Pauta

UI, UX and platform rules for adl's JavaFX desktop apps. Editora is the reference implementation; nearly every rule names the class or file to copy from.

Starting a new app? Use adl-desktop-app-template, a GitHub template repository built to the edition its README names: click **Use this template**, run `scripts/rename.sh <AppName> <package>`, then add any rule from a later edition.

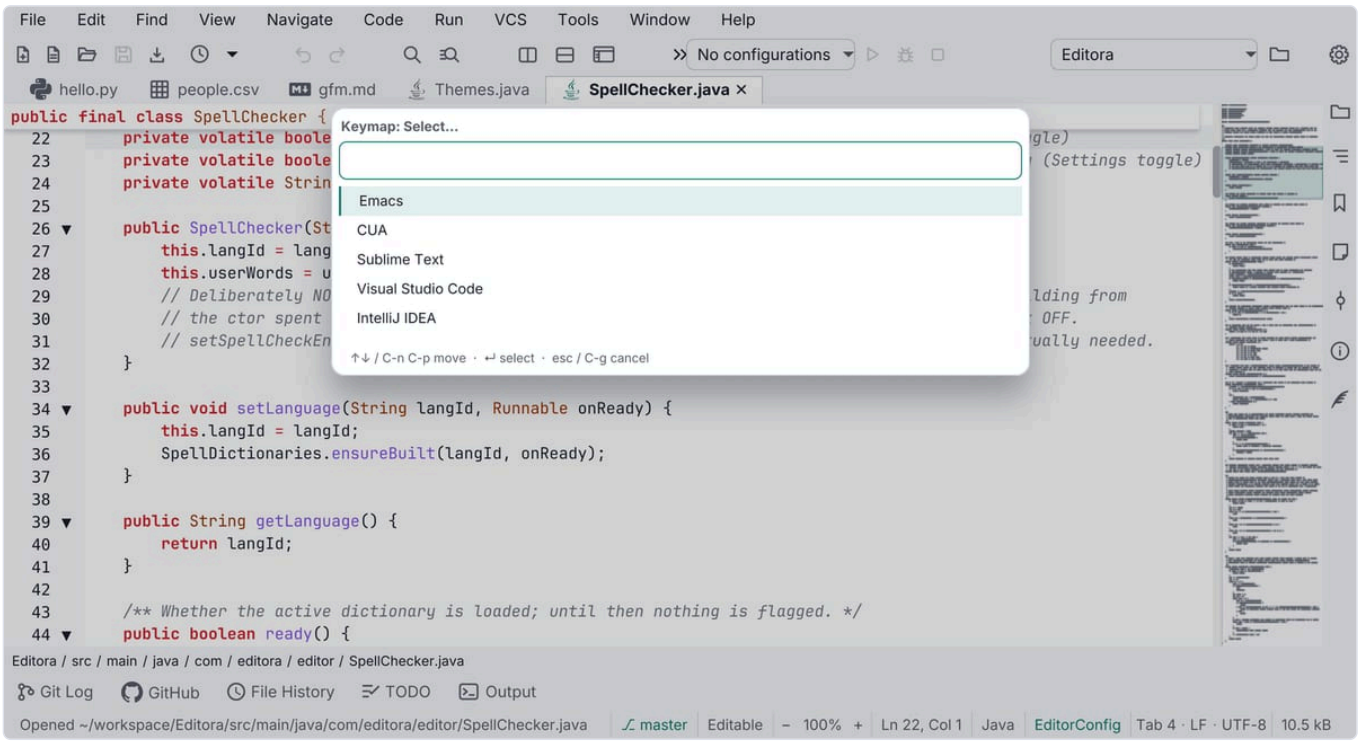
Editora · reference | Torrenta | Termina | Modula | Mirada | nux

`MUST` release blocker for a new app `SHOULD` expected unless the app says why not `MAY` copy when the app grows into it

CONTENTS

Principles
Same on every OS
New app checklist
Conformance
Coding agents and LLMs
Platform baseline
Repository documents
Documentation
Code quality gates
Performance
Keyboard first
Accessibility
Window shell
Icons and color
Toolbars and tooltips
Tabs and tool panels
Menus
Tray and background
Settings
Diagnostics and support
Command line and launch
Internationalization
Command Palette
QuickOpen picker
Welcome page
About window
Pauta window
Confirm dialog
Find bar
Message Log
Settings Sync
Packaging, signing and release

Principles



Editora's keymap picker: five keymaps bind keys to the same commands.

Most of these come from what the six apps already do; where an app still falls short, the principle is the target its rules point to. The first is the reason the family and this guide exist: one app that is the same on Windows, macOS and Linux. The other twelve are how the apps get there and stay there. Editora is the reference; each principle names where else it is written down.

For the people using the apps

1. **Same app on every OS.** A user who moves between Windows, macOS and Linux finds the same features, the same commands and keymaps, the same look and the same installer quality, and never relearns the app. The OS's own way wins only where the app meets the OS itself (the system menu bar, the modifier key, file dialogs, notifications, installers), and Same on every OS lists every such place. Those differences live behind one class, never scattered `if (isMac)` checks. Every other principle serves this one.
2. **Keyboard first.** Every action works without a mouse. The mouse is a shortcut to commands, not the other way round.
3. **Every feature is a command.** One registry of commands with ids. Menus, toolbar buttons, the command palette and key bindings all run commands by id. Nothing is wired straight to logic.
4. **Speed comes first.** Start-up and the app's core action (typing in Editora, stepping to the next image in Mirada, a keystroke reaching the shell in Termina) are the product. When a feature pulls against them, the feature loses or waits until it can be made cheap. *(Mirada's "Priorities" and its "fast first".)*
5. **The UI never waits.** The JavaFX application thread only touches the scene graph. Disk, network, processes, parsing and searching run on other threads and hand their result back; a keystroke, a scroll or a window opening never waits for them. See Performance, which names the only two exceptions: at start-up, before the first window is shown, the wait for the config read and the reading of the bundled resources that window is built from (its fonts, the message catalog, the keymap, its icons); and the last save as the app quits (PERF-01).
6. **Changes apply as they are made.** Settings have no Save button. A change reaches every open window.
7. **Nothing fails silently.** Every failure reaches the status bar, the Message Log or the Debug Log, and the user can turn any of it into a GitHub issue.
8. **Never claim what is not true.** No link that looks like a link and does nothing, no "up to date" before an update check has answered, no setting that seems to apply when it cannot: a setting that waits for a restart says so on its own row. *(Modula's About, Editora's update status, Termina's settings rows.)*
9. **Local first, nothing sent without asking.** No telemetry. Apart from what the user set up (a feed they subscribed to, a transfer they started), the only automatic network call is a once-a-day release check the user can turn off. Problem reports are built locally and submitted by the user.
10. **Users' data outlives versions.** Config files carry a schema version and migrate forward. An older build never overwrites a newer config.

For the people building them

11. **One source of truth for each fact.** The version lives in `pom.xml` and reaches About, `--version`, the update check and problem reports through a filtered `build-info.properties`. A key binding is declared once, so the menu and the accelerator cannot disagree. A list that repeats a fact (About's dependencies, the coverage table in the docs) has a test that fails when the two drift, in both directions. *(Modula's `DependenciesTest`, nux and Torrenta's `AppInfo`, Editora's `BuildHygieneTest`.)*
12. **Separate the decision from the UI, then test it.** Anything with a rule in it (which key does what, sort order, when a bar shows, whether a color is legible) is a small pure method or class with its own unit tests. UI classes stay thin and only wire decisions to controls. *(Termina, Torrenta, Mirada and nux "Conventions"; Editora's pure decision helpers.)* See Code quality gates.

13. **Verify by driving the real app.** UI bugs are visual and none of them throw. Before calling a UI change done, run the real window, press the real keys and look at the result, through a capture hook or a headless screenshot; reasoning about event order is not evidence. *(Termina and Torrenta's `DevCapture`, nux's `--screenshot`, the headless harness behind Editora's website screenshots.)*

Same on every OS

The family exists so that one app can be used the same way on Windows, macOS and Linux (principle 1). This page says where an app may follow the OS's own way and where it may not, and its rules keep other differences out. Each difference is easy to add on its own: an `if (isMac)` that hides a menu row, a JavaFX default that orders buttons one way on Windows and another on macOS. Together they make an app the user has to learn again on each OS.

Where the OS's way wins

Inside the window, Pauta's design wins: the same commands, keymaps, look, dialogs and files on every OS. The OS's own way wins only where the app hands work to the OS or sits in the OS's own interface, because there the user would be fighting the OS rather than learning the app. The table is the whole list. Any other difference between OSes is a bug (OS-04; OS-01 where a command, setting or window is missing): an app records it in `pauta.json` as `pending`, with an issue, until it is fixed (Conformance).

What	Windows	macOS	Linux	Rules
Keymap chords	The keymap as written, with `Ctrl`	The `.mac.json` variant of each keymap that has one (Emacs has none and keeps `Ctrl`): `Cmd` where the other has `Ctrl`, the macOS keymap of the editor it copies, and another chord where macOS keeps one for itself	The keymap as written, with `Ctrl`	KBD-09, KBD-10, KBD-21
How a chord is written	`Ctrl+Shift+P`	`⇧⌘P`, in Apple's modifier order	`Ctrl+Shift+P`	
A click with the primary modifier (open a link, go to a definition)	`Ctrl`+click	`Cmd`+click, as the keymaps put `Cmd` where the others have `Ctrl`	`Ctrl`+click	
Enter on a focused button that is not the default (JavaFX's own behavior on each OS)	Presses the focused button	Presses the default button	Presses the focused button	CONF-13
Quit	File → Quit, the tray's Quit	The same, plus the app menu's and the Dock's Quit, caught as the app's own	File → Quit, the tray's Quit	WIN-14, WIN-13, TRAY-04
Application menu	None	About, Settings and Quit, in the menu macOS names after the app	None	ABOUT-01, WIN-15, WIN-13
Menu bar	In the title bar, by default	The system menu bar	In the title bar, by default	WIN-03
Open and save dialogs	The OS's own (`FileChooser`)	The OS's own	The OS's own	
Opening a file or link, showing it in the file manager	Explorer	Finder (`open`)	`xdg-open`	
Files opened from the OS	Arguments, handed to the running instance	`openFiles` Apple Events, sent to the running app	Arguments, handed to the running instance	CLI-07
File associations	The `.msi` registers the app's file types; a URL scheme goes under `HKCU\Software\Classes`	The bundle's `Info.plist` declares its document types and URL schemes; the user picks the default in Finder	The desktop entry's `MimeType`; `xdg-mime` or the package sets the default	No rule yet
Trash	The Recycle Bin	The Trash, the one Finder empties	The freedesktop.org trash	No rule yet
Tray icon	AWT `SystemTray`	AWT `SystemTray`	StatusNotifierItem	TRAY-01
Notifications	A PowerShell toast, shown as Windows PowerShell's	`osascript`, shown as Script Editor's	`notify-send`, shown as the app's	TRAY-09
Dock or taskbar badge	None	The dock icon	A launcher that honors `LauncherEntry`	TRAY-12
Command line	A console launcher, beside the GUI one	The app's own launcher	The app's own launcher	CLI-11
Middle-click paste	None	None	The X11 primary selection, in a terminal	
The shells and profiles a terminal offers	The OS's own: Command Prompt, PowerShell, Git Bash and each WSL distribution	The OS's own: the shells installed, from `/etc/shells` and the `PATH`	The OS's own: the shells installed, from `/etc/shells` and the `PATH`	
Start at login	The `HKCU` `Run` value	A LaunchAgent	An XDG autostart entry	TRAY-06

What	Windows	macOS	Linux	Rules
Saved secrets	Credential Manager	Keychain	The Secret Service	SET-13
Screen reader	Narrator (UI Automation)	VoiceOver	None: JavaFX has no bridge to the Linux accessibility bus	Accessibility
High contrast	A high-contrast theme while Windows' high-contrast mode is on	None yet	None yet	ACC-09
Installers	`.msi` (unsigned until REL-17 is met), winget on x64	`.dmg`, Homebrew on Apple Silicon	`.deb`, `.rpm`, `.AppImage`, `.tar.gz`; on arm64, `.tar.gz` (and `.deb` where the tooling allows)	REL-11, REL-12, REL-13

Some differences in the apps today are on no row, or miss what a row says, so each breaks a rule until it is fixed:

- Mirada keeps its settings in `~/Library/Application Support` and `%APPDATA%`, not `~/.mirada` (SET-07).
- Torrenta hides its "on metered connections" row on macOS instead of disabling it (OS-01).
- Editora deletes a file for good on Windows, where it reaches no Recycle Bin; Mirada and Torrenta use it (OS-04).
- Editora's "Save as administrator" row is not disabled on Windows, where it does nothing (OS-01).
- Termina removes its `hideMenuBar` command on macOS instead of disabling it (OS-01).
- Termina believes the folder a shell reports for itself on Windows only, so a tab's folder and where a new tab starts can differ by OS for the same shell (OS-04).
- Mirada closes the window on Escape on Linux only, follows the OS's accent color on macOS and Windows only, and puts the title on the left on Windows only (OS-04).
- Editora keeps the user's own key bindings apart by OS, `keybindings` for Windows and Linux and `keybindingsMac` for macOS, so a chord rebound on one is not rebound on the other (OS-04).
- Termina's Settings Sync keeps the user's key bindings in one file for macOS and another for Windows and Linux, so a chord rebound on a Mac is not rebound on a Linux or Windows computer that syncs with it (OS-04).
- Editora runs a folder's tools inside its WSL distribution on Windows only: it registers the WSL targets inside an OS check (OS-04).
- No app fixes the order of a dialog's buttons, so Editora, Torrenta, Termina, Mirada and nux draw them in JavaFX's order for the OS (WIN-11).
- nux raises its Windows notifications as the tray icon's balloon, not a toast, so none shows without the tray (TRAY-09).
- Editora, Torrenta, Modula and nux don't run their quit method from macOS's own Quit: the application menu's and the Dock's Quit ask each window to close, which in Torrenta, Modula and nux, closing to the tray, hides the window (WIN-13).
- Only Mirada puts Settings in the macOS application menu, with a fixed `Cmd+,` where the keymap's chord belongs (WIN-15); only Termina and Mirada add About there (ABOUT-01).
- Mirada has a menu bar on macOS only: on Windows and Linux its menus are one ☰ button (OS-04).
- Editora, Termina, Modula and nux ship no console launcher on Windows, and none attaches to the parent's console; Torrenta's `torrenta-nox` is one (CLI-11).

The same everywhere

Everything else, and in particular:

- **Commands, settings and windows**, with the same defaults: OS-01. What they do and where they sit: OS-04. A keymap's macOS variant binds the same commands: KBD-21.
- **The look.** Inter on every window (ICON-01), not the OS's UI font; the same AtlantaFX themes (PLAT-05); the same icon on a menu row the app draws, where it has one (MENU-14).
- **Dialogs.** The same buttons in the same order (WIN-11), with the same default button (CONF-13). Space presses the focused button everywhere. What Enter does on a focused button that is not the default follows the OS, a row of Where the OS's way wins: it presses that button on Windows and Linux and the default on macOS, which is JavaFX's own behavior and the reason CONF-13 never leaves the default on a red button.
- **Files.** The config folder is `~/.<app>` on every OS (SET-07), not `%APPDATA%` or `~/Library/Application Support`, so the folder a user backs up, syncs or reads about in the docs is the same one.
- **The command line** (Command line) and the user docs: one set for all three, which names a difference only where this page lists it.

Rules

- **OS-01** `MUST` Every command, setting and window exists on Windows, macOS and Linux, with the same default. Where the OS can't do what one asks, it stays where it is, disabled, and says why; it is never hidden. The only exceptions are the rows of the table under Where the OS's way wins, on the page Same on every OS: the places where the app hands work to the OS or sits in the OS's own interface, such as the macOS application menu and the macOS form of a keymap's chords.
  - *Why:* a user who moves to another OS looks for a feature where they left it. A missing row looks like a different app; a disabled one with a reason explains itself.
  - *Check (code):* no command, menu row or settings row is registered or added inside an OS check, and no default in the settings model depends on the OS, unless a row of the table under Where the OS's way wins gives that difference: the rows of the macOS application menu, a default chord in its macOS form. An OS check is a test whose answer can differ among Windows, macOS and Linux; a gate that holds on all three alike (a command registered only when the app finds its own launcher, whatever the OS) is not one. A row the OS can't serve calls `setDisable` and carries its reason (KBD-04 for a command).
  - *Reference:* Editora `ui/SettingsWindow.java` (the extended-window row, disabled on macOS "rather than hidden, so its note still explains why"). Editora doesn't hold the rule everywhere yet: its `file.saveAsAdmin` command (`ui/WindowCommandRegistrar.java`) is registered with nothing that disables it on Windows, where it can do nothing. Termina removes its `hideMenuBar` command on macOS (`ui/TerminalWindow.java`).

- **OS-02** `MUST` In an app's code, one class answers which OS the app runs on (this guide calls it `Os`, with `isWindows()`, `isMac()` and `isLinux()`; the names are the app's own), and no other class reads the `os.name` property. A class that only shows the OS's name, in About or a problem report, asks that one class for it.
  - *Why:* checks scattered through the code are how apps drift apart on one OS without anyone choosing it. In one class, every difference can be found with one search, and a test can run the app as any OS (OS-03).
  - *Check (code):* `grep -rl '"os.name"' src/main/java` lists one file, the `Os` class, and no other file works out the OS for itself. The Command also lists the files that compare `File.separatorChar` or `File.pathSeparatorChar` with a character (`File.separatorChar == '\\'` for "is this Windows"), the other way Editora does it. A class that only prints the name, in About or a problem report, reads the property too, so it is listed. A grep finds only the spellings it is given: when the Command passes, still read for a test that stands in for the OS some other way, such as a separator compared as a string or an environment variable only one OS sets.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; n=$(grep -rl '"os.name"' src/main/java | wc -l); s=$(grep -rlE "eparatorChar *[=≠]= *'" src/main/java | wc -l); u=$(grep -rlE "\"os[.]name\"|eparatorChar *[=≠]= *'" src/main/java | wc -l); ["$n" -eq 1] && ["$u" -eq 1] || { echo "$n files read os.name (one Os class should):"; grep -rl '"os.name"' src/main/java; echo "$s files compare a file or path separator with a character (only that class may):"; grep -rlE "eparatorChar *[=≠]= *'" src/main/java; false; }`
  - *Reference:* the app template's `Os.java`, the one class that reads `os.name` (`current`, `isWindows`, `isMac`, `isLinux`, and `description` for About and a problem report); the Command passes on it. Each of the six apps reads `os.name` in several classes, Editora in about fifty, and eleven of Editora's classes tell Windows apart by a separator, eight of them without reading `os.name` (`lsp/JavaRuntimes.java`, `dap/DapManager.java`, `dap/InstalledAdapter.java`). Editora's `process/DesktopActions.java` has the `Os` enum to start from.
- **OS-03** `SHOULD` An app's tests can run as any OS: the one class that answers which OS the app runs on (OS-02; this guide calls it `Os`) takes an override, and one test asserts that the command ids, the menu bar's rows and the settings pages are the same for Windows, macOS and Linux.
  - *Why:* CI runs the FX tests on one or two OSes (REL-20); with the override, the comparison runs on every pull request on Linux alone.
  - *Check (test):* a test sets each OS in turn through the override and compares the three lists.
  - *Reference:* the app template's `Os.java` (`override`, and the `exampleapp.os` property) and `SameOnEveryOsFxTest` (`theCommandsTheMenusAndSettingsAreTheSameOnEveryOs`), which starts the app as each OS and compares the command ids, the menu bar's rows and every Settings page's rows. Editora's `KeymapsTest` (`guiKeymapsBindSameCommandSetOnBothPlatforms`) does it for the keymaps.
- **OS-04** `MUST` No behavior or layout of an app differs by OS: what a command, a key or a click does, where a control sits and how a window is drawn are the same on Windows, macOS and Linux. The only exceptions are the rows of the table under Where the OS's way wins, on the page Same on every OS: the places where the app hands work to the OS or sits in the OS's own interface, such as the macOS form of a keymap's chords, the system menu bar, the OS's file dialogs, trash, tray and notifications, and the shells a terminal offers. Whether a command, setting or window exists, and its default, is OS-01.
  - *Why:* a feature can be on every OS and still act or sit differently on one: Escape that closes the window on Linux only, a title on the left on Windows only. Each is something to relearn on the next machine, which is what the family exists to spare its users.
  - *Check (code):* list every place the code tests the OS. They are the callers of the methods of the class OS-02 asks for and, in an app that has no such class yet, every file OS-02's Command prints (the files that read `os.name`, and the files that compare `File.separatorChar` or `File.pathSeparatorChar` with a character), together with the callers of whatever those files offer other classes to ask the OS by: a method (Editora's `KeymapManager.isMac()`, which ten files call and eight of them read no `os.name` themselves) or a constant. `grep -rnE 'isMac|isWindows|isLinux' src/main/java` lists the usual names; read the printed files for the app's own. Read each place. It passes when what it changes is what a row of the table under Where the OS's way wins gives that OS, or is nothing a user can see or do (the path of a binary, a process's arguments, a line ending). It fails when it changes anything else a user can see or do: what a command, a key or a click does, where something sits, how it is drawn or what it says. A place that gives an OS something other than what its row gives is judged by the rule that row links, where that rule covers it: a Windows notification raised as the tray's balloon and not a toast fails TRAY-09, not this rule. It fails this rule when the row links no rule, or the rule it links does not cover the difference (a menu bar on one OS and a menu button on the others).
  - *Reference:* the app template, read through: every place it tests the OS calls its `Os.java`, and each is on a row or changes nothing a user sees: the macOS form of the keymap's chords (`Keymap`, `KeyDispatcher`, `AppServices`, and the pickers' keys that `MainWindow` hands `PickerKeys`), the system menu bar and its application menu (`MainWindow`, `Chrome`, `MacAppMenu`), and a program's path, arguments and `PATH` (`Executables`, `Browser`, `ShellPath`). No other app has been read through for it yet. The differences found so far, each on no row or short of its row: Mirada closes its window on Escape on Linux only (`Command.escapeCloses`), follows the OS's accent color on macOS and Windows only (`Theme.followsAccent`), puts its title on the left on Windows only (`TitleBar.Layout`) and has a menu bar on macOS only (`ui/MacMenus.java`); Editora deletes a file for good on Windows (`io/Trash.java`), keeps the user's key bindings apart by OS (`keybindings` and `keybindingsMac` in `config/Settings.java`) and registers its WSL targets on Windows only (`App`); Termina believes the folder a shell reports for itself on Windows only (`ProcessCwd.reportedDirectoryWins`), and its Settings Sync keeps the user's key bindings in one file for macOS and another for Windows and Linux (`sync/KeybindingsFile.java`). On a row, and passing: a click that takes `Cmd` on macOS and `Ctrl` elsewhere (Editora's `editor/EditorBuffer.java`, Termina's `link/LinkGesture.java`), and the profiles Termina lists on Windows only, Command Prompt, PowerShell, Git Bash and the WSL distributions (`shell/ShellDiscovery.java`).

New app checklist

The rules of this edition, cut down to what a new app meets before its first release: every `MUST`, grouped by area, and the SHOULDs worth doing first. Each group links to the page with the rules and the Editora code to copy, and each line names its rules. Every `MUST` of this edition is on a line; a line marked `SHOULD` holds only SHOULDs. A rule's level is the one on its own page, and `npm run check` fails when a line here disagrees with it.

Start from the template. adl-desktop-app-template is a GitHub template repository: a small runnable app with its tests, CI and release workflows and every repository document, made toward the edition its README names; its Pauta window lists what is left. Click **Use this template**, run `scripts/rename.sh <AppName> <package>`, then add any rule on this list from a later edition.

Conformance

Conformance · Pauta window · About window

- [ ] The app implements one edition, named by `pauta.version` in `pom.xml`, and states its conformance: Complete, Core or Partial · PAUTA-01
- [ ] A new app's first release is at Core or Complete · PAUTA-03
- [ ] Every `MUST` and `SHOULD` the app does not meet is an exception in `pauta.json`: not applicable, a deviation (SHOULDs only) with a decision record, or pending with an issue in the app's own repository, each with a one-sentence reason · PAUTA-04, PAUTA-05, PAUTA-06, PAUTA-07, PAUTA-08, PAUTA-09
- [ ] A template repository says so in its `pauta.json` and records the rules only an app made from it can meet as `app-only` · PAUTA-16
- [ ] The conformance is computed from the exceptions by one helper, never stored, and a test holds the README to it · PAUTA-02
- [ ] `PautaConformanceTest` checks `pauta.json` against the edition's `rules.json`, and the README's badge and line show the edition and the conformance · PAUTA-10, PAUTA-15
- [ ] Where `pauta.json` has a `checked` list, each entry records a by-hand rule a person checked and found met, and none is stale; an app that records none meets the rule · PAUTA-18
- [ ] `AGENTS.md` has a section on Pauta that names the edition and links its numbered `llms.txt` · PAUTA-17
- [ ] About links the edition, *Built to Pauta x.y.z* or at Partial *Working toward Pauta x.y.z*, read from `pauta.version`, and shows the conformance with a link that opens the Pauta window · ABOUT-05, ABOUT-06, ABOUT-07, ABOUT-08, ABOUT-09
- [ ] A Pauta window, `help.pauta`, in the palette and in the Help menu where the app has one, listing every exception with fixed links · PWIN-01, PWIN-02, PWIN-03, PWIN-04
- [ ] A *Pauta conformance* page in the user docs with the same list · PAUTA-13
- [ ] That page generated from `pauta.json` `SHOULD` · PAUTA-14

Platform

Platform baseline

- [ ] JDK 25 or newer (`release` 25 is the floor, and no enforcer rule refuses a newer JDK, in the main build or in a profile that turns itself on) and JavaFX 27 or newer · PLAT-01, PLAT-03
- [ ] A jlinked runtime in every installer, so users never install Java `SHOULD` · PLAT-02
- [ ] One named JPMS module · PLAT-04
- [ ] Maven Wrapper committed (`mvnw`, `mvnw.cmd`, `.mvn/wrapper/`) · PLAT-08
- [ ] AtlantaFX 3 controls, and app stylesheets that take their colors from its `-color-*` tokens or from tokens the app builds on them · PLAT-05, ICON-08

Same on every OS

Same on every OS

- [ ] Every command, setting and window on Windows, macOS and Linux, with the same defaults; one the OS can't serve is disabled and says why · OS-01
- [ ] One class answers which OS the app runs on, and no other class reads `os.name` · OS-02
- [ ] A test runs as each OS and finds the same commands, menu rows and settings pages `SHOULD` · OS-03
- [ ] Nothing behaves or is laid out differently by OS, except as the table "Where the OS's way wins" says · OS-04

Code quality

Code quality gates

- [ ] Every package `@NullMarked`; NullAway on Error Prone fails the build · PLAT-09
- [ ] Spotless with Palantir Java Format; CI fails an unformatted change in a Format job of its own · PLAT-10
- [ ] JUnit 5 and headless FX tests (`-Dglass.platform=Headless`), every FX test class carrying one JUnit tag (Editora's `fx`) · PLAT-11, QUAL-06
- [ ] JaCoCo check at 0.90 lines and 0.90 methods, which no profile that activates by OS or by JDK version lowers or skips · PLAT-12
- [ ] What decides whether a control is enabled or shown, or where a key or a click leads, lives in pure, unit-tested helpers; UI classes stay thin `SHOULD` · QUAL-05
- [ ] A capture hook or headless screenshot drives the real window for UI checks · QUAL-09

Performance

Performance

- [ ] No I/O, process, parsing or waiting on the JavaFX application thread, but for the wait for the config read and the reading of the bundled resources the first window is built from (its fonts, the message catalog, the keymap, its icons) at start-up, and the last save as the app quits; results return through `Platform.runLater` · PERF-01, PERF-02
- [ ] Background results applied under a generation guard; blocking work kept off the common pool · PERF-03, PERF-05
- [ ] Redraws coalesced to one per pulse; a stream the app shows delivered in batches; work only on what is visible and changed, with a virtualized control for a list the user or the data can grow without a limit the app sets · PERF-06, PERF-07, PERF-08
- [ ] Text-driven work that leaves the FX thread or outlasts a frame waits for typing to pause; filtering an in-memory list is exempt · PERF-23
- [ ] Logs, histories and caches bounded; large inputs guarded; a closed window releases what it held · PERF-11, PERF-12, PERF-13
- [ ] Only the first frame built before it paints; update check and extras deferred · PERF-14, PERF-15
- [ ] Every thread named and a daemon `SHOULD` · PERF-04

Keyboard and commands

Keyboard first · Command Palette · Find bar

- [ ] One command registry: every action is a command with an id, its title and one-line description from the message catalogs · KBD-01, KBD-03
- [ ] One Command Palette in each main window, `M-x` in the Emacs keymap and `Ctrl+Shift+P` in the others or in an app's one key table, an in-scene overlay, as every picker is (never a `Popup`), that lists every command · KBD-05, KBD-07, PAL-01, PAL-02
- [ ] Emacs and CUA keymaps at least, each `.mac.json` variant binding the same commands, with user overrides on top · KBD-09, KBD-21, KBD-11
- [ ] A find bar in the scene wherever there is text to search, the Debug Log window included: searches as you type with a count, Enter and Shift+Enter step, Escape closes it and, over a terminal, never reaches the shell; an app whose main content is a list filters it instead, and so does a log shown as a list · FIND-01, FIND-02, FIND-03, FIND-06, FIND-07
- [ ] One QuickOpen-style picker for every list choice (themes, keymaps, recent items, a setting's values), and in an app with a picker, the picker in the overlay host `SHOULD` · QOPEN-01, QOPEN-02

Accessibility

Accessibility

- [ ] Every control reachable with Tab and operable with Space or Enter, except controls whose command the keyboard reaches another way; every button, toggle, menu button, link, combo box, check box, radio button and text field shows the focus ring when the keyboard focuses it · KBD-15, KBD-16
- [ ] The focus ring 2 px wide at the default zoom, just outside the control, taking no room in the layout `SHOULD` · KBD-22
- [ ] A find-bar or filter control out of the Tab order only where a command does its work, or a key the bar's field handles does · FIND-16
- [ ] Every control a person can reach has an accessible name, rows drawn from graphics included, held by a test · ACC-01
- [ ] Custom controls state their role and state; canvas content is exposed to the screen reader `SHOULD` · ACC-02, ACC-03
- [ ] Animations follow the OS's reduced motion; color is never the only signal `SHOULD` · ACC-04, ACC-05

Window shell

Window shell · Welcome page · Confirm dialog · About window

- [ ] In an app with a menu bar: on Windows and Linux, the menu bar in the title bar (`HeaderBar`) by default, with a setting to fall back to a decorated window; on macOS, the system menu bar · WIN-03
- [ ] A status bar whose echo area shows the last action's feedback, errors in red and warnings in amber, and opens the Message Log on a click · WIN-01
- [ ] Every destructive action asks first, or can be taken back instead; where the app asks, the question names the thing, the button names the action in red, and Escape cancels · CONF-01, CONF-02, CONF-03, CONF-04
- [ ] Every such question has one default button, never the red one: Cancel, or Save where saving is an option, so Enter destroys nothing · CONF-13
- [ ] The buttons of every dialog with two or more buttons, a text prompt and a form included, are worded from the message catalog, never JavaFX's stock OK, Yes, No, Cancel, Apply or Close · CONF-14
- [ ] Dialog buttons in one order on every OS, Cancel to the left of every action · WIN-11
- [ ] A UI zoom that scales the whole chrome, menus included, applied at once and kept across launches, with commands to zoom in, out and back · WIN-12
- [ ] One quit method for every quit entry point the app has: the Quit row, the `app.quit` command and the system's own Quit · WIN-13
- [ ] A Quit in every app: the `app.quit` command and, where the app has a File menu or a main menu, a Quit row in that menu that runs the command · WIN-14
- [ ] On macOS, Settings in the application menu, showing the active keymap's chord `SHOULD` · WIN-15
- [ ] About from the `help.about` command, the Help menu where the app has one (a menu bar's, or a Help submenu of its main menu), and the macOS application menu: version and build, runtime, license, links to the home page, the source code and the releases, and the last update check · ABOUT-01, ABOUT-02
- [ ] A Welcome page when there is nothing to restore, or an empty state that names the ways in with their chords `SHOULD` · WELC-01, WELC-02

Icons and color

Icons and color

- [ ] Inter's four faces bundled, registered before any CSS, and attached by a scene stylesheet to every scene, dialogs and popups included · ICON-01, ICON-02
- [ ] Light and dark themes switched at run time, a theme setting that offers "follow the system", and an unknown theme falling back to the default · ICON-15, ICON-16, ICON-17
- [ ] State colors named once on `.root` from the theme's roles, no fixed color in a component rule, whatever its spelling (a hex, `rgba()`, `white`) · ICON-07, ICON-08
- [ ] Tinted text at 4.5:1 on its fill and on both grounds; `-color-fg-emphasis` on an emphasis fill; one focus ring; a contrast test holds each · ICON-10, ICON-11, ICON-12
- [ ] Where the icons come in two families, each glyph in one family's style class; the same glyph for an action everywhere it is reached, every glyph set credited in `NOTICE` · ICON-19, ICON-20, ICON-24
- [ ] Readable text in `-color-fg-muted` or stronger, held by a contrast test · ICON-09
- [ ] The contrast tests pure: they read the stylesheets without starting the toolkit `SHOULD` · ICON-14

Toolbars and tooltips

Toolbars and tooltips

- [ ] Icon-only buttons built by one helper that sets the tooltip and the accessible name from one string, chord included (`Icons.button`, `Icons.name`) · TBAR-01
- [ ] In an app with a toolbar, its buttons out of the Tab order; every tooltip names the command and its chord from the active keymap · TBAR-02, TBAR-03
- [ ] Every tooltip shown after the same 400 ms delay `SHOULD` · TBAR-14
- [ ] A toolbar never holds the only way to an action · TBAR-08
- [ ] Groups separated by spacing on one chrome band, no hairlines; a toggle wears `:open` while it is on `SHOULD` · TBAR-04, TBAR-06, TBAR-07
- [ ] The main toolbar: its icons overflow and its pinned tail never does, Settings last with Simple mode before it, buttons disabled while their command can't act, a command hides the bar, and the bar keeps its height `SHOULD` · TBAR-15, TBAR-16, TBAR-17, TBAR-18, TBAR-19

Tabs and panels

Tabs and tool panels

- [ ] In an app whose main window holds documents or sessions in tabs the user opens and closes: every tab operation is a command, on every tab of the strip, whatever it shows; a terminal app's tabs have no close unmodified and no close all · TABS-01
- [ ] One tab header carries the title and the states: `•` and amber italic for unsaved, a pin and bold for pinned, muted italic for read-only (`TabHeaders`); in a terminal, the running program as the title and a mark for a notification, bell or activity · TABS-02, TABS-03
- [ ] Bulk close skips pinned tabs, pinning moves a tab to the end of the pinned group, and Reopen Closed Tab brings the last one back (`ClosedTabs`) · TABS-04, TABS-05, TABS-06
- [ ] A `tool.<id>` command shows and hides each tool panel, on `M-1` to `M-0` in the Emacs keymap, and a panel's content is built on first use (`ToolWindowManager`) · TABS-13, PERF-14
- [ ] Each panel's place and size restored on the next launch `SHOULD` · TABS-15

Menus

Menus

- [ ] In an app with a menu bar, its rows are commands: title, chord and any icon from the registry and the active keymap, no `KeyCombination` accelerators · MENU-01, MENU-02
- [ ] Every row of every menu the app draws, right-click menus and drop-downs included, shows its chord when bound · MENU-15
- [ ] The menus the app draws are in Inter 14 px at the default zoom whatever they were opened from, and follow the UI zoom (`.context-menu` sets the size from the zoom's one value) · MENU-16
- [ ] Menu rows 34 px high with a 22 px icon column, the chord at least 18 px after the title, on a card whose shadow's offset is at least its radius `SHOULD` · MENU-18
- [ ] In an app with a right-click menu: every row, tab, editor and read-only console, a log shown as text such as the Debug Log among them, has one with its own actions, each a command; a list that only navigates has none · MENU-05
- [ ] One right-click menu at a time; a press anywhere outside closes it, and so does Escape, returning focus (`ContextMenuDismissal`) · MENU-06, MENU-07, MENU-08
- [ ] Right-click menus open from the Menu key and Shift+F10 at the selected row, or for the focused view where it has no rows (`RowContextMenu`) · MENU-09

Tray and background

Tray and background (for an app that keeps working without its window)

- [ ] The tray is a StatusNotifierItem on Linux, with no AWT fallback, and AWT's `SystemTray` on Windows and macOS, and the app is fully usable when no icon comes up, its status line saying why · TRAY-01, TRAY-02

- [ ] Closing the window hides it only while an icon is up and the setting says so; the tray's Quit runs the app's one quit method and ends the program with the window hidden · TRAY-03, TRAY-04
- [ ] Start at login writes the OS's own entry (XDG autostart, LaunchAgent, `HKCU` Run), and `--minimized` comes up behind the icon, minimized when the tray setting is off, and shown when an icon was asked for and none came `SHOULD` · TRAY-05, TRAY-06

Notifications (for an app that sends desktop notifications, with a tray or without one)

- [ ] Desktop notifications through one notifier, the text passed as data and never as part of a script and the call timed out after ten seconds or less, for what happened while the window was away, one switch per kind of event, most of them off at first · TRAY-09, TRAY-10

Settings

Settings · Settings Sync

- [ ] Settings window from a palette command (`Ctrl+,`), with a page sidebar and search · SET-01, SET-02, SET-03
- [ ] Live apply with no Save button, a setting that waits for a restart saying so on its row, and a palette command for every setting the Settings window shows · SET-04, SET-06
- [ ] Config in `~/.<app>` on Windows, macOS and Linux alike, unless a flag or, after it, the environment variable names another folder; versioned with migrations, a file newer than the build backed up and defaults loaded · SET-07, SET-09, SET-10
- [ ] Config read with suspicion: values clamped on load, an unreadable file kept as a backup and defaults loaded · SET-11
- [ ] A config file written whole or not at all: to a temporary file beside it, then moved into place · SET-22
- [ ] Export and import of the config folder · SET-12
- [ ] A secret the user gave the app, where the app keeps it, never in a config file: in the OS keychain, or in memory for the session where no keychain answers; a credential the app does not keep held in memory only, never written · SET-13, SET-14
- [ ] A secret in the wrong place refused (a sync URL with a password), and an option that weakens security only a config key · SET-15, SET-16
- [ ] Settings Sync in an app with user data worth carrying between computers, through a Git repository the user owns `SHOULD` · SYNC-01, SYNC-02

Diagnostics

Diagnostics and support · Message Log

- [ ] Message Log of the session's echo-area messages · DIAG-01
- [ ] Debug Log with a session file in its own window, and an uncaught exception handler installed at the start of `main` that posts an echo message · DIAG-02, DIAG-03, DIAG-04
- [ ] Daily update check, started at least three seconds after the first frame, a status-bar notice of a newer version, Check for Updates and an off switch · DIAG-05, DIAG-09, DIAG-10
- [ ] Report a Problem builds a report with the version, the OS, the runtime, a settings summary and the recent Debug Log · DIAG-06
- [ ] A problem report redacted before it is shown: home paths, user names, e-mail addresses, credentials, tokens · DIAG-11
- [ ] Report a Problem shows the report editable, copies it and opens a prefilled GitHub issue; nothing is uploaded · DIAG-12
- [ ] A report too long for the issue link (7,500 characters) is pasted from the clipboard · DIAG-13

Command line and languages

Command line · Internationalization

- [ ] `--help`, `--version`, `--config-dir`, `--dev` and `--report`, which writes its file in the current folder · CLI-01, CLI-02, CLI-05
- [ ] An unknown option stops the launch with a message and exit status 2; in an app that opens files named on its command line, `-` and what follows `--` are files; a launcher's own JVM options are skipped · CLI-12
- [ ] `-h` for help, `-v` and `-V` for the version `SHOULD` · CLI-13
- [ ] On Windows, a console launcher (or an attached console) so the flags print · CLI-11
- [ ] Single instance per config folder · CLI-07
- [ ] Six message catalogs, each with every key · INTL-01, INTL-08
- [ ] The language chosen, else the system's, else English; dates, numbers and the catalog's messages keep the system's region · INTL-14

Repository

Repository documents

- [ ] `README.md` as a landing page of at most 250 lines, with the badge row and the hero screenshot · REPO-01
- [ ] `CHANGELOG.md` in Keep a Changelog format · REPO-02
- [ ] MIT `LICENSE`, and `NOTICE` with every third-party license · REPO-03, REPO-04
- [ ] `AGENTS.md`, with a short `CLAUDE.md` pointing to it · REPO-05, DOCS-14
- [ ] `CONTRIBUTING.md`, `docs/` and `branding/` · REPO-06, REPO-07, REPO-10
- [ ] `.github/ISSUE_TEMPLATE/bug_report.yml` · REPO-09

- [ ] Scripts in Java, run by the JDK as source files, or in `sh` or bash `SHOULD` · REPO-11

Documentation

Documentation

- [ ] User docs on the app's website under `/docs/`: Getting started, a page per feature area, the keyboard reference, Settings and an FAQ, and no page in the app's `docs/` with a section that tells the app's user how to install or use something · DOCS-01, DOCS-02
- [ ] The `help.documentation` command opens the user docs, with a Documentation row where the app has a Help menu · DOCS-06
- [ ] Release notes written for users, and in an app with a Welcome page, its *What's New* opens this build's release · DOCS-05
- [ ] `docs/README.md` indexes the contributor guides · DOCS-12
- [ ] Docs change with the code, the website's in a pull request opened with it; when they disagree with the code, the code wins and the docs are fixed, as `AGENTS.md` and `docs/README.md` say · DOCS-16, DOCS-17
- [ ] Where the app has a Help menu, its rows in the family's order, Documentation first after the Command Palette, then the keyboard reference `SHOULD` · DOCS-07
- [ ] Every link from the app into its user docs opens the running version's docs, `/docs/v-<version>/` `SHOULD` · DOCS-03
- [ ] Command and keybinding references generated from the source, with a check that quoted ids and chords exist `SHOULD` · DOCS-04

Packaging and signing

Packaging, signing and release

- [ ] jpackage installers for Linux x64 and arm64, macOS arm64 and x64 and Windows x64; JReleaser creates the GitHub release with every download · REL-01, REL-03, REL-11
- [ ] The version in `pom.xml` is the one source: a `vX.Y.Z` tag releases it · REL-05
- [ ] macOS `.dmg` signed with Developer ID, notarized and stapled · REL-15
- [ ] Linux downloads with `.asc` signatures from the shared release key · REL-16
- [ ] Build provenance attestations for every download · REL-18
- [ ] Windows `.msi` Authenticode, once the certificate is chosen `SHOULD` · REL-17
- [ ] Homebrew cask in `adriandeleon/homebrew-tap` and a winget package `AdrianDeLeon.<App>` `SHOULD` · REL-12, REL-13

CI/CD (GitHub Actions)

CI/CD

- [ ] The CI workflow on every push to the default branch and every pull request: a format check and the build with its tests, both required checks · REL-19
- [ ] The release workflow on a version tag that matches the pom, with a `workflow_dispatch` dry run · REL-21
- [ ] A release cut only from a commit whose tests passed: the release workflow runs them, or its preflight reads CI's result · REL-30
- [ ] Every third-party action pinned to a commit SHA, with its version in a comment · REL-22
- [ ] Dependabot for `github-actions` and `maven`, weekly · REL-23
- [ ] `permissions: contents: read` by default, or none at all; write only in the jobs that need it · REL-24
- [ ] `timeout-minutes` on every job that runs steps · REL-26
- [ ] A `concurrency` group on every pull-request workflow that cancels a superseded pull-request run, and never a run on the default branch · REL-31
- [ ] Test and coverage reports uploaded; coverage totals in the job summary · REL-27
- [ ] Signing skipped without secrets, but a tag without them fails · REL-28
- [ ] A smoke launch of the packaged app on each OS before release · REL-29
- [ ] The tests on all three OSes, daily or on every pull request, and before every release · REL-20

Conformance

An app implements one edition of Pauta, says how fully it conforms (Complete, Core or Partial), and shows both in its About window. The rules it does not meet are not hidden: each one is an **exception**, written down once in the app's `pauta.json`, shown in the app's Pauta window and listed in its user docs. Its conformance is worked out from that list, never typed in by hand, so the claim and the evidence can't drift apart.

No released app does this yet; the app template does: `pauta.json` holds its exceptions, `PautaConformance` computes its conformance and `PautaConformanceTest` checks both. The audit in Coding agents and LLMs produces the first list: each `MUST` and `SHOULD` it reports as `fail` becomes an exception, of the kind that fits. A `limited` result needs none, and neither does a `manual` one (PAUTA-18) or a `not-checked` one, which the audit could not see from the app's checkout: both are reported, and neither blocks a release.

Complete, Core and Partial

Conformance	The app meets...	About says
Complete	every `MUST` of its edition, met or not applicable, and every `SHOULD`, met, not applicable or a recorded deviation	Built to Pauta x.y.z · Pauta conformance: Complete
Core	every `MUST` in the same way; one or more SHOULDs are still pending	Built to Pauta x.y.z · Pauta conformance: Core
Partial	not every `MUST`: one or more MUSTs are still pending	Working toward Pauta x.y.z · Pauta conformance: Partial

"Built to an edition" keeps its meaning: the app meets that edition's MUSTs. Core and Complete are the two ways to be built to it; a Partial app is working toward it and says so. `MAY` rules never count. A deviation, which only a `SHOULD` can have, leaves the conformance as it is but is always on show: About gives the number of exceptions, and the Pauta window lists every one. A rule checked by hand counts once someone has found it unmet and recorded the exception; until a person has checked it, it leaves the conformance as it is (PAUTA-18). So does a rule an audit reports as `not-checked`, because its Check reads the website's repository or a repository setting.

Partial is how an app that already ships says it is on its way: it took up an edition after its first release, or moved to a newer one with new MUSTs. A new app is not released at Partial.

- **PAUTA-01** `MUST` Every app implements Pauta: it names one edition (the `pauta.version` of ABOUT-08) and states its conformance as Complete, Core or Partial: Complete meets every `MUST` and every `SHOULD` (met, not applicable or a recorded deviation), Core every `MUST` with SHOULDs still pending, Partial not yet every `MUST`.
  - *Why:* an app that is in the family answers to these rules; saying which ones, and how far, is the first of them.
  - *Check (code):* `pom.xml` defines `pauta.version` and `src/main/resources/com/<app>/pauta.json` exists.
  - *Command:* `[ -f pom.xml ] || { echo 'not an app root: no pom.xml here'; exit 1; }; { grep -q '<pauta.version>' pom.xml || { echo 'no pauta.version in pom.xml'; false; }; } && { find src/main/resources -path '*/com/*' -name pauta.json | grep . || { echo 'no pauta.json'; false; }; }`
  - *Reference:* The app template's `pom.xml` (`pauta.version`) and `pauta.json`.
- **PAUTA-02** `MUST` An app's conformance is computed, never stored: one pure helper reads the exceptions and returns Partial when any `MUST` is pending, Core when any `SHOULD` is pending, and Complete otherwise. Nothing else enters it: the `checked` list (PAUTA-18) is a record, and a by-hand rule with no entry there leaves the result as it is. About, the Pauta window and the docs page take it from there; the README's badge and line (PAUTA-15), which can't call the helper, are held to it by a test.
  - *Why: One source of truth for each fact.* A conformance typed in by hand stays "Complete" after a pending exception is added.
  - *Check (code):* one helper returns the conformance (the template's `PautaConformance.of`): it is handed the exceptions and nothing of the `checked` list, returns Partial for a pending `MUST`, Core for a pending `SHOULD` and Complete otherwise, and deviation, not-applicable and app-only entries leave its result alone. About and the Pauta window call it, and nothing in the app's code stores a conformance. A test fails when the README's badge or line names another conformance than the helper returns: the statement requires that one. A unit test of the helper that covers all three results is the surest evidence for the rest. That the docs page shows the same conformance is read with the page itself, in the website's repository (PAUTA-13).
  - *Reference:* The app template's `PautaConformance.java` (`of`, which takes the exceptions alone, and `conformance`), tested by `PautaConformanceTest`, whose `theReadmeNamesTheEditionAndTheConformanceAndLinksTheDocsPage` holds the README and whose `deviationsAppOnlyAndNotApplicableEntriesAndMaysLeaveTheConformanceAlone` covers the entries that leave it alone.
- **PAUTA-03** `MUST` A new app, one that has made no release yet, makes its first release at Core or Complete: the New app checklist's MUSTs are release blockers, and recording one as pending does not lift that. An app with a release behind it is not covered: it may be at Partial, and says so (Complete, Core and Partial). A template repository is never released as an app, so this rule is for the app made from it and the template records no entry for it.
  - *Check (by hand):* in an app with no release yet (no numbered section in its `CHANGELOG.md`, no version tag), the About window of the build to be released says Core or Complete. An app that has released, and a template, are outside the rule.
  - *Reference:* no app yet.

Exceptions

An exception is one `MUST` or `SHOULD` the app does not meet as written; MAYs never need one. There are three kinds for an app, and a fourth for a template repository; each one asks for something different.

Kind	Means	Needs
`not-applicable`	The rule reads as applying to every app, and nothing limits it (PAUTA-05), but what it asks for has no use in this one: a Doctor page (DIAG-07) in an app that depends on no outside tool or library	A reason

Kind	Means	Needs
`deviation`	A `SHOULD` only: the app does it differently on purpose, and would make the same choice again	A reason and a decision record
`pending`	The app does not do it yet	A reason and an issue
`app-only`	A template only: the rule needs what each app made from it has of its own, such as its website, its screenshots or its repository settings	A reason

- **PAUTA-04** `MUST` Every `MUST` and `SHOULD` of the edition that the app does not meet has an exception, and every exception names a rule the app does not meet. An unmet rule with no entry is a failed audit, not an exception; an entry for a rule the app now meets is removed in the same change that meets it. A rule checked by hand counts like any other once a person has checked it in the running app: found unmet, it gets an exception; found met, a `checked` entry (PAUTA-18). Until then it has neither, and that is not a failed audit: the audit reports it as `manual`, not recorded.
  - *Why: Never claim what is not true.* The list is the app's claim; it has to be whole and current.
  - *Check (code):* the audit of the app against its edition (Coding agents and LLMs) is this check, so the audit judges this rule last, from its other results: no `MUST` or `SHOULD` is `fail`, and no result reports an entry for a rule that passes. The evidence names the rules that fail. A `manual` result with no `checked` entry is reported as not recorded (PAUTA-18), and a `not-checked` one as out of the audit's reach: neither is a failure of the app, and neither fails this check. An audit of part of the edition judges it from the rules in its scope and says so.
  - *Reference:* The app template's `pauta.json`, the only list of exceptions so far, beside the audit report that shows it whole: `docs/pauta-audit.json` and `docs/pauta-audit.md`, where no `MUST` or `SHOULD` fails and its nine `app-only` entries are the only exceptions.
- **PAUTA-05** `MUST` A rule limited to apps with a feature needs no entry in an app without that feature. A rule is limited when its own text says so ("An app that stores a secret..."), when its page or section opens with an **Applies to** line (the Tray and background page; `appliesTo` in `rules.json`), or when it details a rule that is limited (WELC-03 onward detail the Welcome page of WELC-01). Nothing else limits a rule: a scope is read from one of the three, never inferred from what the app lacks. `not-applicable` is for a rule that reads as applying to every app.
  - *Check (code):* no `not-applicable` entry names a rule whose text, page, section or parent rule already limits it.
  - *Reference:* the app template's `pauta.json` has no `not-applicable` entry, so it meets the rule with nothing to hold; the six apps have no `pauta.json`.
- **PAUTA-06** `MUST` A `deviation` is a choice, not a gap, and only a `SHOULD` can have one: a `SHOULD` is expected unless the app has a stated reason, and the deviation is that reason. It links a decision record in `docs/decisions/` (DOCS-13) that gives the context, what the app does instead, and what it costs. A `MUST` the app does not meet is `pending`, as is any work it has not done yet.
  - *Why:* a deviation keeps the conformance; a deviation from a `MUST`, or missing work called a deviation, would let the app say it is built to an edition it does not meet.
  - *Check (code):* every `deviation` entry names a `SHOULD`, and its `link` opens a file under `docs/decisions/` that names the rule's ID.
  - *Reference:* The app template's `PautaConformanceTest` (`eachKindHasWhatItNeeds`).
- **PAUTA-07** `MUST` A `pending` exception links an open issue in the app's own repository that names the rule's ID and edition, "MENU-15 (Pauta x.y.z)". One issue may hold several rules, one per page for example, if it names each one that way. Closing the issue removes its entries.
  - *Check (code):* every `pending` entry's `link` is an issue URL in the app's own repository; the source settles that much, and a link that is not one fails the rule. That the issue is open and names the entry's rule ID and edition is read from the issue itself (`gh issue view <number>`), which is on GitHub and not in the checkout. An audit that cannot read the issues reports the rule as `not-checked` and says what would settle it. With no `pending` entry there is nothing to read, and the rule is met.
  - *Reference:* The app template's `PautaConformanceTest` (`eachKindHasWhatItNeeds`), which fails a pending link that is not an issue in the app's repository or the template's. The template has no `pending` entry left, so it meets the rule with nothing to read.
- **PAUTA-08** `MUST` Each exception's `reason` is one plain sentence for the person using the app, in English like the changelog: what is different and, for a deviation, what they get instead. No rule text, no class names.
  - *Check (by hand):* read each reason in the Pauta window as a user would. The source settles two cases: a reason that runs to more than one sentence, or carries rule text or a class name, fails the rule, and with no exception there is no reason to read and the rule is met.
  - *Reference:* The app template's `PautaConformanceTest` (`eachKindHasWhatItNeeds`: a reason ends with a period and names no rule ID).
- **PAUTA-16** `MUST` A template repository, one that new apps are made from, sets `"template": true` in its `pauta.json` and records as `app-only` the rules that only an app made from it can meet. `app-only` leaves the conformance as it is, like `not-applicable`, and nothing but a template may use it. Making an app from the template sets `template` to `false`. From then on the app's everyday build passes but lists what it inherited: each `app-only` entry and each `pending` link to another repository. Its release build fails until each one is met, or recorded as `pending` with an issue in the app's own repository. An app that was not made from a template inherits nothing, and the rule asks nothing of it.
  - *Why:* a template can't have the website or screenshots of an app that doesn't exist yet, and an app must not inherit its template's issues as its own. A build that failed the moment the app was made would fail its first step; the release is where it has to be settled (PAUTA-03).
  - *Check (code):* in a template, `pauta.json` has `"template": true` and the script that makes an app from it sets it to `false`. Outside a template, an `app-only` entry or a `pending` link to another repository is what a template left behind: the test of PAUTA-10 lists each one in the everyday build, and the release build runs it with them treated as failures, so no released app keeps an `app-only` entry. An app whose `pauta.json` has no `template` field, no `app-only` entry and no `pending` link to another repository, or that has no `pauta.json`, shows nothing inherited from a template: it is outside the rule, and an audit reports it as `limited`.
  - *Reference:* the app template: its `pauta.json` has `"template": true` and nine `app-only` entries, its `scripts/rename.sh` sets `"template": false`, and its `PautaConformanceTest` holds both (`onlyATemplateRecordsWhatOnlyAnAppCanMeet`) and, outside a template, lists the `app-only` entries and the `pending` links to another repository in every build and fails on them when the `dist` profile sets `pauta.release` (`theAppListsWhatItInherited`; `pom.xml`). No app has been made from it yet.
- **PAUTA-18** `MUST` `pauta.json`'s `checked` list is the record of the rules checked by hand, and this rule governs only the entries an app chooses to record: each one is well formed and not stale. An app that records none, with no `checked` list or an empty one, meets it. An entry is there only because a person checked the rule in the running app and found it met, and it holds the `rule`, the `edition` it was checked against and the `date`, and may hold the app's `version`, for a rule that is checked again with each release (the screen-reader pass of ACC-07): the audit reports it, and nothing expires by it. It says the rule was met when it was checked, not that the app still meets it. A rule found unmet gets an exception instead (PAUTA-04), never both. A by-hand rule with no entry has not been recorded as checked: it needs no exception and no issue, it does not change the computed conformance (PAUTA-02), and it does not fail this rule. The audit reports it as not recorded, so its report still says which by-hand rules nobody has recorded. An entry is stale when its `edition` is older than the rule's `changed` in the edition's `rules.json`, the edition in which the rule's level, scope or statement last changed, and only then: the app's test fails it (PAUTA-10) until it is removed and the rule checked again.
  - *Why:* an audit can only answer `manual` for these rules, and its report is written again each time. Without a record, a rule someone checked last week and a rule nobody ever looked at read the same. The record stays out of the conformance because nothing in the code tells an unchecked rule from a met one: counting it would make an app record as pending what it may well meet.

- *Check (code):* every `checked` entry has a `rule`, an `edition` and a `date`; the rule is one of the edition's whose check is by hand (`check.kind` in `rules.json`); the edition is no older than the rule's `changed` (one newer than `pauta.version`, and a rule listed twice, are for the test of PAUTA-10 to fail); and no rule has an exception as well. With no `checked` list, or an empty one, there is no entry to fail and the rule is met: the audit's `manual` results say which by-hand rules are not recorded, and that is no failure of this rule. That a person looked is the entry's own claim, which the code can't show.
- *Reference:* no app records a by-hand check yet, so each meets the rule with nothing to hold: the app template's `pauta.json` has no `checked` list, though its `PautaConformanceTest` would hold one (`eachCheckedEntryIsWellFormedAndCurrent`), and the six apps have no `pauta.json`.

The file

The exceptions live in one file, bundled with the app so the Pauta window can read it: `src/main/resources/com/<app>/pauta.json`. The by-hand checks someone has done may be recorded beside them (PAUTA-18). The edition stays in `pom.xml` (ABOUT-08). Each exception copies the rule's `level` and `url` from the edition's `rules.json` so the app needs nothing else at run time, and a test keeps the copies honest.

```
{
  "exceptions": [
    {
      "rule": "MENU-15",
      "level": "MUST",
      "url": "https://pauta-project.dev/<edition>/menus/#menu-15",
      "kind": "pending",
      "reason": "Some right-click menus show no chord yet.",
      "link": "https://github.com/adriandeleon/<App>/issues/212"
    },
    {
      "rule": "DIAG-07",
      "level": "SHOULD",
      "url": "https://pauta-project.dev/<edition>/diagnostics/#diag-07",
      "kind": "not-applicable",
      "reason": "<App> depends on no outside tool or library, so it has nothing for a Doctor page to check."
    },
    {
      "rule": "KBD-12",
      "level": "SHOULD",
      "url": "https://pauta-project.dev/<edition>/keyboard/#kbd-12",
      "kind": "deviation",
      "reason": "<App> has no keymap editor: you change a key in keymap.json, which Settings opens for you.",
      "link": "https://github.com/adriandeleon/<App>/blob/master/docs/decisions/0007-no-keymap-editor.md"
    }
  ],
  "checked": [
    { "rule": "SET-02", "edition": "<edition>", "date": "2026-11-02" }
  ]
}
```

- **PAUTA-09** `MUST` An app's exceptions and its record of by-hand checks have one source: `pauta.json`, bundled with the app's resources (`src/main/resources/com/<app>/pauta.json`) and laid out as The file shows. Its `exceptions` list has `rule`, `level`, `url`, `kind` and `reason` on every entry, `link` on every deviation and pending entry, and one entry per rule. Its `checked` list is the app's to keep or not: an app that records no by-hand check leaves it out or empty, and what an entry of it holds is PAUTA-18's. `"template": true` stands beside the two lists only in a template repository (PAUTA-16).
  - *Check (code):* `pauta.json` is among the app's bundled resources, under `src/main/resources/`; each exception has the fields its kind needs and no rule has two. The entries of the `checked` list are PAUTA-18's to judge, and a file with no `checked` list passes.
  - *Reference:* The app template's `pauta.json`, read by `PautaConformanceTest` (`parsesEveryField`, `everyEntryNamesARuleOfTheEditionWithItsLevelAndAddress`). The file has no `checked` list, which the rule allows.
- **PAUTA-10** `MUST` A test reads `pauta.json` against a copy of the edition's `rules.json` in `src/test/resources/pauta-rules.json` and fails when the copy's `edition` is not `pauta.version`, when an entry names a rule the edition does not have, when its `level` or `url` differs from the edition's, when a rule appears twice, when a deviation names a `MUST`, when a deviation or pending entry has no `link`, or when a `checked` entry (PAUTA-18) names a rule that is not checked by hand, a rule listed twice, a rule with an exception, an edition newer than `pauta.version`, or an edition older than the rule's `changed`, which makes the entry stale. Outside a template, it reports each `app-only` entry and each `pending` link that is not an issue in the app's own repository (the one the app's own code names; `AppInfo.GITHUB_REPO` in the template), and fails on them in the release build (PAUTA-16).
  - *Why:* moving to a newer edition changes levels, retires IDs and changes rules a person checked by hand; the test turns each stale entry into a failure.
  - *Check (test):* `PautaConformanceTest`, or the app's equivalent, runs in the build and has a case for each failure the rule lists.
  - *Reference:* The app template's `PautaConformanceTest` and its copy of the edition's rules, `pauta-rules.json`. It has a case for each failure: `theRulesCopyIsTheEditionThePomNames`, `everyEntryNamesARuleOfTheEditionWithItsLevelAndAddress` (an unknown rule, a level or `url` that differs, a rule twice), `eachKindHasWhatItNeeds` (a deviation on a `MUST`, a missing `link`), `eachCheckedEntryIsWellFormedAndCurrent` (the `checked` list) and `theAppListsWhatItInherited` (what a template left, failed when `pauta.release` is set).
- **PAUTA-11** `SHOULD` Moving to a newer edition is one change: replace `pauta-rules.json` with the new edition's, and the app's copy of the audit skill where it keeps one (the app template's is `.claude/skills/pauta-audit/SKILL.md`); remove the `checked` entries the new `rules.json` makes stale (PAUTA-18) and check those rules again; run the audit again and update `pauta.json`; then change `pauta.version`, the README and `AGENTS.md` together.
  - *Why:* each of these names or follows the edition. The test fails a `pauta-rules.json` or a `checked` entry left behind (PAUTA-10); nothing fails a skill copy left behind, which then audits the new rules by the old procedure.
  - *Check (by hand):* the change that raises `pauta.version` also replaces `pauta-rules.json` and the skill copy, where the app keeps one, leaves no `checked` entry older than its rule's `changed`, and touches `pauta.json` where the audit found something new.
  - *Reference:* Move an app to a newer edition.

Where it shows

The same list reaches three readers: the person using the app, the person reading its docs, and whoever reviews the repository.

In the app, the edition and the conformance are in About (ABOUT-05, ABOUT-09) and the exceptions in the Pauta window, which opens from About, the Help menu and the Command Palette (PWIN-01) and lists every one (PWIN-02).

- **PAUTA-13** `MUST` In the user docs: the app's website has a *Pauta conformance* page under `/docs/` (DOCS-02) with the edition, the conformance and every exception, grouped by kind, each with its rule ID linked to the edition's rule, its reason and its link. The Pauta window's link to it is PWIN-07.
  - *Check (code):* the app names a website for its user docs (DOCS-01): an app whose repository and About name none has nowhere to carry the page, and fails the rule from the app's checkout. The rest is in the website's repository (`<app>-website` , DOCS-02): its docs (the files of `src/content/docs/` in editora-website) have the page, and the page gives the edition `pauta.version` names, the conformance the app's exceptions give, and every exception of the app's `pauta.json` under its kind, each with its rule ID linked to the edition's rule, its reason and its link. A page that lists other rules than the file fails. With no `pauta.json` , which is PAUTA-01's and PAUTA-09's failure, there is no list to compare, and the page is judged on being there with the edition. An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* no app yet.
- **PAUTA-14** `SHOULD` The website's *Pauta conformance* page (PAUTA-13) is generated from the app's `pauta.json` , like the Commands page (DOCS-04), so it cannot list a different set from the app.
  - *Check (code):* the app's website repository has a generator that reads the app's `pauta.json` (beside the others in `scripts/` , in editora-website's layout), and its output is committed. An app that names no website has no page to generate: the rule fails with PAUTA-13. An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* no app yet; editora-website's `scripts/GenCommands.java` is the pattern.
- **PAUTA-15** `MUST` An app's README shows its conformance: the README's Pauta badge (README badges) and its "Built to" line (at Partial, "Working toward") name the edition and the conformance, and both link to the website's *Pauta conformance* page (PAUTA-13). The edition is the one `pauta.version` names (ABOUT-08), and the conformance the one the app computes (PAUTA-02).
  - *Check (code):* `README.md` has the badge and the line; each names the edition `pauta.version` names and the conformance the app's exceptions give (Partial with a pending `MUST` , Core with a pending `SHOULD` , else Complete), the line opens "Working toward" at Partial and "Built to" otherwise, and both link the app's conformance page under its website's `/docs/` . A test that asserts it (the template's `theReadmeNamesTheEditionAndTheConformanceAndLinksTheDocsPage`) is the surest evidence, and PAUTA-02 requires one for the conformance.
  - *Reference:* The app template's `README.md` (the badge and the *Working toward* line), held by `PautaConformanceTest` (`theReadmeNamesTheEditionAndTheConformanceAndLinksTheDocsPage`). Termina's and Mirada's READMEs have a badge that names the edition alone and links the edition; Mirada's has a "Built to" line, which names no conformance. The other four READMEs name no edition.
- **PAUTA-17** `MUST` For the app's coding agent: `AGENTS.md` has a section on Pauta (the app template heads it "Pauta") that names the edition `pauta.version` names and links that edition's `llms.txt` and `llms-full.txt` at its numbered address, never `latest/` (Coding agents and LLMs).
  - *Why:* an agent that reads no rules builds without them, and one that reads `latest/` "fixes" MUSTs of a newer edition the app hasn't moved to.
  - *Check (code):* `AGENTS.md` has a section on Pauta, under whatever heading, that names the edition `pauta.version` names and links `https://pauta-project.dev/<edition>/llms.txt` and `https://pauta-project.dev/<edition>/llms-full.txt` with that number; the file links neither at `latest/` nor at another edition. A test that asserts it is the surest evidence.
  - *Reference:* the app template's `AGENTS.md` , Termina's `AGENTS.md` and Mirada's `AGENTS.md` have the section, each under the heading "Pauta". Termina's `AppInfoTest` (`agentsAndReadmeNameTheSameEdition`) asserts the edition and the `llms.txt` link; the template's `EditionTest` asserts the edition (`theReadmeAndAgentsNameTheSameEdition`) and both links at that edition's address, with no `latest/` and no other edition (`agentsLinksTheEditionsRulesForCodingAgents`). Mirada's `EditionTest` (`theReadmeAndAgentsNameTheSameEdition`) asserts the edition the file names and not its links. Editora's and nux's `AGENTS.md` have no such section, and Torrenta and Modula have no `AGENTS.md` .

Coding agents and LLMs

Pauta has two readers: the developer, and the coding agent working beside them. The pages are written for the developer. Every edition also serves the same rules as plain text and as JSON, so an agent (Claude Code, Codex, Copilot, or any LLM that can fetch a URL) reads the exact rules of the edition the app is built to, instead of what it remembers of some other one.

What each edition serves

File	What it holds	Read it when
`llms.txt`	The index, in the llms.txt format: what Pauta is, how to start an app, build a feature and audit an app, and every page with its rule count and prefix.	First. It is short.
`llms-full.txt`	Every rule of the edition as Markdown, grouped by page and section, with every link made absolute. No prose, screenshots or mockups.	Building or auditing: the whole rule set in one read.
`rules.json`	Every rule as data, with the levels, the three kinds of check and the audit statuses explained, every page, and every retired ID.	Scripts, filtering by page or level, comparing editions.
`<page>.md`	One page as Markdown: its prose, rules and spec tables, each mockup as a sentence and its numbered parts as a list. `keyboard.md` beside `keyboard/`, `templates/about.md` beside `templates/about/`.	Building a feature: the page it belongs to, in one small read.
`changelog.md`	The Changelog page as Markdown: how editions are numbered and what each one changed.	Moving an app to a newer edition.
`skills/pauta-audit/SKILL.md`	The audit procedure as an agent skill: how to judge each rule, read `pauta.json` and write the report.	Auditing an app.
`audit.schema.json`, `rules.schema.json`	The JSON Schema of an audit report and of `rules.json`.	Writing or checking a report; reading `rules.json` in code.
The page itself	The prose, screenshots, mockups and spec tables around the rules.	When the picture matters.

Each one lives under its edition: `https://pauta-project.dev/4.3.1/llms.txt`, `.../4.3.1/rules.json`, `.../4.3.1/keyboard.md`. `latest/` always holds the newest edition, and the site root holds a copy of the latest `llms.txt`, `llms-full.txt` and `rules.json`, where agents look first. Every link inside these files points at the numbered edition, so a rule an agent cites keeps its address. `llms.txt`, `llms-full.txt` and `rules.json` are served from 2.0.0 on, the page Markdown, the skill and the schemas from 4.3.1, and `changelog.md` from the edition after 4.5.0; the 1.x editions, from before rules had IDs, have their pages only. `versions.json` at the site root lists every published edition.

The single Markdown file is the whole guide in one file. It is large. Its pages and rules are the site's, and each mockup reads as it does in `<page>.md`: a sentence saying what it shows.

What a rule looks like

Every rule is one testable statement with a permanent ID. In `llms-full.txt` it is the same list item as on its page: the ID and level in bold, the statement, then *Why*, *Check*, sometimes *Command*, and *Reference* lines. In `rules.json` the same rule is:


```
{
  "id": "KBD-07",
  "level": "MUST",
  "changed": "x.y.z",
  "page": "keyboard",
  "pageTitle": "Keyboard first",
  "section": "Command Palette",
  "appliesTo": null,
  "url": "https://pauta-project.dev/<edition>/keyboard/#kbd-07",
  "text": "The palette and every picker (a card that lists choices to filter and pick one from: [QOPEN-01](https://pauta-projec
  "why": "on Windows a Popup does not reliably take OS keyboard focus, and the whole app can stop receiving keystrokes.",
  "check": {
    "kind": "code",
    "text": "the palette and every picker add their node to the root `StackPane`; none creates a `Popup`.",
    "command": null
  },
  "reference": "Editora [`ui/OverlayHost.java`](https://github.com/adriandeleon/Editora/blob/f314ca3b117829d25dbaa63be6ba2353fd
}
```


- `id` is permanent. It is never renumbered or reused; a removed rule leaves a gap. The prefix names the page (`KBD` is Keyboard first), and `llms.txt` lists every page's prefix. `rules.json` ends with `retired`, every ID no longer in use: the `edition` that removed it and `see`, the rule that holds its requirement now (with its `seeUrl`), or `null`. A `pauta.json` entry for a retired ID is stale; move it to the `see` rule, if the app still misses it.
- `level` is `MUST` (a release blocker for a new app), `SHOULD` (expected unless the app has a stated reason) or `MAY` (worth copying when the app grows into it).

- `check.kind` says how the rule is verified: `code` by reading or searching the source, `test` by a named test the build runs, `by hand` by a person, in the running app or where the check sends them (DOCS-16 reads the repository's pull requests), which source alone can't settle. A rule has a `test` check only when its statement requires the test ("A test holds it.", or a rule that is itself about a test). A `code` check may end "A test that asserts it is the surest evidence": that test is evidence, and an app without it still meets the rule by the behavior its code shows. A check that reads what the app's checkout does not hold, the website's repository or a repository setting, says so in its last sentence (`not-checked`).
- `check.command` , on some `code` checks, is one POSIX shell command (Linux, macOS, Git Bash or WSL): run it with `sh -c` from the app's repository root. Exit 0: the check passes; still read the rest of the rule and cite the file:line that meets it. Any other exit: the check fails, and what it printed is the evidence (OS-02's lists the classes that read `os.name`). Every command only reads, and opens with a guard that fails it outside an app's root. It is `null` where reading the code takes judgment. A search written inside `check.text` (the `grep` of PERF-01, say) is not a command in this sense: it lists candidate lines for a person or an agent to judge, and its exit code is no verdict.
- `reference` names the Editora file, or the other app, that does it. The agent copies from there. From the first edition released after 4.5.0, its links point at the commit each app was at when the edition came out, so they show the code as it was then; `blob/master` in a page's source becomes that commit in what the edition serves. Editions up to 4.5.0 link the apps' default branches.
- **Names in a rule.** Some names in a rule's `text` are required as spelled, because a user, the docs or another tool finds the thing by that name: a command id, a catalog key, a chord, a command-line flag, an environment variable, a property in `pom.xml` or `pauta.json` , and a file the repository must carry, whether a document at its root or a resource or tool configuration the rule names. For example `help.about` , `palette.show` , `--config-dir` , `pauta.version` , `AGENTS.md` , `emacs.json` , `messages_<code>.properties` and `jreleaser.yml` . A platform API the rule names (`javafx.stage.Popup` , `SystemTray` , `Platform.setImplicitExit(false)` , `Future.get()`) is the requirement itself: the JDK and JavaFX spell it one way for every app. Any other name is how Editora spells it, and the app's equivalent meets the rule: a class, method or constant of the app's own code, a CSS file or style class, a branch or workflow file, a file the app writes as it runs (its config files).
- `why` is `null` when the rule has no reason worth stating.
- `appliesTo` is `null` for a rule that covers every app, unless its own text says otherwise. Otherwise it is the **Applies to** line its page or section opens with: "an app whose work goes on without its window" for the `TRAY` rules, and "every app that sends desktop notifications..." for TRAY-09 to TRAY-11. `pages` lists every page with its prefix, its Markdown address and its `appliesTo` .
- `changed` is the edition in which the rule's level, scope or statement last changed, as a version (`"5.0.0"`). A `checked` entry in an app's `pauta.json` whose `edition` is older than it is stale (PAUTA-18). A rule unchanged since it was added has the edition that added it, 2.0.0 at the earliest; a rule's Why, Check, Command and Reference do not move it. Editions up to 5.0.0 don't have the field. In `next/` , the batch that is not an edition yet, a rule new or changed since the last edition has `"next"` .
- `statuses` are what an audit reports for a rule (Audit an app), and `audit` is the address of the report's schema.

Point the app's agent at its edition

An app is built to one edition, the one `pauta.version` in its `pom.xml` names (see About window). Its agent should read that edition, not `latest/` : a newer edition may hold MUSTs the app doesn't meet yet, and an agent that reads them will "fix" what nobody asked it to.

Say so in a section of the app's `AGENTS.md` (PAUTA-17). The app template's `AGENTS.md` has one, headed "Pauta"; an app made before it adds this:

```
## Pauta

This app is built to Pauta <edition>. Its rules: https://pauta-project.dev/<edition>/llms.txt
(every rule in one file: https://pauta-project.dev/<edition>/llms-full.txt).

- Before building a feature, read the rules on its page and copy from the code each rule's Reference names.
- Cite a rule by its ID and edition: "KBD-07 (Pauta <edition>)".
- Read the numbered edition, never latest/. Moving to a newer edition is its own change: pauta.version in
  pom.xml, this file and the README change together.
```

Use the same number everywhere the app names its edition: `pom.xml` , `AGENTS.md` , the README and the About window. An app whose conformance is Partial writes `is working toward Pauta <edition>` instead of `is built to` , as its README and About do (Conformance).

Build a feature to the rules

Give the agent the rules of the page the feature belongs to, the page itself when it has a mockup or a spec table, and the reference code. For example:

```
Add a find bar to the editor. Read the Find bar rules (FIND-*) in
https://pauta-project.dev/<edition>/llms-full.txt and the page
https://pauta-project.dev/<edition>/templates/find-bar/ for its layout. Copy from the
Editora files each rule's Reference names. Meet every MUST; list the SHOULDs you
skipped and why.
```

- **Template pages** keep their mockups and spec tables on the page; `llms-full.txt` holds only their rules. Give the agent the page's address too.
- **Reference code** is in Editora, or the app the reference names. An agent with Editora checked out beside the app reads it faster and more reliably than over the web.
- **When the rule and Editora disagree,** the code is right until someone checks. Ask the agent to report the difference instead of picking a side, and fix the guide or the app in the same change.

Audit an app

The procedure is the **pauta-audit skill**, which every edition serves at `skills/pauta-audit/SKILL.md`. Copy it into the app's `.claude/skills/pauta-audit/` (the app template ships it) or point the agent at it. Use the copy of the edition the app is built to (the newest edition's for an app on an edition before 4.3.1, which serves none), and replace the app's copy when the app moves to a newer edition (PAUTA-11). The skill decides how to audit and how to write the report, and `rules.json` supplies the rules. `llms.txt` and `llms-full.txt` carry the procedure in short; where an edition's short version differs from the skill, the skill wins:

1. Find the edition the app implements: `pauta.version` in its `pom.xml`, or failing that the "Built to Pauta" line of its README. Read that edition's rules. If the app names none, ask which edition to audit against and offer the newest; with no one to ask, use the newest and say so in the report. A 1.x edition has no `rules.json`: audit against 2.0.0 or newer, and report what the app's own edition didn't ask for as upgrade work.
2. Take each rule in turn. A *code* check: run its `check.command` when it has one, else read or search the source as it says. A *test* check: find the named test, or the app's equivalent, and confirm the build runs it. A *by hand* check: the source can't confirm it, so report it as manual, unless `pauta.json` lists the rule (step 4) or the source plainly shows it unmet, which is a `fail` like any other. A check that needs the website's repository or a repository setting, which the app's checkout does not hold, is `not-checked` when the audit can't reach it. Which names in a rule are required as spelled is under What a rule looks like.
3. A rule applies to every app unless it is limited: its text names the apps it covers ("An app with user documents..."), its page or section opens with an **Applies to** line, or it details a limited rule (PAUTA-05). An app outside a limited rule gets `limited`, and needs no exception. Those three are the only places a scope is written: an audit never infers one from what the app lacks.
4. Read the app's `pauta.json` (Conformance). An unmet rule it lists gets the entry's kind as its status; MAYs need no entry. An unmet `MUST` or `SHOULD` it doesn't list is `fail`; so is an unmet `MAY`, which has no consequence. An entry for a rule that passes or is limited, and a deviation or pending entry with no link, are problems. A by-hand rule it lists as an exception gets that kind too, not `manual`. Its `checked` list records the by-hand rules a person checked in the running app and found met (PAUTA-18): a `manual` result carries its entry, and one with no entry is reported as not recorded, which is not a failure: the list is the app's to keep or not, and PAUTA-18 governs only the entries that are there. An entry older than the rule's `changed` is stale, and a problem.
5. Report one result per rule: its ID, level, status and evidence as `file:line`. A failed `MUST` blocks a release; a `manual` or `not-checked` `MUST` does not, and stays in the report until someone settles it.

