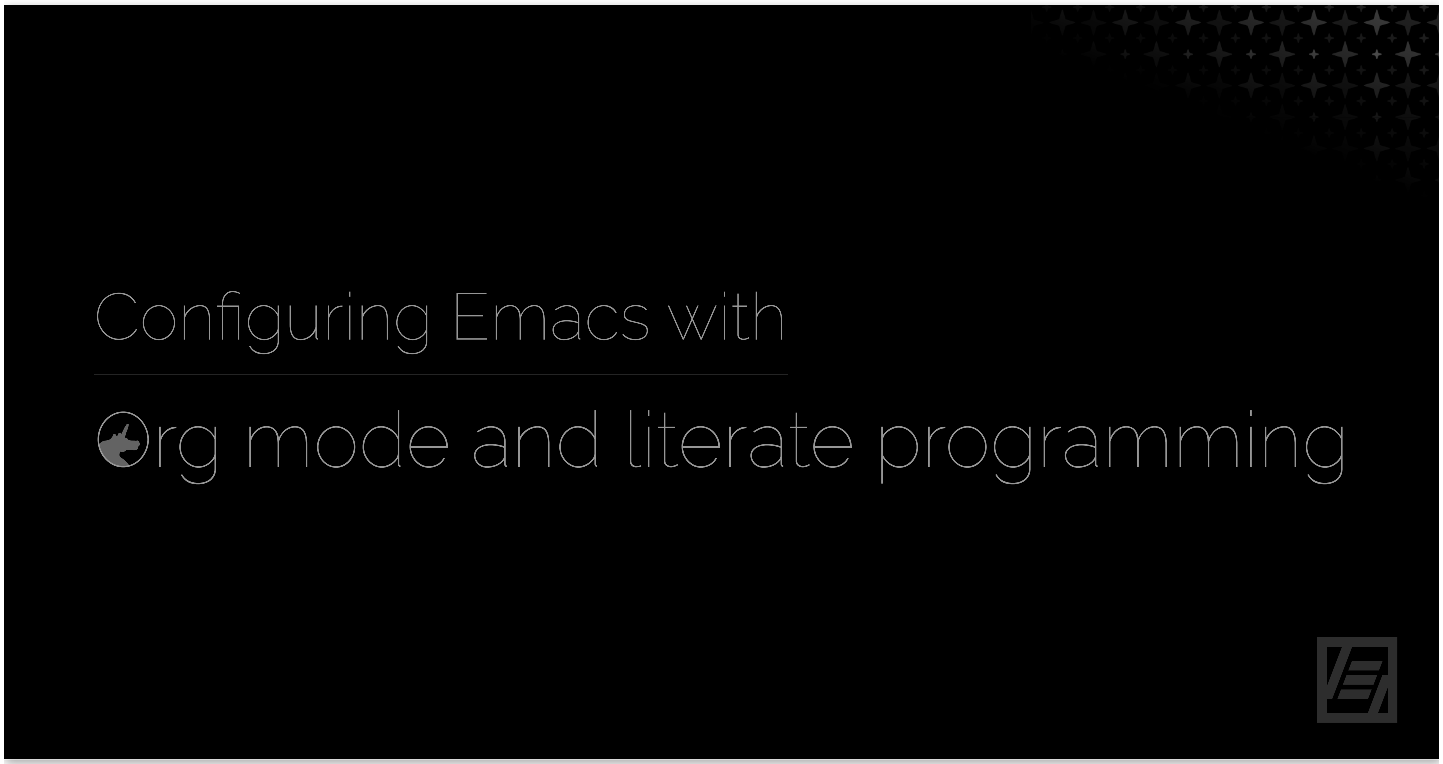
Configuring Emacs with Org mode and literate programming

Thomas Heartman
April 27, 2020 · mod: 08:43, April 27, 2020



Even if you have the source code in front of you, there are limits to what a human reader can absorb from thousands of lines of text designed primarily to function, not to convey meaning. --- Ellen Ullman (from quotes about software aging on literateprogramming.com)

I've been intrigued by *literate programming* for a while now, but never quite found the right opportunity to try it out. All the business with *tangling* and *detangling*, calling code from other snippets, and understanding how or even *if* you can import literate code into non-literate code, made it seem like you'd need to understand a lot just to get started. Fortunately, I found something that doesn't require any of the above: extending your Emacs config.

Using Org mode and literate programming to configure Emacs is something I've heard a

lot about, but I found it difficult to find out exactly how to get started. Luckily it turned out to be pretty simple.
Below, I'll show you how to start configuring Emacs with Org mode and what I've learned about it so far, but if you're really eager to just get started, the trick is to use the function `org-babel-load-file` and give it the path to your Org file.

Heads up!

Disclaimer: I am /not/ an expert at literate programming, or even particularly knowledgeable; these are my first steps into this brave new world. If I've made any mistakes or if you've got tips: don't hesitate to reach out.

Who is this for?

This post is aimed at anyone interested in literate programming with Org mode, but who does not know where or how to get started. It assumes some familiarity with Emacs and Org mode, but no further knowledge of programming is required. Further, I assume no prior knowledge of literate programming.

What is literate programming?

