



Quick User's Guide

Everything you need to be productive in twenty minutes

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 `⌘O`. 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.

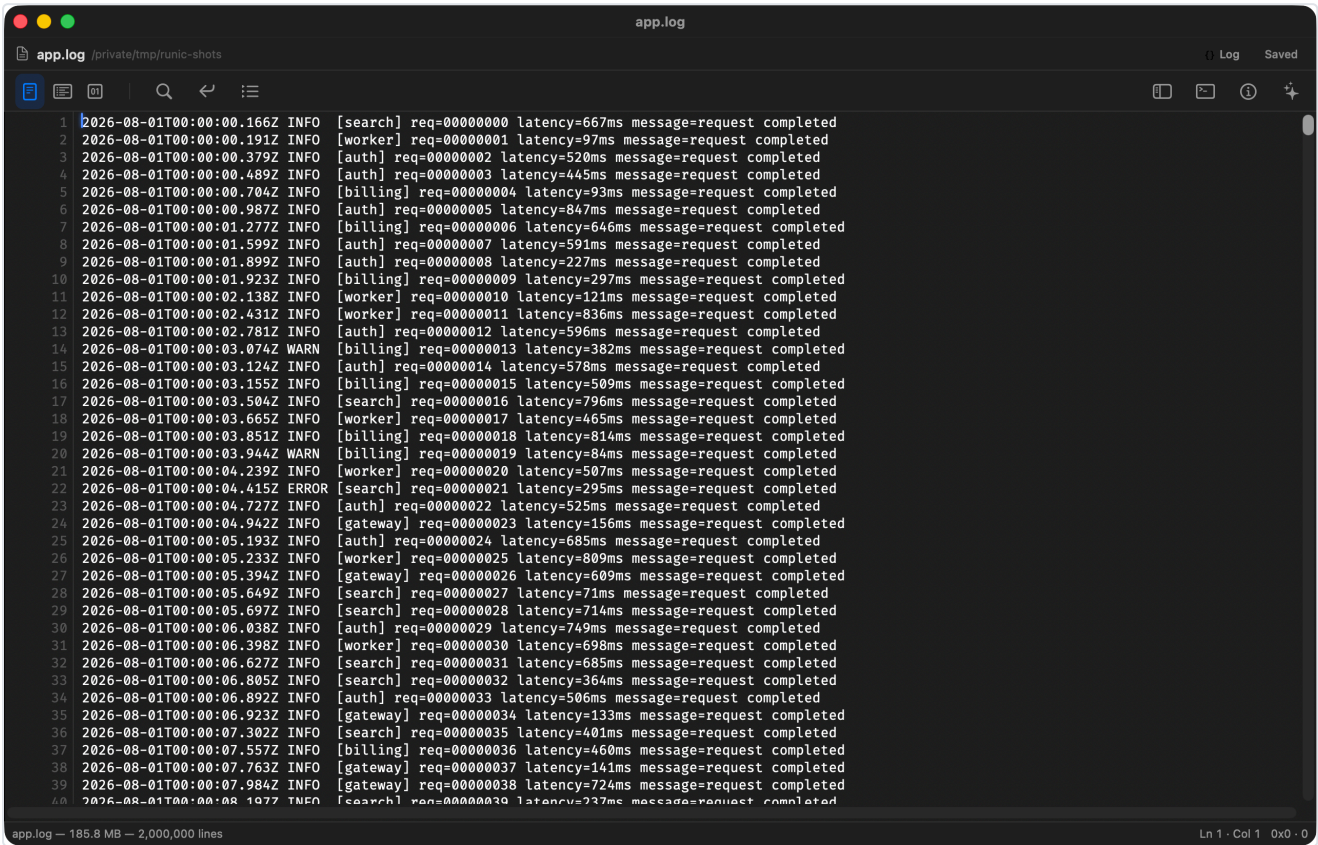
The app is signed with a Developer ID certificate and notarized by Apple, so it launches without warnings — including the first launch on a machine that has never seen it, and offline (the notarization ticket is stapled into the app itself).

