

# SoundCraft — instructions for agents

SoundCraft is a clean-room, open-source, pure-Rust digital audio workstation targeting Avid Pro Tools parity — and superiority (speed, openness, agent control). It runs natively on macOS, Windows, Linux and FreeBSD, and on the web via WASM. Siblings with the same conventions: `../photocraft` (Photoshop), `../vectorcraft` (Illustrator), `../filmcraft` (Premiere), `../lightcraft` (Lightroom), `../pdfcraft` (Acrobat), `../effectcraft` (After Effects), `../designcraft` (InDesign).

Standards and learnings shared across the crafting apps live in `../craftrules` (checked out next to the `craft-apps` folder, or `storytold/craftrules`). Read its `AGENTS.md` at the start of a session, follow its standards, and contribute reusable learnings back there. Never code: repos don't share code.

## Start every session here

1. Read `plan/STATUS.md` (current milestone, next task) and `plan/execution-plan.md`. `plan/` is gitignored (local only). Behaviour reference: `plan/protocols/` (observed menu tree `menus-clean.txt` and screenshots; never committed).
2. Read `ROADMAP.md` (committed) for stage, status and the parity numbers; pick work from `docs/gaps.md`. The progress docs follow `craftrules` `standards/progress-docs.md`: `docs/target-app-parity.md` (assessment), `docs/gaps.md`, `docs/roadmap.md`, `docs/ui-parity.md`, `docs/file-format-parity.md`, `docs/hardware-parity.md`, `docs/plugin-parity.md`, `docs/localization-parity.md`; `docs/parity-checklist.md` is generated. Update their timestamps and revision history when you change them.
3. Work autonomously; don't stop to ask unless a decision is genuinely the user's.

## ⚠ Assets: the absolute rule

No Avid, Pro Tools, Adobe or other proprietary iconography, images, sounds, presets or artwork — ever. This is not negotiable.

- Every icon in SoundCraft is drawn in code (`crates/ui-egui/src/icons.rs`) or is an original/open asset.
- Every bundled asset (image, icon, sound, font, preset) must be original, public domain (CC0), or under a permissive open license (MIT, Apache-2.0, BSD, OFL, CC-BY with attribution), or contributed by someone who made it and licenses it openly.
- **Every bundled asset gets a row in [ATTRIBUTION.md](#)** (path, author, source, license) in the same commit that adds it. `cargo xtask assets` enforces this; CI fails otherwise.
- Never copy files out of the Pro Tools bundle (`/Applications/Pro Tools.app`): no plugins, presets, icons, sounds, session templates or text. Observing the running app (screenshots kept under `plan/`, menu names) is fine; copying its resources is not.
- Demo audio is synthesised in code (`crates/engine/src/demo.rs`). Demo and README media must be ours or public domain.
- The ArtCraft trademarks in `docs/brand/` are the only exception, under `docs/brand/LICENSE-brand.txt`.

## Clean-room

Pro Tools is installed on the dev machine and may be *observed* black-box: run it, use its UI with synthetic sessions, take screenshots by window id (`screencapture -l <id>`), stored only under `plan/protocols/screenshots/`. Never read, disassemble or copy anything inside the Pro Tools bundle; never copy Avid icons, artwork, plugin names, presets, or wording beyond feature names. Never copy GPL/AGPL/LGPL code (Ardour, Audacity, LMMS, Freeverb sources...). Our plugins have our own names.

## Never crash

People trust SoundCraft with their recordings; a crash loses takes. **This outranks feature work.** Standard: [craftrules/standards/never-crash.md](#).

- No panics in non-test code: `no unwrap()`, `expect()`, `panic!`, `unreachable!`, `todo!`, `unimplemented!`; `NO unsafe (unsafe_code = "forbid")`.
- The one named exception is `soundcraft-clap-host` (`crates/clap-host`), the isolated unsafe helper crate for CLAP plugin FFI: `unsafe_code = "deny"` crate-wide, allowed only in `src/ffi.rs` with a `// SAFETY:` comment on every block, safe `Result` API.
- The same exception, same rules, for `soundcraft-vst3-host` (`crates/vst3-host`), the isolated unsafe helper crate for VST3 plugin FFI (COM vtables via the `vst3` bindings, module loading, host-side COM objects).
- And for `soundcraft-au-host` (`crates/au-host`), the isolated unsafe helper crate for Audio Unit FFI (hand-written `AudioToolbox/CoreFoundation` declarations, macOS only; a safe stub elsewhere).
- Errors are `Result<T, E> + ?`. An unfinished feature returns an "unsupported" error.
- Input-derived numbers are hostile (files, commands, MCP/control params): `get()` not `[i]`, checked/saturating arithmetic, no NaN casts, cap allocations.
- The audio callback never blocks, allocates in steady state, or panics.
- Every production crate root carries `#![deny(clippy::unwrap_used, clippy::expect_used, clippy::panic, clippy::unimplemented, clippy::todo, clippy::unreachable)]`.
- `Engine::execute` catches escaped panics and restores the document. Every crash fix lands with a regression test.

