
Xi editor

Maintenance status: *The xi-editor project is currently discontinued. Although we will happily accept bug fixes, no new features are currently planned. You may be interested in the Lapce editor, which can be considered a spiritual successor to the xi-editor. — The Editors

Note: *This repo contains only the editor core, which is not usable on its own. For editors based on it, check out the list in Frontends.*

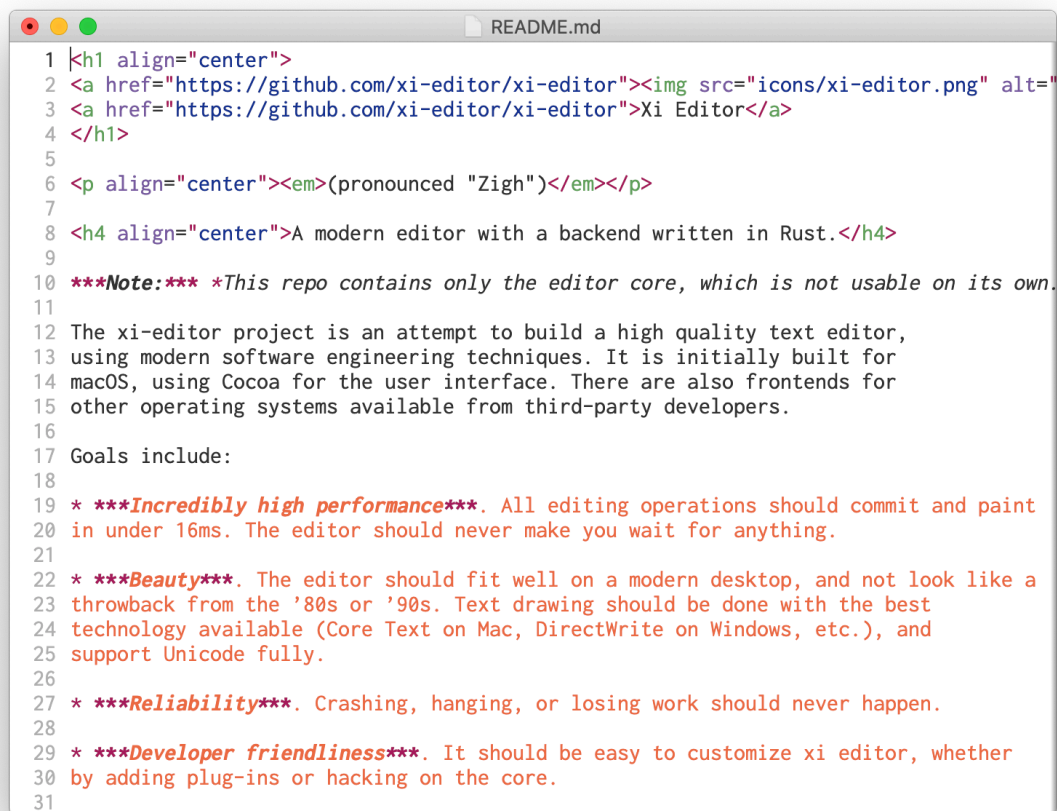
The xi-editor project is an attempt to build a high quality text editor, using modern software engineering techniques. It is initially built for macOS, using Cocoa for the user interface. There are also frontends for other operating systems available from third-party developers.

Goals include:

- **Incredibly high performance.** All editing operations should commit and paint in under 16ms. The editor should never make you wait for anything.
- **Beauty.** The editor should fit well on a modern desktop, and not look like a throwback from the '80s or '90s. Text drawing should be done with the best technology available (Core Text on Mac, DirectWrite on Windows, etc.), and support Unicode fully.
- **Reliability.** Crashing, hanging, or losing work should never happen.
- **Developer friendliness.** It should be easy to customize xi editor, whether by adding plug-ins or hacking on the core.

Learn more with the creator of Xi, Raph Levien, in this Recurse Center Localhost talk.