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.

Use the smaller and larger **A** buttons before Search to adjust the font size without opening Settings. Each click changes it by one point in every open window, and Runic remembers the size. Rendered Markdown and the Assistant panel follow it too.

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	<code>⌘L</code>
Go to a byte offset specifically	<code>⌘G</code>
Go to a symbol in this file	<code>⇧⌘O</code>
Open the command palette	<code>⇧⌘P</code>

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.

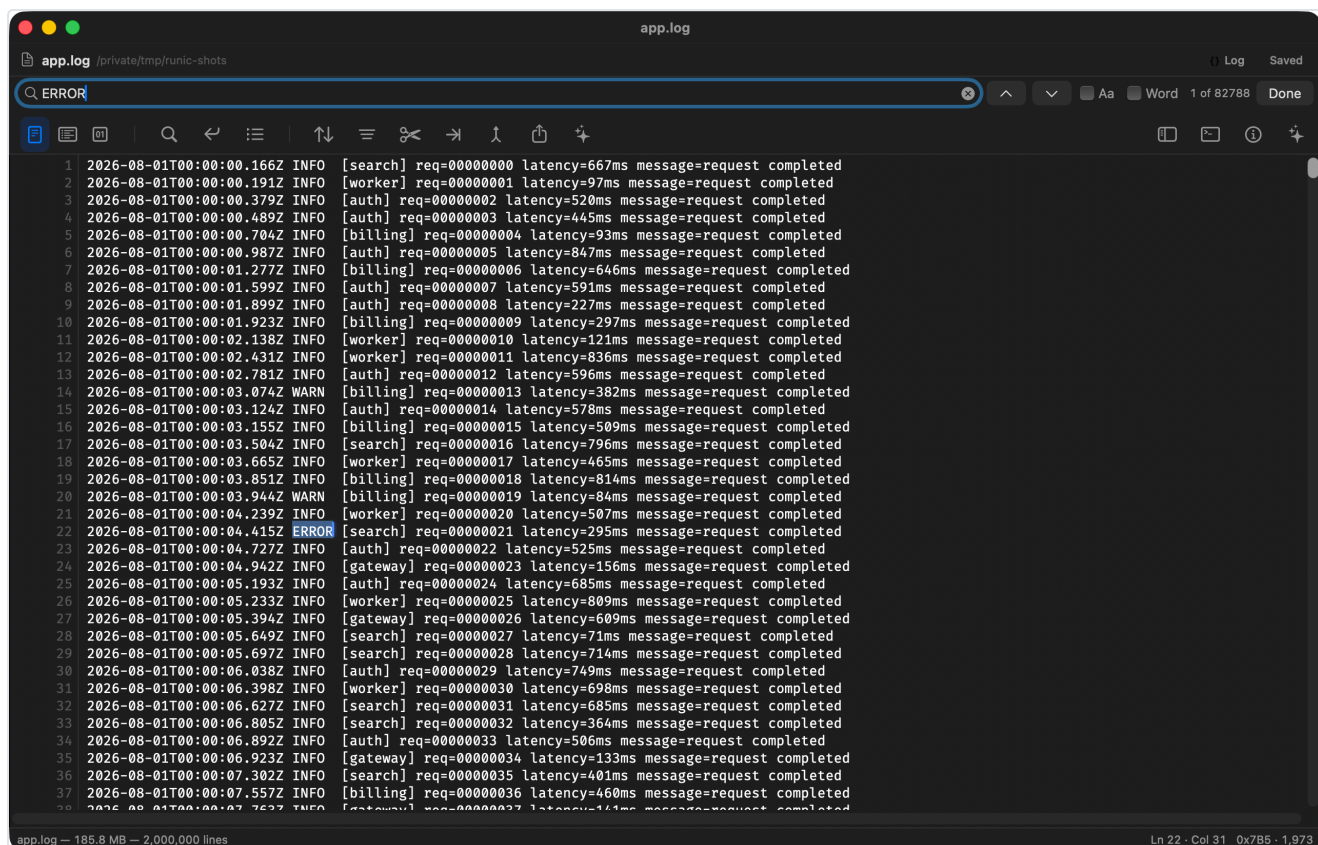
Saving. `%S` saves by writing a complete new copy and atomically swapping it in, so a crash or power loss mid-save can never leave you with half a file. Your original is either fully intact or fully replaced — never in between.

