Load Options [PostCSS](https://github.com/postcss/postcss) Load Plugins

# **Load Config**

## **Install**

npm i -D postcss-load-config

## **Usage**

npm i -S|-D postcss-plugin

Install all required PostCSS plugins and save them to your **package.json** dependencies/devDependencies

Then create a PostCSS config file by choosing one of the following formats

### package.json

Create a **postcss** section in your project's **package.json**

Project (Root)

|– client

|– public

|

|- package.json

{

"postcss": {

"parser": "sugarss",

"map": false,

"plugins": {

"postcss-plugin": {}

}

}

}

### .postcssrc

Create a **.postcssrc** file in JSON or YAML format

ℹ️ It's recommended to use an extension (e.g **.postcssrc.json** or **.postcssrc.yml**) instead of .postcssrc

Project (Root)

|– client

|– public

|

|- (.postcssrc|.postcssrc.json|.postcssrc.yml)

|- package.json

**.postcssrc.json**

{

"parser": "sugarss",

"map": false,

"plugins": {

"postcss-plugin": {}

}

}

**.postcssrc.yml**

parser: sugarss

map: false

plugins:

postcss-plugin: {}

### .postcssrc.js or postcss.config.js

You may need some logic within your config. In this case create JS file named:

* .postcssrc.js
* .postcssrc.mjs
* .postcssrc.cjs
* .postcssrc.ts
* .postcssrc.cts
* postcss.config.js
* postcss.config.mjs
* postcss.config.cjs
* postcss.config.ts
* postcss.config.cts

Project (Root)

|– client

|– public

|- (.postcssrc|postcss.config).(js|mjs|cjs|ts|cts)

|- package.json

You can export the config as an {Object}

**.postcssrc.js**

module.exports = {

parser: 'sugarss',

map: false,

plugins: {

'postcss-plugin': {}

}

}

Or export a {Function} that returns the config (more about the ctx param below)

**.postcssrc.js**

module.exports = (ctx) => ({

parser: ctx.parser ? 'sugarss' : false,

map: ctx.env === 'development' ? ctx.map : false,

plugins: {

'postcss-plugin': ctx.options.plugin

}

})

Plugins can be loaded either using an {Object} or an {Array}

#### {Object}

**.postcssrc.js**

module.exports = ({ env }) => ({

...options,

plugins: {

'postcss-plugin': env === 'production' ? {} : false

}

})

ℹ️ When using an {Object}, the key can be a Node.js module name, a path to a JavaScript file that is relative to the directory of the PostCSS config file, or an absolute path to a JavaScript file.

#### {Array}

**.postcssrc.js**

module.exports = ({ env }) => ({

...options,

plugins: [

env === 'production' ? require('postcss-plugin')() : false

]

})

:warning: When using an {Array}, make sure to require() each plugin

## **Options**

| **Name** | **Type** | **Default** | **Description** |
| --- | --- | --- | --- |
| [**to**](#to) | {String} | undefined | Destination File Path |
| [**map**](#map) | {String|Object} | false | Enable/Disable Source Maps |
| [**from**](#from) | {String} | undefined | Source File Path |
| [**parser**](#parser) | {String|Function} | false | Custom PostCSS Parser |
| [**syntax**](#syntax) | {String|Function} | false | Custom PostCSS Syntax |
| [**stringifier**](#stringifier) | {String|Function} | false | Custom PostCSS Stringifier |

### parser

**.postcssrc.js**

module.exports = {

parser: 'sugarss'

}

### syntax

**.postcssrc.js**

module.exports = {

syntax: 'postcss-scss'

}

### stringifier

**.postcssrc.js**

module.exports = {

stringifier: 'midas'

}

### [**map**](https://github.com/postcss/postcss/blob/master/docs/source-maps.md)

**.postcssrc.js**

module.exports = {

map: 'inline'

