Doom Emacs and Language Servers

Rohit Goswami · 2021-07-20 · workflow · projects · tools · emacs

> *`doom-emacs` as an `ssh` IDE with `TRAMP` using `eglot` and language servers.*

Background

For most of my `emacs` configuration[^fn:1], there normally isn't very much to write about which isn't immediately evident from my configuration site. However, since my shift to a MacBook, I have needed to fine tune my existing `lsp-mode` default setup for `TRAMP` and this post will cover a little bit of that. Though most of the post is about `doom-emacs`, it is also applicable to vanilla emacs after porting the snippets over to `use-package` instead.

There are two primary methods of interfacing `emacs` to language servers, `eglot` and `lsp-mode`. Not for me to do a full comparison, but some thoughts of the top of my head were:

eglot

Feels more minimal, maintained by the author of `yassnippet`

lsp-mode

Much more configuration, better documentation in some aspects, strangely difficult `TRAMP` setup

Note that the comparisons stated are neither fair nor particularly useful. I just need something to work quickly with my main programming languages at the moment. I went with `eglot`.

Desiderata

The things I really need, which both provide are:

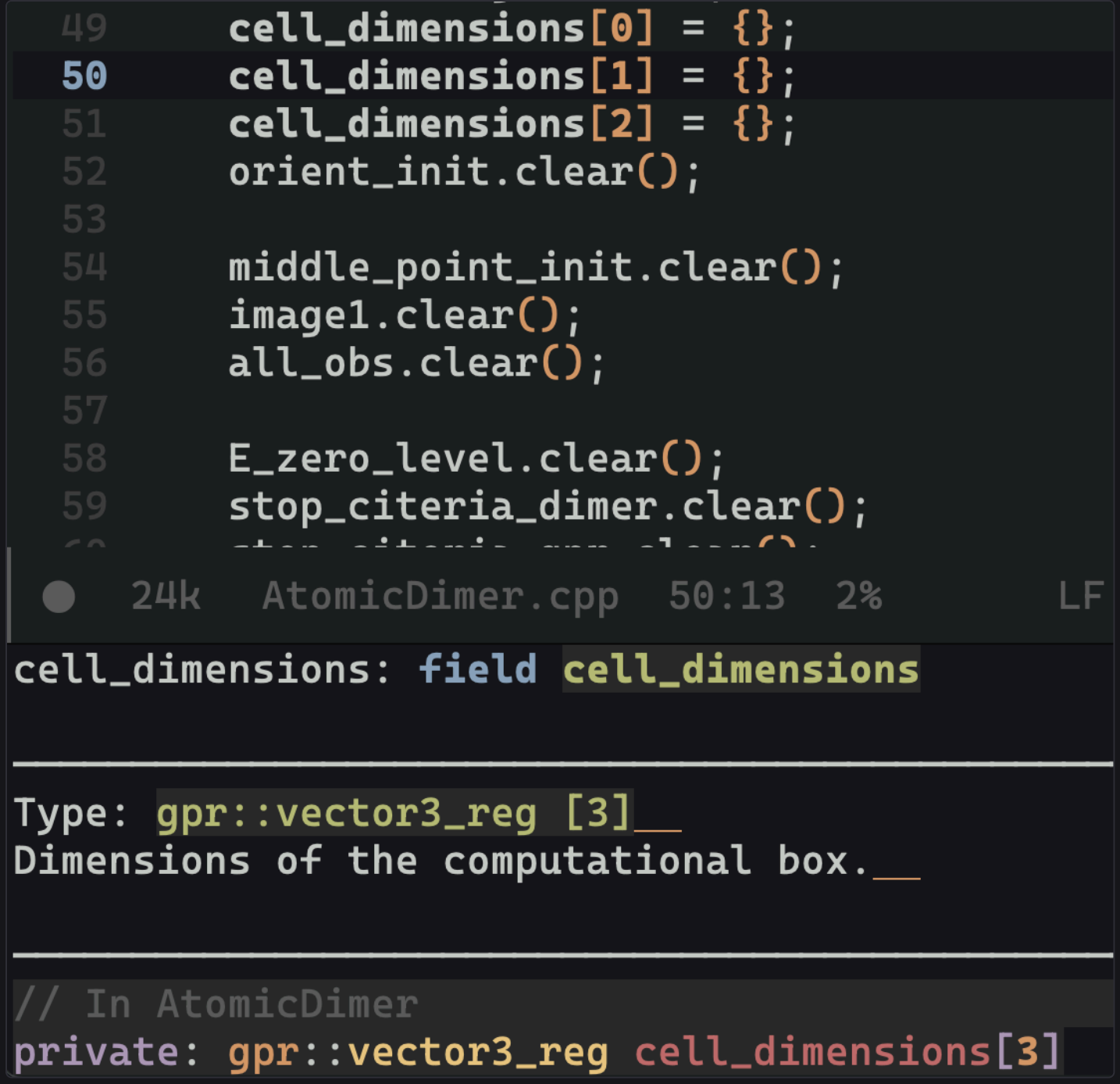
- Easily look up doc-strings
- Auto-completion
- Renaming symbols
- Figuring out common mistakes
- Indexing and lookup for symbol definitions and usage