> *I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: "Literate Programming."*

Those are the words of Donald Knuth, the creator of literate programming. Knuth calls for a form of programming where we shift our focus from instructing the computer what to do, to explaining to another person what we *want* the computer to do. This forms the basis for literate programming.

A literate source code file inverts the typical notion of a source code file: rather than being source code with comments strewn around, is a text with source code blocks inserted. The exact file format and medium of the file doesn't matter. The most well-known literate programming tool[1] is probably Jupyter Notebook.

Org mode

With Emacs, Org mode is arguably the most readily available way to do literate programming. Babel, which has been included in Org mode since version 7.0, enhances Org mode's source blocks by providing (as described in the introductory tutorial)

- interactive and on-export execution of code blocks
- code blocks as functions that can be parameterized, that can refer to other code blocks, and that can be called remotely
- export to files for literate programming

In short, this enables you to write Org documents with source code blocks, where the source code blocks can be interacted with and used to generate pure source code.

Configuring Emacs

It turns out that if you're on a semi-recent version of Emacs, it's really very simple. Babel provides a function called `org-babel-load-file` which 'Load[s] Emacs Lisp source code blocks in the Org [file]'. As such, all you need to do is to call this function from your Emacs configuration with the path to the Org file you want to load.

Here's what I've got in my configuration:

```
(org-babel-load-file
 (expand-file-name
  "config.org"
  user-emacs-directory))
```

This snippet assumes that the file you're loading is located in a file called `config.org` in your `user-emacs-directory` (which defaults to `~/.emacs.d`).

This was tested with vanilla Emacs 26.3, so any recent version of Emacs shouldn't need any more configuration than this, but if you want a more detailed guide, check out the Emacs Initialization with Babel section of the Babel introduction.

Benefits of using Org mode for configuration

So far, I've found a number of benefits to using literate programming for my

So far, I've found a number of benefits to using literate programming for my configuration, including (but not limited to):

I'm able to express /why/ I'm making certain configurations
: Not that you *can't* describe what you're doing or why in pure elisp, but I certainly find it harder to be clear about it. Explaining the reasoning behind making certain configuration decisions is not only useful to other people reading my configuration, but also to myself when I return to something after weeks or months (or years) after having last touched it. 'Why did I configure it this way again? Oh, yeah: that's it!"

Furthermore, being able to use a text-based mode to write text not only makes a lot of sense, but it's also much more ergonomic, and offers much more expressibility than code comments.

Formatting
: Another bonus of using a text-based mode is that you can take advantage of text formatting. For instance, being able to include links that only display a link text and not a full URL is a nice bonus. Lists (numbered, unnumbered, and definition lists) are also readily available.

Expanding and collapsing regions
: If you want to get a quick overview over your configuration, it's easy to quickly toggle the headings in the entire file to find exactly what you're looking for. It's also super easy to narrow your editor view to the section you're focusing on right now.

Using tags
: Even better than being able to fold your config and scan through the headings is using tags. Org mode's support for tags makes it a breeze to show only sections that relate to whatever you're looking for. For instance: one of the tags I've defined is `keybinding`. By tagging every section that deals with key bindings with this, I can quickly whip up a view that shows all the relevant sections by using `org-sparse-tree`.

Summary

That sums up my journey into literate programming thus far. I still have a lot to learn and have probably only truly understood a fraction of the power that lies within Org mode, but I've gotten over that first step. It's not (as) scary anymore, and I'm looking forward to learning more about it.

If you came here looking for how to use Org mode to configure Emacs or to do literate programming, I hope you found what you were looking for. And if you came here wondering what this was all about, I hope your curiosity was rewarded and that you're still wondering what this was all about, I hope your curiosity was rewarded and that you're still at least *as* curious or intrigued as before.

Further reading and references

I've included a few links that could be useful or interesting below. Most of them are already linked to in the text above, but included here again for convenience.

Wikipedia on Literate programming
: As per usual, Wikipedia offers a concise and informative take on what literate programming is and where it comes from. In addition to the basics, it also includes a very nice illustration of literate programming by showing certain parts of the Unix word count utility `wc` written using literate programming techniques.

literateprogramming.com
: A website about literate programming. There's not much information about the purpose of the website, nothing to be found about the author, and it looks like it hasn't been updated since 2009, but there are some resource links, and a great many quotes about literate programming and how programs tend to be underdocumented.

Babel: active code in Org-mode
: This website includes links the Babel reference documentation, the introductory tutorial, a journal paper describing the use of Org mode and Babel for literate programming and reproducible research, and more.

My literate Emacs config
: This is the config I use at the time of writing. Note that it is not my complete Emacs configuration, but rather an addition to my main configuration. I'm in the process of (very slowly) moving away from Spacemacs and into my own configuration, so this document reflects that.

Footnotes

[1]

Or at least the only one that I hear mentioned at semi-regular intervals.

[1]

Or at least the only one that I hear mentioned at semi-regular intervals.

Post navigation: continue?

← Save on typing and improve legibility with Rust's macros

Building a request inspector →

Thomas Heartman is a developer, writer, speaker, and one of those odd people who enjoy lifting heavy things and putting them back down again. Preferably with others. Doing his best to gain and share as much knowledge as possible.