Quick User's Guide

Everything you need to be productive in twenty minutes

Runic 0.61.1 · build 73e0f01 · September 2026

A fast, native macOS editor for very large files.

Quick User's Guide

What Runic is
The five-minute version

Installing

The window
Tabs and windows

Opening and moving around
Opening files
Navigating

Editing

Finding and replacing

Lenses — the big idea
The table lens
The log lens
The hex lens

Running SQL against your data

Panels

Reshaping files

Working on a remote server
The host agent

The AI assistant

Making it yours

Shortcuts worth memorising

What Runic is

Runic is a native macOS text editor built around one promise: **open a multi-gigabyte file in well under a second, scroll it at 60 fps, and use almost no memory doing it.**

Most editors load a file into memory. Runic maps it instead, and only ever reads the lines you can actually see. That single decision is why a 2 GB log opens as fast as a 2 KB config file, and it shapes everything else in the app — search streams, formatting streams, and even a file on a remote server only sends you the bytes on screen.

If you have ever watched an editor beachball on a large file, or been told a file is “too large to open”, Runic is built for that moment.

The five-minute version

1. **Open anything.** Drag a file onto the icon, or press `⌘0`. Size is not a concern.
2. **Jump.** `⌘L` goes to a line or byte offset. `⌘P` opens any file in your folder by name.
3. **Find.** `⌘F` searches by streaming the file — no full read, no waiting.
4. **Change how you look at it.** The same file can be text, hex, a table, or a log. See *Lenses*.
5. **Everything is a command.** `⇧⌘P` opens the palette; type what you want.

Installing

Download `Runic-<version>.dmg`, open it, and drag Runic to your Applications folder. If Runic downloads an update itself, it verifies the package and can install and relaunch after you approve it; otherwise it shows the DMG for manual installation.

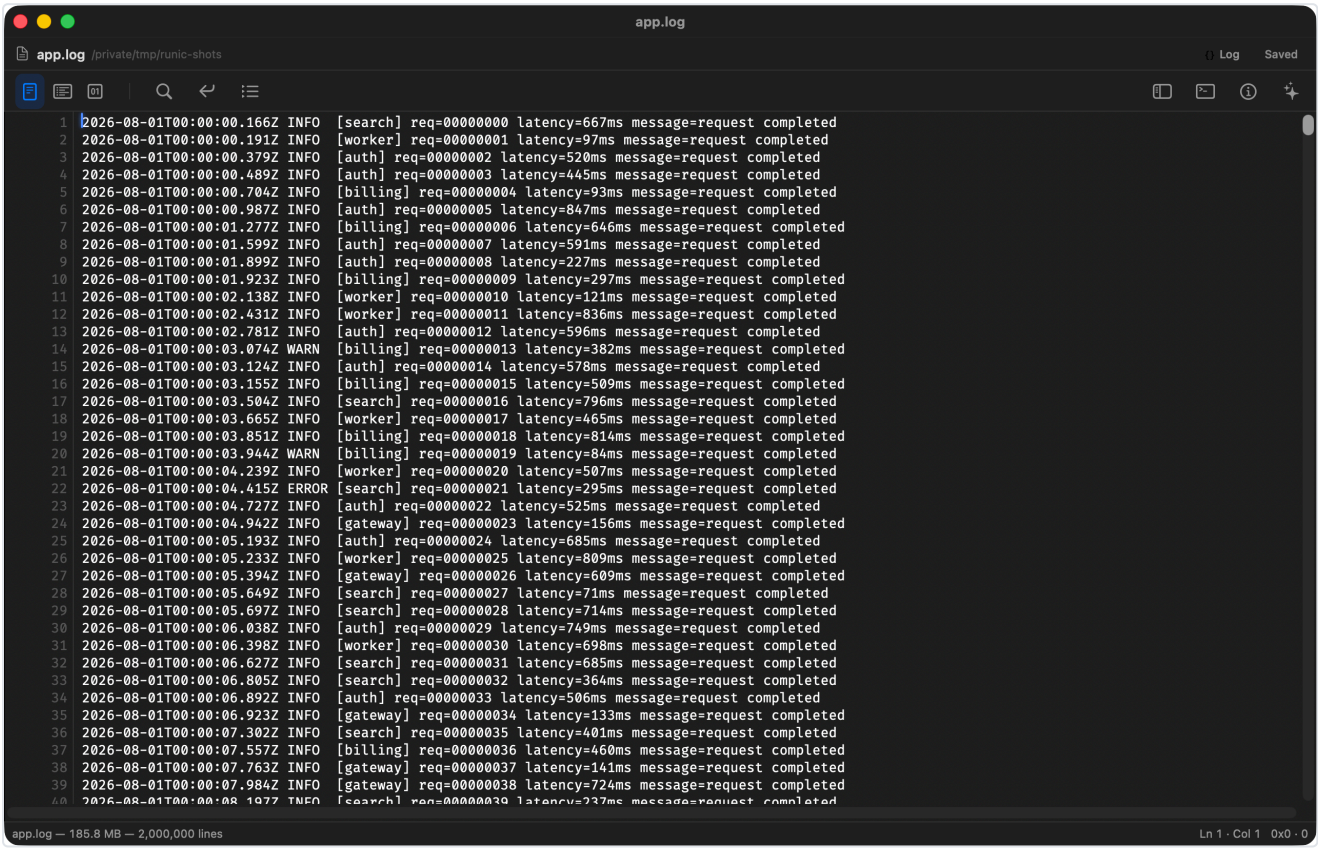
The app is signed with a Developer ID certificate and notarized by Apple, so it launches without warnings — including the first launch on a machine that has never seen it, and offline (the notarization ticket is stapled into the app itself).

To check your version, choose **Runic › About Runic**. It shows the version and the exact build it was made from, like `0.57.0 (281cc34)`.