Every report uses the same statuses, which `rules.json` also lists:

Status	Means	Needs an exception?
`pass`	Met.	No; an entry for it is stale.
`fail`	Not met, and `pauta.json` doesn't list it. A failed `MUST` blocks a release; a failed `MAY` has no consequence.	A `MUST` or `SHOULD`, yes: this is what the audit is for. A `MAY`, no.
`manual`	Checked by hand: the source leaves it open. A person checks it, in the running app or where the Check sends them (the repository's pull requests, for DOCS-16), and may record it in `pauta.json`'s `checked` list once it is met; the report gives that entry, or says "not recorded".	No, recorded or not. Found unmet by the person who checks, it gets one, and its status is that kind from then on.
`not-checked`	The Check needs something the audit could not read: another repository (the website's), what the repository's host keeps and no file shows (a repository setting, an issue's state), or the rules of the edition the app names, in an audit run against another (ABOUT-07). Not a finding and not a pass; the evidence says what would settle it. It does not block a release.	No. Found unmet by whoever settles it, it gets one.
`pending`, `deviation`, `not-applicable`, `app-only`	Not met, and `pauta.json` lists it as that kind.	It has one.
`limited`	The rule covers apps with a feature this one lacks.	No; an entry for it is wrong.

The skill writes the report twice: `docs/pauta-audit.json`, which follows the edition's `audit.schema.json` so two audits can be compared by a script, and `docs/pauta-audit.md`, the same results as tables for people. A prompt as short as this one is enough:

```
Audit this app against Pauta <edition> with the pauta-audit skill
(https://pauta-project.dev/<edition>/skills/pauta-audit/SKILL.md). Failed MUSTs first.
```

Each result reads like this:

```
KBD-07   MUST    pass     src/main/java/com/example/app/ui/CommandPalette.java:57
TBAR-01  MUST    fail     src/main/java/com/example/app/ui/MainToolbar.java:88 builds an icon button by hand
SET-02   MUST    manual   the Settings window's layout needs the running app; checked 2026-11-02 against <edition>
PWIN-02  MUST    manual   compare the Pauta window with pauta.json in the running app; not recorded
TRAY-01  MUST    limited  no tray: no SystemTray or TrayIcon in src/
PAUTA-13 MUST    not-checked  needs the website's repository: its docs have a Pauta conformance page
DIAG-07  SHOULD  not-applicable  pauta.json: the app depends on no outside tool, so a Doctor has nothing to check
MENU-15  MUST    pending  pauta.json: issue #212
```

- **Split a large audit.** An edition has hundreds of rules. Give each page, or each prefix, its own pass or its own agent, and start with the MUSTs (`jq '.rules[] | select(.level == "MUST")' rules.json`). Each page's Markdown (`<page>.md`) is a small read for one pass.
- **Check the evidence.** A verdict without `file:line` is a guess. Open a sample of the passes as well as the failures before you act on the report.
- **Manual means a person.** A *by hand* check needs the running app, often on a particular OS, or in a few rules a look at the repository's pull requests (DOCS-16). The agent lists them; someone runs them and may record each one met in `pauta.json`'s `checked` list (PAUTA-18), which the next audit reads back. One

with no entry is reported as not recorded: that is not a failure, it needs no exception, and it leaves the conformance as it is. One the person finds unmet gets an exception like any other rule (PAUTA-04). The agent settles one case by itself: a by-hand rule the source plainly shows unmet is a `fail`, with the `file:line`.

- **Not checked means out of reach.** A few checks read the app's website repository (DOCS-01, PAUTA-13) or a repository setting (REL-19). An audit run from the app's checkout alone reports them as `not-checked` and says what would settle each; with the website checked out beside the app, or `gh` signed in, it judges them. The checkout still settles what it holds: an app whose repository and About name no website fails DOCS-01, DOCS-02 and PAUTA-13 outright, with no website to read. And an audit run against an edition other than the one the app names reports ABOUT-07 as `not-checked` when About says *Built to*: only the app's own edition's rules settle it.
- **Nothing to judge is not a guess.** A rule about how a required part behaves (the find bar of a log shown as text, the Pauta window) fails with the rule that requires the part, and its evidence names that rule, so a missing find bar reads as one cause. So does a rule that limits every item of a kind some rule requires: with no keymap file, the rule on a keymap's macOS variant fails with the rule that requires the files. A rule that constrains things no rule requires an app to have (every entry, row or file of a kind) passes in an app that has none. The skill's step 4 has the order.
- **Gaps per app** on this site is the same audit, done by hand for the six apps. An agent's report for one of them should agree with Gaps per app; where it doesn't, one of them is wrong.

Move an app to a newer edition

The version says how much work the move is (how editions are numbered): a newer patch asks nothing of the app, a newer minor asks it to add, a newer major asks it to undo or redo something. The Changelog says what each edition added and which rules became stronger. To see it rule by rule, compare two editions' `rules.json`:

```
old=2.0.0 new=2.0.2
diff <(curl -s https://pauta-project.dev/$old/rules.json | jq -r '.rules[] | "\(.id) \(.level)"' | sort) \
     <(curl -s https://pauta-project.dev/$new/rules.json | jq -r '.rules[] | "\(.id) \(.level)"' | sort)
```

A line only in the new edition is a new rule or a raised level; one only in the old is a retired rule (the new edition's `retired` list says where it went) or a lowered level. A rule whose `changed` is newer than the app's edition has a new level, scope or statement since then. Then, in one change (PAUTA-11):

1. Replace `src/test/resources/pauta-rules.json` with the new edition's, and the app's copy of the audit skill (`.claude/skills/pauta-audit/SKILL.md` in an app made from the template) with the one the new edition serves.
2. Remove the `checked` entries the new `rules.json` makes stale, those whose `edition` is older than their rule's `changed`, and have a person check those rules again (PAUTA-18).
3. Give the agent the list and the new edition's `llms-full.txt`, and have it meet each new `MUST` or record it in `pauta.json`. Run the audit again.
4. Change `pauta.version`, `AGENTS.md` and the README together.

Working on Pauta itself

This repository has its own `AGENTS.md`: the rule syntax, each page's prefix, and how a change becomes an edition. `npm run check` fails on a malformed rule, and `rules.json`, `llms.txt` and `llms-full.txt` are built from the pages, so a rule written on a page reaches agents in the next edition with nothing else to update.

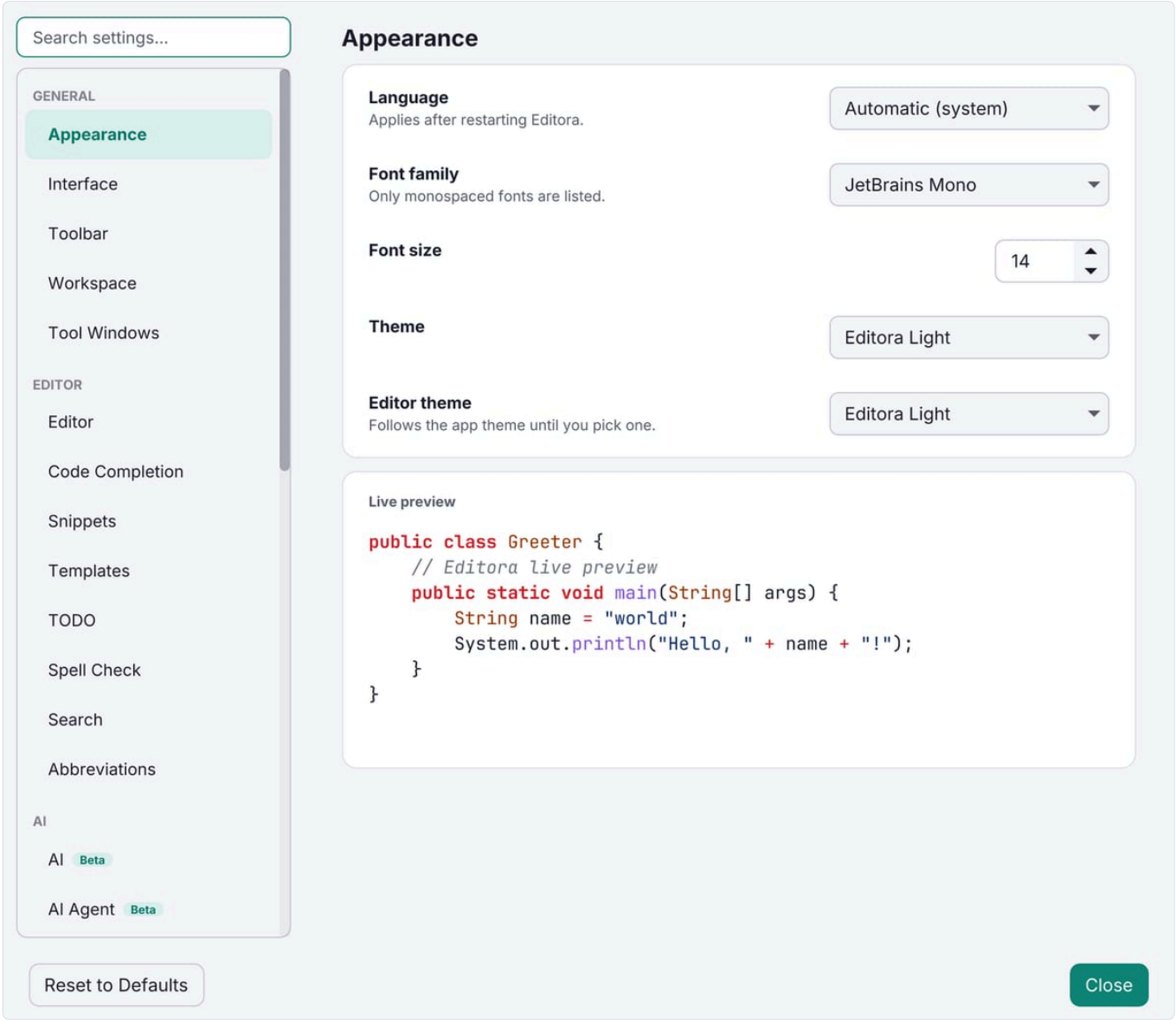
Platform baseline

Every app is built from the same parts: the JDK and JavaFX, one JPMS module, AtlantaFX for the controls, Maven with its wrapper, JSpecify and NullAway for nullness, Spotless for formatting, JUnit on JavaFX's headless platform, JaCoCo for coverage, `java.util.logging` and Jackson. A developer who knows one app's build knows them all, and a fix found in one carries to the others. Each rule names a tool and the version it starts from; how to work with NullAway and the coverage floors day to day is Code quality gates, and how the installers are built, signed and published is Packaging, signing and release. Everyday commands lists the commands that drive the build.

- **PLAT-01** `MUST` The app builds on **JDK 25 or newer**. `maven.compiler.release` 25 is the floor, not the ceiling: a newer JDK builds the app too, so no enforcer rule refuses one, in the main build or in any profile that turns itself on (an `<activation>` by JDK version, OS, property or file, or one that is `activeByDefault`; a profile turned on only by name, with `-P`, is not counted). CI builds every app on JDK 25, the current LTS: every workflow step that sets up a JDK sets up release 25, and the runtime an installer carries (PLAT-02) is that JDK's. A leg that builds a native image carries no JDK: it sets up a GraalVM 25 release instead, 25.0 or any later 25.x. The family moves to the next LTS together, in an edition of Pauta.
  - *Check (code):* `pom.xml` sets `maven.compiler.release` to 25 or later. Every enforcer `requireJavaVersion` rule that counts refuses no JDK from 25 up: its range has no upper bound and starts at 25 or below, as the open-ended `[25,)` does, or a bare `25`, which the enforcer reads as 25 or newer. The rules that count are those of the main build and those inside a profile that turns itself on, one whose `<activation>` holds a `<jdk>`, an `<os>`, a `<property>` or a `<file>`, or an `<activeByDefault>` that is `true`: such a profile is on without being asked for, so its rule is the build's rule wherever it activates. A rule in a profile turned on only by name with `-P` is not counted, and an app with no rule that counts passes. In the workflows, every step that uses a `setup-java` or `setup-graalvm` action (`actions/setup-java`, `graalvm/setup-graalvm`) names 25 or a 25.x release in its `java-version`, and in the `version` a GraalVM step may give beside it or instead. The Command reads the pom as text, with its XML comments left out, and a workflow line by line, not as YAML: a step ends at the next line that opens a list item. A version it cannot read as a number (an expression such as `${{ matrix.java }}`, a `java-version-file`) it prints as not judged and does not fail: read the matrix or the file it names. An enforcer range held in a property (`${…}`) it cannot follow: it prints the property and fails, so read the property's value.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; r=$(sed -n 's:.*<maven.compiler.release>\([0-9]*\)<.*:\1:p' pom.xml | head -n 1); p=$(tr -d ' \t\r\n' < pom.xml | tr '<' '\n' | sed -n '/^!--.*⟶/!p' | sed -n '/^!--/,/⟶/!p'); j=$({ printf '%s\n' "$p" | sed -n '/^profiles>/,/^\/profiles>/!p'; printf '%s\n' "$p" | sed -n '/^profile>/,/^\/profile>/{/^profile>/h;/^profile>/!H;/^\/profile>/{x;/\nactivation>.*\njdk>.*\n\/activation>/{p;d;};/\nactivation>.*\nos>.*\n\/activation>/{p;d} | sed -n '/^requireJavaVersion>/,/^\/requireJavaVersion>/s/^version>//p'); w=$(for f in .github/workflows/*.yml .github/workflows/*.yaml; do [-f "$f"] || continue; for n in $(grep -nE '^[^#]*uses:[^#]*/setup-(java|graalvm)@' "$f" | cut -d: -f1); do c=0; for v in $(tail -n +"$n" "$f" | sed -n '1d;/^ *- /q;p' | grep -E '^ *(java-)?version:' | cut -d: -f2- | cut -d'#' -f1 | tr -d " '\"\r"); do c=1; if printf '%s\n' "$v" | grep -qE '^25([.+-].*)?$'; then :; elif printf '%s\n' "$v" | grep -q '^[0-9]'; then echo "$f:$n sets up release $v"; else echo "not judged: $f:$n version $v" >&2; fi; done; ["$c" -eq 1] || echo "not judged: $f:$n version not given" >&2; done); ["${r:-0}" -ge 25] && ! echo "$j" | grep -vxE '\[(1?[0-9]|2[0-5])([.][0-9]+)*,\)|(1?[0-9]|2[0-5])([.][0-9]+)*' | grep -q . && [-z "$w"] || { echo "maven.compiler.release ${r:-none}; requireJavaVersion in the main build and in profiles that turn themselves on:"; ${j:-none}; [-z "$w"] || echo "$w"; false; }`
  - *Reference:* the app template's `pom.xml` (one enforcer rule, `[25,)` and `.github/workflows/ci.yml` (`java-version: '25'`); Termina's and Mirada's rules have the same range, and Modula and nux have no enforcer rule. Editora doesn't yet: beside the `[25,)` rule of its main build, its `pom.xml` has a profile, `formatter-needs-jdk25`, that activates on JDK `[26,)` and holds an enforcer rule with the range `[25,26)`, so a plain build on JDK 26 or newer stops unless it is told to skip the format check. Torrenta doesn't either: the rule in its main build has the range `[25,26)`. Mirada's profile of the same name skips the format check on those JDKs, says so, and refuses nothing (`pom.xml`). No app's profile that activates by OS, property or file holds an enforcer rule (Termina's `os-windows` and `git-commit` and Modula's `app-image` among them), and no app has a profile that is `activeByDefault`. Every app's workflows set up 25. Mirada's native legs set up Oracle GraalVM with `version: '25.4'`, a GraalVM 25 release (`.github/workflows/release.yml`); its jlink leg sets up JDK 25.
- **PLAT-02** `SHOULD` The installers carry a jlinked runtime; users never install Java.
  - *Check (code):* the `dist` profile's jpackage step builds a jlinked app image (`jLinkOptions`) and wraps the installer around it, and that image is what the release ships.
  - *Reference:* Editora `pom.xml` (`dist` profile). Mirada ships a GraalVM native image instead on every target GraalVM builds one for, all but Intel macOS, which gets the jlinked image: `scripts/native-app-image.sh` takes the Java runtime out of the app image, and users still install nothing.
- **PLAT-03** `MUST` The app builds on **JavaFX 27 or newer**.
  - *Check (code):* `pom.xml` sets `javafx.version` to 27 or later.
  - *Command:* `[ -f pom.xml ] || { echo 'not an app root: no pom.xml here'; exit 1; }; v=$(sed -n 's:.*<javafx.version>\([0-9]*\).*:\1:p' pom.xml | head -n 1); echo "javafx.version ${v:-none}"; [ "${v:-0}" -ge 27 ]`
  - *Reference:* Editora `pom.xml`.
- **PLAT-04** `MUST` The app is one named JPMS module, whose `module-info.java` opens only what reflection, FXML and Jackson need.
  - *Check (code):* `src/main/java/module-info.java` declares one named module, not an `open module`, which opens every package; each `opens` line serves FXML, Jackson or another reflective reader.
  - *Reference:* Editora `src/main/java/module-info.java` (its module is `com.editora`). The name is the app's to choose: the template's is `com.example.app`, and its `scripts/rename.sh` gives the module the name of the app's Java package.
- **PLAT-05** `MUST` Controls use **AtlantaFX 3.x** (`atlantafx-base`). The light and dark themes it carries are ICON-15.
  - *Check (code):* `pom.xml` depends on `atlantafx-base` 3.x.
  - *Reference:* Editora `pom.xml`, `ui/Themes.java`.
- Colors come from AtlantaFX's `-color-*` tokens and the state tokens built on them, never a fixed color in a component rule: ICON-08.
- **PLAT-08** `MUST` The build is **Maven** with the **Maven Wrapper** committed (`mvnw`, `mvnw.cmd`, `.mvn/wrapper/`), so `./mvnw` works on a clean machine.
  - *Check (code):* `mvnw`, `mvnw.cmd` and `.mvn/wrapper/maven-wrapper.properties` are in the repository.
  - *Command:* `[ -f pom.xml ] || { echo 'not an app root: no pom.xml here'; exit 1; }; ! for f in mvnw mvnw.cmd .mvn/wrapper/maven-wrapper.properties; do test -f "$f" || echo "missing $f"; done | grep .`
  - *Reference:* Editora `mvnw`, `.mvn/wrapper/`.
- **PLAT-09** `MUST` Nullness is declared with **JSpecify** and checked by **NullAway** on **Error Prone** at compile time. Every package is `@NullMarked` in its own `package-info.java`.
  - *Check (code):* the compiler arguments include `-Xep:NullAway:ERROR`; every source package has a `package-info.java` with `@NullMarked`.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; [-d src/main/java] || { echo 'no src/main/java'; exit 1; }; { grep -q 'NullAway:ERROR' pom.xml || { echo 'no -Xep:NullAway:ERROR in pom.xml'; false; }; } && ! find src/main/java -name '*.java' ! -name package-info.java ! -name module-info.java -exec dirname {} \; | sort -u | while read -r d; do grep -qs '@NullMarked' "$d/package-info.java" || echo "not @NullMarked: $d"; done | grep .`
  - *Reference:* Editora `pom.xml`, `docs/conventions.md#null-annotations-jspecify-checked-by-nullaway`.

- **PLAT-10** `MUST` Code is formatted by **Spotless** with Palantir Java Format, and CI fails an unformatted change in a Format job of its own: a job that runs `spotless:check` and neither builds nor tests. When CI runs and which of its jobs the default branch requires are REL-19.
  - *Why:* a check bound to a build phase fails in the same job as the tests, so the failed job does not say which of the two is wrong.
  - *Check (code):* the spotless plugin configures `palantirJavaFormat`, and in a workflow that runs on pull requests, which is where REL-19 has CI run and so where an unformatted change is failed, a job runs `spotless:check` and neither builds nor tests: no Maven `compile`, `test-compile`, `test`, `package`, `verify`, `integration-test`, `install` or `deploy` in the same job. The plugin's `check` goal bound to a phase (`./mvnw verify`) does not meet it alone. The Command reads a workflow line by line, not as YAML: a job is what lies between two keys at the indentation of the first key under `jobs:`, and a Maven run counts when its goal is on the line that names `mvn` or `mvnw` (a `name:` line is not a run). A Format job in a workflow of its own that a pull-request workflow calls (`workflow_call`, `uses: ./.github/workflows/<file>`) meets the rule, and the Command follows that call. What it does not see, so read the workflow where one applies: a goal on a continuation line; a build started from a script; a job's `if:`.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; grep -q palantirJavaFormat pom.xml || { echo 'pom.xml configures no palantirJavaFormat'; exit 1; }; for f in .github/workflows/*.yml .github/workflows/*.yaml; do grep -qE '^(*-? *|on:.*[^A-Za-z0-9_-])pull_request(_target)?([^A-Za-z0-9_-]|$)' "$f" 2>/dev/null || { grep -qE '^(*-? *|on:.*[^A-Za-z0-9_-])workflow_call([^A-Za-z0-9_-]|$)' "$f" 2>/dev/null && grep -lE "^[^#]*uses: [^#]*\./$f" .github/workflows/* 2>/dev/null | xargs grep -lE '^(*-? *|on:.*[^A-Za-z0-9_-])pull_request(_target)?([^A-Za-z0-9_-]|$)' 2>/dev/null | grep -q .; } || continue; i=$(sed -n '/^jobs:/,$p' < "$f" | grep -E '^ +[A-Za-z0-9_-]+:' | head -n 1 | sed -n 's/^\(*\).*/\1/p'); for s in $(grep -nE '^[^#]*mvnw?(\.cmd)?[[:space:]]([^#]*[[:space:]])?spotless:check' "$f" | grep -vE '^[0-9]+: *-? *name:' | cut -d: -f1); do a=0; b=999999; for h in $(grep -nE "^$i[A-Za-z0-9_-]+:" "$f" | cut -d: -f1); do if ["$h" -le "$s"]; then a=$h; elif ["$b" -eq 999999]; then b=$h; fi; done; m=; for t in $(grep -nE '^[^#]*mvnw?(\.cmd)?[[:space:]]([^#]*[[:space:]])?(compile|test-compile|test|package|verify|integration-test|install|deploy)([^A-Za-z0-9:_•=/-]|$)' "$f" | grep -vE '^[0-9]+: *-? *name:' | cut -d: -f1); do ["$t" -ge "$a"] && ["$t" -lt "$b"] && m="$m $f:$t"; done; [-z "$m"] && exit 0; echo "spotless:check at $f:$s shares its job with a build or test run:$m"; done; done; w=$(grep -lE '^[^#]*spotless:check' .github/workflows/* 2>/dev/null); echo "no job of its own runs spotless:check in a workflow that runs on pull requests (named in: ${w:-no workflow})"; exit 1`
  - *Reference:* Editora `pom.xml` (spotless plugin), `.github/workflows/ci.yml` (the `format` job, *Format*); the template, Termina and Mirada have the same job in their `ci.yml`. Torrenta and Modula do not yet: their `test` job runs `./mvnw verify`, the phase the check is bound to, and no job runs `spotless:check`. nux has no Spotless.
- **PLAT-11** `MUST` Tests run on **JUnit** (Jupiter, 5 or newer) over JavaFX's built-in **Headless** glass platform (`-Dglass.platform=Headless`), so FX tests need no display, no xvfb and no Monocle. **TestFX** `MAY` drive them; an app that doesn't use it starts the toolkit with `Platform.startup` and runs its steps on the FX thread.
  - *Check (code):* surefire's `systemPropertyVariables` set `glass.platform` to `Headless`; no Monocle dependency, and no xvfb in a step that runs tests (a release leg may use it to train the AOT cache, REL-10).
  - *Reference:* Editora `pom.xml` (surefire), `docs/testing.md`, `FxTestSupport`, with TestFX; Termina `pom.xml` (surefire) and `DialogAnswers`, without it; the app template's `pom.xml` (surefire, and TestFX only to boot the toolkit) and `FxTestSupport`.
- **PLAT-12** `MUST` A **JaCoCo** `check` runs at `mvn verify` with build-wide floors of **≥ 0.90 lines and ≥ 0.90 methods**, and no profile that activates by OS or by JDK version lowers a floor or skips the check. The rule is about the pom's profiles, not about where CI runs the gate: a workflow that runs the tests on some of its runners with JaCoCo skipped is the workflow's choice and passes. The branch floor and each package's own floor are QUAL-03.
  - *Check (code):* an execution of the JaCoCo plugin's `check` goal, whatever its id (Editora's is `jacoco-check`), is bound to `verify` with `BUNDLE` limits of 0.90 `LINE` and 0.90 `METHOD`. Where a limit is a property (`${coverage.bundle.line}` in Termina), its value is 0.90 or more in the main `<properties>` and in every profile that sets it and activates by OS or by JDK version (an `<activation>` with an `<os>` or a `<jdk>`), and no such profile sets `jacoco.skip` or the check's `<skip>`. A `-Djacoco.skip` on a workflow's command line is not judged.
  - *Reference:* Editora `pom.xml` `jacoco-check`, with no profile that touches it; the template's and Mirada's are the same. Editora's `full-tests.yml` runs its tests on the other OSes with `-Djacoco.skip=true`, which the rule leaves to the workflow. Termina doesn't yet: its `os-windows` profile lowers the two floors to 0.45 and 0.47 on Windows (`pom.xml`). Torrenta's check has a line floor and no method floor, and Modula and nux have no JaCoCo check.
- **PLAT-13** `SHOULD` The app logs through `java.util.logging` and no other logging API. The Debug Log captures it: DIAG-02.
  - *Check (code):* the app's own code imports no logging API other than `java.util.logging`.
  - *Command:* `[ -f pom.xml ] || { echo 'not an app root: no pom.xml here'; exit 1; }; [ -d src/main/java ] || { echo 'no src/main/java'; exit 1; }; ! grep -rnE '^import (static )?(org\.slf4j|org\.apache\.logging|org\.apache\.commons\.logging|ch\.qos\.logback|org\.jboss\.logging|org\.tinylog)\.' src/main/java`
  - *Reference:* Editora `ui/DebugLog.java`.
- **PLAT-14** `SHOULD` Config files are JSON, read and written by Jackson as plain classes (POJOs). Their schema versions and migrations are SET-09.
  - *Check (code):* every class written to the config folder is read and written through Jackson's `ObjectMapper`; no store builds or parses its JSON by hand.
  - *Reference:* Editora `config/Settings`, `config/migration/ConfigSchema.java`.

The app's typeface is Inter, bundled and registered before any CSS and attached to every scene: ICON-01 and ICON-02.



Settings → Appearance in Editora: language, fonts, app theme and editor theme, with a live preview.

Everyday commands

```
./mvnw javafx:run                # run the app
./mvnw test                      # unit + headless FX tests, writes the JaCoCo report
./mvnw spotless:apply            # format
./mvnw verify                    # full gate: tests, coverage check, NullAway
./mvnw -Pfatjar package          # runnable fat jar for this OS (Editora and the template only)
./mvnw clean -Pdist package      # app image or native installer for this OS (clean is mandatory)
```

The first four are the same in every repo that has the plugin behind them: nux has no Spotless, JaCoCo or NullAway yet, and Modula no JaCoCo. Only Editora and the template have a `fatjar` profile. `-Pdist package` builds the installer in Editora and Modula. In the template, Termina, Torrenta and Mirada it builds the app image alone, and `-Dskip.installers=false` adds the installers; in nux it builds the app image, and `-Djpackage.type=` names an installer to build instead.

The `clean` in every `-Pdist` build is mandatory: incremental compilation can leave synthetic enum-switch classes out of `target/classes`, and jlink then ships an app whose keyboard fails on the first keypress.

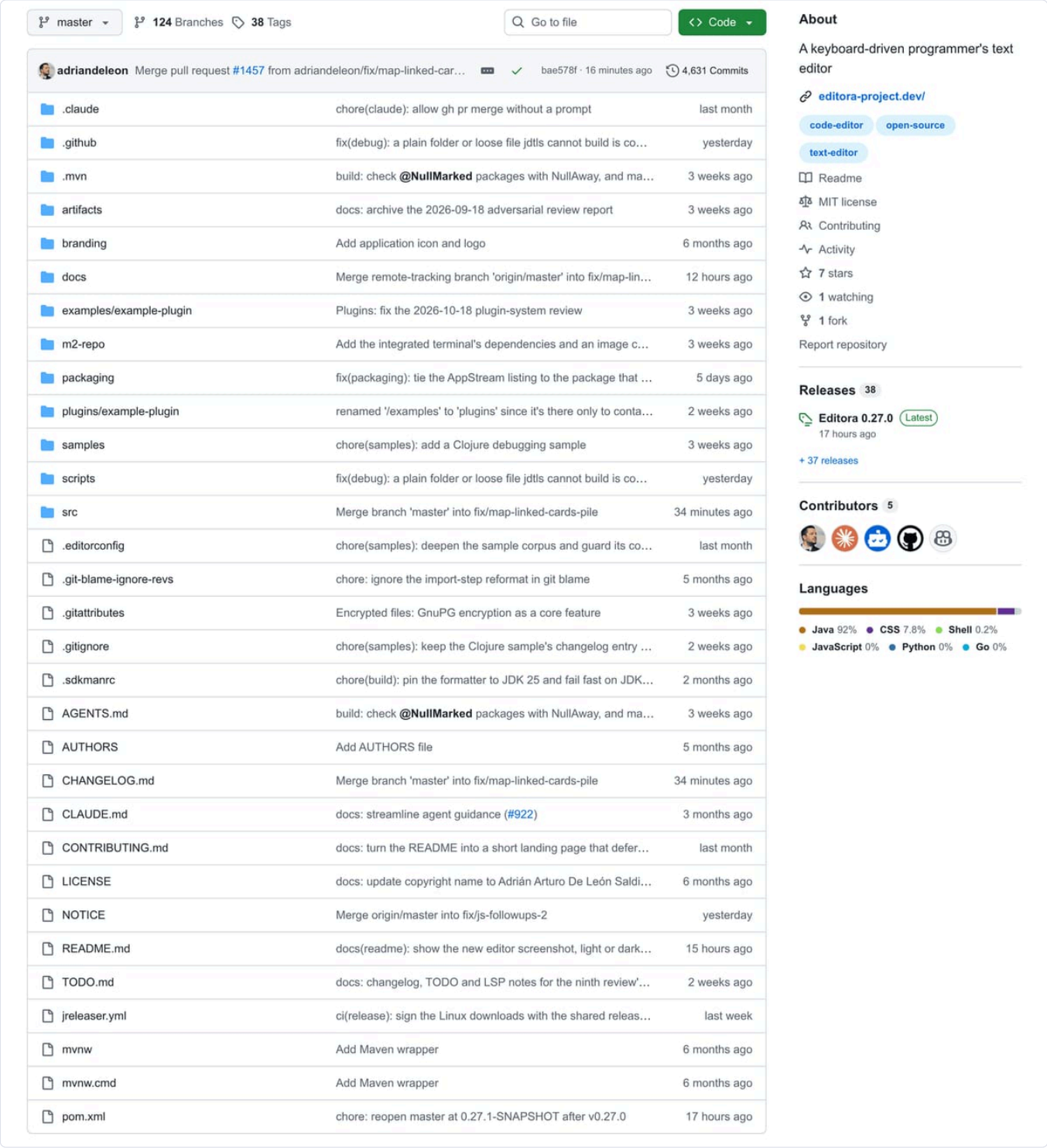
Repository documents

Every repository in the family carries the same documents in the same places: a README that is a landing page, a changelog, the license and its notice, the files that coding agents and contributors read, a `docs/` folder, the bug report form and the source of the icon. Whoever has found their way around one app's repository finds it in any other. This page says which files there are and what each holds; what the documentation as a whole must cover, and for which reader, is Documentation.

Start from the template. adl-desktop-app-template carries an example of every file on this page, filled in for a placeholder app, `ExampleApp`. `scripts/rename.sh <AppName> <package>` renames it to yours.

- **REPO-01** `MUST` `README.md` is a landing page with these parts: logo, badges, a one-paragraph pitch, the hero screenshot, highlights, install, build, command line, config folder, license. Detail lives on the website, which the README links, so the README stays short: at most 250 lines.
  - *Check (code):* `README.md` has each of those parts, links to the website for the rest, and runs to at most 250 lines (`wc -l README.md`).
  - *Reference:* Editora `README.md` (221 lines) and the template's `README.md` (155); Termina's has 152 and Mirada's 172. Three are over the 250: Torrenta's runs to 433 lines, Modula's to 364 and nux's to 313.
- **REPO-02** `MUST` `CHANGELOG.md` follows Keep a Changelog 1.1 and SemVer: an `## [Unreleased]` section with Added / Changed / Fixed / Removed, and each entry opens with a bold one-line summary written for users.
  - *Check (code):* the file's first section is `## [Unreleased]` and the others are headed by a SemVer number and a date (`## [x.y.z] - YYYY-MM-DD`); the entries of a section sit under `### Added` , `### Changed` , `### Fixed` or `### Removed` , whichever it has entries for (Keep a Changelog's other two, `### Deprecated` and `### Security` , are as good); and every entry starts with `- **` .
  - *Reference:* Termina `CHANGELOG.md` and Mirada's, where every entry opens in bold. In Editora's `CHANGELOG.md` 1,804 of 2,173 entries do: the older ones don't, and some sections have headings of its own (`### Performance` , `### Internal` , `### Packaging`), as some of Modula's and nux's do. Torrenta's sections are headed `## Unreleased` and `## 0.4.1 — 2026-10-28` , without the brackets.
- **REPO-03** `MUST` `LICENSE` is MIT, "Copyright (c) 2026 Adrián Arturo De León Saldivar", with the year the app started in: 2026 for the six apps, later for a newer one.
  - *Check (code):* `LICENSE` opens with `MIT License` and has that copyright line, with a year from 2026 on.
  - *Command:* `[ -f pom.xml ] || { echo 'not an app root: no pom.xml here'; exit 1; }; tr -d '\r' < LICENSE | head -n 1 | grep -qx 'MIT License' && tr -d '\r' < LICENSE | grep -qxE 'Copyright \(c\) 20(2[6-9]|[3-9][0-9]) Adrián Arturo De León Saldivar' || { head -n 3 LICENSE; false; }`
  - *Reference:* Editora `LICENSE` .
- **REPO-04** `MUST` `NOTICE` gives the app name, the copyright, "licensed under the MIT License", then every bundled third-party component with its license (Inter is OFL, AtlantaFX is MIT, OpenJFX is GPLv2+CPE...). The installers carry `NOTICE` too.
  - *Check (code):* `NOTICE` opens with the app's name, the copyright and the MIT sentence, and names every runtime dependency in `pom.xml` and every bundled font with its license; and the build puts `NOTICE` into what every installer carries (a `<resource>` that packages it into the application jar in Editora; an argument to `jpackage` or a packaging script does it as well). A test that asserts it is the surest evidence.
  - *Reference:* Editora `NOTICE` and `pom.xml` (the `<resource>` that packages `LICENSE` and `NOTICE` into the application jar); two tests hold it: `NoticeCoverageTest` fails when a runtime dependency in `pom.xml` has no `NOTICE` entry, and `BuildHygieneTest` when the pom no longer packages the two files.
- **REPO-05** `MUST` `AGENTS.md` holds the operational rules for coding agents: essential commands, worktree-per-task, the change checklist. It is not a changelog. A short `CLAUDE.md` points to it.
  - *Check (code):* `AGENTS.md` has sections for commands, worktrees and the change checklist; `CLAUDE.md` is a few lines that send the reader to `AGENTS.md` .
  - *Reference:* Editora `AGENTS.md` , `CLAUDE.md` .
- **REPO-06** `MUST` `CONTRIBUTING.md` says how to build, test, format, and open a PR.
  - *Check (code):* `CONTRIBUTING.md` names the build, test and format commands and the PR step.
  - *Reference:* Editora `CONTRIBUTING.md` .
- **REPO-07** `MUST` `docs/` holds `architecture.md` , `conventions.md` , `testing.md` , `gotchas.md` , `release.md` , `building-and-packaging.md` and `decisions/` (ADRs). The folder's index, `docs/README.md` , is DOCS-12.
  - *Check (code):* each of those paths exists under `docs/` .
  - *Reference:* Editora `docs/` .
- **REPO-08** `MAY` `TODO.md` holds the open list, kept in the same PR as the change.
  - *Check (code):* `TODO.md` exists at the root.
  - *Reference:* Editora `TODO.md` .
- **REPO-09** `MUST` `.github/ISSUE_TEMPLATE/bug_report.yml` is an issue form, and the in-app Report a Problem fills it: the link it opens names the form and carries the environment and the report as query parameters named by the ids of the form's two fields for them (Editora's ids are `environment` and `report`).
  - *Check (code):* the form is at that path and has a field for the environment and a field for the report, each with an `id` ; and the link Report a Problem builds names the form's file and carries one query parameter per field, named by the field's id (`template=bug_report.yml&environment=…&report=…` in Editora). The ids are the app's to choose, as long as the form and the link agree.
  - *Reference:* Editora `.github/ISSUE_TEMPLATE/bug_report.yml` and `report/ProblemReport.java` , which builds the link. The template, Termina, Torrenta and Mirada have the same form, ids and link; Modula and nux have no issue form.
- **REPO-10** `MUST` `branding/` holds the source SVG of the icon. The PNG, ICNS and ICO generated from it sit beside it or in `packaging/` .
  - *Check (code):* `branding/` has an `.svg` ; the `jpackage` icon paths in `pom.xml` point at the generated files.
  - *Reference:* Editora `branding/editora-icon.svg` , with `editora.png` , `.icns` and `.ico` beside it; the template keeps them in `packaging/icons/` .
- **REPO-11** `SHOULD` Every script in `scripts/` is a Java source file the JDK runs as it is (`java scripts/Name.java` , a compact source file or an ordinary class) or an `sh` or bash script.
  - *Why:* every app already needs the JDK, and a Java script runs the same on Windows, macOS and Linux. A Python or Node script makes every contributor and CI runner install a second toolchain for one step. A script that runs on Windows is easier to keep in Java than in bash.
  - *Check (code):* no file under `scripts/` is Python, JavaScript, TypeScript, Ruby, Perl, Groovy, Lua, PHP, PowerShell or a batch file, and every shebang there names `sh` , `bash` or `java` . Data files beside the scripts (fixtures, lists, templates) are not scripts.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; [-d scripts] || exit 0; bad=$(find scripts -type f | while IFS= read -r f; do if printf '%s\n' "$f" | grep -qE '\.(py|js|mjs|cjs|ts|rb|pl|ps1|bat|cmd|groovy|kts|lua|php)$'; then echo "$f"; continue; fi; l=$(head -n 1 "$f" | tr -d '\r'); if printf '%s\n' "$l" | grep -q '^#!'; then printf '%s\n' "$l" | grep -qE '^#! *(/usr)?/bin/(env +(-S +)?)?((ba|da)?sh|java)(|$)' || echo "$f: $l"; fi; done); [-z "$bad"] || { printf '%s\n' "$bad"; false; }`

- *Reference:* Editora `scripts/`, whose Java scripts have their own tests, run by `scripts/lib/RunTests.java` (`docs/testing.md`).

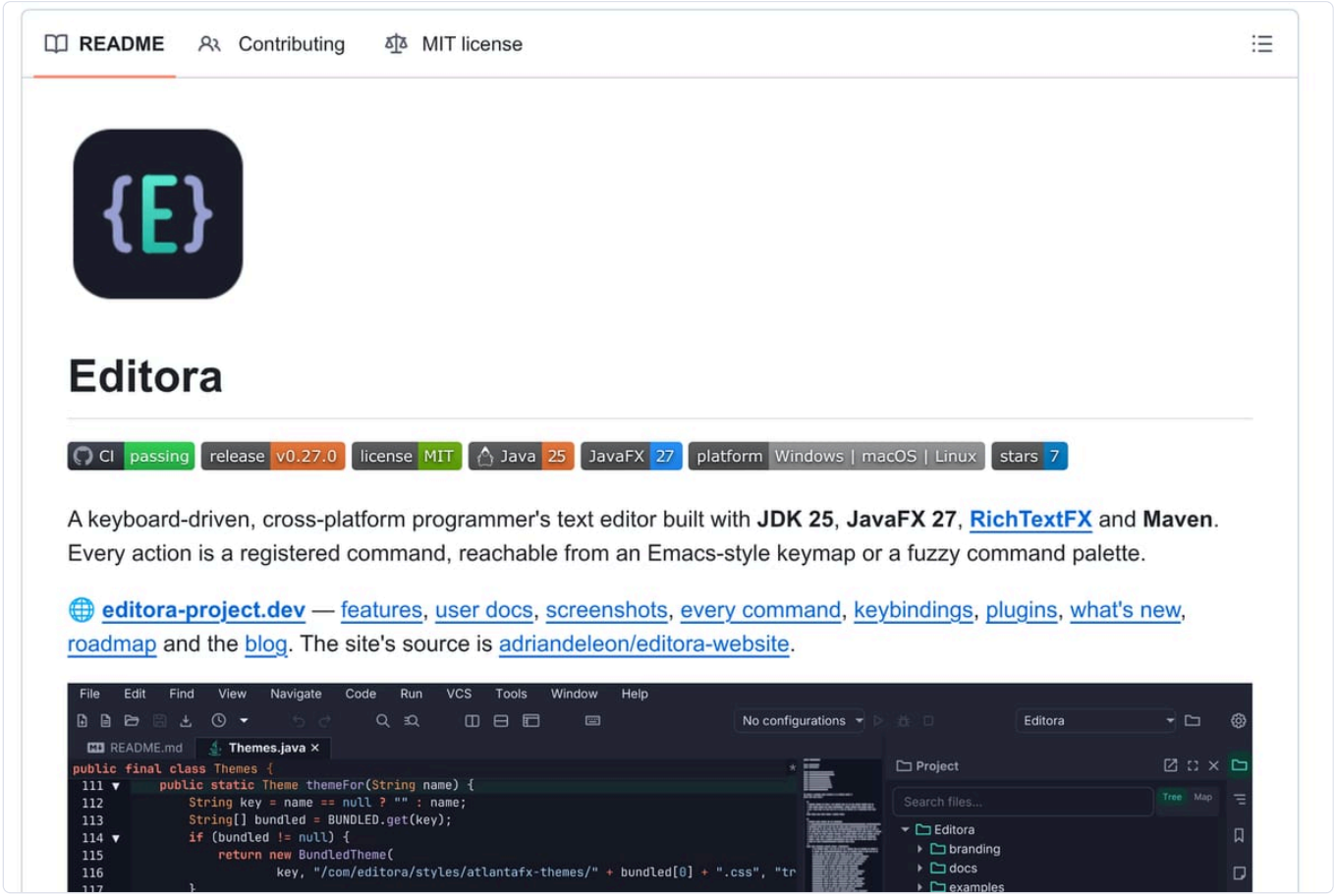


Editora's repository root: every file on this page, with the homepage, Readme, MIT license and Contributing links GitHub derives from them.

README badges (in this order)

The template's README carries this row, and an app made from it starts with it:

```
[![CI](https://github.com/adriandeleon/<App>/actions/workflows/ci.yml/badge.svg)](…/actions/workflows/ci.yml)
[![Release](https://img.shields.io/github/v/release/adriandeleon/<App>)](…/releases/latest)
[![License: MIT](https://img.shields.io/github/license/adriandeleon/<App>)](LICENSE)
![Java](https://img.shields.io/badge/Java-25%2B-orange?logo=openjdk&logoColor=white)
![JavaFX](https://img.shields.io/badge/JavaFX-27%2B-1e90ff)
![Platforms](https://img.shields.io/badge/platform-Windows%20%7C%20macOS%20%7C%20Linux-lightgrey)
[![Pauta](https://img.shields.io/badge/Pauta—<edition>%20%C2%B7%20<conformance>-<color>)](<homepage>/docs/pauta/)
[![Stars](https://img.shields.io/github/stars/adriandeleon/<App>?style=flat)](…/stargazers)
```



The top of Editora's README: logo, seven of the eight badges in this order (it has no Pauta badge yet), a two-sentence pitch, links to the website, then the hero screenshot.

What is required of the row is in two rules: REPO-01 asks for badges, and PAUTA-15 for the Pauta badge among them. The other seven and their order are the template's: a convention, not a rule. Editora's README has those seven in this order, with `Java-25` and `JavaFX-27` where the template writes `25%2B` and `27%2B`, and no Pauta badge yet.

The Pauta badge names the edition and the conformance, in green `2ea043` for Complete, blue `5b8def` for Core and amber `d29922` for Partial, and links to the app's *Pauta conformance* docs page; a test holds it to `pauta.version` and the computed conformance (Conformance). The template's badge reads `Pauta 6.0.0 · Complete`, in green, and the README line beside it says "Built to"; at Partial that line says "Working toward" instead.

The rest of this section describes Editora's README and is not a rule. Its hero screenshot is a `<picture>` with a dark `srcset` under `prefers-color-scheme: dark`, so GitHub shows the screenshot that matches the reader's theme, and it states that the app is built with the help of AI coding tools.

Documentation

An app ships with its documentation. There are three readers and three places: the person using the app reads the **website's docs**; the person changing the app reads `docs/`; a coding agent reads `AGENTS.md`. The README is the landing page that points to each. None of it is optional, and none of it is written after the fact: a change that affects what a reader would learn updates the documentation with it (DOCS-16).

Reference: Editora's `AppInfo.docsUrl()`, `WindowCommandRegistrar` (`help.documentation`), `MenuBarModel` (the Help menu), `WelcomePane` (the Help row), `CommandPalette.docsUrl` and `SettingsFrameBits.helpUrl` (deep links); the repository's `docs/README.md`, `docs/conventions.md#docs-in-the-same-pr` and `CONTRIBUTING.md`; and the editora-website repository's `src/content/docs/` and `scripts/`.

User documentation

- **DOCS-01** `MUST` Every app has user documentation: a *Getting started* page (install, first launch, the first thing to do), one page per feature area, the keyboard reference, the Settings, and an FAQ. It is written for the person using the app, in their words, with screenshots from the real window. The README, a landing page of at most 250 lines (REPO-01), is not it.
  - *Check (code):* the app names a website for its user docs: its README or its About links one, or the address `help.documentation` opens (DOCS-06) is on one. The repository's own address on GitHub is not one. An app whose repository and About name no website has no user documentation to point at, and fails the rule from the app's checkout. The rest is in the website's repository: its docs (the files of `src/content/docs/` in editora-website) have a getting-started page, a keyboard reference (a keymaps page there), a settings or configuration page, an FAQ and a page per feature area, and the pages carry screenshots of the app's real window. The website is a repository of its own (DOCS-02), which the app's checkout does not hold. An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* editora-website `src/content/docs/` (43 pages in nine categories, from *Getting started* to *Help*); torrenta-website (19), termina-website (16), mirada-website (11).
- **DOCS-02** `MUST` The user docs live on the app's website, under `/docs/`, in the app's `<app>-website` repository, beside the feature pages, the screenshots, the news posts and the blog. Each docs page is a Markdown file whose frontmatter gives the page's category and its place in it (editora-website's keys are `category` and `order`); the sidebar, the sitemap and the RSS feed follow. The app's repository carries none of them: a page kept in the app's `docs/` fails the rule when any section of it tells the app's user how to install or use something, even if the rest of the page is for contributors, and whether or not the website has it too.
  - *Check (code):* no page in the app's `docs/` has a section that tells the person using the app how to install or use something: one found there fails the rule from the app's checkout, whoever the rest of the page is written for. So does an app whose repository and About name no website for its user docs (DOCS-01): its user docs live on none. The rest is in the website's repository: every docs page there (the files of `src/content/docs/` in editora-website) gives its category and its order in its frontmatter. An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* editora-website `README.md` (*Content*). Editora doesn't yet, though its `AGENTS.md` says "User-facing feature documentation goes to the separate website repository": `docs/plugins.md`, a page for plugin authors, has a section that tells the user the three ways to install a plugin and one on troubleshooting plugins. Termina doesn't either: `docs/reference/installing.md` tells the person installing it which download to take, how to install each and how to check its signature, and its README sends the reader there. Nor does Torrenta: `docs/headless.md`, `docs/webapi.md` and `docs/search-providers.md` are guides for the person using it.
- **DOCS-03** `SHOULD` Every link from the app into its user docs opens the running version's docs, the address `help.documentation` opens (DOCS-06) as much as a link to one page. Pages live under `/docs/v-<version>/…`; bare `/docs` and every unversioned slug redirect to the current version, so a link in a page never carries a version and a bump just regenerates one file (editora-website's `src/lib/version.ts`). The app builds the base from its own version with `-SNAPSHOT` dropped, so a development build points at the release it is working toward instead of a path that would always 404.
  - *Why:* a release that pointed at another release's docs would claim what is not true (*Never claim what is not true*).
  - *Check (code):* the address the app opens for its docs is its website's `/docs/v-` plus the app's version without `-SNAPSHOT`: another address, the unversioned `/docs/` among them, fails the rule from the app's checkout. An app that opens no docs address has no link to judge and nothing to fail here: the missing address is DOCS-06's. The rest is in the website's repository: it serves `/docs/v-<version>/` and redirects the unversioned paths (in its `astro.config.mjs`, in editora-website). An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* Editora `AppInfo.docsUrl()`, `releaseVersion()`; editora-website `astro.config.mjs`, `README.md` (*Versioned docs*). The app template builds the same address (`docsUrl()` in its `AppInfo.java`). Termina and Mirada don't yet: each opens its site's unversioned `/docs/` (`DOCUMENTATION` in Termina's `AppInfo.java`, `DOCS_URL` in Mirada's `AppInfo.java`). Torrenta, Modula and nux open no docs address. The other three sites carry a comment to port these routes when their app first deep-links.
- **DOCS-04** `SHOULD` The website's docs are generated where the app is the source. The Commands and Keybindings pages are generated from the app's command registrations, `messages.properties` and keymaps; What's new from `CHANGELOG.md`, and the Roadmap from `TODO.md` where the app keeps one (REPO-08); the version from `pom.xml`. The output is committed, so the site builds without the app checked out. A check fails the site when a docs page quotes a command id that does not exist or a chord that is bound to a different command.
  - *Why:* one fact, one source (*One source of truth for each fact*).
  - *Check (code):* the website's repository holds the generators and their committed output (`scripts/` and `src/lib/` in editora-website) and a check of the ids and chords its docs quote (`npm run check:docs` there), and the site's build or its deploy workflow runs that check. An app that names no website has no docs to generate: the rule fails with DOCS-01. All the rest is in the website's repository. An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* editora-website `scripts/GenCommands.java` (every `Command.of` in Editora's `com.editora.ui`, `WindowCommandRegistrar` holding most), `GenRoadmap.java`, `GenFeatures.java`; `scripts/check-doc-commands.mjs` (`npm run check:docs`), which exits non-zero on a missing id or a chord bound to another command. Nothing runs it yet: neither the `build` script in `package.json` nor `pages.yml` calls it, so it fails the site only when someone runs it by hand.
- **DOCS-05** `MUST` The notes of a GitHub release, the text under its title, are written for the person using the app: they say what changed for them, and are not a list of commits or pull requests. In an app with a Welcome page, the page's *What's New* link opens this build's release, so those notes are what it shows. The bold one-line summary that opens each `CHANGELOG.md` entry is REPO-02; the website's What's new is DOCS-04; taking the notes from the version's `CHANGELOG.md` section is REL-04.
  - *Check (code):* the release's notes are not generated from the history: the JReleaser configuration's `changelog` is not a `format` of commit hashes and titles, and no release step sets `generate_release_notes`. They come from text someone wrote, the version's `CHANGELOG.md` section or a notes file. And in an app with a Welcome page, the page's *What's New* link opens the build's own release page (`AppInfo.releaseNotesUrl()` in Editora).
  - *Reference:* Termina's `jreleaser.yml` takes the notes from a file (`changelog.external`) that `scripts/release-notes.sh` writes from the version's `CHANGELOG.md` section; the app template, Torrenta and Mirada do the same. Editora doesn't yet: its `jreleaser.yml` builds the notes from the commits (`format: '- {{commitShortHash}} {{commitTitle}}'`), and that list is what its *What's New* opens (`AppInfo.releaseNotesUrl()`, `WelcomePane`, `welcome.whatsNew`). Modula's `release.yml` and nux's `release.yml` have GitHub generate them (`generate_release_notes: true`).

Reachable from the app

The documentation is one command away, so nobody has to know the website's address.

- **DOCS-06** `MUST` The `help.documentation` command opens the user docs in the browser. It is a command like any other, so it is in the palette and bindable, and where the app has a Help menu (the Help menu of a menu bar, or a Help submenu of the app's main menu, which WIN-14 defines) a **Documentation** row of that menu runs it through the registry like every other row. An app with a Welcome page repeats the command in the page's Help row. The row's place in the Help menu is DOCS-07.
  - *Check (code):* the command `help.documentation` is registered and opens the user docs' address in the browser; where the app has a Help menu, a row of that menu runs the command (the Help `MenuSpec` of Editora's `MenuBarModel` lists it), and a row beside a Help submenu is not in it; and in an app with a Welcome page, the page's Help row runs the same command through the registry (`welcome.documentation` in Editora's `WelcomePane`).
  - *Reference:* Editora `WindowCommandRegistrar` (`help.documentation` → `MainController.openDocumentation()`), `MenuBarModel` (`menubar.help`), `WelcomePane` (`welcome.documentation`). Mirada's macOS Help menu has the row (`MacMenus`); its main menu doesn't yet: there the Documentation row sits beside the Help submenu, not in it (`ui/TitleBar.java`).
- **DOCS-07** `SHOULD` Where the app has a Help menu (the Help menu of a menu bar, or a Help submenu of the app's main menu, which WIN-14 defines), its rows run in this order: Command Palette · Documentation · the keyboard reference (DOCS-08) · Welcome Page · Check for Updates · Message Log · Debug Log · Report a Problem · Pauta Conformance · About. In an app with a Simple mode, that mode's Help menu keeps Command Palette, Documentation, Welcome Page and About. An app without a feature drops its row; nothing moves.
  - *Check (code):* the rows of the Help menu the app has are in that order, in the full menu and in Simple mode's where the app has one (both `menubar.help` menu specs in Editora). An app with a Help menu in a menu bar on one OS and a Help submenu in its main menu is judged on both.
  - *Reference:* Editora `MenuBarModel` , both `MenuSpec("menubar.help", …)` , for the rows its Help menu has. It has no Welcome Page row, since `view.welcome` is still in the View menu (WELC-03), and no keyboard-reference row. Termina's `MenuBarModel` (`documentation` , `whatsNew` , `keyboardShortcuts` , then `checkForUpdates`) and Mirada's macOS menu bar, `MacMenus` (`PALETTE` , `DOCUMENTATION` , `SHORTCUTS` , then `CHECK_FOR_UPDATES`) put the keyboard reference there; Termina's What's New row, between the two, is its own. The Help submenu of Mirada's main menu doesn't follow the order yet: it holds Message Log, Debug Log, Report a Problem, then Check for Updates, and Documentation, the keyboard reference and About are rows of the main menu beside it (`ui/TitleBar.java`).
- **DOCS-08** `SHOULD` Where the app has a Help menu (the Help menu of a menu bar, or a Help submenu of the app's main menu, which WIN-14 defines), a row of it opens a keyboard reference: a searchable window of every key, grouped as the menus are, or the keybinding editor.
  - *Check (by hand):* open it from Help, search for a command, and find its key.
  - *Reference:* Termina's `ShortcutsWindow` , a row of its Help menu, and Mirada's `ShortcutsWindow` , a row of its macOS Help menu; in Mirada's main menu the row sits beside the Help submenu. Editora doesn't yet: its keybinding editor is a page of Settings (Keyboard first), and no row of its Help menu opens it (`MenuBarModel`).

A command's one-line description, shown under its title in the palette, is its in-app documentation: every command's title and description come from the message catalogs, and the parity test holds it (KBD-03).

- **DOCS-10** `MAY` An app with a Search Everywhere picker opens per-command help from it as the Command Palette does: with a command highlighted, the palette's help chord (PAL-08) opens the command's page, `/docs/v—<version>/commands/<id>` . The two pickers share the chord and the URL builder, so they can never disagree.
  - *Check (code):* Search Everywhere takes its help chord the way the palette does and builds the page's address with the palette's own method (Editora's `CommandPalette.docsUrl`).
  - *Reference:* Editora `SearchEverywhereHost` (`CommandPalette.docsUrl`) and `SearchEverywherePopup` (`docsChord` , from `PickerKeys.freeChord` as the palette's); `CommandPalette.docsUrl` .
- **DOCS-11** `MAY` Each page of the Settings window has a link to its docs page, *Help for* and the page's name, from one map of page to slug.
  - *Check (code):* one map from Settings page to docs slug feeds every page's help link.
  - *Reference:* Editora `SettingsWindow` , `SettingsFrameBits.helpUrl` , `settings.help` .

Strings

Key	English
`command.help.documentation`	Documentation
`command.help.documentation.desc`	Open the <App> documentation site for this version in your browser.
`welcome.documentation` / `welcome.whatsNew`	Documentation / What's New
`settings.help`	Help for {0}

Developer documentation

- **DOCS-12** `MUST` `docs/` has an index. `docs/README.md` says who the folder is for (contributors; user docs live on the website) and lists every guide with one line; that the code wins is DOCS-17. Which guides the folder holds is REPO-07.
  - *Check (code):* `docs/README.md` exists, says who it is for, and links each guide in `docs/` .
  - *Reference:* Editora's `docs/README.md` ; the template's `docs/` .
- **DOCS-13** `SHOULD` Every choice a newcomer would question has a decision record: a fork, a rejected dependency, a storage format, an escape hatch. It is one numbered Markdown file, `docs/decisions/NNNN-slug.md` , with the context, the decision and what it costs.
  - *Check (code):* `docs/decisions/` holds `NNNN-slug.md` files, each with context, decision and consequences.
  - *Reference:* Editora's `docs/decisions/` (18 records, from `0001-richtextfx-fork.md`).
- **DOCS-14** `MUST` Three files have three jobs. `AGENTS.md` is the operational guide (what it holds is REPO-05). `CONTRIBUTING.md` is the short version for a person, pointing into `docs/` . `docs/` holds the durable explanations. Exhaustive history goes to `docs/reference/` , not into any of the three.
  - *Check (code):* `CONTRIBUTING.md` links into `docs/` . Neither `AGENTS.md` , `CONTRIBUTING.md` nor a guide in `docs/` (REPO-07) holds exhaustive history: a record of past work, or a catalog that sets out to list every part of a subsystem or every command (Editora's `architecture-catalog.md` and `commands.md` are two). Where the app keeps such material, it is in files under `docs/reference/` ; an app that keeps none needs no `docs/reference/` and passes on the link. An audit report (`docs/pauta-audit.md`) is not such material.

- *Reference:* Editora's `AGENTS.md` (*Documentation map*), `CONTRIBUTING.md`, `docs/reference/README.md`.

- **DOCS-15** `SHOULD` Javadoc sits on its member: a doc comment above another doc comment documents nothing. Comments say why, not what.
  - *Check (code):* no Javadoc comment in the main or the test sources is followed by another with only blank lines or `//` comments between them. A test that asserts it is the surest evidence.
  - *Reference:* Editora `StackedJavadocTest`, which fails the build on one.

Kept in step

- **DOCS-16** `MUST` The docs change with the code. A change that affects behavior updates `CHANGELOG.md` and the contributor guides it touches in the same pull request, the website's user docs in a pull request on the website's repository opened with it, and `TODO.md` where the app keeps one (REPO-08); never as a follow-up. A reviewer treats a feature PR with no docs change as incomplete.
  - *Check (by hand):* read the app's most recently merged feature pull requests on GitHub, as many as it takes to see the habit: each one's diff touches `CHANGELOG.md`, the guides it concerns and, where the app keeps one, `TODO.md`, and one that changes what a user sees names the website's pull request. A recent one that changed behavior and touched none of them fails the rule. That `docs/conventions.md`, `CONTRIBUTING.md` and `AGENTS.md` state the rule shows it is asked, not that it is kept. The person reads GitHub, not the running app, and the rule is still checked by hand: an audit reports it as `manual`.
  - *Reference:* Editora `docs/conventions.md#docs-in-the-same-pr`, `CONTRIBUTING.md` (step 4, which names the website repository), `AGENTS.md`; the template's `docs/conventions.md`. Editora, the template, Mirada and nux keep a `TODO.md`; Termina, Torrenta and Modula keep none.
- **DOCS-17** `MUST` When documentation and code disagree, the code wins, and the documentation is fixed in the same change. `AGENTS.md` and `docs/README.md` say so, where a coding agent and a contributor start reading. Pauta applies the same rule to itself.
  - *Check (code):* `AGENTS.md` and `docs/README.md` say so.
  - *Reference:* Editora's `AGENTS.md`, `docs/README.md`.
- **DOCS-18** `SHOULD` An app's screenshots are retaken, not described: its capture hook or headless fixture (QUAL-09) renders the real window at 2× after any UI change, for the README (its hero in light and dark), the website and bug reports.
  - *Why:* a stale screenshot claims what is not true.
  - *Check (by hand):* the README's hero, in its light and its dark form, and the website's screenshots show the current UI at 2×; they are rendered from the real window by the capture hook or headless fixture QUAL-09 asks for, not drawn or described.
  - *Reference:* Termina's `DevCapture.java` and Torrenta's `DevCapture.java`, and nux's `ui/Screenshot.java` (Shared features). None of the three renders at 2× yet: each writes what `scene.snapshot(null)` returns, the scene at its own size. Editora doesn't yet: it has no capture hook, and its headless fixture writes a screenshot from three FX tests only, when `-Deditora.test.shots=<dir>` is set, and at the scene's own size too (`SplitViewFxTest`).

The README's own shape, at most 250 lines that leave the detail to the website's docs, is REPO-01.

Code quality gates

The Platform baseline turns the checks on: NullAway as an error on every package, each `@NullMarked` in its own `package-info.java` (PLAT-09), and JaCoCo's build-wide coverage floors (PLAT-12). This page is how to live with them and what to test beyond them: fix a nullness report instead of hiding it, keep each decision in a pure helper with unit tests (principle 12), keep the FX tests tagged, free of sleeps and out of the real home folder, look at the real window with nobody at the screen (principle 13), and back the claims that matter (every command runs, every setting survives a restart, the release still signs) with guard tests that have been seen to fail. These are the gates that have paid for themselves across the family.

- **QUAL-02** `SHOULD` A NullAway report is fixed, not suppressed: put `@Nullable` on what may be null.
  - *Check (code):* `grep -rn '@SuppressWarnings("NullAway")' src/main` finds nothing.
  - *Reference:* Editora; one suppression is left, in `recovery/RecoveryCodec.java`.
- **QUAL-03** `SHOULD` Coverage floors are a regression net, not a target. Beyond the build-wide floor of 0.90 lines and methods (PLAT-12), the build holds a branch floor (Editora's is 0.79), and each package of 300+ lines gets its own floor about three points below its measured level; when a package's coverage rises, its floor is ratcheted up.
  - *Check (code):* the `jacoco-check` rules in `pom.xml` carry a `BUNDLE` `BRANCH` limit and a `PACKAGE` line floor for each large package.
  - *Reference:* Editora `pom.xml` `jacoco-check`.
- **QUAL-04** `SHOULD` When a guide quotes the coverage floors, a test fails when the quoted table and the pom disagree.
  - *Check (test):* a build test reads the `jacoco-check` rules from `pom.xml` and compares them with the guide's table.
  - *Reference:* Editora `BuildHygieneTest.theCoverageFloorsQuotedInTheTestingGuideAreTheOnesTheBuildEnforces`, `docs/testing.md`.
- **QUAL-05** `SHOULD` Extract the decision, then test it (principle 12): what decides whether a control is enabled or shown, or where a key or a click leads, sits in a pure helper with unit tests. The headless FX harness is for a test whose point is controller or toolkit behavior.
  - *Check (code):* enablement, visibility and navigation rules sit in classes that import no `javafx` type, each with a unit test that does not carry the FX tests' tag (QUAL-06).
  - *Reference:* Editora `docs/testing.md` (*Prefer pure logic*): `ui/Chrome`, `editops/TextNav`, `config/PathKeys`, each with a plain unit test.
- **QUAL-06** `MUST` Every test class that boots the JavaFX toolkit carries one JUnit tag, the same for all of them (Editora's is `fx`), so the pure suite runs alone by excluding that tag (`-DexcludedGroups=fx` in Editora).
  - *Check (code):* every test class that boots the toolkit (`Platform.startup`, `FxToolkit`, or a helper that calls one, such as Editora's `FxTestSupport`) carries the app's FX tag in a JUnit `@Tag`, whatever the tag is called (`@Tag("fx")` in Editora).
  - *Reference:* Termina holds it by test: `FxTestTagsTest` fails when a test source that names `Platform.startup(` or `FxToolkit.` has no `@Tag("fx")`. Editora states it in `docs/testing.md` and does not hold it fully yet: 38 of its `*FxTest` classes, 18 of them `ProjectMap*FxTest`, carry no tag. Torrenta doesn't yet: 26 of its test classes start the toolkit through its `FxToolkit` extension, and no test in the repository carries a tag.
- **QUAL-07** `SHOULD` Tests do not sleep. A scope owns the workers and FX callbacks (Editora's `AsyncTestScope`), and tests wait on latches.
  - *Check (code):* `grep -rn 'Thread.sleep' src/test` finds nothing.
  - *Reference:* Editora `AsyncTestScope`, `docs/testing.md`. Many older Editora tests still poll with `Thread.sleep`.
- **QUAL-08** `SHOULD` A test window writes only inside its temp folders, never in the real home directory.
  - *Check (code):* the fixture that builds a test window gives it a temp config folder, and points every other folder a command writes to (the one a configuration export lands in, say) at a temp folder too. A test that asserts it is the surest evidence.
  - *Reference:* Editora `FxWindowFixture` (a temp config folder and a temp export folder for each window); `CommandSweepFxTest.leftInTheHomeDirectory` looks in the home directory after running every command.
- **QUAL-09** `MUST` The app has a way to show its real window with nobody at the screen: a capture hook that saves a screenshot of it, or a headless fixture that builds it as the app does. That is what verifies UI work in the running app (principle 13), with real key and mouse events, before a UI change is called done.
  - *Check (code):* the app has a capture hook (Termina's `DevCapture`) or a headless fixture (Editora's `FxWindowFixture`) that builds the real window through the code the app itself starts with.
  - *Reference:* Editora `FxWindowFixture` (the real `MainController` window, headless); Torrenta's `DevCapture.java` and Termina's `DevCapture.java`.
- **QUAL-10** `SHOULD` A guard that cannot fail is decoration. When you add a drift or parity test, break the thing it guards once and watch it fail.
  - *Check (by hand):* break what the new test guards, run it, and see it fail.
  - *Reference:* Modula's `DependenciesTest`.

Guard tests that pay for themselves across the family:

- **QUAL-11** `SHOULD` A command sweep runs every registered command in a real headless window.
  - *Check (test):* one FX test iterates the registry and runs each command.
  - *Reference:* Editora `CommandSweepFxTest`.
- **QUAL-13** `SHOULD` A settings round trip shows that every switch, number and choice survives a restart.
  - *Check (test):* one FX test sets every control, reloads the settings and reads each back.
  - *Reference:* Editora `SettingsRoundTripFxTest`; the app template's `SettingsRoundTripFxTest` (`everyControlSurvivesARestart`), which changes every combo box, switch and field on every Settings page, restarts on the same folder and reads each one back.
- **QUAL-15** `SHOULD` A release supply-chain test keeps the signing and attestation steps where they are.
  - *Check (test):* a test reads the workflows and fails when a signing, notarization or attestation step moves or goes.
  - *Reference:* Editora `ReleaseSupplyChainTest`.

Two more guard tests are rules on the pages they guard: the message parity test is INTL-08 (every key in every catalog) and KBD-03 (every command has a title and a description), and the contrast test is ICON-14.

Performance

A desktop app is judged by how it feels under the hand: a key that lags or a window that hangs while it reads a file is a defect, not a slow feature. These rules come from Editora's `docs/performance.md`, the contract every change on a hot path there must honor.

The UI thread

- **PERF-01** `MUST` Never block the JavaFX application thread. No file or network I/O, no process start or wait, no parsing, searching, hashing or JSON of any size on it, and no `Thread.sleep`, `Future.get()`, `join()` or `Process.waitFor()` there. Two exceptions. At start-up, before the first window is shown, `start()` may wait once for the config read that PERF-16 began before the toolkit, and may read the bundled resources that window is built from: its fonts, registered there or by a thread `start()` waits for once (ICON-02 requires them before any CSS is applied), the message catalog, the keymap and its icons. Nothing else at start-up is excepted: a log file opened, a file or a store of the user's read (presets, profiles, schedules, themes the user added), a ledger read, a socket bound or a lock taken blocks the thread as it would anywhere, and so does a font registered on it after the first frame. And as the app quits, when no frame is left to block, the quit method (WIN-13) or `Application.stop()` may make the last save on it: write the settings, the window state and the session, or wait for a write already under way. An app whose `start()` loads its settings itself, whether or not `main` read the file earlier, or waits for a read begun after the toolkit, has PERF-16's gap and none here.
  - *Check (code):* `grep -rnE 'Files\.[a-z]|File(Input|Output)Stream|File(Reader|Writer)|RandomAccessFile|getResourceAsStream|loadFont|HttpClient|openConnection|openStream|new Socket|SocketChannel\.open|ProcessBuilder|getRuntime\(\)\.exec|\.waitFor\(|Thread\.sleep|\.join\(\)|uture[A-Za-z]*\.get\(|\.get\([^)]*TimeUnit|invokeAndWait|\.await\(' src/main/java` lists the candidates: the lines that read or write a file or a bundled resource, open a connection, start or wait for a process, sleep, or wait for another thread. Each line is judged by reading what calls it. It is fine when only an executor's task or a thread's body reaches it; when `start()` reaches it before the first window is shown and it is one of the things the statement allows there, the one `join()` or `get()` on the config read begun before the toolkit, or the reading of a resource bundled with the app that the first window is built from: the registering of the fonts it draws with (Editora's `Fonts.loadForFirstPaint`) or the one wait for the thread that registers them, the message catalog (Editora's `Messages.init`), the keymap (Editora's `KeymapManager.loadNamed`) or the window's icons; or when it is the last save, made by the quit method or `Application.stop()`, which may do nothing else that blocks. It is a violation when an event handler, a listener, a `Platform.runLater` body, a timer's or an animation's callback, or `start()` reaches it on the FX thread, itself or through a method it calls: a handler that calls a settings store's `save()` writes the file on the FX thread. In `start()` everything else is a violation, before the first window is shown or after: a second wait for the config, a log file opened, a file or a store of the user's read (presets, profiles, schedules, a folder of themes the user added), a ledger read, a socket bound, a lock taken. Only the settings are left to PERF-16: a load of them made in `start()` itself, whether or not `main` read the file earlier, and a wait for a read of them begun after the toolkit. A font registered on the FX thread after the first frame is a violation too. A grep finds only the spellings it is given: read the handlers and listeners too for parsing, searching, hashing or JSON done in place, and for a wait under another name.
  - *Reference:* Editora `docs/performance.md` §1; its one start-up wait is `takeBootstrap` in `App.java`, which joins the `config-prefetch` read, and `Fonts.loadForFirstPaint`, called next, registers its first fonts (`ui/Fonts.java`); the `config-prefetch` thread reads the catalog too, and `start()` reads it only when that thread picked another language (`Messages.init`). Mirada's `start()` makes the two waits the first exception allows: `settings()` joins the settings read, which also reads its catalog (in its jlink image that read begins after the toolkit, which is PERF-16's gap and none here), and `Fonts.ready`, before the window is shown, joins the thread that registers its fonts (`App.java`, `ui/Fonts.java`). For the last save, the app template's `MainWindow.quit` waits for the settings write on a background thread and exits from there, which needs no exception, and its `Application.stop()` then waits for the writer once more on the FX thread (`Running.close` in `ui/Startup.java`), which the second exception allows; Termina's `WindowManager.closeAll`, run from `Application.stop()`, flushes its settings on the FX thread, which the second exception allows. The app template does its start's file work off the FX thread: `prepare` in `ui/Startup.java`, which refuses the FX thread and runs on a thread of its own while the toolkit starts, opens the Debug Log's session file, reads the settings, the catalog and `pauta.json` and binds the single-instance socket; `start()` then registers its first fonts, which the first exception allows (`Fonts.load`), and the remaining font is registered off the FX thread after the first frame (`Fonts.loadDeferred`). No app has been read through for the rule yet, and none holds it everywhere. Found so far: Editora's `start()`, in that `App.java`, does on the FX thread, before its first window is shown, what no exception covers: its calls to `DebugLog.attachFile` (the Debug Log's session file), `SharedConfig.claimInstance` (the lock on a file in the config folder), `ProcessRegistry.reapOrphans` (the ledger of the servers it started), `McpServer.reapStaleEndpoint` (a stale `mcp-endpoint.json`) and `Themes.loadUserThemes` (the user themes' folders). Some of its handlers still touch the disk too, for example `renameFileTo` (`Files.exists`, `Files.move`) and `openPersonalDictionary` (`Files.writeString`) in `ui/MainController.java` and the new-folder prompt in `ui/ProjectPanel.java` (`Files.createDirectories`). Torrenta's `start()` reads its catalog, which the exception allows, and opens that file the same way (`DebugLog.attachFile` in its `App.java`); its main window's constructor, before the window is shown, reads six stores of the user's (the profiles, the labels, the watch folders, the search providers, their definitions and the RSS feeds, in `ui/MainWindow.java`), and `start()` reads `profiles.json` again once the window is shown (`Startup.profile`). Termina reads its settings, its first fonts, its catalog and its imported themes in `Application.init()`, off the FX thread (`prepare` in its `App.java`); building its first window then reads the bundled keymap (`ui/Keymap.java`, on the first `Keymap.defaultFor`) and decodes the window's icons (`ui/Icons.java`, `Icons.applyTo`) on the FX thread, which the exception allows, and once the window is shown `start()` deletes a stale socket file and binds the socket for later launches there, which it does not (`serve` in `instance/SingleInstance.java`). It also registers its remaining font families on the FX thread after the first frame, which no exception covers (`Fonts.loadRest`, called from that `App.java`). Mirada, on macOS, compares two paths on the FX thread when it was started with a file and the OS hands it its first one (`Files.isSameFile` in `os/MacOpenFiles.java`). nux waits up to two seconds for the tray icon's removal in the handler of Settings' Save (`AwtTray.dispose`, reached from `TrayController.settingsSaved`) and writes its `--screenshot` file from an animation's callback (`ui/Screenshot.java`). Torrenta writes its settings from the listeners of its Settings rows (`saveQuietly` in `ui/SettingsWindow.java`) and asks the file store for the free space on each keystroke in a save-folder field (`updateFreeSpace` in `ui/LocationDialog.java`). Modula reads its presets and its schedules in its pane's constructor, before the window is shown (`config.loadPresets` and `config.loadSchedules` in `ui/RadioPane.java`), writes its presets and settings from handlers (`config.savePresets` and `config.saveSettings` in that file) and, building its Settings window, starts `ffmpeg -version` and waits up to five seconds for it (`Encoders.isAvailable`).
- **PERF-02** `MUST` Touch the scene graph only on the FX thread. Background work hands its result back with `Platform.runLater`; nothing else creates, changes or reads live nodes.
  - *Check (code):* `grep -rnE 'new Thread\(|Thread\.(ofVirtual|ofPlatform|startVirtualThread)|\.(submit|execute|schedule|scheduleAtFixedRate|scheduleWithFixedDelay)\(|(supply|run)Async\(|\.(thenAccept|thenRun|thenApply|thenCompose|whenComplete|handle|exceptionally)(Async)?\(' src/main/java` lists the candidates: the lines that start a thread, hand a task to an executor, or add a step to a future that completes off the FX thread. Each line is judged by reading the task's body. It is fine when the body touches nothing a scene shows, or touches it only inside `Platform.runLater`. It is a violation when the body, or a method it calls, sets a control's value, changes an observable list a view shows, or reads a live node's state outside a `runLater`. A callback that a library runs on a thread of its own (a file watcher, a process's reader, a logging handler) is a task body too, and the grep does not list those: read the listeners the app gives its services as well.
  - *Reference:* Editora `git/GitService.java`: every task it gives its workers hands the result to its caller inside `Platform.runLater`. No app has been read through for the rule yet.
- **PERF-03** `MUST` A background result is applied under a generation guard, so an answer to an old request never overwrites a newer one.
  - *Check (code):* `grep -rn 'runLater' src/main/java` lists the candidates: the lines where another thread hands something to the FX thread. Each line is judged by reading it. It needs the guard when what it applies answers a request that can be made again before the answer comes: a search, a load, a status or completion query that typing, a

selection or a refresh starts. There each request takes a mark that the next request replaces (a counter, as Editora's `AtomicLong.incrementAndGet`, or a helper that keeps the latest request only, as Editora's `LatestOnly`), and the answer is applied only while its mark is still the newest, compared inside the `runLater`. It is a violation when such an answer is applied with no comparison, or with one made only before the `runLater`. A line needs no guard when nothing newer can replace what it hands over: an event, a batch of a stream, a result asked for once at start-up.

- *Reference:* Editora `git/GitService.java` (`refresh`, with its `refreshGen` counter), `search/SearchService.java` and `ui/LatestOnly.java` (`LatestOnlyTest`); Termina (`work/LatestOnly.java`) and Mirada (`ui/LatestOnly.java`) carry the same helper. No app has been read through for the rule yet. Found so far: nux applies each Refresh of its Events window, and each load of the Energy and Outages tabs of its History window, with no mark (`ui/EventsWindow.java`, `offFx` in `ui/HistoryWindow.java`), beside a Charts tab that numbers its loads and applies only the newest.

- **PERF-04** `SHOULD` Every thread is named and is a daemon.
  - *Why:* a thread dump then says what is running, and a stray worker never keeps the app alive after its last window closes.
  - *Check (code):* every `Executors.new…` and `new Thread` gets a name and `setDaemon(true)`, through a thread factory such as `daemon("my-feature")`; virtual threads are daemons already.
  - *Reference:* Editora `terminal/TerminalExecutors.java` (its `daemon(name)` thread factory); the canonical shape is in The UI thread.
- **PERF-05** `MUST` Blocking work stays off the common `ForkJoinPool`: no `supplyAsync` or `runAsync` without an executor runs a blocking call, and a `Process` subclass overrides `onExit()`, whose inherited version is a `waitFor()` on that pool.
  - *Why:* the common pool has one worker fewer than the machine has cores, so a few blocked waits (`CompletableFuture.supplyAsync` of a `waitFor`) starve every other user of it.
  - *Check (code):* no `supplyAsync` or `runAsync` without an executor wraps a blocking call (a wait, a read or write of a file, a socket or a process), and every class that extends `Process` overrides `onExit()`. An app with no `Process` subclass has only the first half to show.
  - *Reference:* Editora `process/ExitWatch.java`, `docs/gotchas.md`. Editora's `DapManager.reportJavaExit` still waits up to 3 s for an exit code in a `supplyAsync` on the common pool, and Termina's `handOff` in `instance/SingleInstance.java` connects to a socket, writes and reads in one.