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	<code>%F</code>
Find and replace	<code>⌘%F</code>
Next / previous match	<code>%G</code> / <code>↑%G</code>
Close the find bar	<code>Esc</code>

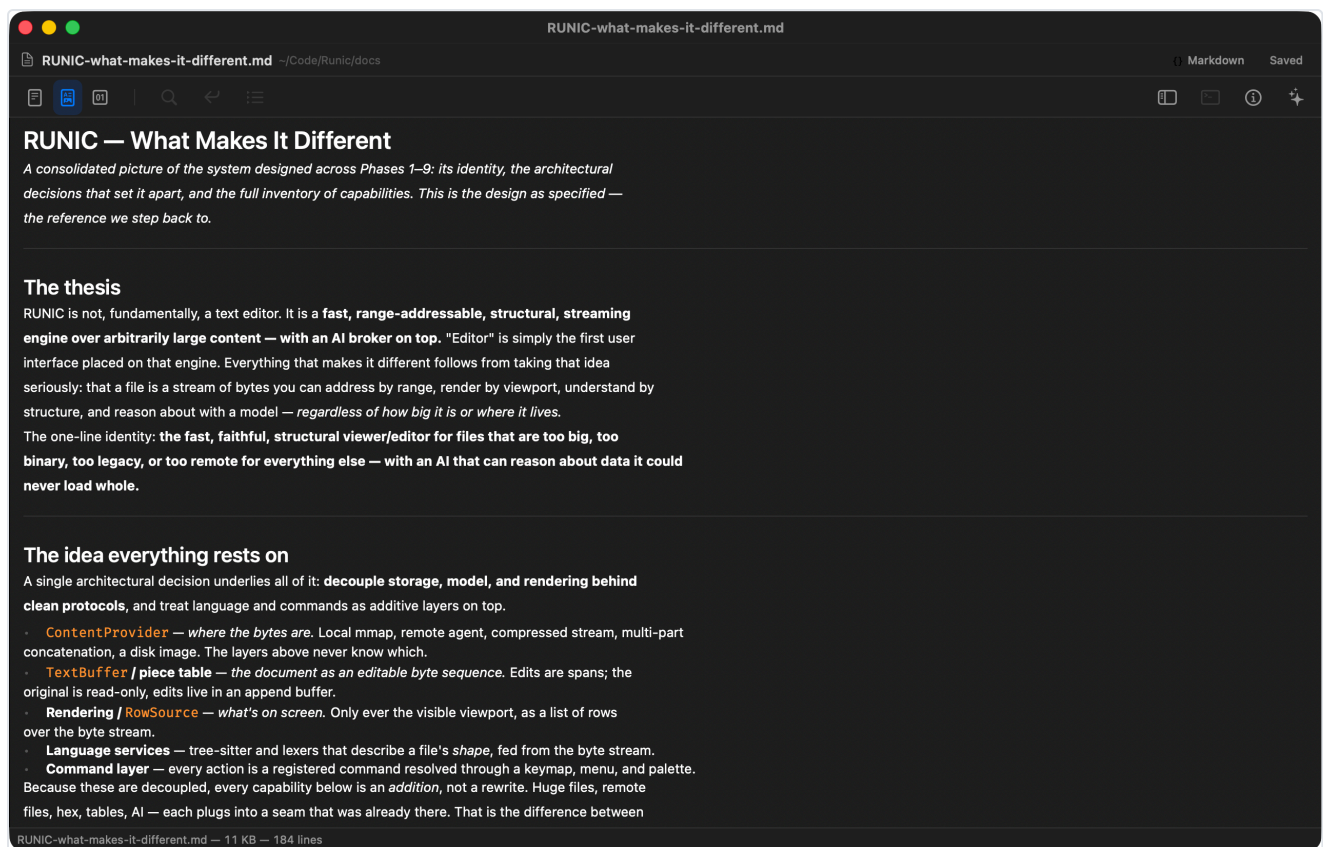


The **find bar** searching a two-million-line log. The match count is live — 82,788 hits — and search runs over the byte stream, so it does not wait for the whole file to be indexed.