}

:warning: In most cases options.from && options.to are set by the third-party which integrates this package (CLI, gulp, webpack). It's unlikely one needs to set/use options.from && options.to within a config file. Unless you're a third-party plugin author using this module and its Node API directly **dont't set options.from && options.to yourself**

### to

module.exports = {

to: 'path/to/dest.css'

}

### from

module.exports = {

from: 'path/to/src.css'

}

## **Plugins**

### {} || null

The plugin will be loaded with defaults

'postcss-plugin': {} || null

**.postcssrc.js**

module.exports = {

plugins: {

'postcss-plugin': {} || null

}

}

:warning: {} must be an **empty** {Object} literal

### {Object}

The plugin will be loaded with given options

'postcss-plugin': { option: '', option: '' }

**.postcssrc.js**

module.exports = {

plugins: {

'postcss-plugin': { option: '', option: '' }

}

}

### false

The plugin will not be loaded

'postcss-plugin': false

**.postcssrc.js**

module.exports = {

plugins: {

'postcss-plugin': false

}

}

### Ordering

Plugin **execution order** is determined by declaration in the plugins section (**top-down**)

{

plugins: {

'postcss-plugin': {}, // [0]

'postcss-plugin': {}, // [1]

'postcss-plugin': {} // [2]

}

}

## **Context**

When using a {Function} (postcss.config.js or .postcssrc.js), it's possible to pass context to postcss-load-config, which will be evaluated while loading your config. By default ctx.env (process.env.NODE\_ENV) and ctx.cwd (process.cwd()) are available on the ctx {Object}

ℹ️ Most third-party integrations add additional properties to the ctx (e.g postcss-loader). Check the specific module's README for more information about what is available on the respective ctx

## **Examples**

**postcss.config.js**

module.exports = (ctx) => ({

parser: ctx.parser ? 'sugarss' : false,

map: ctx.env === 'development' ? ctx.map : false,

plugins: {

'postcss-import': {},

'postcss-nested': {},

cssnano: ctx.env === 'production' ? {} : false

}

})



"scripts": {

"build": "NODE\_ENV=production node postcss",

"start": "NODE\_ENV=development node postcss"

}

const { readFileSync } = require('fs')

const postcss = require('postcss')

const postcssrc = require('postcss-load-config')

const css = readFileSync('index.sss', 'utf8')

const ctx = { parser: true, map: 'inline' }

postcssrc(ctx).then(({ plugins, options }) => {

postcss(plugins)

.process(css, options)

.then((result) => console.log(result.css))

})



"scripts": {

"build": "NODE\_ENV=production gulp",

"start": "NODE\_ENV=development gulp"

}

const { task, src, dest, series, watch } = require('gulp')

const postcss = require('gulp-postcssrc')

const css = () => {

src('src/\*.css')

.pipe(postcss())

.pipe(dest('dest'))

})

task('watch', () => {

watch(['src/\*.css', 'postcss.config.js'], css)

})

task('default', series(css, 'watch'))



"scripts": {

"build": "NODE\_ENV=production webpack",

"start": "NODE\_ENV=development webpack-dev-server"

}

**webpack.config.js**

module.exports = (env) => ({

module: {

rules: [

{

test: /\.css$/,

use: [

'style-loader',

'css-loader',

'postcss-loader'

]

}

]

}

})

## **Maintainers**

| [Michael Ciniawsky](https://github.com/michael-ciniawsky) | [Mateusz Derks](https://github.com/ertrzyiks) |
| --- | --- |

## **Contributors**

| [Ryan Dunckel](https://github.com/sparty02) | [Patrick Gilday](https://github.com/pcgilday) | [Dalton Santos](https://github.com/daltones) | [François Wouts](https://github.com/fwouts) |
| --- | --- | --- | --- |

## Security Contact

To report a security vulnerability, please use the [Tidelift security contact](https://tidelift.com/security). Tidelift will coordinate the fix and disclosure.