The canonical shape, where `daemon(name)` is a small factory that names the thread and marks it a daemon, as in Editora's `TerminalExecutors`:

```
private final ExecutorService exec = Executors.newSingleThreadExecutor(daemon("my-feature"));
private final AtomicLong gen = new AtomicLong();

void request(String input, Consumer<Result> onResult) {
    long mine = gen.incrementAndGet();
    exec.submit(() → {
        Result r = computeOffThread(input);        // heavy work, off the FX thread
        if (mine == gen.get()) {                   // superseded? drop it
            Platform.runLater(() → {
                if (mine == gen.get()) onResult.accept(r);
            });
        }
    });
}
```

For one-off blocking calls (reading a file, stat-ing a path, starting a process), a named virtual thread (`Thread.ofVirtual().name("my-read").start(…)`, the name being PERF-04's) is enough; the result still comes back through `Platform.runLater`.

Hot paths

A hot path is what runs on every keystroke, scroll pulse or progress tick. In Editora they are typing, scrolling, highlighting, the overlays and the gutter. In Torrenta they are the transfer table's refresh, in Termina the terminal grid, in Mirada stepping between images.

- **PERF-06** `MUST` Redraws are coalesced to at most one per pulse, with a flag that one request is already queued. A redraw is the painting of what the app draws itself, a canvas, a chart or a grid of cells; in an app that paints nothing itself, it is the refresh of its list or view.
  - *Check (code):* a redraw request sets a flag (Editora's are named `…Pending`) and schedules one pass, and a request that finds the flag set schedules nothing. A handler that paints, or in an app that paints nothing refreshes its list or view, straight from its event has no such flag and fails: a chart drawn from its width listener and again from its height listener draws twice for one resize. Setting a control's own property in a handler (a label's text, a progress bar's value) is not a redraw: JavaFX draws the control once on the next pulse.
  - *Reference:* Editora `docs/performance.md` §2 (the `redrawPending` shape) and `ui/WindowToolbar.java` (`scheduleEditStateRefresh`, with its `editStateRefreshPending` flag). `ui/FxThrottle.java` is another tool: a leading-edge throttle with a maximum wait and no such flag, which runs the first request of a burst at once and then at most one per interval.
- **PERF-23** `MUST` Work driven by text the user types waits for typing to pause (a debounce) when it leaves the FX thread or cannot finish within a frame: a search of files, a request to a server, a re-highlight. Filtering a list already in memory, as a command palette does, is exempt and runs on every keystroke.
  - *Check (code):* a listener on a text input's text that starts background work, or work too long for one frame, restarts a timer (a `PauseTransition`'s `playFromStart`, or the app's debounce helper) and the work runs when the timer fires; a listener that only filters or ranks an in-memory list needs none.
  - *Reference:* Editora `ui/SearchInFilesPopup.java` (150 ms before `runSearch`), `ui/SearchEverywherePopup.java` (120 ms), `ui/WorkspaceSymbolPopup.java` (150 ms before the language server is asked) and, for the editor's text, `editor/SettledEditDispatcher.java`, one timer that runs the re-highlight 150 ms after the last edit. Its `ui/CommandPalette.java` and `ui/QuickOpen.java` filter their in-memory lists on every keystroke: the exempt case. Editora doesn't hold the rule everywhere yet: the debugger's completion (`ui/DebugCompletion.java`) asks the debug adapter again on every change of the field's text while its list is open, with no timer, and `ui/FileFinder.java` starts reading a folder as soon as the typed path names another one.
- **PERF-07** `MUST` An app that shows a stream as it arrives (the output of a process, log records, progress or readings from another thread) hands it to the FX thread in batches at least 16 ms apart, never one `runLater` per line or record.
  - *Check (code):* in an app that shows a stream, the thread that receives it appends to a buffer that the FX thread drains, with a minimum interval of 16 ms or more between drains, or once a pulse (pulses come a frame apart); no reader calls `Platform.runLater` once for each line or record. A stream is what arrives at its source's pace, however many items that is: a process's output, log records, the results of a scan handed over as each is found. A poll the app paces itself, one hand-over for each tick of a timer of 16 ms or more (a reading fetched every few seconds), already hands over that far apart and needs no buffer. An app whose FX thread is sent no stream, as one whose log window reads its store only when it opens or is refreshed, is not covered.

- *Reference:* Editora `process/OutputPump.java` (`MIN_DRAIN_INTERVAL_MILLIS`), `ui/ConsoleAppender.java`. Torrenta doesn't yet for its event log: the bridge in `ui/EventLog.java` (`logBridge`) hands each warning it catches to the FX thread in a `runLater` of its own. nux doesn't yet for its network scan: `scan` in `service/DiscoveryService.java` hands each server it finds to the FX thread in a `runLater` of its own; its poll hands over one reading every five seconds (`service/PollService.java`), which is paced already. Mirada's Debug Log window (`ui/DebugLogWindow.java`) reads the log when it opens and on Refresh, so no log stream reaches its FX thread.

- **PERF-08** `MUST` Work only on what is visible and only on what changed: no O(document) or O(list) work per keystroke, scroll or tick, and a list the user or the data can grow without a limit the app sets is shown in a virtualized control (`ListView`, `TableView`, a fixed cell size), which builds nodes for the visible rows only. Filtering a list already in memory, as a command palette does, is exempt and runs on every keystroke.
  - *Check (code):* `grep -rnE 'textProperty\(\)|KEY_(TYPED|PRESSED)|setOnKey|setOnScroll|vvalueProperty|AnimationTimer|Timeline|PauseTransition|scheduleAtFixedRate' src/main/java` lists the first candidates: the listeners on text, keys and scrolling, and the timers. Each line is judged by reading what it runs on every keystroke, scroll or tick. It is fine when that handles the edited range or the visible rows, or filters or ranks a list already in memory. It is a violation when it reads, measures, styles or rebuilds the whole document or every item of a list. An editor's own stream of changes is a text listener too (Editora's `plainTextChanges`), which the grep does not list. `grep -rnE 'getChildren\(\)\.(add|addAll|setAll)\(' src/main/java` lists the second: the lines that put nodes into a pane. It is a violation where a loop makes one node for each item of a list the user or the data can grow without a limit the app sets (results, log records, files, transfers), which belongs in a `ListView` or `TableView`; it is fine where the pane holds a set of controls the code fixes, or a list the app caps at a number it names (a bar of at most five preset chips).
  - *Reference:* Editora `editor/highlight/HighlightPass.java`, which restyles only as far as an edit reaches. No app has been read through for the rule yet. Found so far, each a loop that makes a node for every item of a list nothing caps: Modula's preset bar (a chip for each preset, `rebuild` in `ui/PresetBar.java`); nux's dashboard (a card for each saved profile it watches and for each UPS, rebuilt as readings arrive, `setFleet` and `setOverview` in `ui/Dashboard.java`); Torrenta's rule and tags dialogs (a check box for each feed and each tag, `ui/RuleDialog.java`, `ui/TagsDialog.java`); and Editora's crontab preview and pull-request checks card (a node for each job and each check run, `editor/structured/CrontabPreview.java`, `ui/ChecksListCard.java`).
- **PERF-09** `SHOULD` A hot path allocates little: it reuses buffers and arrays, and caches data a repaint only reads.
  - *Check (code):* the per-frame and per-keystroke code allocates no new arrays or images when the size has not changed.
  - *Reference:* Editora `editor/minimap/Minimap.java` (the snapshot image reused at the same size), `MinimapRows.java`. The app template's palette cells keep their text runs, chord chip and tooltip and relabel them as the query changes, making a new run only when a title needs more than the cell had (`ui/Picker.java`).
- **PERF-10** `SHOULD` CSS stays cheap: no `applyCss()` on a large subtree, stable style classes, and spans of the same style merged before they are applied.
  - *Check (code):* `applyCss()` is called only on small nodes; style classes are not toggled per keystroke.
  - *Reference:* Editora `SpanMerger` in `editor/highlight/TextMateHighlighter.java`, `docs/gotchas.md`. The app template's palette draws a title as one text run per change of style, not a node per letter (`ui/Picker.java`).

Memory and size

- **PERF-11** `MUST` Everything that grows is bounded: logs, histories, consoles, caches, undo. The newest is kept.
  - *Check (code):* each store that grows while the app runs (a log, a history, a console's text, a cache, an undo stack) has a named cap, and the code that adds to it drops the oldest entry once the cap is passed. A test that fills a store past its cap and finds the oldest dropped is the surest evidence.
  - *Reference:* Editora `ui/MessageLog.java` (200 entries, the oldest dropped; `MessageLogTest`), `ui/DebugLog.java` (2,000 records, the oldest dropped; `DebugLogTest`), and `test/OutputCap.java`, which keeps the head and the tail of one test's output (200,000 characters) with a note between them of how much was cut. The note is Editora's choice, not part of the rule.
- **PERF-12** `MUST` In an app that opens files or data the user picks, a large input is guarded before it loads: its size is checked before its content is read, or the read is capped, and what will not fit is never read in. The app refuses it with a message or opens only the part that fits. Where an input can open with less, without features the app runs over its content, as a file in an editor can open without highlighting or a minimap, the app steps down as the input grows: those features turn off past one threshold, and past another it opens read-only or capped. An input the app can only read in or refuse (an imported theme, a `.torrent` file) has nothing to step down to.
  - *Check (code):* the code that loads a picked file or import compares its size against a named limit before reading the content, or reads through a cap, and refuses or cuts what passes the limit. Where the app runs features over an input's content that the input can open without (an editor's highlighting, its minimap), a lower named threshold turns them off and a higher one opens the input read-only or capped. Where it runs none, the size check or the cap is the whole of the rule.
  - *Reference:* Editora, whose tiers are the example: `EditorBuffer.LARGE_FILE_BYTES` (5 MB: no highlighting, no minimap) and `HUGE_FILE_BYTES` (50 MB: read-only and cut to 50 MB), and `ui/LoadHeapGuard.java`, which caps a load at what the free heap affords (`LoadHeapGuardTest`). Termina's theme import has nothing to step down to: `importFile` in `ui/Themes.java` reads at most 8 MB and `theme/ThemeImport.java` refuses a file over 4 MB. Torrenta doesn't yet for a `.torrent` file the user picks: `resolve` in `session/Session.java` reads it whole (`Files.readAllBytes`) without looking at its size.
- **PERF-13** `MUST` A closed window releases what it held: listeners, timers, executors, file watchers, native resources.
  - *Check (code):* the code that runs when a window closes (its close or hide handler, or the dispose method that handler calls) stops every timer and animation the window started, shuts down every executor, thread and file watcher it owns, frees its native resources (a process, a socket, a file handle), and removes each listener it added to something that outlives it: the shared settings, a service, another window. A test that opens and closes windows and finds nothing left running is the surest evidence.
  - *Reference:* Editora's `MainController.disposeWindow`, which shuts down each owner of a window's workers in turn, one failure never skipping the rest (`ui/WindowDisposal.java`). Its tests: `FxWindowFixtureLifecycleFxTest`, which finds a disposed window's workers shut down, and `ClosedLspManagers`. The app template (`WindowLifecycleFxTest`), Termina (`WindowLifecycleTest`) and Mirada (`ClosedWindowTest`) have such a test too.

Start-up

- **PERF-14** `MUST` The first frame comes first: only what the first window shows is built before it. Everything else builds on first open: a closed panel's content, the Settings pages, the dialogs and the other windows.
  - *Check (code):* the code that builds the first window (`start()`, and the main window's constructor and what it calls) makes only what that window shows: no Settings window or page, no dialog, no other window and, in an app with tool panels, no closed panel's content. Each of those is made by the code that first opens it, through a supplier or a field filled on first use. A test that opens the first window and finds none of them built is the surest evidence.
  - *Reference:* Editora's lazy `ToolWindow` suppliers, and its Settings pages, built on the first `show` (`ui/SettingsWindow.java`). `LazyToolWindowsFxTest` asserts the tool-panel part: no closed tool window has built its panel. Editora doesn't hold the rule in one place yet: it builds the shared Output console (`buildOutput`), an empty tab pane, with the window, because every build writes to it while it is closed, and the test leaves that panel out. The template's `StartupDeferralFxTest` and Mirada's `StartupDeferralTest` assert it for their first windows. Torrenta doesn't yet: the constructor of its main window makes the Log, the Statistics and the Settings windows (`new LogWindow`, `new StatisticsWindow`, `new SettingsWindow` in `ui/MainWindow.java`).
- **PERF-15** `MUST` What nobody waits for starts after the first frame has painted: the update check, extra fonts, indexes and caches.
  - *Check (code):* the update check, the loading of fonts the first window does not draw with, and the building of indexes and caches start from a callback that runs once the first frame has painted (a pulse after the window is shown, as Editora's `loadRemainingAfterNextPaint` waits for), never from `main`, `init()` or the body of `start()`. A test

that finds the deferred work not started before the first paint is the surest evidence.

  - *Reference:* Editora `ui/UpdateCheckScheduler.java`, `Fonts.loadRemainingAfterNextPaint` (`UpdateCheckSchedulerTest`, `FontsDeferredLoadFxTest`, `StartupDeferralFxTest`). The template (`FirstPaint.then` in `ui/Startup.java`), Termina (`FirstFrame.after` in its `App.java`) and Mirada (`afterFirstPaint` in `ui/ViewerWindow.java`) defer theirs the same way. Modula and nux don't yet: each starts its update check in the body of `start()` (`checkForUpdates` in `ModulaApp.java`, `maybeCheckForUpdates` in nux's `App.java`).

- **PERF-16** `SHOULD` The config read starts on a thread of its own before the toolkit does, so the two overlap and the wait for it that PERF-01 allows at start-up, `start()` taking the result, is short. An app whose `start()` loads its settings itself does not meet this, whether or not `main` read the file earlier.
  - *Check (code):* `main` starts the config read on its own thread before `launch`, from an entry point that is not the `Application` subclass (the launcher starts the toolkit for one before its `main` runs), and `start()` takes that read's result instead of reading the files itself. A load `start()` makes only when the earlier read failed is a fallback and passes.
  - *Reference:* Editora's `config-prefetch` thread in `App.java` (`prefetchConfig`, joined once by `takeBootstrap`), entered through `Launcher.java`. Mirada's `settings-read` thread in its `App.java` (`readSettings`, joined by `settings()`) does the same in the native image, which enters through `Launcher`; its jlink image's main class is `App` itself, so there the toolkit is up before the read starts. Termina reads its settings in `Application.init()` (`prepare` in its `App.java`): off the FX thread, so `start()` waits for nothing, but after the toolkit has started. The app template meets it: its `Main.java`, not the `Application` subclass, starts `Startup.readAhead` on a thread of its own before `launch`, and `App.init` takes the result (`takePrepared` in `ui/Startup.java`), so `start()` reads no settings file. Torrenta (`App.java`), Modula (`ModulaApp.java`) and nux (`App.java`, which also reads the file once in `main` for the language) load their settings inside `start()`. Torrenta's `main` reads the file once as well, on the main thread, for its tray and badge settings.
- **PERF-17** `SHOULD` The app ships precompiled stylesheets (`.bss`). The AOT cache, the other half of a fast start, is REL-10.
  - *Check (code):* the build converts the app's `.css` files to `.bss`, and the packaged app carries them.
  - *Reference:* Editora `scripts/css2bss.java`, run by the `css-to-bss` execution in `pom.xml`. The app template's `dist` profile runs its `ui/StyleCompiler.java` over the built stylesheets, which writes a `.bss` beside each `.css`, and `ui/Themes.java` loads the `.bss` where there is one (`StyleCompilerTest`). No other app converts its own stylesheets yet.

Measure, don't guess

- **PERF-18** `SHOULD` Every change on a hot path says what it costs in its pull request: added FX-thread work, allocation per keystroke or scroll, extra layout passes, memory. "Negligible" is an answer; no answer is not.
  - *Check (by hand):* read the app's recent pull requests that touch a hot path.
  - *Reference:* Editora `docs/performance.md` (its opening paragraph).
- **PERF-19** `SHOULD` Start-up is measured by a script that takes the median of several cold launches, and by phase marks in the app.
  - *Check (code):* the repository has the script, and `main` and the first paint record phase marks.
  - *Reference:* `scripts/measure-startup.sh` in Editora, Termina, Mirada and the app template, whose marks are recorded by `StartupMarks.java` from `main` and the first paint; Editora `perf/Startup.java`.
- **PERF-20** `SHOULD` The app can sample its FX thread when something stalls.
  - *Check (code):* an environment variable or property turns on a sampler of the FX thread's stack.
  - *Reference:* Editora `perf/Startup.java` (`EDITORA_PERF_SAMPLE=1`, with `EDITORA_PERF=1`); Mirada `perf/FxSampler.java` (`MIRADA_PERF_SAMPLE=1`); the app template's `ui/FxSampler.java` (`-Dexampleapp.fxSampler=<ms>` or `EXAMPLEAPP_FX_SAMPLER`, logging the FX thread's stack on a stall; `FxSamplerTest`).
- **PERF-21** `SHOULD` Each hot path that once regressed keeps a headless test.
  - *Check (test):* the fix for a hot-path regression adds a headless test that fails on the old code.
  - *Reference:* Editora `MinimapDeferredRenderFxTest`.
- **PERF-22** `SHOULD` A suspected regression gets a number before and after, from temporary `System.nanoTime` instrumentation if nothing better exists.
  - *Check (by hand):* the pull request that fixes it states both numbers.
  - *Reference:* Editora `docs/performance.md`.

Keyboard first

Every app can be used from the keyboard alone (principle 2), and that rests on one idea: every action is a command with an id (principle 3). Menus, toolbar buttons, the Command Palette and key bindings all run commands, so whatever the mouse can do, the palette or a chord can do too. Keymaps bind chords to those commands in layers, with the user's own bindings on top, and every control the keyboard reaches shows where the focus is. The palette's own shape is the Command Palette template; what a screen reader and the OS's accessibility settings need is Accessibility.

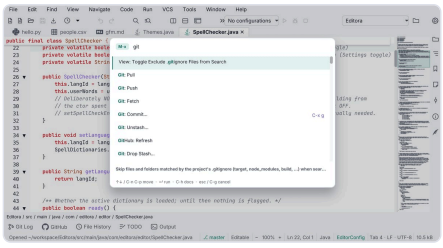
Commands

- **KBD-01** `MUST` Every user-facing action is a command (an id, a title and a runnable) registered in one command registry (Editora's `Command` and `CommandRegistry`).
  - *Check (code):* what a menu item, a toolbar button, a palette row or a key binding does is a registered command: the control runs the command's id through the registry, or calls what the command's runnable calls. No button, menu row or key handler does work no command can reach.
  - *Reference:* Editora `command/Command.java`, `command/CommandRegistry.java`. Editora doesn't hold the rule everywhere yet: the Clear button of its Message Log (`ui/MessageLogPopup.java`) empties the log in its own handler, and the Previous Run and Save Output buttons of its build console (`ui/BuildToolPanel.java`) work the same way, with no command behind any of the three. Termina doesn't either. These buttons, among others, do their work in handlers of their own with no command behind them: Clear in its Message Log (`ui/MessageLogWindow.java`), Refresh, Copy, Export and Clear in its Debug Log (`ui/DebugLogWindow.java`), and Import theme in Settings (`ui/SettingsWindow.java`).
- **KBD-02** `SHOULD` Command ids are dotted and grouped by area, as in `file.open`, `view.toggleStatusBar` and `help.reportProblem`. A rule that names a command's id requires that id; these three are examples.
  - *Check (code):* every registered id is two or more names joined by dots, and the first is the area the command belongs to: `file.open`, and with a sub-area `git.log.checkout`.
  - *Reference:* Editora `ui/MenuBarModel.java`.
- **KBD-03** `MUST` A command's title comes from `command.<id>` and its one-line palette description from `command.<id>.desc` in the message catalogs.
  - *Why:* a registered command is then localized and discoverable without extra work.
  - *Check (code):* every registered command has a `command.<id>` and a `command.<id>.desc` key in the message catalogs, and the registry reads its title and description from them. A catalog parity test that walks the registered commands and fails on a missing key is the surest evidence.
  - *Reference:* the app template's `CommandSweepFxTest` (`everyCommandIsLocalizedAndRuns`) and Mirada's `CommandCatalogTest` (`everyCommandHasItsTitleAndDescriptionInTheCatalog`) walk the registered commands. Editora `i18n/messages.properties`; its `MessagesTest` (`everyCommandTitleHasAMatchingDescription`) walks the catalog instead, so it finds a title with no description but not a registered command with neither.
- **KBD-04** `SHOULD` A command that cannot run in the current context is shown grayed, with a tooltip that says why, not hidden.
  - *Check (code):* the palette and menus list a disabled command with a reason, rather than filtering it out.
  - *Reference:* Editora `ui/Chrome.java` (`paletteEnabled`, `disabledReason`). The app template's menu rows of a command that can't run stay, grayed, with the reason as their tooltip (`ui/AppMenus.java`, `showAvailability`; `RightClickMenusFxTest`, `anUnavailableRowStaysAndSaysWhy`), and its palette lists such a command with the reason.

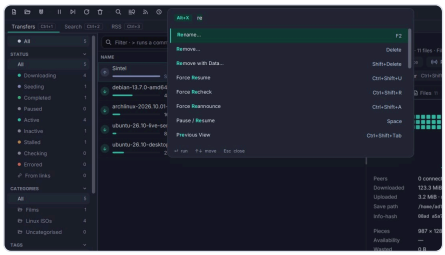
Command Palette

- **KBD-05** `MUST` The Command Palette opens with `M-x` in the Emacs keymap and `Ctrl+Shift+P` (`Cmd+Shift+P` on macOS) in the others; the IntelliJ keymap may use IntelliJ's own Find Action chord, `Ctrl+Shift+A` (`Cmd+Shift+A`), instead. An app with one key table and no Emacs keymap, which is a terminal app's own keymap (KBD-09) or the built-in table of an app that ships no keymap files, counts as one of the others: it binds `Ctrl+Shift+P` (`Cmd+Shift+P` on macOS). An app that ships no keymap files owes this chord and fails KBD-09 all the same: KBD-09 requires the files of every app.
  - *Check (code):* each bundled keymap binds that chord to the palette command, and so does the one key table of an app that has no keymap files. An app whose chord opens the palette from a handler, with no palette command registered, passes this rule and fails PAL-01 for the missing command.
  - *Reference:* Editora `keymaps/emacs.json` (`M-x`), `cua.json` and `vscode.json` (`C-S-p`), `cua.mac.json` (`Cmd-S-p`), all to `palette.show`; `intellij.json` binds `C-S-a`. Termina's one keymap binds `Shortcut+Shift+P`, which is `Ctrl+Shift+P` in `keymap.json` and `Cmd+Shift+P` in `keymap.mac.json`, and nux, which ships no keymap file, binds the same chord in `App`. Torrenta and Modula don't yet: Torrenta's one key table opens the palette with `Alt+X` alone (`ui/Chords.java`), and Modula's key filter tests for `Ctrl` on every OS, so its chord is `Ctrl+Shift+P` on macOS too (`ui/RadioPane.java`).
- **KBD-06** `SHOULD` The palette matches fuzzily over the command's title and over the words of its id or its keywords; the description is shown, not matched. A query that names a command's title outranks a fuzzy hit. Each row shows the command's binding in the active keymap. See the Command Palette template.
  - *Check (code):* the palette's matcher matches the query fuzzily against each command's title and finds a command by a word of its id or a keyword that its title doesn't carry; a query that spells a command's title ranks that command above the fuzzy hits; the description is drawn and never matched, and each row draws the binding the active keymap gives the command. A ranking test that asserts the order is the surest evidence.
  - *Reference:* Editora `ui/CommandPalette.java` (`orderedMatches`), `search/FuzzyMatch.java`. It scores the title first; when the title doesn't match, the id's namespace word as a keyword (`Command.keywords`); then the names IntelliJ and VS Code give the command (`command/OtherIdeNames.java`). It doesn't match the rest of the id or the description, which is shown under the list. `CommandPaletteRankTest` holds the order (`aCommandIsFoundByItsNamespaceWhenItsTitleDoesNotCarryIt`, `aKeywordMatchRanksBelowEveryTitleMatch`, and for a typed title `anExactTitleLeadsAnotherCommandsAlias` and `aWholeOneWordTitleLeads`). The app template's `ui/Picker.java` (`rank`, with `idWords`) matches the title fuzzily and the id's words, never the description, and puts a title that holds the query first (`PickerFxTest`, `ranksTitlesFirstAndSkipsUnavailableRows`).
- **KBD-07** `MUST` The palette and every picker (a card that lists choices to filter and pick one from: QOPEN-01) are in-scene overlays in the window's root `StackPane`, never a `javafx.stage.Popup`.
  - *Why:* on Windows a Popup does not reliably take OS keyboard focus, and the whole app can stop receiving keystrokes.
  - *Check (code):* the palette and every picker add their node to the root `StackPane`; none creates a `Popup`.
  - *Reference:* Editora `ui/OverlayHost.java`, the one host installed in the scene's root `StackPane`, which shows the palette (`ui/CommandPalette.java`), the picker (`ui/QuickOpen.java`) and the picker's relatives (QOPEN-12).

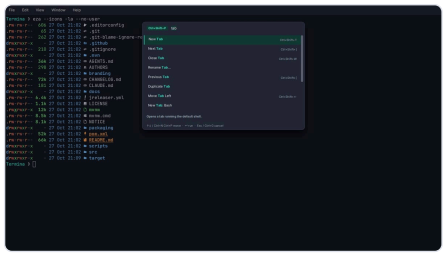
Every other list choice (recent files, themes, keymaps, a setting's values) uses the same picker: QOPEN-01.



Editora: M-x over the editor; typing git lists every Git command.



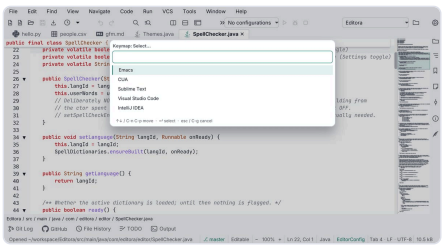
Torrenta: the same palette, filtered to torrent actions with their shortcuts.



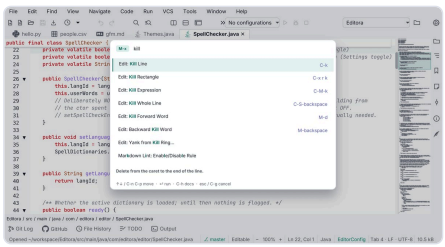
Termina: the palette over a shell session, filtered to tab commands.

Keymaps

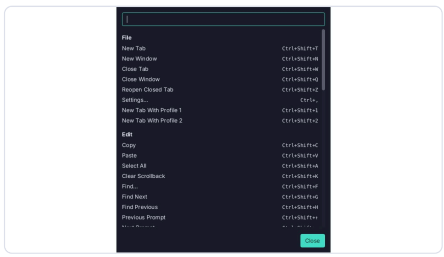
- **KBD-09** `MUST` Except in a terminal app, Emacs and CUA keymaps ship as JSON resources, `emacs.json` and `cua.json`, CUA with a `cua.mac.json` variant where Cmd replaces Ctrl. A terminal's keys belong to the program it runs (`C-a`, `M-x` reach the shell), so a terminal app ships one keymap of its own instead, `keymap.json` and `keymap.mac.json`, binding its commands to chords programs don't use. Neither file binds a plain `Ctrl` or `Alt` (`Option` on macOS) with a letter: off macOS that leaves `Ctrl+Shift+T`, not `Ctrl+T`, and on macOS `Cmd+T`, since no program in the terminal receives `Cmd`.
  - *Why:* an Emacs keymap in a terminal would take the shell's own keys away from it: `Ctrl+W` deletes a word, `Ctrl+C` interrupts, `Alt+F` and `Alt+B` move by word. The shell reads the same keys on macOS, where `Option` with a letter also types a character.
  - *Check (code):* `emacs.json`, `cua.json` and `cua.mac.json` are in the app's resources. In a terminal app, `keymap.json` and `keymap.mac.json` are there instead, and no default binding in either file is `Ctrl` alone or `Alt` alone with a letter. Read each chord as the OS reads it: a modifier that stands for the OS's own shortcut key (JavaFX's `Shortcut`, as Termina writes it) is `Ctrl` in `keymap.json` and `Cmd` in `keymap.mac.json`.
  - *Reference:* Editora `src/main/resources/com/editora/keymaps/`; Termina's `keymaps/`, whose `KeymapTest` (`offMacOsAnApplicationChordNeverTakesPlainCtrlAndALetter`) holds the `Ctrl` half off macOS; no test holds the `Alt` half or reads `keymap.mac.json`, where every chord carries `Cmd` today.
- **KBD-10** `SHOULD` Except in a terminal app (KBD-09), an app also ships IntelliJ, VS Code and Sublime keymaps as JSON resources beside its Emacs and CUA keymaps: `intellij.json`, `vscode.json` and `sublime.json`, each with its `.mac.json` variant.
  - *Check (code):* `intellij`, `vscode` and `sublime` `.json` and `.mac.json` files are in the resources. An app that ships a terminal's keymap, `keymap.json` and `keymap.mac.json` with no `emacs.json` and no `cua.json` (KBD-09), needs none of them.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; has() { find src/main/resources -name "$1" | grep -q .; }; if has keymap.json && has keymap.mac.json && ! has emacs.json && ! has cua.json; then exit 0; fi; ! for k in intellij vscode sublime; do for f in "$k.json" "$k.mac.json"; do has "$f" || echo "missing $f"; done; done | grep .`
  - *Reference:* Editora `src/main/resources/com/editora/keymaps/`. Termina ships the terminal's two files only (`keymaps/`). The app template ships all five, each IDE keymap with its `.mac.json` (`PRESETS` in `command/Keymap.java`). Mirada ships Emacs and CUA and none of these three yet; Torrenta, Modula and nux ship no keymap file.
- **KBD-21** `MUST` A keymap's `.mac.json` variant binds the same commands as the keymap it varies. A chord may differ, to put `Cmd` where the other has `Ctrl`, to follow the macOS keymap of the editor it copies, or where macOS keeps the chord for itself (`Cmd+Space`, `Cmd+Tab`). See Same on every OS.
  - *Why:* a command with no chord on one OS is a command that user never finds from the keyboard.
  - *Check (code):* each keymap file and its `.mac.json` variant bind the same set of command ids: no command is bound in one and missing from the other. An app that ships no keymap files has no pair to compare and fails with KBD-09. A test that loads both and compares the commands they bind is the surest evidence.
  - *Reference:* Editora `KeymapsTest` (`guiKeymapsBindSameCommandSetOnBothPlatforms`), Termina's `KeymapTest` (`bothPlatformsBindTheSameCommands`) and the app template's `KeybindingsTest` (`cuaHasAMacVariantWhereCmdReplacesCtrl`). Mirada's `cua.json` and `cua.mac.json` bind the same commands, with no test that compares the two.
- **KBD-11** `MUST` Keymaps are applied in layers: base keymap → user overrides → plugin keymaps → the user's own bindings again. An app without plugins has the first two.
  - *Why:* the user always has the last word, and a plugin can still use a chord the user freed.
  - *Check (code):* one place applies the layers in that order.
  - *Reference:* Editora `ui/KeymapLayers.java`.
- **KBD-12** `SHOULD` A keymap editor in Settings lists every command with its chord, is searchable, and warns of conflicts.
  - *Check (by hand):* open Settings → Keymap, search, and bind a chord that is taken.
  - *Reference:* Editora `ui/SettingsKeymapsPage.java`; conflicts from `KeybindingEdits.conflicts`, and the defaults a chord would take from `EditorSettingsCoordinator.takenDefaults`.
- **KBD-13** `SHOULD` No command's chord is written into translated text. Every other label that shows a chord (a hint, the palette, the status bar) shows the binding from the active keymap; tooltips are TBAR-03 and menu rows MENU-15. Fixed keys of a control (Enter, Tab, Esc, arrows) and keys meant for another program may be written.
  - *Check (code):* no message catalog value names a command's chord (`Ctrl+`, `Cmd+`, `C-x`, `M-x`); chords are rendered by one formatter.
  - *Reference:* Editora `command/ChordFormat.java`, the one place a binding is rendered. Editora's `command.edit.universalArgument` title still carries `C-u`.
- **KBD-14** `SHOULD` A multi-key prefix (`C-x …`) shows a which-key style hint listing the possible next keys.
  - *Check (by hand):* press `C-x` in the Emacs keymap and wait.
  - *Reference:* no app lists the next keys yet. Editora echoes only the pending prefix (`C-x -`) in the status bar (`command/KeyDispatcher.java`).



Keymap: Select… offers Emacs, CUA, Sublime Text, Visual Studio Code and IntelliJ IDEA.



M-x kill: each command shows its chord from the active keymap.



Termina's shortcuts reference: every key, grouped by menu.

Focus and accessibility

- **KBD-15** `MUST` Every interactive control is reachable with Tab and operable with Space or Enter. A control is the exception when the keyboard does what it does another way: the buttons of a toolbar stay out of the Tab order (TBAR-02, which says what a toolbar is: a bar of labeled buttons is not one, and its buttons are held to this rule), and the find-bar and filter controls that FIND-16 names may leave it. See Accessibility.
  - *Why:* a control that Tab can't reach is one a keyboard-only or screen-reader user can't use at all.
  - *Check (by hand):* first in the source, find every control the keyboard cannot focus: one the app takes out of the Tab order (`setFocusTraversable(false)` in the code, `-fx-focus-traversable: false` in a stylesheet), and a label or a pane that acts on a click. The keyboard does all that each one does another way: it is a button of a toolbar as The toolbar defines one (TBAR-02; a bar of labeled buttons, such as nux's, is not one), a find-bar or filter control that FIND-16 lets leave the Tab order (one that fails FIND-16, such as an option's toggle that no command toggles, fails here too), a list whose text field moves its selection with the arrow keys, or a control whose every action, the rows of its right-click menu included, a registered command also runs. A control only the mouse can work fails the rule, and the source alone shows it. Then in the running app, Tab reaches every other control of each window and dialog, and Space or Enter works it.
  - *Reference:* the app template, where each control the keyboard cannot focus has its other way: the status bar's echo area, error count and update notice run commands (`view.messageLog` , `help.openNewRelease` : `ui/StatusBar.java`), its toolbar's buttons are TBAR-02's (`Icons.toolbarButton` in `ui/Icons.java`), a tool window's header buttons run commands (`ui/ToolWindowManager.java`), the find bars' buttons are FIND-16's (`ui/FindBar.java`), and the picker's list is moved from its field (`ui/Picker.java`). Editora doesn't hold the rule everywhere yet: the Clear button of its Message Log (`ui/MessageLogPopup.java`) and the Previous Run and Save Output buttons of its build console (`ui/BuildToolPanel.java`) are out of the Tab order with no command and no key behind them. Modula doesn't either: it takes every control of its window out of the Tab order (`RadioPane.keepOutOfFocus`) and gives most of what they do a key or a palette row, but a preset's Rename and Remove are rows of its chip's right-click menu alone (`ui/PresetBar.java`). Termina doesn't either: the regular-expression toggle of its find bar is out of the Tab order, and no command and no key toggles it (`ui/SearchBar.java` ; FIND-16 fails for it too).
- **KBD-16** `MUST` Every button, toggle, menu button, link, combo box, check box, radio button and text field shows the focus ring when the keyboard focuses it. The ring is one layer of `:focus-visible` rules in the app stylesheet, after every rule that sets a border or background on the same controls, since rules of equal specificity are decided by order. Its color and contrast are ICON-12.
  - *Why:* a keyboard user who can't see where focus is can't use the window; the ring's color is a `MUST` , and so is drawing it.
  - *Check (code):* the app stylesheet holds the ring as one block of `:focus-visible` rules that covers each of these kinds of control the app uses (buttons, toggles, menu buttons, links, combo boxes, check boxes, radio buttons, text fields), and no rule after the block, of the same or higher specificity, sets again what the block sets on those controls (the border; on a text field, the background insets). A test that tabs to each control family and asserts the ring is the surest evidence.
  - *Reference:* Editora `app.css` (*Keyboard focus*, one layer for every control), held by `FocusRingFxTest` , which tabs to each control family. Termina's `app.css` and Mirada's `mirada.css` end with a layer of the same kind; neither app has a test that tabs to a control.
- **KBD-22** `SHOULD` The focus ring is 2 px wide at the default zoom and sits just outside the control with a 1 px gap: a border with negative insets, which adds nothing to the control's own, so focusing a control never moves the layout. Two kinds of control draw it inside instead, at the same 2 px and still without moving anything: a text field, whose own focus border is widened, and a control flush with the window's edge, where a ring outside would be cut off.
  - *Why:* a ring drawn on the control's own border is lost on a flat or filled button, and one that takes room makes the row jump as Tab moves along it. A text field already draws an accent border on focus, so a second ring around it would be one too many.
  - *Check (code):* in the `:focus-visible` rules of KBD-16, the one for the controls that take the outside ring (buttons, toggles, menu buttons, links, combo boxes, the boxes of check boxes and radio buttons, as many of them as the app has) sets `-fx-border-width: 2` and `-fx-border-insets: -3` ; the one for text fields widens the field's focus border to 2 px instead (`-fx-background-insets: 0, 2px`); and a rule that draws the ring inside a control at the window's edge keeps the 2 px width and takes the room back out of the control's padding.
  - *Reference:* Editora `app.css` (*Keyboard focus*), Termina's `app.css` and Mirada's `mirada.css` : the same outside ring, and the same text-field rule (Mirada's covers text areas too). Editora alone has the second exception: `.tool-stripe-button:focus-visible` draws the ring at `-fx-border-insets: 1` and gives 3 px of padding back, since the stripes sit against the window's edge. The app template's `app.css` has the same outside ring and text-field rule; its check boxes and radio buttons take the ring around the whole control, label included, not around the box alone.
- **KBD-18** `SHOULD` A row that turns a setting on or off (in Settings, a panel or a dialog) uses a switch the keyboard operates: it takes focus, Space toggles it, and its accessible role is `TOGGLE_BUTTON` .
  - *Why:* a bare AtlantaFX `ToggleSwitch` is mouse-only.
  - *Check (code):* every row that turns a setting on or off uses a switch that takes focus, toggles on Space and sets `AccessibleRole.TOGGLE_BUTTON` (Editora has one class for it, `SettingSwitch`); a row that uses AtlantaFX's `ToggleSwitch` as it comes fails.
  - *Reference:* Editora `ui/SettingSwitch.java` .

Related rules on other pages: every icon-only button gets its name from one helper, TBAR-01; rows wrap instead of being cut short in every language, INTL-04; readable text and its contrast are one rule, ICON-09.

Accessibility

An app that needs a mouse, good color vision or steady eyes shuts people out. Much of what keeps the apps usable without them is already a rule on the page it belongs to; this page lists those and adds what a screen reader and the OS's accessibility settings need.

Where JavaFX stands

JavaFX hands its controls to the screen reader on Windows (UI Automation, which Narrator reads) and on macOS (VoiceOver). On Linux it has no bridge to the desktop's accessibility bus, so Orca reads nothing. That is JavaFX's limit, not the app's, and one of the two accessibility differences between OSes; the other is the high-contrast theme, which only Windows' high-contrast mode asks for (ACC-09). Both are rows of Same on every OS.

The standard controls (`Button` , `Label` , `TextField` , `CheckBox` , `ListView` , `TreeView`) announce their name, role and state on their own. What the app builds itself does not: a row drawn from an icon and two labels, a status-bar segment that acts as a button, a map painted on a `Canvas` . The rules below are about those.

The menus have no mnemonics (`_File`): `Alt` and `Option` chords belong to the keymaps (`M-x` in Emacs). The keyboard reaches every menu command through the Command Palette (KBD-05), which lists them all (PAL-02).

Covered on other pages

- **Keyboard.** Every control is reachable with Tab and works with Space or Enter (KBD-15); toolbar buttons stay out of the Tab order because their commands are reached another way (TBAR-02). Closing an overlay or a panel gives focus back where it was (PAL-07, QOPEN-07, MENU-08, TABS-19).
- **Focus ring.** One ring, drawn on every control (KBD-16), at 3:1 or better on every surface (ICON-12).
- **Names.** Every icon-only button gets its name from one helper (TBAR-01); an on/off row is a switch with the `TOGGLE_BUTTON` role (KBD-18).
- **Contrast.** Readable text at 4.5:1 (ICON-09, ICON-10, ICON-11), each held by a contrast test, and those tests run without the toolkit (ICON-14).
- **Size and theme.** A UI zoom scales the whole chrome (WIN-12) and the content zooms from the keyboard (WIN-09); the light or dark theme can follow the system (ICON-16).

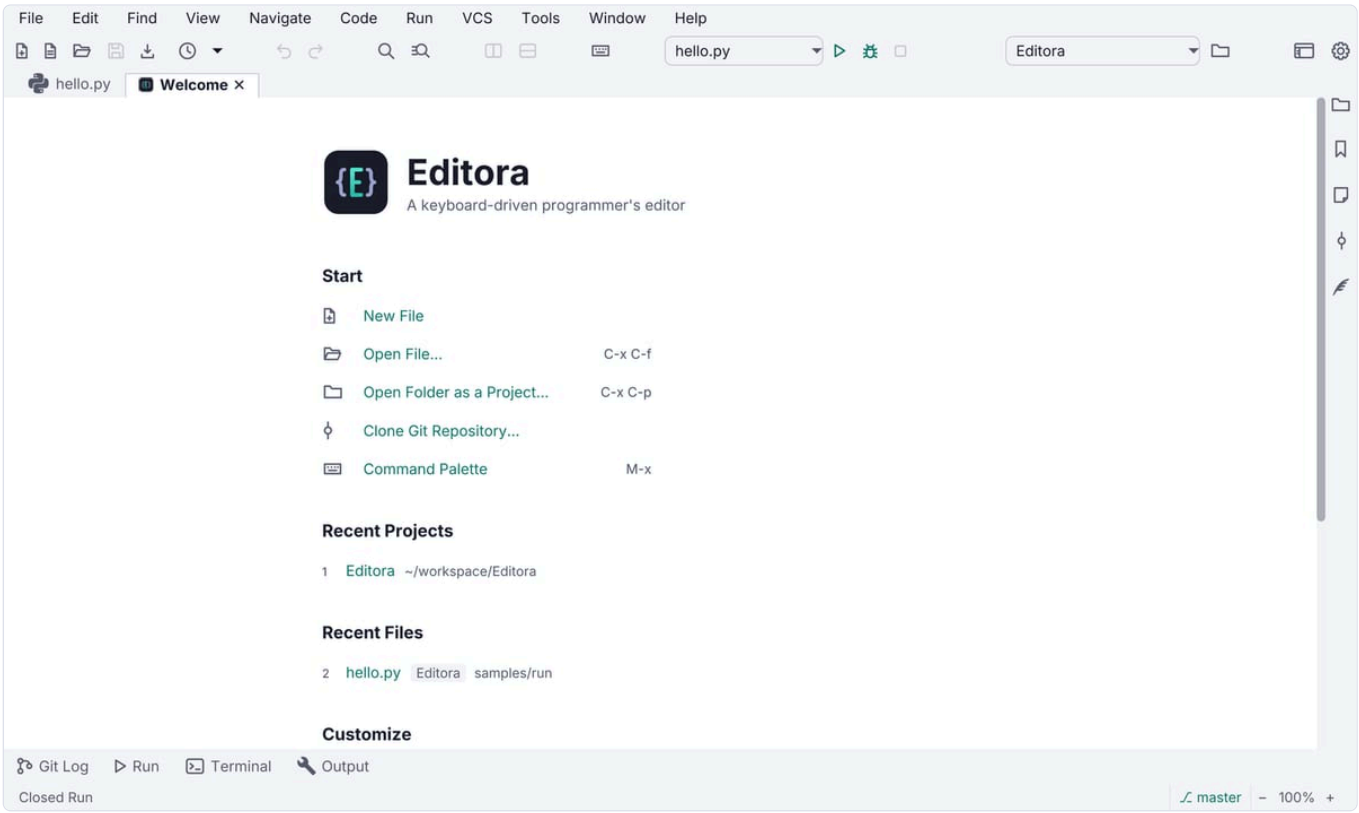
Rules

- **ACC-01** `MUST` Every control a person can reach with Tab or the mouse has an accessible name: its own text, a `Label` whose `labelFor` is the control, or `setAccessibleText` . A row drawn from a graphic (a `CustomMenuItem` , a list cell that lays out its own nodes) carries its text too. A test holds it.
  - *Why:* a screen reader announces an unnamed control as "button", or says nothing.
  - *Check (test):* an FX test opens each window and dialog and fails on a control (a button, a toggle, a field, a combo box or any other `Control` a person can focus or click, in the Tab order or out of it) that has no text, no accessible text and no label pointing at it, and on a list cell or menu row that reads as nothing. JavaFX names a `CustomMenuItem` row from its content's accessible text, and a cell from its text or its graphic's, so a row that lays out its own nodes sets `setAccessibleText` on them.
  - *Reference:* Editora names its controls area by area (`NamedControlsFxTest`), and sets each row's text on the macOS system menu bar. Its in-window command rows, each a `CustomMenuItem` that draws an icon, a title and a chord (`ui/MainMenuBar.java`), carry no text and no accessible text yet; its submenus keep their own. The app template has the test that sweeps every window for every control: `NamedControlsFxTest` (`everyControlInEveryWindowHasAName` , and `everyMenuRowReadsAsItsTitleAndWearsAGlyph` for the menu rows, which `AppMenus.inWindowRow` names from their title and chord) opens the main window and its panels, the find bars, the Message Log, Settings, About, the Pauta window, the Debug Log and Report a Problem, the palette, the switcher, a question and a prompt. Termina's `IconOnlyButtonsFxTest.java` comes nearest: it opens the main window, the menus, Settings and About and fails an icon-only button with no name, and no other kind of control. Torrenta's one call to `setAccessibleText` is in a helper (`ui/Icons.java` , `name`) that names every button of its main toolbar and the Search view's Clear button, and no other control; Modula never calls it.
- **ACC-02** `SHOULD` A control the app builds from panes or draws itself says what it is and does what its role offers: `setAccessibleRole` with the role it acts as (`BUTTON` , `TOGGLE_BUTTON` , `LIST_VIEW` , `TREE_ITEM`), its state answered from `queryAccessibleAttribute` (selected, expanded, checked), `notifyAccessibleAttributeChanged` when the state changes, and the role's actions (`FIRE` , `EXPAND` , `COLLAPSE`) handled in `executeAccessibleAction` .
  - *Why:* without a role, a screen reader can't tell a status-bar segment that opens a picker from a label; and it presses a control through its action, not a mouse click, so a `BUTTON` that ignores `FIRE` can be found but not used.
  - *Check (code):* every `Pane` , `Region` , `Label` or `Canvas` that handles a click or a key sets an accessible role and overrides `executeAccessibleAction` for that role's actions.
  - *Reference:* Editora `ui/SettingSwitch.java` (the `TOGGLE_BUTTON` role; it handles `FIRE` only) and `ui/MapAccessibleRows.java` (`FIRE` , `EXPAND` and `COLLAPSE` on each row). Its status-bar segments (`ui/StatusBar.java`) take the `BUTTON` role but don't handle `FIRE` yet.
- **ACC-03** `SHOULD` An app that paints content on a `Canvas` (a map, a strip of images, a terminal grid) gives it to the screen reader: an accessible text that sums it up and, where a person moves through items in it, one accessible node for each item on screen.
  - *Check (code):* the canvas, or the node that stands for it, sets an accessible text, and where a person moves through items it exposes a node with a role and a name for each item on screen. A test that reads the accessible text, or the roles and names of those nodes, is the surest evidence.
  - *Reference:* Editora `ui/MapAccessibleRows.java` (the Project Map's rows on screen, as `TREE_ITEM` nodes under a `TREE_VIEW`), read by `MapAccessibleRowsFxTest` .
- **ACC-04** `SHOULD` Motion follows the OS: while `Platform.getPreferences().isReducedMotion()` is true, an animation jumps to its end, and a change to the setting applies at once.
  - *Why:* sliding and fading can make a person dizzy or sick, and they turned motion off in the OS for that reason.
  - *Check (code):* every `Transition` , `Timeline` or `AnimationTimer` that moves, fades or flashes part of the UI starts through one helper that reads `reducedMotionProperty()` .
  - *Reference:* no app reads the setting yet. The app template animates nothing, so it has nothing to judge: its timelines and pauses are timers that move, fade or flash no part of the UI (the Output sample's lines in `ui/SamplePanels.java` , the debounce of `ui/Debounce.java` , the sync interval and the update check's delay). Mirada fades its on-screen controls in and out (`ui/OnScreenControls.java` , a `FadeTransition`), and Editora's Project Map flashes a revealed row for 900 ms (`ui/MapSurface.java` , an `AnimationTimer`); neither reads a motion setting.
- **ACC-05** `SHOULD` Color is never the only signal: a state shown in color (an error, a changed line, a status) also shows a word, a glyph or a shape.
  - *Why:* about one man in twelve can't tell red from green.
  - *Check (by hand):* look at a grayscale screenshot of each window and find every state still told apart.

- *Reference:* Editora's echo area colors an error and also says what failed (`ui/StatusBar.java`; WIN-01). The app template gives a file's Git status its letter beside the color, on a tab (`ui/TabHeaders.java`; `TabsAndPanelsFxTest`, `aTabShowsItsGitStatusAsALetter`) and on the file's row in its Documents window (`ui/SamplePanels.java`).

- **ACC-06** `SHOULD` In an app whose window has more than one region (the content and at least one panel beside it), a command moves focus to the next region, another moves it to the previous one, and every keymap binds both.
  - *Why:* Tab walks through every control inside a region; with several panels open, a keyboard user needs a way to jump between them.
  - *Check (code):* the two commands are registered and bound in each bundled keymap.
  - *Reference:* the app template's `window.focusNextRegion` and `window.focusPreviousRegion` (`ui/Commands.java`, `focusRegion` in `ui/ToolWindowManager.java`), on `F6` and `Shift+F6` in its CUA keymaps and `C-x o` and `C-x S-o` in Emacs, bound in every keymap it ships (`TabsAndPanelsFxTest`, `theRegionCommandsWalkTheDocumentsAndTheOpenToolWindows`). Editora's `window.other` ("Cycle focus between the editor and tool windows", `ui/WindowCommandRegistrar.java`), one command that cycles, bound only in the Emacs keymap (`C-x o`, `keymaps/emacs.json`). `F6`, the Windows convention, is taken in every other Editora keymap: CUA and VS Code bind it to `debug.pause`, IntelliJ to `lsp.move` and Sublime to `view.toggleSpellCheck`.
- **ACC-07** `SHOULD` Before a release, a person makes a pass with the screen reader on Windows and on macOS: with Narrator and with VoiceOver, open the Command Palette, run a command, and move through Settings, hearing each control's name and role. It is a step of its own in `docs/release.md` (REPO-07): the smoke test of REL-29 is a CI step, with nobody there to listen.
  - *Check (by hand):* `docs/release.md` has the step, naming Narrator and VoiceOver, and someone made the pass for this release. The file settles the first half: with no such step the rule is unmet, whoever listened. The pass itself is a person's to record, as a `checked` entry that carries the app's `version` (PAUTA-18).
  - *Reference:* no app yet. No app's `docs/release.md` names either screen reader, and Modula and nux have no `docs/release.md`.
- **ACC-08** `SHOULD` In an overlay list that keeps focus in its text field (the Command Palette, a picker: QOPEN-01), the row the arrow keys select is announced, with its state: the list tells the screen reader its focused item moved (`notifyAccessibleAttributeChanged(FOCUS_ITEM)`), and a row that can't run says so in its accessible text.
  - *Why:* the arrows move the highlight, not the focus, so a screen reader stays on the text field and reads nothing as the user moves through the list.
  - *Check (code):* the code that moves the selection calls `notifyAccessibleAttributeChanged(AccessibleAttribute.FOCUS_ITEM)` on the list, the list answers `FOCUS_ITEM` with the selected row, and a row that can't run says so in its accessible text.
  - *Reference:* the app template's `ui/Picker.java`, behind its palette and pickers: `select` moves the list's focus model and calls `notifyAccessibleAttributeChanged(FOCUS_ITEM)`, and a row's accessible text is its title, its keys and, for a row that can't run, the reason. Editora's palette rows are graphics with no text (`ui/CommandPalette.java`).
- **ACC-09** `SHOULD` In Windows' high-contrast mode the app switches to a high-contrast theme, and back when the mode ends: it follows `Platform.getPreferences()`'s `Windows.SPI.HighContrast`.
  - *Why:* JavaFX applies its own high-contrast stylesheet only over its built-in Modena theme, which AtlantaFX replaces, so without this the app ignores a setting the person turned on to read the screen at all.
  - *Check (code):* a listener on the preference switches the theme, and the theme list has a high-contrast theme.
  - *Reference:* the app template's `ui/Themes.java` (`WINDOWS_HIGH_CONTRAST`, a listener on `Platform.getPreferences()` that applies its `HIGH_CONTRAST` theme while the mode is on and the chosen one when it ends), held by `StyleContrastTest` (`theOsHighContrastModeWinsWhileItIsOn`, `theHighContrastThemeReadsAtSevenToOne`). No app does it yet.

Window shell



Editora's Welcome page: Start actions with their chords, Recent Projects, Recent Files and Customize.

The window shell is what every app has around its content: the status bar and its echo area, the menu bar in the title bar, dialogs and the order of their buttons, the window's size and place kept across launches, zoom, and Quit. A user meets these parts before any of the app's own features, so they look and work the same in every app, and on every OS apart from the places Same on every OS lists (the macOS application menu among them). The Welcome page and the About window have templates of their own.

- **WIN-01** `MUST` The main window has a status bar, shown by default; the user may hide it. Its left side is the *echo area*: one line of feedback for the last action, errors in red and warnings in amber. Clicking it opens the Message Log (DIAG-01).
  - *Check (code):* the main window builds a status bar that is visible unless the user hid it; the echo label switches between an error and a warning style class, and its click handler runs the Message Log command.
  - *Reference:* Editora `ui/StatusBar.java` (`status-echo-error` , `status-echo-warn` ; a click runs `view.messageLog`) and the Message Log template. Editora hides and shows the bar with `view.toggleStatusBar` and a setting, and Zen mode hides it (`ui/Chrome.java` , `statusBar`). Termina doesn't meet the rule yet: its status bar is a setting that starts off (`config/Settings.java` , `showStatusBar`).
- **WIN-03** `MUST` In an app with a menu bar, the menu bar sits in the title bar by default on Windows and Linux: JavaFX 27's `HeaderBar` with `StageStyle.EXTENDED` carries the menu, and a setting that starts on lets the user fall back to a decorated window. On macOS the menus are in the system menu bar (`useSystemMenuBar`), and the window is decorated, or extended with a header that holds no menus.
  - *Why:* the family looks the same on every OS only if every app starts with the same title bar; an app that ships it off looks like another family.
  - *Check (code):* on Windows and Linux the main stage is created `StageStyle.EXTENDED` with a `HeaderBar` that holds the menu bar, and the setting that falls back to a decorated window defaults to on (in the title bar); on macOS the menu bar sets `useSystemMenuBar` , and a header, where the window has one, holds no menus.
  - *Reference:* the app template (`config/Settings.java` , `titleBarMenus = true` , read by `Chrome.usesHeaderBar`). Editora (`ui/ExtendedWindow.java` , `ui/MainMenuBar.java`), Torrenta and Termina have the setting but start it off. Mirada has no setting: its `ui/TitleBar.java` is a `HeaderBar` on a `StageStyle.EXTENDED` stage wherever JavaFX supports one, on macOS too. Its header holds the file name and buttons, the main menu's among them, and no menu bar; on macOS `ui/MacMenus.java` puts the menus in the system menu bar. Modula and nux have no menu bar.
- **WIN-04** `SHOULD` Native dialogs go through one helper that attaches the app stylesheet. The UI font reaches every scene, a dialog's included, by ICON-01.
  - *Why:* a dialog is its own scene and does not inherit the window's stylesheet.
  - *Check (code):* every `Alert` and `Dialog` the app builds passes through the helper.
  - *Reference:* Editora `ui/Dialogs.java` (`Dialogs.styled`), which attaches `app.css` only. The UI font reaches an Editora dialog another way: `App.installUiFont` adds `ui-font.css` to every window's scene (ICON-01).
- **WIN-11** `MUST` A dialog's buttons are in the same order on every OS, with Cancel to the left of every action, as the Confirm dialog draws them (Cancel, Save and Discard where there are three: CONF-05): the dialog's `ButtonBar` gets one fixed order, never JavaFX's default for the OS. See Same on every OS.
  - *Why:* JavaFX orders a `ButtonBar` by the OS it runs on (`BUTTON_ORDER_WINDOWS` , `_MAC_OS` , `_LINUX`), so the same confirmation puts its action on the left on Windows and on the right on macOS and Linux.
  - *Check (code):* the dialog helper (WIN-04), or each dialog, calls `setButtonOrder` on the dialog pane's `ButtonBar` with one order string, the same on every OS, in which `C` (`CANCEL_CLOSE`) comes before the letter of every action button the dialogs use (`O` for `OK_DONE` , `Y` , `N` , `A` , `U` for `OTHER`). `BUTTON_ORDER_WINDOWS` on every OS fails: it puts `Y` , `N` and `O` before `C` .
  - *Reference:* the app template's `ui/Dialogs.java` (`BUTTON_ORDER` , `L_HE+CUYNBXIAO_R` , set by `styled` , which every dialog of the template goes through), held by `DialogsFxTest` (`everyDialogHasOneButtonOrderAndCatalogWords`). Editora's Git confirmations give their buttons the `OK_DONE` and `CANCEL_CLOSE` roles and leave the order to JavaFX.

- **WIN-08** `SHOULD` The main window's size, position and maximized state are restored on the next launch, and in an app that opens documents or sessions, so are the items that were open. `--no-session` skipping the restore is CLI-04.
  - *Check (code):* closing the window saves its size, position and maximized flag, and the open items in an app that has them, and the next launch reads them back and applies them. A test that saves the state and reads it back is the surest evidence.
  - *Reference:* Editora `config/WorkspaceState.java` (`windowX` … `windowMaximized`), `ui/WindowSessionCoordinator.java` , `App.noSessionFlag` ; tests `AppArgsTest` , `NoSessionStartupFxTest` and, for the window's geometry, `WindowManagerLaunchFxTest` (`aWindowComesBackAtTheSizeAndPlaceItsSessionRecorded` , `aWindowThatWasMaximizedComesBackMaximized`), which restore a window from a saved `workspace-state.json` . No Editora test saves the geometry and reads it back. The app template does both: `saveWindowState` in `ui/MainWindow.java` writes the size, place, maximized state and the open documents to `session.json` , and the next launch restores them (`ui/DocumentTabs.java` ; `SessionAndWelcomeFxTest` , `theOpenDocumentsComeBackFromTheSessionFile`).
- **WIN-09** `SHOULD` In an app with content to zoom (text, an image, a terminal), it zooms from the keyboard (`Ctrl+=` , `Ctrl+-` and `Ctrl+0` , with `Cmd` on macOS; `=` is the key that carries `+`), kept across launches. The UI zoom is WIN-12.
  - *Check (code):* the content's zoom in, out and reset are commands bound in every keymap, and the zoom level is a saved setting.
  - *Reference:* Editora `view.textZoomIn` , `view.textZoomOut` and `view.textZoomReset` , kept in `Settings.fontZoom` and bound to `C-=` , `C--` and `C-0` (`keymaps/cua.json`); Termina's `zoomIn` , `zoomOut` and `actualSize` , on `Shortcut+Equals` , `Shortcut+Minus` and `Shortcut+0` (`keymaps/keymap.json`), kept in `font.zoom` (`config/Settings.java`); the app template's `view.zoomIn` , `view.zoomOut` and `view.zoomReset` for its documents' text, bound to `Ctrl+=` , `Ctrl+-` and `Ctrl+0` (`Cmd` on macOS) in every keymap it ships and kept in `textZoom` (`config/Settings.java` ; `SessionAndWelcomeFxTest` , `theDocumentsTextZoomsOnItsOwn`).
- **WIN-12** `MUST` The app has a UI zoom: one setting scales the whole chrome, the menus (MENU-16), toolbar, tabs and status bar included. It applies at once, is kept across launches, and three commands zoom it in, out and back to the default. In an app with content of its own to zoom (an editor's text, a terminal's grid, an image) `Ctrl+=` , `Ctrl+-` and `Ctrl+0` (`Cmd` on macOS) stay with the content zoom (WIN-09) and this rule fixes no chord for the UI zoom's commands; in an app without, those chords are the UI zoom's.
  - *Why:* a person who needs larger text gets it everywhere, and the menus can only follow a zoom that exists.
  - *Check (code):* the chrome's font sizes come from one root size the setting changes, and its fixed sizes are multiplied by the zoom, not written as absolute pixels. A change to the setting restyles every open window with no restart, and the value is saved with the settings and read back at start-up. Three registered commands zoom it in, out and back to the default. The rule names no ids for them: they are the app's own. In an app with a content zoom they are three more commands, with ids that are not the content zoom's (Mirada's `view.zoomIn` and Termina's `zoomIn` zoom the content), and the keymaps bind `=` , `-` and `0` with `Ctrl` (`Cmd` on macOS) to the content zoom's commands; in an app without, to the UI zoom's.
  - *Reference:* the app template's `ui/UiZoom.java` and `ui/Themes.java` (`zoom`): one setting, `uiZoom` in `config/Settings.java` , scales every pixel size of the stylesheets, the root font size among them, and the sizes the code fixes are bound to the zoom; a change restyles every open window at once. Its commands are `view.uiZoomIn` , `view.uiZoomOut` and `view.uiZoomReset` (`ui/Commands.java`), on `Ctrl+Alt+=` , `Ctrl+Alt+-` and `Ctrl+Alt+0` (`Cmd+Alt` on macOS) in its CUA keymaps, which leave `Ctrl+=` , `Ctrl+-` and `Ctrl+0` to its documents' text zoom (WIN-09); `UiZoomTest` holds the scaling. Editora, Termina and Mirada zoom their content only (`view.textZoomIn` , `zoomIn` , `view.zoomIn`), and Editora writes its menu font as absolute pixels (`MainMenuBar.UI_FONT_SIZE` , `.context-menu` in `app.css`). Torrenta, Modula and nux have no zoom command.
- **WIN-13** `MUST` Every quit entry point an app has runs one quit method: the Quit row and the `app.quit` command (WIN-14 says which of them an app must have), and the system's own Quit, which on macOS is the application menu's and the Dock's. That method asks what the Confirm dialog rules require (CONF-01), closes open dialogs and runs the save the app runs when its window closes, so quitting loses nothing a close would have kept; what an app keeps across launches is WIN-08. A tray's Quit is TRAY-04.
  - *Why:* a second way out skips what the first one does: the question about unsaved work, the saved window, or, in a tray app, the quit itself. Quitting from inside a dialog's nested event loop has aborted the JVM.
  - *Check (code):* one quit method. The Quit row and the quit command run it, where the app has them (an app with neither fails WIN-14, not this rule), and on macOS the system's Quit reaches it. JavaFX turns the application menu's and the Dock's Quit into a close request on every window (`GlassStage.requestClosingAllWindows`) unless the app catches Glass's quit action (`handleQuitAction`), so the quit handler calls the quit method, or, in an app with one main window, that window's close request does. In an app with one main window and no quit command, what the close request runs is the quit method, and it is judged as one; an app with several main windows whose close requests each ask for themselves has no quit method. The method itself asks before it discards anything CONF-01 covers (unsaved work; an app with nothing to lose asks nothing), closes its open dialogs before it exits, and runs the same save a close of the window runs (the window state of WIN-08, in an app that keeps it).
  - *Reference:* the app template's `MainWindow.quit` : the `app.quit` command, which File → Quit runs, and the window's close request both call it, so macOS's Quit reaches it as a close request; it hides the other windows and dialogs, saves the window state and waits for the last settings write before it exits, and has nothing unsaved to ask about. That wait is the last save PERF-01 allows as the app quits; the template makes it on a background thread all the same. Editora's File → Quit and `app.quit` run `MainController.onQuit` , which asks about the quit once and then about each window's unsaved files; macOS's Quit doesn't reach it yet: `MacOpenFiles` wraps Glass's handler and passes `handleQuitAction` on to JavaFX's own, so each window gets a close request and asks for itself. Torrenta (`MainWindow.quit`), Modula (`ModulaApp.quit`) and nux (`Platform.exit` , from `app.quit` and from the tray, in `App`) don't catch macOS's Quit either. Termina and Mirada have no quit command (WIN-14). Mirada has one main window, and what its close request runs is its quit method: it asks about an unsaved edit, closes its tool windows and saves the window's place (`ui/ViewerWindow.java` , `mayClose` , `release` , `rememberPlace`), and macOS's Quit arrives there as a close request. Termina doesn't meet the rule yet: it has several windows and no quit method, so macOS's Quit arrives at each window as a close request and each asks for itself (`TerminalWindow.close`).
- **WIN-14** `MUST` Every app has a Quit, on every OS: the `app.quit` command, in the Command Palette, and, where the app has a File menu or a main menu, a Quit row in that menu that runs the command. A main menu is a menu button in the window that holds the app's commands as a menu bar would (Mirada's ☰, which it shows on every OS). A menu button that lists one kind of thing is not a main menu (nux's Profiles and Commands), and neither is a right-click menu (the only menu Modula has): an app with no File menu and no main menu meets the rule with the command alone. What Quit does is WIN-13. Closing the last window may end the app as well, but it is never the only way out.
  - *Why:* someone who quits one app of the family from the palette or the File menu looks for the same in the next. An app that ends only with its last window has no answer once it has a tray or a second window.
  - *Check (code):* `app.quit` is registered and the palette lists it. Where the app builds a File menu, a row of it runs `app.quit` ; where it has no menu bar and a main menu instead, a row of that menu does. An app with neither menu passes on the command.
  - *Reference:* the app template's `Commands.java` (`app.quit`) and `AppMenus.java` (Quit, the File menu's last row); Editora's `app.quit` , the last row of its File menu (`ui/MenuBarModel.java`). Torrenta's File menu ends with Quit, and it registers the command as `quit` . Termina and Mirada have no Quit. Termina's File menu has Close Window and ends with Settings, Sync Now and Configure Profiles (`ui/MenuBarModel.java`); Mirada's ☰ menu, on every OS (`ui/TitleBar.java`), and, on macOS, its File menu (`ui/MacMenus.java`) end with Close (`window.close`). nux has neither menu and meets the rule with `app.quit` (`App`). Modula has neither menu and no quit command: it quits from its tray and its window's close button (`ModulaApp.quit`).
- **WIN-15** `SHOULD` On macOS, Settings is in the application menu, the one macOS names after the app, and runs the same command as the Settings row on Windows and Linux. Its row shows the chord the active keymap binds to that command (`Cmd+,` in the CUA keymap: SET-01), appended to the title as MENU-02 has the system menu bar's rows show theirs, and the item is given no key equivalent of its own. About, in the same menu, is ABOUT-01, and the Quit JavaFX puts there is WIN-13.
  - *Why:* it is where every Mac app has Settings, one of the places the OS's way wins. A key equivalent set on the item is an accelerator under another name: macOS runs the item before the app's key dispatcher sees the key, and the menu goes on showing `⌘,` in a keymap that binds another chord, or none.

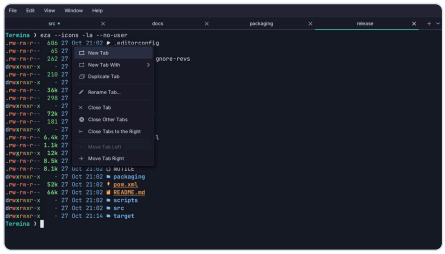
- *Check (code):* on macOS the app inserts a Settings item into the application menu that runs the settings command. The item's title carries the chord the active keymap binds to that command and is written again when the keymap changes; the item is created without a key equivalent.
- *Reference:* Mirada `MacAppMenu.java` adds the item, and it runs `app.settings`. It doesn't show the keymap's chord yet: it creates the item with the native key equivalent `Cmd+,` (`createMenuItem(…, ',', COMMAND)`), which stays whatever the keymap binds, and Mirada's Emacs keymap binds `C-,` (`keymaps/emacs.json`). The app template's `ui/MacAppMenu.java` meets it: it inserts a Settings item that runs `app.settings` , created with no key equivalent, whose title carries the keymap's chord and is written again on a keymap change (`refresh` , from `ui/MainWindow.java`). Termina's `MacAppMenu.java` adds About only. Editora, Torrenta, Modula and nux add nothing to that menu.

- **`WIN-10`** `MAY` The window has modes that change how much chrome it shows: Zen shows none, Expert keeps only the status bar, and Simple shows fewer menus, for new users.
  - *Check (code):* each mode is a command, and one truth table decides which chrome each mode shows.
  - *Reference:* Editora `ui/WindowChromeCoordinator.java` (`toggleZen` , `toggleExpert` , `applySimpleMode`) over the truth table in `ui/Chrome.java` , where Expert hides the menu bar, the toolbar, the tabs, the breadcrumb and the tool stripes (`menuBar` keys on both focus modes; in an extended window the title and the window buttons stay).

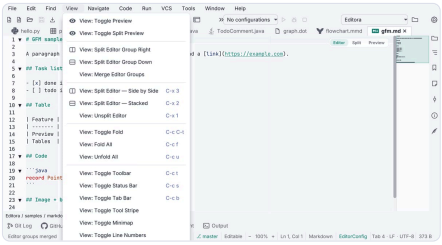
The rest of the window has its own pages. Light and dark themes switched at run time, the "follow the system" choice and what the state colors mean are ICON-15, ICON-16 and ICON-18. A destructive action asks first through the Confirm dialog (CONF-01 to CONF-04). The Welcome page is WELC-01, and the About window is ABOUT-02 and ABOUT-09.



Mirada's HeaderBar title bar carries the file name and folder; the info panel and thumbnails are commands too.



Termina's tab strip is the row under the menu bar; its HeaderBar holds the menu bar and the window title, not the tabs.



Editora's View menu in an earlier build: every row is a command, and a row whose command is bound shows its chord.

Icons and color

One typeface pair, one palette, one color vocabulary and one way of drawing icons, so a tab, a status segment, a tree row and a Settings row say the same thing in the same way in every app. Reference: Editora's `styles/ui-font.css` and `ui/Fonts.java` (type), `styles/atlantafx-themes/editora-light.css` and `editora-dark.css` (the Caret & Ink palette, which Torrenta and Termina vendor), the `.root` block at the top of `styles/app.css` (the state vocabulary), `ui/Themes.java` and `ui/EditorThemes.java` (themes), `ui/Icons.java`, `ui/FileIcons.java`, `ui/MenuBarIcons.java` and `editor/MenuIcons.java` (icons), and `StyleContrastTest`, which pins the color rules.

Type

Inter speaks, JetBrains Mono works. Inter is the chrome: menus, buttons, settings, pickers, status bar. The monospace face is for text that is code or output: the editor, a terminal, the environment block in About, log timestamps.

Property	Value	Reference
UI family	Inter, in four faces, bundled in the app's resources and registered before any CSS is applied	`Fonts.UI`, `Fonts.loadForFirstPaint`, `resources/com/editora/fonts/inter/`
Root size	14 px at the default zoom, the AtlantaFX theme's root size, scaled by the UI zoom (WIN-12); a popup sets its own size from that one value, so an owner's font never leaks into it (MENU-16)	`MainMenuBar.UI_FONT_SIZE`
How it is attached	A scene stylesheet (`ui-font.css`, one `.root` rule whose `-fx-font-family` names Inter) on every window's scene, dialogs and popups included	`App.installUiFont`
Monospace	JetBrains Mono by default; Cascadia Code, Fira Code, IBM Plex Mono and Source Code Pro are bundled as editor choices	`Fonts.DEFAULT`, `Fonts.BUNDLED`

- **`ICON-01`** `MUST` Inter is attached by a *scene* stylesheet on every window's scene, dialogs, popups and tooltips included, not by a rule in the app stylesheet.
  - *Why:* dialogs, the Settings window, the palette, context menus and tooltips each live in a scene of their own that does not share `app.css`, and a scene sheet is re-applied cleanly when the theme switches at run time (`setUserAgentStylesheet`). AtlantaFX neither bundles nor names Inter, and JavaFX does not fall through a font-family list, so without the sheet the whole interface renders in the platform's "System" face.
  - *Check (code):* every window's scene gets the font sheet (Editora's `ui-font.css`): a listener on `Window.getWindows()` adds it to each window's scene, and to a new scene when the window's scene changes, or every window and dialog the app builds adds it itself, none left out. A popup or a tooltip needs no call of its own: when JavaFX shows one, it gives it the scene stylesheets of the window it is shown from (`PopupWindow.applyStylesheetFromOwner`). No `.root` rule in the app stylesheet (Editora's `app.css`) sets `-fx-font-family`.
  - *Reference:* Editora `App.installUiFont`, a listener on `Window.getWindows()`, and `styles/ui-font.css`; the app template's `ui/Themes.java` (`installUiFont`, `hookUiFont`) and Mirada's `ui/Fonts.java` (`installSheet`) do the same. Termina and Torrenta carry the same sheet with no listener: each window and dialog adds it by calling `Fonts.installUiFont`, about forty times in Torrenta. Termina doesn't meet the rule yet: its helper is called for seven windows and `ui/Confirm.java` adds the sheet to its questions, but four dialogs get none: the two alerts and the rename-tab `TextInputDialog` built in `ui/TerminalWindow.java`, and the `TextInputDialog` in `ui/SettingsWindow.java`. Modula doesn't yet: the `.root` rule of its `modula.css` sets IBM Plex Mono for the whole window.
- **`ICON-02`** `MUST` The app bundles Inter's four faces and registers them before any CSS is applied; it never maps the platform's own UI font.
  - *Why:* an earlier macOS-only sheet used San Francisco, which JavaFX faux-bolds; Inter has a real bold on every platform.
  - *Check (code):* `Inter-Regular`, `Inter-Bold`, `Inter-Italic` and `Inter-BoldItalic` `.ttf` are under the app's `fonts/inter/` with their `OFL.txt`, and `Font.loadFont` runs on them before the first window is shown.
  - *Reference:* Editora `src/main/resources/com/editora/fonts/inter/`, `ui/Fonts.java` (`loadForFirstPaint`, called from `App`). Termina and Torrenta bundle the same four faces. Modula doesn't yet: its `fonts/` holds IBM Plex Mono only.

Modula sets everything in IBM Plex Mono today. Its readings may stay monospace, as content; Inter on its menus, settings and dialogs is a gap (Gaps per app).

The palette: Caret & Ink

The flagship theme pair, "Editora Light" and "Editora Dark", is authored (not vendored) and derived from Editora's icon: a **caret** teal accent, **ink** navy as the text in light and the surface in dark, **porcelain** and **abyss** grounds, and **brace** periwinkle reserved for the keyboard. Amber, red and green are semantic and never decorate.

	LIGHT	DARK	TOKEN
Ground	#ffffff	#1b1e2a	-color-bg-default
Porcelain / Abyss	#f4f5f9	#12141d	-color-bg-subtle / -color-bg-inset
Ink	#1b1e2a	#e8eaf3	-color-fg-default
Muted ink	#575d6e	#9ba1b5	-color-fg-muted
Subtle ink	#8b91a3	#6c7288	-color-fg-subtle
Caret	#0e8577	#43dec5	-color-accent-emphasis
Caret, as text	#0d786b	#43dec5	-color-accent-fg
Amber	#8d6309	#e3a93c	-color-warning-fg
Red	#c03434	#ee7a6e	-color-danger-fg
Green	#2c784c	#5bc98a	-color-success-fg
Brace	#3c4380	#b8bee5	-state-key (a ladder of the ground)

- **ICON-04** `SHOULD` A theme file follows the vendored AtlantaFX community themes: a `.root` block of `-color-*` tokens, then component rules that reference only those tokens.
  - *Check (code):* the file's first rule is `.root`, and no rule after it holds a `#` hex.
  - *Reference:* Editora `styles/atlantafx-themes/editora-light.css`, `editora-dark.css`.
- **ICON-05** `SHOULD` A new app ships the Caret & Ink pair as its default theme, with `editora-light.css` and `editora-dark.css` copied unchanged into its own `styles/`.
  - *Why:* the palette is the family's, not the editor's.
  - *Check (code):* the app's two files are identical to Editora's.
  - *Reference:* Termina `styles/` and the app template's `styles/`, whose two files match Editora's. Torrenta's `editora-dark.css` matches too; its `editora-light.css` still has the earlier `-color-accent-fg`, `-color-warning-fg` and `-color-success-fg`, which Editora has since darkened.
- **ICON-06** `SHOULD` Brace, the palette's periwinkle for the keyboard (chord chips, the menus' chord column), is one token on `.root` (Editora's `-state-key`): a `ladder()` over `-color-bg-default` that picks a deep ink on a light ground and a lighter periwinkle the darker the ground gets, and reads at 4.5:1 or more on every bundled theme's default and subtle surfaces.
  - *Why:* Brace has no AtlantaFX role, so it has to be a literal.
  - *Check (code):* the token is a `ladder()` of `-color-bg-default`, and the color it resolves to on each bundled theme's default and subtle grounds is 4.5:1 or more against that ground. A contrast test that resolves the ladder against every bundled theme's grounds is the surest evidence.
  - *Reference:* Editora `app.css` (`.root`), `StyleContrastTest.theKeyboardPeriwinkleReadsOnEveryBundledAppTheme`.

State colors

A color is a meaning, named once on `.root` and referenced everywhere. The same state wears the same color in a tab, a status segment, a tree row, a picker and a Settings row.

Token	Chains to	Meaning	Where it appears
`-state-amber`	`-color-warning-fg`	Not durable yet, a warning, or an update available in About	An unsaved tab, a read-only file, a narrowed view, a snapshot build; warnings in the echo area and the Message Log; "Update available" in About (ABOUT-10)
`-state-red`	`-color-danger-fg`	Broken	Errors in the echo area and the Message Log, breakpoints, a recording
`-state-green`	`-color-success-fg`	Verified	Found, added, passing, running
`-state-accent`	`-color-accent-fg`	The app is offering something	An update, a link, the current branch, a selected row's bar
`-state-olive`, `-state-violet`	A ladder of the ground	Git only: untracked, renamed	The Project tree and tabs, the Map's structure columns
`-state-key`	A ladder of the ground	The keyboard, and nothing else	Chord chips, the menu chord column, brace matching
`-state-sel`	`-color-accent-subtle`	The selected overlay row	The palette, pickers, the Message Log (about a 13 % accent wash)

Each of the first four also has a `-bg` (the role's `-subtle`) and a `-hover` (the role's `-muted`) variant, so a pill, a row tint and a hover state come from the same place. `.state-pill` is the shape (bold, radius 999, padding 1 × 8); the color variant beside it carries the meaning.