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 <code>.md</code> 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.

id	customer	region	status	amount	note
6	Customer 6	apac	refunded	415.84	note, with comma
9	Customer 9	apac	refunded	103.26	note, with comma
10	Customer 10	apac	paid	346.32	note, with comma
15	Customer 15	apac	paid	419.67	note, with comma
17	Customer 17	apac	paid	299.76	note, with comma
18	Customer 18	apac	pending	266.63	note, with comma
19	Customer 19	apac	paid	463.92	note, with comma
29	Customer 29	apac	refunded	130.13	note, with comma
30	Customer 30	apac	pending	220.83	note, with comma
34	Customer 34	apac	paid	7.22	note, with comma
42	Customer 42	apac	pending	12.91	note, with comma
44	Customer 44	apac	refunded	378.16	note, with comma
50	Customer 50	apac	refunded	159.31	note, with comma
54	Customer 54	apac	paid	253.08	note, with comma
59	Customer 59	apac	paid	450.26	note, with comma
63	Customer 63	apac	pending	156.29	note, with comma
64	Customer 64	apac	refunded	256.69	note, with comma
69	Customer 69	apac	refunded	403.66	note, with comma
70	Customer 70	apac	paid	183.69	note, with comma
76	Customer 76	apac	paid	60.46	note, with comma
79	Customer 79	apac	refunded	383.06	note, with comma
82	Customer 82	apac	pending	22.03	note, with comma
85	Customer 85	apac	pending	209.04	note, with comma
88	Customer 88	apac	paid	483.1	note, with comma
89	Customer 89	apac	refunded	84.94	note, with comma
91	Customer 91	apac	pending	50.95	note, with comma
93	Customer 93	apac	paid	409.62	note, with comma
94	Customer 94	apac	pending	248.54	note, with comma
95	Customer 95	apac	paid	399.07	note, with comma
97	Customer 97	apac	paid	168.61	note, with comma
100	Customer 100	apac	pending	288.75	note, with comma
102	Customer 102	apac	refunded	19.57	note, with comma
105	Customer 105	apac	paid	325.92	note, with comma
106	Customer 106	apac	refunded	318.74	note, with comma
109	Customer 109	apac	paid	339.78	note, with comma
122	Customer 122	apac	refunded	323.71	note, with comma
123	Customer 123	apac	pending	469.46	note, with comma
124	Customer 124	apac	pending	209.99	note, with comma
126	Customer 126	apac	paid	444.55	note, with comma