Updates. Runic checks quietly on launch and shows a banner when a newer version exists. Choose **Help › Check for Updates…** to check manually.

The window

Runic’s window is deliberately plain. From top to bottom:



The window, showing a 186 MB log open in the text lens. Header bar with name and path at the top, the ribbon beneath it, the gutter and editor filling the middle, and the status bar reporting file size and line count.

AREA	WHAT IT SHOWS
Header bar	The document’s name and path, plus editor state — encoding, whether it’s remote, the breadcrumb of where your cursor sits
Find bar	Hidden until you press ⌘F
Ribbon	A single row of controls that changes with what you’re doing — the table lens shows filter and query, the log lens shows Follow
Editor	The file itself
Status bar	Line and column, file size, encoding, and indexing progress on a large file

The **ribbon** is worth understanding: it is not a fixed toolbar. It shows the controls that make sense for the current lens and document type, and nothing else. Switch to the table lens and the filter box appears; switch back to text and it’s gone.

Tabs and windows

Runic uses native macOS tabs. `⌘T` opens a new tab, `^Tab` and `^⇧Tab` move between them, and **Merge All Windows** collects scattered windows into one tabbed window.

Opening and moving around

Opening files

- `⌘O` — open a file
- `⌥⇧⌘O` — open a *folder* into the Navigator sidebar
- Drag a file onto the Dock icon, or use Finder's "Open With"

Once a folder is open, `⌘P` (**Quick Open**) finds any file in it by typing part of the name.

Navigating

DO THIS	PRESS
Go to line or byte offset	`⌘L`
Go to a byte offset specifically	`^⌘G`
Go to a symbol in this file	`⇧⌘O`
Open the command palette	`⇧⌘P`

On a very large file, the status bar shows indexing progress. **You do not have to wait for it.** The part that has been indexed is already scrollable and searchable while the rest catches up.

Editing

Editing works the way you expect — type, select, undo with `⌘Z`. What is different is underneath: Runic never rewrites the whole file to make a change, so editing a line in the middle of a 3 GB file is instant, and so is undoing it.

Multiple cursors. `⌘D` selects the next occurrence of what you have selected, letting you edit several places at once. **Add Cursor Above** and **Add Cursor Below** do what they say.

Saving. `⌘S` saves by writing a complete new copy and atomically swapping it in, so a crash or power loss mid-save can never leave you with half a file. Your original is either fully intact or fully replaced —

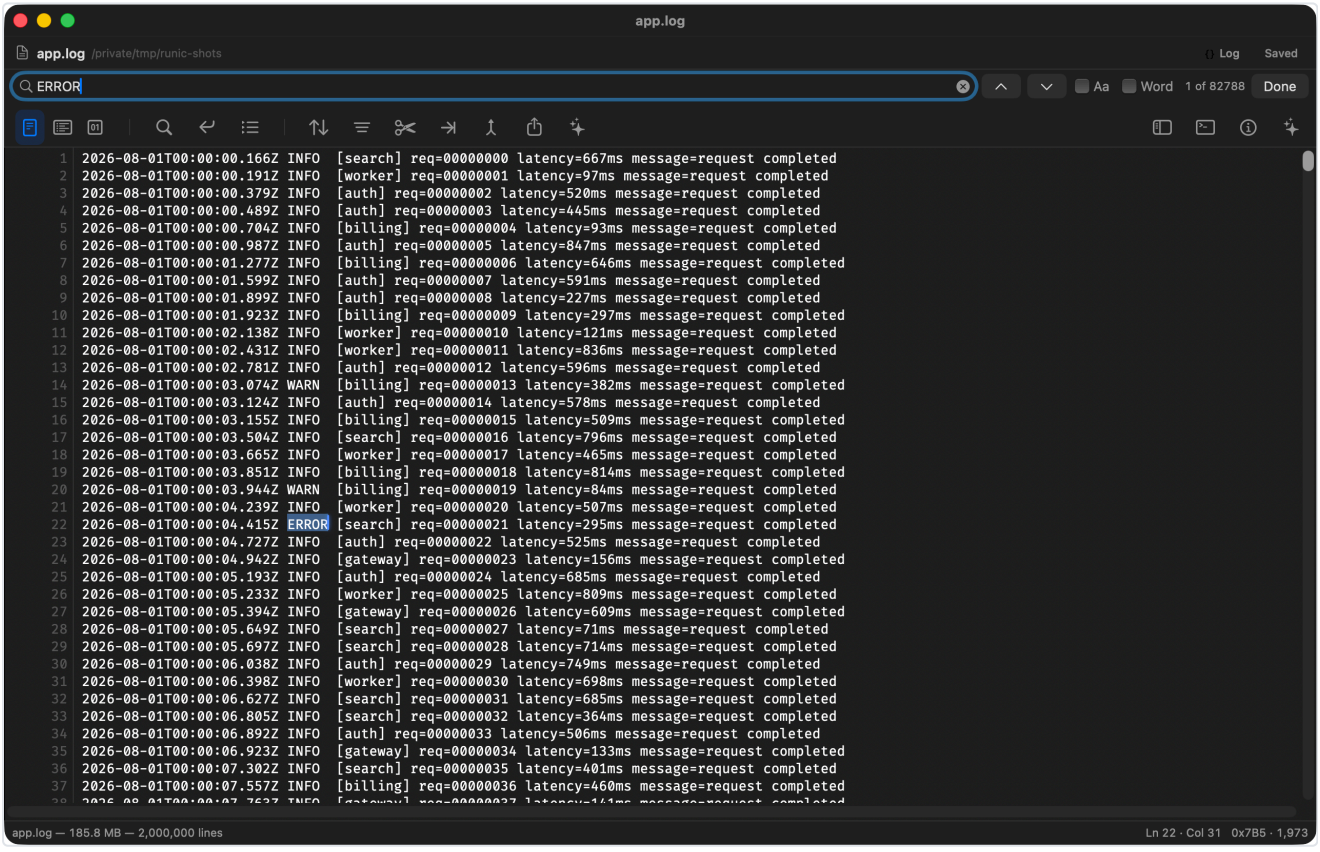
never in between.