- **ICON-07** `MUST` Each state color is named once, as a token on `.root` (Editora's `-state-*`) that chains to one of the theme's roles (or a `ladder()` of its ground), and component rules use the token.
  - *Check (code):* each state color has one token (Editora's `-state-*`), declared once, in a `.root` block of the app stylesheet, as a `-color-*` role or a `ladder()` of the ground. A component rule that colors one of those states uses the token: none names the role behind it (`-color-danger-fg` for an error) or writes the ladder out again.
  - *Reference:* the app template's `styles/app.css` (`.root`), held by `StyleContrastTest` (`theStateTokensAreDeclaredOnceAtTheTopAndChainToARole`, and `severityColorsComeFromTheStateTokens`, which fails a component rule that names a danger, warning or success role). Termina meets it too: its `styles/app.css` declares `-state-red`, `-state-amber` and `-state-green` on `.root`, and no component rule of its stylesheets names the role behind one. The other five apps don't yet. Editora's `styles/app.css` declares its tokens the same way (`.root`), but Editora doesn't hold the last clause: 79 declarations in that file's component rules name `-color-danger-fg`, `-color-warning-fg` or `-color-success-fg` directly, errors among them (`.crontab-error`, `.md-math-error`). Torrenta's tokens are on `.root` too (`-torrenta-*`), and two of its component rules write out the ladder of `-torrenta-up` again (`styles/app.css`, `.speed-graph-up`, `.speed-graph-legend-up`). Torrenta doesn't hold the last clause either: 24 declarations in that file's component rules and five in `add-dialog.css` name `-color-danger-fg`, `-color-warning-fg` or `-color-success-fg` directly (`.port-dot.port-down`, `.add-fetch-failed`), not the `-torrenta-error`, `-torrenta-stall` and `-torrenta-done` tokens that chain to them. Mirada and nux have no token for a state (Mirada's one `-state-*` token is `-state-key`, the keyboard's): Mirada's `styles/mirada.css` redefines the roles themselves on `.root`, as ladders, and four of its component rules name `-color-danger-fg` or `-color-warning-fg` (`.status-echo-error`, `.status-echo-warn` and the Message Log's two); nux's `nux.css` has no `.root` block, and three of its rules name `-color-warning-fg` or `-color-danger-fg` (`.on-battery-note`, `.settings-problem`, `.detail-warn`). Modula's `modula.css` has tokens its rules use (`-lock`, `-fault`), set as fixed hexes on `.root` and again on `.root.daylight`, chained to no role.
- **ICON-08** `MUST` App stylesheets take their colors from AtlantaFX `-color-*` tokens, or from tokens the app builds on them; no component rule carries a fixed color, whatever its spelling: a hex, an `rgb()`, `rgba()` or `hsb()` value, or a named color such as `white`. A fixed color may stand only in a token's definition, on `.root` or in a scoped block that sets `-color-*` tokens, and only where the theme has no role to build on: a scrim's or a shadow's color, a color drawn over an image or a canvas and not on the theme's ground, or a color inside a `ladder()` of the ground (ICON-13). The ground, ink or accent of the app's own windows given there as a fixed color, in place of the theme's role, fails this rule; a state color given so is ICON-07's failure and is not counted here again. `transparent` is not a color.
  - *Why:* a fixed color suits one theme; the roles track every theme for free.
  - *Check (code):* every color in a component rule of the app's stylesheets is an AtlantaFX `-color-*` role or a token, and a fixed color stands only where a token is defined, in a `.root` block or in a scoped block that sets `-color-*` tokens: as the plain value of a token for a scrim, a shadow or what is drawn over an image or a canvas (an `rgba()` of black, say), or inside a `ladder()` of the ground. A token for the ground, ink or accent of the app's own windows whose value is a fixed color and no role fails; a state color's token with no role behind it is ICON-07's to judge, not this rule's. A fixed color is a hex (`grep -nE '#[0-9a-fA-F]{3,8}\b'` over those stylesheets lists them), an `rgb()`, `rgba()`, `hsb()` or `hsba()` value, or a named color such as `white`; `transparent` is not one. No component rule holds one, whichever property it sets (a `-fx-*-color`, a `-fx-fill`, an effect's color), inside a `ladder()` or not. A test that fails a fixed color outside the token block is the surest evidence.
  - *Reference:* Termina's `AppCssColorsTest` (`componentRulesUseTokensNotLiteralColours`) and Mirada's `StyleContrastTest` (`noRuleAfterTheTokenBlockHoldsAFixedColour`) fail a fixed color outside the `.root` block: a hex, an `rgb()` or an `hsb()` value, a shadow's color included, and in Mirada's a named color too. Both apps keep the scrim behind an overlay and their shadows' colors as tokens on `.root`, and so does the app template: `-scrim` and `-shadow` on `.root` in its `styles/app.css`, used by `.overlay-backdrop` and the cards' shadows, with no test that holds it. Editora's `StyleContrastTest.severityAndDebugColoursComeFromTheStateTokens` holds six selectors it names, and Editora doesn't yet pass the whole check: `.breakpoint-marker`, `.debug-menu-logpoint`, `.exec-arrow`, `.view-mode-bar`, `.recent-remove`, `.personal-note-tooltip-content`, the fold-gutter markers and the default editor and Markdown preview palettes in `app.css` still carry fixed hexes, other rules an `rgba()` (the scrim of `.overlay-backdrop`, the diff and merge tints and shadows among them), and `.breakpoint-condition-mark` a fixed `white`. Torrenta doesn't either: `.speed-graph-up` and `.speed-graph-legend-up` in its `styles/app.css` hold a ladder of two hexes, and `.palette-backdrop` and two shadows an `rgba()`. Modula and nux write the palette's scrim as an `rgba()` background in a component rule, and a shadow's color the same way (`.palette-backdrop` and `.palette-card` in `modula.css` and `nux.css`), so neither meets the rule yet; nux also sets `-fx-text-fill: white` on `.status-pill` and `.update-badge`.
- **ICON-09** `MUST` Readable text uses `-color-fg-muted` or stronger, which is 4.5:1 on the default, subtle and overlay grounds in both flagship themes, Editora Light and Editora Dark: a line of UI text is `-color-fg-default` and secondary text `-color-fg-muted`. `-color-fg-subtle` (2.7:1 to 3.9:1, under 3:1 on Editora Light's subtle and inset grounds) is for disabled or decorative text only. An app whose default light and dark themes are another pair is held to the same on that pair. A test holds it.
  - *Check (test):* a contrast test computes muted ink on the default, subtle and overlay grounds of both flagship themes (Editora Light and Editora Dark), or of the app's own default light and dark themes where it ships another pair, and finds 4.5:1 or more; and it walks every rule of the app's stylesheets: it fails a rule that sets a text fill (`-fx-text-fill`, or `-fx-fill` on text) to `-color-fg-subtle` unless its selector is on a list the test names, of disabled and decorative selectors only. A test that checks a list of readable selectors instead passes every selector it doesn't name.
  - *Reference:* only the app template's test walks every rule. Editora's `StyleContrastTest.textThatMustBeReadUsesTheMutedRoleNotSubtle` computes the ratios and checks six selectors it names (the palette's and the switcher's hints, an unbound shortcut, the line numbers of the notes and structure trees, the blame date), so it passes while `.encryption-locked-hint` and `.pom-version-unknown` in `app.css` set text a person reads in `-color-fg-subtle`. The app template's test of the same name (`StyleContrastTest`) walks every rule of its `app.css` and both themes and fails a text fill in `-color-fg-subtle` on any selector but the disabled and decorative ones it lists (`SUBTLE_TEXT`), and computes the muted role's ratio on the grounds of both themes. Mirada's `StyleContrastTest` has no test for this rule, and Termina, Torrenta, Modula and nux have no contrast test of their stylesheets.
- **ICON-10** `MUST` Text in a role's tint (the theme's `-color-<role>-fg`, for the accent, success, warning and danger roles) reaches 4.5:1 on that role's subtle fill (`-color-<role>-subtle`) and on the default and subtle grounds (`-color-bg-default`, `-color-bg-subtle`), in both flagship themes, Editora Light and Editora Dark, or, in an app whose default light and dark themes are another pair, in that pair. A test holds it.
  - *Check (test):* a contrast test computes each role's `-fg` on its `-subtle`, on `-color-bg-subtle` and on `-color-bg-default`, in both flagship themes (Editora Light and Editora Dark), or in the app's own default light and dark themes where it ships another pair, and finds 4.5:1 or more.
  - *Reference:* Editora `StyleContrastTest.lightTintedTextReachesAaOnItsOwnGroundAndOnTheSubtleSurface`, `darkTintedTextStaysAa`. Mirada's `StyleContrastTest` (`tintedTextReachesAaOnItsOwnFillAndOnBothGrounds`) holds it on Cupertino Light and Cupertino Dark, the pair it ships.
- **ICON-11** `MUST` Ink on an emphasis fill is `-color-fg-emphasis`, never a fixed white: the token is white in the light themes and a dark ink in the dark ones, 4.5:1 or more on the emphasis fill of the accent, success, warning and danger roles in the two themes ICON-09 names. A test holds it.
  - *Why:* white on the dark accent is 1.68:1.
  - *Check (test):* a contrast test computes `-color-fg-emphasis` on the `-emphasis` fill of the accent, success, warning and danger roles, in both flagship themes (Editora Light and Editora Dark) or in the app's own default light and dark themes where it ships another pair, and finds 4.5:1 or more; and the rules that paint on an emphasis fill use it.
  - *Reference:* Editora `StyleContrastTest.inkOnAnEmphasisFillIsTheThemesOnEmphasisColour`, and Mirada's test of the same name (`StyleContrastTest`), on Cupertino Light and Cupertino Dark: each holds 4.5:1.
- **ICON-12** `MUST` One focus ring color, the theme's `-color-accent-emphasis`, named once on `.root` (Editora's `-focus-ring`) and drawn by the `:focus-visible` block in the app stylesheet (KBD-16), at 3:1 or better on every surface: the default, subtle, inset and overlay grounds of the two themes ICON-09 names (see Keyboard first). A test holds it.
  - *Check (test):* a test reads the ring's token from `.root` (Editora's `-focus-ring`) and computes it on the default, subtle, inset and overlay grounds of both flagship themes (Editora Light and Editora Dark), or of the app's own default light and dark themes where it ships another pair, and finds 3:1 or more.
  - *Reference:* Editora `app.css` (*Keyboard focus*), `StyleContrastTest.theFocusRingIsAtLeastThreeToOneOnEverySurface`.

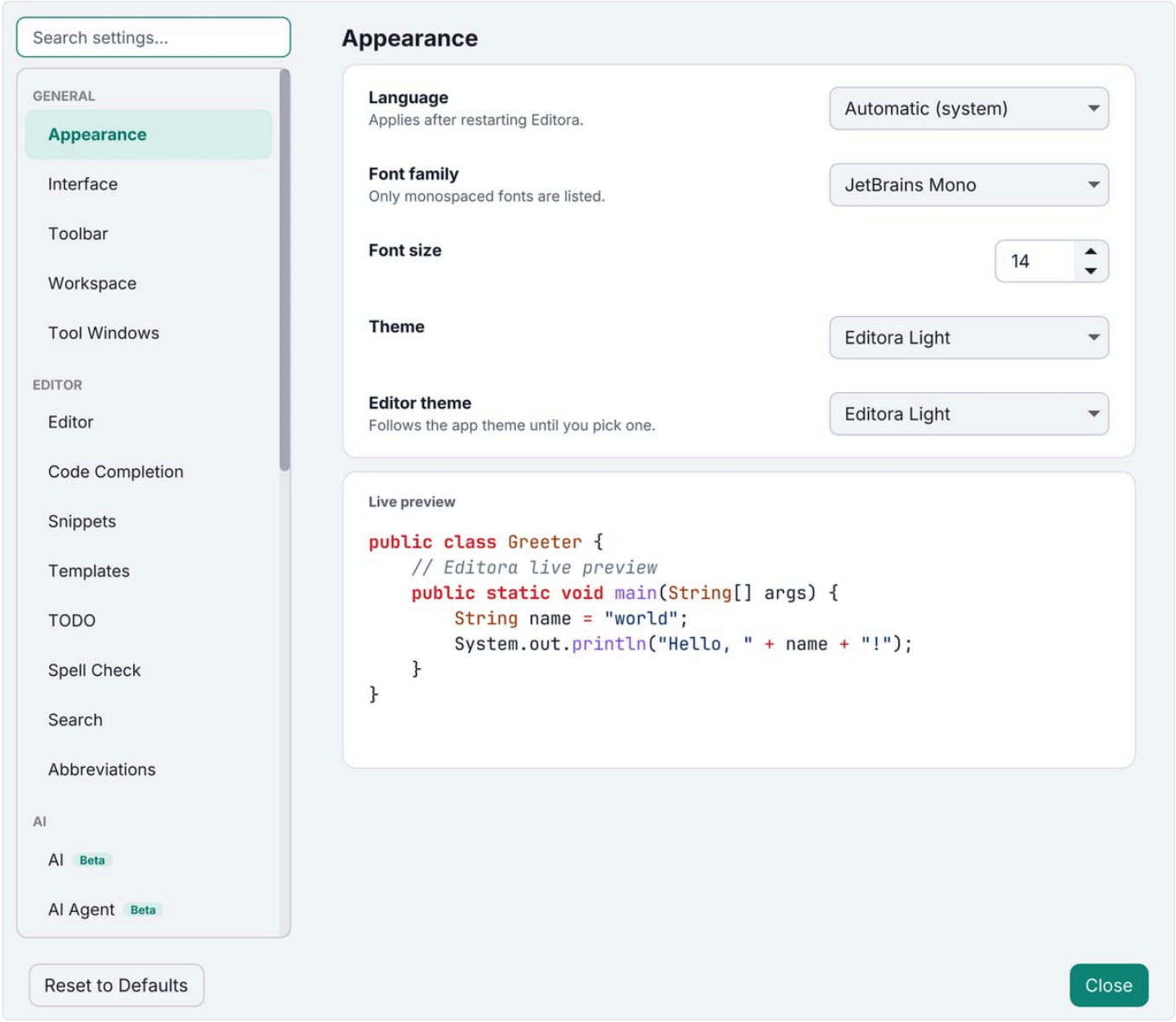
- **ICON-13** `SHOULD` A state color the AtlantaFX theme has no role for (Editora's Git olive and violet) is a `ladder()` of `-color-bg-default`, never one fixed hex for light themes and another for dark. Brace, the periwinkle, is ICON-06.
  - *Why:* the old fixed light hexes read at 3.2:1 on Editora Dark.
  - *Check (code):* each such token is declared as a `ladder()` of `-color-bg-default`; none is a fixed hex, or a pair of hexes switched by theme. A contrast test that resolves each ladder on every bundled theme's ground is the surest evidence.
  - *Reference:* Editora `app.css` (`-state-olive`, `-state-violet`, `-state-key`), `StyleContrastTest.theGitOliveAndVioletFollowTheAppThemesGround`.
- **ICON-14** `SHOULD` The contrast tests are pure: they read the stylesheets and do the WCAG arithmetic without starting the toolkit, so they run with the unit tests on every OS.
  - *Why:* a test that starts the toolkit is an FX test, tagged and run apart from the unit tests (QUAL-06); a pure one fails the plain unit run on every OS, the first time a color changes.
  - *Check (code):* every test class that computes a contrast ratio reads the app's `.css` resources as text and starts no toolkit: none is an FX test.
  - *Reference:* the app template's `StyleContrastTest` and Mirada's `StyleContrastTest`: each reads the stylesheets as text and uses `javafx.scene.paint.Color` for the arithmetic alone. The template's key color is held there too (`theKeyColorReadsOnBothThemesGrounds`), and no FX test of its computes a contrast. Mirada's `AccentTest` is pure too, over the accent it derives from the system's, and so is Termina's `ThemeTest`, which holds the colors of its terminal schemes and not the chrome's stylesheet. Editora's `StyleContrastTest` is pure as well, but Editora doesn't meet the rule yet: FX tests compute contrast ratios on rendered nodes too, `ChordChipThemeFxTest`, `DebugValuePopupThemeFxTest` and `SettingsFrameFxTest` among them.

Themes

- **ICON-15** `MUST` The app has light and dark themes and switches between them at run time, without a restart.
  - *Check (code):* the app has a light theme and a dark one, and picking a theme calls `Application.setUserAgentStylesheet` in the running app.
  - *Reference:* Editora `ui/Themes.java` (`applyAppTheme`).
- **ICON-16** `MUST` The theme setting offers "follow the system": a choice that applies the light or dark theme to match the system's color scheme, and changes with it.
  - *Check (code):* the theme setting has a system value, the place where the user picks a theme lists it, and while it is chosen a listener on `Platform.getPreferences().colorSchemeProperty()` re-applies the theme.
  - *Reference:* Mirada `settings/Settings.java` (`ThemeChoice.SYSTEM`), `ui/Theme.java`; nux `config/Advanced.java` (`THEME_SYSTEM`), `ui/Theme.java`; Termina `ui/Themes.java` (`SYSTEM_ID`, `followSystem`), with the listener in `ui/WindowManager.java`; the app template's `ui/Themes.java` (`ThemeChoice.SYSTEM`). Each lists the choice in its Settings. Editora, Torrenta and Modula don't offer it yet: Editora's and Torrenta's pickers list named themes only (`ui/Themes.java`, Torrenta's `ui/Theme.java`), and Modula's theme is one on/off setting, `daylight` (`config/Settings.java`).
- **ICON-17** `MUST` A theme name the app does not know falls back to the first-run default theme.
  - *Check (code):* the lookup from a saved name to a theme ends in the default, not an exception.
  - *Reference:* Editora `Themes.themeFor`, falling back to `Themes.DEFAULT` (Editora Light).
- **ICON-18** `SHOULD` The state colors never decorate: in every theme amber means "not durable yet", a warning (WIN-01 requires a warning in the echo area in amber) or an update available in About (ABOUT-10 asks for it there), red "broken" and green "verified", and nothing is colored amber, red or green for any other reason. A theme may change the accent, the ground and the ink, never these meanings.
  - *Check (code):* every rule that uses the amber, red or green state token (Editora's `-state-amber`, `-state-red` and `-state-green`) marks one of the meanings in the table under State colors.
  - *Reference:* Editora `app.css`.

Editora lists 28 app themes: the Caret & Ink pair, AtlantaFX's Primer, Nord, Cupertino and Dracula, and 19 community themes (Army, Autumn, Blacky, Blue, Browny, Fall, Navy, News, Spring, Summer, Winter, Yacht, several with light and dark variants) vendored unmodified from the MIT-licensed dlsc `atlantafx-themes` collection. User themes under `<config>/themes/` join the list (`Themes.names()`).

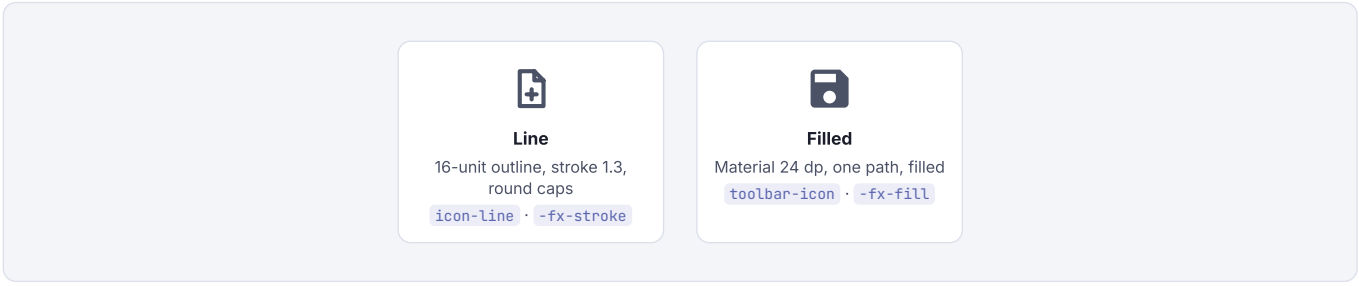
Editor themes are separate (`EditorThemes`). The flagship pair carries its own editor stylesheet; every community theme is served by one `adaptive` editor theme whose syntax palette is expressed in the theme's own `-color-*` tokens, so it follows whatever ground the theme has.



Editora's Settings, Appearance: the app theme and the editor theme both on Editora Light, the editor's font and size, and a live preview.

Icons

Every icon is an SVG path drawn in code, colored by the stylesheet so it follows the theme, and returned as a fresh node per call (a JavaFX node has one parent). There are two families, mid-migration to the UI Kit, and they are not interchangeable.



Property	Line family	Filled family	Reference
Source	The UI Kit's 16-unit outline glyphs, drawn for the kit	Material Design 24 dp single-path glyphs (Apache-2.0), plus Simple Icons brand marks (CC0)	`Icons.line`, `Icons.of`, `NOTICE`
Style class	`icon-line`	`toolbar-icon`	`app.css`
Color	`-fx-stroke: -color-fg-muted`, stroke width 1.3, round caps and joins, fill transparent	`-fx-fill: -color-fg-muted`	`.icon-line`, `.toolbar-icon`

Property	Line family	Filled family	Reference
Hover, disabled	`-color-fg-default` on hover, `-color-fg-subtle` disabled; every context mirrors the matching rule of the other family	The same	`.button:hover .icon-line`, `.button:hover .toolbar-icon`
Drawn size	16 × 1.2 = 19.2 px (`LINE_SCALE` = `ICON_SCALE` × 24 ÷ 16)	24 × 0.8 = 19.2 px (`ICON_SCALE`)	`Icons`
Used for	The chrome the kit draws: toolbar, tool stripe, tabs, and every action that has a kit glyph	Everything the kit specifies no glyph for: file types, context-menu rows, build-tool and browser brand marks	`Icons` , `FileIcons`

- **ICON-19** `MUST` In an app whose icons come in two families, one colored by its stroke and one by its fill, a glyph node carries the style class of one family, never both (Editora's two are `icon-line` and `toolbar-icon`). An app with one icon family, or with no icons in its window, has nothing to mix.
  - *Why:* a line glyph is colored by its stroke, and any `X .toolbar-icon` fill rule (there are dozens) would paint its outline solid. A glyph that moves family moves its CSS with it.
  - *Check (code):* the method that builds a line glyph adds the line family's class alone, the one that builds a filled glyph adds the filled family's alone (Editora's `Icons.line` and `Icons.of`), and no other code adds the second class to a glyph node. A test that builds every glyph and fails when one carries both classes is the surest evidence.
  - *Reference:* Editora `ui/Icons.java` (`line` , `of`) and `editor/MenuIcons.java` , whose glyphs `MenuIconFamilyFxTest.aLineGlyphIsNeverAlsoTaggedAsAFilledOne` holds. Torrenta's `IconsTest` (`everyGlyphIsAStyledScaledPath`) builds every glyph and fails one that carries both classes or neither. Termina, Mirada and Modula draw one family each.
- **ICON-20** `MUST` The same action wears the same glyph wherever it is reached: Save is one floppy on the toolbar, in the File menu and in a tab's menu.
  - *Check (code):* wherever an action is drawn (a toolbar button, a menu row, a tab's menu), its glyph comes from one icon class or one lookup keyed by the command. Where a second icon class holds a copy of its own, the copy has the same path data and style class. A test that compares the path data and style class of each action's glyph across the icon classes is the surest evidence. An app that draws no glyph anywhere has nothing to compare and passes.
  - *Reference:* Editora `ui/MenuBarIcons.java` reuses `Icons` ; `editor/MenuIcons.java` (the editor package may not depend on `ui`) copies each twin's family and path data verbatim, held by `MenuIconFamilyFxTest.everySharedActionUsesTheSameGlyphInBothIconSets` . Mirada gives a command's menu row and its button one glyph from one lookup (`ui/Icons.java` , `forCommand`).
- **ICON-23** `SHOULD` In an app that shows files with an icon each (a project tree, file tabs, a list of files), a file's icon comes from a pure function from file name to icon key, unit-tested without the toolkit, and any type it does not know gets the generic document glyph, so a new type never regresses the look.
  - *Check (test):* a unit test resolves names to keys, an unknown one to the generic key.
  - *Reference:* Editora `ui/FileIcons.java` : `iconKeyFor(name)` resolves the key, `forFileName` turns it into a node; brand logos from Simple Icons, category glyphs (image, archive, PDF, table, lock, JSON, settings) from Material. `FileIconsTest.unknownAndNullFallBackToGeneric` .
- **ICON-24** `MUST` Every glyph set and font the app ships is credited in `NOTICE` with its license.
  - *Check (code):* each icon source named in the icon classes and each family under `fonts/` has an entry in `NOTICE` . An app that ships no glyph set and no font has nothing to credit here.
  - *Reference:* Editora `NOTICE` : Material Design Icons (Apache-2.0), Simple Icons (CC0; the logos stay their owners' trademarks), Inter and the monospace faces (OFL), and the glyph sets inside the bundled Symbols Nerd Font, Octicons and Seti UI (MIT) among them.

An icon-only control gets its tooltip and its accessible name from one helper (`Icons.button` , `Icons.toolbarButton` , `Icons.name`): TBAR-01.

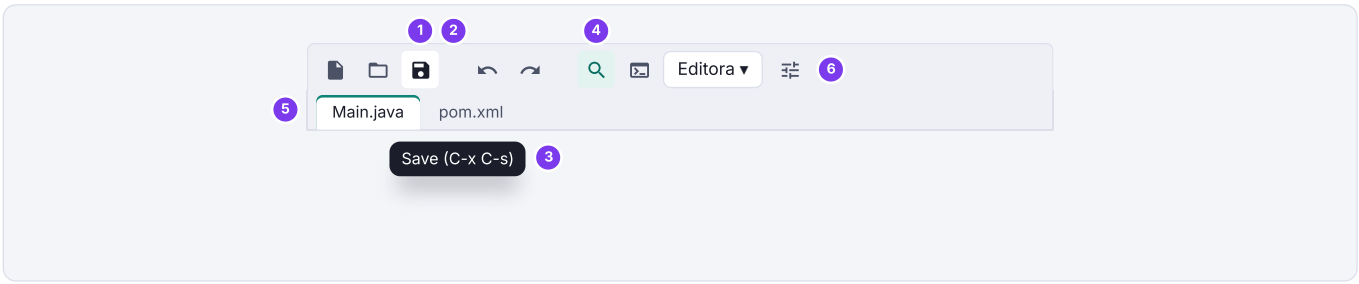
In the apps

App	Icons	Size
Editora	Both families, `ui/Icons` , `FileIcons` , `MenuBarIcons` , `editor/MenuIcons`	19.2 px on the toolbar and in menus
Torrenta	Both families, copied from Editora as `Icons.Glyph` , which knows its box and resizes with `at(px)` ; the toolbar is all line glyphs (stroke 1.5, `-torrenta-icon`)	17 px toolbar, 16 px menu and tab (`MENU_SIZE`), 13 px sidebar
Termina	Material filled, `ui/MenuIcons` , colored by `.menu-icon` (`-color-fg-muted` ; `-color-fg-default` focused; subtle and half opacity disabled)	24 × 0.7 = 16.8 px, menus only
Mirada	Hand-drawn stroked paths on a 24-unit grid, two-unit strokes, round caps, in GNOME's symbolic style, so nothing carries a license; `.icon` strokes `-color-fg-default`	16 px, centered in a 16 px box
Modula	`Glyphs` : the transport symbols (tune, seek, star) as shapes, because the characters are missing from most faces and render differently on every platform	11 px
nux	No in-window icons; the tray icon is rendered as a bitmap by severity (`TrayIconRenderer`)	Tray only

Toolbars and tooltips

A toolbar is a bar of icon-only buttons in a window's chrome, the fast path to commands that are also in the menus and the Command Palette: nothing lives only on it. Every button is a command, so its icon, title and chord come from the registry and the active keymap. The toolbar says which bars count as one. Reference: Editora's `ui/Icons.java` (`button` , `toolbarButton` , `name`), `ui/ToolbarCoordinator.java` (the customizable bar), `ui/WindowChromeCoordinator.java` (`collapseToolbarSeparators`), `ui/ToolWindowManager.java` (the tool window stripes) and the `.tool-bar` , `.toolbar-button` , `.tool-stripe-button` and `.tooltip` rules in `app.css` .

Anatomy



1. **Icon button**, 28 px: padding 3 × 5, radius 4, no border, the glyph in `-color-fg-muted` and `-color-fg-default` under the pointer. It has a name even though it shows none: the same string is its tooltip and its accessible text.
2. **Group gap.** A vertical `Separator` node whose line is transparent: its padding is what separates the groups. No drawn rule.
3. **Tooltip.** The command's title and its chord in the active keymap, "Save (C-x C-s)". A popup window with the menus' radius 8.
4. **Open toggle.** A button whose overlay is showing (Find, the palette, a split) wears `:open` : an accent-subtle fill and the glyph in the accent color. Simple mode's own button does the same while the mode is on.
5. **One band.** The menu bar, the toolbar and the tab strip sit on `-color-bg-subtle` with no hairline between them; spacing does the grouping.
6. **Pinned tail.** The run configuration controls, the project picker, the Simple mode toggle and Settings live in their own container beside the `ToolBar` , so a narrow window overflows the icons (which are all in the menus too) and never the tail.

Spec

Property	Value	Reference
Button	28 px high: padding 3 × 5, radius 4, style classes `button-icon flat toolbar-button` ; every control on the bar is pinned to 28 px, combos and menu buttons included	`.toolbar-button` , `.tool-bar .combo-box-base`
Bar	Padding 5 1 2 6 (asymmetric: the tab strip below adds its own inset); 18 px font for the bar's own labels, controls inside it at 14 px; `-color-bg-subtle` , no rule above or below	`.tool-bar` , `.menu-bar, .tool-bar`
Icon	The command's glyph, 17 px drawn (Torrenta) or a kit glyph at the toolbar scale (Editora); `-color-fg-muted` , `-color-fg-default` on hover, `-color-accent-fg` while `:open`	`Icons` , `.toolbar-icon` , `.icon-line` , `.button:open`
Group gap	A vertical `Separator` with its line transparent; its 0.75 em padding is the gap	`.tool-bar .separator:vertical > .line`
Tail	Its own container (`.toolbar-tail`), same insets and font as the bar, never overflowed	`ToolbarOverflowFxTest`
Run controls	The run configuration combo, then Run, Debug and Stop, the bar's only state colors: the glyph in `-state-green` , `-state-accent` and `-state-red` ; hidden in Simple mode and while there is nothing to launch	`.run-config-play` , `.run-config-bug` , `.run-config-stop` , `RunConfigToolbar.visible`
Tail gaps	Blank `Region` s, no separators: 78 px (about three icons) before the project switcher, 40 px before Simple mode and Settings, 14 px before a badge	`WindowToolbar.appendFixedTail`
Tooltip	`<title> (<chord>)` , the chord from `KeymapManager.displayChord` ; only the title when unbound; radius 8 like the menus	`ToolbarCoordinator.buildExtraButton` , `ToolWindowManager.tooltipFor` , `.tooltip`
Stripe button	A toolbar button on the tool window rail: min 24 px, padding 9 × 5, radius 6, 8 px apart on a 30 px rail; the same muted, hover and `:open` colors; the bottom stripe adds a 14 px label	`.tool-stripe-button` , `ToolWindowManager.register`
Customize mode	The bar tinted `-color-accent-subtle` , a move cursor, the dragged button at 0.4 opacity, a 2 px accent insertion line on the drop target's edge	`.tool-bar.toolbar-customizing` , `.toolbar-drop-before`

Rules

- **TBAR-01** `MUST` Every icon-only button, one that shows a glyph or a symbol such as "?" or "⋯" and no word, has a name: one helper sets its tooltip and its accessible text from the same string, the chord included where the tooltip carries one (TBAR-03), so a screen reader says what a sighted user reads.
  - *Why:* a hand-rolled button gets the tooltip and forgets the name, which is how Editora once had seven copies of the helper drifting apart, and how a screen reader comes to announce "button".
  - *Check (code):* every icon-only button gets its tooltip and its accessible text in one call that sets both from one string (Editora's `Icons.button`, `Icons.toolbarButton` and `Icons.name`). A button with a `setTooltip` and no `setAccessibleText` fails, and so does one whose tooltip is written again later, to add a chord, while its accessible text keeps the first string.
  - *Reference:* Editora `ui/Icons.java` (`button`, `toolbarButton`, `name`), used by its panel toolbars. Torrenta's copy of all three (`ui/Icons.java`): its `name` is the one place Torrenta sets an accessible text, and it names every button of the main toolbar (`MainWindow.toolButton`) and names them again when the keys change. Termina's `ui/IconNames.java` (`name`), which binds both to one string. Five apps don't meet it yet. Editora sets the tooltip only on its main toolbar (`WindowToolbar.setupButton` and `ToolbarCoordinator.buildExtraButton`) and on the floating buttons that leave Zen and Expert mode and bring the toolbar back (`WindowChromeCoordinator.installZenOverlay`), and its stripe buttons' accessible text is the tool window's title without the chord the tooltip carries (`ToolWindowManager.register`). Torrenta's detail pane gives its "⋯" button a tooltip and no accessible text (`ui/DetailPane.java`). Mirada's `ui/Icons.java` (`button`, `name`) makes the accessible text the command's title and puts the chord in the tooltip only. nux's `App.java` sets the tooltip and the accessible text of its "?" button from one string, then its `shortcut` helper writes the tooltip again as "About Nux (F1)" and leaves the accessible text. Modula's transport, Settings and Record buttons have a tooltip and no accessible text (`ui/RadioPane.java`), and so has the "+" button of its preset row (`ui/PresetBar.java`).
- **TBAR-03** `MUST` The tooltip of a button that runs a command names the command and its chord, read from the active keymap and never typed into a string: "Save (C-x C-s)". The name is the command's title (KBD-03), which a tooltip may show without its category prefix, as a menu row may (MENU-01): "Save" for "File: Save". An unbound command shows its title alone.
  - *Check (code):* the tooltip shows the command's title, whole or without its category prefix, and the keymap's display chord for its id, and is rebuilt when the keymap changes. The chord is the one of the command the button runs, and no chord is typed into the string. A fixed key of the control itself (Enter in a find field) is no command's chord and may be written, as KBD-13 allows.
  - *Reference:* Editora `ToolbarCoordinator.buildExtraButton` (the `command.<id>` title, prefix and all) and `ToolWindowManager.tooltipFor`. Its main toolbar (`WindowToolbar.setupButton`, `refreshToolbarTooltips`) takes the chord from the keymap and the name from a `tooltip.*` catalog key of its own: for most buttons that is the title without its prefix (`tooltip.save=Save` beside `command.file.save=File: Save`), for six another name, which doesn't meet the rule: "Find / Replace" for `find.show`, whose title is "Find: Incremental Search"; "Open" for `file.find` ("File: Find File..."); "Close Tab" for `buffer.close` ("Buffer: Close"); "Open Folder as Project..." for `project.open` ("Project: Open Folder..."); "New From Template" for `template.new` ("Template: New File From Template..."); and "Simple UI mode" for `view.toggleSimpleMode` ("View: Toggle Simple UI Mode"). Two more buttons miss it another way: `file.clearRecent` ("File: Clear Recent Files") reads "Clear recent files", and the pinned tail's Stop button, which stops the run as `run.stop` ("Run: Stop Program") does, is set up with no command id, so it reads "Stop the running program" and shows no chord. Torrenta's `toolbar.withChord`. Mirada's `Icons.tooltip` builds the same "label (key)" form with the chord from the keymap in use (`Command.chord`), and `TitleBar.relabel` names the header buttons again when the keys change; Termina's `IconNames.withChord` takes the chord from its keymap too, but its find bar's arrows don't meet the rule yet: `ui/SearchBar.java` names them from `search.older` and `search.newer`, "Previous match, up (Enter)" and "Next match, down (Shift+Enter)", not from a command's title, and gives each arrow the chord of the command that steps the other way. The up arrow steps to an older match, as `findNext` does (`ui/TerminalWindow.java`), and shows the chord of `findPrevious`; the down arrow steps to a newer one, as `findPrevious` does, and shows the chord of `findNext`. The Enter and Shift+Enter in the two strings are the field's own keys. Modula's and nux's tooltips are written in the code, not read from a command: Modula's carry no chord (`ui/RadioPane.java`), and nux's add the display text of a `KeyCombination` to the button's label (`App.java`, `shortcut`).
- **TBAR-14** `SHOULD` Every tooltip shows after the same delay, 400 ms, set where the app makes its tooltips and not tooltip by tooltip.
  - *Why:* JavaFX waits a full second, which reads as no tooltip at all on a row of icons, and a delay set here and there makes some buttons answer faster than their neighbors.
  - *Check (code):* every `Tooltip` the app creates gets a 400 ms show delay from one place: each is made by one factory that calls `setShowDelay`, or the stylesheet's `.tooltip` rule sets `-fx-show-delay: 400ms`. A tooltip made with `new Tooltip` outside the factory and given no delay fails, as does any other delay.
  - *Reference:* the app template: its `styles/app.css` sets `-fx-show-delay: 400ms` on `.tooltip`, and the tooltips its code makes get the same from one constant, `TOOLTIP_DELAY` in `ui/Icons.java` (`bindName`, and `ui/TabHeaders.java` for a tab's). Termina is nearest among the apps: `ui/IconNames.java` (`name`, `SHOW_DELAY`) sets 400 ms on every tooltip it makes, and `ui/TerminalWindow.java` sets the same inline on a tab's folder tooltip, but the tooltip on a disabled row of its `ui/CommandPalette.java` sets none. Editora sets 300, 350 or 400 ms on some (`editor/LazyTooltip.java`'s callers, `ui/StructurePanel.java`, `ui/MapSurface.java`, `ui/DiffViewerPane.java`) and leaves the rest at JavaFX's second; Mirada sets 250 ms on one (`ui/MarksBadge.java`) and none on the others; Torrenta, Modula and nux set none. No app's stylesheet sets `-fx-show-delay`.
- **TBAR-06** `SHOULD` The bars an app stacks at the top of its main window are one chrome band: whichever of the menu bar, the toolbar and the tab strip it has share `-color-bg-subtle` and draw no hairline between them. The bars it stacks at the bottom, whichever of a breadcrumb, a bottom stripe and the status bar it has, do the same. An app with one bar at an edge has nothing to join there.
  - *Why:* three rules within forty pixels read as clutter, not structure.
  - *Check (by hand):* look at the top and the bottom of the main window in a light and a dark theme for a line between two bars, or a change of ground from one to the next.
  - *Reference:* Editora `app.css` (`.menu-bar, .tool-bar`).

The toolbar

Applies to: an app with a toolbar: a bar of icon-only buttons in a window's chrome (its title bar, a band of its own under the title bar or the menu bar, or the header of a panel), each a shortcut to a command. Editora's and Torrenta's `ToolBar`, the toolbars of Editora's panels and Mirada's header buttons are toolbars. So is a tool window stripe, on every side: the label its bottom buttons add to the glyph does not change what they are. These are not toolbars: a bar of labeled buttons (nux's `ToolBar`, whose one icon-only button, "?", only TBAR-01 covers); controls that belong to the window's content, icon-only or not (Modula's transport and footer controls, Mirada's controls over the image); the buttons at the end of a tab strip (Termina's "+" and its chevron); and a find bar's buttons (FIND-16 is theirs). Controls that belong to the window's content include a row of preset chips with its add button (Modula's) and the buttons inside a panel's rows (Mirada's info panel), and an image, PDF or Diff viewer's own bar in the content area (Editora's) is content too.

- **TBAR-02** `MUST` Toolbar buttons stay out of the Tab order: the command, the palette and the menus reach the action, and Tab is for the content. The keyboard page has the general focus rules.
  - *Check (code):* every button on a toolbar is taken out of the Tab order with `setFocusTraversable(false)`, where it is built or by the helper that builds it (Editora's `Icons.toolbarButton`).
  - *Reference:* Editora `Icons.toolbarButton`, used by the toolbars of its Git and Markdown lint tool windows; Torrenta's copy of it, which builds every button of its main toolbar (`MainWindow.toolButton`); Mirada's header buttons: its `Icons.button` takes Rotate out of the Tab order, and `ui/TitleBar.java` takes Info and ☰ out where it builds them. Editora doesn't yet, on these toolbars among others: the buttons of its main toolbar, built by `WindowToolbar.setupButton` and `ToolbarCoordinator.buildExtraButton`, the

stripe buttons, the buttons of the Test Results panel's toolbar, built by `Icons.button` (`ui/TestRunnerPanel.java`), and the float, maximize and close buttons in the header of every tool window (`ui/ToolWindowPanel.java`) take focus.

- **TBAR-04** `SHOULD` A toolbar's groups of buttons are separated by spacing, not rules: where a toolbar holds `Separator` nodes, keep them (their padding is the gap) and make the line transparent.
  - *Check (code):* the stylesheet makes the line of every `Separator` on a toolbar transparent (Editora's `.tool-bar .separator:vertical > .line` , a transparent border), on a panel's toolbar as on the main one. A toolbar that holds no `Separator` (Mirada's header buttons) has nothing to restyle and passes.
  - *Reference:* Editora `app.css` , for its main toolbar. Editora doesn't meet the rule on its Debug panel's toolbar yet: the separators there carry a class of their own, `debug-toolbar-separator` , whose one rule sets their padding and leaves the line drawn (`ui/DebugPanel.java`). Torrenta's separators still draw a line (`.main-toolbar .separator:vertical > .line` in its `styles/app.css`).
- **TBAR-05** `SHOULD` Hiding a button group (Simple mode, a build tool that is not detected) hides the leading, trailing and doubled separators it leaves, so no gap stands on its own, and shows them all again when nothing is hidden.
  - *Check (code):* every change to a toolbar item's visibility ends in one pass over the bar's separators.
  - *Reference:* Editora `WindowChromeCoordinator.collapseToolbarSeparators` , for its main toolbar.
- **TBAR-07** `SHOULD` A toggle looks on while it is on: a button that opens an overlay (Find, the palette, a split) and the Simple mode button carry a pseudo-class (Editora's `:open`) while their state holds, so the way back out does not look like the way in.
  - *Check (code):* each such button sets a pseudo-class from its state, and the stylesheet gives that state a fill.
  - *Reference:* Editora `MainController.OPEN` , `WindowChromeCoordinator` (Simple mode), `.button:open` in `app.css` . Torrenta's Simple mode button carries `:on` , which its stylesheet fills (`.toolbar-button:on`), but Torrenta doesn't meet the rule yet: its palette button carries no pseudo-class while the palette shows (`ui/MainWindow.java`). Its `:active` marks the alternative speed limit on a label of the status bar, not on a toolbar button.
- **TBAR-08** `MUST` A toolbar never holds the only way to an action: every icon on it (in Editora, every icon of the cluster the user can customize) is also in the menus and the palette.
  - *Why:* that is what makes it safe to hide icons in a narrow window and to let the user remove them.
  - *Check (code):* every button on the toolbar runs a registered command, so the palette lists it, and a menu lists that command too: the menu bar, or the app's main menu where it has no bar (WIN-14 says what a main menu is). A button that only opens a menu (Editora's Recent, Mirada's ☰) is no action of its own. In an app whose toolbar is built from a catalog (Editora's `ToolbarCatalog`), every catalog item's command id is also in the menus' model.
  - *Reference:* Mirada's header buttons, Rotate and Info, which its ☰ menu lists too (`ui/TitleBar.java`). Eleven of the twelve buttons of Torrenta's toolbar run a command that a menu row runs too (`MainWindow.buildToolbar`), but Torrenta doesn't meet the rule yet: the twelfth, the palette button, opens the palette from a handler of its own, as the View menu's Command Palette row does, and no command does (PAL-01). Editora `toolbar/ToolbarCatalog.java` : each item carries the `commandId` it runs, and the only items without one are declared special widgets, such as the Recent menu button (`ToolbarCatalogTest`). Editora doesn't meet the rule yet: three catalog items run commands that are in the palette and in no menu, `tool.todo` , `tool.bookmarks` and `view.toggleZen` , which `ui/MenuBarModel.java` doesn't list.
- **TBAR-09** `MAY` The toolbar is customizable: the user reorders, removes and restores its buttons, and the layout is saved in Settings and shared by every window.
  - *Check (code):* the user can reorder the toolbar's buttons, remove one and bring the default layout back (Editora: a customize mode in which a drag along the bar reorders its buttons and a drag off it removes one, and a menu row that restores the default); and the layout is written to the settings, which every window reads. A test that drives a reorder and a removal is the surest evidence.
  - *Reference:* Editora `ui/ToolbarCoordinator.java` , `ToolbarCustomizeModeFxTest` , `ToolbarCustomizationFxTest` . The cluster is a list of catalog ids in Settings (`toolbarLayout`), sanitized on read. The bar's right-click menu offers *Customize Toolbar…* (or *Done*), *Configure…* (the Settings page) and *Restore Default*. In customize mode the bar tints, buttons drag to reorder with an accent insertion line and drag off the bar to remove; Escape leaves the mode.
- The toolbar's right-click menu follows the rules for every right-click menu: MENU-06 and MENU-07.

The main toolbar

Applies to: an app with a main toolbar: a toolbar (The toolbar says what one is) that is a band of its own across the top of the main window, as Editora's and Torrenta's `ToolBar` are. Mirada's header buttons, which share the title bar with the file's name, are a toolbar and not a main toolbar; nux's bar of labeled buttons is neither.

The main toolbar reads left to right in two parts. The **icon cluster** holds the everyday commands in groups (in Editora: file, with Recent as a menu button; edit; find; split; the palette) and is the part that gives way: a narrow window moves its last icons into the `ToolBar` 's overflow chevron (»). The **pinned tail** sits beside it in a container of its own and keeps every control at any width. Editora's holds the run configuration selector with Run, Debug and Stop, the project switcher with Open Folder, a *snapshot* or *dev* badge when the build is one, and Simple mode and Settings at the end. A command the bar can lose to a narrow window goes in the cluster. One the user must always reach from the bar, or the way back out of a mode, goes in the tail.

- **TBAR-15** `SHOULD` The main toolbar's icon cluster and its pinned tail (Settings and the other controls the user must always reach) are separate containers: when the window narrows, the cluster's icons go into the overflow chevron and the tail keeps every control.
  - *Why:* a `ToolBar` overflows from its end, so with everything in one bar a narrow window hides Settings and the project switcher first and keeps icons that are all in the menus anyway (TBAR-08).
  - *Check (code):* the tail's controls sit in a container of their own beside the `ToolBar` that holds the icon cluster, and are not items of that `ToolBar` , so its overflow cannot take them. A test that narrows the window and finds the cluster overflowing while the tail's last control stays flush with the bar's right edge is the surest evidence.
  - *Reference:* Editora `WindowToolbar.appendFixedTail` , `ToolbarOverflowFxTest` . Torrenta keeps Simple mode and Settings inside its `ToolBar` , after a spacer, so they are the first to overflow.
- **TBAR-16** `SHOULD` Settings closes the main toolbar at its right end, with the Simple mode toggle just before it in an app that has Simple mode, whether the mode is on or off.
  - *Why:* the same two buttons in the same place in every app, and the toggle is the way back out of a mode that empties much of the bar.
  - *Check (by hand):* look at the bar's right end with Simple mode off, then on.
  - *Reference:* Editora `WindowToolbar.appendFixedTail` ; Torrenta `MainWindow.buildToolbar` .
- **TBAR-17** `SHOULD` A main toolbar button is disabled while its command can't act (Save on a saved file, Undo with nothing to undo, Cut and Copy with no selection, a split on a tab that can't be split), and the events that change that state update it; nothing polls. KBD-04 has the same state in the menus and the palette.
  - *Why:* a button that does nothing when clicked reads as a broken button.
  - *Check (code):* each button's disabled state is bound to its command's availability, or refreshed from the document's edit, selection and dirty events and on a tab switch.

- *Reference:* Editora `WindowToolbar.refreshEditState`, `refreshSplitButtons` and `refreshPasteState`, which reads the clipboard only on a tab switch, on focus and after a copy, never on a keystroke; the buttons added from the catalog (`ToolbarCoordinator.buildExtraButton`) stay enabled. Torrenta `MainWindow.toolButton` binds each button's `disableProperty` to its command's `enabled`.

- **TBAR-18** `SHOULD` A command hides the main toolbar and shows it again, and the choice is kept across launches.
  - *Check (code):* a registered command toggles the bar, and the setting it writes is read back at startup.
  - *Reference:* Editora `view.toggleToolbar` (`WindowChromeCoordinator.toggleToolbar`), with a floating button at the top right that brings a hidden bar back; Torrenta `toggle-toolbar`, kept as `settings.toolbar()`.
- **TBAR-19** `SHOULD` The main toolbar keeps its height when a control in it hides or shows (in Editora, the project switcher in Simple mode or with Projects off), so the tabs and the content under it never move.
  - *Check (by hand):* hide and show each control the bar can lose (in Editora, turn Simple mode on and off) and watch the top edge of what sits under the bar: it stays where it was.
  - *Reference:* Editora `WindowToolbar.stabilizeToolbarHeight`, which pins the row's minimum height to the project switcher's.

Tool window stripes

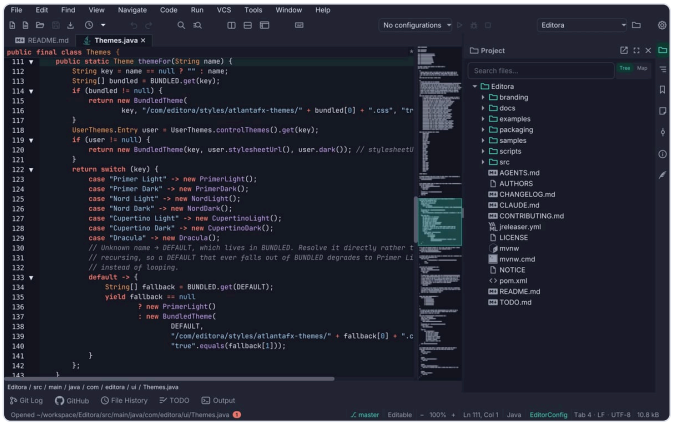
Applies to: an app with tool window stripes.

A stripe is a toolbar (The toolbar), so its buttons stay out of the Tab order (TBAR-02) and an icon-only one has its name as tooltip and accessible text (TBAR-01). These two rules add what only a stripe has.

- **TBAR-12** `SHOULD` A tool window stripe button takes its name from its tool window: the name its tooltip shows (TBAR-03) is the window's title, with the chord of the command that toggles the window, and it follows the title when the title changes. The button's right-click menu has *Open in Split* and *Hide*.
  - *Check (code):* the stripe button's tooltip is built from the tool window's title and the keymap's display chord for the window's command, and is built again when the title changes; the button has a right-click menu with those two rows.
  - *Reference:* Editora `ToolWindowManager.register` (the menu) and `tooltipFor`, which `refreshTooltips` runs again on a keymap switch only. Editora doesn't meet the rule yet: a title change (the Project window becoming "Current Folder") reaches the button's accessible text and the bottom stripe's label, not its tooltip.
- **TBAR-13** `SHOULD` The right stripe's first icon lines up with the editor's first line of text, not with the tab strip.
  - *Check (code):* the code that lays out the side stripes pads the stripe's top by the tab strip's measured height, so its first button starts where the editor's text does. A test that compares the first stripe button's top with the editor's first line is the surest evidence.
  - *Reference:* Editora `ToolWindowManager.alignVerticalStripesWithEditorText`, `ToolWindowStripeAlignmentFxTest`.

In the apps

- **Editora**: the customizable cluster, the pinned tail, the stripes.
- **Torrenta**: the same `Icons.toolbarButton` and `Icons.name`, 32 px buttons, Simple mode as `:on`; its separators still draw a line, and Simple mode and Settings sit inside the `ToolBar`, so a narrow window overflows them first.
- **Mirada**: a `HeaderBar` title bar with flat icon buttons around the file name, stroked symbolic icons in the text color, tooltips of the form "label (key)" with the key read from the keymap; `Icons.button` and `Icons.name` set each button's tooltip and accessible text together, and the buttons are out of the Tab order.
- **Termina**: no toolbar (the tab strip is the row under the menu bar, and the "+" and the chevron at its end are the strip's own buttons); its find bar's buttons (FIND-16) are 24 × 22 flat icons, their tooltips and accessible text set from one string by `ui/IconNames.java`, out of the Tab order.
- **Modula**: no toolbar: the transport buttons and the footer's Settings and Record are the radio's own controls.
- **nux**: no toolbar: its JavaFX `ToolBar` is a bar of labeled buttons, a combo box and two menu buttons, with one icon-only button, "?".
- **The app template**: no toolbar.



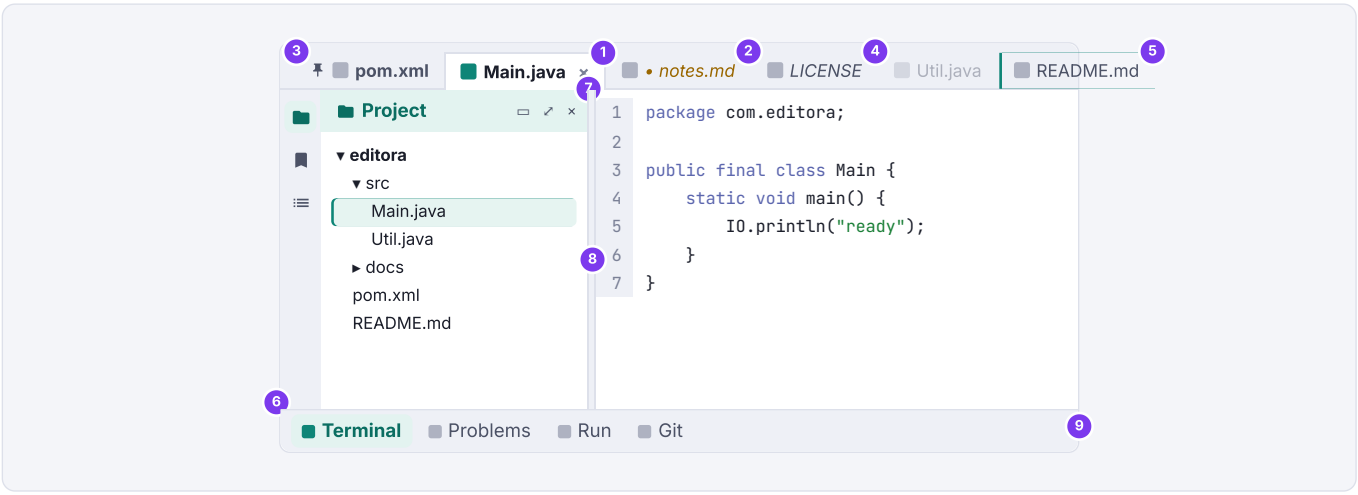
Editora's toolbar: grouped icons on the chrome band, the run configurations, the project picker and Settings pinned at the right.



Mirada's HeaderBar: flat icon buttons around the file name.

Tabs and tool panels

A window shows its documents in a **tab strip** and its tools in **panels** docked beside or under them. Both are driven by commands, both remember their state, and both draw their chrome as one band with the toolbar, not as boxes inside boxes. Reference: Editora's `ui/TabHeaders.java` (the tab header), `ui/ClosedTabs.java`, `ui/Switcher.java`, `ui/ToolWindowManager.java`, `ui/ToolWindowPanel.java` and the `.tab-pane`, `.tool-stripe` and `.tool-window` rules in `app.css`.



1. **Active tab.** Its fill is the editor's own surface (`-editor-surface`) with a border on three sides and none at the bottom, so the tab opens into the editor instead of sitting on it. Title bold in `-color-fg-default`; the file-type glyph tints accent.
2. **Unsaved tab.** A `•` before the name, amber (`-state-amber`, "not durable yet") and italic. Amber outranks the Git-status color: "you have not saved this" matters more than "this differs from HEAD".
3. **Pinned tab.** A pin glyph and a bold title, kept at the front of the strip in pin order. Every bulk close (Others, All, Unmodified, Left, Right) skips it.
4. **Read-only tab.** Muted and italic, echoing the status bar's Read-Only segment.
5. **Drag to reorder.** The header is the drag handle and a snapshot of the whole tab follows the pointer; the dragged tab dims to 40 % and a 2 px accent line marks the edge it would land on (left half of a tab: before it; right half: after).
6. **Tool stripe.** A 30 px rail of icon buttons, one per tool window, 8 px apart, on the subtle ground with no drawn edge. The open window's button is tinted; the one holding keyboard focus is tinted stronger. The first icon lines up with the editor's first line of text, not with the tab strip.
7. **Panel header.** The window's glyph and title (14 px bold, muted) with float, maximize and close; the same size and glyph as its stripe button, so the two name the same thing. On the subtle ground, with no rule under it; when the panel holds focus the band tints accent-subtle and the title turns accent.
8. **The divider is the edge.** The panel draws no frame: the SplitPane divider between it and the editor is the real, grabbable edge, and a hairline beside it would only double it. Its position is remembered.
9. **Bottom stripe.** Its buttons carry a 14 px label beside the glyph, side by side; the open one is accent and bold.

Window title

- **TABS-10** `SHOULD` The window title names the file, the project and the app, with a `●` prefix for an unsaved file. An app with no files or projects names what the window shows in their place (a terminal, the working folder or the title its program set), or the app alone where there is nothing to name.
  - *Check (code):* the code that sets the stage title adds all three and the marker; in an app with no files or projects, what stands in for them, or the app's name alone.
  - *Reference:* Editora `MainController.updateWindowTitle`. Termina's title follows the selected tab (`TerminalTitle.window`: the name the user gave the tab, the title the shell set or the working folder's path, and the app's name when it knows none). The app template names the file, its sample project and the app, with the `●` prefix (`windowTitle` in `ui/DocumentTabs.java`; `TabsAndPanelsTest`, `theWindowTitleNamesTheFileTheProjectAndTheApp`). Modula and nux show the app's name.

Tabs

Applies to: an app whose main window holds documents or sessions in tabs the user opens and closes, as Editora's files and Termina's terminals are. The pages of a Settings window (nux's `TabPane`), the fixed sections of a detail pane (Torrenta's) and a strip that switches the window between views (Torrenta's Transfers, Search and RSS) are not such tabs: the user neither opens nor closes them. The Tabs rules cover every tab of the main window's strip, whatever it shows (Editora's image, hex and PDF viewer tabs as much as its editor tabs), not tabs inside a tool window (those of Editora's Terminal and Build Output panels).

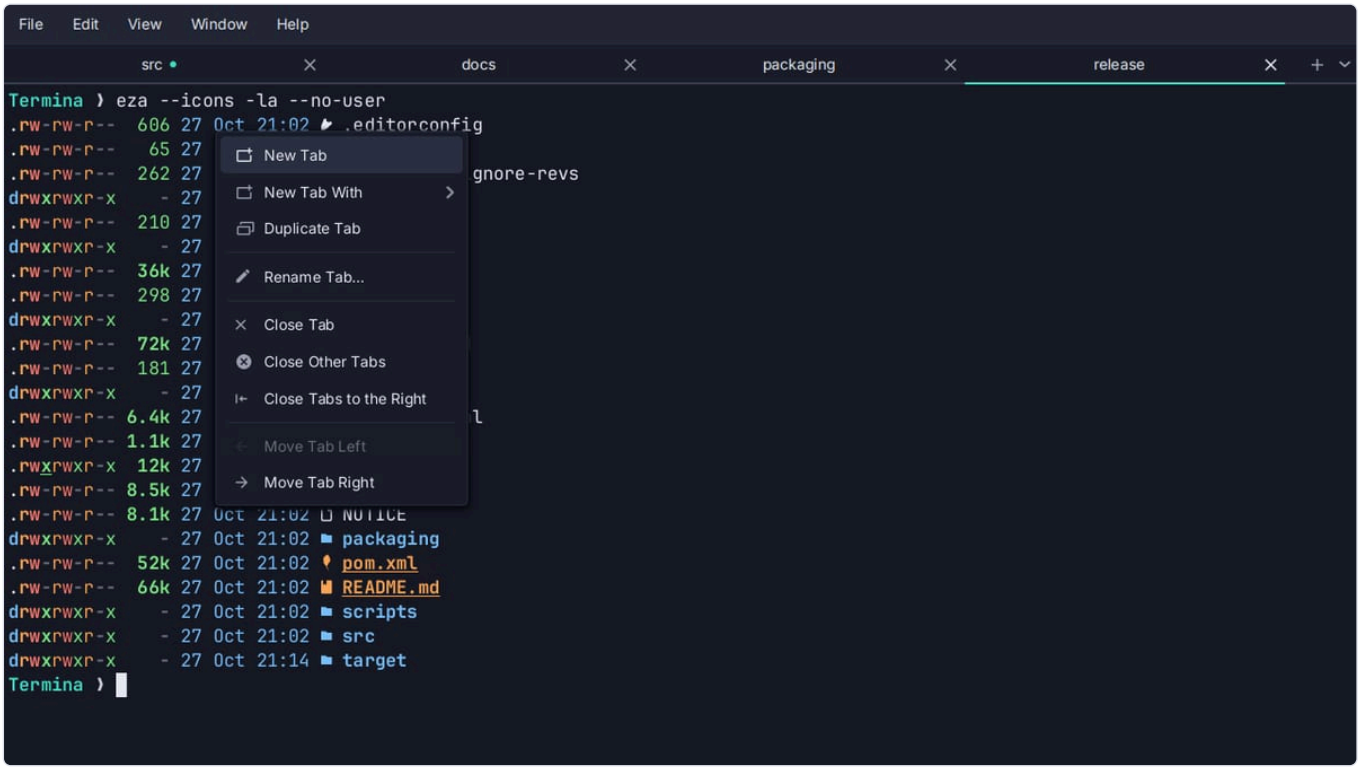
- **TABS-01** `MUST` Every tab operation is a command, so it can be bound to a chord, is in the palette and has a menu row: close, close others, close all, close unmodified, close to the left and right, pin, reopen closed, next and previous. The tab's right-click menu lists the ones that act on a tab. A terminal app's tabs are never saved or unsaved, so a terminal app has no close unmodified, and closing its window closes all its tabs, so it needs no close all; the other operations are its commands too.
  - *Check (code):* the registry holds a command for each operation, each has a row in the menu bar or in the tab's right-click menu, and that menu (Editora's `TabContextMenu`), on every tab of the strip, lists the ones that act on a tab; in a terminal app, each operation but close unmodified and close all.
  - *Reference:* Editora `ui/WindowCommandRegistrar.java` (`buffer.close`, `buffer.closeOthers`, `buffer.closeAll`, `buffer.closeUnmodified`, `buffer.closeLeft`, `buffer.closeRight`, `buffer.togglePin`, `buffer.reopenClosed`, `buffer.next`), `ui/TabContextMenu.java`. Editora has no previous-tab command yet, and its image, hex

and PDF viewer tabs have no right-click menu: `openImageTab`, `openHexTab` and `openPdfTab` in `ui/FileWorkflowCoordinator.java` install none, and `installTabMenu` in `ui/MainController.java` is called for an editor's tab alone. Termina `ui/TerminalWindow.java` (`closeTab`, `closeOtherTabs`, `closeTabsToTheLeft`, `closeTabsToTheRight`, `pinTab`, `reopenClosedTab`, `nextTab` and `previousTab` in `menuCommands`; the tab's menu in `buildTabMenu`). Its File menu closes the tab, the others, the ones to either side, or the window, which is its close all (`ui/MenuBarModel.java`).

- **TABS-02** `MUST` The header is the title: the Tab's own text stays empty, and a graphic header carries the file-type glyph, the markers and the title in a label of its own (Editora's style class for it is `tab-title`). The tooltip is the full path, in which `~` may stand for the home folder. A terminal app's tab shows no file: its header has no file-type glyph, its title names the tab (the user's own name for it, the title the shell set, the running program or the working folder), and its tooltip is the working folder's full path.
  - *Why:* the graphic is what makes the header a drag handle and lets each state set its own color.
  - *Check (code):* every tab that shows a file, an editor's or a viewer's, gets `setText("")` and a graphic that holds the file-type glyph and the title in a label of its own (Editora's `.tab-title`); its tooltip is the file's path. In a terminal app: every tab's own text is empty, its graphic holds the title label and no file-type glyph, and the tooltip is the working folder of the tab's active pane. A path that writes the home folder as `~` is a full path.
  - *Reference:* Editora `ui/TabHeaders.java`, for its editor tabs (`bufferHeader`, `tooltip`). Its image, hex and PDF viewer tabs don't yet: `contentHeader`, which builds their header, sets no tooltip. Termina `TerminalWindow.addTab` (a `tab-title` label as the tab's graphic, and `installDirectoryTooltip`) over `ui/TerminalTitle.java` (`tab`, and `tooltip`, which writes the home folder as `~`).
- **TABS-03** `MUST` A tab's header, the one graphic header of TABS-02, carries the tab's states: a `•` prefix in amber italic for unsaved, a pin and bold for pinned, muted italic for read-only. A terminal app's tab is never unsaved or read-only, and its header says in their place what the tab is doing: the title is the running program's name while one runs, unless the user or the shell named the tab, and a mark beside the title says what happened while the tab was not selected (a notification, a bell, or output after a quiet spell), cleared when the tab is selected. A pinned terminal tab is drawn as any other pinned tab, with a pin and a bold title.
  - *Check (code):* the code that builds a tab's header marks it unsaved, pinned or read-only (Editora's style classes `dirty`, `pinned` and `read-only` on `.tab-header`) and adds the `•` prefix and the pin, and the stylesheet gives each mark the look the rule names. In a terminal app: the stylesheet styles the pinned title, the code that names the tab puts the running program before the folder, and the header shows the mark and clears it when the tab is selected.
  - *Reference:* Editora `ui/TabHeaders.java`, `app.css`; the Unsaved, Pinned and Read-only rows of the Spec table. Termina `ui/TerminalTitle.java` (`tab`), `ui/TabMarks.java` and `TabMarkState` in `ui/TerminalWindow.java` (a dot after the title, in the terminal palette's magenta, yellow or cyan, and a pin), and `.tab-title.pinned` in its `app.css`.
- **TABS-04** `MUST` Bulk close skips pinned tabs: close others, close all, close unmodified and close to the left or right never close a pinned tab. A terminal app has three of them (TABS-01), and each skips its pinned tabs too.
  - *Check (code):* every bulk close the app has (close others, close all, close unmodified, close to the left and to the right) builds the list of tabs it will close without the pinned ones: it tests each tab for pinned before taking it (Editora's `closeOtherTabs`, `closeAllTabs`, `closeUnmodifiedTabs`, `eligibleToLeft` and `eligibleToRight`). A test that asserts it is the surest evidence.
  - *Reference:* Editora `ui/MainController.java`. Termina `ui/TabRanges.java` (`closing`, which leaves out every pinned position and which each bulk close of `ui/TerminalWindow.java` goes through) and `TabRangesTest`.
- **TABS-05** `MUST` Pinning moves the tab to the end of the pinned group, unpinning to just after it.
  - *Check (code):* the code that pins or unpins a tab then moves it to the boundary between the pinned tabs and the rest, the index equal to the number of other pinned tabs in its strip: the last place of the pinned group for a tab just pinned, the first place after the group for one just unpinned (Editora's `togglePin`, which calls `moveToPinBoundary`).
  - *Reference:* Editora `ui/MainController.java` (`togglePin`), `ui/EditorArea.java` (`moveToPinBoundary`). Termina `ui/TerminalWindow.java` (`togglePin`, which counts the other pinned tabs and moves the tab to that index).
- **TABS-06** `MUST` Reopen Closed Tab brings back the files whose tabs this window closed, newest first, bounded to twenty, a file listed once, with the caret where it was. In a terminal app the *tab* comes back (its shells, their directories, the pane arrangement, the names the user gave them, its place in the strip) and not the scrollback.
  - *Why:* the process is gone, and pretending otherwise would be worse than saying so.
  - *Check (code):* each window keeps a list of the tabs it closed, newest first and no longer than twenty; the command takes the newest entry and opens it. In an app whose tabs hold files, an entry is the file, whether an editor's tab or a viewer's showed it, with the caret where the tab has one; closing a file again replaces its earlier entry, and reopening puts the caret back. In a terminal app, an entry holds the tab's panes with each one's shell and directory, their arrangement, the names the user gave them and the tab's place in the strip, and no scrollback. A test over the list is the surest evidence.
  - *Reference:* Editora `ui/ClosedTabs.java`, `ClosedTabsTest`; `C-S-t` in the CUA keymap. Editora doesn't yet for its image, hex and PDF viewer tabs: `rememberClosed` in `ui/MainController.java` records the file of an editor's tab alone, so a closed viewer tab does not come back. Termina `ui/ClosedTabs.java` and `ClosedTabsTest`, on `Ctrl+Shift+Z` (`Cmd+Z` on macOS).
- **TABS-07** `SHOULD` Tabs reorder by dragging, with this feedback: the dragged header dimmed and an accent line on the edge where it will land.
  - *Check (by hand):* drag a tab along the strip and watch the landing edge.
  - *Reference:* Editora `TabHeaders.enableTabDrag`; Termina `installTabDrag` and the pure `TabReorder.insertIndex`. In Termina a double-click on a tab renames it, and a pane dragged onto a tab joins it (the whole header outlined, `tab-drop-into`).
- **TABS-08** `SHOULD` In an app whose tabs hold files, a tab shows its file's Git status in the Project tree's colors: modified accent, added green, untracked olive, renamed violet, deleted muted, conflict red. A read-only tab still shows it; an unsaved one shows amber. The tab's tooltip names the file's Git status too, so color is not the status's only sign (ACC-05). A terminal app's tabs hold no files, so the rule passes it by.
  - *Check (code):* the code that styles a tab gives it, for its file's Git status, the style class the Project tree gives that file's row, read-only or not, and the stylesheet lets the unsaved tab's amber win over it; and the code that sets the tab's tooltip adds the status in words (modified, added, untracked and the rest) when the file has one. A test that finds the tree's class on a changed file's tab, and the status in its tooltip, is the surest evidence.
  - *Reference:* for the colors, Editora `ui/TabGitStatus.java`, `TabGitStatusTest`. For the tooltip, no app yet: Editora's tab tooltip is the file's full path (`tooltip` in `ui/TabHeaders.java`), so the tint is the tab's only sign of the status. Its Project tree adds a letter to the color; its tabs add no word, glyph or shape. The app template meets both halves for its sample files: `ui/TabGitStatus.java` gives a tab the class its Documents window gives the file's row (`TabsAndPanelsTest`, `aChangedFileWearsTheDocumentsWindowsClass`), and `ui/TabHeaders.java` adds the status in words to the tab's tooltip, after the path.
- **TABS-09** `SHOULD` An app whose tabs hold files has a switcher that lists the open files in tab order. A command opens it; the arrows or the active keymap's line chords move, Enter or releasing Ctrl (the Control key, on macOS too) activates, Esc or the keymap's cancel chord cancels, and Backspace, Delete or the keymap's delete-character chord closes the highlighted file. The switcher's width is fixed to the longest path on each open, so it never resizes while you move. A terminal app's tabs hold no files, so the rule passes it by.
  - *Check (by hand):* open it with several tabs; move with the arrows and with the active keymap's line chords, close a file from it, cancel it, and open it again to activate another file; the list keeps its width.
  - *Reference:* Editora `ui/Switcher.java`, which reads its keys through `ui/PickerKeys.java` and the keymap, and `SwitcherFxTest`. In Editora's Emacs keymap `C-x C-b` opens it (`switcher.show`), `C-n` and `C-p` move, `C-g` cancels and `C-d` closes the file (`keymaps/emacs.json`); no other Editora keymap binds `switcher.show`.

- **TABS-12** `SHOULD` A command hides the tab bar.
  - *Check (code):* a registered command hides the tab header, with any number of tabs open. One that hides it only while the window has a single tab does not meet the rule.
  - *Reference:* Editora `view.toggleTabBar`, `.tab-pane.no-tab-header` in `app.css`. Termina doesn't yet: the nearest it has is a setting, with a palette command that toggles it, that hides the strip only while the window has one tab (`settings.hideTabBar` in `ui/SettingCommands.java`).

Termina's `HeaderBar` holds the menu bar and the window title (`ui/TerminalWindow.java`, `buildHeaderBar`), and its tab strip is the row under it.



Termina's tab strip, the row under the menu bar, with a tab's right-click menu.

Tool panels

Applies to: an app with tool panels: a tool panel is a panel docked at an edge of the main window, beside or under the content, that the user shows and hides as a whole, the content taking its room while it is hidden. A sidebar, a detail pane, an info panel and a thumbnail strip are tool panels, on a stripe or not. A section that folds in place and leaves its heading showing is content (nux's All variables table, a `TitledPane` under its dashboard), and so are a terminal's split panes: neither is a tool panel. Nor is a bar of the window's chrome that a command hides (a toolbar, a status bar, a tab strip).

- **TABS-13** `MUST` Every tool panel has a command that shows and hides it, with the id `tool.<id>`, as Editora's `tool.project` is, and where the app has a stripe, the panel's stripe button runs that command. In the Emacs keymap, `M-1` through `M-9` and then `M-0` run those commands, one chord to a panel: an app with fewer than ten panels uses the first of the chords, and one with more chooses the ten panels that get them. An app with one key table and no Emacs keymap (KBD-09) owes none of these chords.
  - *Check (code):* every tool panel has a registered command whose id is `tool.` and the panel's own id, which shows the panel when it is hidden and hides it when it shows; a stripe button, where the app has stripes, runs that command, and one that toggles the panel directly and skips what the command does first fails; and `emacs.json`, in an app that ships one, binds `M-1`, `M-2` and on through `M-0` to those commands, one for each panel up to ten.
  - *Reference:* Editora registers a `tool.<id>` command for each of its tool windows (`ui/WindowCommandRegistrar.java`) and binds ten of them to `M-1` … `M-0` (`keymaps/emacs.json`). It doesn't meet the stripe clause yet: `ToolWindowManager.register` gives each stripe button the manager's own `toggle`, not the command, so the button skips what some commands do first (`tool.remote` refreshes its panel; `tool.project` and `tool.notes` test that the feature is on). Nor does every one of its commands hide a panel that shows: `tool.gitLog`, for one, only opens its panel (`openGitLog` in `ui/GitWindowCoordinator.java`), and so do the `tool.<id>` commands of its build tools (`registerCommands` in `ui/BuildCoordinator.java`). Torrenta and Mirada don't meet the rule yet. Torrenta has no `tool.<id>` command: its two panels' commands are `toggle-sidebar` and `toggle-detail` (`ui/MainWindow.java`), and it ships no keymap file, so it owes no chord. Mirada's two are `view.properties` and `view.thumbnails` (`ui/Command.java`), and its `keymaps/emacs.json` binds them to `i` and `t`, where the rule requires `M-1` and `M-2`.
- A panel's content is built on first use: PERF-14.
- **TABS-15** `SHOULD` Panel state is restored: which panels are open, where each sits and every divider position go to the session-state file (SET-08; Editora's is `workspace-state.json`) and come back on the next launch. In an app with tool window stripes that includes each window's side and order, the stripe buttons the user hid, and the windows that float.
  - *Check (code):* the session state holds each of those, and the code that builds the panels reads them back at startup.
  - *Reference:* Editora `ui/ToolWindowManager.java`, `config/WorkspaceState.java`.
- **TABS-19** `SHOULD` Closing the panel that holds keyboard focus returns focus to where it was in the content (in Editora, the editor).
  - *Check (code):* the code that closes a panel keeps the content's last focus owner and refocuses it on close.
  - *Reference:* Editora `ToolWindowManager.focusEditor`, a weak reference to the last editor focus owner.

Tool window stripes

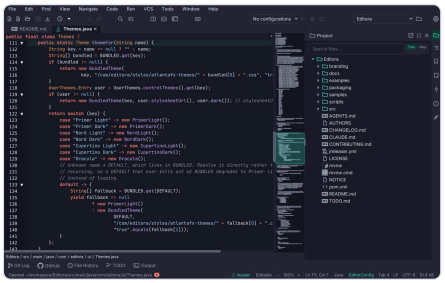
Applies to: an app with tool window stripes.

A stripe is the rail of buttons along an edge of the window, one button per tool window, that Editora has (pins 6 and 9 of the mockup). Its tool windows are tool panels too, so the rules of Tool panels cover them; these cover what only a docking system has.

- **TABS-16** `SHOULD` Tool windows dock on three sides, left, right and bottom, and a side holds at most two windows stacked (Project over Structure, two consoles along the bottom).
  - *Why:* past that a side stops being a dock and becomes a list of slivers.
  - *Check (code):* the sides are an enum of three and the per-side limit is two.
  - *Reference:* Editora `ToolWindow.Side`, `ToolWindowManager.MAX_PER_SIDE`.
- **TABS-17** `SHOULD` A tool window re-docks by dragging its stripe button to another stripe, with the same dim-and-accent-line feedback as a tab; an empty stripe appears as a dashed accent drop zone for the duration of the drag.
  - *Check (by hand):* drag a stripe button to the opposite side.
  - *Reference:* Editora `ui/ToolWindowDock.java`, `ToolWindowVisibilityTest` (an empty stripe shows while dragging).
- **TABS-18** `SHOULD` Tool windows float, maximize, resize and close from the keyboard, and at most one window is maximized at a time.
  - *Check (code):* floating, maximizing, resizing and closing a tool window are each a registered command, so the palette runs them without the mouse; and maximizing one first restores the one that was maximized.
  - *Reference:* Editora `view.floatToolWindow` (palette only), `view.maximizeToolWindow` (`C-x f`), `view.closeFocusedToolWindow` (`C-x 0`), the resize commands (`C-x S-.`, `C-x S-,`, `C-x S-6`); `ui/ToolWindowManager.java` keeps one `maximized` panel and restores it before it maximizes another.
- **TABS-20** `SHOULD` Hidden is not unavailable: the user's show/hide choice for a stripe button is kept apart from a context-driven hide (the Commit window outside a Git repository), so the second never clobbers the first.
  - *Check (code):* the stripe keeps two values for a tool window, the user's saved show/hide choice and whether the context offers the window now, shows the button only when both hold, and never writes the first when the second changes. A test that asserts it is the surest evidence.
  - *Reference:* Editora `ui/ToolWindowVisibility.java`, `ToolWindowVisibilityTest`.
- **TABS-21** `SHOULD` A tool window whose content needs more room than the default share of its side can ask for more on its first open (Editora's Debug window asks for 42 %); the user's own size wins afterwards.
  - *Check (code):* a tool window can ask for a larger share of the side on its first open (Editora's `setFirstOpenShare`), the docking code grants it only while the user has set no size for that window, and the other windows keep the side's size. A test that asserts it is the surest evidence.
  - *Reference:* Editora `ToolWindow.setFirstOpenShare`, `ToolWindowFirstOpenShareTest`.
- **TABS-22** `SHOULD` A stripe button badges a spinner while its window has something under way.
  - *Check (code):* a tool window has a busy state, and its stripe button shows a spinner badge while the state holds and hides it when it ends. A test that asserts it is the surest evidence.
  - *Reference:* Editora `ToolWindow.busy`, `ToolWindowManagerFxTest`.
- **TABS-23** `SHOULD` Tool windows follow the active document where they should: the Git UI follows the active tab's repository, the status bar's environment segment the active `.http` file, Local History the file whose tab menu opened it.
  - *Check (by hand):* switch between tabs from two repositories with the Git window open.
  - *Reference:* Editora `ui/GitPathScope.java`.