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

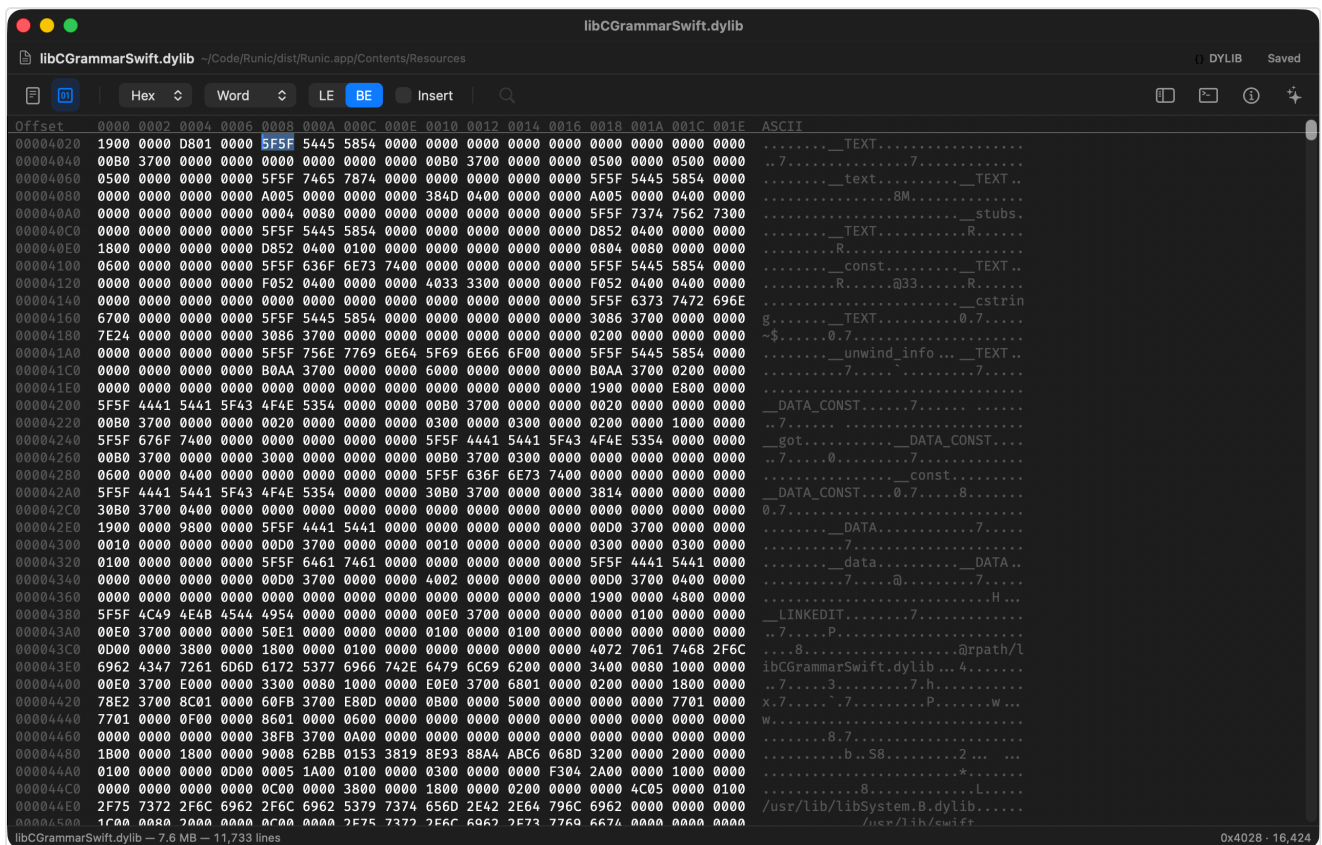
- **Follow (Tail)** sticks to the bottom as the file grows, like `tail -f`
- **Filter Log...** shows only matching lines, streaming as it goes
- **Add Highlight Rule...** colours anything matching a pattern — `ERROR` in red, a request ID in blue — and the rule is saved and applies everywhere

```
live.log /private/tmp/runic-shots
[Follow] Search Clear
2026-08-09T12:07:39.488Z INFO [billing] req=00012170 latency=419ms message=request completed
2026-08-09T12:07:40.350Z INFO [gateway] req=00012171 latency=331ms message=request completed
2026-08-09T12:07:40.918Z INFO [billing] req=00012172 latency=454ms message=request completed
2026-08-09T12:07:41.349Z INFO [billing] req=00012173 latency=126ms message=request completed
2026-08-09T12:07:41.846Z INFO [auth] req=00012174 latency=213ms message=request completed
2026-08-09T12:07:42.474Z INFO [worker] req=00012175 latency=95ms message=request completed
2026-08-09T12:07:42.917Z ERROR [search] req=00012176 latency=158ms message=request completed
2026-08-09T12:07:42.980Z INFO [worker] req=00012177 latency=784ms message=request completed
2026-08-09T12:07:43.865Z WARN [auth] req=00012178 latency=792ms message=request completed
2026-08-09T12:07:44.492Z WARN [search] req=00012179 latency=351ms message=request completed
2026-08-09T12:07:44.725Z INFO [search] req=00012180 latency=396ms message=request completed
2026-08-09T12:07:44.754Z INFO [billing] req=00012181 latency=695ms message=request completed
2026-08-09T12:07:45.505Z ERROR [search] req=00012182 latency=111ms message=request completed
2026-08-09T12:07:45.931Z INFO [auth] req=00012183 latency=178ms message=request completed
2026-08-09T12:07:46.172Z INFO [billing] req=00012184 latency=816ms message=request completed
2026-08-09T12:07:46.315Z INFO [worker] req=00012185 latency=249ms message=request completed
2026-08-09T12:07:47.027Z WARN [search] req=00012186 latency=452ms message=request completed
2026-08-09T12:07:47.489Z INFO [gateway] req=00012187 latency=513ms message=request completed
