Configuration File

EDIT

You can configure Prettier via (in order of precedence):

- A `"prettier"` key in your `package.json`, or `package.yaml` file.
- A `.prettierrc` file written in JSON or YAML.
- A `.prettierrc.json`, `.prettierrc.yml`, `.prettierrc.yaml`, or `.prettierrc.json5` file.
- A `.prettierrc.js`, or `prettier.config.js` file that exports an object using `export default` or `module.exports` (depends on the `type` value in your `package.json`).
- A `.prettierrc.mjs`, or `prettier.config.mjs` file that exports an object using `export default`.
- A `.prettierrc.cjs`, or `prettier.config.cjs` file that exports an object using `module.exports`.
- A `.prettierrc.toml` file.

The configuration file will be resolved starting from the location of the file being formatted, and searching up the file tree until a config file is (or isn't) found.

Prettier intentionally doesn't support any kind of global configuration. This is to make sure that when a project is copied to another computer, Prettier's behavior stays the same. Otherwise, Prettier wouldn't be able to guarantee that everybody in a team gets the same consistent results.

The options you can use in the configuration file are the same as the API options.

Basic Configuration

JSON:

```
{
  "trailingComma": "es5",
  "tabWidth": 4,
  "semi": false,
```

```
// prettier.config.js, .prettierrc.js, prettier.config.mjs, or .prettierrc.mjs

/**
 * @see https://prettier.io/docs/en/configuration.html
 * @type {import("prettier").Config}
 */
const config = {
  trailingComma: "es5",
  tabWidth: 4,
  semi: false,
  singleQuote: true,
};

export default config;
```

JS (CommonJS):

```
// prettier.config.js, .prettierrc.js, prettier.config.cjs, or .prettierrc.cjs

/**
 * @see https://prettier.io/docs/en/configuration.html
 * @type {import("prettier").Config}
 */
const config = {
  trailingComma: "es5",
  tabWidth: 4,
  semi: false,
  singleQuote: true,
};

module.exports = config;
```

YAML:

```
# .prettierrc or .prettierrc.yaml
trailingComma: "es5"
tabWidth: 4
```

```toml
# .prettierrc.toml
trailingComma = "es5"
tabWidth = 4
semi = false
singleQuote = true
```

Configuration Overrides

Overrides let you have different configuration for certain file extensions, folders and specific files.

Prettier borrows ESLint's override format.

JSON:

```json
{
  "semi": false,
  "overrides": [
    {
      "files": "*.test.js",
      "options": {
        "semi": true
      }
    },
    {
      "files": ["*.html", "legacy/**/*.js"],
      "options": {
        "tabWidth": 4
      }
    }
  ]
}
```

YAML:

```yaml
semi: false
overrides:
  - files: "*.test.js"
```

```
  options:
    tabWidth: 4
```

`files` is required for each override, and may be a string or array of strings. `excludeFiles` may be optionally provided to exclude files for a given rule, and may also be a string or array of strings.

Sharing configurations

Sharing a Prettier configuration is simple: just publish a module that exports a configuration object, say `@company/prettier-config`, and reference it in your `package.json`:

```
{
  "name": "my-cool-library",
  "version": "9000.0.1",
  "prettier": "@company/prettier-config"
}
```

If you don't want to use `package.json`, you can use any of the supported extensions to export a string, e.g. `.prettierrc.json`:

```
"@company/prettier-config"
```

An example configuration repository is available here.

> Note: This method does **not** offer a way to *extend* the configuration to overwrite some properties from the shared configuration. If you need to do that, import the file in a `.prettierrc.js` file and export the modifications, e.g:
>
> ```
> import companyPrettierConfig from "@company/prettier-config";
>
> export default {
>   ...companyPrettierConfig,
>   semi: false,
> };
> ```

Combined with `overrides` you can teach Prettier how to parse files it does not recognize.

For example, to get Prettier to format its own `.prettierrc` file, you can do:

```
{
  "overrides": [
    {
      "files": ".prettierrc",
      "options": { "parser": "json" }
    }
  ]
}
```

You can also switch to the `flow` parser instead of the default `babel` for .js files:

```
{
  "overrides": [
    {
      "files": "*.js",
      "options": {
        "parser": "flow"
      }
    }
  ]
}
```

Note: *Never* put the `parser` option at the top level of your configuration. *Only* use it inside `overrides`. Otherwise you effectively disable Prettier's automatic file extension based parser inference. This forces Prettier to use the parser you specified for *all* types of files – even when it doesn't make sense, such as trying to parse a CSS file as JavaScript.

Configuration Schema

If you'd like a JSON schema to validate your configuration, one is available here: https://json.schemastore.org/prettierrc.

EditorConfig

```
# Stop the editor from looking for .editorconfig files in the parent directories
# root = true

[*]
# Non-configurable Prettier behaviors
charset = utf-8
insert_final_newline = true
# Caveat: Prettier won't trim trailing whitespace inside template strings, but your editor m
# trim_trailing_whitespace = true

# Configurable Prettier behaviors
# (change these if your Prettier config differs)
end_of_line = lf
indent_style = space
indent_size = 2
max_line_length = 80
```

Here's a copy+paste-ready `.editorconfig` file if you use the default options:

```
[*]
charset = utf-8
insert_final_newline = true
end_of_line = lf
indent_style = space
indent_size = 2
max_line_length = 80
```

← OPTIONS

EDITOR INTEGRATION →

Prettier stable