Version
v8.23.0

Search

USER GUIDE
Getting Started
Configuring
Configuration Files (New)
Configuration Files
Configuring Language Options
Configuring Rules
Configuring Plugins
Ignoring Code
Command Line Interface
Rules
Formatters
Integrations
Migrating to v8.x

DEVELOPER GUIDE
Architecture
Getting the Source Code
Set Up a Development Environment
Run the Tests
Working with Rules
Working with Plugins
Working with Custom Formatters
Working with Custom Parsers
Shareable Configs
Node.js API
Contributing

MAINTAINER GUIDE
Managing Issues
Reviewing Pull Requests
Managing Releases
Governance

# Plugins

## Specifying Parser

By default, ESLint uses Espree as its parser. You can optionally specify that a different parser should be used in your configuration file so long as the parser meets the following requirements:

1. It must be a Node module loadable from the config file where the parser is used. Usually, this means you should install the parser package separately using npm.
2. It must conform to the parser interface.

Note that even with these compatibilities, there are no guarantees that an external parser will work correctly with ESLint and ESLint will not fix bugs related to incompatibilities with other parsers.

To indicate the npm module to use as your parser, specify it using the `parser` option in your `.eslintrc` file. For example, the following specifies to use Esprima instead of Espree:

```
{
    "parser": "esprima",
    "rules": {
        "semi": "error"
    }
}
```

The following parsers are compatible with ESLint:

- Esprima
- `@babel/eslint-parser` - A wrapper around the Babel parser that makes it compatible with ESLint.
- `@typescript-eslint/parser` - A parser that converts TypeScript into an ESTree-compatible form so it can be used in ESLint.

Note when using a custom parser, the `parserOptions` configuration property is still required for ESLint to work properly with features not in ECMAScript 5 by default. Parsers are all passed `parserOptions` and may or may not use them to determine which features to enable.

## Specifying Processor

Plugins may provide processors. Processors can extract JavaScript code from other kinds of files, then let ESLint lint the JavaScript code or processors can convert JavaScript code in preprocessing for some purpose.

To specify processors in a configuration file, use the `processor` key with the concatenated string of a plugin name and a processor name by a slash. For example, the following enables the processor `a-processor` that the plugin `a-plugin` provided:

```
{
    "plugins": ["a-plugin"],
    "processor": "a-plugin/a-processor"
}
```

To specify processors for specific kinds of files, use the combination of the `overrides` key and the `processor` key. For example, the following uses the processor `a-plugin/markdown` for `*.md` files.

```
{
    "plugins": ["a-plugin"],
    "overrides": [
        {
            "files": ["*.md"],
            "processor": "a-plugin/markdown"
        }
    ]
}
```

Processors may make named code blocks such as `0.js` and `1.js`. ESLint handles such a named code block as a child file of the original file. You can specify additional configurations for named code blocks in the `overrides` section of the config. For example, the following disables the `strict` rule for the named code blocks which end with `.js` in markdown files.

```
{
    "plugins": ["a-plugin"],
    "overrides": [
        {
            "files": ["*.md"],
            "processor": "a-plugin/markdown"
        },
        {
            "files": ["**/*.md/*.js"],
            "rules": {
                "strict": "off"
            }
        }
    ]
}
```

ESLint checks the file path of named code blocks then ignores those if any `overrides` entry didn't match the file path. Be sure to add an `overrides` entry if you want to lint named code blocks other than `*.js`.

## Configuring Plugins

ESLint supports the use of third-party plugins. Before using the plugin, you have to install it using npm.

To configure plugins inside of a configuration file, use the `plugins` key, which contains a list of plugin names. The `eslint-plugin-` prefix can be omitted from the plugin name.

```
{
    "plugins": [
        "plugin1",
        "eslint-plugin-plugin2"
    ]
}
```

And in YAML:

```
---
  plugins:
    - plugin1
    - eslint-plugin-plugin2
```

**Notes:**

1. Plugins are resolved relative to the config file. In other words, ESLint will load the plugin as a user would obtain by running `require('eslint-plugin-pluginname')` in the config file.
2. Plugins in the base configuration (loaded by `extends` setting) are relative to the derived config file. For example, if `./.eslintrc` has `extends: ["foo"]` and the `eslint-config-foo` has `plugins: ["bar"]`, ESLint finds the `eslint-plugin-bar` from `./node_modules/` (rather than `./node_modules/eslint-config-foo/node_modules/`) or ancestor directories. Thus every plugin in the config file and base configurations is resolved uniquely.

### Naming convention

#### Include a plugin

The `eslint-plugin-` prefix can be omitted for non-scoped packages

```
{
    // ...
    "plugins": [
        "jquery", // means eslint-plugin-jquery
    ]
    // ...
}
```

The same rule does apply to scoped packages:

```
{
    // ...
    "plugins": [
        "@jquery/jquery", // means @jquery/eslint-plugin-jquery
        "@foobar" // means @foobar/eslint-plugin
    ]
    // ...
}
```

#### Use a plugin

When using rules, environments or configs defined by plugins, they must be referenced following the convention:

- `eslint-plugin-foo` → `foo/a-rule`
- `@foo/eslint-plugin` → `@foo/a-config`
- `@foo/eslint-plugin-bar` → `@foo/bar/a-environment`

For example:

```
{
    // ...
    "plugins": [
        "jquery",   // eslint-plugin-jquery
        "@foo/foo", // @foo/eslint-plugin-foo
        "@bar"      // @bar/eslint-plugin
    ],
    "extends": [
        "plugin:@foo/foo/recommended",
        "plugin:@bar/recommended"
    ],
    "rules": {
        "jquery/a-rule": "error",
        "@foo/foo/some-rule": "error",
        "@bar/another-rule": "error"
    },
    "env": {
        "jquery/jquery": true,
        "@foo/foo/env-foo": true,
        "@bar/env-bar": true,
    }
    // ...
}
```

Edit this page



Table of Contents