> **If the file changes on disk while you have it open**, Runic notices. A file you have not edited reloads automatically, which makes it a good log viewer. A file you *have* edited shows a conflict banner instead, and your edits are never silently discarded.

Finding and replacing

Press `⌘F`. The find bar streams through the file rather than loading it, so searching a huge file starts returning matches immediately instead of freezing.

DO THIS	PRESS
Find	`⌘F`
Find and replace	`⌥⌘F`
Next / previous match	`⌘G` / `⇧⌘G`
Close the find bar	`Esc`

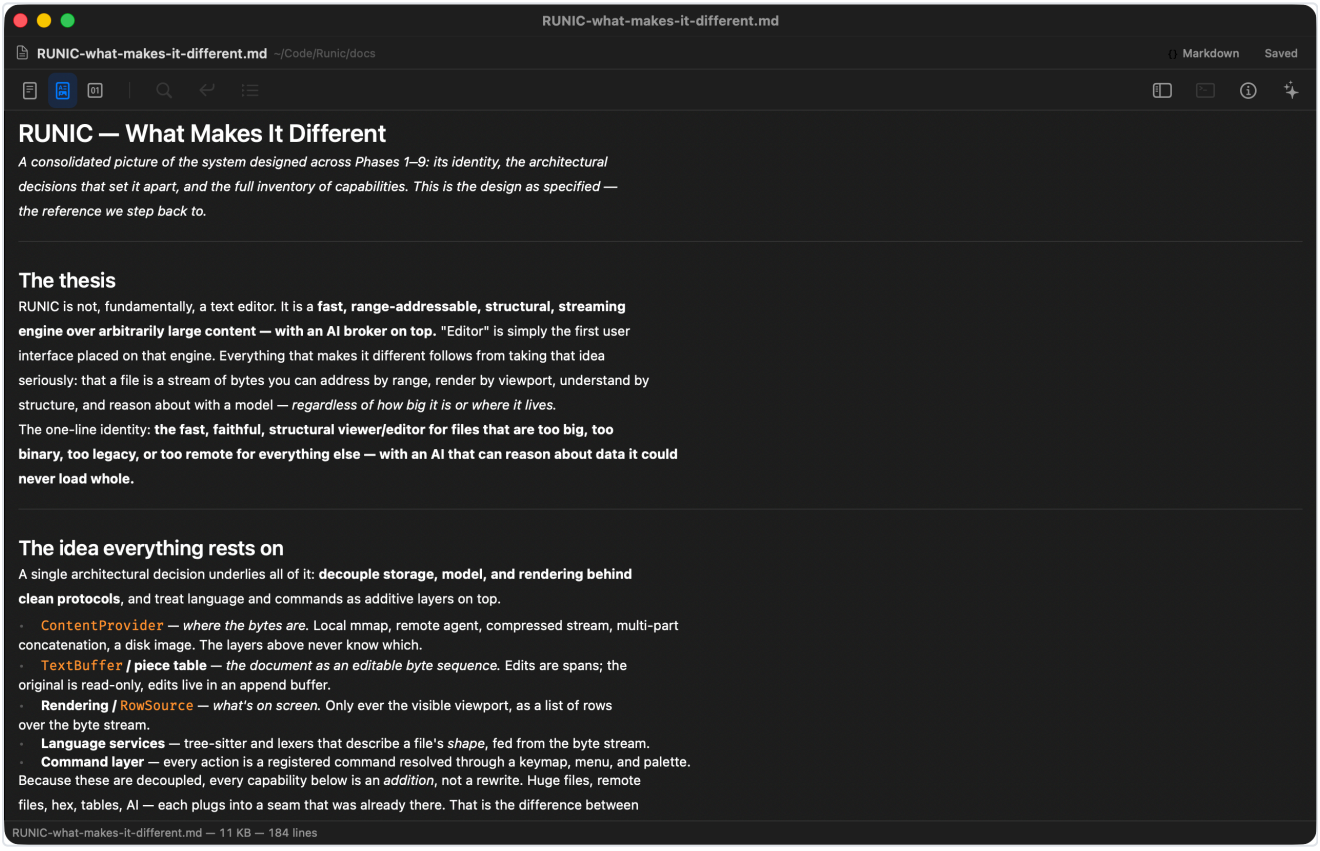


The find bar searching a two-million-line log. The match count is live — 82,788 hits — and search runs over the byte stream, so it does not wait for the whole file to be indexed.

Lenses — the big idea

A **lens** is a way of looking at the same bytes. Runic does not convert your file or open a different document; it just draws it differently. Switch with **View › Show as**.

LENS	GOOD FOR
Text	Normal editing. The default.
Hex	Binary files, or seeing exactly what bytes are there. Offsets, bytes, and ASCII side by side.
Table	CSV, TSV, and NDJSON — shown as real rows and columns
Log	Log files — adds tailing and colour rules
Markdown	Reading `.md` files rendered, rather than as source



The Markdown reading lens. Rendered natively — headings, emphasis, lists, and inline code are drawn by the app, with no web view anywhere.

