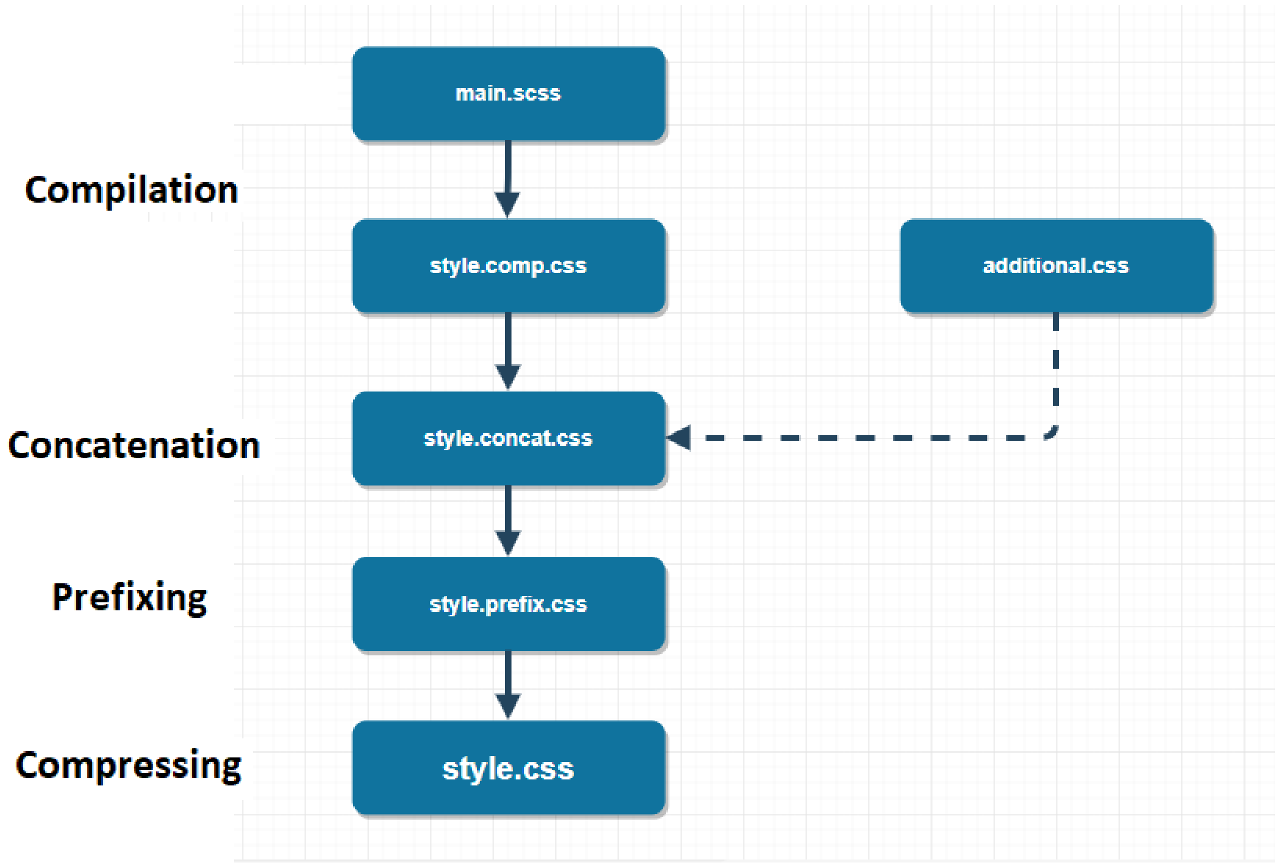
We will look at how we can use npm scripts to set up a Sass build process and boost our development workflow!

What is a build process?

A build process is essentially just a sequence of tasks that perform automatically — which we run after project completion. Our production files are generated and are then ready to be deployed to a web server.

The Build Process

Firstly, let's break down the steps involved. We will be **compiling**, **concatenating**, **prefixing** and **compressing** our stylesheets. As shown in the below diagram:



Over the next few lessons, we'll be looking at each step in detail.

The `main.scss` file is our main SASS file which performs the **compilation** to CSS.

Next we'll take a look at **concatenation**. In this process, we want to merge all of our CSS files into one file. To test out this process, We'll use an additional CSS file called `additional.css`.

Then we'll look at **prefixing** with an autoprefixer.

As a refresher, sometimes when working with CSS we'll use nonstandard properties. Browser vendors add prefixes for these experimental properties, so they can be used before they're standardized.

When we use an autoprefixer, any vendor prefixes (-webkit, -moz, etc) will be automatically added to our code, to help ensure its functionality across the major browsers.

And the final step in our process will be **compression**, where we compress all our code to maximize performance.

So let's go ahead and create our build process with npm scripts!