Screenshot:



```
1 |<h1 align="center">
2 |<a href="https://github.com/xi-editor/xi-editor">Xi Editor</a>
4 |</h1>
5 |
6 |<p align="center"><em>(pronounced "Zigh")</em></p>
7 |
8 |<h4 align="center">A modern editor with a backend written in Rust.</h4>
9 |
10| ***Note:*** *This repo contains only the editor core, which is not usable on its own.
11|
12| The xi-editor project is an attempt to build a high quality text editor,
13| using modern software engineering techniques. It is initially built for
14| macOS, using Cocoa for the user interface. There are also frontends for
15| other operating systems available from third-party developers.
16|
17| Goals include:
18|
19| * ***Incredibly high performance***. All editing operations should commit and paint
20| in under 16ms. The editor should never make you wait for anything.
21|
22| * ***Beauty***. The editor should fit well on a modern desktop, and not look like a
23| throwback from the '80s or '90s. Text drawing should be done with the best
24| technology available (Core Text on Mac, DirectWrite on Windows, etc.), and
25| support Unicode fully.
26|
27| * ***Reliability***. Crashing, hanging, or losing work should never happen.
28|
29| * ***Developer friendliness***. It should be easy to customize xi editor, whether
30| by adding plug-ins or hacking on the core.
31|
```

Getting started

This repository is the core only. You'll also need a front-end, from the list below.

Building the core

Xi-editor targets 'recent stable Rust'. We recommend installing via rustup. The current minimum supported version is 1.40.

To build the xi-editor core from the root directory of this repo:

```
1 > cd rust
2 > cargo build
```

Frontends

Here are some front-ends in various stages of development:

- xi-mac, the official macOS front-end.
- xi-gtk, a GTK+ front-end.
- xi-term, a text UI.
- xi-electron, a cross-platform front-end based on web-technologies.
- Tau, a GTK+ front-end written in Rust. Forked from <https://github.com/bvinc/gxi>, which was abandoned.
- xi-win, an experimental Windows front-end written in Rust.
- kod, a terminal frontend written in Golang.
- xi-qt, a Qt front-end.
- vixi, a Vim like front-end in Rust.

The following are currently inactive, based on earlier versions of the front-end protocol, but perhaps could be revitalized:

- xi_glium, an experimental GL-based front-end in Rust.
- XiEditorVS2015, C#.
- xi-android, an Android frontend.

There are notes (I wouldn't call it documentation at this point) on the protocol at [frontend.md](#). If you're working on a front-end, feel free to send a PR to add it to the above list.

Design decisions

Here are some of the design decisions, and motivation why they should contribute to the above goals:

- **Separation into front-end and back-end modules.** The front-end is responsible for presenting the user interface and drawing a screen full of text. The back-end (also known as “core”) holds the file buffers and is responsible for all potentially expensive editing operations.
- **Native UI.** Cross-platform UI toolkits never look and feel quite right. The best technology for building a UI is the native framework of the platform. On Mac, that's Cocoa.

-
- **Rust.** The back-end needs to be extremely performant. In particular, it should use little more memory than the buffers being edited. That level of performance is possible in C++, but Rust offers a much more reliable, and in many ways, higher level programming platform.
 - **A persistent rope data structure.** Persistent ropes are efficient even for very large files. In addition, they present a simple interface to their clients - conceptually, they're a sequence of characters just like a string, and the client need not be aware of any internal structure.
 - **Asynchronous operations.** The editor should never, ever block and prevent the user from getting their work done. For example, autosave will spawn a thread with a snapshot of the current editor buffer (the persistent rope data structure is copy-on-write so this operation is nearly free), which can then proceed to write out to disk at its leisure, while the buffer is still fully editable.
 - **Plug-ins over scripting.** Most text editors have an associated scripting language for extending functionality. However, these languages are usually both more arcane and less powerful than "real" languages. The xi editor will communicate with plugins through pipes, letting them be written in any language, and making it easier to integrate with other systems such as version control, deeper static analyzers of code, etc.
 - **JSON.** The protocol for front-end / back-end communication, as well as between the back-end and plug-ins, is based on simple JSON messages. I considered binary formats, but the actual improvement in performance would be completely in the noise. Using JSON considerably lowers friction for developing plug-ins, as it's available out of the box for most modern languages, and there are plenty of the libraries available for the other ones.

Current status

This is still a project in its early stages. The Mac build has basic editing functionality (it was used to write this README), but looks very spare and is still missing essentials such as auto-indent. At the moment, it's expected that its main community will be developers interested in hacking on a text editor.

Authors

The xi-editor project was started by Raph Levien but has since received contributions from a number of other people. See the AUTHORS file for details.

License

This project is licensed under the Apache 2 license.

Contributions

We gladly accept contributions via GitHub pull requests. Please see CONTRIBUTING.md for more details.

If you are interested in contributing but not sure where to start, there is an active Zulip channel at #xi-editor on <https://xi.zulipchat.com>. There is also a #xi channel on irc.mozilla.org. Finally, there is a subreddit at [/r/xi_editor](https://www.reddit.com/r/xi_editor).