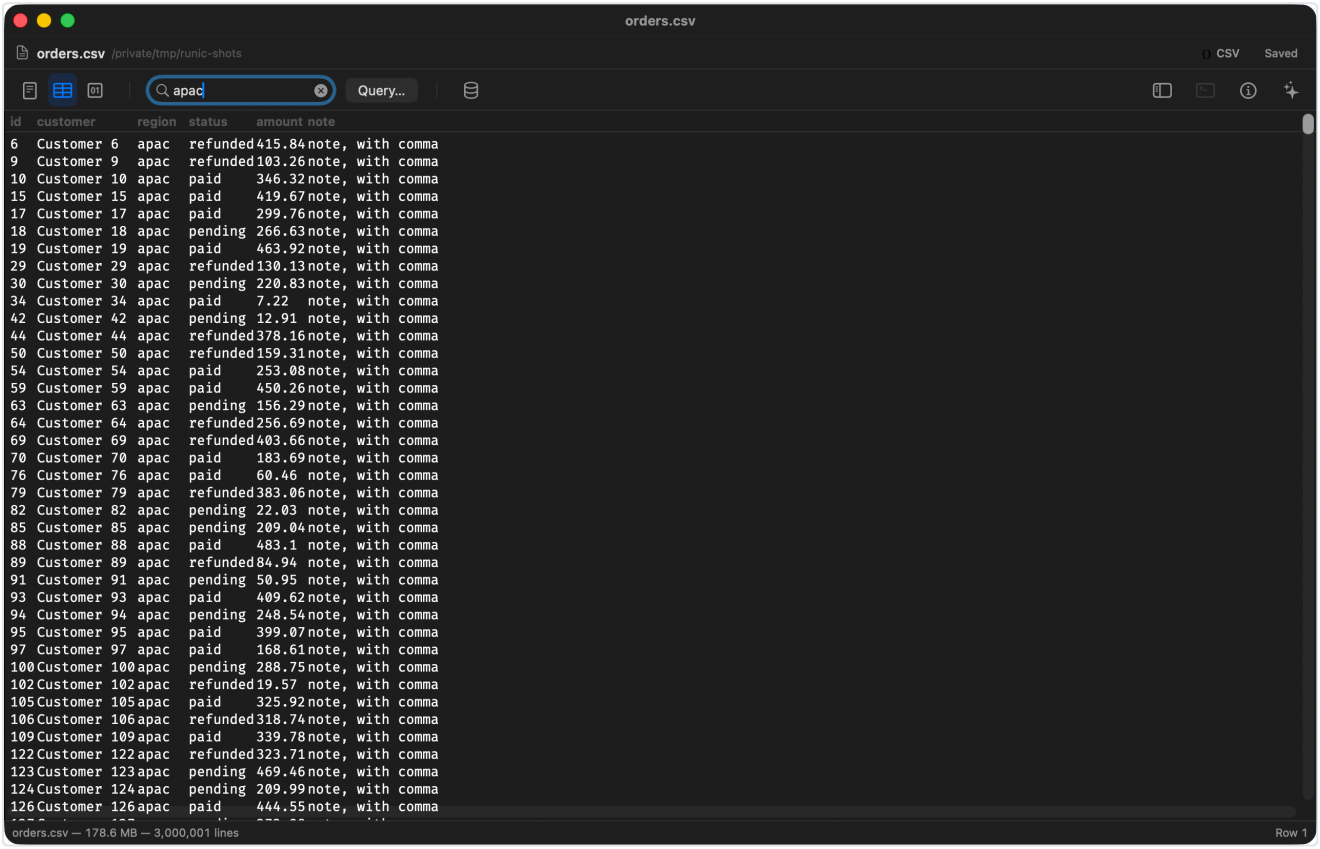
Runic suggests a lens when you open a file — a `.csv` opens as a table, a binary file opens as hex — but you are never locked in. The text view is always one menu click away.

The table lens

CSV, TSV, and NDJSON files become a real grid. Both rows *and* columns are virtualized, so a million-row CSV scrolls smoothly.

- **Filter** narrows to matching rows, with a live count as it scans
- **Query Table…** runs actual SQL against the file — see below
- Clicking a row pretty-prints that record, which is the fastest way to read one dense NDJSON line

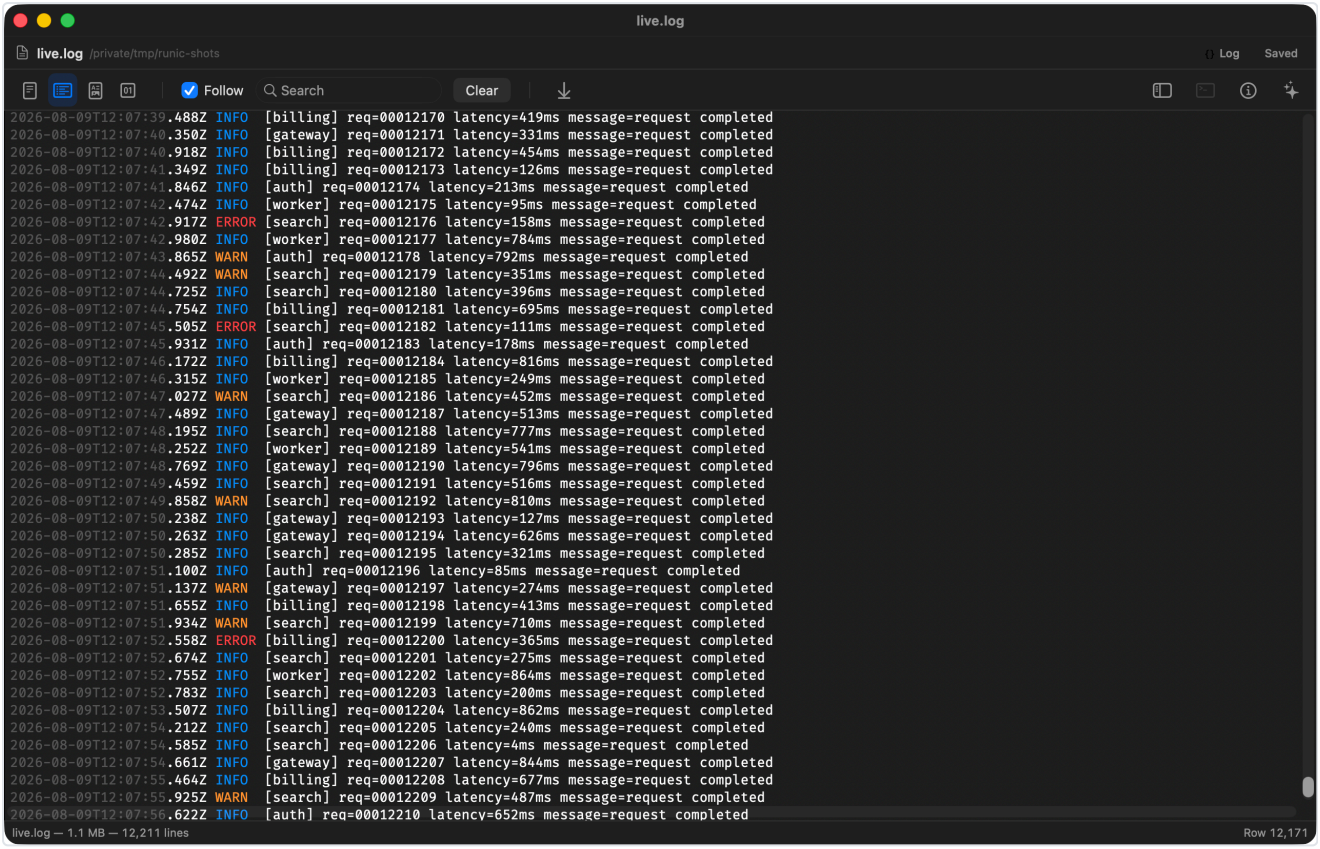
Quoted fields containing commas or newlines are handled correctly, not approximated.