In the apps

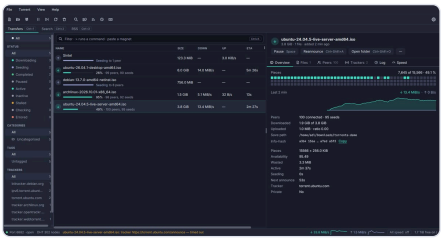
The other apps have panels and no stripes, so only Tool panels covers them. Torrenta's filter **sidebar** sits down the left or the right of the list or nowhere, kept as two settings keys so hiding it keeps its side (`SidebarPlace`); its inner edge drags to resize within readable bounds and a double-click resets it (`SidebarWidth`); the **detail pane** docks under the list in two columns or beside it in one, from the View menu (`DetailPane`). Mirada's info panel opens on the right and its thumbnail strip along the bottom or down the left side, each closed until its command asks for it (`ThumbnailStrip.place`). Termina splits a tab into **panes** instead: a pane is picked up with a drag while `Ctrl+Alt+Shift` (`Cmd+Option+Shift` on macOS) is held, since every smaller chord is spoken for (`PaneDrag`), and a maximized pane wears a "Pane 2 of 3" badge so the others are known to still be there.



Editora: the Project tool window docked right of the editor, its stripe on the window's right edge, and the bottom stripe's labeled buttons under the editor.



Mirada: the info panel on the right and the thumbnail strip along the bottom, both commands.



Torrenta: the filter sidebar left of the list and the detail pane beside the list, on its right.

Spec

Property	Value	Reference
Tab strip	`-color-bg-subtle`, padding 3 10 0 10, tabs bottom-aligned so the active one touches the editor; no bottom rule	`.tab-pane > .tab-header-area`
Tab	Transparent, square corners, padding 2 11 3 11; hover `-color-bg-inset`	`.headers-region > .tab`
Active tab	Two background layers: `-color-border-default` outside, `-editor-surface` inset 1 1 0 1; title bold `-color-fg-default`; glyph `-color-accent-fg`	`.tab:selected`

Property	Value	Reference
Title	Inter 14 px, `-color-fg-muted`, in a graphic label; the full path as tooltip	`.tab .tab-title`, `TabHeaders.tooltip`
Unsaved	`•` prefix, `-state-amber`, italic	`.tab-header.dirty .tab-title`
Pinned	`Icons.pin()` in the header, bold; first in the strip	`.tab-header.pinned`, `togglePin`
Read-only	`-color-fg-muted`, italic	`.tab-header.read-only`
Drag	Dragged header at 0.4 opacity; 2 px `-color-accent-emphasis` on the landing edge	`.tab-dragging`, `.tab-drop-before`, `.tab-drop-after`
Closed tabs	Twenty, newest first, one entry per file, with the caret	`ClosedTabs.DEFAULT_CAPACITY`
Switcher	26 px rows, 12 visible, width fixed per open	`Switcher.CELL_HEIGHT`, `MAX_VISIBLE`
Tool stripe	30 px wide, `-color-bg-subtle`, 8 px spacing, no edge; buttons min 24, padding 9 5, radius 6	`.tool-stripe`, `.tool-stripe-button`
Stripe states	Open: `-color-accent-subtle` fill, accent glyph. Focused window: `-state-accent-hover`	`:open`, `:tw-active`
Bottom stripe	Label 14 px beside the glyph, padding 3 8; open = accent, bold	`.tool-stripe-horizontal .tool-stripe-button`
Panel	`-color-bg-subtle` ground, radius 0 0 8 8, no border	`.tool-window`
Panel header	Padding 6 6 6 12, radius 7 7 0 0, no rule under it; title 14 px bold muted with the window's glyph; active: `-color-accent-subtle` band, accent title	`.tool-window-header`, `.tool-window:active`
Panel content	`-color-bg-default`, padding 8; its tree or list draws no frame of its own	`.tool-window-content`
Per side	Two windows at most, stacked in an inner SplitPane	`MAX_PER_SIDE`
Persisted	Open state, sides, order, visibility, floating windows, divider positions	`WorkspaceState` in `workspace-state.json`

Menus

Two surfaces show the same rows: the **menu bar**, a browsable map of every command, and **right-click menus**, what can be done with the thing under the pointer. A row built from the command registry takes its title, icon, chord and enabled state from the command, so it never drifts from the palette or the toolbar. Editora builds its menu bar that way (`ui/MenuBarModel.java`); its right-click menus are still built by hand, each row with its own title, glyph and action (`LazyContextMenu.item`). The app template's `ui/CommandMenu.java` builds a right-click menu from registered command ids. Reference: Editora's `ui/MainMenuBar.java` (menu bar), `ui/ContextMenuDismissal.java` , `ui/LazyContextMenu.java` , `ui/RowContextMenu.java` and `ui/TabContextMenu.java` (right-click menus), and the `.menu-item` and `.context-menu` rules in `app.css` .

Look and feel

Both surfaces draw their rows the same way, and the sizes in the table are the ones at the default zoom. A menu sets its own font size on `.context-menu` , 14 px at the default zoom, from the one value the UI zoom sets (WIN-12), so it never inherits its owner's: a popup takes the font of the node it was opened from, and `em` would resolve against a file tree's or an 18 px toolbar's size (Editora's toolbar menu once rendered a third larger than every other menu in the app).

Property	Value	Reference
Font	Inter 14 px at the default zoom, regular, in every row and label; set on `.context-menu` from the value the UI zoom sets, never inherited from the owner	`.context-menu` , `.context-menu .menu-item .label` in `app.css` ; `MainMenuBar.UI_FONT_SIZE`
Row	34 px high: padding 8 × 12, the same in every theme	`.menu-item { -fx-padding: 8px 12px }`
Icon column	22 px on every row, so the titles line up: a row's glyph, a check row's tick, or empty; 8 px to the title	`MainMenuBar.ICON_COLUMN` , `rowContent`
Icon	Optional; where a row has one, the command's glyph, the same one as on the toolbar and in the palette (Save is the same floppy everywhere), filled `-color-fg-muted`	`Icons` , `MenuBarIcons` , `.toolbar-icon`
Title	Inter 14 px, `-color-fg-default`	`.menu-item-title`
Chord	Right-aligned in its own column, Inter 11 px, in the keyboard color (`-state-key` , brace periwinkle); at least 18 px from the title column; in every menu, right-click menus included	`.menu-item-chord` , `alignChordColumns`
Submenu row	Laid out like a command row, with an invisible chord as wide as the menu's widest, so the arrow lands at the right edge	`nestedMenu` , `.menu-submenu`
Card	Radius 8; shadow `rgba(27,30,42,0.28)` , the value of a token defined on `.root` (ICON-08), radius 16, **offset Y 16**	`.context-menu`
Danger item	Text in `-color-danger-fg`	`.menu-item.danger`
Disabled item	The theme's dimmed label, and the icon dimmed with it	Termina's `.menu-item:disabled .menu-icon`
Separators	Never two in a row and none at either end: when items hide, their orphaned separators hide too	`LazyContextMenu.collapseSeparators` , Torrenta's `MenuSeparators`

The shadow's offset is not cosmetic. A menu is a native window sized to its root *including the effect*, so a shadow that reaches above the menu is invisible window over the bar the menu hangs from, and the pointer lands on it instead of the next menu button. MENU-18 asks for an offset of at least the radius; Editora pins it with `MenuPopupBoundsFxTest` .

The rules of this section hold for every menu the app draws itself: the menu bar, right-click menus, the drop-down of a menu button or a split button, and the toolbar's overflow. The menus a platform draws (the macOS system menu bar, a tray icon's menu) show text only; there the chord is appended to the title (MENU-02) and the tray shows neither.

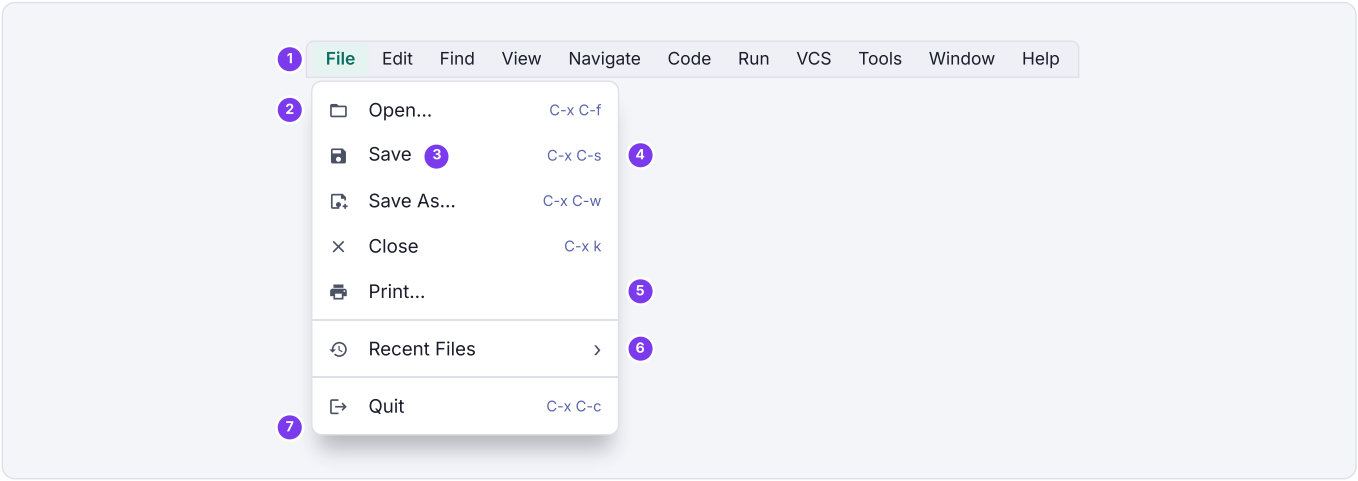
- **MENU-14** `MAY` A row of a menu the app draws may carry an icon, where a glyph depicts what the row does or names: a command row its command's glyph, a row that names a thing rather than an action (a recent file, a branch, a project) that thing's icon, a submenu its family's. A row with no fitting glyph goes without one rather than borrow another command's, and a check or radio row carries none, because its mark fills the icon column.
  - *Why:* a glyph the eye already knows from the toolbar and the palette (ICON-20) finds a row before its title is read, but only when it depicts the row: an icon forced onto every row ends up saying something else, and an empty cell says nothing wrong.
  - *Check (code):* where a row the app builds carries a glyph, it comes from the lookup the toolbar and the palette use, keyed by the row's command, and no row wears another command's glyph. A row without one is not judged.
  - *Reference:* Editora `ui/MenuBarIcons.java` , `editor/MenuIcons.java` , `ui/TabContextMenu.java` : about half its menu-bar rows wear a glyph, and `MenuBarIconsTest` holds that a command without a fitting glyph gets none. Termina's tab menu has an icon on every row. The app template gives every row of its menus the command's glyph (`inWindowRow` in `ui/AppMenus.java` , from `BY_COMMAND` in `ui/Icons.java`), and so does Mirada (`MenusFxTest` , `everyRowOfEveryMenuHasAGlyph`).
- **MENU-15** `MUST` In every menu the app draws itself, a row whose command has a chord in the active keymap shows that chord, in right-click menus, drop-downs and the toolbar's overflow as in the in-window menu bar: right-aligned in the menu's chord column, in the keyboard color, the one color the app keeps for chords and key chips (Editora's token for it is `-state-key` ; ICON-06 says how the color is made), at the size MENU-16 gives. An unbound command leaves the cell empty, never a wrong chord, and a keymap switch rewrites every cell. The menus a platform draws are outside this rule: the macOS system menu bar, whose rows append the chord to the title (MENU-02), and a tray icon's menu, which shows none.
  - *Why:* the menu is where a user learns the keys. A right-click menu that hides them teaches the mouse, while the palette and the toolbar's tooltips already show the same chord.

- *Check (code):* in the in-window menu bar, in every right-click menu and in every drop-down, a row that runs a command has a chord cell, right-aligned in one column of the menu, that the code fills with the active keymap's display chord for the command's id and leaves empty for an unbound command: a label of the row's own (Editora's `menu-item-chord`) or, outside the menu bar, the item's accelerator text, never a chord typed into a string. The cell is filled again on each showing or when the keymap changes, and the stylesheet draws it in the app's keyboard color at MENU-16's size. An app whose menu rows run only commands that have no chord (nux's drop-downs: its one key table binds none of their commands) has nothing to show, and passes. A row of the macOS system menu bar is not judged here, since MENU-02 has its chord appended to its title, and neither is a tray icon's menu. A test that binds a command, opens a right-click menu that lists it, finds its chord, and finds the new chord after a keymap switch is the surest evidence.
- *Reference:* Editora `MainMenuBar.rowContent` and `alignChordColumns`, `.menu-item-chord` in `app.css`, for the menu bar. Its editor, editor tab, preview and output-console right-click menus show them too: a row tagged with the command it does (`editor/MenuChords.java`, `tag`) carries that command's chord as an accelerator that only displays, re-read from the active keymap each time a right-click menu opens (`refresh`), and `.accelerator-text` in its `app.css` draws it in `-state-key` at 11 px (`MenuChordsTest`, `ContextMenuFontFxTest`). Editora doesn't meet the rule yet: its other right-click menus and its drop-downs tag no row, so the Copy, Select All, Add Bookmark and Add Note of a Project Map card's menu show no chord (`ui/ProjectMapPreview.java`). The app template's right-click menus show them: `ui/CommandMenu.java` draws each row as the menu bar's (`AppMenus.inWindowRow`) and re-reads its chord on every showing (`refresh`), in `-state-key` at 11 px (`.menu-item-chord` in its `app.css`); its `MenuLookFxTest` (`aRightClickMenuShowsEachChordInTheKeyColorAndFollowsTheKeymap`) opens a right-click menu, reads the chord and reads the new one after a keymap switch. Termina's menu bar colors its chords `-color-fg-muted`, not a keyboard color (`app.css`).

- **MENU-16** `MUST` Every menu the app draws itself (the in-window menu bar, right-click menus, the drop-down of a menu button or a split button, the toolbar's overflow) draws its titles in Inter 14 px regular and its chords in Inter 11 px at the default zoom, scaled by the UI zoom (WIN-12), whatever node it was opened from: the size is set on `.context-menu` and its rows' labels from one value the zoom sets, never inherited from the node. The macOS system menu bar and a tray icon's menu are the platform's, drawn in the platform's font, and outside this rule.
  - *Why:* a popup inherits the font of the node it was shown from, and `em` resolves against it: Editora's toolbar menu once rendered at the bar's 18 px, a third larger than every other menu, and a file tree's or a monospace editor's font leaks the same way.
  - *Check (code):* the stylesheet sets the font family and the size on `.context-menu` and on its rows' title and chord labels from one value, in pixels and never in `em`, so the font of the node a menu was opened from cannot reach it; and the code that applies the UI zoom rewrites that value, so the 14 px and the 11 px scale with the zoom. A test that opens a menu from the toolbar, an editor and a tree row and reads Inter 14 px from each row's title, and 14 px times the zoom after the UI zoom changes, is the surest evidence.
  - *Reference:* Editora `.context-menu` and `.menu-item-chord` in `app.css`, `MainMenuBar.UI_FONT_SIZE`, `ToolbarMenuFontFxTest` and `ContextMenuFontFxTest`. Editora writes 14 px and 11 px as absolute pixels and has no UI zoom yet (WIN-12), so its menus keep their size whatever opens them but don't scale. The app template meets it: `.context-menu` and its rows' labels in its `app.css` are pinned to Inter 14 px and the chord to 11 px, in pixels, which its UI zoom multiplies (`ui/UiZoom.java`), and its `MenuLookFxTest` (`everyMenuDrawsInInterFourteenWhateverOpenedIt`, `theUiZoomScalesEveryMenu`) holds both.
- **MENU-18** `SHOULD` At the default zoom a menu row is 34 px high (padding 8 × 12), its icon column is 22 px wide with 8 px to the title, and its chord starts at least 18 px after the title column. The popup's card has a radius of 8 and a shadow of radius 16 at offset Y 16: the offset is at least the radius.
  - *Why:* one row in every app is what makes the menus one family's. And a menu is a native window sized to its root with the effect included, so a shadow that reaches above the menu is invisible window over the bar the menu hangs from, and the pointer lands on it instead of the next menu button.
  - *Check (code):* the stylesheet sets the row's padding and the popup's radius and shadow, with the shadow's offset no smaller than its radius; the row's layout sets the icon column's width and the chord's margin.
  - *Reference:* Editora `.menu-item` and `.context-menu` in `app.css`, and `MainMenuBar.ICON_COLUMN` and `rowContent`; `MenuPopupBoundsFxTest` holds the shadow below the popup. The app template draws the same card and row: `.context-menu` in its `app.css` (radius 8, a shadow of radius 16 at offset Y 16, rows padded 8 × 12) and `inWindowRow` in its `ui/AppMenus.java` (a 22 px icon column, `ICON_SLOT`, 8 px to the title and an 18 px chord margin). Termina and Torrenta get the row's padding from the Editora themes they ship, and keep those themes' popup: a radius of 4 and a shadow of radius 6 (8 in the dark theme) at offset Y 2 (`.menu-item` and `.context-menu` in each one's `editora-light.css` and `editora-dark.css`). Termina's `ui/MenuRows.java` has the same 22 px column and 18 px chord margin. Mirada and nux write no radius or shadow for `.context-menu` and take their AtlantaFX theme's popup (Cupertino, Primer). Modula's `modula.css` gives `.context-menu` a radius of 8 and a shadow of radius 20 at offset Y 6, and its rows a padding of 4 × 8.

The menu bar

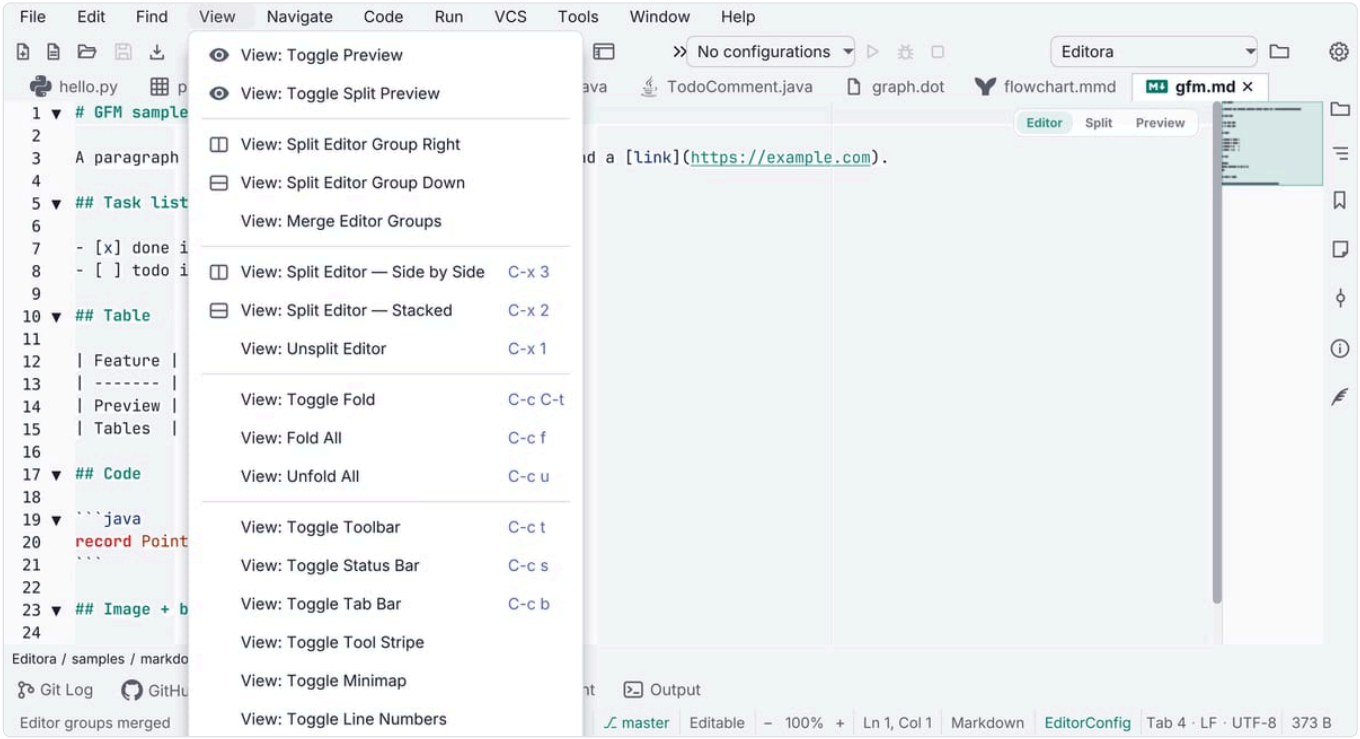
Applies to: an app with a menu bar.



1. **Menu button.** Padding 3 × 10 on a bar with 4 px side insets; the bar and the toolbar share one `-color-bg-subtle` band with no rule under it.
2. **Icon column**, 22 px. A row may wear the glyph that depicts its command; where none fits, the cell stays empty and the titles still line up.
3. **Title**, 14 px Inter, the command's title with its category prefix dropped ("Save", not "File: Save") unless two rows would then read the same. Editora drops the prefix only where two words or more are left ("File: Save As..." reads "Save As..."), so its File menu still reads "File: Save" (`editor/MenuTitles.java`).
4. **Chord column.** Inter 11 px, periwinkle, right-aligned into one column per menu (not app-wide), read from the active keymap. Drawn by the app, not a JavaFX accelerator: Emacs chords are multi-key, which a `KeyCombination` cannot express, and on macOS an accelerator would bypass the key dispatcher.

5. **Unbound command.** An empty chord cell, never a wrong chord. A live keymap switch rewrites every cell.
6. **Submenu row.** Icon, title and an invisible chord as wide as the menu's widest, so its arrow sits at the right edge instead of halfway across.
7. **Card.** Radius 8; the shadow falls below the menu (offset ≥ radius), so nothing invisible covers the bar.

- `MENU-01` `MUST` Every menu-bar item is a command from the registry, with the command's title (KBD-03), its chord from the active keymap and its enabled state from the command's gate. A row may show the title without its category prefix ("Save As..." for "File: Save As..."); it never carries a title of its own. The bar assembles and refreshes; it holds no behavior of its own.
  - *Check (code):* the bar's model lists command ids only, beside the separators and the submenus that group them, and every row runs its id through the registry. A row that is no command id fails: one built by hand, with a title and a handler of its own.
  - *Reference:* Editora `ui/MenuBarModel.java` over `command/CommandRegistry.java`, `ui/MainMenuBar.java`: the model's entries are command ids, separators and the titles of its submenus. Torrenta and Termina don't meet the rule yet. Rows of Torrenta's View menu, the Command Palette, Toolbar, Sidebar and Simple Mode rows among them, are built by hand with `menu.*` titles and handlers of their own (`ui/MainWindow.java`). Termina's model (`ui/MenuBarModel.java`) holds rows that are no command id: the New Tab With Profile submenu, whose rows carry each profile's name and open the tab from a handler of their own (`profileBarRows` in `ui/TerminalWindow.java`), and, off macOS, the zoom row, one row of buttons for its four zoom commands.
- `MENU-02` `MUST` Menu-bar rows use no `KeyCombination` accelerators. In the in-window bar each row is a `CustomMenuItem` laying out icon, title, spacer and chord. On the macOS system menu bar (`useSystemMenuBar`) the chord is appended to the title and rows carry no icon.
  - *Why:* Emacs chords are multi-key, which a `KeyCombination` cannot express, and on macOS an accelerator would bypass the key dispatcher. The native macOS menu draws text only and no graphics, which is also the platform's convention.
  - *Check (code):* no menu-bar row calls `setAccelerator`; each row of the in-window bar is a `CustomMenuItem` whose content lays out the icon, the title, a spacer and the chord; and where the bar is the macOS system menu bar, a row's text is the title with the chord appended and the row has no graphic.
  - *Reference:* Editora `ui/MainMenuBar.java` (`rowContent`, `refresh`).
- `MENU-03` `SHOULD` Enablement is re-read every time a menu opens, and titles and chords on a keymap or locale change.
  - *Why:* a menu built before Git finished detecting a repository must not stay grayed out.
  - *Check (code):* each menu's on-showing handler reads every row's enabled state from its command again, and the code that switches the keymap or the locale rewrites every row's title and chord. A test that changes a command's state after the window is built (a repository detected late) and finds the row enabled is the surest evidence.
  - *Reference:* Editora `MainMenuBar.refreshEnablement`, `MenuBarEnablementFxTest`.
- `MENU-04` `SHOULD` An app with a Simple mode shows a shorter menu in it, rebuilt only when the mode actually changes.
  - *Check (code):* the bar rebuilds only when the simple flag differs from the current one.
  - *Reference:* Editora `MainMenuBar.setSimple`, `MenuBarModel.menus(simple)`.
- `MENU-17` `SHOULD` The in-window menu bar, the one an app shows on Windows and Linux (WIN-03), is reachable from the keyboard in every keymap. JavaFX opens the bar's first menu on `F10` and on a lone `Alt`; a keymap that binds `F10` to something else binds a command that focuses the menu bar instead. So does a terminal app, where a plain `F10` goes to the program it runs. The macOS system menu bar is reached with the OS's own keys and asks nothing of the app.
  - *Why:* the menus are a keyboard user's map of the app; a chord that never reaches them hides it.
  - *Check (by hand):* on Windows or Linux, in each keymap, reach the first menu with the keyboard alone and walk the bar with the arrows.
  - *Reference:* no app yet. Editora's CUA, VS Code and Sublime keymaps bind `F10` to `debug.stepOver` (`keymaps/cua.json`), and no command focuses the bar. Termina leaves a plain `F10` to the program in the terminal (`ui/ViewKeys.java`) and has no such command either.



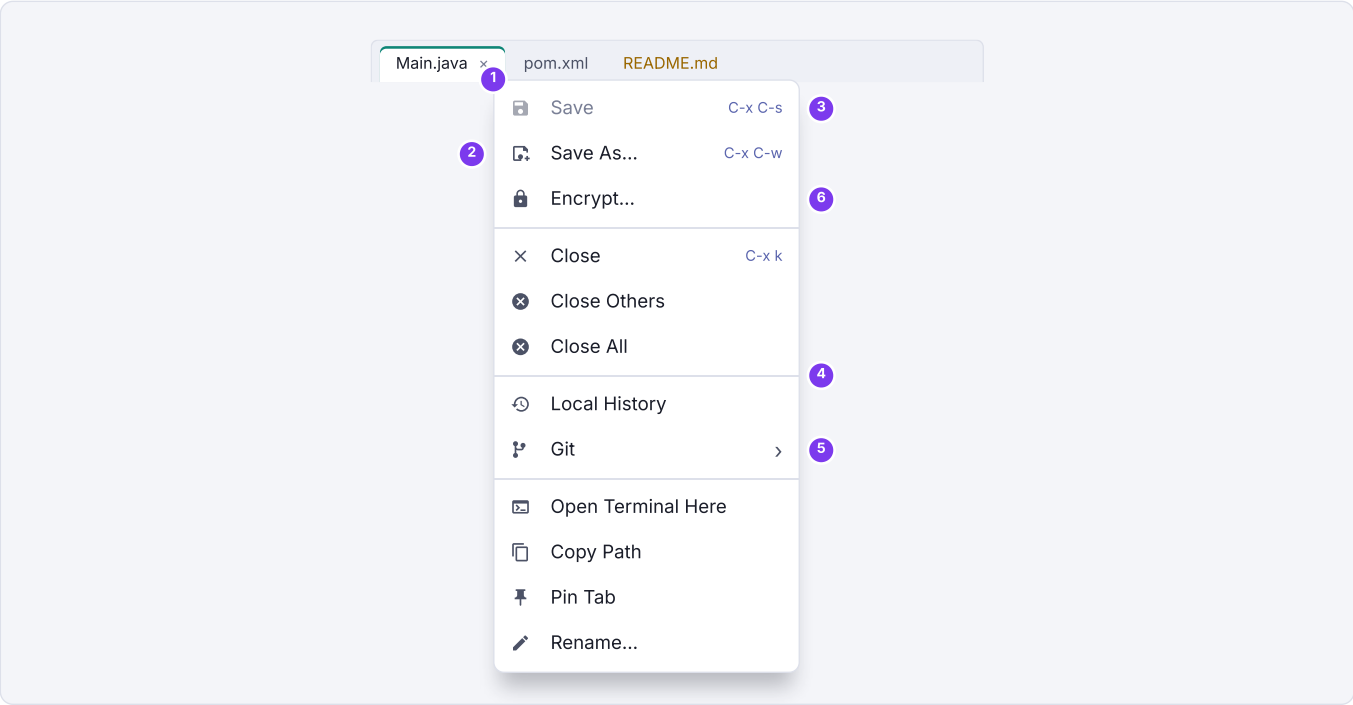
Editora's View menu in an earlier build: chords in one periwinkle column, but every title repeats the menu's name.

Right-click menus

Applies to: an app with a right-click menu: a menu the app builds and opens on a right-click, the Menu key or `Shift+F10`, over something in one of its windows. The stock Cut, Copy and Paste menu that JavaFX gives a text field does not make an app one, and neither does a tray icon's menu, which the platform draws: nux has both and no right-click menu.

- **`MENU-05`** `MUST` In an app with a right-click menu, every list row, tree row, tab, editor and read-only console has one, with the actions for *that* thing, and every action in a right-click menu is a command (KBD-01). The rule covers the lists, trees, tabs, editors and read-only consoles in the app's windows. A log shown as text, the Debug Log among them, is a read-only console and has its right-click menu. A list that only navigates, switching what its window shows (a Settings sidebar, a list of pages), has no actions of its own and is outside the rule, as are the overlay list of a picker or a palette and a plain text area in a dialog.
  - *Check (code):* list the lists, trees, tab strips, editors and read-only consoles in the app's windows, the text of the Debug Log window and of any other log shown as text among the consoles, and find the right-click menu each one installs; then read every row of every right-click menu the app builds: its action is one a registered command runs too, by the command's id through the registry or by calling what the command calls, so the palette reaches it. A row whose action no command runs fails, as does a list, tree, tab strip, editor or console with no menu; the stock Cut, Copy and Paste menu that JavaFX gives a text area is not one. A list that only switches what its window shows (a Settings sidebar, a list of pages), the overlay list of a picker or a palette and a plain text area in a dialog are not judged.
  - *Reference:* the app template's `ui/CommandMenu.java`, which builds each menu from registered command ids and runs a row through the registry (`onRow` for a list row, `onConsole` for a text area): the tabs and the editor (`ui/DocumentTabs.java`), the Output console (`ui/MainWindow.java`), the Debug Log (`ui/DebugLogWindow.java`), the Documents list, the Message Log's rows and Settings' pages and key bindings (`RightClickMenusFxTest`). Editora's `ui/TabContextMenu.java` and `ui/ConsoleContextMenu.java` are menus this rule describes (`ui/RowContextMenu.java` builds none: it sends the keyboard's request to the selected row's own menu), but their rows are built by hand and call the window's methods, not a command id, so nothing holds that each row has a command. Editora, Termina, Mirada, Torrenta and Modula don't meet the rule yet, in these places among others. In Editora the trees and the list of the Hierarchy, References and Undo History panels have no right-click menu (`ui/HierarchyPanel.java`, `ui/ReferencesPanel.java`, `ui/UndoHistoryPanel.java`), and the Customize Toolbar... row of the toolbar's menu turns on a mode that no command does (`showContextMenu` in `ui/ToolbarCoordinator.java`). In Termina the list of the Message Log has no menu (`ui/MessageLogWindow.java`), and neither has Mirada's (`ui/MessageLogView.java`). The Debug Log windows of Editora, Torrenta, Termina and Mirada show the log in a read-only `TextArea` and give it no menu (Editora's `ui/DebugLogWindow.java`, Torrenta's `ui/DebugLogWindow.java`, Termina's `ui/DebugLogWindow.java` and Mirada's `ui/DebugLogWindow.java`). In Torrenta the rows of the files and peers menus, Ban Peer among them, run handlers that no command reaches (`ui/DetailPane.java`). In Modula the Rename... and Remove rows of a preset chip's menu (`ui/PresetBar.java`) run no command, and its command list has neither (`ui/RadioPane.java`, `commands`).

The mockup is an editor tab's menu, shortened; Editora's has twenty-one rows.



1. **Anchored at the pointer.** The menu's top-left corner is the click. From the keyboard (Menu key or `Shift+F10`) it opens under the selected row, a quarter of the way in and never off a narrow one, or just below the caret in an editor; never in the middle of the control.
2. **Icon and chord.** A row wears its command's own icon where one fits, and its chord when it has one, in the same periwinkle column as the menu bar's. A `CheckMenuItem` has no icon: its tick lives in the graphic column.
3. **Grayed, not hidden.** An action that does not apply right now (Save on an unchanged file, Reveal on an unsaved one) stays in place, dimmed. Only a row that cannot apply to this thing at all is hidden (the Git submenu on an untitled buffer).
4. **Separators** group the rows; one that hiding has left alone hides too.
5. **Submenu** for a family of actions (Git: stage, unstage, revert, compare, history), with its own icon.
6. **Labels follow state.** Encrypt... becomes Lock File on an encrypted buffer, Pin Tab becomes Unpin Tab. Decided on every showing, not when the menu was built.

Behavior

- **`MENU-06`** `MUST` One right-click menu at a time: a right-click while a menu is open replaces it, and the first one closes before the second shows.

- *Why:* JavaFX stacks a second menu beside the first when each row owns its own `ContextMenu`, and the stale one is then no longer dismissible.
- *Check (code):* the code that opens a right-click menu closes the one already showing before the next appears, in every window: the app keeps one menu and shows it again at the new place, or hides the showing menu in the handler of the press or the request that opens the next. Menus that each row or cell owns, with nothing to close the one shown last, fail. A test that right-clicks another row with a menu open and finds one menu showing is the surest evidence.
- *Reference:* Editora `ui/ContextMenuDismissal.java` closes every open right-click menu on the press that opens the next (`ContextMenuDismissalFxTest`), and its toolbar keeps one menu and shows it again, so a second right-click on the bar moves it (`ToolbarCoordinator`, `ToolbarContextMenuDismissFxTest`); Termina's `ui/OneMenu.java` closes every other menu as one starts to show (`OneMenuTest`), and the app template's `RightClickMenusFxTest` (`aRightClickOnAnotherRowLeavesOneMenu`) holds it for a list's rows. Modula's `RadioPane` hides its own previous menu before it shows a new one, but Modula doesn't meet the rule yet: each preset chip owns a menu (`ui/PresetBar.java`), and no code closes the pane's menu when a chip's opens: the pane's press filter skips the right button, and the chip answers the menu request before the pane's handler can.

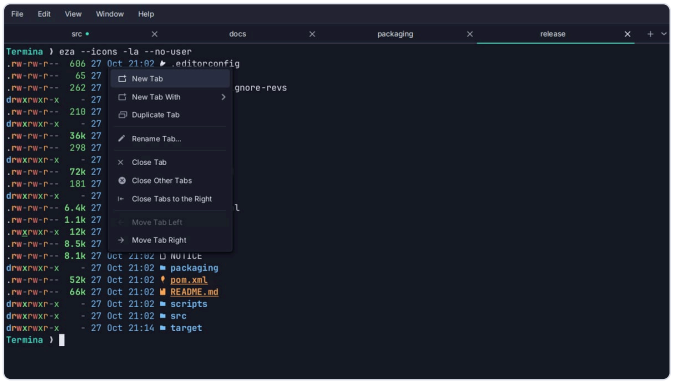
- **`MENU-07`** `MUST` A press anywhere outside a right-click menu closes it, including inside the control the menu was opened on. The press is not consumed: a click in a list still selects the row, and a right-click on another row opens that row's menu straight away. Drop-downs opened by a plain click (menu buttons, choice boxes, the menu bar) keep the stock behavior, and a menu that turned auto-hide off is left alone.
  - *Why:* JavaFX's auto-hide deliberately ignores a press on the menu's *owner node*, so a stock menu on a tree or a text area stays open when the user clicks another row or the text.
  - *Check (code):* the app installs a mouse-press filter, on every window or on the scene of each window that has a right-click menu, that hides a showing right-click menu on a press outside the menu, a press inside its owner control included, and does not consume the press; a press in another window of the app closes the menu too, by the same filter or by the stock auto-hide. A right-click menu left with the stock auto-hide alone fails, since JavaFX ignores a press on the owner node. A test that presses inside the owner control and in another window and finds the menu closed is the surest evidence.
  - *Reference:* Editora `ContextMenuDismissal.install()`, called once in `App` at startup for every current and future window; `ContextMenuDismissalFxTest`, and for the toolbar's menu, whose owner is the whole bar, `ToolbarContextMenuDismissFxTest`. The app template installs the same class at start-up. Mirada turns auto-hide off on its three menus and closes them from a press filter on the scene of each window that has one (`ui/Popups.java`, `closeOnPressOutside`). Termina's terminal view hides its menu from its own press filter, but its tab menus have the stock auto-hide alone (`ui/TerminalWindow.java`); so have Modula's preset chips (`ui/PresetBar.java`), beside a pane menu that its own filter closes, and every right-click menu in Torrenta. nux has no right-click menu.
- **`MENU-08`** `MUST` Escape closes a right-click menu and focus goes back where it was. In a terminal app the Escape that closes the menu does not reach the shell.
  - *Check (by hand):* open a menu, press Escape, and type: the keys reach the control that had focus.
  - *Reference:* the stock `ContextMenu` hide-on-Escape (the right-click menus' Spec); Termina's `onKeyPressed` filter hides the menu and consumes the key (`TerminalView`).
- **`MENU-09`** `MUST` The Menu key and `Shift+F10` open the right-click menu of the selected row, scrolled into view first, so every row action a mouse can reach the keyboard can too. In a view with no rows (an editor, a terminal, an image) they open the focused view's menu. The chord is the same on macOS, where keyboards have no Menu key: `Shift+F10`, with no `Cmd` form.
  - *Why:* those keys arrive at the focused control, not at a row's cell, so a menu installed on cells never saw them.
  - *Check (code):* each list, tree or table whose rows have a right-click menu answers the keyboard's request (JavaFX's `ContextMenuEvent` with `isKeyboardTrigger()`, which it raises for both keys, or a key handler of the app's own for the Menu key and `Shift+F10`) by scrolling the selected row into view and showing that row's menu anchored to the row; a view with no rows shows its own menu for the same keys; and no branch gives macOS another chord. A test that sends the request to a tree and a list and finds the selected row's menu, anchored to that row, is the surest evidence.
  - *Reference:* Editora `RowContextMenu.install(tree)`, `RowContextMenuFxTest`: it answers the request JavaFX raises for those keys (`ContextMenuEvent.isKeyboardTrigger`). Termina's `ui/ViewKeys.java` and Mirada's `Popups.isMenuKey` read the key press themselves, the same on every OS. One of Editora's keymaps gives the chord to a command as well: IntelliJ's binds `S-f10` to `run.rerun` (`keymaps/intellij.json`), a clash `RowContextMenuFxTest` does not cover: it raises the request itself and presses no key. The app template's `ui/RowContextMenu.java` answers the same request for a list's rows (`RightClickMenusFxTest`, `theMenuKeyOpensTheSelectedRowsMenuAtThatRow`).
- **`MENU-10`** `SHOULD` A right-click selects first. In an editor, a right-click outside every selection moves the caret to the click (so Paste lands there); inside a selection it keeps carets and selections (so Cut and Copy act on what was clicked). From the keyboard the target is the caret. In a list, the menu is the clicked row's.
  - *Check (by hand):* in an editor, select a word, right-click elsewhere and paste: the text lands at the click; then right-click inside a selection and copy: the selection is what was copied. In a list, select one row and right-click another: the menu that opens is the clicked row's, and its actions act on that row. An app with no editor is judged on its lists alone.
  - *Reference:* Editora `editor/ContextMenuTarget.java`.
- **`MENU-11`** `SHOULD` A menu attached to something that exists once per open file is built when first shown, and set up for each showing: an `onShowing` handler sets the labels, enablement and visibility for *this* showing, then collapses the separators.
  - *Why:* such a menu is built for every file and opened for almost none.
  - *Check (code):* a menu that exists once per open file is not built with the file: its items are created in the handler of its first request, and an on-showing handler sets their labels, enablement and visibility and then collapses the separators (Editora's `LazyContextMenu.of(build)`).
  - *Reference:* Editora `ui/LazyContextMenu.java`, `LazyContextMenuFxTest`. A read-only console's menu is just Select All and Copy, both disabled while it is empty (`ConsoleContextMenu`).
- **`MENU-12`** `SHOULD` A destructive row is styled as dangerous, its text in `-color-danger-fg` (Editora's style class for it is `danger`). What the action does before it destroys anything is not this rule's: the Confirm dialog rules hold it (CONF-01, CONF-03: it asks first with a button that names it, or is undoable).
  - *Check (code):* rows that delete, discard or reset carry the style class that colors their text `-color-danger-fg` (Editora's `danger`). Whether the action asks first is judged under CONF-01 and CONF-03, not here.
  - *Reference:* Editora `.menu-item.danger` in `app.css`, `ui/BranchPopup.java`; the app template's `ui/AppMenus.java` (`DESTRUCTIVE`: the rows that clear a log, reset a binding or import a configuration take `danger`), held by `RightClickMenusFxTest` (`destructiveRowsAreStyledAsDangerous`).
- **`MENU-13`** `MAY` In a terminal view, a program that grabbed the mouse owns the right-click; `Shift`+right-click always opens the menu, and the keyboard request is never contested.
  - *Check (code):* the terminal view decides whether to show its menu from the mouse mode, Shift and the trigger: a right-click while a program has the mouse goes to the program unless Shift is held, and a request from the keyboard always shows the menu. A test of that decision is the surest evidence.
  - *Reference:* Termina's `TerminalView.shouldShowMenu`, `ContextMenuContentionTest`.

Spec

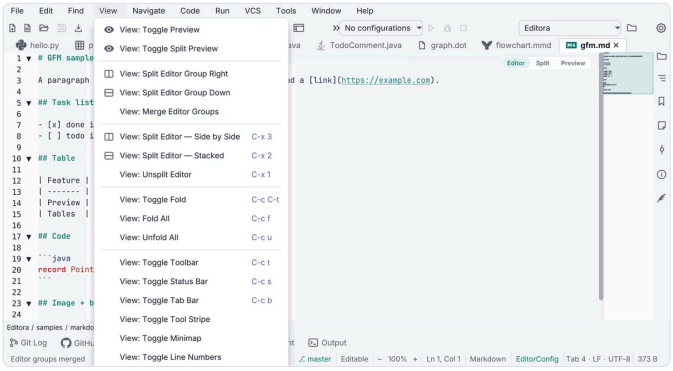
Property	Value	Reference
Kind	`javafx.scene.control.ContextMenu`, auto-hide on, hide on Escape on: a popup window, unlike the palette	`TabContextMenu`, `ConsoleContextMenu`

Property	Value	Reference
Opens at	The pointer; from the keyboard, the selected row's bottom edge at min(width ÷ 4, 48) px in, or under the caret	`RowContextMenu.anchorX`, `ContextMenuTarget`
Font	Inter 14 px at the default zoom, set on `.context-menu` from one value the UI zoom sets, so it never inherits the owner's font	`app.css`
Rows	34 px (padding 8 × 12); icon column 22 px; glyph scaled 0.8 from a 24 dp Material path, or a 16-unit line glyph	`Icons.ICON_SCALE`, `Icons.line`
Card	Radius 8, shadow radius 16 at offset Y 16	`.context-menu`
Dismissal	Any press outside the menu, in any window; the next right-click; Escape	`ContextMenuDismissal`, `ContextMenuDismissalFxTest`
Marker	A showing menu opened by a right-click or the Menu key carries the window property `editora.rightClickMenu`	`ContextMenuDismissal.RIGHT_CLICK_MENU`

In the apps



Termina's tab menu: an icon on every row, following Editora's convention.



Editora's in-window menu bar with the chord column, in an earlier build whose titles repeat the menu's name.

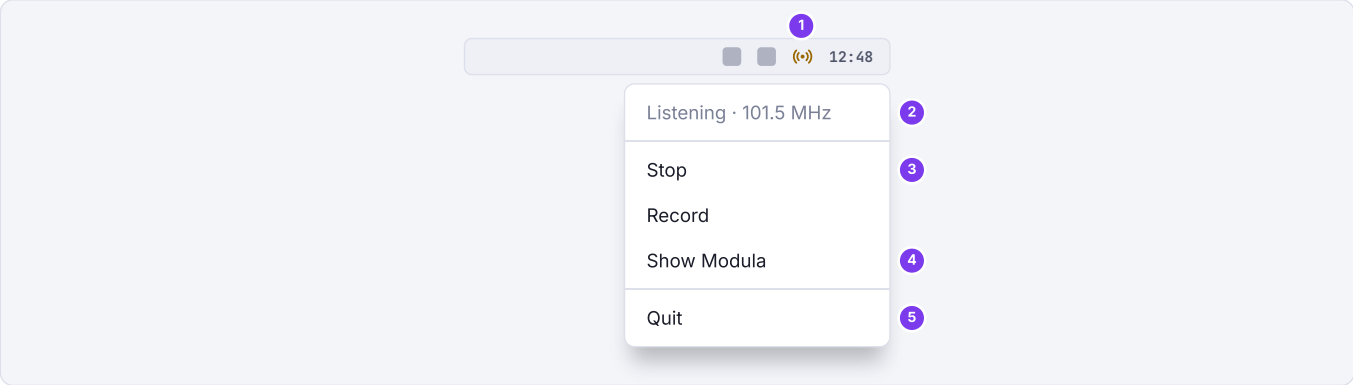
Tray and background

An app whose work goes on without its window (Torrenta's transfers, Modula's radio, nux's monitoring) lives behind a tray icon. Reference: Torrenta's `tray/` package (`Trays` , `TorrentaTray` , `SniTray` , `AwtTray` , `TrayIconRenderer` , `SniMenuModel`), `os/Autostart` with `os/AutostartSpec` , and `automation/Notifiers` . Modula's and nux's `tray/` packages are earlier copies of it with their own menus: on Linux both fall back to the AWT tray when the StatusNotifierItem fails, and `Trays.create` returns a tray or nothing, with no reason (nux `tray/Trays.java` , Modula `tray/Trays.java`). Only Torrenta has `Trays.Attempt` , the reason and no fallback. nux's `TrayController` holds the close policy.

Applies to: an app whose work goes on without its window, as Torrenta's, Modula's and nux's does.

For the others (Editora, Termina, Mirada) a tray is optional and none has one. Only Notifications applies to them, and only to an app that sends them.

The icon and its menu



1. **The icon** is the app's own mark, recolored by state, not a badge. Torrenta's arrow is teal while something downloads, muted when nothing does, red on an error; Modula's broadcast glyph amber while listening, ink when stopped, coral on a fault. The exception is an app whose state is a number: nux draws a rounded whole number (charge, load or minutes left) colored by severity, gray when disconnected, never a decimal like "101.5", which is unreadable at 22 px.
2. **Status line**, disabled, first. It is also the tooltip, because GNOME's AppIndicator extension renders a StatusNotifierItem's menu but not its tooltip, so the menu is the one place the live figures are sure to appear.
3. **The main actions** the window would offer, so the common jobs need no window: Listen becomes Stop, Record is grayed (not hidden) until the receiver runs; Torrenta's menu has none; nux offers Connect, Switch UPS and the instant commands, which still go through the app's confirmation.
4. **Show *App*** brings the window back, as a left click on the icon does.
5. **Quit**, after a separator. The only way to end the program apart from File → Quit and the system's Quit.

Tray menus are text only: AWT's `PopupMenu` and dbusmenu show no item icons, so the menu rules about icons and chords do not apply here. The rows come in the mockup's order: the status line, the actions, Show *App*, then Quit after a separator.

- **`TRAY-01`** `MUST` The tray is a StatusNotifierItem on Linux, with no AWT fallback, and AWT's `SystemTray` on Windows and macOS. The app creates it on a background thread, never on the FX thread, and no sooner than the code that shows the main window, so the first frame never waits for the tray.
  - *Why:* D-Bus and AWT can block, and neither is worth a stalled first frame. On Linux AWT's tray is the XEmbed one, which GNOME dropped in 3.26 and Wayland never had, so there is nothing worth falling back to.
  - *Check (code):* on Linux the code that creates the tray makes the StatusNotifierItem or reports that there is no tray, and no branch of it falls back to AWT's tray (Torrenta's `Trays.create` , which makes an `AwtTray` off Linux only); on Windows and macOS it makes the AWT `SystemTray` icon; and the main window runs it on a background thread that it starts as it shows itself, or later.
  - *Reference:* Torrenta `tray/Trays.java` (`create`) and `MainWindow.installTray` , which `show` calls and which starts a `tray-init` thread; the Spec table. Modula and nux don't pass yet: on Linux their `Trays.create` falls back to `AwtTray` when `SniTray` fails (Modula `tray/Trays.java` , nux `tray/Trays.java`).

Spec

Property	Value	Reference
Linux backend	A StatusNotifierItem on the session bus, found through `org.kde.StatusNotifierWatcher` : KDE natively, GNOME with the AppIndicator extension, X11 and Wayland alike. No AWT fallback in Torrenta: the XEmbed tray GNOME dropped in 3.26 and Wayland never had. Modula and nux still fall back to it	`SniTray` , `StatusNotifierWatcher` , `Trays.create`
Windows and macOS	AWT `SystemTray` , every mutation on the EDT, a multi-resolution image so a HiDPI tray picks a crisp variant	`AwtTray`
Created	Off the FX thread, after the first frame (D-Bus and AWT can block); in Torrenta the outcome is a tray or the one-line reason there is none, in Modula and nux a tray or nothing	`Trays.Attempt` , `MainWindow.installTray`
Icon	Pure Java2D from the branding SVG's geometry, at each size the backend asks for, so the panel entry and the dock entry are one mark	`TrayIconRenderer` , `TrayDisplay`
Attention	A state that needs a person sets the SNI status `NeedsAttention` ; otherwise `Active`	`SniTray.update`

Property	Value	Reference
Left click	Shows the window: SNI `Activate` and `SecondaryActivate`, a single primary click on the AWT icon	`SniTray.Item`, `AwtTray`
Right click	The menu; the host pulls it from the item's `Menu` property	`SniMenu`, `SniMenuModel`
Updates	Only when the value-equal snapshot changed: a repaint crosses a process boundary on Linux. Torrenta polls once a second, Modula nine times	`TrayDisplay`, `MainWindow.pushTray`
Package	Free of JavaFX: it takes callbacks, and the window marshals them onto the FX thread	`TorrentaTray`

Behavior

- **TRAY-02** `MUST` An app that lives behind a tray icon is fully usable when no icon comes up. No icon is a normal outcome (stock GNOME has no StatusNotifier host), not an error, and the main window then says in its status line why there is none and that closing the window quits the app.
  - *Why:* the bug they actually hit is silence: the box says close to tray, and closing quits.
  - *Check (code):* when the attempt to create the tray returns none, the code sets the main window's status message (the echo area of its status bar, where it has one) to the reason and to the fact that closing the window quits; and the tray's menu holds no action the window lacks.
  - *Reference:* Torrenta `MainWindow.installTray` ("No tray icon (…); closing the window quits Torrenta", `status.noTray`); Modula's status says closing the window will quit it (`RadioPane.reportNoTray`). nux doesn't pass yet: when no tray comes it shows the window, if it was hidden, and says nothing (`TrayController.onCreated`).
- **TRAY-03** `MUST` Closing the window hides it only while an icon is actually up and the setting says so, both read at close time; otherwise a close quits. `Platform.setImplicitExit(false)` is set only once a tray exists, and turning the tray off brings a hidden window back.
  - *Why:* the checkbox can change while the window is open, and hiding a window that leaves no icon behind is how a program becomes unreachable.
  - *Check (code):* the close handler tests that a tray exists and reads the live setting; `setImplicitExit(false)` sits after the tray is created.
  - *Reference:* Torrenta `MainWindow.setOnCloseRequest`, Modula's `installCloseBehaviour`, nux's `TrayController.closeHidesWindow` and `updateImplicitExit`.
- **TRAY-04** `MUST` The tray's Quit runs the app's one quit method (WIN-13), and a quit ends the program with the window showing or hidden to the tray: the close handler never takes a quit for a close and hides the window, and the method exits outright, since closing the last window no longer does (TRAY-03).
  - *Why:* a close hides a tray app's window, so a quit that goes through a close leaves the program running behind its icon. The tray's Quit also arrives while a dialog may be open, and quitting from inside a nested event loop aborted the JVM, which is why the quit method closes dialogs first.
  - *Check (code):* the tray's Quit callback runs the quit method on the FX thread; the method marks the quit, so the close handler doesn't hide the window, tears down when the window is already hidden, and calls `Platform.exit`. On macOS the app menu's and the Dock's Quit reach the method too (WIN-13): JavaFX turns them into a close request on every window (`GlassStage.requestClosingAllWindows`), and a tray app that doesn't catch Glass's quit action hides to the tray instead of quitting.
  - *Reference:* Torrenta `MainWindow.quit` for the tray and File → Quit: it sets `quitting`, which the close handler reads, closes open dialogs and waits for their loops, then closes the window or, when it is hidden, shuts down directly. It doesn't catch macOS's Quit yet, so there Quit hides the window. Modula's tray Quit runs `ModulaApp.quit` and nux's calls `Platform.exit` (`App`), as its `app.quit` command does; neither closes open dialogs first or catches macOS's Quit.
- **TRAY-05** `SHOULD` `--minimized` starts the app with no window in front, in one of two ways. With the tray setting on it comes up behind the icon: the window is not shown until the tray attempt has answered, and if the icon that was asked for does not come, the window is shown after all. With the tray setting off there is no icon to wait for, and the window starts minimized.
  - *Why:* a start at login can then never leave a program with no way back: a hidden window has its icon, and a minimized one is on the taskbar or in the Dock. A person who turned the tray off asked for the minimized window. A person who asked for an icon looks for the icon, so when none comes the window is what tells them, with the reason on its status bar (TRAY-02).
  - *Check (code):* with the tray setting on, a `--minimized` start leaves the window hidden until the tray attempt answers, and shows it when the attempt returns no tray; with the setting off it shows the window minimized and waits for nothing.
  - *Reference:* Torrenta `MainWindow.show` and `installTray`, which calls `showFromTray` when no icon comes. nux's `App` and `TrayController.onCreated` do the same with the tray setting on; with it off nux doesn't pass yet: it ignores the flag and shows the window. Modula has no `--minimized`.
- **TRAY-06** `SHOULD` Start at login writes the OS's own entry, and nothing else: an XDG autostart entry in `~/.config/autostart` (quoted `Exec`, `StartupWMClass` matching the installed launcher, `X-KDE-autostart-after=panel` so a tray host is up first), a LaunchAgent in `~/Library/LaunchAgents`, the `HKCU\…\Run` value on Windows (read together with `StartupApproved`, where Task Manager keeps the person's "off"). The entry is the one truth: turned off in GNOME Tweaks, KDE's Autostart page or Task Manager, the box shows unticked. The one setting beside it is *start behind the tray icon*, written into the entry as `--minimized`.
  - *Check (code):* the code that turns start at login on or off writes or removes one entry for the OS it runs on, and keeps no setting of its own for it: the XDG entry with a quoted `Exec`, the installed launcher's `StartupWMClass` and `X-KDE-autostart-after=panel`; the LaunchAgent; the `Run` value. The box's state is read back from the entry, with `StartupApproved` on Windows, and start behind the tray icon adds `--minimized` to the entry's command. A test of each platform's entry, byte for byte, is the surest evidence.
  - *Reference:* Torrenta `os/Autostart.java`, `os/AutostartSpec.java`, `AutostartSpecTest`. nux's `autostart/AutostartSpec.java` is the earlier version: the same three entries, without `X-KDE-autostart-after` and without reading `StartupApproved`.
- **TRAY-07** `SHOULD` Only the app writes the login entry, and only from an installed package: the installers never do it, and a run from source says it has nothing to start.
  - *Why:* an installer runs for every account.
  - *Check (code):* the entry's target comes from `jpackage.app-path`, and is absent when it is unset.
  - *Reference:* Torrenta `os/Autostart.java`.
- A start-at-login call may block on the file system or `reg.exe`, so it runs off the FX thread like any other I/O: PERF-01. Torrenta's run on their own threads (`autostart-read`, `autostart-write`, `autostart-refresh`), as does nux's (`nux-autostart`).
- **TRAY-12** `MAY` A dock badge: an app may put a figure of its own on the macOS dock icon through AWT's `Taskbar` and on a Linux launcher that honors Unity's `LauncherEntry` signal (KDE, Dash to Dock); nothing on Windows, whose taskbar badges only AWT's own windows. The badge is cleared when there is nothing to show and has its own switch. Torrenta's is the download rate on the dock and a count on the launcher, cleared when nothing downloads.
  - *Check (code):* the badge is set through AWT's `Taskbar` on macOS and through the `LauncherEntry` signal on Linux, and not at all on Windows; the code clears it when there is nothing to show (in Torrenta, when nothing downloads); and a setting of its own turns it off. A test of the badge's state is the surest evidence.
  - *Reference:* Torrenta `os/Badges.java`, `BadgeState`, `BadgeStateTest`.
- **TRAY-13** `SHOULD` A second launch brings the hidden window forward: the single-instance hand-off shows a window hidden to the tray before it opens what it was given.
  - *Check (by hand):* close to the tray, then launch the app again with a file.

- *Reference:* Torrenta `MainWindow.showFromTray`.

- **TRAY-14** `SHOULD` A tray makes AWT headful on Windows and macOS only, where the tray is AWT's (TRAY-01); on Linux the app stays headless with its tray on. CLI-09 holds the rule this is part of, for an app with a tray or without one: the whole list of uses that may make AWT headful, and that `main` decides it once, at start-up. Because it is decided once, a tray turned on later tells the user to restart ("restart to show the tray icon"), never that the tray is unsupported.
  - *Check (code):* the code that decides `java.awt.headless` at start-up counts the tray setting as a reason on Windows and macOS and never on Linux, and the code that creates the tray, finding AWT headless on Windows or macOS, reports that a restart will show the icon. A test of the decision per OS is the surest evidence.
  - *Reference:* Torrenta `os/AwtUse.java` (`headful`, which `App.main` asks once with the tray, badge and handler settings), `AwtUseTest` (`awtIsHeadfulOnlyWhereSomethingAwtIsWanted`); `Trays.usesAwt` and `TraysTest` for the tray alone, and `Trays.create`, which gives the restart reason (`tray.reason.restart`) when AWT was left headless. Modula and nux never set `java.awt.headless`, so AWT is headful on every OS.

Notifications

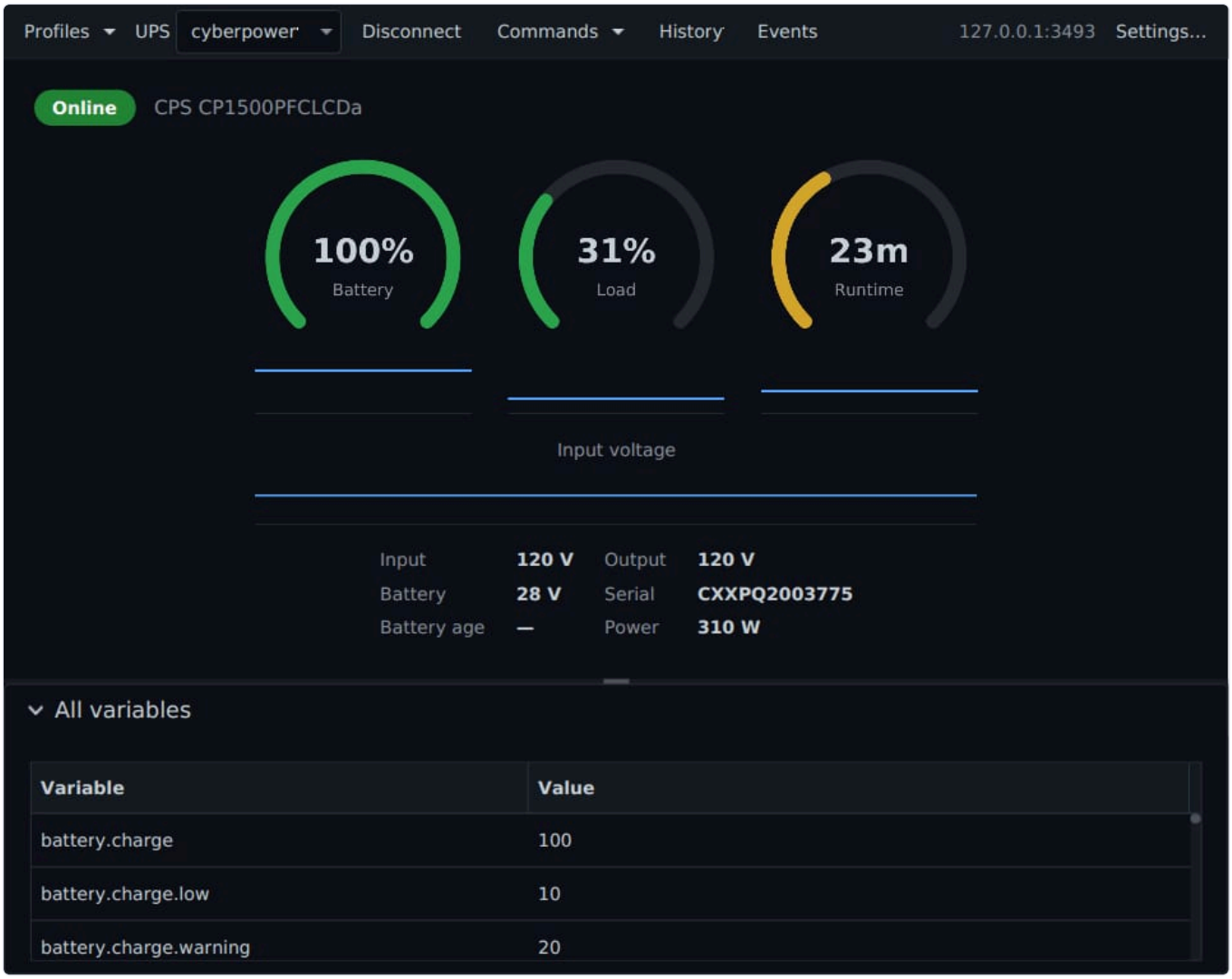
Applies to: every app that sends desktop notifications, with a tray or without one: Torrenta and nux for their own events, Editora and Termina for a program's OSC 9 and OSC 777.

- **TRAY-09** `MUST` Desktop notifications go through one notifier: `notify-send` on Linux, `osascript` with `display notification` on macOS, a PowerShell toast on Windows. On Linux an app may also fall back to `gdbus` against the freedesktop service where `notify-send` is missing; the rule does not ask for it. Each is a short program, and the text reaches the program as data (an argument, or an environment variable a fixed script reads), never as part of a script. It runs off the FX thread, and the call times out after ten seconds or less; a failure is logged and never breaks the app.
  - *Why:* text from a torrent name or a remote shell then cannot run anything.
  - *Check (code):* one class builds every notification's command line: `notify-send` on Linux, `osascript` on macOS, PowerShell on Windows. Each carries the title and the message as arguments or environment variables of their own, and the script it runs is fixed and never has the text concatenated into it. The program runs on a background thread with a timeout of ten seconds or less that kills it, and a failure is written to the app's log, never thrown further and never dropped: an exception from starting the program or waiting for it, a program that is not there to run, the timeout, and an exit code that is not zero. A test of each platform's command line is the surest evidence.
  - *Reference:* Torrenta `automation/Notifiers.java`, `ProcessNotifier`, `NotifierTest` (it ships `toast.ps1`, which takes the title and message as parameters); on Linux it runs `notify-send` and has no fallback. It runs each on a virtual thread with a ten-second ceiling and reports a failure to its event log, the first of a session only: the ones after it are dropped (`automation/Automation.java`, `notify`), so Torrenta doesn't meet the last clause yet. Termina's `ui/DesktopNotifier.java` builds its command lines as the rule asks on every OS: the text is an argument of `notify-send` and of `osascript` (a fixed `on run argv` script), a GVariant string for `gdbus`, and on Windows two environment variables that a fixed PowerShell script (`-EncodedCommand`) reads; the program is killed after five seconds and an exception from running it is logged (`DesktopNotifierTest`). Termina doesn't meet the last clause in full yet: `post` logs nothing when it finds no tool to run (on Linux, neither `notify-send` nor `gdbus`), when it kills the tool at the timeout, or when the tool exits with an error. Editora's `terminal/DesktopNotifier.java` builds the same command lines with the same five seconds, but doesn't meet the rule yet: its `post` catches a failure and drops it, with nothing logged. Both fall back to `gdbus`, and so does nux. nux doesn't meet the rule on Windows yet: its `service/Notifier` has no PowerShell toast, and hands the notification to the tray, which shows AWT's balloon (`tray/AwtTray.java`, `displayMessage`). Nor does it on macOS: it escapes the title and the message into an AppleScript string (`appleScriptEscape`) and passes the whole script to `osascript -e`. It logs no failure either: a tool that is missing or fails hands the notification to the tray, or to nothing. Its timeout is in the measure: it kills the tool after five seconds.
- **TRAY-10** `MUST` Desktop notifications are for what happened while the window was away: the app posts none for an event whose window, or whose tab, is the one in front. Each kind of event has a switch of its own, and where the app notifies of several kinds, most of the switches start off.
  - *Check (code):* the code that posts a notification first tests that the window, or the tab the event belongs to, is not the one in front; each notified kind of event has its own setting; and where the app notifies of several kinds, more of those settings default to off than to on.
  - *Reference:* Editora (`ui/TerminalPanel.java`) and Termina (`ui/TabMarks.java`) relay a program's own OSC 9 and OSC 777 notifications, one event with one switch, and post nothing while that terminal is focused and in view. Torrenta notifies on added, finished, seeding done, error and updated, each with its switch and only *finished* on at first, and a Test button in Settings shows one now (`automation/AutomationSettings.java`); it posts them while its window has focus too (`automation/Automation.java`). nux has one switch for all its power events and a rule per UPS (all, critical or none), not a switch per event, and also posts while its window is in front (`App`).
- **TRAY-11** `SHOULD` A notification is never the only place a state is said: the in-app status already shows it.
  - *Check (by hand):* trigger each notified event with notifications off; the window still shows it.
  - *Reference:* Torrenta's status bar and transfer list.

Settings copy

Key	English
`ui.tray`	A tray (menu-bar) icon with the rates and a menu; on by default
`ui.closeToTray`	Closing the window keeps Torrenta in the tray. *The torrents go on; the tray icon's Show Torrenta brings the window back, its Quit ends the program, as does Quit in the File menu. Without a tray, closing quits.* On by default; nux grays it while the tray is off
(the OS entry)	Start Torrenta when you log in. *Adds Torrenta to your login items: an entry in ~/.config/autostart on Linux, a LaunchAgent on macOS, the Run key on Windows. Turned off in the system's own settings, it shows unticked here. Only for an installed Torrenta, not a run from source.*
`os.autostartInTray`	Start behind the tray icon. *At login Torrenta comes up behind its tray icon instead of as a window, or minimized where the tray icon is turned off. If the icon cannot be shown, the window opens.* `torrenta --minimized` *does the same by hand.*
`notify.<event>`	One switch per event; `notify.finished` on by default
`status.noTray`	No tray icon ({reason}); closing the window quits {app}

In the apps



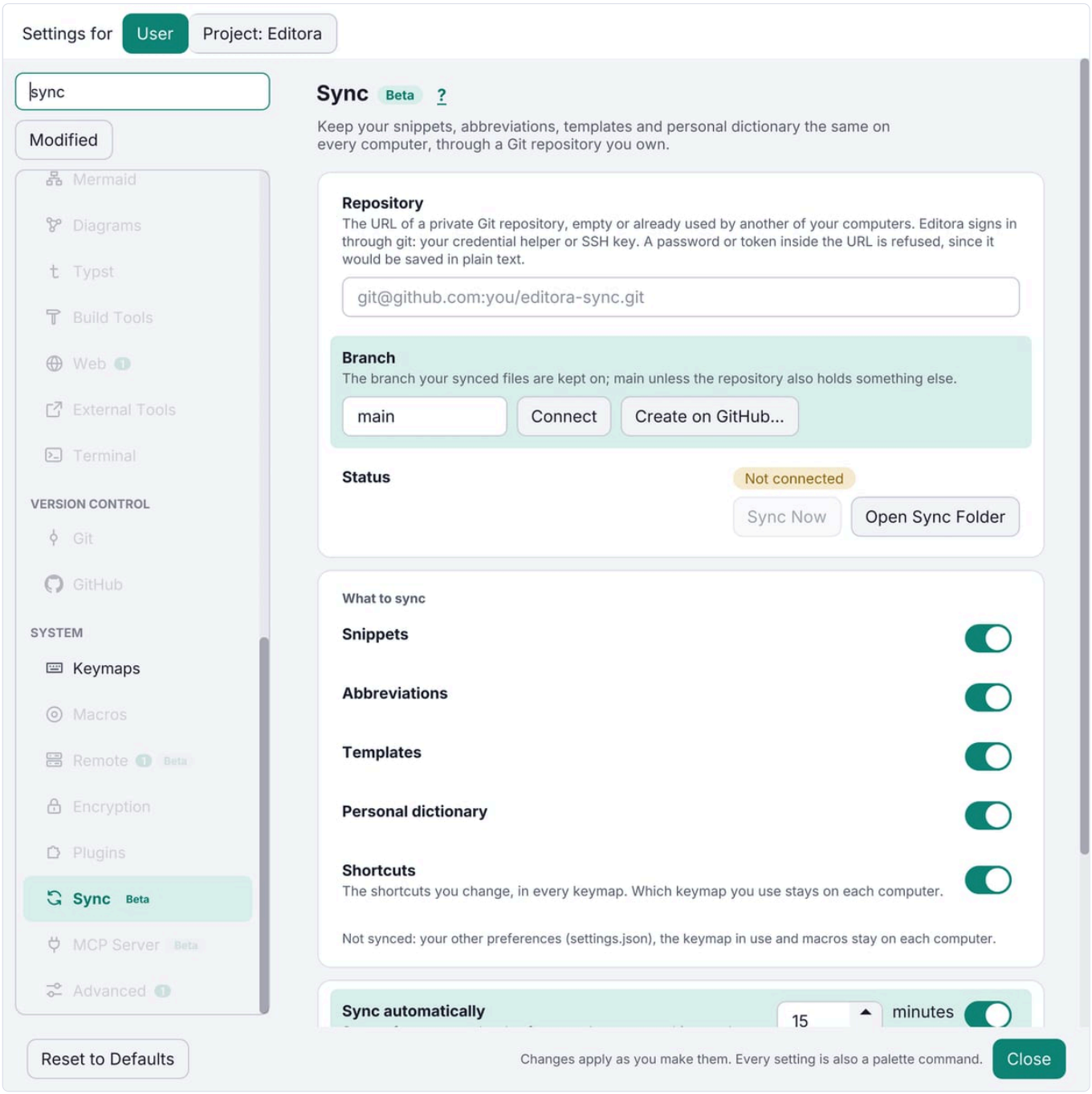
nux lives in the tray: its icon carries the charge number, and this dashboard is one left click away.

Settings

The Settings window is built the same way in every app: pages in a sidebar, a search field that finds any row, and changes that apply as they are made, each also reachable as a palette command. Behind it, the settings live in a config folder in the user's home, in versioned files the app never trusts when it reads them, writes whole, and can export and import; a secret the user gives the app goes to the operating system's keychain, never into those files. An app with user data worth carrying between computers keeps it in step through Settings Sync. Before any of that, ask whether the setting should exist at all (Before adding a setting).

The Settings window

- **SET-01** `MUST` The Settings window opens from a palette command, bound to `Ctrl+,` (`Cmd+,` on macOS) in the CUA keymap. An app with one key table and no CUA keymap, which is a terminal app's own keymap (KBD-09) or the built-in table of an app that ships no keymap files, binds the chord in that table.
  - *Check (code):* a registered command opens Settings, and the CUA keymap binds it to `Ctrl+,` (`C-,` in Editora's `cua.json`) and, on macOS, to `Cmd+,` (`Cmd-,` in `cua.mac.json`). In an app with one key table (a terminal app's `keymap.json` and `keymap.mac.json`, or a table in code), that table binds the two.
  - *Reference:* Editora `view.settings`, bound in `keymaps/cua.json` and `cua.mac.json` (and in the VS Code and Sublime keymaps; IntelliJ's uses `C-M-s`). Termina, a terminal app, binds `settings` to `Shortcut+Comma` in its one keymap (`keymaps/keymap.json` and `keymap.mac.json`). Torrenta and nux, which ship no keymap files, bind the same chord in code (`ui/Chords.java`; nux's `App.java`, `Shortcut+Comma`). Modula doesn't yet: its Settings command has no chord (`ui/RadioPane.java`, `commands`).
- **SET-02** `MUST` The Settings window has a sidebar of pages with icons on the left, the open page on the right, and a search field on top.
  - *Check (by hand):* open Settings: the pages are listed in a sidebar on the left, each with an icon, the open page fills the right, and the search field sits above both.
  - *Reference:* Editora `ui/SettingsWindow.java`, `ui/SettingsSidebarNav.java`, `ui/SettingsIcons.java`.
- **SET-03** `MUST` The Settings window's search finds a row by its title, description, visible text, its card or section heading, its page name and English keywords. Title hits rank above description hits; whole words above word starts. The page with the best match opens, and every non-matching row, heading and card is hidden.
  - *Check (code):* the Settings search matches a query against each row's title, description, visible text, card or section heading, page name and keywords; its score puts a title hit above a description hit and a whole word above a word start; and it opens the page of the best match and hides every row, heading and card that does not match. A unit test that ranks a title hit above a keyword hit and a whole word above part of one is the surest evidence.
  - *Reference:* Editora `ui/SettingsSearch.java`, `SettingsSearchTest` (`aTitleHitOutranksAKeywordHitAndAWholeWordOutranksPartOfOne`). Torrenta's search doesn't rank a whole word above part of one yet: its score looks only at where the text was found, title, description or keywords (`ui/SettingsWindow.java`, `rank`).
- **SET-04** `MUST` Settings apply live, with no Save button. Text fields commit on Enter or focus loss; invalid values are marked and never committed. A setting that cannot apply until a restart says so on its own row.
  - *Check (code):* the Settings window has no Save or Apply button, and each control writes its setting and applies it when it changes; a text field commits its value on Enter and when it loses focus (Editora's `commitOnEnterOrBlur`), and a value that fails validation is marked and not written; and every setting the app reads only at startup (the language, the window's style) names the restart in its own row's text.
  - *Reference:* Editora `SettingsWindow.commitOnEnterOrBlur`; `SettingsSyncPage`'s branch field marks an invalid branch. Rows that wait for a restart say so: Editora's interface language ("Applies after restarting Editora.", `settings.uiLanguage.note`), and Torrenta's tray icon, which on Windows and macOS "applies after restarting Torrenta" (`settings.general.trayIcon.desc`, in `ui/SettingsWindow.java`). nux doesn't yet: its Settings window writes nothing until its Save button is pressed, and has a Cancel beside it (`ui/SettingsWindow.java`).
- **SET-05** `SHOULD` A setting changed in one window reaches every other open window of the app.
  - *Check (code):* every window reads its settings from one model the windows share, and a change to that model is announced to each open window, which applies it. An FX test that changes a setting in one window and finds it applied in the other is the surest evidence.
  - *Reference:* Editora: every window holds the same `config/SharedConfig.java`, which announces each change (`onSettingsChanged`); `CrossWindowSettingsFxTest`.



Typing sync in Settings: pages without a match gray out and the best page opens.

Before adding a setting

A setting is a question the user has to answer. When the question has one right answer, choose it, make it the behavior, and say why in the code and the docs; do not ship a knob for it. Modula's test: *would a car radio have it?* A preset button, yes; a "decimation stages" spinner, no. So Modula has no gain, bandwidth or squelch control: each has one right value for a listener, and the app supplies it. Add a setting when people really differ (a keymap, a theme, a language), not when the developer could not decide.

Adding a setting touches five places

1. **Model** field, default and accessors, with a schema bump and migration.
2. **Control**, built once, writing the field and applying it.
3. **Row** on its page with search keywords.
4. **Load** line that sets the control from the model, safe to run at any time.
5. **Palette command** that flips or prompts the same field, plus strings in every catalog.

- `SET-06` `MUST` Every setting the Settings window shows has a palette command that flips the field or prompts for its value. A key that SET-16 keeps out of the UI has no row, and takes no command.
  - *Check (code):* list the settings the Settings window's rows change and the commands the palette lists; each setting has a command that flips it or prompts for its value. A row that only shows something, or runs an action, has no setting to give a command to. A test that fails on a row with no command is the surest evidence.

- *Reference:* Termina `ui/SettingCommands.java`, a command for each setting or the reason a row needs none (`SettingCommandsTest`, `everySettingsRowHasACommandOrSaysWhyNot`), and Mirada `ui/SettingsPages.java`, where every row names its command (`SettingsPagesTest`, `everyPageHasRowsAndEveryRowAPaletteCommand`). Editora has its `view.toggle…` and picker commands (`ui/WindowCommandRegistrar.java`) but not one for every setting yet: `terminalNotifications`, for one, is only a Settings row. Modula's palette changes one of its settings, the daylight theme (`ui/RadioPane.java`, `commands`), and nux's three: the tray metric, the notification rule and start at login (`App.java`, `registerCommands`).