2026-08-09T12:07:48.195Z INFO [search] req=00012188 latency=777ms message=request completed
2026-08-09T12:07:48.252Z INFO [worker] req=00012189 latency=541ms message=request completed
2026-08-09T12:07:48.769Z INFO [gateway] req=00012190 latency=796ms message=request completed
2026-08-09T12:07:49.459Z INFO [search] req=00012191 latency=516ms message=request completed
2026-08-09T12:07:49.858Z WARN [search] req=00012192 latency=810ms message=request completed
2026-08-09T12:07:50.238Z INFO [gateway] req=00012193 latency=127ms message=request completed
2026-08-09T12:07:50.263Z INFO [gateway] req=00012194 latency=626ms message=request completed
2026-08-09T12:07:50.285Z INFO [search] req=00012195 latency=321ms message=request completed
2026-08-09T12:07:51.100Z INFO [auth] req=00012196 latency=85ms message=request completed
2026-08-09T12:07:51.137Z WARN [gateway] req=00012197 latency=274ms message=request completed
2026-08-09T12:07:51.655Z INFO [billing] req=00012198 latency=413ms message=request completed
2026-08-09T12:07:51.934Z WARN [search] req=00012199 latency=710ms message=request completed
2026-08-09T12:07:52.558Z ERROR [billing] req=00012200 latency=365ms message=request completed
2026-08-09T12:07:52.674Z INFO [search] req=00012201 latency=275ms message=request completed
2026-08-09T12:07:52.755Z INFO [worker] req=00012202 latency=864ms message=request completed
2026-08-09T12:07:52.783Z INFO [search] req=00012203 latency=200ms message=request completed
2026-08-09T12:07:53.507Z INFO [billing] req=00012204 latency=862ms message=request completed
2026-08-09T12:07:54.212Z INFO [search] req=00012205 latency=240ms message=request completed
2026-08-09T12:07:54.585Z INFO [search] req=00012206 latency=4ms message=request completed
2026-08-09T12:07:54.661Z INFO [gateway] req=00012207 latency=844ms message=request completed
2026-08-09T12:07:55.464Z INFO [billing] req=00012208 latency=677ms message=request completed
2026-08-09T12:07:55.925Z WARN [search] req=00012209 latency=487ms message=request completed
2026-08-09T12:07:56.622Z INFO [auth] req=00012210 latency=652ms message=request completed
live.log - 1.1 MB - 12,211 lines Row 12,171
```