The table lens on a 179 MB, three-million-row CSV, filtered to `apac`. Columns are parsed from the header row; the quoted `note` field keeps its embedded comma intact.

The log lens

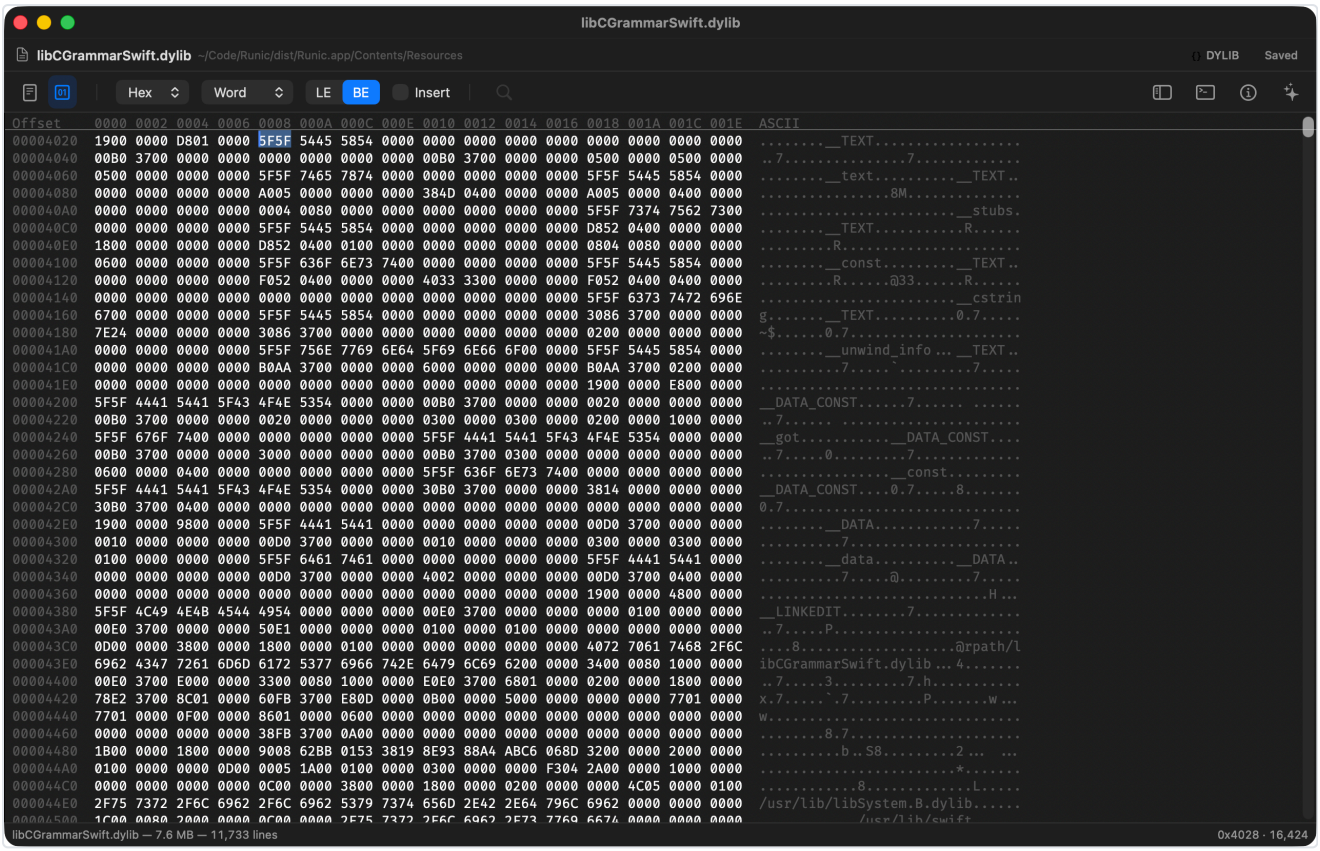
- **Follow (Tail)** sticks to the bottom as the file grows, like `tail -f`
- **Filter Log…** shows only matching lines, streaming as it goes
- **Add Highlight Rule…** colours anything matching a pattern — `ERROR` in red, a request ID in blue — and the rule is saved and applies everywhere



The log lens tailing a file that is still being written. **Follow** is on, so the view stays pinned to the newest line; the default rules colour `ERROR` red and `WARN` amber.