Config folder and schema

- **SET-07** `MUST` Config lives in `~/.<app>`, a folder in the user's home on Windows, macOS and Linux alike, unless `<APP>_CONFIG_DIR` or a command-line flag (CLI-02) names another folder. A flag wins over the variable, since what is typed on the command line is the more specific instruction: `--config-dir` first, then `--dev`, then the variable.
  - *Check (code):* one resolver picks the folder from `--config-dir`, `--dev`, the environment variable and the default, with `--config-dir` first, `--dev` next, the variable after it and the default last, whichever of the two flags is typed first; and the default is `<user.home>/.<app>` on Windows, macOS and Linux alike: the resolver doesn't branch on the OS, or a test resolves the default as each OS and finds the same folder.
  - *Reference:* the app template's `config/ConfigDir.java` (`resolve`: `--config-dir`, `--dev`, the variable, then `~/.exampleapp`, with no branch on the OS; `ConfigDirTest`, `configDirThenDevThenEnvironmentThenPropertyThenTheDefault`). No app has all of it yet. Editora (`App` and `ConfigManager.configDirFor`, `ConfigDirTest`), and Termina (`config/Settings.java`, `defaultFile`) put `--config-dir` first and have one default for every OS, but the variable wins over `--dev`, which only changes the default. In Mirada either flag wins over the variable, but the two flags set one value and the one typed last wins, so `--config-dir X --dev` gives `~/.mirada-dev` (`Launch.java`, `parse`); and its default depends on the OS (`settings/Settings.java`, `defaultFile` and `folder`): `~/Library/Application Support/Mirada` on macOS, `%APPDATA%\Mirada` on Windows, `~/.mirada` elsewhere. Torrenta has `--config-dir`, the variable and the default in that order and no `--dev` (`store/ConfigDir.java`). Modula and nux read a fixed `~/.modula` and `~/.nux`, with no flag and no variable.
- **SET-08** `SHOULD` Preferences live in one file and session state (where the window was, what was open) in another, and every other store has a file of its own. Editora's are `settings.json` and `workspace-state.json`.
  - *Check (code):* preferences and session state are separate files: the file that holds what the user chose in Settings holds no window geometry and no list of open items.
  - *Reference:* Editora `config/ConfigManager.java` (`SETTINGS_FILE_NAME`, `WORKSPACE_FILE_NAME`), `config/WorkspaceState.java`; Termina `config/Settings.java`, which writes the same two names and sorts each key into one of them (`isStateKey`). Torrenta and Mirada keep the window's bounds in their preferences file (`ui.window.bounds` and `window.bounds` in `settings.properties`). The app template meets it: the window and the open documents go to `session.json`, beside `settings.json` and never in it (`SESSION_FILE` in `config/ConfigStore.java`; its `config/Migrations.java` moves them out of an older `settings.json`).
- **SET-09** `MUST` Every config file the app writes and reads back carries a schema version, and each version has a `v → v+1` migration step. A purely additive field uses an identity step. Which files are config files is in SET-22; a plain list with one item to a line, such as a word list, has no shape to version and is exempt.
  - *Check (code):* the store of every config file (SET-22 says which files those are) declares a schema version (`SCHEMA_VERSION` in Editora), writes it into the file and reads it back, and the migrations hold one step for every version below the current one, an identity step where the version only added a field.
  - *Reference:* Editora `config/migration/ConfigMigrations.java`, `ConfigSchema.java` (one entry for each store), `ConfigMigrationsTest`; its `dictionary.txt`, one word to a line, is the kind of list the rule exempts. Termina (`config/ConfigJson.java`, for `settings.json` and `workspace-state.json`) and Mirada (`settings/Migrations.java`, for `settings.properties` and `bookmarks.properties`) do it too, and so does the app template for `settings.json` (`config/Migrations.java`) and for `session.json` (`SESSION_SCHEMA` in `config/ConfigStore.java`), Settings Sync's `base.json` and `keybindings.json` (`sync/SyncData.java`, `SCHEMA_VERSION` and `migrate`) and the account it pins (`account.json`, `sync/AccountPin.java`). Nor does Termina for the color schemes it imports: each is a file Termina writes whole, in a format of its own, and reads back at start (`themes/<name>.properties`, `theme/ThemeFile.java`), so a config file by SET-22, and it carries no version. Torrenta's files (`store/Settings.java` and the other stores beside it) and Modula's (`config/ConfigStore.java`: `settings.properties`, `presets.txt`, `schedules.txt`, each line a record with fields) carry no version, and nux writes `schema=1` into its file and never reads it (`config/ConfigStore.java`, `format` and `parse`).
- **SET-10** `MUST` A config file whose schema version (SET-09) is higher than the running build's is kept as a backup beside it (Editora names it `<name>.v<n>.bak`) and defaults load.
  - *Why:* an older build never clobbers a newer config.
  - *Check (code):* the load of each config file compares the file's schema version with the build's, and for a higher one keeps the file as a backup beside it and loads the defaults. A test that loads such a file and finds the backup beside it and the defaults loaded is the surest evidence.
  - *Reference:* Editora `ConfigVersioningTest` (`tooNewSettingsAreBackedUpAndDefaultsLoaded`). Termina (`config/Settings.java`), Mirada (`settings/ConfigFile.java`, `newerBackupOf`) and the app template (`config/ConfigStore.java`) do the same, under the same name. Torrenta, Modula and nux read no version (SET-09), so they cannot tell a newer file, and nor can Termina for an imported color scheme, which carries none.
- **SET-11** `MUST` Never trust a config file. Clamp every value to its valid range when it is **read**, not only when it is written; an unknown value falls back to its default; a file that cannot be decoded is kept as a backup (Editora names it `<name>.corrupt.bak`) and defaults load. A bad file never stops the app from starting.
  - *Why:* config is plain JSON (or properties) and people edit it by hand. A hand-edited two-million-line scrollback must not exhaust memory at the next launch (Termina).
  - *Check (code):* the load clamps each value to its range as it reads it and puts the default in place of an unknown one; a file the parser cannot decode is kept as a backup and the defaults load; and the load catches what the parser throws besides an `IOException` (a malformed `\uXXXX` escape in a `.properties` file throws an `IllegalArgumentException`), so no file stops the start. A test that loads out-of-range numbers and an unparseable file of that kind is the surest evidence.
  - *Reference:* Editora `config/migration/ConfigMigrations.java` (`keepCorrupt`), `ConfigLoadProblemsTest` (`outOfRangeNumbersAreClampedOnLoad`, `anUnparseableFileIsReportedWithItsBackup`); Termina `config/Settings.java`, `SettingsTest` (`aMalformedEscapeNoLongerStopsTheAppStarting`); Mirada `settings/ConfigFile.java`, which moves a file that will not decode to `<name>.corrupt.bak` (`ConfigFileTest`, `aFileThatWillNotDecodeIsKeptAsABackup`). Torrenta, Modula and nux keep no backup, and in Torrenta and nux a malformed escape stops the app from starting.
- **SET-22** `MUST` A config file is written whole or not at all: the app writes a temporary file beside it and moves that into place, so a crash mid-write never leaves a half-written file. A config file is any file the app writes as a whole and reads back as its state: the settings, the session state, a store of bookmarks or presets, what a sync engine keeps beside its clone. A file the app only appends to (a log, a ledger, a word list) is not one, and neither is a file in the working tree of a sync repository's clone, which Git holds a committed copy of and the next sync writes again. Nor is a file the app writes for one run and deletes at exit, such as one that tells another process where this run listens.
  - *Why:* without it, a crash or a full disk in the middle of a save leaves a file that will not decode, and the corrupt-file path of SET-11, a backup and then defaults, is what the person gets for it.
  - *Check (code):* every write of a config file as a whole (a store's save, an import, a sync engine's own state outside its clone) goes to a temporary file in the same folder and ends in a move onto the target (`Files.move`). `ATOMIC_MOVE` is the surest form; a move that only replaces the target passes too, since the statement asks for the temporary file and the move. An append to a log, a ledger or a list is not such a write, and neither is a write into a sync clone's working tree, nor the write of a file the app deletes when it exits.

- *Reference:* Editora `config/ConfigWriter.java` (`writeAtomicOrThrow`: a temporary file of its own beside the target, forced to disk, then moved; `ConfigWriterTest`, `atomicReplacementFailuresKeepThePreviousConfiguration`), which its settings and stores go through. Editora's `mcp-endpoint.json` is not a config file: its MCP server writes it for the run when it starts and deletes it when it stops, as the last window closes (`mcp/McpServer.java`, `writeEndpointFile` and `stop`), and one that a crash left behind is deleted at the next launch (`reapStaleEndpoint`). Every other app's settings store does it too: Torrenta `store/Settings.java` (`save`), Termina `config/Settings.java` (`writeOne`, `moveIntoPlace`), Mirada `settings/Settings.java` (`write`), Modula `config/ConfigStore.java` (`writeAtomically`), nux `config/ConfigStore.java` (`save`) and the app template's `config/ConfigStore.java` (`writePending`, for `settings.json` and `session.json`). The template's other whole-file writes go the same way, through `config/ConfigFiles.java` (`write`): an import (`config/ConfigImport.java`, `apply`), Settings Sync's `base.json`, the merge base it keeps beside the clone (`sync/SyncData.java`, `write`), and the account it pins there (`sync/AccountPin.java`). Termina's `sync/SyncEngine.java` and Torrenta's `sync/SyncEngine.java` write in place only inside the clone, and keep their merge base as a Git ref (`refs/termina/base`, `refs/torrenta/base`).

- **SET-12** `MUST` Export configuration writes a timestamped `.zip` of the config folder; Import restores from one.
  - *Check (code):* the export command writes a `.zip` that carries the time of the export in its name and the config folder's files among its entries, and the import command reads such a zip and restores its files. A test of each is the surest evidence.
  - *Reference:* Editora `config/ConfigExporter.java`, `config/ConfigImport.java`, `ConfigExporterTest` (`zipNameDropsLeadingDotAndEmbedsVersionUserTime`, `exportZipsAllFilesWithRelativeEntries`), `ConfigImportTest` (`theTickedPartsReplaceTheirsHereAndNothingElseChanges`: Editora's import restores the parts the user ticks).

Secrets stay out of files

- **SET-13** `MUST` An app that stores a secret the user gave it (a password, a token, a key) puts none in a config file. Each goes to the operating system's keychain: macOS Keychain, Windows Credential Manager, the Secret Service on Linux. Calls may block on a prompt, so they run off the FX thread. Where no keychain answers, as in a Linux session with no Secret Service, the secret is kept in memory for the session only and its row in Settings says so: it is never written to a file instead. A run with no window says so on standard error. A key or token the app generates itself is not such a secret: it may live in a file only the owner can read, outside the settings files (Torrenta's `keys/<name>.key`). Whether a credential is kept from one session to the next is the app's to decide: one it keeps is this rule's, and one it only holds for the session, and asks for again in the next, is SET-14's.
  - *Check (code):* the rule covers an app that keeps a password, an API key or a token the user gave it from one session to the next; an app that keeps none is outside it. In an app it covers, no such secret is a field of the settings model or is written to any other file; reads and writes go through the secret store, and every caller reaches it from a background thread, never from a handler on the FX thread; the store's fallback, for when no keychain answers, holds the secret in memory and writes no file, and the secret's row in Settings says when it is kept for the session only, as a line on standard error does in a run with no window. A key or token the app generates is not judged as such a secret where it is in a file of its own, outside the settings files, that the app makes readable by its owner alone (owner-only permissions, where the file system has them).
  - *Reference:* Editora `secrets/SecretStore.java`, `config/ApiKeyVault.java`. Editora doesn't yet: `Settings` keeps the AI keys as a fallback for when no keychain answers, and saving a key from Settings writes the keychain on the FX thread. Its Settings row does say which of the two holds the key. The bearer token Editora's MCP server generates is in a file of its own for the run, `mcp-endpoint.json`, made owner-only where the file system has POSIX permissions and deleted when the server stops (`mcp/McpServer.java`, `writeEndpointFile`), which the rule allows. Torrenta uses no keychain: its proxy password and the password of its Remote access certificate file (`proxy.password` and `webapi.httpsPassword` in `settings.properties`, `store/Settings.java`), its Torznab keys (`search-providers.json`) and its remote-profile passwords (`profiles.json`) are in plain text in its config folder; of its Remote access password it keeps only a hash. The keys Torrenta generates for the updatable torrents it publishes are where the rule allows: a file each, outside the settings files and owner-only where the file system has permissions (`keys/<name>.key`, `store/KeyStore.java`). Termina, Mirada and Modula store no secret, so the rule does not cover them: Termina's sync signs in through the user's own Git and `gh` (`sync/SyncAccount.java`), and the other two ask for none. nux's one credential is for the session only, and so are the SSH passwords and passphrases Editora asks for (SET-14).
- **SET-14** `MUST` An app that asks for a credential it does not keep from one session to the next holds it in memory for the session only and never writes it. Whether to keep a credential is the app's to decide: one it keeps is SET-13's, in the keychain and never in a config file, and this rule covers only one it does not keep. Work that needs such a credential waits until the user has signed in this session; do not "fix" that by writing the password to a file.
  - *Check (code):* for each credential the app asks the user for (a login, a password, a passphrase), find whether the app keeps it for the next session. One it keeps is SET-13's and is not judged here. One it does not keep is a field of a runtime object, absent from every config class, and no code writes it to a file. An app that asks for no credential, or keeps every one it asks for, is outside the rule.
  - *Reference:* nux's UPS login: the user name and password it asks for are two fields of a runtime object (`app/CommandRunner.java`), in no config class. Editora asks for an SSH password or a key's passphrase while a login runs (`vfs/LoginPrompt.java`) and stores neither: a saved connection holds the host, the user and the key's path (`config/ConnectionStore.java`). What an app keeps is SET-13's alone: Editora's AI keys, and Torrenta's proxy, Torznab and remote-profile secrets. Termina, Mirada and Modula ask for no credential.
- **SET-15** `MUST` An app with Settings Sync refuses a repository URL with a password or token in it, and says what to use instead: a Git credential helper or an SSH URL (Settings Sync). A problem report's redaction is DIAG-11.
  - *Check (by hand):* paste `https://user:token@host/repo.git` as the sync repository: the page refuses it, with a message that says what to use instead.
  - *Reference:* Editora `ui/SettingsSyncPage.java` (`settings.sync.repo.credentials`), which asks `sync/SyncEngine.java` (`hasCredentials`) as the URL is typed, when the field commits and again at Connect.
- **SET-16** `MUST` An option that weakens security has no UI: it exists only as a key in the config file, so nobody turns it on by clicking. An option weakens security when it switches off a check or a question the app otherwise makes to protect the user; a feature, such as remote access or a proxy, is not such an option.
  - *Check (code):* no Settings row or command writes the key of an option that switches off a check or a question the app otherwise makes to protect the user, whichever of its values is the weaker one: on for nux's `tlsTrustAll`, off for a check the app makes by default.
  - *Reference:* nux's trust-all TLS (`tlsTrustAll` in `config/Config.java`), and Termina's paste confirmation, which only its config file turns off (`SecurityOptionsTest`, `noControlOrCommandWritesAWeakeningOption`). Editora doesn't yet: *Require registry signature* for plugins is a Settings checkbox and the `plugins.toggleRequireSignature` command.

Settings Sync (Git / GitHub)

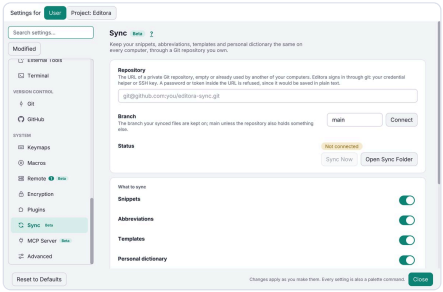
Applies to: an app with Settings Sync.

Which apps should offer Settings Sync is SYNC-01: one with user data worth carrying between computers. It syncs into a Git repository the user owns (SYNC-02). The rules of this section are how sync behaves in an app that has it; an app without it owes them nothing.

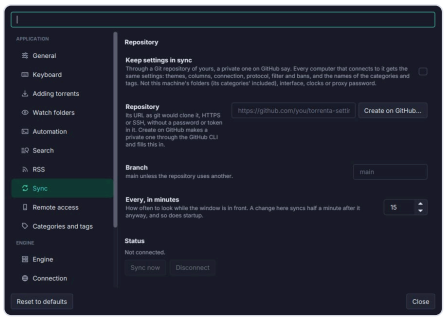
- **SET-18** `SHOULD` Settings Sync is set up by pasting a repository URL, or with **Create on GitHub** through the `gh` CLI. The account that can read the repository is pinned per clone, so machines with two GitHub accounts work.

- *Check (code):* the Sync page takes a repository URL in a field and offers a Create on GitHub action that runs `gh`; and the sync code finds which of the signed-in GitHub accounts can read the repository, records it with the clone, and uses that account for the clone from then on. A test that remembers the first account that can read the repository and uses it without asking again is the surest evidence.
- *Reference:* Editora `sync/SyncAccount.java`, `SyncAccountTest`, `ui/SettingsSyncPage.java`. The app template's `sync/AccountPin.java` finds the first signed-in `gh` account whose token can read the repository, records it beside the clone and passes it to every fetch and push from then on (`AccountPinTest`, `theFirstAccountThatCanReadIsPinnedAndKept`).

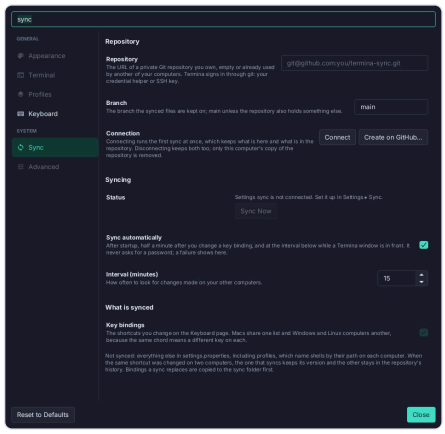
- **SET-19** `SHOULD` Settings Sync runs automatically (on change and at start) and on demand. Machine-local keys (window size, paths) never sync.
  - *Check (code):* a sync starts by itself after startup and after a local change, and a command starts one on demand (`sync.now` in Editora); and no machine-local key or file (window size, a path) is among those the sync sends to the repository.
  - *Reference:* Editora `ui/SettingsSync.java` (after startup, after a local change, on an interval, and `sync.now`), `sync/SyncCategory.java`; Torrenta `sync/SyncKeys.java`.
- **SET-20** `SHOULD` When Settings Sync finds one entry changed on both sides, *this machine wins*, and a backup is taken before any sync that replaces local files.
  - *Check (code):* the merge keeps this machine's value for an entry both sides changed, and the sync copies the local files to a backup before it replaces them. A merge test where both sides changed one entry is the surest evidence.
  - *Reference:* Editora `sync/SyncMerge.java`, `SyncMergeTest`; the app template's `sync/SyncMerge.java` and `ui/SyncController.java` (`backUp`, which copies `settings.json` to `backups/` before a merge replaces the local bindings; `SyncControllerFxTest`, `aSyncThatReplacesLocalSettingsBacksThemUpFirst`).
- **SET-21** `SHOULD` Settings Sync shows its status as a pill in Settings and as echo messages; its failures go to the Message Log.
  - *Check (by hand):* point Sync at a repository that does not exist: the pill in Settings shows the failure, the echo area says it, and the Message Log keeps it.
  - *Reference:* Editora `ui/SettingsSyncPage.java`, `SyncStatus`.



Editora's Sync page, not connected: the repository and branch fields, Connect, the status pill and what to sync.



Torrenta's Sync page.



Termina's Sync page.

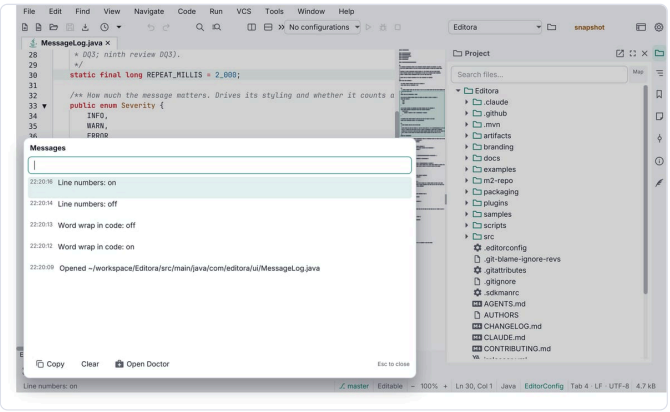
Diagnostics and support

When something goes wrong, the user and the developer need the same evidence, and every app keeps it the same way. The Message Log holds what the status bar said this session; the Debug Log holds every log record and uncaught exception, and keeps the previous session's file so a crash leaves its trace; the update check says when a newer version is out; and Report a Problem turns the evidence into a redacted GitHub issue the user reads and edits before it is sent. Two SHOULDs complete it: a Doctor page that checks the environment, and, in an app with user documents, unsaved work offered back after a crash.

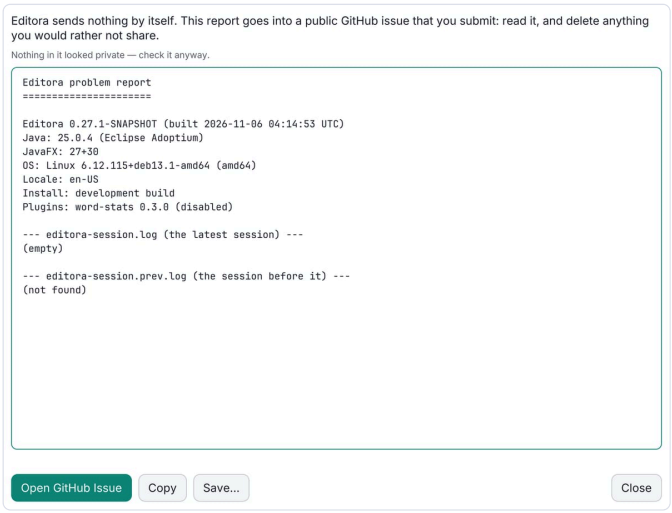
- **DIAG-01** `MUST` The **Message Log** is a session-only, bounded history of every echo-area message with `HH:mm:ss` times, newest first. Esc closes it; Clear empties it. What opens it, a click on the echo area, is WIN-01's to require; the Message Log template has the rest.
  - *Check (code):* every message the echo area shows is added, with its time, to one log held in memory and never written to disk; the log drops its oldest entry past a fixed count and lists the rest newest first with `HH:mm:ss` times; and its window closes on Esc and has a Clear action that empties it. A test that records messages newest first, evicts the oldest past the cap, and clears is the surest evidence.
  - *Reference:* Editora `ui/MessageLog.java` (`MessageLogTest`), `ui/MessageLogPopup.java` .
- **DIAG-02** `MUST` The **Debug Log** captures all `java.util.logging` records and uncaught exceptions in memory (bounded) **and** mirrors them to a session file in the config folder, keeping the previous session's file so a crash is still visible next launch. Editora names the two `<app>-session.log` and `<app>-session.prev.log` .
  - *Check (code):* a handler on the root `java.util.logging` logger and the uncaught exception handler feed one bounded buffer in memory and a session file in the config folder, and at start the last session's file is kept under a second name before the new one is begun. Tests that bound the buffer and find the session file written and the previous one kept are the surest evidence.
  - *Reference:* Editora `ui/DebugLog.java` (`DebugLogTest` , `DebugLogSessionFileTest`).
- **DIAG-03** `MUST` The Debug Log opens in a modeless window with Refresh, Copy, Clear, Export and Report a Problem. The window's find bar is FIND-01.
  - *Why:* it is the only way to see logs in a packaged build.
  - *Check (code):* the Debug Log's window is a stage with no modality, so the main window keeps working while it is open, and it carries the five actions: Refresh, Copy, Clear, Export and Report a Problem.
  - *Reference:* Editora `ui/DebugLogWindow.java` . Termina's, Mirada's and the app template's windows have the five actions too; Torrenta's has no Export (`ui/DebugLogWindow.java`). Modula and nux have no Debug Log (DIAG-02).
- **DIAG-04** `MUST` An **uncaught exception handler** is installed at the start of `main` , before the toolkit starts, and routes to the Debug Log and an echo message. Never a stack trace dialog.
  - *Check (code):* `main` calls the install method before any argument handling or `launch` , and the handler posts an error to the echo area, and opens no dialog.
  - *Reference:* Editora `DebugLog.install()` for the install and the log. The echo message is posted by Termina (`ui/WindowManager.java` , through `DebugLog.onUncaught`), the app template (`ui/Startup.java`) and Mirada (`App.java` , `uncaught`). Editora's handler and Torrenta's (`ui/DebugLog.java`) log the exception but do not yet post one.
- **DIAG-05** `MUST` The **update check** reads GitHub `/releases/latest` . The automatic check runs once per app session (not per window), at most once a day, and starts at least three seconds after the first frame (PERF-15); a check the user asks for (DIAG-10) runs when asked. It never downloads or installs anything by itself.
  - *Check (code):* the code that starts the automatic check waits for the first frame and then a delay of three seconds or more before it asks GitHub's `/releases/latest` for the app's repository (Editora's `AfterFirstPaint` , then a pause of five seconds); it asks once for the whole process, however many windows open, and not at all when the last check, whose time the app keeps from one session to the next, was less than a day ago, or when the setting is off (DIAG-10); and what it does with the answer is show it: no code downloads or installs a release. A test that asserts the wait, the one check a session and the two skips is the surest evidence.
  - *Reference:* Editora `update/UpdateCheck.java` , `ui/UpdateCheckScheduler.java` (`UpdateCheckSchedulerTest` , `waitsForTheFirstPaintThenTheDelayThenStampsAndChecks` and `onlyOnceASessionAndNotWhenOffOrNotDue`), `ui/UpdateCoordinator.java` ; Editora waits five seconds (`UpdateCheckScheduler.DELAY`). Termina (`App.java` , three seconds after its first frame), Mirada (`update/UpdateCoordinator.java` , `scheduleStartupCheck` , five seconds) and the app template (`ui/UpdateCoordinator.java` , started from `ui/Startup.java` after the first paint, five seconds) wait the same way. Three apps don't wait yet: Torrenta starts its check as soon as its engine is ready (`ui/MainWindow.java` , `maybeCheckForUpdates`), and Modula (`ModulaApp.java` , `checkForUpdates`) and nux (`App.java` , `maybeCheckForUpdates`) start theirs inside `start()` .
- **DIAG-09** `MUST` When the update check (DIAG-05) finds a newer version, the status bar says so, naming the version and linking its release. About's line for the last check is ABOUT-02.
  - *Check (code):* when a check finds a newer version, the app shows a notice in the status bar that names that version, and the notice, or a link beside it in the bar, opens that release's page (Editora's `UpdateCoordinator` shows a status-bar segment whose click opens the release).
  - *Reference:* Editora `ui/UpdateCoordinator.java` . Mirada (`ui/StatusBar.java`) and the app template (`ui/StatusBar.java`) show a link in the bar, and Torrenta a badge there that opens the release (`ui/MainWindow.java` , `updateBadge`). Termina and Modula don't link the release yet: their status message names the version and nothing in the bar opens it (Termina's `status.updateAvailable` , posted from `ui/WindowManager.java` ; Modula's `ui/RadioPane.java` , `setAvailableUpdate`). nux doesn't meet it either: it has no status bar, and shows a button in its toolbar that opens the release (`App.java` , `refreshUpdateNotice`).
- **DIAG-10** `MUST` **Check for Updates** runs the update check (DIAG-05) by hand, as a command in the palette and in the Help menu, where the app has a Help menu (the Help menu of a menu bar, or a Help submenu of the app's main menu, which WIN-14 defines), and Settings has an off switch for the automatic check.
  - *Check (code):* the command (`help.checkForUpdates` in Editora) is registered and, in an app with a Help menu, listed there, and the scheduler skips the automatic check while the setting is off.
  - *Reference:* Editora `ui/MenuBarModel.java` (`help.checkForUpdates`), `ui/SettingsWindow.java` (`settings.checkForUpdates`), `ui/UpdateCheckScheduler.java` . Mirada lists the command in the Help submenu of its main menu (`ui/TitleBar.java` , `help`) and in the Help menu of its macOS menu bar. nux, which has no menu bar, registers the command for its palette (`app.checkUpdates` in `App.java`).
- **DIAG-06** `MUST` **Report a Problem** is a command, listed in the Help menu where the app has one, that builds a report: the app's version, the OS, the runtime, a summary of the settings and the recent Debug Log. Its redaction is DIAG-11, its window DIAG-12 and the limit on the issue link's length DIAG-13. The same report from the command line, for when the app cannot start, is CLI-05.
  - *Check (code):* the command (`help.reportProblem` in Editora) is registered and, in an app with a Help menu, listed there, and the text it builds carries the app's version, the OS, the Java and JavaFX versions, a summary of the settings and the end of the Debug Log.
  - *Reference:* Termina `report/ProblemReport.java` (`ProblemReportTest` , `itCarriesTheEnvironmentTheUsefulSettingsAndTheEndOfTheLog`), the app template's `report/ProblemReport.java` and Mirada's `report/ProblemReport.java` . Editora's (`report/ProblemReport.java` , `help.reportProblem` in `ui/MenuBarModel.java`) and Torrenta's (`report/ProblemReport.java`) carry the system details and the last two session logs, and no settings summary yet. Modula and nux have no problem report.
- **DIAG-11** `MUST` An app's problem report (DIAG-06) is **redacted** before it is shown: home paths, user names, e-mail addresses, credentials and tokens are replaced.

- *Check (code):* the text the Report a Problem window shows has been through the redactor, and the redactor replaces home paths, user names, e-mail addresses, credentials in URLs and tokens. Redaction tests that cover each of the five are the surest evidence.
- *Reference:* Editora `report/Redactor.java` (`RedactorTest`), applied in `ProblemReport.assemble` before the window gets the text. The other four apps with a report do it too: Termina (`RedactorTest`), Torrenta (`RedactorTest`), Mirada (`RedactorTest`) and the app template (`ReportTest` , `redactsPrivateDetails`).

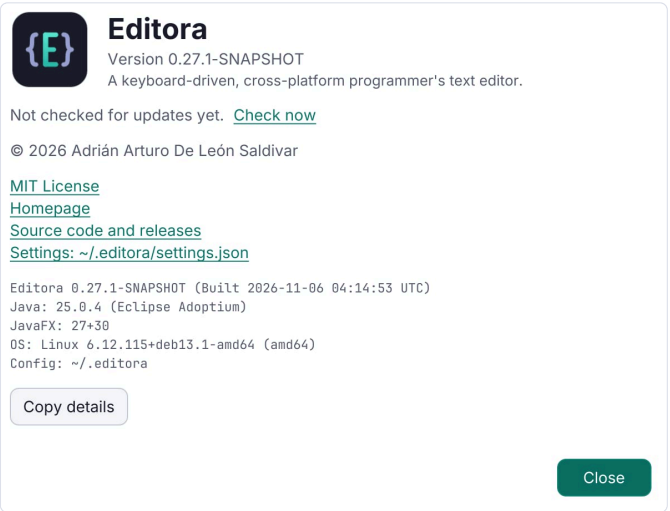
- **`DIAG-12`** `MUST` The Report a Problem window shows the report in a field the user can edit. Its submit action copies the report, as edited, to the clipboard and opens a GitHub issue prefilled from the repository's `bug_report.yml` form. Nothing is uploaded: the user submits the issue.
  - *Check (code):* the window's report field is editable; its submit action puts the field's text on the clipboard and opens a new-issue link that names `bug_report.yml` as its template and carries the report in the form's fields; and the report's code opens that link in the browser and makes no network request of its own.
  - *Reference:* Editora `ui/ReportProblemWindow.java` (`ReportProblemWindowFxTest` , `theEditedReportIsCopiedAndFilledIntoTheIssue`), `ProblemReport.issueUrl` . Termina's `ui/ReportProblemWindow.java` , Torrenta's `ui/ReportProblemWindow.java` , Mirada's `ui/ReportProblemWindow.java` and the app template's `ui/ReportProblemWindow.java` do the same. The five differ in one thing the rule does not ask. Only Termina sends everything as the user left it: its environment block (version, runtime, OS) is a second editable field. The other four fill the form's environment field from the block as the app built it, whatever the user edits: Editora and Torrenta show the block at the top of the editable report, Mirada shows it as a label, and the app template does not show it.
- **`DIAG-13`** `MUST` When the issue link that Report a Problem prefills (DIAG-12) would be longer than 7,500 characters, the report stays out of the link: the form's report field asks the user to paste the report from the clipboard, where DIAG-12 has put it.
  - *Check (code):* the code that builds the issue link puts the report in the form's report field while the link stays within 7,500 characters, and past that puts the paste hint in that field instead. A test that builds the link for a short report and for a long one is the surest evidence.
  - *Reference:* Editora `ProblemReport.issueUrl` (`MAX_URL_LENGTH` , 7500; `ProblemReportTest` , `aShortReportGoesIntoTheIssueLinkItself` and `aLongReportLeavesThePasteHintInstead`). Torrenta (`ProblemReportTest` , `aShortReportGoesInTheLinkAndALongOneLeavesTheHint`), Mirada (`ProblemReportTest` , `aLongReportAsksForAPaste`) and the app template (`ReportTest` , `issueLinkFillsTheFormOrFallsBackToTheClipboard`) do the same at the same limit. Termina has the limit (`MAX_LINK_LENGTH` ; `ProblemReportTest` , `aLongReportFallsBackToTheClipboard`) but not the hint yet: past it, its link leaves the form's report field empty, and only its own window says to paste (`report/ProblemReport.java` , `link` ; `report.paste`).
- **`DIAG-07`** `SHOULD` A **Doctor** page checks the environment (external tools, versions, permissions, paths), with a status per row and a fix or link for each failure.
  - *Check (by hand):* open the Doctor with a tool missing and find its row failed, with what to install.
  - *Reference:* the app template's `sync/Doctor.java` and `ui/DoctorWindow.java` , opened by `help.doctor` and from Settings: a row for each tool it uses, `git` failed with what to install and a link when it is missing (`DoctorTest` , `aMissingToolFailsItsRowWithWhatToInstall`). Editora `doctor/*` , `ui/DoctorPane.java` .
- **`DIAG-08`** `SHOULD` An app with user documents snapshots unsaved work off the FX thread and offers it back after a crash.
  - *Check (code):* unsaved documents are written to snapshot files from a background thread while the user works, and the first window opened after a crash finds the abandoned session's snapshots and offers them back. A test that finds an abandoned session's snapshots and offers them to the first window only is the surest evidence.
  - *Reference:* Editora `recovery/RecoveryService.java` (`RecoveryServiceTest` , `CrashRecoveryFxTest`).



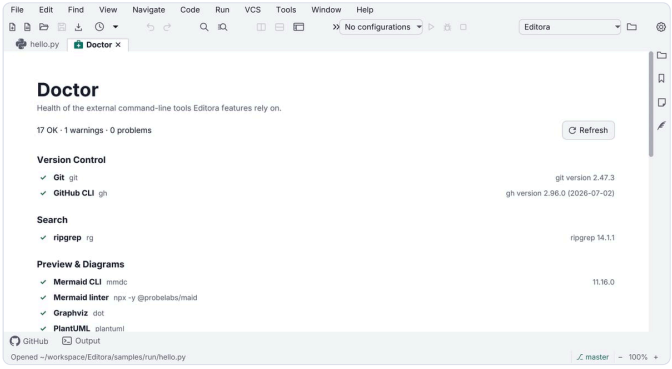
Clicking the status message opens the Message Log: this session's messages, newest first.



Report a Problem: a redacted, editable report that opens as a prefilled GitHub issue.



About: version, runtime, license, links and the update status.



Doctor: each external tool with its version, or what to install.

Secrets never reach a log, a report or a config file: see Secrets stay out of files.

Command line and launch

Every app takes the same command-line flags, and they mean the same thing in each: print the help or the version, use another config folder, start without the last session, write a problem report, open the files named. They work the same on every OS, Windows' console included, and refuse an option they don't know. A second launch doesn't start a second app: it hands its arguments to the one already running for that config folder and exits. Scripts and habits depend on these names, so a flag a rule names is required as written.

Flags

- **CLI-01** `MUST` `--help` and `--version` print before the GUI starts, in plain English, not translated.
  - *Check (code):* `main` handles both and exits before `launch`, printing fixed English text rather than catalog messages.
  - *Reference:* Editora `App.main`, `App.helpText`; `AppArgsTest`.
- **CLI-02** `MUST` `--config-dir <dir>` uses another config folder, and `--dev` uses `~/.<app>-dev`, a folder in the user's home on Windows, macOS and Linux alike.
  - *Check (code):* the config folder is resolved from both flags before anything reads `~/.<app>`, and what `--dev` resolves to is `<user.home>/.<app>-dev` on Windows, macOS and Linux alike, with no branch on the OS.
  - *Reference:* Editora `App.main` (`ConfigManager.configDirFor`, `App.devFlag`).
- **CLI-03** `SHOULD` `--new-instance` starts a second process instead of handing off to the running one.
  - *Check (code):* the flag skips the single-instance hand-off.
  - *Reference:* Editora `App.newInstanceFlag`.
- **CLI-04** `SHOULD` An app that reopens what was open in its last session (documents, tabs, folders) takes `--no-session`, which starts it without them.
  - *Check (by hand):* launch with `--no-session` and find only the items given on the command line.
  - *Reference:* Editora `App.helpText`. Mirada reopens nothing from its last session, so the rule does not cover it; it takes the flag and does nothing with it (`Launch.java`).
- **CLI-05** `MUST` `--report` writes a redacted problem report to a file in the current folder, prints the file's path and opens the GitHub issue, without starting the GUI. Where the current folder can't be written, the file goes to the temp folder.
  - *Why:* the person who ran the command is looking at that folder, and has to attach the file. A report in the config folder is hidden from them, and one printed to the terminal is cut by its scrollback.
  - *Check (code):* `main` handles `--report` before `launch`: it writes the redacted report to a file in the current folder, or in the temp folder when that one can't be written, prints the file's path and opens the new-issue page. A test that runs the report command in a folder and finds the redacted report there, with its path printed, is the surest evidence.
  - *Reference:* Editora `report/ReportCommand.java` (`ReportCommandTest`); Torrenta's `ReportCommand.java` does the same, and so does the app template's `report/ReportCommand.java` (`ReportTest`, `theReportGoesInTheCurrentFolderAndItsPathIsPrinted` and `aCurrentFolderThatCantBeWrittenSendsTheReportToTheTempFolder`). Mirada writes its report beside the settings, and Termina prints its report without writing a file (`ReportCommand.java`).
- **CLI-12** `MUST` An option the app does not know stops the launch: it names the option on standard error, says how to get help, and exits with status 2 before the GUI starts. That holds for an option with one dash as for one with two: it is never ignored and never taken for a file. Two kinds of argument are not unknown options. In an app that opens files named on its command line, a lone `-`, and every argument after `--`, is one of those files, whatever it starts with; that sentence does not cover an app that opens none. And the JVM options a launcher can leave among the program's arguments, `--add-exports`, `--add-opens` and `-p`, are skipped by name, each with its value, whether the value is the next argument or follows `=` in the same one (`--add-exports=…`): never refused, and the value never opened as a file. On Windows the message prints in the console, which is CLI-11.
  - *Why:* a mistyped `--config-dir` that is ignored starts the app on the real settings, which is the one thing the flag was typed to avoid. Status 2 is what a script tests for a usage error. `--` is how a file whose name starts with a dash is opened at all. A launcher's own option is not the user's mistake, so it must not stop the app; it is skipped by name because a parser that skips whatever it does not know is the one that ignores a typo.
  - *Check (code):* the parser `main` runs before the GUI refuses an option the app does not define, with two dashes or with one, and `main` then prints the option and how to get help on standard error and exits with status 2 before `launch`. In an app that opens files named on its command line, the parser takes a lone `-`, and a name that starts with a dash after `--`, for files; in an app that opens none, this part has nothing to find. And the parser skips `--add-exports`, `--add-opens` and `-p` with their values, in both forms a launcher writes (`--add-exports <value>` and `--add-exports=<value>`), with no error and no file. A test that hands the parser each of these arguments is the surest evidence.
  - *Reference:* the app template's `cli/CommandLine.java` does all of it: it refuses an unknown option with one dash or two (`Main.run` prints the message and returns 2) and takes `-` and whatever follows `--` for files, and skips `--add-exports`, `--add-opens` and `-p` with their values in both forms (`CommandLineTest`, `reportsMistakes`, `acceptsTheEqualsFormAndDoubleDash` and `skipsTheLaunchersJvmOptionsAndTakesCapitalVForVersion`). No other app does all of it yet. Termina's `cli/CommandLine.java` and Torrenta's `cli/CommandLine.java` refuse both kinds of unknown option with status 2, and the launcher's options with them. Torrenta, which opens the `.torrent` files and magnet links named on its command line, also refuses `-` and `--`: anything that starts with a dash and is not a magnet link. Termina opens no files from its command line, so that part of the rule does not cover it (it refuses `-` as an unexpected argument and `--` as an unknown option); the launcher's options are all it misses. Mirada's `Launch.java` exits 2 on an unknown option with two dashes, `--add-exports` and `--add-opens` among them, and takes `-` and whatever follows `--` for files, but it takes a single-dash option, `-p` included, for a file too. Editora skips every option it does not know without a word, a lone `-` included, and has no `--` (`App.fileTargets`); that is how it gets past a leaked `--add-exports`, `--add-opens` or `-p`, whose value it opens only when a file of that name exists (`AppArgsTest`). nux reads `--minimized` and `--screenshot=` and ignores every other argument (`App.java`), and Modula reads none (`ModulaApp.java`); neither opens files, so what they miss is the refusal.
- **CLI-13** `SHOULD` The short flags are the same in every app: `-h` for `--help`, and both `-v` and `-V` for `--version`.
  - *Check (code):* the command-line parser takes `-h` for `--help`, and `-v` and `-V` for `--version`.
  - *Reference:* Termina's `cli/CommandLine.java` and Torrenta's `cli/CommandLine.java`; the `--help` text of both lists `-h` and `-v` and leaves `-V` out. The app template's `cli/CommandLine.java` takes all three, and its `--help` text lists them. Editora takes `-h` and `-V` but not `-v`, Mirada has no short flags, and Modula and nux take neither flag in any form.
- **CLI-11** `MUST` On Windows, `--help`, `--version` and `--report` print in the console they were typed in, and so does the message that refuses an unknown option (CLI-12). A GUI launcher has no console, so the installer also carries a console launcher (jpackage's `win-console`) that runs the same entry point, or `main` attaches to the parent's console before it prints.
  - *Why:* without it, the three flags and a mistyped option print nothing from cmd or PowerShell, and the command line differs on one OS.
  - *Check (code):* the Windows jpackage configuration adds a launcher with `win-console=true` that runs the app's own main class and lets the three flags and an unknown option reach it, or `main` calls `AttachConsole` before printing.

- *Reference:* Torrenta meets it with its second launcher, `torrenta-nox` : `src/jpackage/torrenta-nox-windows.properties` sets `win-console=true` , and its `pom.xml` adds that launcher to the image on Windows (the `os-windows` profile picks the file), beside the main launcher and on the same module and main class. The launcher's own `arguments=--headless` applies only when the person gives none (its `CLAUDE.md` says so), so `torrenta-nox --help` , `--version` , `--report` and an option it does not know reach `main` as typed, and `main` refuses the unknown option and answers the three flags before it looks for `--headless` (`App.java`). The app template does it the same way: its `os-windows` profile in `pom.xml` adds a second launcher, `exampleapp-cli` , whose `packaging/windows/exampleapp-cli.properties` sets `win-console=true` and nothing else, so it runs the same main class with the arguments as typed. No other app has a `win-console` launcher or calls `AttachConsole` . Mirada's one Windows executable is its native image, copied over jpackage's launcher (`native-app-image.sh`); whether it prints in the console depends on how that image is linked, which Mirada's files do not say, and no Windows run has confirmed it.

- **`CLI-06`** `SHOULD` An app that opens files or folders (documents, images, a `.torrent`) opens the ones named as arguments on its command line, which its `--help` lists as `[files…]` .
  - *Check (by hand):* launch with two files and find both open, and run `--help` and find `[files…]` in its usage line.
  - *Reference:* Editora `App.fileTargets` . Termina, Modula and nux open no files, so the rule does not cover them.

Launch

- **`CLI-07`** `MUST` One instance runs per config folder, unless a launch asks for another with `--new-instance` (CLI-03). Any other second launch hands its arguments, when it has any, to the running app over a local endpoint and exits. In an app that opens files, the `openFiles` Apple Events of macOS reach the running app too.
  - *Check (code):* the first launch on a config folder becomes the primary and listens on a local endpoint; a second launch without `--new-instance` sends it its arguments and exits; and, in an app that opens files, a handler for the macOS open-files event passes them to the running app (Editora's `MacOpenFiles`). A test that makes the first launch primary and delivers a second launch's arguments to it is the surest evidence.
  - *Reference:* Editora `ipc/SingleInstance.java` (`SingleInstanceTest`), `MacOpenFiles.java` . nux doesn't yet: its second launch asks the running app to show its window and exits, handing over none of its arguments (`instance/SingleInstance.java` , `NUX-SHOW`), and what lets a second instance run on the folder is a setting (`singleInstance` , read in `main` of `App.java`), not `--new-instance` .
- **`CLI-08`** `SHOULD` An advisory lock tells two processes on one config folder about each other.
  - *Check (code):* each process takes an advisory file lock in its config folder, and can tell that another process already holds one there. A test that takes the lock from two processes is the surest evidence.
  - *Reference:* Editora `config/InstanceLock.java` (`InstanceLockTest`).
- **`CLI-09`** `SHOULD` `main` sets `java.awt.headless=true` before any AWT class loads. Only these uses make AWT headful, in an app that has them: the tray on Windows and macOS, and on macOS the dock badge, the open-file handlers and Move to Trash. Linux never needs it. `main` decides once, before that point, which of them the settings ask for on this OS. What a tray app says when its tray is turned on after that is TRAY-14.
  - *Why:* on macOS, AWT's native toolkit and JavaFX contend for the one AppKit run loop and can hang; headless Java2D still draws to images.
  - *Check (code):* `main` calls `System.setProperty("java.awt.headless", …)` before `launch` and before anything that loads an AWT class, and the value is `"true"` unless one of the listed uses is wanted on this OS: always on Linux, on Windows unless the tray is on, and on macOS unless the tray, the dock badge, an open-file handler or, in an app that trashes through AWT, Move to Trash is.
  - *Reference:* Editora `App.main` , always headless, as Termina (`App.java`) and the app template (`Main.java`) are. Torrenta `os/AwtUse.java` (`headful` , which its `App.main` asks once: never on Linux, for the tray on Windows, and for the tray, the dock badge or the open-file handlers on macOS). Mirada `App.java` (headless except on macOS, for Move to Trash: `os/Trash.java` , `headless`). Modula and nux never set the property, so AWT is headful on every OS, Linux included.
- **`CLI-10`** `SHOULD` An app that starts external tools (`git` , a compiler, `ffmpeg`) resolves them through the login shell's `PATH` on macOS and Linux. Windows has no login shell's `PATH` to ask for, and the one the app inherits is used there.
  - *Why:* a `.app` launched from Finder inherits a stripped `PATH` .
  - *Check (code):* on macOS and Linux, every process the app starts gets a `PATH` that includes the login shell's. An app that starts no external tool is outside the rule.
  - *Reference:* Editora `process/ProcessRunner.java` (`augmentedPath` ; on Windows its login-shell probe returns nothing and the inherited `PATH` is used). The app template's `sync/ShellPath.java` does the same for the `git` and `gh` it starts: the login shell's `PATH` before the inherited one on macOS and Linux, the inherited one on Windows (`ShellPathTest`).

Internationalization

Every app ships in at least six languages, and the user's language is chosen the same way in each. Every string the user reads comes from a message catalog by its key, and the rules keep each catalog in step with the English base, keep each message right in its language (apostrophes, numbers, plurals), keep the layout whole in the longest language, and leave alone what must read the same everywhere: command ids, setting keys, file names, chord modifiers and the `--help` text.

Reference: Editora's `i18n/Messages.java` (`tr` , and `available()` behind the Settings language picker), the catalogs in `src/main/resources/com/editora/i18n/` , and `i18n/MessagesTest.java` , which holds them in step.

Languages

Every app ships **at least these six**. They are a floor, not the set: an app `SHOULD` add languages as people ask for them, and a new app can start with more.

Language	Catalog
English (base)	`messages.properties`
Spanish	`messages_es.properties`
Portuguese	`messages_pt.properties`
French	`messages_fr.properties`
Italian	`messages_it.properties`
German	`messages_de.properties`

- **INTL-01** `MUST` The app ships a message catalog for each of at least six languages: English (the base), Spanish, Portuguese, French, Italian and German.
  - *Check (code):* the six `messages*.properties` files are in the app's `i18n` resources.
  - *Reference:* Editora `src/main/resources/com/editora/i18n/` .

Adding a language:

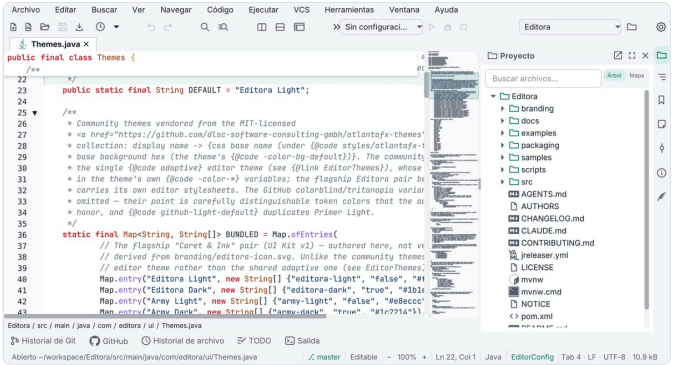
- **INTL-02** `SHOULD` A new language is a `messages_<code>.properties` with every key, listed in the Settings language picker and in the parity test.
  - *Check (code):* the language's code is in `Messages` ' language list (which the picker shows) and in the catalog list the parity test walks.
  - *Reference:* Editora `i18n/Messages.java` (`LANGUAGES` , `available()`); `MessagesTest` 's `LOCALES` and `ALL_CATALOGS` , after which `translationCatalogsHaveExactlyTheBaseKeySet` holds it to the same keys as the others.
- **INTL-03** `SHOULD` Every value is translated for real: no English value is shipped in another language's catalog.
  - *Check (code):* compare each catalog's values with the base; a value identical to the English one is a name, a term or a bare pattern that stays the same in that language ("Git: Pull (Rebase)", "{0} · {1}"), never an untranslated sentence. No test catches it, so it is also checked in review.
  - *Reference:* Editora `messages_*.properties` .
- **INTL-04** `SHOULD` The layout holds in every language at 1024 px wide: rows wrap, never cut. German is today's longest.
  - *Check (by hand):* run the app in each language with the window 1024 px wide, open each window and Settings page, and find no row cut off.
  - *Reference:* Editora `messages_de.properties` .
- **INTL-05** `SHOULD` A Chinese, Japanese or Korean catalog comes with a font fallback for its script.
  - *Why:* Inter has no CJK glyphs. JavaFX falls back to whatever font the OS has, so without a bundled one the text looks different on each OS, and a minimal Linux install may have none.
  - *Check (by hand):* run the app in that language and look for missing-glyph boxes.
  - *Reference:* no app yet.
- **INTL-06** `SHOULD` An Arabic or Hebrew catalog sets `NodeOrientation.RIGHT_TO_LEFT` on every scene.
  - *Check (code):* the scene setup sets the node orientation from the current language.
  - *Reference:* no app yet.

Rules

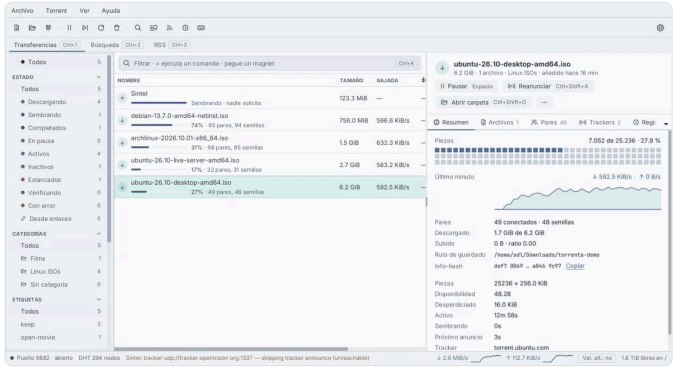
- **INTL-07** `SHOULD` No raw English literal is passed to a control: every string comes from the catalog, by its key, through one lookup in one class (`tr("some.key")` in Editora's `Messages`).
  - *Check (code):* no source file passes a string literal in a text position: a control's constructor, `setText` , `setPromptText` , `setTitle` , a tooltip, a menu item or a command's label. Each takes its text from the catalog's lookup, which is one method of one class (Editora's `Messages.tr`). A test that scans the sources and fails on such a literal is the surest evidence.
  - *Reference:* Editora `i18n/Messages.java` . The tests that hold the rule are Torrenta's `UiStringSweepTest` (`noStringLiteralInATextPosition`) and nux's `NoLiteralUiTextTest` (`userFacingTextGoesThroughTheCatalogue`). Editora's `MessagesTest.everyLiteralKeyUsedInTheSourcesExists` checks only that every key a `tr("…")` call names is in the base catalog, not that no literal reaches a control.
- **INTL-08** `MUST` Every catalog has every key of the base catalog, and no more, each with a non-empty value.
  - *Check (code):* compare each translated catalog's keys with the base catalog's: none is missing, none is extra, and no value is empty. A parity test that fails the build on any of the three is the surest evidence.
  - *Reference:* Editora `MessagesTest.translationCatalogsHaveExactlyTheBaseKeySet` , `translationsAreNonEmptyAndDistinctFromKeys` .
- **INTL-09** `SHOULD` In a catalog value with a placeholder, a literal apostrophe is doubled; in a value without one, it is written once. Inside a `{0,choice,…}` limb (INTL-12) that itself holds a placeholder, it is written four times; in a limb without one it is doubled, like the rest of the value.

- *Why:* a value with a placeholder is a `MessageFormat` pattern, where a lone apostrophe starts a quoted section and can swallow the placeholder after it; a value without one is shown exactly as written. A choice limb that holds a placeholder is parsed a second time as a pattern of its own, and each pass halves the apostrophes.
- *Check (code):* in every catalog, a value with a placeholder has no lone apostrophe and a value without one has no doubled apostrophe; a choice limb that holds a placeholder writes a literal apostrophe four times, and a limb that holds none writes it twice. Catalog tests that fail each of the four mistakes are the surest evidence.
- *Reference:* Editora `MessagesTest.aValueWithAPlaceholderNeverHasALoneApostrophe`, `aValueWithoutAPlaceholderNeverDoublesAnApostrophe`. The choice limbs are held by Torrenta's `MessagesTest.apostrophesInsideChoiceLimbsAreQuotedForTheRightNumberOfPasses` (`diagnosis.unavailable` in its `messages_fr.properties` writes "n''''est" in a limb with `{1}`); Editora's test has no such case.

- **INTL-10** `SHOULD` Identifiers such as line numbers and ports use `{0,number,#}`, so they are not digit-grouped.
  - *Check (code):* in every catalog, a placeholder the code fills with an identifier (a line number, a port) carries the `number,#` format, as in `{0,number,#}`, and is never a bare `{0}`. A test that formats each identifier argument in every locale and fails on a grouped number is the surest evidence.
  - *Reference:* Editora `MessagesTest.identifierArgumentsAreNeverDigitGrouped`.
- **INTL-11** `SHOULD` Technical identifiers stay untranslated: command ids, setting keys, file names and protocol names read the same in every language. The `--help` and `--version` text is CLI-01.
  - *Check (code):* compare each translated catalog with the base, key by key: a command id, a setting key, a file name or a protocol name in the base value appears, spelled the same, in the translated value. A made-up example name inside a value (`name.gpg`) is not an identifier, and may be translated.
  - *Reference:* Editora's catalogs (`messages.properties` and its translations).
- **INTL-12** `SHOULD` A message with a count words it for the count, with a `{0,choice,…}` limb for each form the language needs: never "file(s)".
  - *Why:* "(s)" is English; other languages change more than one ending, and the translator can only get it right with the forms written out.
  - *Check (code):* no catalog value has a placeholder and `(s)`; a message with a count has a `{0,choice,…}` pattern with a limb for each form its language needs, and every such pattern parses. A catalog test that fails a value with a placeholder and `(s)`, and parses every `choice` pattern in every catalog, is the surest evidence.
  - *Reference:* Torrenta's `omni.targets.matching` in `messages.properties`, parsed in every language by `MessagesTest.placeholdersMatchAcrossLanguagesAndEveryPatternParses`. That test only parses the patterns: no app has the `(s)` test yet, and Editora's base catalog has 22 values with a placeholder and `(s)` (`find.replaced`, "Replaced {0} occurrence(s)").
- **INTL-13** `SHOULD` A chord's modifiers read the same in every language: `Ctrl`, `Alt`, `Shift` and the macOS glyphs are never translated ("Ctrl", not "Strg").
  - *Why:* the keymaps, the docs and the screenshots use one spelling, and a user who changes the language keeps the chords they learned.
  - *Check (code):* the code that writes a chord out takes its modifier names from constants, not the catalog, and no catalog value writes a modifier in another language: a catalog test looks for Strg, Umschalt, Maj, Mayús and Maiusc.
  - *Reference:* Editora `command/ChordFormat.java` writes chords with untranslated modifiers. Its catalogs do not hold the rule yet: they translate modifiers in prose ("Strg+D", "Mayús+Tab", "Maj+Entrée") and in the Project Map help's key names (`messages_de.properties`, `project.map.help.key.control=Strg`, read by `ui/ProjectMapHelp.java`). No app has the catalog test yet.
- **INTL-14** `MUST` The app starts in the language the user chose in its settings, where it offers that choice; with none chosen, in the system's language when it ships a catalog for it; otherwise in English. The locale that formats dates and numbers keeps the system's region: with no language chosen it is the system's locale whole, whatever language the menus are in, and with one chosen it is still the system's locale when that is the system's own language, and the bare language only when it is another. That locale is the JVM's default once the app has started: the app sets it wherever it is not the locale the JVM started with. The message catalog formats its values with it too, never with a locale of its own made from the bare language.
  - *Why:* `de-AT` collapsed to `de` writes an Austrian's numbers and dates the German way, and an English UI on a French system has no reason to change how its dates are written. Apps that settle this differently format the same date two ways side by side, and so does one app whose catalog formats a message's number with another locale than the rest of the window.
  - *Check (code):* the start-up code takes the stored choice, the system's locale and the catalogs the app ships and gives the language and the locale as the four cases ask: none chosen with a catalog for the system's language (the system's locale whole); none chosen without one (English, the system's locale kept whole); the system's own language chosen (its region kept); another language chosen (the bare language). It sets that locale as the JVM's default (`Locale.setDefault`) as it starts, wherever that is not the locale the JVM started with: an app that offers no language choice and ships no catalogs leaves the default alone, which is the system's locale whole, and meets this part. And the catalog's lookup formats a value with the default locale (`new MessageFormat(pattern)`, or `MessageFormat.format`), not with one it builds from the language code (`Locale.forLanguageTag`). A test that covers the four cases is the surest evidence.
  - *Reference:* Termina `i18n/Messages.java` (`resolve`, `defaultLocale`, and `tr`, which formats with the default locale) with `MessagesTest.java` (`automaticKeepsTheOperatingSystemsLocaleWhole`, `anExplicitChoiceMovesTheLocaleButKeepsAMatchingCountry`). Torrenta carries the same two functions and tests and sets the default from them (`Startup.java`, `language`), but doesn't meet the rule yet: its catalog formats every message with the bare language (`i18n/Messages.java`, `locale`), so a number inside a message loses the region the rest of the window keeps. nux's `i18n/Messages.java` (`uiLocale`, `UiLocaleTest`) keeps the region when the language is the system's, but with none chosen and no catalog for the system's language it sets bare English. Editora (`App.java`) and Mirada (`App.java`) set the default to the bare language every time, so the region is lost; Mirada's catalog formats with the bare language as well (`i18n/Messages.java`). The app template meets it: its `i18n/Messages.java` gives the four cases (`locale`), sets the result as the JVM's default where it differs (`start`) and formats with `new MessageFormat(value)` (`render`); its `MessagesTest` (`theFormattingLocaleKeepsTheSystemsRegion`) covers the four. Modula ships no catalogs and offers no language choice, and leaves the JVM's locale as the system set it, which is the locale the rule asks for: it meets this rule, and its missing catalogs are INTL-01's.



Editora in Spanish: menus, tool windows and the status bar all come from the catalogs.



Torrenta in Spanish.

Command Palette

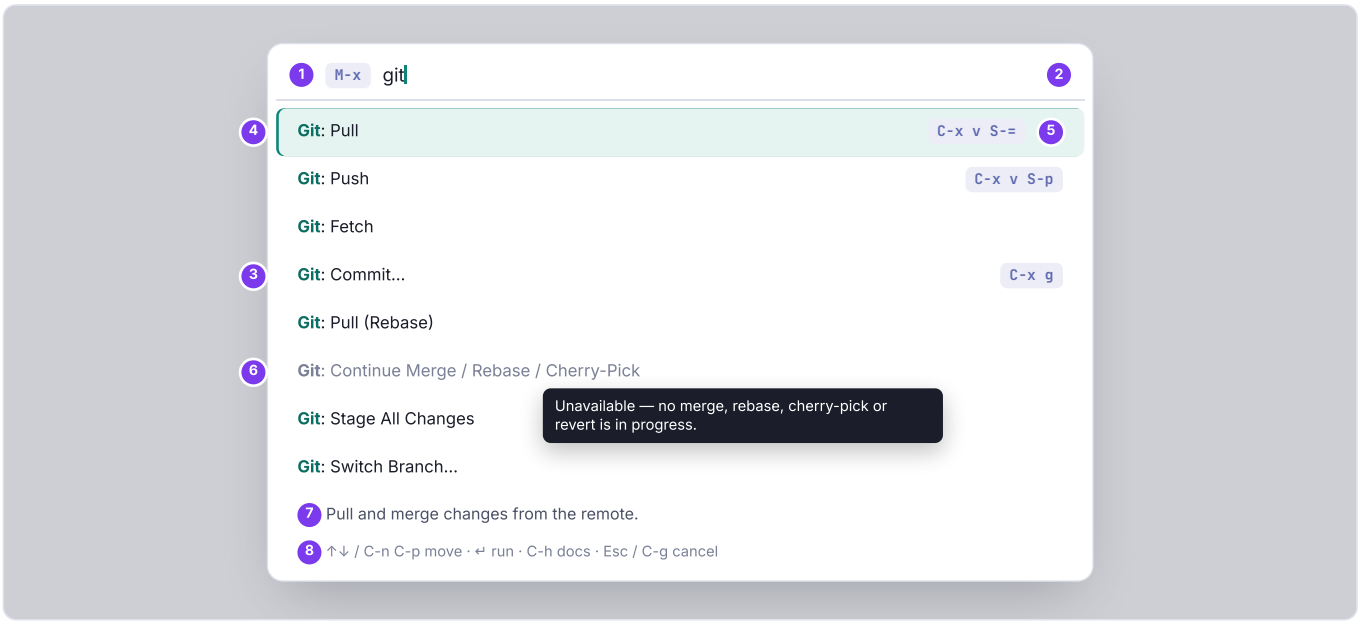
One palette per main window lists every registered command, so a command that exists is a command people can find. The pure parts (`FuzzyMatch` , `MatchText` , the key helper, `PickerKeys` in Editora and `PaletteKeys` in the copies, and the caret-key helper, `TextInputKeymap` in Editora and `TextInputKeys` in the copies) are shared with Termina and Mirada. Torrenta shares `MatchText` and ranks with its own `CommandMatcher` .

Rules

- **`PAL-01`** `MUST` Every app has a Command Palette, one in each main window, opened by the `palette.show` command. A Settings, About or log window needs none of its own. Its chords, `M-x` in the Emacs keymap and `Ctrl+Shift+P` (`Cmd+Shift+P` on macOS) in the others (`Ctrl+Shift+A` allowed in the IntelliJ keymap), are KBD-05.
  - *Check (code):* each main window builds one palette, and the command registered as `palette.show` opens it (in Editora the window controller builds one `CommandPalette`). A window that is not a main window (Settings, About, a log, a history or events window) is not judged.
  - *Reference:* Editora `ui/CommandPalette.java` , built once in `MainController` .
- **`PAL-02`** `MUST` The palette lists every registered command, with its title and description (KBD-03).
  - *Check (code):* the palette's rows come from the whole registry, filtered only by the query; a command that cannot run stays listed, grayed; each row shows its command's title; and the palette shows the description of the command under the cursor, in the row or in a line of the card. A palette that shows no description fails this rule; that every command has one is KBD-03's.
  - *Reference:* Editora `CommandPalette.filter` (`registry.all()`), `MessagesTest` ; the page's Spec. Torrenta's, Modula's and nux's palettes don't yet. Torrenta's lists only the commands that can run now: `show` in `ui/CommandPalette.java` drops every disabled command, after `offeredCommands` in `ui/MainWindow.java` has dropped what the engine cannot run and what Simple mode hides; and it shows no description, since its `Command` has none. Modula's leaves out one (`stereo` , while its control is disabled, in `ui/RadioPane.java`), and its rows show a title and a shortcut, with no description (`ui/CommandPalette.java`). nux's lists its whole registry, and each row shows the title and the command's id, with no description: its commands have none (`ui/CommandPalette.java` , `CommandCell` ; `command/CommandRegistry.java`).
- **`PAL-10`** `SHOULD` The palette is 620 px wide, its rows 36 px high with 8 visible before the list scrolls, at the default zoom (WIN-12), and its field's prompt reads "Run command...", in words from the catalog (Editora's key for them is `palette.prompt`).
  - *Why:* the palette is the one window every app shares; a different width or prompt is the first thing a person moving between apps notices.
  - *Check (code):* the palette's width, row height and visible rows come from constants of these values, and its prompt is read from the catalog, where the English is "Run command...". The rule asks for the words, not for a catalog key.
  - *Reference:* Editora `ui/CommandPalette.java` (`ROW_HEIGHT` , `VISIBLE_ROWS`), Torrenta, Termina and the app template's `ui/Picker.java` . Mirada's `ui/CommandPalette.java` matches the three sizes and reads the same words from the catalog under another key, `palette.runCommandPrompt` . Modula's is 420 px with a literal "Type a command...", and nux's 540 px with a `palette.prompt` that reads "Type a command...".

The palette is an in-scene overlay in the window's root `StackPane` , over a faint scrim, never a `javafx.stage.Popup` : KBD-07.

Anatomy



1. **Opener chip.** The chord that opened the palette, read from the active keymap. No chip when the user has unbound it, never a wrong one.
2. **Query field.** The first row is the input; there is no title. Prompt "Run command...". Emacs caret keys work inside it.

3. **Match highlight.** The characters the fuzzy matcher scored, bold in the accent color. The highlight and the ranking come from the same call, so they never disagree.
4. **Selected row.** Tinted background plus a 2 px accent bar on its inside edge. The bar is what makes the selection readable in a long list.
5. **Key chip.** The command's binding in the active keymap, in the keyboard color (brace periwinkle). Absent when unbound.
6. **Unavailable command.** Listed, grayed, never selected and inert to clicks. Its tooltip says why and what to do ("Unavailable — no merge, rebase, cherry-pick or revert is in progress.").
7. **Description.** The selected command's `command.<id>.desc`, one line with an ellipsis, always one line tall so the card never jumps.
8. **Key legend.** The palette's own keys, rendered from the active keymap. The mockup shows the Emacs keymap's: `C-n`, `C-p` and `C-g` are its line and cancel chords (PAL-06, PAL-07).

Spec

Property	Value	Reference
Kind	In-scene overlay in the window's root `StackPane`, over a faint scrim. **Not** a `javafx.stage.Popup`: on Windows a popup does not reliably take keyboard focus.	`OverlayHost`
Width	620 px (PAL-10)	`content.setPrefWidth(620)`
Rows	36 px high, 8 visible before the list scrolls (PAL-10)	`ROW_HEIGHT`, `VISIBLE_ROWS`
Card	`-color-bg-overlay`, 1 px `-color-border-default`, radius 12, padding 6, one wide drop shadow	`.command-palette` in `app.css`
Row text	12.5 px; padding 7 × 16	`.command-palette .list-cell`
Scrim	`rgba(0,0,0,0.18)`, the value of a token defined on `.root` (ICON-08); a click on it closes the palette	`.overlay-backdrop`
Content	Every registered command: title from `command.<id>`, description from `command.<id>.desc`	`CommandRegistry`, `MessagesTest`
Ranking	Editora scores the title; when the title does not match, the id's namespace word (`markdown` for `markdown.tableAddRow`, "Table: Add Row"), then the names other IDEs give the command. The description is shown, not matched. A query that names a title outranks a fuzzy hit. The app template scores the title and the words of the id, and shows the description	`FuzzyMatch`, `Command.keywords`, `OtherIdeNames`, `titlePhraseScore`; the template's `ui/Picker.java`

Behavior

- **PAL-04** `SHOULD` Every open starts with an empty query: the palette does not keep the last one, so it opens on the whole list (PAL-02).
  - *Check (code):* `show` clears the field and filters on the empty query before the card appears.
  - *Reference:* Editora `CommandPalette.show`.
- **PAL-05** `SHOULD` Typing in the palette filters its list, and the first *enabled* match is selected.
  - *Check (code):* after each filter the palette selects the first row whose command can run, stepping over disabled rows at the top (Editora's `selectFirstEnabled`). A test that asserts it, with a disabled command at the top, is the surest evidence.
  - *Reference:* Editora `CommandPalette.selectFirstEnabled`, `CommandPaletteDisabledFxTest`.
- **PAL-06** `SHOULD` The active keymap's line and page chords move the selection, as do ↑ ↓ and Page Up/Down, which always work. In Editora they are the chords of its `nav.lineDown`, `nav.lineUp`, `nav.pageDown` and `nav.pageUp` commands, `C-n` and `C-p` among them in the Emacs keymap. Enter, or a click on an enabled row, runs the command. The palette closes first, then the command runs.
  - *Check (code):* the palette's key handler looks a press up in the active keymap and moves on the commands it binds for a line and a page up and down, not on fixed letters, and on ↑ ↓ and Page Up/Down whatever the keymap; the run path hides the card before it calls the registry, and does nothing on a disabled row.
  - *Reference:* Editora `CommandPalette.runSelected`, `PickerKeys.resolve`, and the app template's `ui/PickerKeys.java` (`resolve`, for its `ui/Picker.java`), which reads the keymap's `nav.lineDown`, `nav.lineUp`, `nav.pageDown` and `nav.pageUp` beside the fixed keys: the two palettes that ask the keymap. The others move on fixed keys: Termina's `ui/PaletteKeys.java` and Mirada's `ui/PaletteKeys.java` on a fixed table of Emacs chords beside them (`EMACS`: `C-n`, `C-p`, `C-v`, `M-v`); Torrenta's `ui/CommandPalette.java` (`onKey`) on the arrows and `Ctrl+N` and `Ctrl+P`, with no page keys; Modula's `ui/CommandPalette.java` and nux's `ui/CommandPalette.java` on the arrows alone.
- **PAL-07** `SHOULD` The active keymap's cancel chord, Esc or a click on the scrim cancels the palette, and focus returns to where it was. In Editora the chord is that of its `edit.cancel` command, `C-g` in the Emacs keymap.
  - *Check (code):* Esc, the chord the active keymap binds to the cancel command, and a press on the backdrop all hide the overlay, and hiding focuses the node that had focus before it opened. In an app with one key table (a terminal app's keymap, or the built-in table of an app that ships no keymap files), the chord is the one that table binds to a cancel command; where it binds none, Esc and the backdrop are all there is to test.
  - *Reference:* Editora `PickerKeys` (`isCancel`), `OverlayHost` (`previousFocus`), and the app template's `ui/OverlayHost.java` (`cancelOn`, which `ui/MainWindow.java` hands the keymap's `edit.cancel` chord, `C-g` in its Emacs keymap, and `hide`, which gives the focus back): the two that ask the keymap for the chord. `Ctrl+G` is fixed in the `EMACS` table of Termina's `ui/PaletteKeys.java` and Mirada's `ui/PaletteKeys.java`. Torrenta's, Modula's and nux's palettes cancel on Esc and on a press on the backdrop: none of the three apps has a keymap or a cancel command, so the chord asks nothing more of them.
- **PAL-08** `SHOULD` In the palette, a help chord opens the selected command's page in the docs on the website, for the running version. The chord is the first of `Ctrl+H`, `F1` and `Shift+F1` that the active keymap leaves free. It is the same on macOS, where `Ctrl+H` is the Control key and has no `Cmd` form (`C-h` in Emacs notation).
  - *Check (code):* the palette's help chord is the first of the three that the active keymap neither binds nor starts a longer chord with, and the URL it opens is built from the app's versioned docs address and the selected command's id.
  - *Reference:* Editora `CommandPalette` (`PickerKeys.freeChord` over `C-h`, `f1` and `S-f1`; `docsUrl`), `AppInfo.docsUrl`. The app template's `ui/PickerKeys.java` (`HELP_CHORDS`, `helpChord`) picks the chord the same way, and the palette opens the command's page under its versioned docs (`PickerFxTest`,

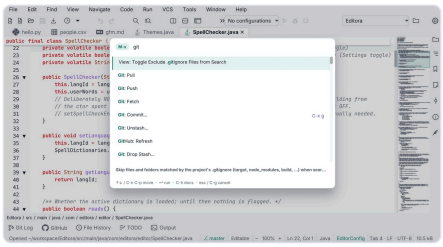
`helpTitleCurrentValueAndTheEnds`). No other app's palette has a help chord yet.

- **PAL-09** `SHOULD` A live keymap switch re-renders the palette's key chips, its opener chip and its key legend.
  - *Check (code):* applying a new keymap calls the palette's rebind method, which rebuilds the opener, the legend and the rows.
  - *Reference:* Editora `CommandPalette.refreshBindings`, called from `SettingsReapply`; the app template's `Picker.rebind` (`ui/Picker.java`), which its palette runs on a keymap change with the new opener, rows and legend (`ui/MainWindow.java`).

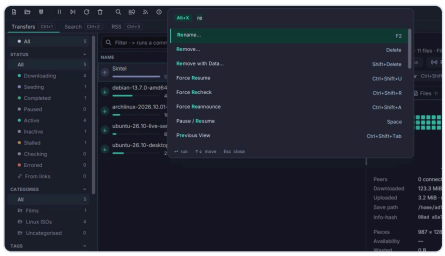
Copy

Key	English
`palette.prompt`	Run command…
`palette.disabled.<reason>`	Unavailable — *what is missing. What to do first.*

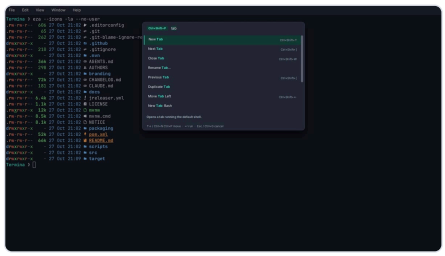
In the apps



Editora.



Torrenta.



Termina.

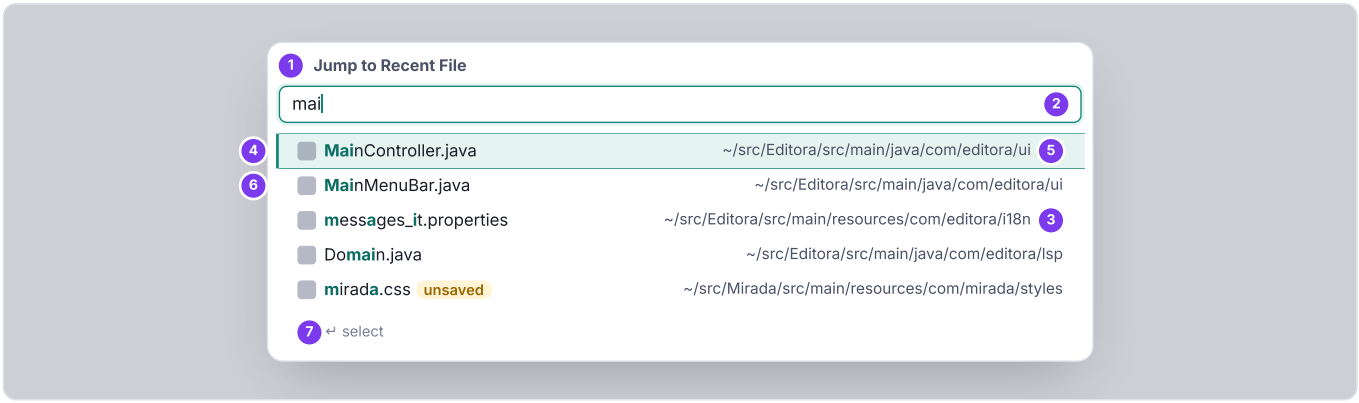
QuickOpen picker

The picker for every "choose one from a list" moment: recent files, themes, keymaps, a setting's values, a branch to compare with. It is the Command Palette card with a title over the field and a prompt in it, so a list choice is made the way a command is run: type a few letters, Enter. QOPEN-01 asks for the one picker; every other rule on this page is the picker's shape and applies to an app that has one. Four of the apps have none (Uses).

Rules

- **`QOPEN-01`** `SHOULD` One picker serves every list choice: recent files, themes, keymaps, a setting's values, a branch to compare with.
  - *Check (code):* those choices are instances of one generic picker class; there is no second list-picker class or `ChoiceDialog` for them.
  - *Reference:* Editora `ui/QuickOpen.java` (one generic class, instantiated more than seventy times), driven by `PickerKeys`. Editora still asks with a `ChoiceDialog` in three places: the Git reset mode and the merge parent to revert against (`ui/GitWindowCoordinator.java`), and the settings backup to restore (`ui/SettingsResetFlow.java`).
- **`QOPEN-02`** `SHOULD` In an app with a picker (a card, opened by a command, for choosing one thing from a list by typing; a Command Palette that lists only commands is not one, and nor is a menu, a combo box or a field's completion list), the picker is the palette's card in the shared in-scene overlay host, with a title naming the command over the field and a prompt in the field. That it is never a `javafx.stage.Popup` is KBD-07.
  - *Check (code):* each picker shows its card through the overlay host the palette uses and sets a title and a prompt text.
  - *Reference:* Editora `QuickOpen.setOverlayHost`, `ui/OverlayHost.java`; the page's Spec. The app template's pickers (theme, keymap, language, Bind Key) are titled cards of its `ui/Picker.java` in its `ui/OverlayHost.java`.

Anatomy



1. **Title.** The command that opened the picker ("Jump to Recent File", "Set App Theme"), bold 12 px in the muted color. This is what tells the picker apart from the palette, which has no title.
2. **Query field** with a prompt per picker: "Type to filter recent files...", "Type to filter themes...". The Emacs caret keys work in it (`TextInputKeymap.installShared`).
3. **Match highlight.** The characters the fuzzy matcher scored, bold in the accent color; the same `MatchText.runs` as the palette, so the highlight and the rank never disagree.
4. **Selected row.** The first match, or with nothing typed the value in force when the picker knows it (`setCurrentItem`), so a settings picker opens on the current value.
5. **Detail**, right-aligned, in the palette's key-chip style (the keyboard color, `-state-key`, in Editora): the file's folder, a branch's upstream, "current". It gives way with an ellipsis; the title keeps its width and never wraps.
6. **Icon** (optional, `setItemIcon`): a file-type glyph from `FileIcons`. A per-row class marks state (an unsaved buffer in amber).
7. **Key legend.** "↵ select" plus any row action's key, rebuilt from the active keymap on every showing.