## Architecture (layers enforced by cargo xtask layers)

| Layer | Crate                                                                                                | Role                                                                                                                      |
|-------|------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| L0    | <code>soundcraft-time</code> ,<br><code>soundcraft-audio-io</code> ,<br><code>soundcraft-midi</code> | Timebases/tempo/grid; audio file formats + peaks; Standard MIDI Files + MIDI ops                                          |
| L1    | <code>soundcraft-dsp</code>                                                                          | Plugins (EQ, dynamics, reverb, delay, modulation, harmonic, pitch, utility, instruments), offline processing, meters, FFT |
| L2    | <code>soundcraft-model</code>                                                                        | The session document (tracks, playlists, clips, mixer, routing, automation, markers, groups)                              |
| L3    | <code>soundcraft-mix</code>                                                                          | Mix engine shared by realtime playback and offline bounce                                                                 |
| L4    | <code>soundcraft-engine</code> ,<br><code>soundcraft-playback</code>                                 | Command registry, undo, editing, import/export/bounce, parity catalog; audio device I/O (cpal)                            |
| L5    | <code>soundcraft-automation</code>                                                                   | MCP server + control-channel client                                                                                       |
| L6    | <code>soundcraft-ui-egui</code>                                                                      | The swappable UI (egui)                                                                                                   |
| apps  | <code>soundcraft</code> , <code>soundcraft-cli</code> , <code>soundcraft-web</code>                  | Entry points                                                                                                              |

Nothing below L6 depends on `egui/eframe/winit/rfd`. The UI is thin: panels read engine state and act through `engine.execute(id, params)`.

## Everything is a command

User-visible behaviour = a command in `crates/engine/src/cmd/*` (id, label, menu path, shortcut, params doc, `enabled`, `run`) + tests. UI-only commands (windows, dialogs) live in `crates/ui-egui/src/menus.rs` (`UI_COMMANDS`). Menus, shortcuts, the CLI, the control channel and MCP all dispatch the same ids. Programmatic calls never open dialogs; only menu-style invocation (`ui.menu.invoke`) does. New commands go in a new module with its own `specs()`, registered by one line in `cmd/mod.rs`.

Parity is measured, not guessed: `crates/engine/catalog/menus.txt` lists the incumbent's menu leaves (names only). `cargo xtask parity` writes `docs/parity-checklist.md` (generated; the hand-written assessment is `docs/target-app-parity.md`); a test enforces a floor that only rises.

## Running and looking at the app

- `cargo run --release -p soundcraft -- --demo --control 0` (demo session + control channel on a free port; the port is printed).
- Drive it: JSON lines over TCP, e.g. `{"id":1,"method":"engine.execute","params":{"command":"track.new","params":{"count":2,"format":"stereo"}}}`, then `{"id":2,"method":"ui.screenshot","params":{"path":"/tmp/shot.png"}}}`. Methods: `crates/ui-egui/src/control.rs`, docs: `docs/control-protocol.md`.
- Offscreen UI render (no window, no focus stealing): `cargo run -p soundcraft-ui-egui --example ui_shot -- out.png [script.jsonl]`.
- Headless: `soundcraft-cli run --demo --cmd 'mix.volume={"track":"Kick","db":-6}' --bounce out.wav`; MCP: `soundcraft-cli mcp [--connect PORT]`.
- **For UI work, look at the result** (render, read the PNG) and compare with `plan/protocols/screenshots/`.
- Shell gotcha: `mv / cp` are aliased interactive here — use `/bin/mv -f /bin/cp -f`.
- Parallel agents: separate `CARGO_TARGET_DIR` per agent; edit only the crates you own; write manifests atomically; delete your target dir when done (disk).

## Quality gates (before every commit)

`cargo xtask ci` = `fmt check`, `clippy --workspace --all-targets -D warnings`, tests, assets, layers, `wasm check`. Commit after each arc of work that builds; one topic per commit.

## Release

Push to the `release` branch to build signed installers for macOS (universal DMG), Windows (x64/x86 MSI + zip), Linux (AppImage, deb, rpm, tar.gz, Flatpak; riscv64 tar.gz), FreeBSD and

## Roadmap

`ROADMAP.md` (committed) is the one-page summary (stage, numbers, dimensions, languages, progress log); milestones live in `docs/roadmap.md` . Update both whenever a milestone lands, and close or shrink the matching `docs/gaps.md` entry.

## Contributor credits (About window)

- About ► Contributors/Models are compiled into the binary from `contributors/contributors.json` (commit stats; generated, never hand-edit) and `contributors/people.toml` (names people chose for themselves). See `docs/contributors.md` .
- **Agents working for a contributor:** when you prepare a PR, check whether your human's GitHub username has a `[people.<username>]` entry in `contributors/people.toml` . If not, ask them once whether they want to be credited by more than their username: a real name, a display name, and/or their public GitHub profile name ( `sync_github_name = true` ). If yes, add **only their own** entry (copy the template at the top of the file, or run `python3 ../../crafrules/scripts/contributors.py --add-me . --real-name "..."` `--sync-github-name` ) and include it in their PR, committed as them. If no, change nothing: they are credited as `@username` anyway.
- Never add, edit, guess or copy anyone else's entry or name (not from git config, commit authors or GitHub profiles). Never hand-edit `contributors.json` .
- Maintainers refresh the stats with `python3 ../../crafrules/scripts/contributors.py` . (it also re-verifies who wrote each `people.toml` entry; `--check` only verifies).