The hex lens

Shows `offset | bytes | ASCII`. You can edit the raw bytes, and non-UTF-8 bytes survive exactly. The ribbon lets you change the number base (hex, decimal, octal, binary), group bytes into words or quadwords, and switch byte order.



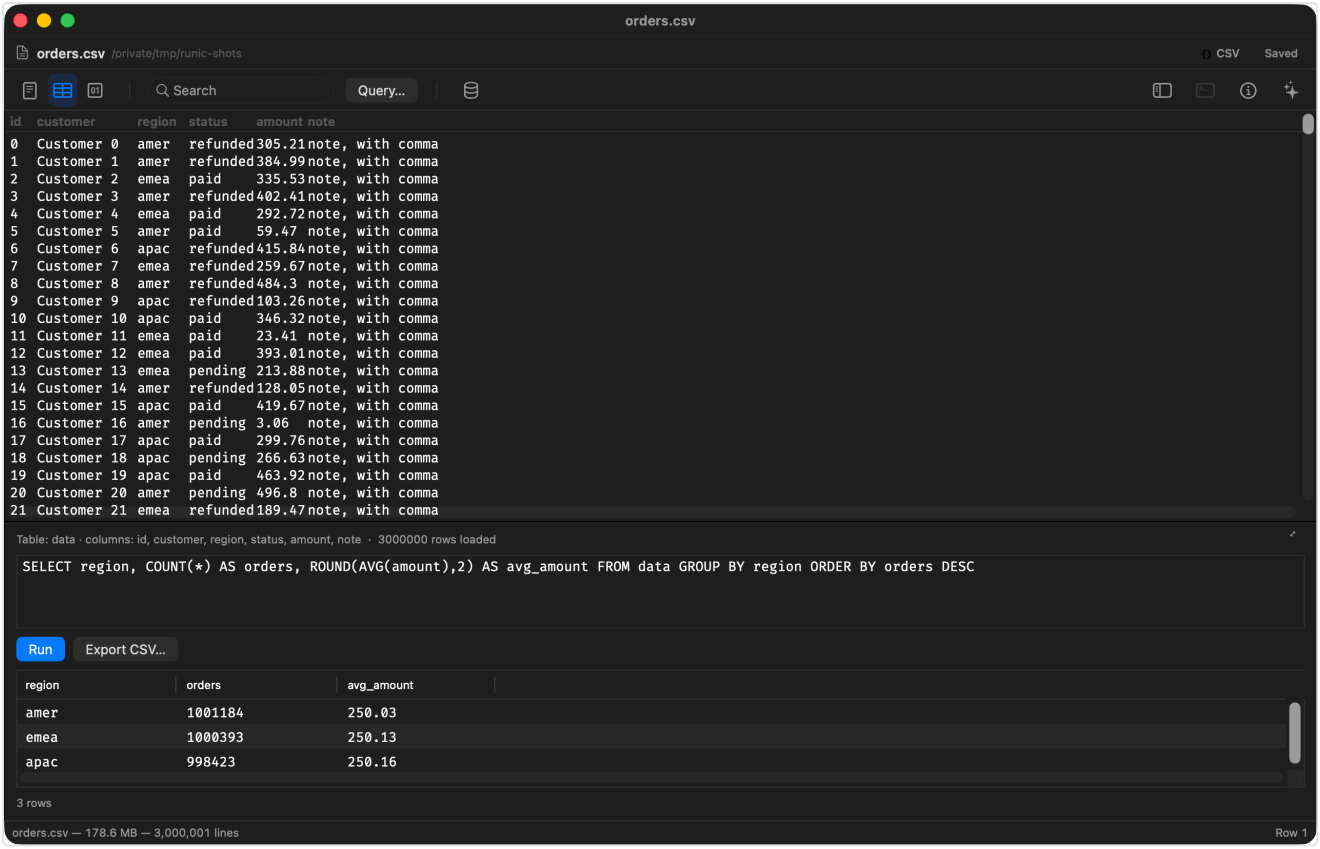
The hex lens on a Mach-O dylib, at the segment table. Offset, bytes, and ASCII side by side; the ribbon carries the radix, grouping, and byte-order controls.

Running SQL against your data

With a CSV or NDJSON file open in the table lens, **Query Table…** lets you write SQL against it:

```
SELECT status, COUNT(*) FROM data GROUP BY status ORDER BY 2 DESC
```

Nothing is imported and nothing is copied — the query reads straight from the mapped file, so there is no row limit and no waiting for a load step. Results appear in the SQL panel at the bottom.