Spec

Property	Value	Reference
Kind	The palette's card in the shared in-scene `OverlayHost`, over the same scrim; never a `javafx.stage.Popup`	`QuickOpen.setOverlayHost`, `OverlayHost`
Width	620 px; a picker may ask for its own width and a minimum number of rows (a registry browser with one entry still gets a roomy box)	`DEFAULT_WIDTH`, `setPreferredSize`
Rows	26 px fixed cells; the list hugs its match count, at least 1 row, 12 before it scrolls	`CELL_HEIGHT`, `MAX_VISIBLE`, `resizeList`
Title	Bold 12 px, `-color-fg-muted`, over the field	`.command-palette .palette-title`
Row	Title as matched runs, an optional icon before it, the detail in the palette's key-chip style, `-state-key` in Editora	`ItemCell`, `.keybinding`

Property	Value	Reference
Legend	11 px muted under the list	`.command-palette .palette-hint`
Card	`-color-bg-overlay`, radius 12, 1 px `-color-border-default`, padding 6: the palette's	`.command-palette`
Ranking	`FuzzyMatch` over the search key (the label unless the picker names another, a note's body for instance); equal scores keep the supplier's order	`filter`

Behavior

Applies to: an app with a picker: a card, opened by a command, for choosing one thing from a list by typing (a Command Palette that lists only commands is not one, and nor is a menu, a combo box or a field's completion list).

- **QOPEN-03** `SHOULD` A picker reads its items fresh on every showing and snapshots them, so typing filters a stable list. A source that answers late (an outline the language server has not returned yet) has the picker read its items again and keep the query (`reload()` in Editora).
  - *Check (code):* showing a picker calls its items supplier, and a late source makes the picker read the items again while it keeps the query (Editora's `show` and `reload`).
  - *Reference:* Editora `QuickOpen.show`, `QuickOpen.reload`.
- **QOPEN-04** `SHOULD` With nothing typed, a picker's list keeps the supplier's own order (recency for recent files, document order for symbols), with the current value pre-selected when there is one.
  - *Check (code):* a picker's empty query passes the items through unsorted and selects the value in force when the picker was given one (Editora's `setCurrentItem`).
  - *Reference:* Editora `QuickOpen.filter`, `setCurrentItem`; the app template's `ui/Picker.java` (`current`), whose theme, keymap and language pickers open on the choice in force (`PickerFxTest`, `helpTitleCurrentValueAndTheEnds`).
- **QOPEN-05** `SHOULD` Typing in a picker ranks its rows by fuzzy score, and ties keep the supplier's order, so rows do not shuffle under the cursor between keystrokes.
  - *Check (code):* the last tiebreak of the picker's sort is the supplier's index.
  - *Reference:* Editora `QuickOpen.filter`.
- **QOPEN-06** `SHOULD` In a picker, the active keymap's line and page chords move the selection (in Editora, the chords of its `nav.lineDown`, `nav.lineUp`, `nav.pageDown` and `nav.pageUp` commands), as do ↑ ↓, Page Up/Down and `Ctrl+Home` / `Ctrl+End` (`Cmd` on macOS), which always work; plain Home and End stay with the field's caret. Enter or a click on a row chooses; the card hides first, then the choice runs, so focus is already back where it belongs.
  - *Check (code):* the picker's key handler moves on the commands the active keymap binds for a line and a page up and down, and on the fixed keys whatever the keymap; the accept path hides the card before it calls the choice handler.
  - *Reference:* Editora `QuickOpen` (`hide()` then `onChoose`), `PickerKeys.resolve`, the only picker that asks the keymap. The app template's `ui/Picker.java` asks the keymap the same way, through its `ui/PickerKeys.java` (`resolve`), and takes `Ctrl+Home` and `Ctrl+End` (`Cmd` on macOS); Mirada's picker is its palette, whose `ui/PaletteKeys.java` carries a fixed table of Emacs chords (`EMACS`) beside the fixed keys. Termina, Torrenta, Modula and nux have no picker (Uses).
- **QOPEN-07** `SHOULD` The active keymap's cancel chord, Esc or a click on the scrim cancels a picker, and the overlay host restores focus. In Editora the chord is that of its `edit.cancel` command, `C-g` in the Emacs keymap. A picker that previewed something undoes it.
  - *Check (code):* Esc, the chord the active keymap binds to the cancel command, and a press on the backdrop each hide the picker; hiding gives the focus back to the node that had it before the picker opened; and a picker that previewed a row runs its undo when it is dismissed without a choice. A test that asserts the undo is the surest evidence.
  - *Reference:* Editora `OverlayHost` (`previousFocus`; it cancels on `PickerKeys.isCancel`), `QuickOpenPreviewFxTest` (`dismissingRestoresWhatWasPreviewed`). The app template's `ui/OverlayHost.java` cancels its pickers on Esc, on the scrim and on the keymap's `edit.cancel` chord (`cancelOn`), and gives the focus back; none of its pickers previews anything. `Ctrl+G` is fixed in the `EMACS` table of Mirada's `ui/PaletteKeys.java`.
- **QOPEN-08** `MAY` A picker previews the highlighted row as the selection moves and undoes the preview on cancel (Editora's Recent Locations shows each place and returns to the origin on Esc). A preview without an undo is worse than none, and nothing is previewed for the row the picker opens on, only for one the user moved to. Editora's theme and keymap pickers do not preview; they apply on Enter.
  - *Check (code):* a previewing picker previews from its selection listener once it is showing, never for the row it opens on, restores what it previewed when dismissed, and does not restore after a choice. A test that asserts it is the surest evidence.
  - *Reference:* Editora `QuickOpen.setPreview(onPreview, onCancelled)`, `QuickOpenPreviewFxTest`.
- **QOPEN-09** `MAY` A row action acts on the highlighted row and keeps the picker open (Editora's breakpoints picker enables or removes one), re-reading the list afterwards and keeping the cursor in place.
  - *Check (code):* a row action's key runs the action on the highlighted row without hiding the card, and the picker then reads its items again and keeps the cursor on that row. A test that asserts it is the surest evidence.
  - *Reference:* Editora `QuickOpen.addRowAction(key, legend, action)`, `QuickOpenPreviewFxTest` (`aRowActionActsOnTheHighlightedRowAndKeepsThePickerOpen`).
- **QOPEN-10** `SHOULD` A picker that can open on an empty list shows a placeholder sentence in it, never a blank card. A picker with nothing to offer may instead say so in the status line and not open.
  - *Check (code):* each picker whose source can be empty either calls `setPlaceholder` or returns before `show` with a status message.
  - *Reference:* Editora `QuickOpen.setPlaceholder` (`BreakpointsPicker`); `NavigationCoordinator` (`status.nav.noRecentLocations`).
- **QOPEN-11** `SHOULD` The overlay host a picker shows in holds one card at a time; opening a picker replaces whatever card was up.
  - *Check (code):* showing a card in the overlay host removes the previous card before it adds the new one, and keeps the original focus owner (Editora's `OverlayHost.show`).
  - *Reference:* Editora `OverlayHost.show`.
- **QOPEN-12** `SHOULD` A picker's relatives, the other list overlays an app has (in Editora the open-files switcher, the find-file field, the multi-select picker, and the branch and stash popups), move, accept and cancel with the same keys as the picker.
  - *Check (code):* each of those overlays moves, accepts and cancels on the keys the picker does: the moves of QOPEN-06, Enter, and the cancel chord and Esc of QOPEN-07. In Editora each maps its keys through the helper the picker uses (`PickerKeys`), which is one way to hold it.
  - *Reference:* Editora `Switcher`, `FileFinder`, `MultiSelectPicker`, `BranchPopup`, `GitStashPopup`.

The relatives: the Open Files `Switcher` lists tabs in tab order, opens on the current one and closes a file with `C-d` or Backspace. `FileFinder` is the Emacs `find-file` field with Tab completion. `MultiSelectPicker` ticks several rows with Space where exactly one would be wrong. `BranchPopup` and `GitStashPopup` are sectioned pickers with their own actions. The app template's one relative, its tab switcher (`ui/Switcher.java`), moves through the same `ui/PickerKeys.java` and accepts and cancels as its pickers do.

Uses

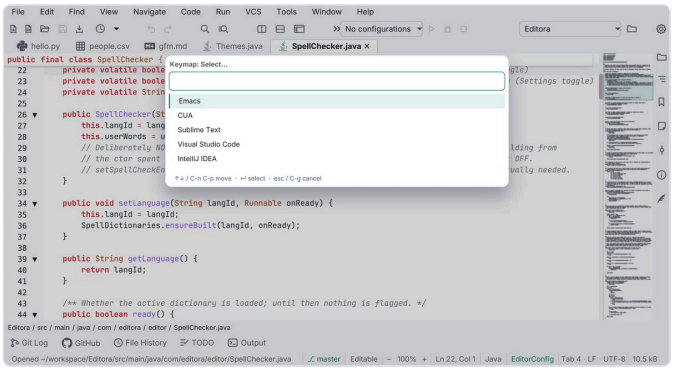
App	Picked with it	Reference
Editora	Recent files (with file icons), recent locations (previewed), bookmarks, the structure outline, tool windows, open files, projects, keymap, app and editor theme, spell language, a setting's values (opened on the current value), branches, tags and commits to compare with, remotes, commit message history, the kill ring, Occur lines, external tools, macros, the debug main class, breakpoints, breakpoint groups, loaded sources and modules, agent models	`NavigationCoordinator`, `EditorSettingsCoordinator`, `DiffCoordinator`, `DebugCoordinator`, `BreakpointsPicker`, `BreakpointGroups`, `LoadedSourcesPicker`
Mirada	Go to bookmark..., Open recent folder...: the palette shown over a list of names with a detail per row and an "open" legend	`CommandPalette.Choices`, `ViewerWindow`
Termina, Torrenta, Modula, nux	Commands only; profiles and other lists are menus or combo boxes today	`CommandPalette` in each

Copy

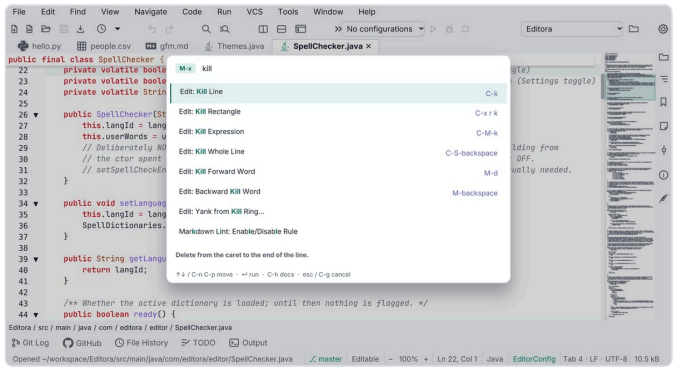
Key	English
`nav.recentFiles.title` / `.prompt`	Jump to Recent File / Type to filter recent files...
`palette.theme.appTitle`, `palette.theme.editorTitle` / `palette.theme.prompt`	Set App Theme, Set Editor Theme / Type to filter themes...
`command.keymap.select` / `palette.keymap.prompt`	Keymap: Select... / Type to filter keymaps...
`palette.setting.pick`	Type to filter...
`pickerKeys.select`, `pickerKeys.open`, `pickerKeys.cancel`	{0} select, {0} open, {0} cancel
`switcher.header`	Open Files

A title names the command; a prompt starts with "Type to filter" and ends with an ellipsis. A placeholder says what is missing and what to do ("There are no breakpoints. Click left of a line number to set one.", `status.debug.noBreakpoints`).

In the apps



Editora: Keymap: Select... over the editor, the title above the field.



The palette the picker is built from.

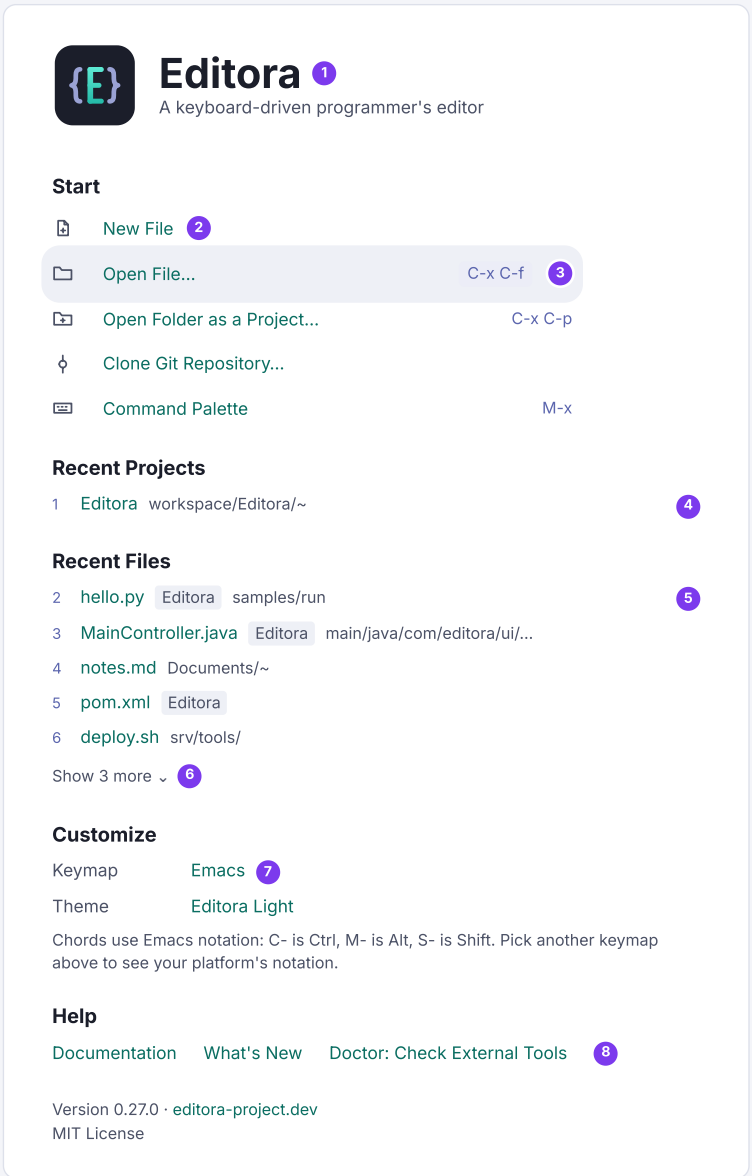
Welcome page

The page shown when there is no session to restore, and the answer to "what do I do now?". The text rules are in Editora's `ui/WelcomeText.java` and the strings under `welcome.*`.

Rules

- **`WELC-01`** `SHOULD` An app that opens files or projects in tabs shows a Welcome page when there is no session to restore.
  - *Check (code):* startup with nothing restored opens the Welcome page.
  - *Reference:* Editora `ui/WelcomePane.java`, `MainController.addWelcomeTab`.
- **`WELC-02`** `SHOULD` An app whose window *is* its content, and can start with nothing in it (a list with no items, a viewer with nothing open), shows its own empty state in place of a Welcome page (WELC-01), with the same ingredients: the ways in, each with its chord.
  - *Check (by hand):* start the app with nothing in it and read the empty state's keys. An app that always opens into content (a terminal with its shell, a radio, a monitor) has no empty state to judge.
  - *Reference:* Torrenta `ui/Welcome.java`; Mirada `ui/ViewerWindow.java` (`offerToOpen`, `viewer.emptyHint`); both are described under In the apps.

Anatomy



1. **Identity.** The app icon at 64 px beside the name (2.4 em, bold) and the README's one-line tagline, muted. One centered column, at most 640 px wide and never narrower than 340; a longer window scrolls.
2. **Start actions**, each an existing command: New File, Open File, Open Folder as a Project (only with Projects on), Clone Git Repository (only with Git on), and the Command Palette always last, the gateway to everything else. Accent links with no box; the icon sits in a 26 px column so every label starts at the same

x, and the whole row answers a click.

3. **Chord**, right-aligned past a 32 px gap, in the keyboard color, from the active keymap. Unbound: no chord. Open File shows find-file's chord in the Emacs keymap, where that is the key to learn. The hovered row tints and the chord becomes a chip.
4. **Entry number.** The first nine recent entries, projects first, are numbered in the keyboard color; typing that digit opens the entry. Projects come in the order of their most recent file.
5. **Recent file row.** The name keeps its width; a chip names the project (or the host of a remote file); the folder is muted, relative to the project root, with `~` for home, and elides from its start so the part that differs stays visible. The full path is the row's tooltip; right-click offers Copy Path and Remove from recent files.
6. **Disclosure.** Five rows per list; the rest wait behind "Show N more", which becomes "Show fewer".
7. **Customize.** The active keymap and app theme by name; each link runs the picker command. Under an Emacs-style keymap a caption explains the notation. On a first run, with nothing recent, this block moves up under Start.
8. **Help and footer.** Documentation, What's New (the release notes) and the Doctor. Then the version, the home page and the license; a dev build adds its commit on its own line once the lookup, which runs `git` off the FX thread, has answered.

Spec

Property	Value	Reference
Kind	A real tab content in its own closeable "Welcome" tab, so the tab strip handles switching and closing; not an overlay	`WelcomePane implements TabContent`, `MainController.addWelcomeTab`
Column	640 px max, 340 px min, 48 px margin, centered in a scroll pane; below the minimum the page scrolls sideways	`COLUMN_WIDTH`, `MIN_COLUMN_WIDTH`, `MARGIN`
Zoom	Every px size is multiplied by the text zoom and the fonts are `em`, so zoom scales the layout, not only the words	`setFontScale`, `.welcome-*`
Surface	`-color-bg-default`, the UI surface, not the editor's; its tab takes the same fill	`.welcome-pane`, `.ui-surface-tab`
Header	Icon 64 px; title 2.4 em bold; tagline `-color-fg-muted`	`.welcome-title`, `.welcome-caption`
Section heading	1.15 em bold, padding 28 0 8 0	`.welcome-section`
Links	`Hyperlink` in `-color-accent-fg`, no border, underlined only on hover or keyboard focus; visited stays the accent	`.welcome-action`, `.welcome-link`
Action row	Icon column 26 px, gap 12 to the label, at least 32 px to the chord; hover fills `-color-bg-subtle` with 6 px radius, insets 8 px past the row	`WelcomePane.action`, `.welcome-action-row`
Chord	`-state-key`, 0.9 em; on hover a `-state-key-bg` chip, radius 4	`.welcome-key`
Recent row	Number `-state-key` 0.8 em; folder `-color-fg-muted` 0.9 em with a leading ellipsis; project chip muted on `-color-bg-subtle`, radius 3	`.welcome-entry-number`, `.welcome-recent-dir`, `.recent-project-name`
Lists	5 rows, then a flat disclosure button	`COLLAPSED_RECENT_LIMIT`, `addCollapsed`
Focus	The first recent entry, or New File when there is none, with no focus ring; Enter and Space open it	`requestFocus`, `focusPage`

Behavior

Applies to: an app with a Welcome page (WELC-01).

- **WELC-03** `SHOULD` The Welcome page is shown at startup when there is no session to restore (so always with `--no-session`), by the `view.welcome` command (Help → Welcome Page where the app has a Help menu, in the place DOCS-07 gives it), and when a switch to another project leaves the tab strip empty (Editora's project swap). In an app that can start straight into one view, as Editora does for a standalone diff, that startup suppresses the page while its worker reads the files.
  - *Check (code):* those three paths open the one Welcome tab, and where the app has a Help menu, a row inside it runs `view.welcome`; a startup that goes straight to a view holds a flag that skips the page.
  - *Reference:* Editora `MainController.addWelcomeTab`, `showWelcomeIfNoTabs`; `WindowSessionCoordinator` (`suppressWelcome`). Editora's menu row is in View, not in Help yet (`MenuBarModel`). The app template shows the page when there is no session to restore (`ui/DocumentTabs.java`), registers `view.welcome` (`ui/Commands.java`) and lists it in the Help menu (`ui/AppMenus.java`; `SessionAndWelcomeFxTest`, `theWelcomePageIsInHelpClosesAloneAndIsRebuiltWhenShown`); it has no project swap and no view to start straight into.
- **WELC-04** `SHOULD` Closing the last tab does not bring the Welcome page back. The empty content area shows its own quiet hint instead: the app's ways in, each with its chord (in Editora: Open File, Open Folder, Command Palette and Welcome Page), and takes focus so chords still reach the window.
  - *Check (code):* closing the last tab shows the hint in the empty content area, not the Welcome page, and gives it the focus; opening a tab hides it. A test that asserts it is the surest evidence.
  - *Reference:* Editora `ui/EditorEmptyState.java`, `EditorEmptyStateFxTest`; the app template's `ui/DocumentTabs.java` (`emptyHint`, which lists its ways in with their chords and takes focus when the last tab closes).
- **WELC-05** `SHOULD` The page is rebuilt on every show, so the recents, the Projects and Git toggles, the keymap and the theme are current. A refresh keeps the focus where it was.
  - *Check (code):* showing the page calls `refresh`, which re-requests focus when the page had it.

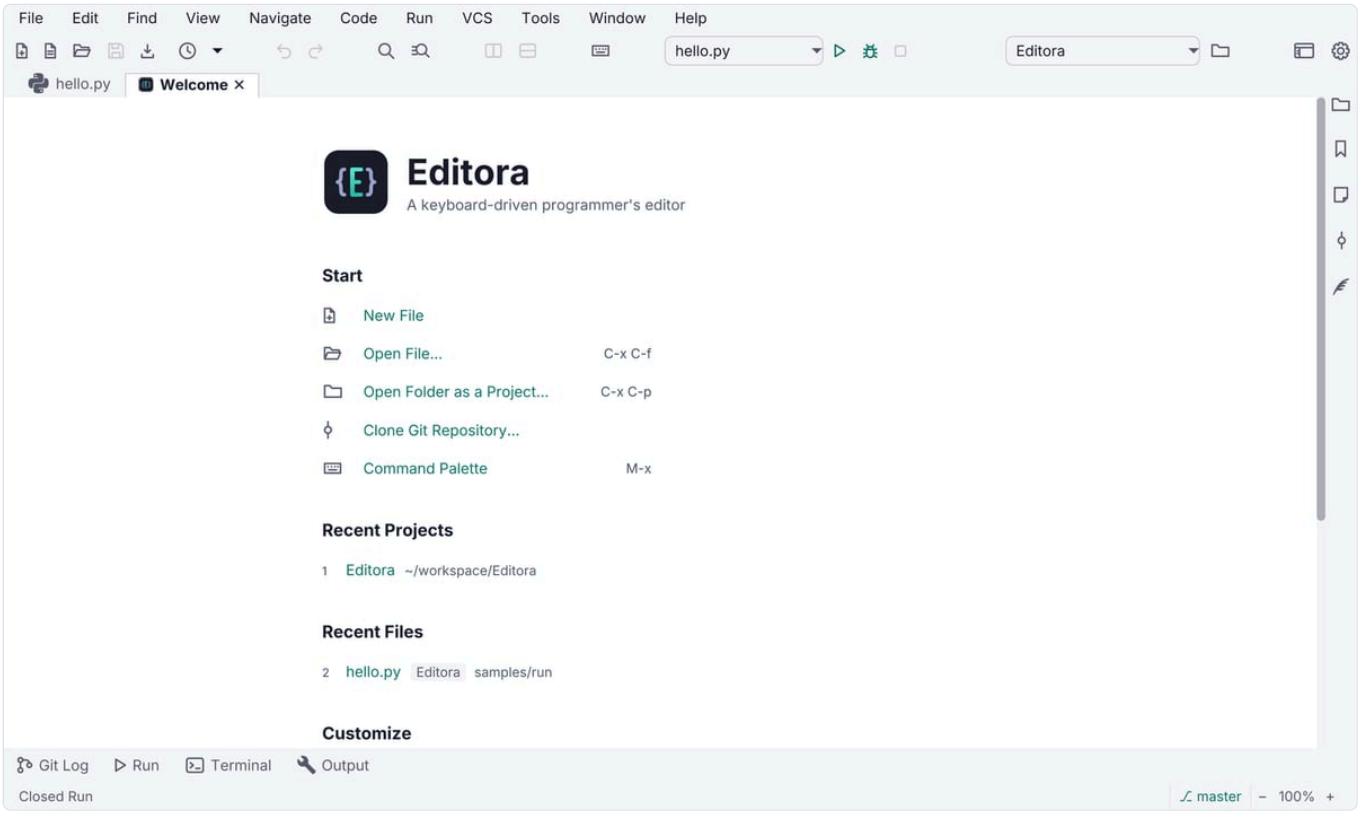
- *Reference:* Editora `WelcomePane.refresh`; the app template's `refreshWelcome` (`ui/MainWindow.java`), run on every show (`SessionAndWelcomeFxTest`, `theWelcomePageIsInHelpClosesAloneAndIsRebuiltWhenShown`).

- **`WELC-06`** `SHOULD` The recent lists offer no file that no longer exists and no project whose root is gone. Remote files are kept unchecked.
  - *Check (code):* the recent files pass through the same filter as the toolbar's dropdown; projects are kept only when their root is a directory.
  - *Reference:* Editora `RecentFiles.showable`, `WelcomePane`.
- **`WELC-07`** `SHOULD` Remove from recent files asks first, through the caller, then refreshes the page.
  - *Check (code):* the remover the page is given shows a confirmation before it removes the entry.
  - *Reference:* Editora `WelcomePane.setRecentRemover`; `MainController` (`dialog.removeRecent.*`).
- **`WELC-08`** `SHOULD` The digits 1 to 9 open the numbered entries: typed characters only, with no modifier, so chords and Enter are untouched.
  - *Check (code):* the digit handler is on `KEY_TYPED` and ignores Ctrl, Alt and Meta.
  - *Reference:* Editora `WelcomePane.onKeyTyped`.
- **`WELC-09`** `SHOULD` In an app with saved remote sites, they appear as their own section; a click opens the connection form prefilled. The section is hidden when there are none.
  - *Check (code):* the page adds a Remote Sites section only when the saved-sites list is not empty.
  - *Reference:* Editora `WelcomePane` (`remoteSites`, `welcome.remote`).
- **`WELC-10`** `SHOULD` The Welcome page's Customize links run the commands that pick the keymap and the app theme (Editora's are `keymap.select` and `theme.setAppTheme`); a change made there re-renders the page, chords included.
  - *Check (code):* the two Customize links run the app's keymap and theme picker commands through the registry, and applying a keymap or a theme refreshes the Welcome page.
  - *Reference:* Editora `WelcomePane` (`customizeRow`), refreshed by `SettingsReapply`.
- **`WELC-11`** `SHOULD` The Welcome tab is not a document: its menu is Close only, with no Save, Rename or Pin, and the title bar shows the app or project name, not a file.
  - *Check (code):* the Welcome tab's context menu holds one Close item.
  - *Reference:* Editora `MainController.addWelcomeTab`; the app template's `ui/DocumentTabs.java`, whose Welcome tab's menu holds `buffer.close` alone, is not pinned, and leaves the title to the project and the app.

Copy

Key	English
`welcome.tab`	Welcome
`welcome.tagline`	A keyboard-driven programmer's editor
`welcome.start`	Start
`welcome.newFile` / `welcome.openFile`	New File / Open File...
`welcome.openFolder` / `welcome.clone`	Open Folder as a Project... / Clone Git Repository...
`welcome.recentProjects` / `welcome.recentFiles` / `welcome.remote`	Recent Projects / Recent Files / Remote Sites
`welcome.noRecent`	No recent files yet. Open a file or a folder to start, or run editora FILE in a terminal.
`welcome.recent.showMore` / `welcome.recent.showFewer`	Show {0} more / Show fewer
`welcome.entryNumber`	Press {0} to open *(accessible help)*
`welcome.customize` / `welcome.keymap` / `welcome.theme`	Customize / Keymap / Theme
`welcome.notation.emacs`	Chords use Emacs notation: C- is Ctrl, M- is Alt, S- is Shift. Pick another keymap above to see your platform's notation.
`welcome.help` / `welcome.documentation` / `welcome.whatsNew` / `welcome.doctor`	Help / Documentation / What's New / Doctor: Check External Tools
`welcome.version`	Version {0}
`welcome.page`	Welcome Page *(the row in the empty editor's hint, WELC-04; the menu row takes the command's title, `command.view.welcome`)*

In the apps



Editora: Start actions with their chords, Recent Projects and Recent Files numbered 1 and 2, Customize below.

Torrenta has no documents to come back to, so its welcome is an **empty state over the table** until the first torrent (`ui/Welcome.java` , bound to the list being empty): a drop zone that says "Drop a .torrent, a magnet or a link anywhere", the four keys that matter with their chords (open a .torrent file, add what is on the clipboard, filter or run a command, Settings), the clipboard's magnet or link offered with an *Add it* button when there is one, and where downloads go with a *change* link. Nothing on it takes focus: the keys are the way in. Mirada, with no folder open, shows an empty state too (`offerToOpen` in `ui/ViewerWindow.java`): a hint that names the Open command's chord ("Open an image with {0}, or drop one here"), buttons for the Open Image... and Open Folder... commands and a Documentation link. Termina, Modula and nux open straight into their content and have neither.

About window

The About window answers "what exactly am I running?" in one place, and its details are what a bug report needs. Torrenta (`AboutDialog`), Termina, Modula, Mirada and nux (`AboutWindow`) follow Editora's order.

Rules

- **`ABOUT-01`** `MUST` Every app has an About window, reached by the `help.about` command, by Help → About where the app has a Help menu (the Help menu of a menu bar, or a Help submenu of the app's main menu, which WIN-14 defines), and on macOS by the About item of the application menu, the one macOS names after the app. The app adds that item itself: JavaFX builds the menu with Hide and Quit and no About.
  - *Why:* on macOS that menu is where every app keeps About, so it is one of the places the OS's way wins.
  - *Check (code):* `help.about` is registered and opens the About window; where the app has a Help menu, a row inside that menu runs the command (a row beside a Help submenu is not in it); and on macOS the app inserts an About item into the application menu that opens the same window.
  - *Reference:* Editora `ui/AboutDialog.java` and `ui/MenuBarModel.java` (`help.about` ends the Help menu), and the app template's `ui/Commands.java` and `ui/AppMenus.java` , do the Help row and the command, and the template's `ui/MacAppMenu.java` inserts the macOS item, which runs `help.about` , so the template meets the whole rule. Editora doesn't add the macOS item yet, and nor do Torrenta, Modula and nux. Termina (`ui/MacAppMenu.java`) and Mirada (`os/MacAppMenu.java`) add it too: JavaFX has no public API for that menu, so the three insert the item into the one Glass keeps (`MacApplication.getAppleMenu`). Neither of those two is the whole rule: Termina's command is `about` , not `help.about` , and Mirada's About row sits beside the Help submenu of its main menu, not in it (`ui/TitleBar.java` , `finish`).
- **`ABOUT-02`** `MUST` About shows the version and build, the runtime, the license, links to the app's home page, its source code and its releases, and the result of the last update check. The Pauta edition is ABOUT-05.
  - *Check (code):* the About window shows the version and the build as the build recorded them (in Editora both come from `AppInfo` , which `--version` prints from too); the Java and JavaFX versions; the license; a link that opens the app's home page (its website, or its repository's page on the host where About links no website), one that opens its source code (the repository) and one that opens its releases (the repository's releases page or its latest release), each there whatever the update check answered; and the update checker's last answer whatever it was (an update, up to date, a failed check or not checked yet), not only an update that is available. One link meets two of the three when the page it opens is both: the repository's page is the home page and the source code where About links no website, and a releases page of the repository holds its source too (Editora's "Source code and releases"). A link to the running build's release notes, and a link shown only when an update is available, are not the link to the releases.
  - *Reference:* Editora `ui/AboutDialog.java` , `AppInfo` (`HOMEPAGE` , its website, and `RELEASES_PAGE` , the repository's latest release, behind "Source code and releases"); the page's Spec. Four apps lack a link. Termina's About (`ui/AboutWindow.java`) links its website and not its source code, and its releases only when an update is available. Mirada's (`ui/AboutWindow.java`) links its repository and the running build's release notes, and its releases only when an update is available. Modula's About (`ui/AboutWindow.java`) links its repository and never its releases; it also shows no Java version, and it shows the update check only when an update is available. nux's (`ui/AboutWindow.java`) has the runtime lines and links its repository; it shows the update check, and with it the link to its releases, only when there is a newer version.
- **`ABOUT-03`** `SHOULD` About's *Copy details* button copies the environment block and leaves the window open; its label becomes "Copied".
  - *Check (code):* the button's handler puts the environment block's text on the clipboard and sets the button's label to the copied text (Editora's `about.copied`); the button sits in the content, not the dialog's button bar, so pressing it can't close the window.
  - *Reference:* Editora `AboutDialog` (`about.copyDetails` , `about.copied`).
- **`ABOUT-04`** `SHOULD` About's update status claims nothing before a check has answered: with no answer, or with the check switched off, it says it has not checked, never "up to date".
  - *Check (code):* About's update status falls back to the not-checked text when the checker has no result.
  - *Reference:* Editora `AboutDialog` (`about.notChecked`); the Update status states.
- **`ABOUT-10`** `SHOULD` In About, an available update stands out: "Update available" is bold in amber, a color no link or label in About uses, while the links stay in the accent, "up to date" is green, a failed check red and the last-checked line muted (Editora's tokens for the four are `-state-amber` , `-state-accent` , `-state-green` and `-state-red`).
  - *Why:* in the accent every link wears, the one line that asks for action reads as one more link.
  - *Check (code):* the style of About's update-available line (the app template's `.about-update-available`) is bold in the amber token, and no other About style uses that token; the up-to-date line is in the green token, a failed check in the red and the last-checked line in the muted color, and the links keep the accent.
  - *Reference:* the app template's `.about-update-available` in `styles/app.css` , set by `ui/AboutDialog.java` . No released app does yet: Editora's `.about-update` in `app.css` is `-state-accent` , the links' color.

Anatomy



1. **Identity.** The app icon at 72 px beside the name (26 px, bold). No alert header band, no info graphic: the panel reads as one block, like a native About panel.
2. **Version and tagline.** Version 14 px muted; a snapshot build appends its Git branch, "0.28.0-SNAPSHOT (feat/map)". The tagline is the README's one-line pitch, 13 px muted.
3. **Update status.** What the last update check found (see the states below), when it ran, and *Check now*. Nothing is claimed before a check has answered.
4. **Copyright**, muted.
5. **Links.** License, homepage, source code and releases, the Pauta edition the app is built to and its conformance, which opens the Pauta window (see below), and the settings file, which opens in the app (or in the file manager for apps with no editor).
6. **Environment.** Monospace 11.5 px, muted: version and build time, commit, Pauta edition and conformance, Java and JavaFX versions with vendor, OS with architecture, config folder. Home folders shown as `~`.
7. **Copy details** copies the environment block and stays open; its label becomes "Copied". It sits in the content, not the button bar, so pressing it does not close the panel.
8. **Close**, the one button, and the default: Enter and Esc both close.

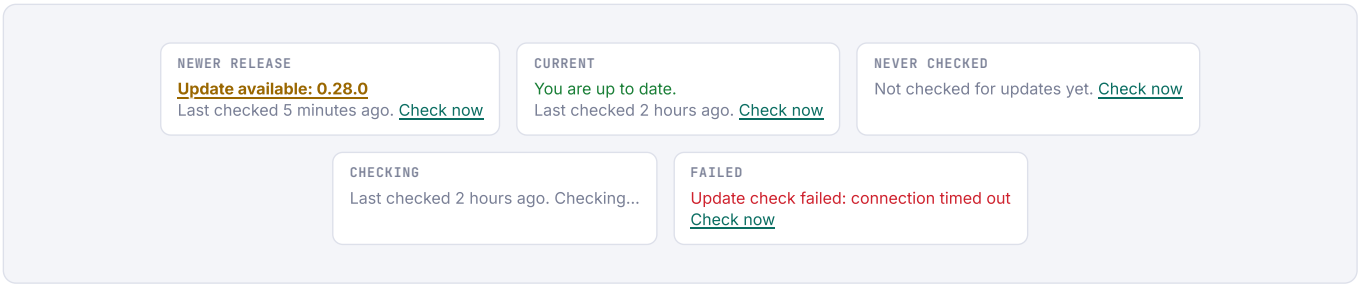
Pauta edition and conformance

About names the edition of these guidelines the app implements and how fully it conforms (Conformance), so a user, a bug report and a reviewer all know which rules the app answers to. The app template shows both; Termina's and Mirada's About name their edition but not their conformance.

- **`ABOUT-05`** `MUST` About names the Pauta edition in two places: a link, *Built to Pauta x.y.z*, among its other links (at Partial it reads *Working toward Pauta x.y.z*, ABOUT-09), and a line `Pauta: x.y.z` in the environment block, right after the commit, so *Copy details* carries it.
  - *Check (code):* one of About's links reads *Built to Pauta* and the edition, or *Working toward Pauta* and the edition at Partial, in words from the catalog, and About's environment block writes a `Pauta:` line right after the commit line. The rule asks for the words, not for a catalog key: the app template's and Termina's key is `about.pauta`, Mirada's `about.pautaLink`.
  - *Reference:* The app template's `ui/AboutDialog.java` and `AppInfo.java`; Termina's `ui/AboutWindow.java` and `AppInfo.java`; Mirada's `ui/AboutWindow.java` (`about.pautaLink`) and `AppInfo.java`.
- **`ABOUT-06`** `MUST` About's edition link (ABOUT-05) opens the edition's own address, `https://pauta-project.dev/<edition>/`, which never changes, not `latest/`, which moves on to newer editions.
  - *Check (code):* the URL About's edition link opens is built from the edition number (`pauta.version`) and never contains `latest`. An About with no edition link has nothing to open: the rule fails with ABOUT-05, which requires the link, and the evidence names that rule.
  - *Reference:* The app template's `AppInfo.java` (`pautaUrl`); Termina's `AppInfo.java` (`pautaUrl`); Mirada's `AppInfo.java` (`pautaUrl`).
- **`ABOUT-07`** `MUST` The Pauta edition About names is the edition the app meets, not the newest. Moving to a newer edition means reading its changelog for new or stronger MUSTs, meeting them, then changing the number. An app at Partial names the edition it is working toward instead, and says so (ABOUT-09).
  - *Check (code):* read which words About uses for the edition (ABOUT-09). *Working toward* meets the rule. An About that names no edition claims none: this rule has nothing to judge and passes, and the missing edition is ABOUT-05's finding. *Built to* is judged last, from the audit's other results: audited against the edition the app names

(`pauta.version`), with that edition's own rules (`https://pauta-project.dev/<edition>/rules.json`), not the newest edition's, the app has no `MUST` that is `fail` or `pending` . The evidence names the MUSTs that are. An audit of part of the edition judges the rule from the MUSTs in its scope and says so. Only the rules of the edition the app names settle a *Built to*: an audit run against another edition reports the rule as `not-checked` and names the edition whose rules would settle it.
    - *Reference:* The app template, at Complete: its About says *Built to Pauta 6.0.0* (`ui/AboutDialog.java` , `editionKey`) and so does its README, and its audit against 6.0.0's own rules finds no `MUST` that is `fail` or `pending` (`docs/pauta-audit.md`). Termina and Mirada don't meet it yet. Termina's About says "Built to Pauta 2.0.2" (`pom.xml` , `pauta.version`), an edition that already required a find bar in its Debug Log, the UI font on every scene and the command ids `help.about` and `palette.show` , none of which it has. Mirada's says "Built to Pauta 4.0.1", an edition that already required its settings in `~/.mirada` on every OS.
- **`ABOUT-08`** `MUST` The Pauta edition an app names has one source: a `pauta.version` property in `pom.xml` , which the build filters into the app's resources and the app reads from there, like its own version (`build-info.properties` and `AppInfo` in the app template). The README's badge and "Built to" line, which name that edition too, are PAUTA-15.
    - *Check (code):* `pom.xml` defines `pauta.version` ; a resource the build filters carries it (the app template's `build-info.properties`); and the edition the app shows is read from that resource (the template's `AppInfo`), never from a literal in its code.
    - *Reference:* The app template's `pom.xml` , `build-info.properties` and `AppInfo.java` ; Termina's the same, with `AppInfo.java` , and Mirada's, with its `pom.xml` and `AppInfo.java` .
- **`ABOUT-09`** `MUST` About shows the app's conformance, Complete, Core or Partial, as the conformance helper computes it from `pauta.json` (PAUTA-02) and never higher: a link, *Pauta conformance: Core*, right after the edition's link, that runs the `help.pauta` command and so opens the Pauta window (PWIN-01), and a line `Pauta conformance: Core (2 exceptions)` in the environment block, right after the `Pauta:` line; with one it says `(1 exception)` , with none `(no exceptions)` . At Partial the edition's link reads *Working toward Pauta x.y.z*: the app does not meet that edition yet, so it does not say it is built to it.
    - *Why: Never claim what is not true.*
    - *Check (code):* About calls the conformance helper and holds no conformance name of its own; its conformance link runs `help.pauta` ; the environment block has the conformance line after the `Pauta:` line; and at Partial the edition's link reads *Working toward Pauta* and the edition (the app template's key for it is `about.pautaWorkingToward`).
    - *Reference:* The app template's `ui/AboutDialog.java` and `PautaConformance.java` (`environmentLine`). Termina's and Mirada's About show the edition but not the conformance.

Update status states



"Update available" links to that release's page. An update check that is switched off must never read as "up to date": with no answer, the panel says it has not checked.

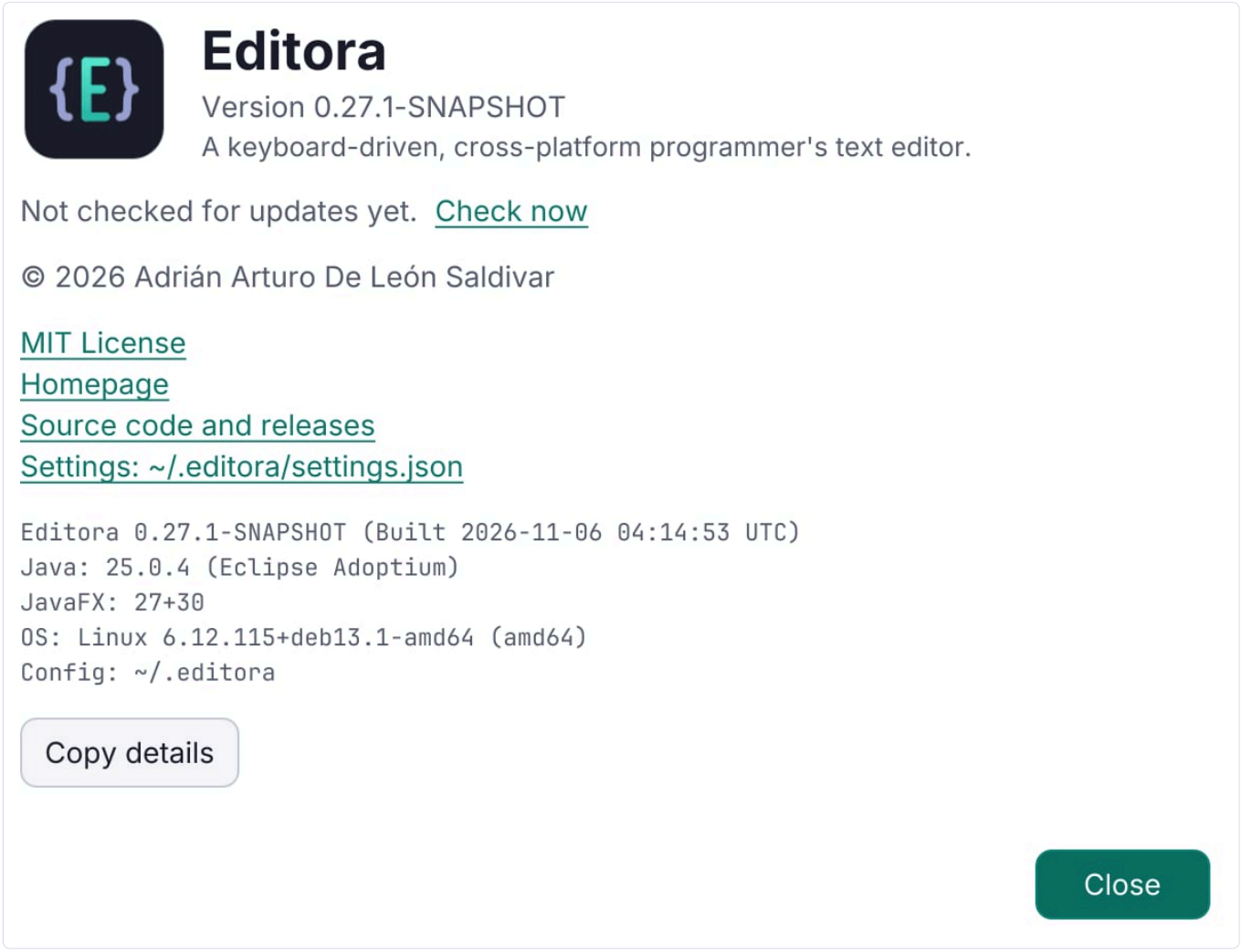
Spec

Property	Value	Reference
Kind	A modal dialog owned by the window, styled through `Dialogs.styled` so it gets `app.css` and the UI font	`AboutDialog.show`
Layout	One column, 14 px between blocks; content padding 6 8 2 8	`.about-content`
Icon	The app icon PNG at 72 px	`icon-128.png`
Name / version / tagline	26 px bold / 14 px muted / 13 px muted	`.about-name` , `.about-version` , `.about-tagline`
Update colors	Available: amber, bold (ABOUT-10). Up to date: green. Failed: red. Last checked: muted	`.about-update*`
Environment	JetBrains Mono 11.5 px, muted	`.about-env`
Source of the version	Maven-filtered `build-info.properties` , read by `AppInfo` : the same text as `--version` and a problem report	`AppInfo`
Pauta edition	`pauta.version` in `pom.xml` , filtered into `build-info.properties` ; the link opens `https://pauta-project.dev/<edition>/`	the app template's, Termina's and Mirada's `AppInfo.pautaUrl`
Pauta conformance	Computed from `pauta.json` by the conformance helper; the link opens the Pauta window	Conformance

Copy

Key	English
`dialog.about.title`	About {0}
`about.version`	Version {0}
`about.updateAvailable`	Update available:
`about.upToDate`	You are up to date.
`about.lastChecked`	Last checked {0}.
`about.notChecked`	Not checked for updates yet.
`about.checkNow` / `about.checking`	Check now / Checking…
`about.homepage` / `about.releases`	Homepage / Source code and releases
`about.pauta` / `about.pautaWorkingToward`	Built to Pauta {0} / Working toward Pauta {0}
`about.pautaConformance`	Pauta conformance: {0}
`about.copyDetails` / `about.copied`	Copy details / Copied

In the apps



Editora's About before its first update check: identity, links and the environment block, with no Pauta link or Pauta line yet, which ABOUT-05 asks for.

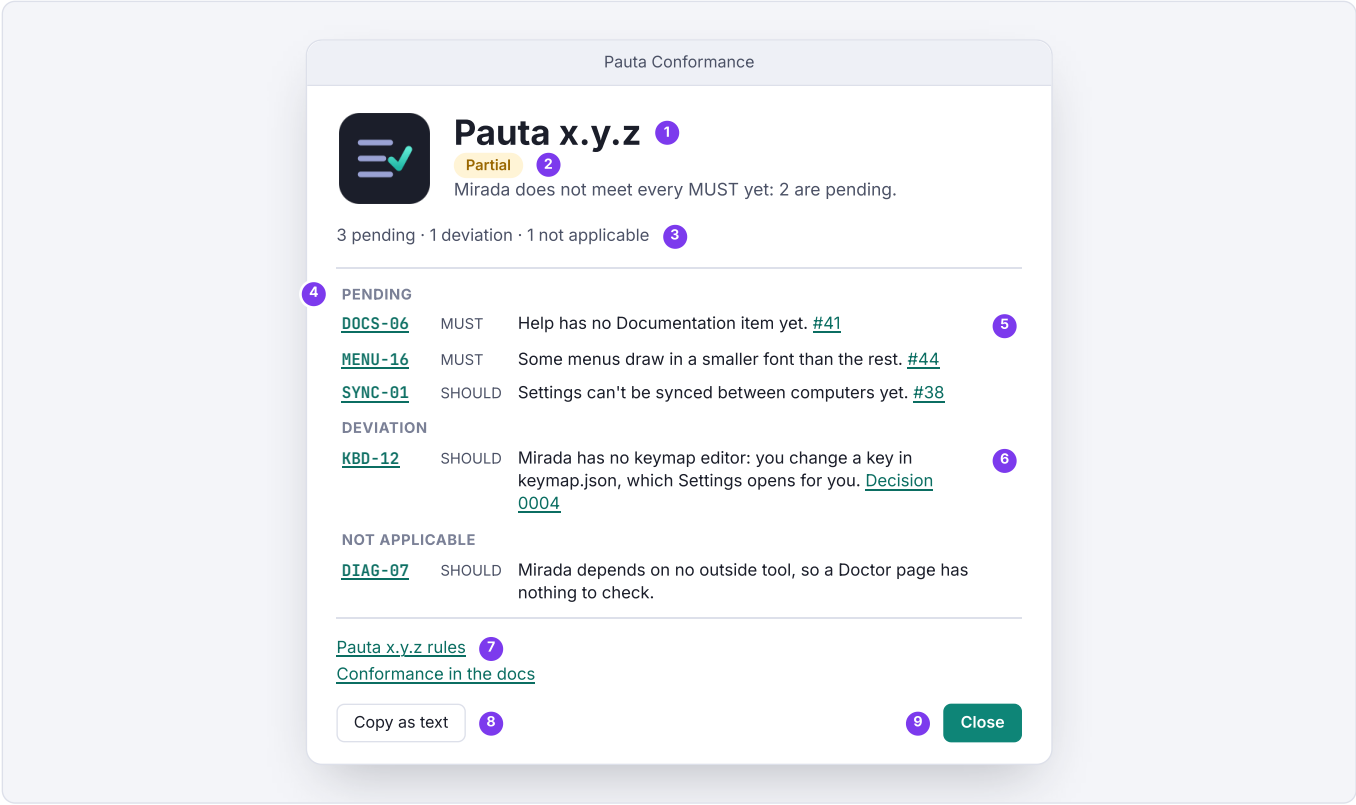
Pauta window

About says which edition of Pauta the app is built to and how fully it conforms; the Pauta window shows the evidence. It lists every exception the app records in its `pauta.json` (see Conformance): the rules that do not apply, the ones it does differently on purpose, and the ones still pending. It is built like the About window: one modal panel, read-only, with a copyable summary. No released app has one yet; the app template's `ui/PautaWindow.java` is the one to copy.

Rules

- **PWIN-01** `MUST` Every app has a Pauta window, opened by the `help.pauta` command: from the Command Palette like every command, and from a *Pauta Conformance* row in the Help menu where the app has a Help menu (the Help menu of a menu bar, or a Help submenu of the app's main menu, which WIN-14 defines). The row's place in that menu is DOCS-07, and About's *Pauta conformance* link, which runs the same command, is ABOUT-09.
  - *Check (code):* a command with the id `help.pauta` is registered and opens the Pauta window, and where the app has a Help menu, a row inside that menu runs the command (the app template's Help menu is a list of command ids that holds `help.pauta`). A row beside a Help submenu is not in it.
  - *Reference:* The app template's `ui/Commands.java` (`help.pauta`, which runs `openPauta`) and `ui/AppMenus.java` (`help.pauta` in `menu.help`, just above `help.about`). No released app has a Pauta window yet.
- **PWIN-02** `MUST` The Pauta window shows the edition, the conformance with one sentence that says what it means, a count of exceptions by kind, and every exception: the rule's ID, its level (`MUST` or `SHOULD`), the kind, the reason, and for a deviation or a pending exception its link.
  - *Check (by hand):* open the Pauta window and compare it with the app's `pauta.json`, entry by entry.
  - *Reference:* The app template's `ui/PautaWindow.java`.
- **PWIN-03** `MUST` The Pauta window reads the exceptions from `pauta.json` and the edition from the build, as About does (ABOUT-08; `AppInfo` in the template), and takes the conformance from the same helper as About (PAUTA-02). No edition, conformance or exception is written in the window's own code.
  - *Check (code):* the Pauta window loads the exceptions through the one loader of `pauta.json`, reads the edition where About reads it, and calls the conformance helper; its code holds no rule ID, edition number or conformance name of its own.
  - *Reference:* The app template's `ui/PautaWindow.java` (`PautaConformance.of`).
- **PWIN-04** `MUST` Every link in the Pauta window opens a fixed address, never one under `latest/`: the edition's link opens the edition's own address, the one ABOUT-06 gives About's link, and each rule ID opens the rule's address in that edition (the entry's `url` in `pauta.json`). The edition's link, in the Pauta window and in About, runs the `help.pautaRules` command, so the palette reaches it too.
  - *Why:* an app built to an edition answers to that edition's rules; `latest/` moves on.
  - *Check (code):* `help.pautaRules` is registered and opens the edition's address, built from `pauta.version`; the edition's link in the Pauta window and the one in About both run that command; and each rule ID in the Pauta window opens its entry's `url`.
  - *Reference:* The app template's `ui/Commands.java` (`help.pautaRules`, which opens `AppInfo.pautaUrl()`), run by the edition's link in `ui/PautaWindow.java` and `ui/AboutDialog.java`; `PautaWindow` opens each rule ID at its entry's `url`.
- **PWIN-05** `SHOULD` The Pauta window lists pending exceptions first, then deviations, then app-only (in a template), then not-applicable; within a kind, MUSTs before SHOULDs, then by ID.
  - *Why:* what the app still owes is what a reader came to find.
  - *Check (code):* the list the Pauta window draws is sorted by kind in that order, then MUSTs before SHOULDs, then by rule ID (the app template's `PautaConformance.sorted`). A test that asserts it is the surest evidence.
  - *Reference:* The app template's `PautaConformance.java` (`sorted`, by the order of its `Kind`), tested by `PautaConformanceTest` (`pendingFirstThenDeviationsThenAppOnlyThenNotApplicableMustsBeforeShouldsThenById`).
- **PWIN-06** `SHOULD` The Pauta window's *Copy as text* button copies the edition, the conformance and the list as Markdown, one line per exception, and leaves the window open; its label becomes "Copied".
  - *Check (code):* the button's handler copies the edition, the conformance and one line per exception as Markdown and sets the button's label to the copied text (the template's `pauta.copied`); the button sits in the content, not the button bar, like About's *Copy details*, so pressing it can't close the window.
  - *Reference:* the app template's `ui/PautaWindow.java` (`pauta.copy`, `pauta.copied`), built like Editora's About *Copy details*.
- **PWIN-07** `SHOULD` A *Conformance in the docs* link runs the `help.pautaDocs` command, which opens the app's docs page (PAUTA-13) for this version, `/docs/v–<version>/pauta/`, where the site serves versioned docs (DOCS-03), and `/docs/pauta/` until it does. The README's badge links the unversioned page, which always shows the latest release.
  - *Check (code):* `help.pautaDocs` is registered and builds the URL from the app's docs base, like Help → Documentation.
  - *Reference:* the app template's `AppInfo.pautaDocsUrl()`, registered as `help.pautaDocs` in `ui/Commands.java`.
- **PWIN-08** `SHOULD` With no exceptions, the Pauta window says so in one line where the list would be, naming the app and the edition (for an app called Mirada, "No exceptions: Mirada meets every `MUST` and `SHOULD` of Pauta x.y.z."), and its copy button still works.
  - *Check (by hand):* with an empty `exceptions` array in `pauta.json`, open the Pauta window.
  - *Reference:* The app template's `ui/PautaWindow.java` (`pauta.none`).

Anatomy



The entries above show the layout with Mirada's name; they are not Mirada's record.

1. **Edition.** The Pauta mark at 72 px beside "Pauta" and the edition, in About's name style.
2. **Conformance.** A pill: Complete green, Core accent, Partial amber. Below it, one sentence in the tagline style that says what it means for this app.
3. **Counts.** How many exceptions of each kind, muted. A kind with none is left out.
4. **Group header.** One per kind present, in this order: Pending, Deviation, For each app made from it (the app-only entries of a template), Not applicable. The list scrolls past about ten rows; the rest of the window stays put.
5. **Exception row.** The rule's ID in monospace, linked to the rule in this edition; its level, `MUST` or `SHOULD`, muted; the reason; for a pending exception, the issue number as a link.
6. **Decision link.** A deviation links its decision record by number.
7. **Links.** The edition's rules at its own address, and the app's conformance page in its docs.
8. **Copy as text** copies the summary as Markdown and stays open; its label becomes "Copied".
9. **Close**, the one button and the default: Enter and Esc both close.

Spec

Property	Value	Reference
Kind	A modal dialog owned by the window, styled through `Dialogs.styled` like About	`AboutDialog.show`
Width	560 px; the list scrolls at about ten rows	`.pauta-list`
Head	The Pauta mark PNG at 72 px; "Pauta" and the edition 26 px bold	`.about-name`
Conformance pill	Complete `-state-green` on `-state-green-bg`, Core `-state-accent` on `-state-accent-bg`, Partial `-state-amber` on `-state-amber-bg`	`.pauta-conformance`
Rule ID	JetBrains Mono 12 px, accent, a link	`.pauta-rule`
Source	`pauta.json` in the app's resources and `pauta.version` through `AppInfo`	Conformance
Conformance	The conformance helper, never stored	PAUTA-02

Copy

Key	English
`command.help.pauta`	Pauta Conformance
`command.help.pauta.desc`	Show which edition of Pauta this app implements, how fully it conforms, and the rules it does not meet.
`command.help.pautaRules` / `.desc`	Pauta Rules / Open the rules of the Pauta edition this app implements.
`command.help.pautaDocs` / `.desc`	Pauta Conformance in the Docs / Open the user guide's page on this app's Pauta conformance and exceptions.
`dialog.pauta.title`	Pauta Conformance
`pauta.edition`	Pauta {0}
`pauta.conformance.complete` / `.core` / `.partial`	Complete / Core / Partial
`pauta.conformance.complete.desc`	{0} meets every `MUST` and `SHOULD` of this edition.
`pauta.conformance.core.desc`	{0} meets every `MUST` of this edition; {1,choice,1#1 `SHOULD` is|1<{1} SHOULDs are} pending.
`pauta.conformance.partial.desc`	{0} does not meet every `MUST` yet: {1,choice,1#1 is|1<{1} are} pending.
`pauta.kind.pending` / `.deviation` / `.appOnly` / `.notApplicable`	Pending / Deviation / For each app made from it / Not applicable
`pauta.count.pending`	{0} pending
`pauta.count.deviation`	{0,choice,1#1 deviation|1<{0} deviations}
`pauta.count.appOnly`	{0} for each app
`pauta.count.notApplicable`	{0} not applicable
`pauta.decision` / `pauta.link`	Decision {0} / Details
`pauta.rules` / `pauta.docs`	Pauta {0} rules / Conformance in the docs
`pauta.none`	No exceptions: {0} meets every `MUST` and `SHOULD` of Pauta {1}.
`pauta.copy` / `pauta.copied`	Copy as text / Copied

The reasons themselves come from `pauta.json`, in English (PAUTA-08); the window's own words are in the catalogs like every other string. Counts use `MessageFormat`'s `choice` so English reads "1 `SHOULD` is pending", not "1 SHOULDs are". A command description takes no arguments, so it says "this app". The app template's catalog has every key, in each of its six languages.

Confirm dialog

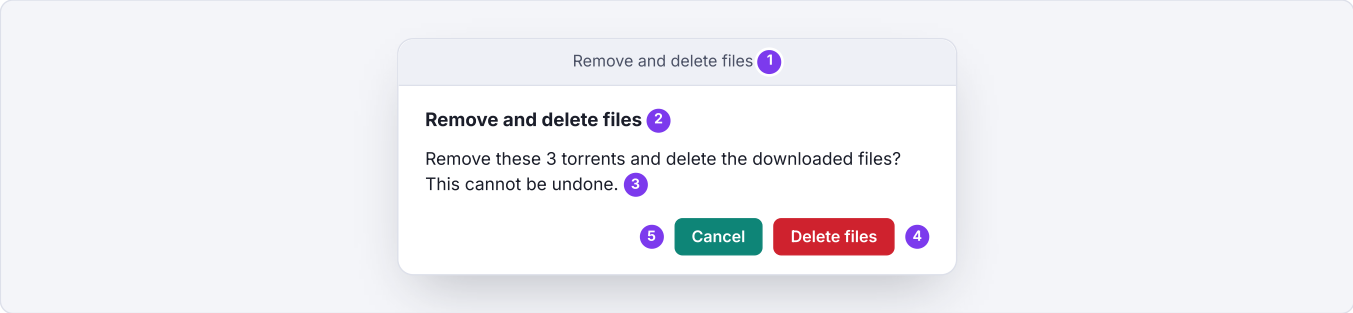
One question before anything the user made is removed, deleted, discarded or overwritten, asked the same way everywhere: the text names the thing and what happens to it, the confirming button is the action's own verb in red, and Cancel is the way out and the default button. An action that can be taken back needs no question (CONF-01, CONF-07). One rule here reaches past the question before a destructive action: CONF-14, on where the buttons' words come from, covers every dialog with two or more buttons.

Rules

- **CONF-01** `MUST` Before anything the user made is removed, deleted, discarded or overwritten, the app asks first, unless the action can be taken back instead.
  - *Check (code):* each destructive action calls a confirmation before it changes anything, or records an undo.
  - *Reference:* Editora `ui/CloseCoordinator.java` (unsaved changes, Quit) and `GitCoordinator.confirmDestructive`; Torrenta `ui/RemoveConfirm.java`, which asks it for the torrent list, the RSS tab and every list in Settings. Modula and nux don't yet: the Remove row of a Modula preset's menu (`ui/PresetBar.java`, `removePreset` in `ui/RadioPane.java`) and nux's Profiles → Delete (`withProfileRemoved`, called from `App.java`) remove at once, with no question and no undo.
- **CONF-02** `MUST` In an app that asks before a destructive action (before it removes, deletes, discards or overwrites something, ends a running program, or runs a command that cannot be taken back), the question names the thing and what happens to it. A question that warns before something is run or pasted and destroys nothing (Termina's before a multi-line paste), and a question before a harmless command (nux's before a UPS command it does not count as risky), are not questions before a destructive action. This list says which questions the rules on a question's form cover; it is wider than what CONF-01 requires a question or an undo for, which is only what the user made.
  - *Check (code):* the string of every question before a destructive action names the thing, by an argument for its name or count or in its own words ("Quit Editora?"), and says what happens to it.
  - *Reference:* Editora `dialog.unsaved.header`, `dialog.discard.tracked`; Torrenta `RemoveConfirm.question`, `Actions.removeQuestion`.
- **CONF-03** `MUST` In an app that asks before a destructive action, the question's confirming button carries the action's own verb (Delete files, Remove, Discard, Quit), never OK or Yes, and is drawn in red, the danger style (Editora's style class for it is `danger`).
  - *Check (code):* in every question before a destructive action (CONF-02 says which those are), the confirming button's `ButtonType` is named with the action's verb, never OK or Yes, and its button is given the red danger style (the `danger` style class in Editora). Where the buttons' words come from is CONF-14's.
  - *Reference:* Torrenta `RemoveConfirm.ask` meets it: the action is named with its verb and styled `danger`. Four other questions of Torrenta's, in `ui/SettingsWindow.java` (disconnecting sync, Reset to Defaults, Reset All Shortcuts and taking a shortcut back from another command), don't yet: the confirming button says OK and is not red. Its question before connecting sync is not one of them: it destroys nothing (what the repository holds comes to this machine, and a value that differs here is kept), like Editora's (`ui/SettingsSyncPage.java`). Termina's `ui/Confirm.java`, Mirada's `ui/Confirm.java` and the app template's `ui/Confirm.java` meet it too: the action is named with its verb and styled `danger`. Editora meets it in some questions and not in others. `GitCoordinator.confirmDestructive`, which every destructive Git action asks through, `MainController.styleUnsavedChangesButtons` and `styleQuitButtonAsDanger` name and style the action, and so do its Local History purges (`ui/HistoryCoordinator.java`). It doesn't yet where the answer is the stock OK, as in these among others: the Project panel's file delete (`confirmDelete` in `ui/ProjectPanel.java`), deleting a project (`ui/MainController.java`), deleting a note (`ui/NotesCoordinator.java`), deleting a macro (`ui/MacroSettingsPane.java`), Replace in Files (`ui/SearchCoordinator.java`), and Revert and overwriting a file from Save As (`confirmRevertDialog` and `confirmOverwrite` in `ui/FileWorkflowCoordinator.java`); nor where the action is named and not red, as in: deleting a remote site (`ui/RemoteSitesPane.java`), deleting a TODO pattern (`ui/TodoSettingsPane.java`), resetting the settings (`ui/SettingsResetFlow.java`), discarding recovery records (`ui/RecoveryCoordinator.java`) and removing a plugin (`ui/PluginDialogs.java`). nux's risky-command question names Run, with no red (`ui/CommandConfirmation.java`).
- **CONF-14** `MUST` Every dialog with two or more buttons words each button from the message catalog, never with JavaFX's stock OK, Yes, No, Cancel, Apply or Close, which are worded in the JVM's locale. That is every JavaFX `Dialog` the app shows, whatever it is for: a question, before a destructive action (CONF-02) or not, a text prompt, a pick from a list, a form. Only a message with one button is outside the rule. Like CONF-08 and CONF-12, it reaches past the question before a destructive action.
  - *Why:* a stock button is labeled from JavaFX's own resources, in the JVM's locale, so under menus in the language chosen in Settings a dialog can still say "Cancel" in another.
  - *Check (code):* in every `Dialog` the app shows with two or more buttons (an `Alert` of any type, a `TextInputDialog`, a `ChoiceDialog`, a `Dialog` the app fills itself), each button shows a string from the message catalog: its `ButtonType` is built with one, or its button's text is set from the catalog before the dialog is shown. None is left as `ButtonType.OK`, `ButtonType.YES`, `ButtonType.NO`, `ButtonType.CANCEL`, `ButtonType.APPLY` or `ButtonType.CLOSE`, and none is the stock OK or Cancel that an `Alert` of type `CONFIRMATION`, a `TextInputDialog` and a `ChoiceDialog` come with. A dialog whose stock buttons are swapped for catalog-worded ones before it is shown passes (Torrenta's `StockButtons.translate`). A dialog with one button is not judged. An app that shows no dialog with two or more buttons has nothing to judge, and passes.
  - *Reference:* Torrenta: `RemoveConfirm.ask` names the action from the catalog and takes Cancel from its `StockButtons` (`cancel()`, the catalog's word on the `CANCEL_CLOSE` role), and every other dialog of two buttons goes through `StockButtons.translate`, which swaps the stock OK and Cancel for the catalog's: the four alerts that ask in its Settings (`ui/SettingsWindow.java`: connecting and disconnecting sync, Reset to Defaults and the shortcuts), its nine text prompts and its two picks from a list (`ui/MainWindow.java`, `ui/DetailPane.java` and `ui/SettingsWindow.java`). Modula's one dialog is a message with one button, so it has nothing to judge. The app template meets it: every dialog it shows goes through `Dialogs.styled`, which words the stock OK, Cancel, Yes, No, Apply and Close from the catalog before the dialog is shown and again when its buttons change, and its `ui/Confirm.java` names the action from the catalog (`DialogsFxTest`, `everyDialogHasOneButtonOrderAndCatalogWords`). Termina and Mirada meet it in their questions and not in their text prompts. Termina's `ui/Confirm.java`, which its removals, its close-running-program question and its multi-line paste warning ask through, names both buttons from the catalog, and so does its Import configuration question (`ui/ConfigTransfer.java`); its Rename Tab prompt words OK from the catalog and keeps the stock Cancel (`askForTabName` in `ui/TerminalWindow.java`), and the prompt its setting commands ask a value through (`settingHost` in the same file) and its Create on GitHub prompt (`createSyncRepositoryOnGitHub` in `ui/SettingsWindow.java`) are `TextInputDialog`s with the stock OK and Cancel. Mirada's `ui/Confirm.java` and its note form (`ui/NoteDialog.java`) name both buttons from the catalog; its Rename prompt (`rename` in `ui/ViewerWindow.java`) is a `TextInputDialog` with the stock OK and Cancel. Editora doesn't yet. Its unsaved-changes and Quit questions name every button from the catalog (`unsavedAlert` and `confirmQuit` in `ui/CloseCoordinator.java`), but of its 64 `CONFIRMATION` alerts, 26 keep the stock OK and Cancel, the Project panel's file delete (`confirmDelete` in `ui/ProjectPanel.java`) and Replace in Files (`ui/SearchCoordinator.java`) among them, and 22 more name the action beside a stock `ButtonType.CANCEL`, among them `GitCoordinator.confirmDestructive`, which every destructive Git action asks through. Its warnings that ask do the same: the four trust questions of `ui/TrustPrompts.java` name the action beside a stock Cancel, and the question before its MCP server is enabled (`confirmEnableMcp` in `ui/MainController.java`) keeps the stock OK and Cancel. So do its prompts and forms, these among them: the Create on GitHub prompt (`ui/SettingsSyncPage.java`), the reset-mode and revert-merge picks (`ui/GitWindowCoordinator.java`), the pick of a settings backup (`ui/SettingsResetFlow.java`), a macro's command picker (`ui/MacroSettingsPane.java`) and the Java cleanups form (`ui/JavaCleanupsDialog.java`) keep the stock OK and Cancel, and the login form (`ui/RemoteCoordinator.java`) and the forms that export and import the configuration (`ui/ExportConfigDialog.java`, `ui/ConfigImportFlow.java`) name the action beside a stock `ButtonType.CANCEL`. nux doesn't in any of its four dialogs of two buttons: the question before a risky command names Run beside a stock `ButtonType.CANCEL`, the one before a harmless command is a `CONFIRMATION` alert with the stock OK and Cancel (`ui/CommandConfirmation.java`), its credentials form adds `ButtonType.OK` and `ButtonType.CANCEL` (`ui/CredentialsDialog.java`), and its Save Profile prompt is a `TextInputDialog` with the stock pair (`saveCurrentAsProfile` in `App.java`).

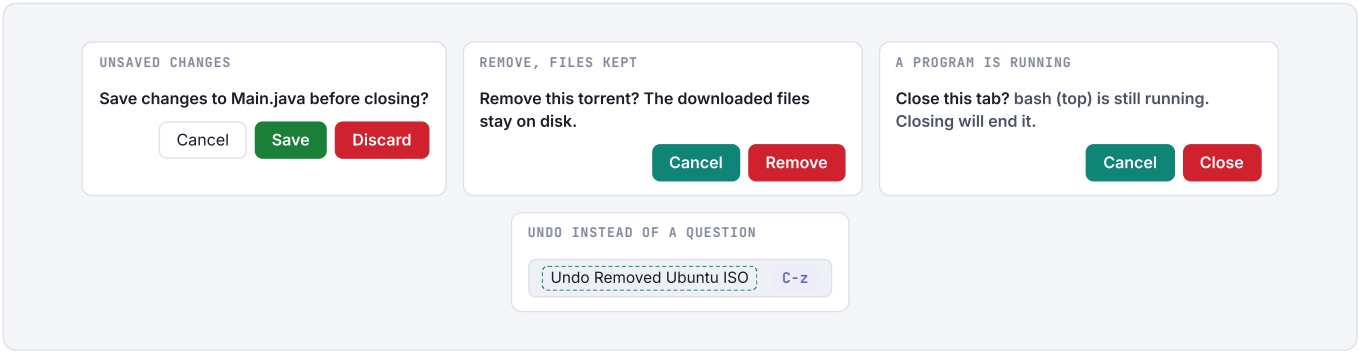
- **CONF-04** `MUST` In an app that asks before a destructive action, Cancel is the way out of the question: Escape and the window's close button both answer Cancel, and nothing is lost on a dismissal.
  - *Check (code):* in every question before a destructive action (CONF-02 says which those are), Cancel has the `CANCEL_CLOSE` role, and the result defaults to Cancel when the dialog closes any other way (`showAndWait().orElse(cancel)`).
  - *Reference:* Editora `GitCoordinator.confirmDestructive` , `CloseCoordinator.unsavedAlert` ; Torrenta `RemoveConfirm.ask` .
- **CONF-13** `MUST` In an app that asks before a destructive action, every such question has one default button, and it is never the red one (Editora's `danger` style class): Cancel where the only action is destructive, Save where saving is an option (CONF-05). The dialog opens with keyboard focus on the default button, so Enter in a question that has just opened keeps what the user has, on every OS, and the red action runs only from its own button: a click, or the focus moved to it with Tab and then Space, which presses the focused button on every OS.
  - *Why:* Enter is the key a hurried hand presses, and Escape already cancels (CONF-04): with Cancel as the default, no single key destroys anything. JavaFX makes a button with the `OK_DONE` role the default by itself, so a named action is the default unless the dialog says otherwise. It also presses a focused button on Enter on Windows and Linux but not on macOS, so a question with no default button answers Enter differently on each OS. With Cancel as the default one difference is left, and it is JavaFX's: once Tab has put the focus on the red button, Enter runs it on Windows and Linux and answers Cancel on macOS. Space is the key that does the same everywhere.
  - *Check (code):* in every question before a destructive action (CONF-02 says which those are), the code calls `setDefaultButton(true)` on Cancel, or on Save in the unsaved-changes question, and `setDefaultButton(false)` on the destructive action, so that one is not left a default button too, and it does both before the dialog is shown. JavaFX gives the opening focus to the first button that is a default button, so with one default button the dialog opens with the focus on it. No such question leaves the red action, or no button at all, as the default.
  - *Reference:* Termina `ui/Confirm.java` (`Options.cancelIsDefault` , which its close-running-program question sets), the app template's `ui/Confirm.java` (`dialog` , which every destructive question of the template asks through) and nux `ui/CommandConfirmation.java` (a risky UPS command): each unsets the action's default and sets Cancel's as it builds the dialog, before it is shown. Editora does it in about twenty of its questions, each by hand: deleting for good what the trash cannot take (`ProjectPanel.cancelByDefault`), closing a terminal with a program running (`ui/TerminalCoordinator.java`), resetting settings (`ui/SettingsResetFlow.java`), trusting a folder (`ui/TrustPrompts.java`) among them. The rest don't yet: Editora's Git confirmations and Local History purges (`ui/HistoryCoordinator.java`), Torrenta's `RemoveConfirm` , Mirada's `Confirm` , and Termina's removals and its Import configuration question leave the red action as the default, so Enter runs it; Torrenta's four Settings questions (CONF-03) leave OK as the default, and so do questions of Editora's that keep the stock OK, its delete to the trash (`confirmDelete` in `ui/ProjectPanel.java` , which moves the default only when a file cannot go to the trash), its Revert and its Save As overwrite among them; Editora's unsaved-changes and Quit questions have no default button. nux's question before a command it does not count as risky leaves the stock OK as the default too, but it is not a question before a destructive action (CONF-02).

Anatomy



1. **Window title**: a short name for the question ("Remove feed", "Unsaved changes"). Torrenta's header repeats it; Editora's unsaved-changes question has the question itself as its header ("Save changes to Main.java before closing?"). A plain confirmation, no icon in the title and no app name.
2. **Header**, when the question is long enough to need one. Editora's Git confirmations set none and put the whole message in the content; Torrenta's header names the action and the content asks the question.
3. **The question names the thing and the consequence.** How many ("these 3 torrents", a `{0,choice,…}` plural), which one ("the feed “Linux ISOs” and its articles"), what stays ("The downloaded files stay on disk."), and "This cannot be undone." only when it is true. Termina's multi-line paste shows the text itself in an expandable area, so the decision is made on what will be sent.
4. **The action button** carries the action's verb (Delete files, Remove, Discard, Quit, Close), never OK or Yes, and the `danger` style: red, not the accent. It sits to the right of Cancel on every OS, and it is never the default button.
5. **Cancel**, in the catalog's word, with the `CANCEL_CLOSE` role: Escape and the window's close button both answer Cancel. It is the default button, so it wears the accent, holds the focus when the dialog opens, and Enter answers it too.

States



- **CONF-05** `SHOULD` A question before unsaved changes are lost, where saving is an option, has three answers, drawn in this order: Cancel, Save in green and Discard in red (Editora's style classes for the two are `success` and `danger`), with Cancel to the left of the actions as in every dialog (WIN-11). A Save that fails leaves the tab open.
  - *Check (code):* the unsaved-changes alert draws Cancel, Save and Discard in that order, on the `ButtonBar`'s one fixed order and not the order its `ButtonType`s were added in, and styles Save and Discard; and when Save is chosen and the save fails, the close is abandoned, so the tab stays open.
  - *Reference:* Editora `CloseCoordinator.unsavedAlert`, `MainController.styleUnsavedChangesButtons`, for the three answers and their styles; it still draws them Save, Discard, Cancel.
- **CONF-06** `SHOULD` A question about ending a running program, which is not the user's data, words the button Close, in red like every confirming button (CONF-03). Cancel is its default button, as in every question with a red action (CONF-13), so dismissing the dialog by any route keeps the program alive.
  - *Check (code):* the close-running-program alert names its action Close. Its red and its default button are CONF-03's and CONF-13's to judge.
  - *Reference:* Termina `ui/TerminalWindow.java` (`dialog.closeAnyway`), through `ui/Confirm.java`.
- **CONF-07** `SHOULD` Where the action can be taken back, the app asks nothing and offers Undo instead.
  - *Check (by hand):* close a clean tab and reopen it with Reopen Closed Tab; in Mirada, trash an image and undo.
  - *Reference:* Mirada moves an image to the trash at once and offers Undo; Torrenta asks before removing a torrent but also keeps the removal on its undo stack for ten minutes (`UndoStack.KEEP_MS`); Editora `ClosedTabs`.