Eglot

Getting started with `eglot` was surprisingly pleasant, for `doom` ships a pretty out-of the box configuration anyway.

```
;; init.el
(lsp +eglot) ;; Activate eglot
(python +lsp) ;; Python with pyls by default
(cc +lsp) ;; C++ with clangd by default
```

Usage was pretty sweet (after getting `clang-tools` for `clangd`), it can be activated by running `eglot` in any supported buffer, and it came with all the standard bells and whistles.



For working with `tramp` too, after `Emacs 27.1`, in most cases it **just works**, one simply needs to supply the location of the language server executable and we're off to the races. In-fact, even adding the full path to the binary to `PATH` is good enough for `eglot`.

Caveats

- No `projectile` support planned (though there are workarounds)
  - Uses `project.el` to pick root directories (i.e. `.git`)
    - In particular this means it works poorly with `git` sub-modules

Overall the main issue with `eglot` seems to be the insistence to be accepted into `emacs` core someday. This is great, and a good direction for the project to grow in, but it constrains my workflow unnecessarily. I'm more interested in working with `doom-emacs` than vanilla `emacs` and so am pretty invested in non-core libraries like `projectile` [^fn:2].

Language Servers

For getting the language server providers themselves, we will mostly leverage direct binaries where possible, but also, depending on the implementation, virtual environments[^fn:3]. Essentially we have the following Rosetta stone.

Table 1: Language servers and installation methods

Language	Server	Installation	Management
C++	`clangd`	Manual (binary)	None
Python	`pylsp`	`pip`	Micromamba
Bash	`bash-language-server`	`npm`	NVM
R	`languageserver`	`install.packages`	N/A
TeX	`digestiff`	Manual (wrapper)	None
Nix	`rnix-lsp`	Nix	N/A
Fortran	`fortran-language-server`	`pip`	Micromamba

So we need to setup the following, assuming an existing `conda` helper (`micromamba` as described here) setup.

```
# Get Micromamba
mkdir -p ~/.local/bin
export PATH=$HOME/.local/bin:$PATH
wget -qO- https://micromamba.snakepit.net/api/micromamba/linux-64/latest | tar -xvj bin/micromamba
mv bin/micromamba ~/.local/bin
rm -rf bin
micromamba shell init -s bash -p $HOME/.micromamba
. ~/.bashrc
```

Now we can use it.

```
micromamba create -n lsp
micromamba activate lsp
micromamba install python==3.9 pip -c conda-forge
```

Note that for this to work out well, we need to activate this environment by default, with something like `micromamba activate lsp` added to your `.bashrc`. In fact, it is best to also add the full path, since `eglot` doesn't seem to pick it up after `micromamba activate`.

```
export PATH=$HOME/.micromamba/envs/lsp/bin/:$PATH
```

Also, we need to setup `nvm`.

```
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
. $HOME/.bashrc
nvm install node
nvm use node
```

C++

For `clangd` (details here) and `tramp` this amounted to:

```
mkdir -p ~/.local/lsp
cd ~/.local/lsp
wget https://github.com/clangd/clangd/releases/download/12.0.0/clangd-linux-12.0.0.zip
unzip clangd-linux-12.0.0.zip
mv clangd-linux-12.0.0/* .
```

In combination with the standard `doom-emacs` cc module, this is a very good workflow, with the only issue being the ability to set the project root and files to be considered. There is `doom-emacs` version of appending and setting the language server used, which will be used here:

```
(after! eglot
  :config
  (set-eglot-client! 'cc-mode '("clangd" "-j=3" "--clang-tidy"))
  )
```

Note that the best way to make a language server behave is to have a compilation database, which can be generated (for a `CMake` project) as follows:

```
# From src
mkdir build
cd build
# cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 .. -G 'Unix Makefiles' # default on *nix
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 .. -G 'Ninja' # way faster
ninja # or make
cp compile_commands.json ..
```

This will ensure the best experience, since the `compile_commands.json` (described here) file contains paths to all the files used, including system and write-only files [^fn:4]. The setup here works well with enough with the `ccls` language server as well, which in turn is a more maintained fork of `cquery`.