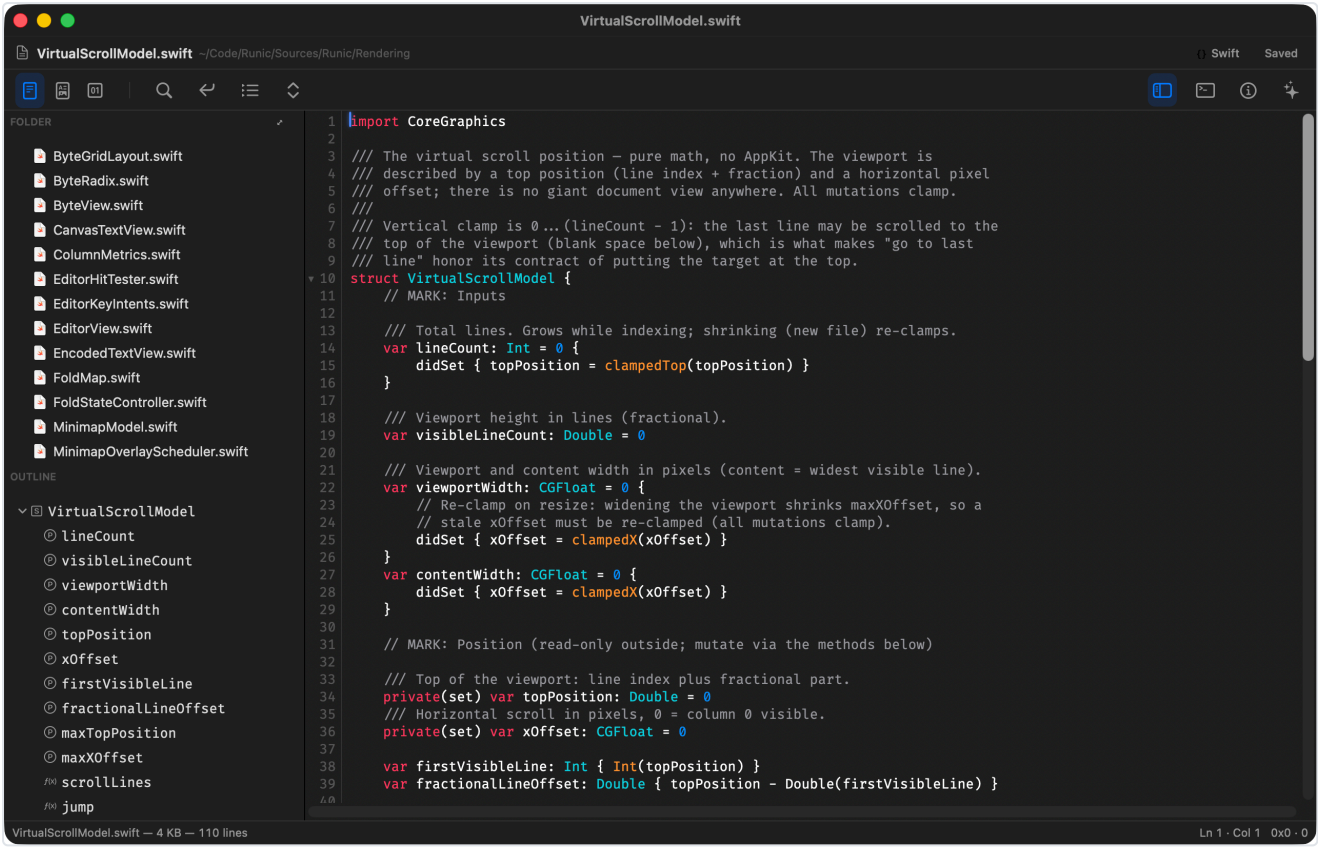
The SQL panel, docked below the table, aggregating all three million rows. The file is queried in place — nothing was imported.

Panels

Panels dock to the left, right, or bottom, and can be floated into their own window with the ⤢ button. Toggle them under **View › Panels**.

PANEL	PRESS	WHAT IT IS
Navigator	⌥⌘0	Folder tree plus an outline of the current file
Assistant	⇧⌘A	The AI chat panel
Inspector	^⌘I	Details about the current document
SQL	—	Query results for the table lens
Shell	—	A terminal, docked at the bottom
Performance	—	Live timings, if you want to see the engine working

The **Navigator** holds two collapsible sections: a **Folder** tree, and an **Outline** of the file you are editing — headings in Markdown, structure in code.

