



Writing C# in Neovim

In this post, we'll be looking at how to set up our development environment to effectively write C# in Neovim. We'll also be looking a little more in-depth specifically at the tooling in Neovim that makes this all possible.

Aaron Bos | Thursday, April 28, 2022

My Configuration

Historically C# and .NET were meant to run *mostly* on Windows (Mono has been around for some time, which allowed cross-platform development) and the ubiquitous development environment was Visual Studio. While that may still be the case for the majority of developers, we now have the option to develop and run .NET code with any text editor and on any platform.

My personal machine is a 2016 MacBook Pro and I do my best to keep the amount of software that I need to download to a minimum. For this reason, I wanted to have an editor that was fast and could be adapted to be effective for any language. Enter Neovim, my editor of choice. If you're not familiar with Neovim, I recommend checking out the project's [vision](#). It's essentially building on the parts of Vim that everyone loves, while also making it more extensible for tools and features to be built on top of it. Neovim is not an IDE, but in my opinion, there are a lot of great, community-driven plugins that make it an enjoyable experience to write code with.

I will admit, as with learning something completely new, there is a steep learning curve to remembering the numerous key mappings to navigate Vim effectively. Once the fundamentals have been learned, navigating and editing code with Vim feels so smooth with everything focused on the keyboard and little need for a mouse. Next, we'll be taking a look at all the steps to set our environment up from scratch.

Setting up OmniSharp

Before we think about configuring anything in Neovim we need to download the OmniSharp language server. This particular language server is based on Roslyn (open-source C# and Visual Basic compiler) and is responsible for providing the editor with information to drive the code editing experience.

First, head over to the releases [page](#) on GitHub and download the latest stable version for your platform. I downloaded `'omnisharp-osx-x64-net6.0.zip'` for v1.38.2, but your version may differ based on when you read this and your platform.

Next, we need to extract the binaries from the zip file and move them to a permanent location. I decided to create a directory in `'/usr/local/bin'` called `'omnisharp-roslyn'`. I then moved all the files from the extracted download to this location.

At this point, I think it's best to address a few issues that will pop up eventually. (1) We need to make sure that all the files can be executed by the current user. To do this I ran `'chmod 744 /usr/local/bin/omnisharp-roslyn/*'`. This allows the current user to read, write and execute files in that directory while limiting other users with read-only access. (2) We need to remove the quarantine label from the files since they were downloaded from the internet and Apple will not allow them to be run by default. I did this by running `'find /usr/local/bin/omnisharp-roslyn/* | xargs xattr -r -d com.apple.quarantine'`.

The final thing to note is the name of the actual OmniSharp executable for us to run. For MacOS and Linux the file will be named just `'OmniSharp'`, but on Windows, the file will be named `'OmniSharp.exe'`.

Configuring Neovim

With OmniSharp in place, we are ready to set up the Neovim configuration to use the language server. The following code should be added to a lua section in `'init.vim'` or directly inside `'init.lua'` depending on which is being used.

```
local pid = vim.fn.getpid()

local omnisharp_bin = "/usr/local/bin/omnisharp-roslyn/OmniSharp"
```

```
require'lsppconfig'.omnisharp.setup{
    cmd = { omnisharp_bin, "--languageserver" , "--hostPID", tostring(pid) }
    -- Additional configuration can be added here
}
```

This provides the basic setup for the built-in language server client to work properly with OmniSharp. By default, the OmniSharp language server is only run for `.cs` and `.vb` files. My configuration includes an `on_attach` property which is a function that handles setting key bindings for common actions in code.

```
-- simply add on_attach below cmd declaration in previous snippet
require'lsppconfig'.omnisharp.setup{
    cmd = { omnisharp_bin, "--languageserver" , "--hostPID", tostring(pid) },
    on_attach = on_attach
}

local on_attach = function(client, bufnr)
    -- Enable completion triggered by <c-x><c-o>
    vim.api.nvim_buf_set_option(bufnr, 'omnifunc', 'v:lua.vim.lsp.omnifunc')

    -- Mappings.
    -- See `:help vim.lsp.*` for documentation on any of the below functions
    vim.api.nvim_buf_set_keymap(bufnr, 'n', 'gD', '<cmd>lua vim.lsp.buf.declaration', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', 'gd', '<cmd>lua vim.lsp.buf.definition', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', 'K', '<cmd>lua vim.lsp.buf.hover()', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', 'gi', '<cmd>lua vim.lsp.buf.implementation', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<C-k>', '<cmd>lua vim.lsp.buf.signature_help', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>wa', '<cmd>lua vim.lsp.buf.add_workspace_folder', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>wr', '<cmd>lua vim.lsp.buf.remove_workspace_folder', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>wl', '<cmd>lua print(vim.inspect(vim.lsp.buf.get_workspace_folders()))', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>D', '<cmd>lua vim.lsp.buf.type_definition', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>rn', '<cmd>lua vim.lsp.buf.rename', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>ca', '<cmd>lua vim.lsp.buf.code_action', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', 'gr', '<cmd>lua vim.lsp.buf.references', { noremap=true })
    vim.api.nvim_buf_set_keymap(bufnr, 'n', '<space>f', '<cmd>lua vim.lsp.buf.format', { noremap=true })
end
```

I also use a [plugin](#) called `'cmp'` that manages code completion via the Neovim `'lsp'` and language server. The `'cmp'` configuration is a bit complex and lengthy so I'll just link to it [here](#). The main bit to care about from the configuration is this.

```
-- Setup lspconfig.
local capabilities = require('cmp_nvim_lsp').update_capabilities(vim.lsp.protocol.get_default_capabilities)
-- Replace <YOUR_LSP_SERVER> with each lsp server you've enabled.
require('lspconfig')['omnisharp'].setup {
  capabilities = capabilities
}
```

I use the following plugins (via VimPlug) to make all of this come together nicely for a clean development experience.

```
Plug 'neovim/nvim-lspconfig'

Plug 'hrsh7th/cmp-nvim-lsp'
Plug 'hrsh7th/cmp-buffer'
Plug 'hrsh7th/cmp-path'
Plug 'hrsh7th/cmp-cmdline'
Plug 'hrsh7th/nvim-cmp'





" For vsnip users.
Plug 'hrsh7th/cmp-vsnip'
Plug 'hrsh7th/vim-vsnip'
```

If you've got to this point, it's probably clear that most things in Neovim do not just work out of the box. Currently, our environment should be configured and if we load a C# file, we should be able to use all of the rich functionality that we added with OmniSharp and related plugins!

The Results

Here is a quick gif to show the code completion and language server working. This small snippet only shows a small bit of functionality, but I assure you that the experience feels very similar to what you might expect from writing C# in VS Code!

A screenshot of the nvim editor interface. The top bar shows 'nvim'. The left pane displays a file explorer view of the 'EFRelatedData' directory, listing files like 'bin/', 'Context/', 'Models/', 'obj/', 'EFRelatedData.csproj', and 'Program.cs'. The right pane shows the content of 'Program.cs', which is currently empty except for a comment on line 1: '// See https://aka.ms/new-console-template for more information'. The bottom status bar indicates the current file is 'Program.cs[+]', the encoding is 'utf-8', the line ending is 'dos', the file size is '100%', and the cursor is at line '2:1'.

 25
  Share
  Subscribe
  Copy link

#learning #dotnet #csharp #neovim

As always thank you for taking the time to read this blog post!

[See More Posts](#)

