# TypeScript ESLint Parser

An ESLint custom parser which leverages [TypeScript ESTree](https://github.com/typescript-eslint/typescript-eslint/tree/master/packages/typescript-estree) to allow for ESLint to lint TypeScript source code.

    

## Getting Started \*\*[You can find our Getting Started docs here](../../docs/getting-started/linting/README.md)\*\* These docs walk you through setting up ESLint, this parser, and our plugin. If you know what you're doing and just want to quick start, read on... ## Quick-start ### Installation ```sh yarn add -D @typescript-eslint/parser ``` ### Usage In your ESLint configuration file, set the `parser` property: ```json { "parser": "@typescript-eslint/parser" } ``` There is sometimes an incorrect assumption that the parser itself is what does everything necessary to facilitate the use of ESLint with TypeScript. In actuality, it is the combination of the parser \_and\_ one or more plugins which allow you to maximize your usage of ESLint with TypeScript. For example, once this parser successfully produces an AST for the TypeScript source code, it might well contain some information which simply does not exist in a standard JavaScript context, such as the data for a TypeScript-specific construct, like an `interface`. The core rules built into ESLint, such as `indent` have no knowledge of such constructs, so it is impossible to expect them to work out of the box with them. Instead, you also need to make use of one more plugins which will add or extend rules with TypeScript-specific features. By far the most common case will be installing the [`@typescript-eslint/eslint-plugin`](https://github.com/typescript-eslint/typescript-eslint/tree/master/packages/eslint-plugin) plugin, but there are also other relevant options available such a [`@typescript-eslint/eslint-plugin-tslint`](https://github.com/typescript-eslint/typescript-eslint/tree/master/packages/eslint-plugin-tslint). ## Configuration The following additional configuration options are available by specifying them in [`parserOptions`](https://eslint.org/docs/user-guide/configuring#specifying-parser-options) in your ESLint configuration file. ```ts interface ParserOptions { ecmaFeatures?: { jsx?: boolean; }; project?: string | string[]; projectFolderIgnoreList?: (string | RegExp)[]; tsconfigRootDir?: string; extraFileExtensions?: string[]; warnOnUnsupportedTypeScriptVersion?: boolean; } ``` ### `parserOptions.ecmaFeatures.jsx` Default `false`. Enable parsing JSX when `true`. More details can be found [here](https://www.typescriptlang.org/docs/handbook/jsx.html). \*\*NOTE:\*\* this setting does not affect known file types (`.js`, `.jsx`, `.ts`, `.tsx`, `.json`) because the TypeScript compiler has its own internal handling for known file extensions. The exact behavior is as follows: - if `parserOptions.project` is \_not\_ provided: - `.js`, `.jsx`, `.tsx` files are parsed as if this is true. - `.ts` files are parsed as if this is false. - unknown extensions (`.md`, `.vue`) will respect this setting. - if `parserOptions.project` is provided (i.e. you are using rules with type information): - `.js`, `.jsx`, `.tsx` files are parsed as if this is true. - `.ts` files are parsed as if this is false. - "unknown" extensions (`.md`, `.vue`) \*\*are parsed as if this is false\*\*. ### `parserOptions.project` Default `undefined`. This option allows you to provide a path to your project's `tsconfig.json`. \*\*This setting is required if you want to use rules which require type information\*\*. Relative paths are interpreted relative to the current working directory if `tsconfigRootDir` is not set. If you intend on running ESLint from directories other than the project root, you should consider using `tsconfigRootDir`. - Accepted values: ```js // path project: './tsconfig.json'; // glob pattern project: './packages/\*\*/tsconfig.json'; // array of paths and/or glob patterns project: ['./packages/\*\*/tsconfig.json', './separate-package/tsconfig.json']; ``` - If you use project references, TypeScript will not automatically use project references to resolve files. This means that you will have to add each referenced tsconfig to the `project` field either separately, or via a glob. - TypeScript will ignore files with duplicate filenames in the same folder (for example, `src/file.ts` and `src/file.js`). TypeScript purposely ignore all but one of the files, only keeping the one file with the highest priority extension (the extension priority order (from highest to lowest) is `.ts`, `.tsx`, `.d.ts`, `.js`, `.jsx`). For more info see #955. - Note that if this setting is specified and `createDefaultProgram` is not, you must only lint files that are included in the projects as defined by the provided `tsconfig.json` files. If your existing configuration does not include all of the files you would like to lint, you can create a separate `tsconfig.eslint.json` as follows: ```jsonc { // extend your base config so you don't have to redefine your compilerOptions "extends": "./tsconfig.json", "include": [ "src/\*\*/\*.ts", "test/\*\*/\*.ts", "typings/\*\*/\*.ts", // etc // if you have a mixed JS/TS codebase, don't forget to include your JS files "src/\*\*/\*.js" ] } ``` ### `parserOptions.tsconfigRootDir` Default `undefined`. This option allows you to provide the root directory for relative tsconfig paths specified in the `project` option above. ### `parserOptions.projectFolderIgnoreList` Default `["/node\_modules/"]`. This option allows you to ignore folders from being included in your provided list of `project`s. Any resolved project path that matches one or more of the provided regular expressions will be removed from the list. This is useful if you have configured glob patterns, but want to make sure you ignore certain folders. For example, by default it will ensure that a glob like `./\*\*/tsconfig.json` will not match any `tsconfig`s within your `node\_modules` folder (some npm packages do not exclude their source files from their published packages). ### `parserOptions.extraFileExtensions` Default `undefined`. This option allows you to provide one or more additional file extensions which should be considered in the TypeScript Program compilation. The default extensions are `.ts`, `.tsx`, `.js`, and `.jsx`. Add extensions starting with `.`, followed by the file extension. E.g. for a `.vue` file use `"extraFileExtensions: [".vue"]`. ### `parserOptions.warnOnUnsupportedTypeScriptVersion` Default `true`. This option allows you to toggle the warning that the parser will give you if you use a version of TypeScript which is not explicitly supported ### `parserOptions.createDefaultProgram` Default `false`. This option allows you to request that when the `project` setting is specified, files will be allowed when not included in the projects defined by the provided `tsconfig.json` files. \*\*Using this option will incur significant performance costs. This option is primarily included for backwards-compatibility.\*\* See the \*\*`project`\*\* section above for more information. ## Supported TypeScript Version Please see [`typescript-eslint`](https://github.com/typescript-eslint/typescript-eslint) for the supported TypeScript version. \*\*Please ensure that you are using a supported version before submitting any issues/bug reports.\*\* ## Reporting Issues Please use the `@typescript-eslint/parser` issue template when creating your issue and fill out the information requested as best you can. This will really help us when looking into your issue. ## License TypeScript ESLint Parser is licensed under a permissive BSD 2-clause license. ## Contributing [See the contributing guide here](../../CONTRIBUTING.md)