Python

Palantir's `python-language-server` is EOL, and was weird to begin with, but the Spyder team has taken up the gauntlet and maintains the `python-lsp-server` (repo).

```
micromamba activate lsp # should be done in the ~/.bashrc
micromamba install python-lsp-server[all] -c conda-forge
```

For working with this in an optimal manner the `eglot` readme suggests using `.dir-local.el` files. For example:

```
((python-mode
  . ((eglot-workspace-configuration
      . ((:pylsp . (:plugins (:jedi_completion (:include_params t)))))))))
```

It so happens that this method is flexible enough to allow multi-configuration of servers as well. Additionally, we have options for working with the setup:

```
(after! eglot
  :config
  (set-eglot-client! 'python-mode '("pylsp"))
  (set-eglot-client! 'cc-mode '("clangd" "-j=3" "--clang-tidy"))
)
```

Or in vanilla `emacs`:

```
(add-to-list 'eglot-server-programs
             '(python-mode "pylsp"))
```

Bash

This is fairly straightforward, and implemented in `js` here.

```
nvm use node
npm i -g bash-language-server
```

Along with:

```
(sh +lsp) ;; in the init.el file
```

Fortran

The installation of `fortls` (repo) is simple enough.

```
micromamba activate lsp
pip install fortran-language-server
```

Unfortunately, `doom-emacs` does not actually support the `fortran` language server out of the box, so some additional `emacs-lisp` needs to go into the configuration.

```
(add-hook 'f90-mode-hook 'eglot-ensure)
```

This works quite well for larger projects.

```
21 use types, only: dp
22 use lapack, only: dgesv, dgbsv
23 use utils, only: stop_error, assert, newunit, white_char
24 implicit none
25 private
26 public spline3pars, spline3valder, iix, iixmin, iixun, iixexp, poly3,
27     d2poly3, spline3, spline3ders, hermite5interp, lagrange3interp, &
28     hermite_basis_1, hermite_basis_2, loadtxt
29
30 contains
31
● 19k  dftatom/src/interpolation.f90  27:22  3%  LF  F90  maste
SUBROUTINE spline3ders(x, y, xnew, ynew=ynew, dynew=dynew, d2ynew=d2ynew)
 REAL(dp), DIMENSION(:), INTENT(IN) :: x
 REAL(dp), DIMENSION(:), INTENT(IN) :: y
 REAL(dp), DIMENSION(:), INTENT(IN) :: xnew
 REAL(dp), OPTIONAL, DIMENSION(:), INTENT(OUT) :: ynew
 REAL(dp), OPTIONAL, DIMENSION(:), INTENT(OUT) :: dynew
 REAL(dp), OPTIONAL, DIMENSION(:), INTENT(OUT) :: d2ynew
```

Nix

`rnix-lsp` (repo) has no non `nix` installation, and a slightly uncertain future, but it is still fantastic and shouldn't be left out[^fn:5].

```
nix-env -i -f https://github.com/nix-community/rnix-lsp/archive/master.tar.gz
nix-env -i nixkpgs-fmt
```

As this is another unsupported server in `doom-emacs`, the setup needs some tweaking.

```
(add-hook 'nix-mode-hook 'eglot-ensure)
```

TeX

The last of the language servers, `digestif` (repo) is implemented in `lua`, and probably shouldn't be on a remote machine[^fn:6], but nevertheless.

```
mkdir -p ~/.local/lsp/bin
cd ~/.local/lsp/bin
wget https://raw.githubusercontent.com/astoff/digestif/master/scripts/digestif
chmod +x digestif
./digestif
```

This sets up `$HOME/.digestif/bin` which is yet another path to be managed. Note that this assumes a standard `TexLive` installation with `luatex`.

Local Usage

For local use, assuming similar installation instructions, it is easier to manipulate `exec-path`.

```
(after! eglot
  :config
  (add-hook 'nix-mode-hook 'eglot-ensure)
  (add-hook 'f90-mode-hook 'eglot-ensure)
  (set-eglot-client! 'cc-mode '("clangd" "-j=3" "--clang-tidy"))
  (set-eglot-client! 'python-mode '("pylsp"))
  (when (string= (system-name) "blah")
  (setq exec-path (append exec-path '(
                                      (concat (getenv "HOME") "/.micromamba/envs/lsp/bin/") ;; python, fortran
                                      (concat (getenv "HOME") "/.local/lsp/bin/") ;; clangd
                                      (concat (getenv "HOME") "/.digestif/bin/") ;; tex
                                      (concat (getenv "HOME") "/.nvm/versions/node/v16.1.0/bin/bash-language-
server")
                                      )))
    )
  )
```

Where you will need to replace `"blah"` with the output of `(system-name)`.

Conclusions

Embarrassingly, neither setup provided as pleasant an SSH based workflow as VS Code. However, that probably has a lot more to do with TRAMP than any of the language servers, and in any case Magit still makes `emacs` far superior. In any case, the setup described here is still far superior to not using anything other than `projectile` and `ripgrep` so I am satisfied for now. The path setup in this post is deliberately polluting for simplification however, and in practice my Dotfiles are managed a lot better with `bombadil` which is something for a later post. Each of these will take some getting used to, and new keybindings will probably be needed, especially for my idiomatic Colemak setup.