
ARCHIVED

typescript.nvim is now archived and will no longer receive updates. Please see this issue for details.

typescript.nvim

A minimal `typescript-language-server` integration plugin to set up the language server via `nvim-lspconfig` and add commands for convenience. Written in TypeScript and transpiled to Lua using TypeScriptToLua.

This plugin is in **beta status**. It's stable enough for daily use, but breaking changes are possible.

Requires Neovim 0.7.

Setup

Install the plugin with your favorite plugin manager, then add `require("typescript").setup()` to your config. This will set up the plugin and `typescript-language-server` with default settings.

The following example shows all available options and their defaults:

```
1 require("typescript").setup({
2   disable_commands = false, -- prevent the plugin from creating Vim
    commands
3   debug = false, -- enable debug logging for commands
4   go_to_source_definition = {
5     fallback = true, -- fall back to standard LSP definition on
    failure
6   },
7   server = { -- pass options to lspconfig's setup method
8     on_attach = ...,
9   },
10 })
```

Note that command-specific configuration affects Vim commands, not the Lua API.

Important: if you have `require("lspconfig").tsserver.setup({})` anywhere in your config, make sure to remove it and pass any options you were using under the `server` key. `lspconfig` doesn't allow more than one setup call, so your config will not work as expected.

Features

Commands

The plugin exposes Vim commands as well as a Lua API. Vim commands are buffer-local, so you'll have access to them once `tsserver` has attached.

The following commands are async by default, but you can make them run synchronously by adding a `!` to Vim commands or passing `{ sync = true }` to Lua commands.

- Add missing imports: `:TypescriptAddMissingImports / require("typescript").actions.addMissingImports()`
- Organize imports: `:TypescriptOrganizeImports / require("typescript").actions.organizeImports()`
- Remove unused variables: `:TypescriptRemoveUnused / require("typescript").actions.removeUnused()`
- Fix all: `:TypescriptFixAll / require("typescript").actions.fixAll()`

Despite the name, this command fixes a handful of specific issues, most notably non-async functions that use `await` and unreachable code.

- Rename file: `:TypescriptRenameFile / require("typescript").renameFile(source, target)`

This command is always asynchronous. The Vim command will always operate on the current buffer and prompt for a rename target, while the Lua version requires specifying the full path to a `source` and `target`.

- Go to source definition: `:TypescriptGoToSourceDefinition / require("typescript").goToSourceDefinition(winnr, opts)`

TypeScript 4.7 contains support for a new experimental editor command called Go To Source Definition. It's similar to Go To Definition, but it never returns results inside declaration files. Instead, it tries to find corresponding implementation files (like `.js` or `.ts` files), and find definitions there — even if those files are normally shadowed by `.d.ts` files.

The command requires a position and derives it from the current window when using the Vim command or from the given `winnr` when using the Lua API.

`opts` is a table containing the following options:

- `fallback`: determines whether to fall back to a standard LSP definition request when the server fails to find a source definition.

Handlers

The plugin defines handlers for off-spec methods that are not otherwise supported by Neovim.

- `_typescript.rename`: invoked after certain code actions (e.g. when extracting a function to local / global scope).

Integrations

null-ls

Instead of using the above Vim commands, you can instead add commands to your code action menu using null-ls.

Commands Available as Code Actions

- Add missing imports
- Fix all
- Organize imports
- Remove unused

Code Action Setup

```
1 require("null_ls").setup({
2   sources = {
3     ..., -- add to your other sources
4     require("typescript.extensions.null-ls.code-actions"),
5   },
6 })
```

Will not support

- Anything not supported by `typescript-language-server` itself

FAQ

- How do I enable inlay hints?
 - Ensure you have installed Neovim `>= v0.10.0`. Inlay hints will not work in previous versions.

-
- Enable the `inlay_hint` option in your `on_attach` function **AND** set your desired settings in the server setup, e.g.:

```
1 require("typescript").setup({
2   server = {
3     on_attach = function(client, bufnr)
4       -- your other on_attach stuff here if you have any
5       -- ...
6       vim.lsp.buf.inlay_hint(bufnr, true)
7     end,
8     settings={
9       -- specify some or all of the following settings if you want
10      to adjust the default behavior
11      javascript = {
12        inlayHints = {
13          includeInlayEnumMemberValueHints = true,
14          includeInlayFunctionLikeReturnTypeHints = true,
15          includeInlayFunctionParameterTypeHints = true,
16          includeInlayParameterNameHints = "all", -- 'none' | '
17            literals' | 'all';
18          includeInlayParameterNameHintsWhenArgumentMatchesName =
19            true,
20          includeInlayPropertyDeclarationTypeHints = true,
21          includeInlayVariableTypeHints = true,
22        },
23      },
24      typescript = {
25        inlayHints = {
26          includeInlayEnumMemberValueHints = true,
27          includeInlayFunctionLikeReturnTypeHints = true,
28          includeInlayFunctionParameterTypeHints = true,
29          includeInlayParameterNameHints = "all", -- 'none' | '
30            literals' | 'all';
31          includeInlayParameterNameHintsWhenArgumentMatchesName =
32            true,
33          includeInlayPropertyDeclarationTypeHints = true,
34          includeInlayVariableTypeHints = true,
35        },
36      },
37    },
38  },
39})
```

Contributing

TypeScript Code

1. Clone the repo and run `npm install`.

-
2. Change or add TypeScript source files under the `src/` directory.
 3. Try out your changes locally using `npm run dev`.
 4. Build your changes before committing using `npm run build`.

Lua Code

Lua-only extensions live in the `extensions/` directory. Running `npm run build` copies extensions to the appropriate plugin directory and makes them available to Neovim under the `typescript.extensions` namespace (or a subdirectory if you choose to use one).

If you update or add a Lua file, make sure to run `npm run build` before committing! Your changes will not take effect otherwise.

Tests

Integration tests are in Lua and depend on `plenary.nvim`. Run `make test` from the root of the repo.