Spec

Property	Value	Reference
Kind	`Alert(CONFIRMATION)` owned by the window (`initOwner`), shown with `showAndWait`; the answer defaults to Cancel when the dialog is closed any other way	`GitCoordinator.confirmDestructive`, `RemoveConfirm.ask`
Buttons	A `ButtonType` with the action's name and the `OK_DONE` role, and Cancel with `CANCEL_CLOSE`, both named from the catalog (CONF-14). Drawn in one fixed order on every OS, Cancel first (WIN-11)	`RemoveConfirm.ask`, Torrenta's `StockButtons`
Action button	`danger` style class: the theme's `.button.danger` fills it `-color-danger-emphasis` with emphasis text. Save in the unsaved question: `success`	`MainController.styleUnsavedChangesButtons`, `styleQuitButtonAsDanger`
Default button	Cancel, or Save in the unsaved-changes question: `setDefaultButton(true)` on it and `setDefaultButton(false)` on the action, because JavaFX makes an `OK_DONE` button the default by itself. CONF-13 is the rule, and its Reference says which questions in which app do it	Termina `Confirm`, nux `CommandConfirmation`
Font	The UI font sheet attached to the dialog pane, because a dialog is its own scene and inherits nothing from the window	Torrenta `Fonts.installUiFont(DialogPane)`, Editora `App.installUiFont` (every window)
Stylesheet	`Dialogs.styled` adds `app.css` to a dialog that needs the app's own rules (About); the `-color-*` tokens and `.button.danger` resolve from the theme either way	`Dialogs`
Text	The question in the content; a header only when the question needs a name over it. Counts as plurals, names in quotes, the consequence spelled out	`Actions.removeQuestion`, `confirm.remove.<kind>`
Strings	One key per kind, header and question both, tested to exist for every kind	`RemoveConfirm.Kind`, `RemoveConfirmTest`

Behavior

- **CONF-08** `SHOULD` Every confirmation question, whether or not it is before a destructive action, comes before anything changes, and its action runs only on the action button. For a question before a destructive action both are MUSTs already (CONF-01, CONF-04); what this rule adds is every other question, such as a warning before something is pasted or run.
  - *Check (code):* every confirmation helper returns true on the action button alone, and its caller acts only on true, for the questions that are not before a destructive action as for those that are.
  - *Reference:* Torrenta `RemoveConfirm.ask`; Editora `GitCoordinator.confirmDestructive`. For a question that is not before a destructive action: Termina's multi-line paste warning, asked through `ui/Confirm.java`, whose `ask` is true for the action button alone, and nux's question before a harmless command (`ui/CommandConfirmation.java`, `confirmed`).

- **CONF-09** `SHOULD` One gesture asks one question. An action on many things asks once with the count ("Remove these 3 torrents?", Torrenta's "Apply to many torrents" for a bulk command). Closing every tab asks per unsaved buffer, selecting each tab first so the user sees what the question is about, and remembers a Discard answer for exactly the state that was shown.
  - *Check (code):* a bulk command builds one question with a plural count; the close-all sweep selects each tab before it asks.
  - *Reference:* Torrenta `Actions.removeQuestion`; Editora `CloseCoordinator.confirmSweep`.
- **CONF-10** `SHOULD` A delete that cannot be undone says so; one that leaves the files says that instead.
  - *Check (code):* "This cannot be undone." appears only in the strings of actions that cannot be undone.
  - *Reference:* Editora `dialog.discard.tracked`; Torrenta `actions.removeQuestion.delete`.
- **CONF-11** `SHOULD` A dangerous alternative (Force with Lease, Delete Anyway) is offered beside the safe one (Pull, then Push), in red, and only once the safe one has failed.
  - *Check (code):* the force path is reached only from the safe command's failure result.
  - *Reference:* Editora `GitBranchCoordinator.offerAfterRejection` (Force with Lease after a rejected push), `confirmForceDelete` (Delete Anyway after `NOT_FULLY_MERGED`).
- **CONF-12** `SHOULD` A confirmation question can be tested without a screen: its words come from pure code, and a capture hook answers the dialog, so a UI test never presses a native button.
  - *Check (code):* the header and the question of every kind are built by pure functions, which need no dialog to call, and the confirmation helper has a hook that answers the dialog in a test without showing it. A test that asserts it is the surest evidence.
  - *Reference:* Torrenta `RemoveConfirm.header`, `question`, `answerForCapture`; `RemoveConfirmTest`.

Copy

Key	English
`dialog.unsaved.title` / `dialog.unsaved.header`	Unsaved changes / Save changes to {0} before closing?
`dialog.save` / `dialog.discard` / `dialog.cancel`	Save / Discard / Cancel
`dialog.quit.header` / `dialog.quit.button`	Quit Editora? / Quit
`dialog.discard.tracked`	Discard all changes to "{0}"? This cannot be undone.
`dialog.discardAll.action`	Discard All
`dialog.deleteBranch.force`	Delete Anyway
`confirm.remove.feed.header` / `confirm.remove.feed`	Remove feed / Remove the feed “{0}” and its articles? The torrents already added stay.
`actions.removeQuestion.delete`	Remove this torrent and delete the downloaded files? This cannot be undone.
`window.remove.button.delete` / `common.remove` / `common.cancel`	Delete files / Remove / Cancel
`dialog.closeRunning` / `dialog.closeAnyway` (Termina)	Close this tab? / Close

In the apps

Editora asks for unsaved changes, Quit with busy terminals, pinned tabs, Git discards and branch deletions, and Local History purges through `Alert` with named buttons. Not all of its questions do yet: 26 of its confirmations still offer JavaFX's stock OK and Cancel, among them the Project panel's file delete (`ui/ProjectPanel.java`, `confirmDelete`) and Replace in Files (`ui/SearchCoordinator.java`), which CONF-03 and CONF-14 do not allow, and 22 more, the Git confirmations' helper among them, name the action but keep JavaFX's stock Cancel, which CONF-14 does not allow. Torrenta's `RemoveConfirm` serves the torrent list, feeds, rules, watched folders, search providers, engine profiles, categories, tags and saved views with one shape. Termina asks before closing a tab with a running program and before a multi-line paste. Mirada's `Confirm` asks before removing a bookmark or discarding an edit, with a named red button; its trash is undoable. nux asks before a risky UPS command with a named Run button and Cancel as the default, though not in red, and its Cancel is JavaFX's stock one; its question before a harmless command keeps the stock OK and Cancel. Modula asks nothing: the Remove row of a preset's right-click menu removes it at once, with no undo, which CONF-01 does not allow; nor does it allow nux's Profiles → Delete, which also asks nothing. Four of Torrenta's questions in Settings (disconnecting sync, Reset to Defaults, Reset All Shortcuts and taking a shortcut back) are answered with a catalog-worded OK, not the action's verb in red.

Find bar

The bar for searching text: in an editor, a read-only console, a terminal's scrollback, a log shown as text. It is part of the window and searches as you type. The page reaches every app, not only the ones with an editor or a terminal: every app has a Debug Log (DIAG-03), which is text to search, so FIND-01 asks for a find bar there. Mirada has no other text to search: its *Find Image...* is a picker, see the QuickOpen template.

Rules

- **FIND-01** `MUST` Wherever there is text to search (an editor, a read-only console, a terminal's scrollback, a log shown as text, such as the Debug Log), the app has a find bar that is part of the window, never a popup. A log shown as a list of entries is narrowed by a filter instead (FIND-03).
  - *Why:* every app has a Debug Log (DIAG-03), and a log is read by searching it.
  - *Check (code):* each view of searchable text, the Debug Log window among them, has a find bar that is a node in its window's scene graph (docked or floating); no `Popup` or `Stage` is created for it. A log drawn as a `ListView` or a `TableView` is not such a view. An app with no Debug Log window has no find bar there either, so it does not meet this rule: it fails with DIAG-03, which requires the window, and the evidence names that rule.
  - *Reference:* Editora `ui/FindReplaceBar.java` (the editor's find and replace), with `ui/ConsoleFindBar.java` and `ui/TerminalFindBar.java` as the same shape over a console and a terminal; Termina `ui/SearchBar.java`. Editora doesn't meet it in every console yet: the build console has the bar (`ui/BuildToolPanel.java`); the Run, External Tools and Debug consoles (`ui/RunPanel.java` , `ui/ExternalToolPanel.java` , `ui/DebugPanel.java`) are read-only text areas with none. The app template's `ui/FindBar.java` is one bar over a text area, a row in the window: over each document (`ui/DocumentView.java`), over the Output console (`ui/SamplePanels.java`) and over its Debug Log (`ui/DebugLogWindow.java`). No released app's Debug Log has a find bar yet. Editora's Message Log (`ui/MessageLogPopup.java`) and Torrenta's log window (`ui/LogWindow.java`) are lists, which FIND-03 covers.
- **FIND-02** `MUST` A find bar searches as its query is typed and shows a match count.
  - *Check (code):* in each find bar the app has, the query field's text listener runs the search and updates the count label. A view where FIND-01 asks for a find bar and the app has none fails this rule with FIND-01, and so does an app with no find bar and no Debug Log window at all; the evidence names that rule.
  - *Reference:* Editora `FindReplaceBar` (`debounce` , `find.count`); Termina `SearchBar` (`search.count`); the app template's `ui/FindBar.java` (`searching` , `find.count`), in each of its three bars.
- **FIND-03** `MUST` A list is narrowed by a filter box where text is searched by a find bar. In an app whose main content is a list, a filter box over that list takes the place of a find bar there, with the same field and the same Escape. The main content is a list when it is a list or a table of items that grows without bound and that the user finds things in; Torrenta, with its torrents, is the one such app today. In every app, a log shown as a list of entries (a Message Log, an event log, a Debug Log drawn as a list or a table) is covered the same way: a filter field over the list narrows it, and it needs no find bar. An event log and a log tab that lists one item's events count as logs shown as a list (nux's event log, the Log tab of Torrenta's detail pane); a table of readings or of history does not.
  - *Check (code):* in an app whose main content is a list (a list or a table of items that grows without bound and that the user finds things in), a filter field sits over that list and filters it as it is typed, and Escape pressed in it leads back to the list, as it leads back to the text from a find bar (FIND-07). Whether the first Escape empties the box before the next one leaves it is FIND-14's. In every app, each log shown as a list or a table of entries has a filter field over it that narrows the entries as it is typed; check boxes that narrow a log by level are not one. An event log and a tab that lists one item's events are such logs; a table of readings or of history is not judged.
  - *Reference:* Torrenta `Omnibox` , `FilterQuery` , `MainWindow.onOmniboxKey` . Over a log shown as a list: Editora's `ui/MessageLogPopup.java` , a filter field over its `ListView` (`applyFilter`), and the Message Logs of Torrenta (`ui/MessageLogPopup.java`), Mirada (`ui/MessageLogView.java`) and the app template (`ui/MessageLogView.java`), each a filter field over a `ListView` . Four logs don't meet it yet: Torrenta's `ui/LogWindow.java` filters its table by level with three check boxes and has no field to type in; the Log tab of Torrenta's detail pane (`ui/DetailPane.java` , `buildLog`) lists the selected torrent's events with no filter; and Termina's Message Log (`ui/MessageLogWindow.java`) and nux's event log (`ui/EventsWindow.java`) are a list and a table with no filter.

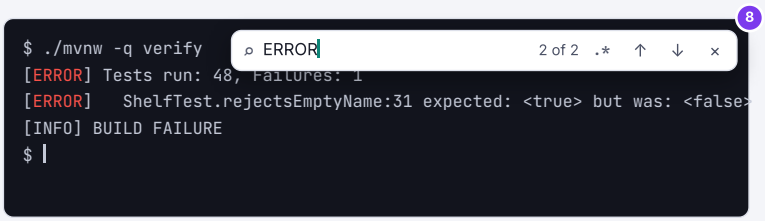
Anatomy



1. **Query field**, focused on open with its text selected, so typing replaces the last query. A single-line selection in the editor is the first query; a multi-line one becomes the scope instead and switches *Sel* on. A clear button appears while the field has text.
2. **Count** "3 of 12". Empty when there is no query. When nothing matches, Termina's card and Editora's console and terminal bars say "No matches"; Editora's editor bar empties the count and says "Not found: x" in the status line. Termina draws it 11 px muted and at least 64 px wide, so the buttons beside it do not shuffle as the digits grow; Editora's label has no style of its own.
3. **Previous and next**, 28 px flat icon buttons, the same as Shift+Enter and Enter.
4. **Options** as compact toggles with a two-letter caption (match case, regular expression, whole word, preserve case, in selection); the tooltip and the accessible name carry the full meaning. Disabled while the query is empty, except *Sel*, which is a scope.

5. **Replace** replaces the active match and steps to the next; *All* replaces every match and reports "Replaced 12 occurrence(s)" in the status line (Editora's string, which does not meet INTL-12 yet). A console or a terminal has no replace row.
6. **Close**, top right whatever the bar wraps to. Escape does the same.
7. **Matches** are all highlighted while the bar is open, the active one distinctly, and the highlights follow the text as it is edited. They go when the bar closes.

The same bar over a terminal floats at the top right of the pane it searches, in the palette's materials, so the two read as one family: a lens, the field as the card's own text, the count, a regex switch, and up, down and close.



8. **Floating card**, 380 px wide, radius 8, over the output being read. Enter steps *up* to older output and Shift+Enter down: a terminal is searched for something recent, so the search starts at the bottom (iTerm2's convention). The arrow keys do the same.

Spec

Property	Value	Reference
Kind	In-scene, never a popup. Editora docks the bar above the editor area; Termina floats it inside the `TerminalView`, so each pane of a split has its own and a search never wonders which terminal it is about	`MainController` (`topBox`), `SearchBar`
Docked bar	`-color-bg-subtle`, padding 4 8, spacing 6; icon buttons 28 × 28, flat; the groups (find, replace, options) wrap onto a second line in a narrow window instead of shrinking the fields	`.find-bar`, `.find-bar-icon`, `WrapRow`
Floating card	`-color-bg-overlay`, 1 px `-color-border-default`, radius 8, 380 px wide, padding 3 4 3 8, shadow 18 at offset 6; the field is the card's own text at 12.5 px	`.search-bar`, `.search-field`
Options	Toggles with a short caption; tooltip and accessible name carry the meaning; selected: `-color-accent-subtle` with `-color-accent-fg`	`OptionToggle`, `.search-toggle:selected`
Count	`{0} of {1}`. Termina: 11 px `-color-fg-muted`, right-aligned, minimum width 64 px, "No matches" when there are none. Editora's label has no style of its own, and its editor bar empties it when nothing matches	`find.count`, `search.count`, `search.none`, `COUNT_MIN_WIDTH`, `.search-count`
Debounce	The search runs 150 ms after the last keystroke in Editora and 120 ms in Termina. In Editora an edit to the text re-highlights after the same 150 ms without moving the caret	`debounce`, `editDebounce`, `TYPING_SETTLE`; Termina's `searchTyping` in `TerminalView`
Threading	A document up to 512 K characters is searched where the request is made; larger ones on one shared daemon thread, latest query only, the result applied on the FX thread. A scrollback is always scanned off the FX thread	`SYNC_SEARCH_CHARS`, `SEARCH_POOL`, `LatestOnly`, `TerminalView.searchAsync`
Bounds	A page of matches around the active one, not every match of a query that matches millions of times; a regular expression has a time budget. Termina keeps the newest 1,000 matches	`SearchMatcher.DEFAULT_PAGE`, `find.tooComplex`, `ScrollbackSearch.MAX_MATCHES`
Bad regex	Said in place, never by a dialog: the status line ("Bad regex: …") or, in Termina, the field in `-state-red` and the count "Invalid"	`find.badRegex`, `.search-field:invalid`
Focus	A button may leave the Tab order where a command, or Enter, Shift+Enter or Escape in the field, does what it does; an option toggle stays in it unless a command toggles the option (FIND-16). A terminal's bar is no exception: Editora's keeps its field and its match-case toggle in the Tab order, Termina's only its field	`setFocusTraversable(false)`

Behavior

- **`FIND-04`** `SHOULD` A command opens each find bar, and run while its bar is already open it finds the next match. For the main window's own text (the editor; the terminal in a terminal app) the command is `find.show`, bound to `C-s` in the Emacs keymap and `Ctrl/Cmd+F` in CUA, Sublime, VS Code and IntelliJ, with `F3` as well in CUA and Sublime (`Cmd+G` in place of `F3` in CUA on macOS); a terminal binds it to `Ctrl+Shift+F`, `Cmd+F` on macOS, since a plain `Ctrl+letter` belongs to the shell. Any other bar (over a log window, or over a console or a terminal panel beside the editor) has a command with an id of its own, as the app template's `debugLog.find` is for its Debug Log.
  - *Check (code):* each bundled keymap binds that chord to `find.show`; every other find bar is opened by a registered command of its own; and each of these commands, run while its bar is shown, steps to the next match.
  - *Reference:* Editora `keymaps/*.json` and `MainController.findShowOrNext`, which finds the next match while the editor's bar is open. The app template's bars all do: `find.show`, bound in each of its keymaps, over the documents, `output.find` over the Output console and `debugLog.find` over the Debug Log each open their bar and, run while it is open, step to the next match (`openOrNext` in its `ui/FindBar.java`; `ui/Commands.java`; `FindBarFxTest`). No other app's bar does both yet. Termina's find command (`TerminalWindow`, `menu.find`, `MenuAction.appChord`) runs `TerminalView.showSearch`, which puts the caret back in the field of a bar that is already open. So does Editora's `terminal.find` over its terminal panel (`TerminalCoordinator.find`). The template's Debug Log window also runs `debugLog.find`

(`MainWindow.findInDebugLog`) on a fixed `Ctrl/Cmd+F` . Editora's console bar has no command: the find command's chord pressed in the console opens it (`ConsoleNav.onCommand` in `ui/BuildToolPanel.java`), as does a button in the panel's header.

- **FIND-05** `SHOULD` Typing in a find bar highlights every match, the active one distinctly, and selects and scrolls to the match the search starts on. In a document (an editor, a console, a log) that is the first match at or after the caret as it was when the bar opened, so refining the query does not run away down the document. In a terminal it is the newest match at or above the bottom of the view, since a terminal is searched upward (FIND-06). A view that can draw one selection and nothing else, such as a log in a plain text area, shows the active match alone, and the count (FIND-02) says how many there are.
  - *Check (code):* every match in view is painted while the bar is open, not only the active one, unless the view can draw a single selection only. In a document the search starts from an anchor saved when the bar opens, not from the current selection; in a terminal, from the bottom row of the view.
  - *Reference:* Editora `FindReplaceBar` (`searchAnchor` , `recompute`) over the editor. Over a terminal, Editora's `terminal/TerminalView.java` (`startIndex` , `drawSearch`) and Termina's `TerminalView` (`drawSearchMatches`) with `ScrollbackSearch.initial` . The app template's `ui/FindBar.java` saves the caret as its `anchor` when the bar opens and `refine` searches from it (`FindBarFxTest` , `theSearchStartsAtTheCaretAndTheFieldKeepsItsEditingKeys`). Editora's `ConsoleFindBar` doesn't yet: it searches from the current selection (`refine`), not from an anchor saved when the bar opened. Editora's console is a `CodeArea` , which could paint every match, and its bar paints the active one only. The template's documents, Output and Debug Log are each a `TextArea` , which draws one selection and nothing else, so its bar rightly shows the active match alone.
- **FIND-06** `MUST` In a find bar, Enter steps to the next match and Shift+Enter to the previous one, wrapping around at the ends. In a find bar over a terminal the directions are up and down the scrollback: Enter steps up to older output, Shift+Enter down.
  - *Check (code):* in each find bar the app has, the query field's key handler steps to the next match on Enter and to the previous one on Shift+Enter (over a terminal, up to older output on Enter and down on Shift+Enter), and a step past the last match goes on to the first, and back. A view where FIND-01 asks for a find bar and the app has none fails this rule with FIND-01, and so does an app with no find bar and no Debug Log window at all; the evidence names that rule.
  - *Reference:* Editora `ConsoleFindBar` , `TerminalFindBar` ; Termina `SearchBar` . Editora's editor bar (`FindReplaceBar`) does not yet: its field steps forward on Enter, and back only with the previous button, `find.showBackward` (`C-r` , `Shift+F3`) or `find.previous` . Alt+Enter there turns every match into a caret. The app template's `ui/FindBar.java` steps and wraps the same way (`step`), in each of its three bars.
- **FIND-07** `MUST` Escape or the close button closes a find bar: its highlights go, a search still pending is canceled so nothing moves after the bar is gone, and focus returns to the text the bar searched (the editor, the console, the shell or the log). In a find bar over a terminal, the Escape that closes the bar does not reach the shell.
  - *Check (code):* in each find bar the app has, the close path, run by Escape in the field and by the close button, stops the debounces, cancels the running search, clears the highlights and focuses the text the bar searched. In a find bar over a terminal, the field's key handler consumes the Escape it closes the bar on, so the terminal never sends that key to the shell. A view where FIND-01 asks for a find bar and the app has none fails this rule with FIND-01, and so does an app with no find bar and no Debug Log window at all; the evidence names that rule.
  - *Reference:* Editora `FindReplaceBar.hideBar` ; the app template's `ui/FindBar.java` (`close`), in each of its three bars. Over a terminal, Termina's `SearchBar` (`onKey`) and Editora's `TerminalFindBar` each close the bar on Escape and consume the key.
- **FIND-08** `SHOULD` A terminal with a find bar keeps its own key and mouse filters off the bar: they leave every event aimed at the bar alone, so what is typed or clicked there never reaches the shell. The Escape that closes the bar is FIND-07's.
  - *Check (code):* no key or mouse filter of the terminal acts on an event aimed at the find bar: the filters return early for an event whose target is inside the bar (Termina's `fromSearchBar`), or the bar is outside the node the filters are on, so its events never pass them.
  - *Reference:* Termina `TerminalView.fromSearchBar` , which its key, mouse and scroll handlers ask first.
- **FIND-09** `SHOULD` In a find bar's fields, the keymap's caret and editing chords (`C-a` , `C-e` , `C-k` , `C-y` in the Emacs keymap; `Ctrl+V/X/C/Z` in CUA, `Cmd+V/X/C/Z` on macOS) act on the field, not on the document. Find's own chords stay global, so next, previous, replace and close work while the field has focus.
  - *Check (code):* in each find bar's fields, the active keymap's caret and editing chords are handled by the field and never run the document's commands (Editora installs its shared text-input keymap on the fields), and the commands for next, previous, replace and close still run while a find field has the focus.
  - *Reference:* Editora `TextInputKeymap.installShared` , installed on `FindReplaceBar` 's find and replace fields. Editora's `ConsoleFindBar` and `TerminalFindBar` don't yet: neither installs it on its field. Nor does Termina's find field: its `TextInputKeys` is installed on the palette only. The app template's `ui/TextInputKeys.java` (`install`) is on the field of each of its find bars, and its key dispatcher leaves a text field's editing chords to the field (`TextInputKeysTest` ; `FindBarFxTest` , `theSearchStartsAtTheCaretAndTheFieldKeepsItsEditingKeys`).
- **FIND-10** `SHOULD` Nothing found is one status-line message ("Not found: x") or the count "No matches". No beep, no dialog.
  - *Check (code):* the no-match path sets `find.notFound` or `search.none` and opens nothing.
  - *Reference:* Editora `FindReplaceBar` (`find.notFound`); Termina `SearchBar` (`search.none`).
- **FIND-11** `SHOULD` In a find bar that replaces (an editor's), Replace keeps each match's case when *AB* (preserve case) is on.
  - *Check (code):* with preserve case on, the replace path gives each replacement the casing of the match it replaces (Editora's `PreserveCase.apply`). A test that asserts it is the surest evidence.
  - *Reference:* Editora `FindReplaceBarFxTest` (`preserveCaseFollowsEachMatchesCasing`).
- **FIND-12** `SHOULD` In a find bar that replaces, Replace and Replace All run under a time budget, as the search does: a replacement that outruns it stops and says so in the status line (in Editora's words, "Replace stopped: the pattern is too slow") instead of hanging the window. How long the budget is is the app's own.
  - *Check (code):* Replace and Replace All match under a time budget, as the search does, and when it runs out they stop and say so in the status line (Editora's `find.replaceTimeout`) instead of hanging the window. A test that asserts it, with a catastrophic regular expression in Replace All, is the surest evidence.
  - *Reference:* Editora `find.replaceTimeout` , `FindReplaceBarFxTest` (`catastrophicRegexInReplaceAllAbortsInsteadOfHanging`).
- **FIND-13** `SHOULD` In an app whose main content is a list (FIND-03 says which apps those are), the filter box filters the names as plain words are typed, turns `key:value` terms into chips and runs a command after `>` . A command puts the focus in the box (`filter` in Torrenta), bound to `Ctrl+K` , `Cmd+K` on macOS, and the box draws that command's chord as a key badge.
  - *Check (code):* the box's text listener filters the names by the plain words, turns a `key:value` term into a chip and runs the command typed after `>` ; a registered command focuses the box, on `Ctrl+K` (`Cmd+K` on macOS); and the badge in the box is read from that command's chord, not written out.
  - *Reference:* Torrenta `Omnibox` , `MainWindow.buildFilterBar` .
- **FIND-14** `SHOULD` In an app whose main content is a list (FIND-03 says which apps those are), Escape in the filter box peels one layer: a box with something in it empties, an empty one gives the list back.
  - *Check (code):* the box's Escape handler clears a non-empty box and otherwise focuses the list.
  - *Reference:* Torrenta `MainWindow.onOmniboxKey` .
- **FIND-15** `SHOULD` In an app whose main content is a list (FIND-03 says which apps those are), the filter box is out of the Tab order, so a fresh launch does not land every keystroke in it, and the window's own chords (in Torrenta, undo and the saved views' `Ctrl+Alt+1…9` , `Cmd+Alt+1…9` on macOS) do not fire while a

text field has focus: `Ctrl+Z` (`Cmd+Z` on macOS) there is the field's own undo.
  - *Check (code):* the box is not focus-traversable, and the window's chord handlers return early when the focus owner is a `TextInputControl`.
  - *Reference:* Torrenta `MainWindow.buildFilterBar`, `MainWindow` window chords.
- **`FIND-16`** `MUST` A control of a find bar or of a filter box leaves the Tab order only when the keyboard does what it does another way: a registered command does the same, or the bar's own field handles a key that does. So the Next, Previous and Close buttons may leave it, since Enter, Shift+Enter and Escape in the field do the same (FIND-06, FIND-07), and so may a field's clear button, since the field's editing keys empty it. An option's toggle (match case, whole word, regular expression) stays in it unless a command toggles that option. A list app's filter box leaves it (FIND-15) because its command puts the focus there (FIND-13). A terminal's bar is no exception: Tab belongs to the shell only while the terminal has the focus, so a toggle taken out of the Tab order there needs a command like any other.
  - *Why:* KBD-15 lets a control out of the Tab order only when the keyboard does its work another way. A toggle that is out of it with no command can be changed with the mouse alone.
  - *Check (code):* for every control of a find bar or filter box that is set not focus-traversable, a registered command does what the control does, or the bar's field handles a key that does: Enter, Shift+Enter or Escape for Next, Previous and Close, the editing keys for a clear button. A toggle that is not focus-traversable has a command that toggles its option. An app with no find bar and no filter box has no such control: nothing breaks this rule and it passes, the missing bar being FIND-01's finding.
  - *Reference:* Editora's `FindReplaceBar.java` keeps every button and option box in the Tab order, all but the clear button inside each field. Its `ConsoleFindBar.java` and `TerminalFindBar.java` take Previous, Next and Close out of it, which Shift+Enter, Enter and Escape in the field do, and leave the field and the match-case toggle in. The app template's `FindBar.java` takes the same three out: Previous and Next run its bar's step commands (`find.previous` and `find.next` over a document, `output.findPrevious` and `output.findNext`, `debugLog.findPrevious` and `debugLog.findNext`), Close is the field's Escape, and the Match Case box stays in. Torrenta's filter box is out of it and reached by its `filter` command (`MainWindow.buildFilterBar`). In Mirada's `BookmarksWindow.java` the three kind toggles are out of it and have a command each (`bookmarks.listAll`, `bookmarks.listBookmarked`, `bookmarks.listNotes`). Termina's `SearchBar.java` leaves only its field in the Tab order: the two arrows and Close are the field's Enter, Shift+Enter and Escape, and the arrows its `findNext` and `findPrevious` commands too, but its regex toggle doesn't meet the rule yet: it is out of the Tab order and no command toggles it.

Copy

Key	English
`find.prompt`	Find
`find.replacePrompt`	Replace
`find.count`	{0} of {1}
`find.notFound`	Not found: {0}
`find.badRegex`	Bad regex: {0}
`find.tooComplex`	Search stopped early: the pattern is too complex for this text
`find.replaced`	Replaced {0} occurrence(s) *(does not meet INTL-12 yet)*
`search.caseTip` · `search.regexTip` · `find.wholeWord`	Match case · Regular expression · Whole word
`search.older` · `search.newer` · `search.close` (Termina)	Previous match, up (Enter) · Next match, down (Shift+Enter) · Close (Escape)
`search.none` (Termina, and Editora's console and terminal bars) · `search.invalid` (Termina)	No matches · Invalid
`window.filter.prompt` (Torrenta)	Filter · > runs a command · paste a magnet

In the apps

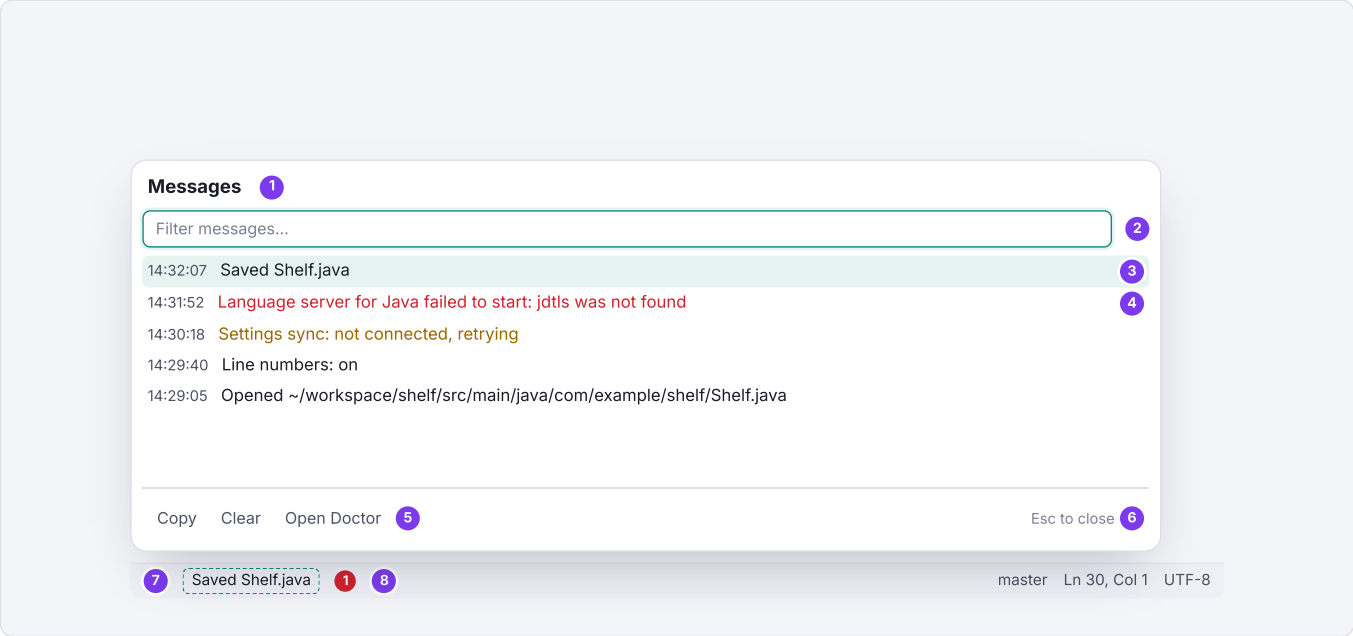
- **Editora:** find and replace over the editor (`FindReplaceBar`), with *Find in Files* (`search.inFiles`) as a separate results window; the build console and the terminal panel carry the same bar without the replace row. The Run, External Tools and Debug consoles have no find bar yet (FIND-01).
- **Termina:** the floating card over each pane, with the regex switch; matches are found across soft-wrapped lines, so a word that reached the right edge and carried on is still found.
- **Torrenta:** the omnibox above the transfers, which filters, runs commands and adds what is pasted.
- **Mirada:** no text of its own to search; it finds an image through a picker.
- **Modula, nux:** no text to search, and no Debug Log yet (DIAG-03).
- **The app template:** one find bar (`ui/FindBar.java`) over each document, over the Output console and over its Debug Log, opened by `find.show`, `output.find` and `debugLog.find` (and `Ctrl/Cmd+F` in the Debug Log window).
- **The Debug Log:** Editora's, Termina's, Torrenta's and Mirada's is a read-only text area with no find bar (`ui/DebugLogWindow.java` in each), which FIND-01 asks for.

Message Log

The status bar's echo line shows one message and replaces it with the next; the Message Log keeps them, so nothing the app said is lost because it scrolled past.

Every app keeps one, session-only and bounded, newest first, with Esc to close and Clear to empty it: DIAG-01. A click on the status bar's echo area opens it: WIN-01. This page is what the log looks like and how it behaves beyond that.

Anatomy



1. **Title** "Messages", 14 px bold.
2. **Filter**, focused on open. Filters as you type; ↓ or Enter moves into the results without leaving the field.
3. **Entry.** The time as `HH:mm:ss` (11 px, muted, in the UI font like the rest of the card, not monospace) and the message as it appeared in the echo line. Newest first.
4. **Severity** colors the message, not the row: errors red, warnings amber, information in the default color. The time stays muted.
5. **Actions:** Copy (the selected entries as text, or every listed entry when none is selected), Clear (empties the session log and its unread count), and Open Doctor: messages carry no actions of their own, and a missing tool or a server that would not start is where Doctor helps.
6. **Key hint**, muted, 11 px.
7. **Echo line.** The status bar's latest message; a click opens the log.
8. **Unread errors.** A red count beside the echo line while errors have arrived since the log was last opened; a click opens the log.

Spec

Property	Value	Reference
Kind	A card in the shared in-scene overlay host, anchored just above the status bar's echo area, over the window	`MessageLogPopup.show`, `OverlayHost`
Size	773 px wide; the list 336 px high, scrolling past that	`WIDTH`, `LIST_HEIGHT`
Card	`-color-bg-overlay`, radius 12, like the palette	`.message-log`
Rows	Padding 2 4, radius 5; selected row `-state-sel`	`.message-log-list .list-cell`
Capacity	The newest 200 messages; older ones drop off	`MessageLog.MAX_ENTRIES`
Repeats	A message repeated within 2 s moves to the top as one entry, unread once	`REPEAT_MILLIS`
Lifetime	This session only, in memory. Not written to disk: the Debug Log is the persistent record	`MessageLog`
Blank messages	Not recorded (a blank message only clears the echo line)	`MessageLog.add`

- **`MLOG-08`** `SHOULD` The Message Log is a card in the window's in-scene overlay host, anchored just above the status bar's echo area, never a separate window: 773 px wide, its list 336 px high, at the default zoom, holding the newest 200 messages of the session (the cap DIAG-01 asks for).

- *Why:* the log answers "what just happened?" next to the line that said it; a separate window, or another size and cap, makes the same log look and behave differently from app to app.
- *Check (code):* the log adds its node to the overlay host (no `Stage`), with width 773, list height 336 and a cap of 200.
- *Reference:* Editora `ui/MessageLogPopup.java` (`WIDTH`, `LIST_HEIGHT`) and `ui/MessageLog.java` (`MAX_ENTRIES`); the app template's `ui/MessageLogView.java`. Torrenta's card is 640 × 300, and Termina's log is a separate window that keeps 500 messages.

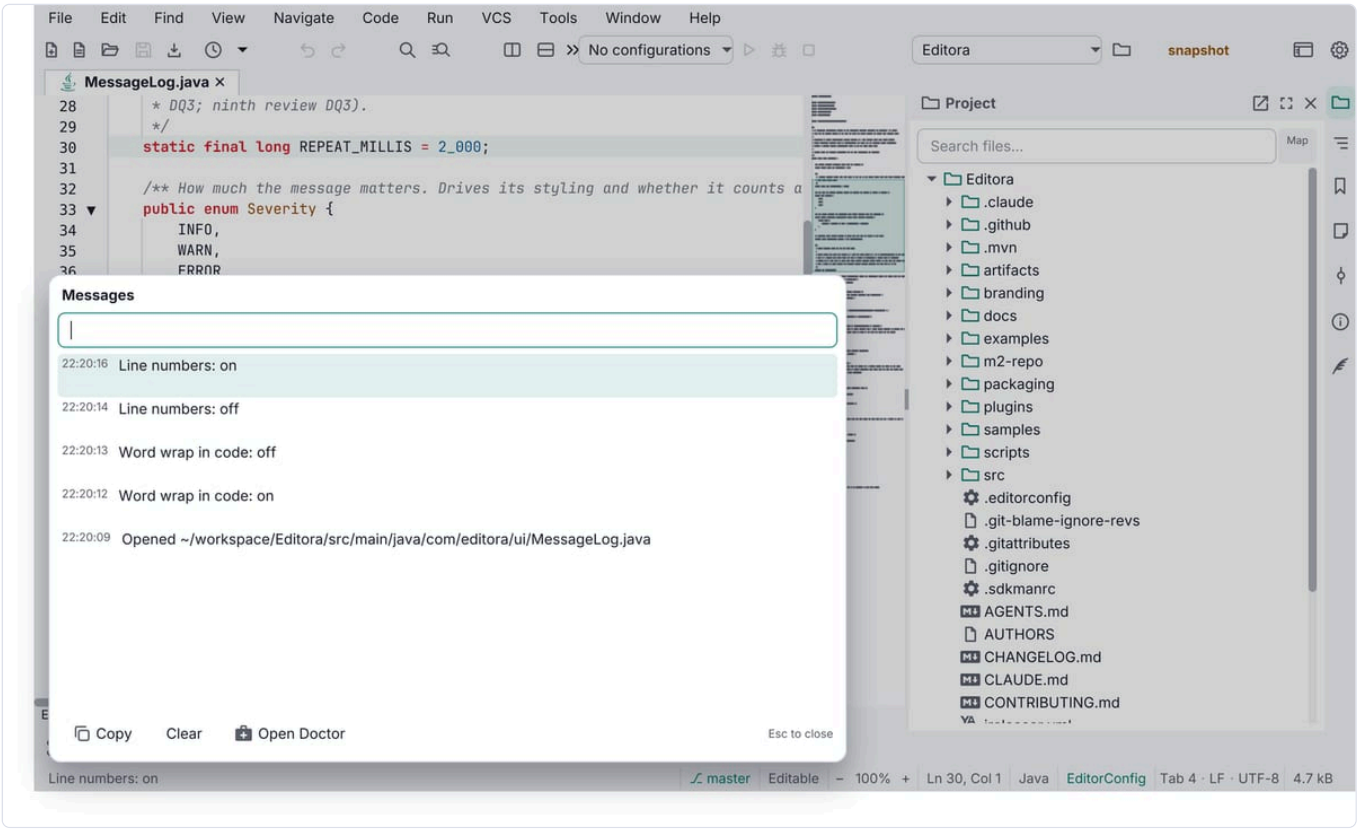
Behavior

- **`MLOG-02`** `SHOULD` A click on the status bar's unread-error count opens the Message Log, as a click on the echo area does (WIN-01), and so does the `view.messageLog` command.
  - *Check (code):* the unread-error count runs `view.messageLog` on a click, and the command is registered.
  - *Reference:* Editora `StatusBar` (`echo`, `unreadErrors`), `StatusBar.showMessageLog`.
- **`MLOG-03`** `SHOULD` The active keymap's cancel chord or a click outside closes the Message Log, as Esc does.
  - *Check (code):* the Message Log closes on the chord the active keymap binds to its cancel command, not on a fixed `Ctrl+G`, and a press outside the log closes it: on the backdrop, where the log is the in-scene card MLOG-08 asks for. In an app with one key table (a terminal app's keymap, or the built-in table of an app that ships no keymap files), the chord is the one that table binds to a cancel command; where it binds none, Esc and the press outside are all there is to test.
  - *Reference:* Editora `OverlayHost`, which hides the card on a press on its backdrop and on `PickerKeys.isCancel`: Esc, or the chord the keymap binds to `edit.cancel` (Editora's id; `C-g` in its Emacs keymap). Its `MessageLogPopup` also closes on `Ctrl+G` whatever the keymap, though the CUA keymap binds `C-g` to `nav.goToLine`; its handler reads `Alt+G` too, which the Emacs keymap's `M-g` prefix takes first. The app template's Message Log (`ui/MessageLogView.java`) is shown through its `ui/OverlayHost.java`, which closes it the same way, on the keymap's `edit.cancel` chord (`cancelOn`). Elsewhere the chord is fixed: `Ctrl+G` in Termina (`ui/MessageLogWindow.java`) and Mirada (`ui/MessageLogView.java`). Torrenta's (`ui/MessageLogPopup.java`) closes on Esc and on a press on its backdrop: Torrenta has no keymap and no cancel command, so the chord asks nothing more of it.
- **`MLOG-04`** `SHOULD` The Message Log copies the selected entries on the platform's copy chord, `Ctrl+C` (`Cmd+C` on macOS), in every keymap, and on the active keymap's copy chord too.
  - *Check (code):* the Message Log's key handling copies the selected entries on C with the platform's shortcut modifier down (`isShortcutDown`), whatever the active keymap binds to copy, and also on the chord the active keymap binds to its copy command (`M-w` in an Emacs keymap), read from the keymap and not written into the handler.
  - *Reference:* for the platform's chord, Editora `MessageLogPopup.copySelection`, on a fixed shortcut-down C, and the Message Logs of Torrenta (`ui/MessageLogPopup.java`), Termina (`ui/MessageLogWindow.java`), Mirada (`ui/MessageLogView.java`) and the app template (`ui/MessageLogView.java`), each the same way. For the keymap's chord, no app yet: none of the five reads the keymap's copy binding in its log, and Editora's Emacs keymap binds copy to `M-w`, which its log does not read. The app template meets the rule all the same: none of its keymaps binds a copy command, so there is no second chord to read, and its key dispatcher steps aside while the log, an overlay, is open (`KeyDispatcher`), so `Ctrl+C` reaches it under the Emacs keymap's `C-c` prefix too. Editora doesn't hold the first half under that keymap on Windows and Linux either: the keymap starts longer chords with `C-c`, a prefix the window's key dispatcher (`command/KeyDispatcher.java`) consumes in a scene filter, ahead of the log's own.
- **`MLOG-05`** `SHOULD` A message's severity comes from its key: a `status.*` key whose last segment contains `fail` or `error` is an error, one containing `cannot` or `invalid` a warning. Name new keys accordingly, or post the message as an error directly (Editora's status bar has an error channel for that).
  - *Check (code):* one function gives a message's severity from its key by those rules (Editora's `StatusSeverity.ofKey`). A test that asserts it is the surest evidence.
  - *Reference:* Editora `ui/StatusSeverity.java` (`ofKey`), `StatusSeverityTest`.
- **`MLOG-06`** `SHOULD` Opening the Message Log, or clearing it, resets the unread-error count.
  - *Check (code):* opening the Message Log marks it read, and clearing it zeroes the count (Editora's `markRead` and `clear`).
  - *Reference:* Editora `MessageLog.markRead`, `MessageLog.clear`, `StatusBar.showMessageLog`.
- **`MLOG-07`** `SHOULD` An unread error is never evicted from the Message Log silently: the unread-error count keeps it even when the entry drops off the end.
  - *Check (code):* the Message Log counts unread errors apart from the entries it holds, so dropping an unread error at the cap leaves the count as it was. A test that asserts it is the surest evidence.
  - *Reference:* Editora `MessageLogSeverityTest` (`anEvictedErrorStillCounts`).

Copy

Key	English
`messagelog.title`	Messages
`messagelog.filter`	Filter messages...
`messagelog.empty`	No messages yet
`messagelog.copy` / `messagelog.clear`	Copy / Clear
`messagelog.doctor`	Open Doctor
`messagelog.hint`	Esc to close

In the apps



Editora.

Settings Sync

Keeps the user's data (snippets, templates, shortcuts, profiles, saved views) the same on every computer. No server and no account of ours: the store is a private Git repository the user owns.

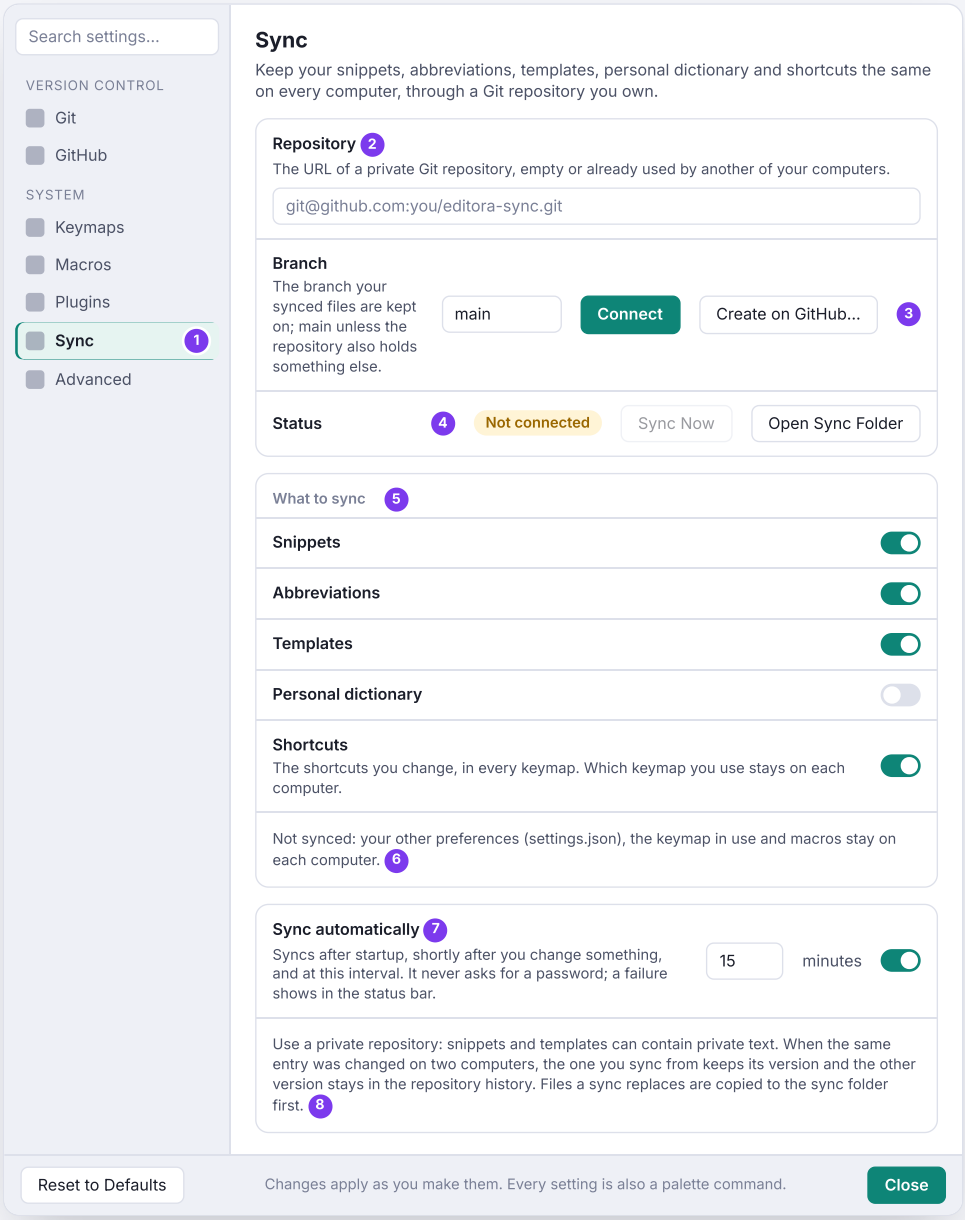
SYNC-01 says which apps offer it: those with user data worth carrying between computers. Every other rule about sync, SYNC-02 here and SET-18 to SET-21 on the Settings page, applies to an app with Settings Sync.

Rules

- **SYNC-01** `SHOULD` An app with user data worth carrying between computers offers Settings Sync, on a page of its own in the Settings window that the Settings search finds by "sync". Such data is what a person builds up in the app by hand and would set up again on another computer: snippets, templates, shortcuts, profiles, presets, bookmarks, saved views.
  - *Check (by hand):* open Settings, search "sync", and land on the Sync page. An app that keeps none of the data the statement lists is outside the rule.
  - *Reference:* Editora `ui/SettingsSync.java` (the service) and `ui/SettingsSyncPage.java`, built by `SettingsWindow` (`Category.SYNC`). Termina, Torrenta and the app template have Settings Sync too. Modula, Mirada and nux keep such data (presets, bookmarks, profiles) and have no Settings Sync yet.
- **SYNC-02** `SHOULD` In an app with Settings Sync (SYNC-01), the store is a Git repository the user owns, reached through their own Git credentials (a credential helper or an SSH key), with no server or account of ours. Refusing a URL with a password or token in it is SET-15.
  - *Check (code):* the app's sync code runs `git` against the URL the user gave, with their own Git credentials, and talks to no server or account of the app's own.
  - *Reference:* Editora `sync/` package (`SyncEngine` , `SyncMerge` , `SyncAccount`), `SyncEngine.hasCredentials` ; Termina carries the same package, Torrenta its own `sync/` with `SyncEngine` and `SyncAccount` ; the page's Spec.

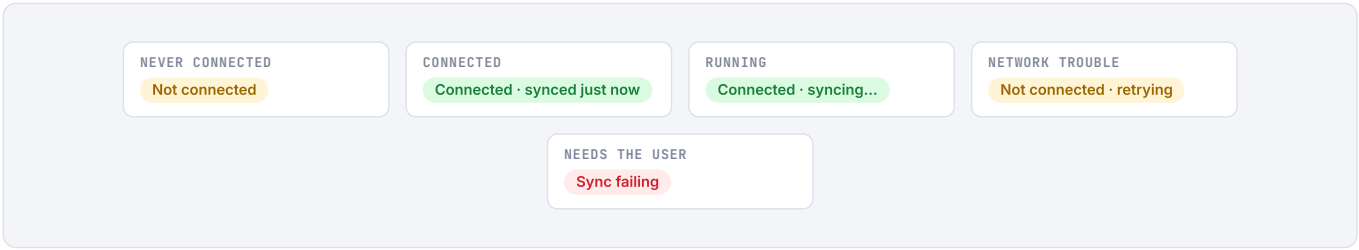
How sync behaves is on the Settings page: setup and the pinned account (SET-18), when it runs and what never travels (SET-19), who wins a conflict and the backup (SET-20), and where its status shows (SET-21). This page shows them and does not state them again as rules.

Anatomy



1. **Its own page** in the Settings sidebar, under System, found by the Settings search ("sync", "git", "github").
2. **Repository URL.** SSH or HTTPS. A URL with a password or token in it is refused, with the reason and the fix (a credential helper such as `gh auth setup-git`, or an SSH URL).
3. **Branch, Connect, Create on GitHub....** Connect clones or adopts the repository. Create on GitHub makes a private repository through the `gh` CLI and fills the URL in.
4. **Status pill** plus Sync Now and Open Sync Folder (see the states below).
5. **What to sync:** one switch per kind of data, each with a description when its scope is not obvious.
6. **What is not synced**, said plainly: machine-local preferences, the keymap in use, anything with paths.
7. **Automatic sync** with its interval in minutes (default 15, at least 1). Automatic runs never prompt.
8. **Privacy and conflicts**, in one note: use a private repository; on a same-entry conflict the computer you sync from wins, and the other version stays in the history; files a sync replaces are copied to the sync folder first.

Status pill states



Green, amber and red are the Settings status-pill colors. Amber means "will recover by itself" (offline, the remote unreachable); red means "the user must act" (Git missing, the repository written by a newer version, a sync that would remove most of a kind of data and waits for confirmation). The pill's accessible text is "Sync status: ...".

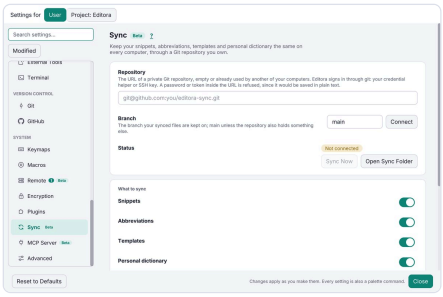
Spec

Property	Value	Reference
Store	A Git repository the user owns, cloned into `<config>/sync/`; one folder per kind of data	`SyncEngine`, `SyncTarget`
Sign-in	The user's own Git credentials (credential helper or SSH key); the GitHub account that can read the repository is pinned per clone (SET-18)	`SyncAccount`
Automatic runs	After startup, 30 s after a local change, and every *n* minutes (default 15) (SET-19)	`SettingsSync.DIRTY_DELAY_SECONDS`, `syncIntervalMinutes`
Conflicts	Per entry; this computer wins; the other version stays in history (SET-20)	`SyncMerge`
Safety	Files a sync replaces are copied to the sync folder first (SET-20); a sync that would remove most of a kind of data stops and asks	`SyncReport`
One runner	Sync runs in the first instance only; a second instance says so	`status.sync.secondaryInstance`
Failures	To the status bar and the Message Log, said once per failure, not once per interval (SET-21)	`SettingsSync`
Palette	Settings Sync: Set Up..., Sync Now, and Toggle Sync Automatically	`sync.setup`, `sync.now`, `sync.toggleAuto`

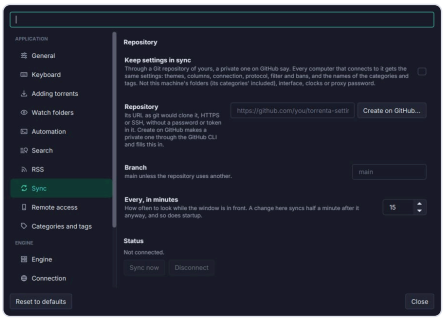
Copy

Key	English
`settings.sync.subtitle`	Keep your snippets, abbreviations, templates, personal dictionary and shortcuts the same on every computer, through a Git repository you own.
`settings.sync.connect` / `settings.sync.createGithub`	Connect / Create on GitHub...
`settings.sync.now` / `settings.sync.openFolder`	Sync Now / Open Sync Folder
`settings.sync.pill.*`	Connected · Not connected · Sync failing · syncing... · retrying · synced {0} min ago
`status.sync.notSetUp`	Settings sync is not connected. Set it up in Settings ▸ Sync.

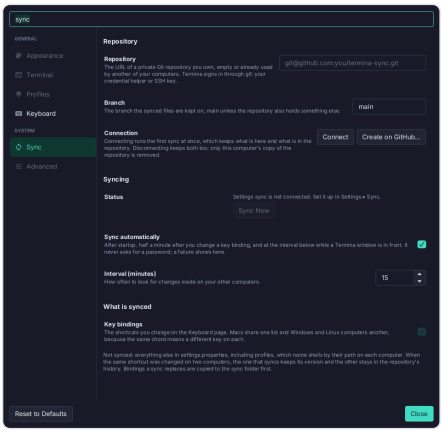
In the apps



Editora.



Torrenta.



Termina.

Packaging, signing and release

Every app ships the same way: a version tag on a commit whose tests passed builds the installers for five targets, each on its own runner, signs the macOS and Linux downloads, attests every one, and publishes them in one GitHub release, with Homebrew and winget packages where the app has them. The workflows that do it share one posture (actions pinned to a commit, the least permissions, a timeout on every job) and leave their results readable after the run. A user gets the same kind of download, checked the same way, for any app on any OS (principle 1).

Targets

Each target builds on its own GitHub-hosted runner. jpackage and JavaFX natives are host-specific, so there is no cross-building.

Target	Runner	Downloads
Linux x64	`ubuntu-latest`	`.deb`, `.rpm`, `.AppImage`, `.tar.gz`, fat jar
Linux arm64	`ubuntu-24.04-arm`	`.tar.gz` (and `.deb` where the app's tooling allows), fat jar
macOS arm64	`macos-15`	`.dmg`
macOS x64	`macos-15-intel`	`.dmg` (the last Intel image, good to about Aug 2027)
Windows x64	`windows-latest`	`.msi`, fat jar
Windows arm64	—	**Not possible yet**: OpenJFX publishes no `win-aarch64` natives (JDK-8314064). ARM users run the x64 MSI under emulation.

The fat jar is Editora's and the template's: only their release workflows build one (`-Pfatjar`), on the Linux and Windows legs. Termina, Torrenta, Mirada, Modula and nux release none.

- **REL-01** `MUST` jpackage builds the installers for five targets, Linux x64 and arm64, macOS arm64 and x64, and Windows x64, each on its own runner, through a Maven profile that packages the app (`dist` in Editora).
  - *Check (code):* the release workflow's build matrix has a leg for each of the five targets (`linux-x64`, `linux-arm64`, `macos-arm64`, `macos-x64` and `windows-x64` in Editora), each on a runner of that OS and architecture, and each leg builds with a profile of `pom.xml` that runs jpackage, whatever the profile is called. The runner images under Targets are Editora's: another image of the same OS and architecture meets it (Termina, Torrenta, Mirada, Modula and nux build macOS arm64 on `macos-latest`).
  - *Reference:* Editora `release.yml` (the `build` job), `pom.xml` (`dist`). The template and the five other apps name their profile `dist` too.
- **REL-02** `SHOULD` Downloads are named `<App>-<version>-<target>.<ext>`.
  - *Check (code):* the release-asset check builds the expected names this way.
  - *Reference:* Editora `scripts/release/CheckRelease.java` (`expectedAssets`).
- **REL-03** `MUST` **JReleaser** (`jreleaser.yml`) builds nothing: it creates the GitHub release, attaches every file and `checksums_sha256.txt`, and runs the packagers of REL-12 and REL-13 where the app has them.
  - *Check (code):* `jreleaser.yml` has no step that compiles or packages the app: it attaches the build matrix's files and the checksums file, and its `brew` and `winget` packagers, where the app has them, take those same files. The job that runs it needs the build jobs.
  - *Reference:* Editora `jreleaser.yml`, the `release` job in `release.yml`.
- **REL-04** `SHOULD` The release notes are the version's section of `CHANGELOG.md`.
  - *Check (code):* the release job gives that section to the step that publishes (JReleaser, by REL-03) as the release's notes.
  - *Reference:* Torrenta `scripts/release/ReleaseNotes.java`. Editora's release notes are still its commit list.
- **REL-05** `MUST` The Maven version is the single source of truth: a tag `vX.Y.Z` releases the version in `pom.xml`.
  - *Check (code):* the release workflow fails when the tag and the pom version differ.
  - *Reference:* Editora's "Check the tag against the pom version" step in `release.yml`.
- **REL-06** `SHOULD` The default branch carries a `-SNAPSHOT` version, and a post-release job opens the pull request that bumps it to the next snapshot.
  - *Check (code):* the pom's version on the default branch ends in `-SNAPSHOT`, and `release.yml` has the bump job.
  - *Reference:* Editora's `bump` job in `release.yml`.
- **REL-07** `SHOULD` A **release-asset check** fails the run before publishing when an expected file is missing.
  - *Why:* GitHub releases are immutable.
  - *Check (code):* a step before the one that publishes compares the gathered files with the expected list.
  - *Reference:* Editora's "Verify the expected release assets" step in `release.yml`.
- **REL-08** `SHOULD` On Linux, a `.desktop` file's `StartupWMClass` is the JavaFX app class.
  - *Why:* the running window then belongs to the menu entry and shows its icon.
  - *Check (code):* `packaging/linux/<App>.desktop` sets `StartupWMClass` to the fully qualified name of the app's own class that extends `Application`: `com.editora.App` in Editora, `com.modula.ModulaApp` in Modula.
  - *Reference:* Editora `packaging/linux/Editora.desktop`, Torrenta `packaging/linux/Torrenta.desktop`, Modula `packaging/linux/Modula.desktop`. The template, Termina, Mirada and nux set theirs the same way.
- **REL-09** `SHOULD` On Linux, AppStream metainfo has the id `io.github.adriandeleon.<App>`.
  - *Check (code):* `packaging/linux/io.github.adriandeleon.<App>.metainfo.xml` carries that `<id>`.
  - *Reference:* Torrenta `packaging/linux/`. Editora's metainfo id (`com.editora.Editora`) predates the rule.
- **REL-10** `SHOULD` Packaging trains an **AOT cache** (JDK 25 Leyden) to cut start-up time; Linux legs run the training under `xvfb`.
  - *Check (code):* the packaging build (Editora's `dist` profile) runs a training launch with `-XX:AOTCacheOutput`, and the Linux release legs install `xvfb`.
  - *Reference:* Editora `scripts/aot_build.java`; Termina's and Mirada's `scripts/aot-train.sh`.

Install methods

- **REL-11** `MUST` The GitHub release offers every download of every target: the `.dmg`, `.msi`, `.deb`, `.rpm`, `.AppImage` and `.tar.gz`, with checksums. Which target gets which type is under Targets.
  - *Check (code):* the files the release attaches include each of these types and a checksums file: the files `jreleaser.yml` lists or, in a workflow that publishes another way, the ones its publishing step uploads. That JReleaser is what publishes is REL-03.
  - *Reference:* Editora `jreleaser.yml`.
- **REL-12** `SHOULD` On macOS (Apple Silicon) the app installs with `brew install --cask adriandeleon/tap/<app>`: JReleaser's `brew` packager writes the cask to `adriandeleon/homebrew-tap`. A cask takes exactly one artifact, so it serves Apple Silicon, and Intel Macs use the `macos-x64` DMG.
  - *Why:* the official homebrew-cask repository has notability rules a new project does not meet, so the family shares one personal tap.
  - *Check (code):* `jreleaser.yml` has a `brew` packager for that tap; `jreleaser package --basedir .` writes the cask locally, to check it without releasing.
  - *Reference:* Torrenta and Termina `jreleaser.yml`; Editora has no packager yet.
- **REL-13** `SHOULD` On Windows x64 the app installs with `winget install AdrianDeLeon.<App>`: JReleaser's `winget` packager opens a pull request on `microsoft/winget-pkgs` from the `adriandeleon` fork. The package identifier is `AdrianDeLeon.<App>`, the moniker the app's lower-case name.
  - *Check (code):* `jreleaser.yml` has a `winget` packager with that identifier; `jreleaser package --basedir .` writes the manifests locally.
  - *Reference:* Torrenta and Termina `jreleaser.yml`; Editora has no packager yet.
- **REL-14** `SHOULD` The Homebrew and winget packagers (REL-12, REL-13) read one repository secret, a classic personal access token with `repo` scope (`PACKAGES_TOKEN` in Torrenta, Termina and the app template). Without it JReleaser skips both packagers, and the GitHub release still publishes.
  - *Why:* the workflow's own token cannot write to the tap or open a pull request on the fork.
  - *Check (code):* the release workflow passes a secret of the repository to JReleaser as the two packagers' token, whatever the secret is called, and leaves both packagers out when it is empty.
  - *Reference:* the publishing jobs of Torrenta's `release.yml`, Termina's `release.yml` and the app template's `release.yml`, which read `PACKAGES_TOKEN` and exclude the packagers without it.

winget is reviewed by Microsoft: the first submission for a new publisher is checked by a person, not merged automatically.

Status on 2026-11-06: the tap exists and holds no casks yet. Torrenta and Termina have both packagers configured, waiting for the token. Mirada's first winget submission (microsoft/winget-pkgs#471599) is open.

Signing

- **REL-15** `MUST` The macOS `.dmg` is signed with the Developer ID Application certificate (team 8JKSF797L8), with the hardened runtime and per-app entitlements, notarized with `notarytool`, stapled and checked with `spctl`; then the notarized app is launched once.
  - *Check (code):* `release.yml`, or a script it runs, imports `MACOS_CERT_P12` and `MACOS_CERT_PASSWORD`, notarizes with `APPLE_API_KEY_P8`, `APPLE_API_KEY_ID` and `APPLE_API_ISSUER`, staples, runs `spctl` and launches the app.
  - *Reference:* Editora's "Import signing certificate (macOS)" and "Notarize DMG (macOS)" steps in `release.yml`, `packaging/macos/Editora.entitlements`. All six apps sign, notarize, staple and run `spctl`; Editora, Termina, Mirada, Modula and nux then launch the notarized app. Torrenta does not yet: its "Smoke-test the packaged app" step launches the signed app before it is notarized, and its "Notarize the dmg (macOS)" step in `release.yml` ends at `spctl`. Mirada's steps are in `scripts/sign-mac.sh` and `scripts/notarize-mac.sh`.
- **REL-16** `MUST` Linux downloads are signed with the one shared OpenPGP release key ("Adrian De Leon (release signing)") in a job of their own, apart from the build legs (`sign-linux` in Editora): `rpmsign` for the `.rpm`, a debsigs origin signature for the `.deb`, and an armored `.asc` beside every file. The public key is committed as `packaging/linux/signing-key.asc` and attached to each release.
  - *Check (code):* the release workflow has a job apart from the build legs, whatever it is called, that reads the release key and its passphrase from the repository's secrets (`LINUX_GPG_KEY` and `LINUX_GPG_PASSPHRASE` in Editora), signs each `.rpm` and `.deb` and writes an `.asc` beside every Linux file; and `packaging/linux/signing-key.asc` exists, and the release attaches that public key (as `editora-signing-key.asc` in Editora).
  - *Reference:* Editora `scripts/release/sign-linux-packages.sh`, `scripts/release/check-linux-signatures.sh`. Done in Editora, the app template, Torrenta, Termina, Mirada and Modula, all with the same key; nux has no `sign-linux` job and no committed key yet.
- **REL-17** `SHOULD` The Windows `.msi` is signed with Authenticode, once the certificate is chosen.
  - *Check (code):* the Windows leg signs the `.msi` before it is uploaded.
  - *Reference:* no app signs one yet. The app template's `release.yml` has the step: once the repository holds a certificate (`WINDOWS_CERT_PFX`), it signs each `.msi` with `signtool` before the upload. **Open decision**: SignPath Foundation (free for OSS), a Certum Open Source certificate, or Azure Trusted Signing. Until then MSIs are unsigned and SmartScreen warns.
- **REL-18** `MUST` Every download has a GitHub **build provenance attestation** (`actions/attest-build-provenance`), which `gh attestation verify` checks.
  - *Check (code):* the release workflow has an `actions/attest-build-provenance` step whose subjects cover every file the release attaches. The test that keeps the two lists the same is QUAL-15's.
  - *Reference:* Editora's "Attest build provenance" step in `release.yml`, `ReleaseSupplyChainTest`. Done in Editora, the app template, Termina and Mirada.

The certificate renews before 2027-11-30; the Apple membership before 2027-10-31.

CI/CD (GitHub Actions)

- **REL-19** `MUST` The CI workflow (`ci.yml` in Editora) runs on every push to the default branch and every pull request, with two checks: the format check of PLAT-10 (`spotless:check`; the job Editora names **Format**) and the build with its tests (`./mvnw verify`, coverage gate included; Editora's **Build & Test**), on the JDK PLAT-01 names. Both are required status checks for the default branch.
  - *Check (code):* the CI workflow triggers on `push` to the default branch and on `pull_request`. One of its jobs is the Format job of PLAT-10, which runs `spotless:check` and neither builds nor tests, and its jobs between them run the whole suite and `verify` with the coverage gate: one job that runs `./mvnw verify`, or, as in Editora, jobs that run `test` in shards and one that runs `verify` over their results. What the jobs are called is the app's own. A workflow that lacks a trigger or one of the two checks fails the rule. That the default branch requires both is a repository setting, which no file shows. An audit run from the app's checkout alone reports the rule as `not-checked` and says what would settle it.
  - *Reference:* Editora `ci.yml`, on `master` (its `format` job, its four `tests` shards and its `build` job, which runs `verify` over the shards' results); the app template's `ci.yml` and Termina's `ci.yml`, on `master`, and Mirada's `ci.yml`, on `main`, each with a job for `spotless:check` and one for `verify`. Torrenta (`ci.yml`), Modula (`build.yml`)

and nux (`ci.yml`) don't yet: none has a format check of its own. Torrenta's and Modula's one test job runs `./mvnw verify` , which their poms bind `spotless:check` to, so a single status stands for both checks; nux's pom has no Spotless, and its job runs `./mvnw test` , not `verify` . Modula's and nux's builds have no coverage gate either (PLAT-12). No file of any of them shows the required checks.

- **REL-20** `MUST` The tests run on all three OSes: the pure suite (no JavaFX toolkit) on Linux, macOS and Windows, and the FX suite on Linux and at least one more. They run there every day or on every pull request. How the runs are split is the app's own: one workflow that runs everything on every pull request meets this, and so does a quick workflow for pull requests with a fuller one on a schedule. A release waits for these runs on its own commit, on all three OSes: that it waits at all is REL-30.
  - *Why:* path separators, line endings, case-insensitive file systems and process spawning fail on one OS only, and the pure suite is where they show; the FX suite catches layout and focus differences. The FX suite is the slow one (three shards on macOS in Editora), so it is asked on two OSes and not three: the third is covered by the pure suite alone.
  - *Check (code):* taken together, the workflows' test jobs run the pure suite on `ubuntu-*` , `macos-*` and `windows-*` runners and the FX suite on `ubuntu-*` and one more, under a `schedule` or a `pull_request` trigger that needs no label; and the tests a release waits for (REL-30) ran on those runners, in the release workflow itself or in the run it reads.
  - *Reference:* Editora splits them: `ci.yml` runs the whole suite on Linux on every pull request, and `full-tests.yml` the pure suite on Windows and macOS and the FX suite on macOS, in three shards, daily, on a pull request labeled `full-ci` and before every release: the "Check that the Full tests passed on this commit" step in `release.yml` . It lists flaky tests in the job summary (`scripts/ci/ReportFlakes.java`). The app template splits them the same way, and its release calls `full-tests.yml` itself. Termina (`ci.yml`), Torrenta (`ci.yml`), Mirada (`ci.yml`), Modula (`build.yml`) and nux (`ci.yml`) run the whole suite on the three OSes on every pull request.
- **REL-21** `MUST` The release workflow (`release.yml` in Editora) runs on a version tag (`v*` meets it; the version-shaped `v[0-9]+.[0-9]+.[0-9]+*` is the better pattern, since a tag like `vendor-import` doesn't match it), with `workflow_dispatch` as a dry run that builds and signs and publishes nothing. The tag matching the pom is REL-05, the tests it waits for REL-30, the release-asset check REL-07, and the bump to the next snapshot REL-06.
  - *Check (code):* the release workflow triggers on a tag pattern that starts with `v` and on `workflow_dispatch` . A dispatched run goes through the build and signing jobs and publishes nothing: the step that publishes runs in a mode that publishes nothing (JReleaser's `--dry-run`), or the job that publishes runs only for a tag (an `if:` on the ref).
  - *Reference:* Editora `release.yml` , whose "JReleaser" step takes `release --dry-run` on a dispatch; the template, Termina, Torrenta and Mirada pass JReleaser `--dry-run` the same way. Modula's `release.yml` and nux's `release.yml` , which have no JReleaser, run their publishing job only for a tag (`if: startsWith(github.ref, 'refs/tags/…')`).
- **REL-30** `MUST` A release is cut only from a commit whose tests passed: the release workflow runs the suite before it builds the installers, or a preflight job reads CI's result for the tagged commit and stops when it is not green.
  - *Why:* the build legs often skip the tests (`-DskipTests`), and a tag can be pushed on any commit, so without this a release can be cut from code whose checks failed or never ran.
  - *Check (code):* in the release workflow, every job that publishes needs, directly or through the jobs between, a job that settles the tests on the tagged commit: it runs them (a Maven run that reaches the `test` phase, `./mvnw test` , `package` , `verify` or `install` , with no `-DskipTests` and no `-Dmaven.test.skip` , or a call to the full-tests workflow), or it reads the conclusion of the CI run on `GITHUB_SHA` and fails on anything but success.
  - *Reference:* both ways are in use, and all seven release workflows meet it. A preflight that reads CI: Editora's "Check that CI passed on this commit" step in `release.yml` (`scripts/release/CheckRelease.java` `ci` , which waits up to 30 minutes for the required checks; its build legs run `-DskipTests`), and Torrenta's "Require CI on this commit" step in `release.yml` , which reads the newest `ci.yml` run on the commit and starts one when there is none. The suite in the release: the template's `full-tests` job in `release.yml` calls `full-tests.yml` before `build` ; the "Build, test and package" step of Termina's `release.yml` and Mirada's `release.yml` runs `./mvnw … verify` on every build leg; Modula's "Build the installer" step in `release.yml` runs `package` with the tests; nux's build legs run a "Test" step (`./mvnw test`) before they package, in `release.yml` . In each of the five the job that publishes needs the build legs, directly or through the signing and asset-check jobs. Mirada's `codecs.yml` also creates a GitHub release, of the codec libraries its build downloads by checksum: that is a dependency, not a release of the app, and the app's own release runs the suite with them.
- **REL-22** `MUST` Every workflow pins each third-party action to a commit SHA, with its version in a comment (`actions/setup-java@b843ebf… # v6.1.0`).
  - *Check (code):* every `uses:` in `.github/workflows/` , and in a composite action under `.github/actions/` where the repository has one, ends in a 40-character SHA and a comment naming its version (`# v6.1.0` , `# 2.5.0` , `# v2`), never a branch such as `# main` . The `v2` in Editora's and Torrenta's `jreleaser/release-action … # v2` names that action's `v2` line, whose head the SHA is; the template, Termina and Mirada pin its 2.5.0 release.
  - *Command:* `[-f pom.xml] || { echo 'not an app root: no pom.xml here'; exit 1; }; [-d .github/workflows] || { echo 'no .github/workflows'; exit 1; }; ! grep -rhE '^[[:space:]]*-?[[:space:]]*uses:' .github/workflows .github/actions 2>/dev/null | grep -vE 'uses:[[:space:]]*\./' | grep -vE '@[0-9a-f]{40}[[:space:]]+#[[:space:]]*v?[0-9]'`
  - *Reference:* Editora `.github/workflows/` .
- **REL-23** `MUST` Dependabot updates `github-actions` and `maven` weekly.
  - *Check (code):* `.github/dependabot.yml` lists both ecosystems with `interval: weekly` .
  - *Reference:* Editora `.github/dependabot.yml` .
- **REL-24** `MUST` Every workflow starts from `permissions: contents: read` , or from less; only a job that needs write access asks for it, in its own block.
  - *Check (code):* each workflow has a top-level `permissions` that grants no write: `contents: read` , or less (`permissions: {}`); and every `write` sits in a job's own `permissions` .
  - *Reference:* Editora `release.yml` : `release` (publish), `bump` (the pull request) and `preflight` (start a Full tests run). Modula's `notify-catalog.yml` starts from `permissions: {}` .
- **REL-25** `SHOULD` Every `actions/setup-java` step caches Maven's repository (`cache: maven`). The JDK it sets up is PLAT-01's.
  - *Check (code):* each `setup-java` step has `cache: maven` .
  - *Reference:* Mirada (`ci.yml`), Modula, nux and the app template (`ci.yml` , `full-tests.yml` , `release.yml`), on every step. Editora (`ci.yml` , `release.yml`), Termina and Torrenta each have a `setup-java` step without it. Editora sets up Temurin for its tests and Liberica for the release build, for linux-aarch64 coverage.
- **REL-26** `MUST` Every job that runs steps has `timeout-minutes` . A job that calls a reusable workflow (`uses:` in place of `steps:`) takes no `timeout-minutes` : GitHub does not accept the key there, and the called workflow's own jobs carry theirs.
  - *Check (code):* in every workflow, every job with `steps:` sets `timeout-minutes` on the job itself; a timeout on one of its steps does not stand for it. A job with `uses:` and no `steps:` is not judged.
  - *Reference:* every job of Editora's five workflows (`ci.yml` , `full-tests.yml` , `release.yml` and the rest), of the template's three, Termina's two and Mirada's five sets one; the template's `full-tests` job in `release.yml` , which calls `full-tests.yml` , carries none, as the rule says, and nor does the `winget` job of Mirada's `release.yml` , which calls `winget.yml` . Torrenta, Modula and nux don't yet: the two jobs of Torrenta's `ci.yml` and the one of its `engine-soak.yml` set none (its `release.yml` does); no job in Modula's three workflows (`build.yml` , `release.yml` , `notify-catalog.yml`) or in nux's two (`ci.yml` , `release.yml`) sets one: each app has it on one step only, the one that notarizes.
- **REL-31** `MUST` Every workflow that runs on pull requests has a `concurrency` group that cancels a superseded pull-request run, and never a run on the default branch.
  - *Check (code):* every workflow with a `pull_request` trigger has a `concurrency` block whose `group` is shared by the runs of one pull request and by no run on the default branch (keyed by `github.ref` , or by the pull request's number), and whose `cancel-in-progress` is true for a pull-request run: true, or an expression that is true there. And

in that workflow `cancel-in-progress` is never true for a run on the default branch, whatever starts it (a push, a schedule, or a run started by hand, which `workflow_dispatch` allows on any branch): it is an expression that is false there, and it may be plain true only in a workflow that nothing can start on the default branch. A workflow with no `pull_request` trigger is not judged.

  - *Reference:* Editora `ci.yml` and `full-tests.yml` (`cancel-in-progress: ${{ github.event_name == 'pull_request' }}`, in a group keyed by the pull request's number). The template's (`ci.yml`, `full-tests.yml`), Termina's (`ci.yml`) and Mirada's (`ci.yml`) pull-request workflows use the same expression, Termina's and Mirada's in a group keyed by `github.ref`. Editora's third pull-request workflow, `terminal-image.yml`, doesn't yet: its `cancel-in-progress` is plain true, and beside pull requests and pushes to `probe/**` it can be started by hand, so a second run started on `master` cancels the first. Torrenta, Modula and nux don't either: Modula's `build.yml` and nux's `ci.yml` run on pull requests with no `concurrency` block, so they cancel nothing, and Torrenta's `ci.yml` sets `cancel-in-progress: true` for its pushes to `master` as well.

- **REL-27** `MUST` A CI run leaves its results where they can be read without running it again: it uploads the surefire reports even when tests fail, uploads the JaCoCo report, and writes the coverage totals to the job summary.
  - *Check (code):* the step that uploads the surefire reports runs under `if: always()`, a step uploads the JaCoCo report, and a step appends the coverage totals to `$GITHUB_STEP_SUMMARY`.
  - *Reference:* Editora's "Upload test reports and coverage data" and "Summarise coverage" steps in `ci.yml`.
- **REL-28** `MUST` Signing skips cleanly when the secrets are absent (forks, dry runs), so they still build; a tag without them fails.
  - *Check (code):* each signing step checks its secrets first, skips without them, and fails on a tag push.
  - *Reference:* Editora's "Import signing certificate (macOS)" step in `release.yml` checks `MACOS_CERT_P12` first.
- **REL-29** `MUST` The packaged app is launched once on each OS before a release (a smoke test).
  - *Check (code):* the release workflow runs the packaged launcher on each OS (a Linux, a macOS and a Windows leg) before the step that publishes.
  - *Reference:* Torrenta's "Smoke-test the packaged app" step in `release.yml` runs the launcher in the image of each of its three targets, one per OS. Editora runs only the notarized macOS app (`--version`, `--check-terminal`).

Web version: https://pauta-project.dev/. Source: github.com/adriandeleon/adl-desktop-apps-guidelines. Version 6.1.0, 2026-11-14.