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.



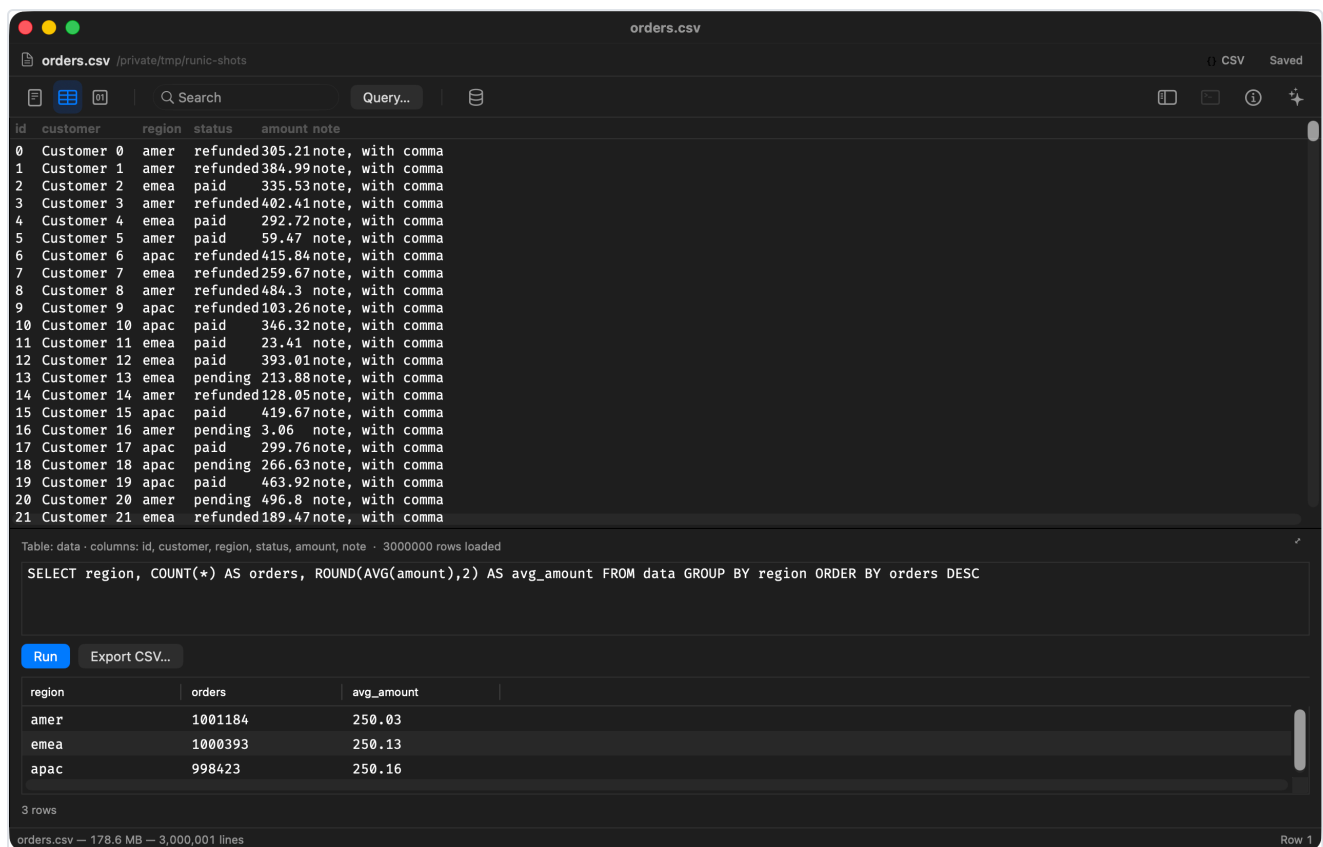
The hex lens on a Mach-O dylib, at the segment table. Offset, bytes, and ASCII side by side; the ribbon carries the radix, grouping, and byte-order controls.

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.