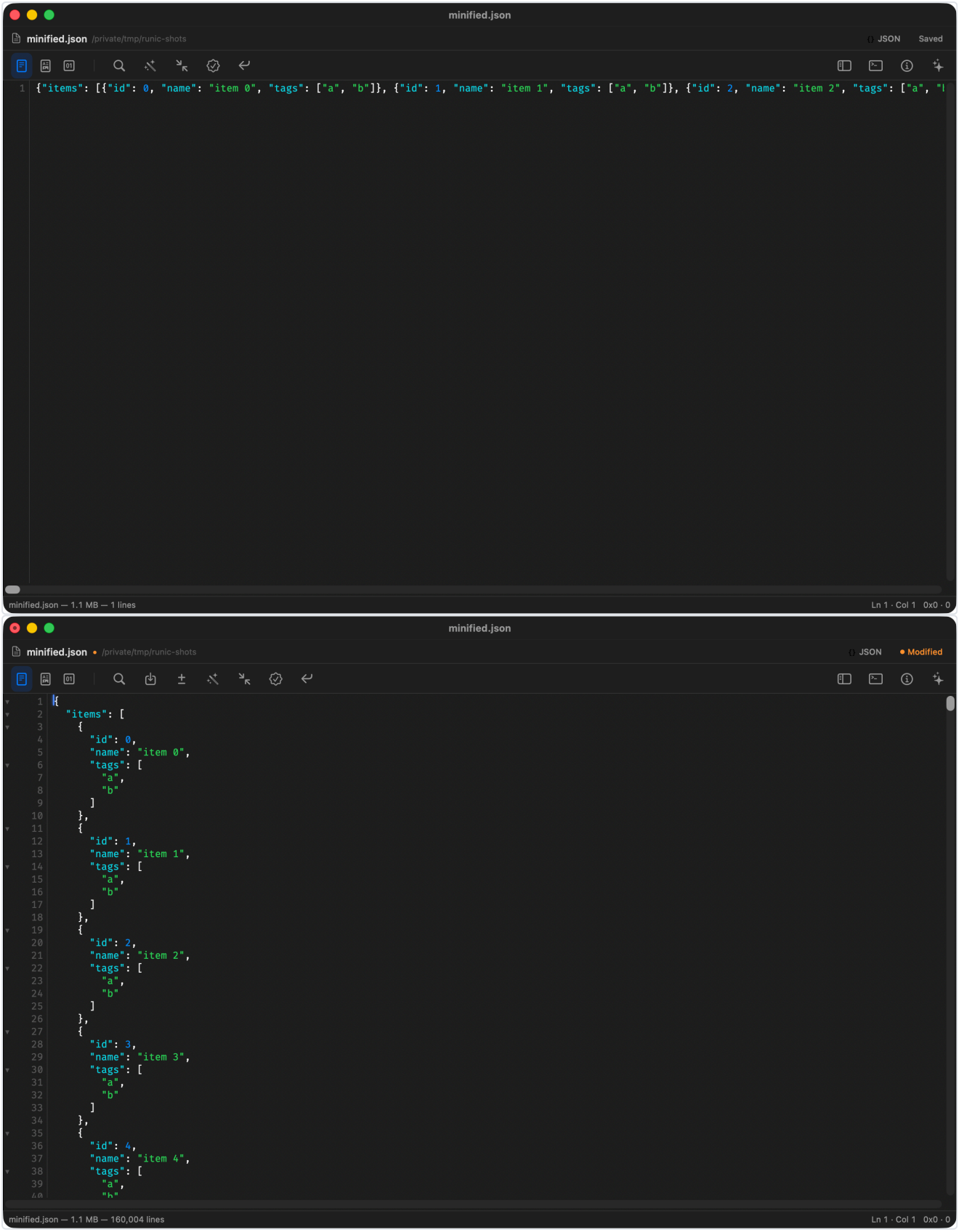


The Navigator on a Swift file: the **Folder** tree above, the **Outline** below. The symbol list comes from the parse tree, so it tracks real declarations rather than a regex. Either section collapses to its header with the disclosure triangle.

Reshaping files

Format Document (⌥⇧F) pretty-prints JSON, XML, or TOML. **Minify Document** goes the other way for JSON and XML.

These stream rather than parsing the whole file into memory, so you can pretty-print a JSON file far larger than RAM. Your strings, key order, and number formatting are preserved exactly — it re-indents, it does not rewrite your data.



Format Document on a 1.1 MB minified JSON file. Before (top): the whole document is a single line, so it scrolls horizontally rather than wrapping. After (bottom): 160,004 lines, indented, with fold arrows in the gutter — one undo away from where it started.

Other tools under the **Code** menu:

- **Trim Trailing Whitespace**, **Collapse Whitespace**, **Join Lines**

- **Sort Selected Lines**, **Remove Duplicate Lines**
- **Filter Through Command…** — pipe the selection through any shell command and replace it with the output
- **Validate Against JSON Schema…** — check a JSON file against a schema
- **Fold / Unfold** (`⌥⌘.`) — collapse structure

Wrap Lines (`⌥Z`) toggles soft wrapping. Very long lines — a minified JSON file on one line, say — are deliberately *not* wrapped, because laying out millions of visual rows would be slow; Runic offers to pretty-print instead.

Working on a remote server

Runic can edit files on another machine over SSH, using **File › Connect to Host…**. Enter a host as `user@host` and a path.

Give it a file path and the file opens in a tab. **Give it a directory** and the folder opens in the Navigator, and you can browse and open files from it.

Authentication is entirely your own `ssh` — keys, `~/.ssh/config`, `known_hosts`, ssh-agent. **Runic never asks for or stores a password.** If `ssh user@host` works in Terminal, Runic works.

> **Use `user@host`, not just `host`,** unless your local username is also the remote one. Plain `host` tells ssh to log in as *you*, which is usually not what you want on a server.

The host agent

For the full experience, install the Runic agent on the server — a small headless program with no interface. With it, searching and indexing happen *on the server*, and only matches and the lines on your screen cross the network. That is what makes editing a 50 GB remote log practical.

Without it, Runic still works over plain ssh, but everything streams to your Mac. The header shows **· no agent** so you always know which mode you are in.

Saving only ever uploads what actually changed — editing a few lines of a huge remote file sends kilobytes, not gigabytes.

The AI assistant

Open the Assistant panel with `⇧⌘A`, or use **AI › Edit Selection with AI…** to rewrite a selection.

Three things are worth knowing:

Your whole file is never sent. Runic assembles a bounded context — your selection, the enclosing function or block, what is on screen — and shows it as removable chips with a byte count *before* anything leaves your machine. A cursor inside a function in a 2 GB file sends that function.

Your API key lives in the macOS Keychain, never in a file and never in a URL.

AI edits never bypass review. A response becomes a diff you accept or reject hunk by hunk, and applying it is a single undo step.

Choose your provider under **AI › Choose AI Provider…** — Anthropic, any OpenAI-compatible endpoint, or Ollama running locally on your own machine.

Making it yours

- **Settings…** — fonts, tab width, theme, and defaults
- **Keyboard shortcuts** — every shortcut is remappable by editing `~/Library/Application Support/Runic/keymap.json`. Every command has an id (like `file.save`), listed in the full manual.
- **Themes** — Runic follows your system light/dark setting automatically

Shortcuts worth memorising

⌘O Open	⌘S Save
⌘P Quick Open	⇧⌘P Command palette
⌘L Go to line	⇧⌘O Go to symbol
⌘F Find	⌥⌘F Find & replace
⌘G Find next	⇧⌘G Find previous
⌘D Add next occurrence	⌥⌘. Fold
⌥Z Wrap lines	⌥⇧F Format
⌥⌘O Navigator	⇧⌘A Assistant
⌘T New tab	^Tab Next tab

Everything else is in the command palette (`⇧⌘P`) — and in the full manual, which documents every menu, every option, and all commands.