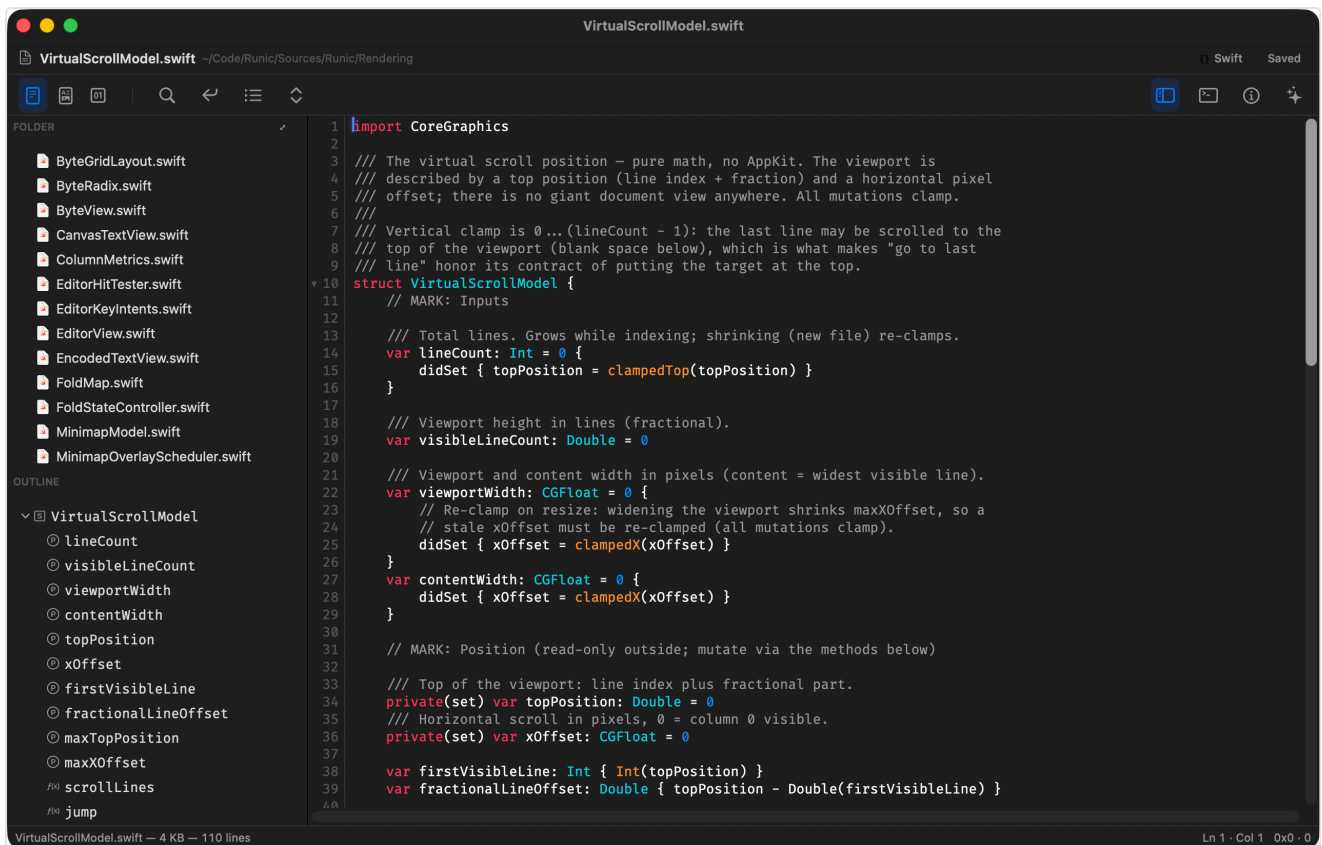
The **SQL panel**, docked below the table, aggregating all three million rows. The file is queried in place — nothing was imported.

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		Folder tree plus an outline of the current file
Assistant		The AI chat panel
Inspector		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.

This screenshot shows a code editor window titled "minified.json" with the file path "/private/tmp/runic-shots". The editor displays a single line of minified JSON: `1 {"items": [{"id": 0, "name": "item 0", "tags": ["a", "b"]}, {"id": 1, "name": "item 1", "tags": ["a", "b"]}, {"id": 2, "name": "item 2", "tags": ["a", "b"]}, {"id": 3, "name": "item 3", "tags": ["a", "b"]}, {"id": 4, "name": "item 4", "tags": ["a", "b"]}]}`. The status bar at the bottom indicates "minified.json — 1.1 MB — 1 lines" and "Ln 1 - Col 1 0x0 - 0".

This screenshot shows the same code editor window after formatting the minified JSON. The JSON is now properly indented and wrapped across 160,004 lines. The status bar at the bottom indicates "minified.json — 1.1 MB — 160,004 lines" and "Ln 1 - Col 1 0x0 - 0". The editor also shows a "Modified" status in the top right corner.

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

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

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