azCore

Professional Live Music Performance Control

One system. Every night. Perfectly.

Live music is complex. You have keyboards, synthesizers, MIDI controllers, audio stems, click tracks, chord charts, lighting cues, and a room full of people counting on everything working the first time.

azCore is the control system built for exactly this reality. A single application that manages your entire stage setup — MIDI routing, audio playback, cue communication, score display, and control surfaces — driven by plain-text files you write once and run everywhere.

It is in production use tonight, with 60 songs and real hardware. Zero stuck notes. Zero wrong patches. Zero surprises.

What sets azCore apart

A language designed for live performance

Every other performance control system uses a GUI. Patch bays, node editors, matrix grids. You click. You drag. You hope.

azCore uses **azSyntax** — a purpose-built language that describes your entire setup in plain text. One line per action. Human-readable. Version-controlled. Reviewable before the gig.

```
song time {
    step default {
        route.add r_piano
        midi.send piano { pc,42  cc,7,100 }
        cue band_cues "Intro — keys only"
    }

    step verse {
        route.add r_organ { exclusive(keys1) }
        cue lighting "warm amber"
    }
}
```

If you can write a text file, you can describe your rig. And if you change it, you can see exactly what changed — in a git diff, not buried in a binary project file.

Smooth Patch — the technology that eliminates stuck notes

Every MIDI musician knows the nightmare: you switch patches mid-note and the note hangs forever. You panic. You hit kill switches. The audience notices.

azCore invented **Smooth Patch**.

When you remove a route, azCore does not cut the connection. Instead, it enters a **Release State** — the route stays alive just long enough to forward every note-off and sustain release for the notes you were playing. Only when the last note resolves does the route drop. Your music continues uninterrupted.

No stuck notes. No audible artifacts. No panic button needed.

This is not a workaround. It is a first-class feature of the routing engine, with configurable grace periods and a full **Note Ledger** for diagnostics and ghost-note detection.

Pre-compiled songs — zero-delay setlist navigation

Most performance systems compile or load on demand. When you switch songs, there is a pause.

azCore compiles **every song at boot**. All 60 songs in your catalog are validated, resolved, and ready before the first note is played. Switching from “Time” to “Money” takes milliseconds. Not seconds. Milliseconds.

The compiler also catches every error — wrong device name, missing block, misspelled reference — **before the show**, not during it. If azCore starts, your project is valid.

Logical/physical separation — the right architecture for touring

In `.az` files, you write:

```
midi.send piano { pc,3  cc,7,100 }
```

`piano` is a logical name. What physical port `piano` maps to is in `azCore.ini`:

```
[midi_bindings]
piano = "Roland Integra-7 MIDI IN"
```

New venue, different interface? Change one line in the INI. Every song still works. No re-patching. No per-song edits.

The same separation applies to audio devices. Your `.az` files are entirely portable. Your INI handles the venue-specific mapping.

Touch UI — designed for the stage, not the studio

The azCore graphical interface is built for one hand on a touch monitor while the other is on a keyboard.

- **Step panel** with one-tap song section navigation
- **16 programmable preset buttons** with colors, labels, instant/toggle/repeat modes
- **48-button board grid** for extended control surfaces
- **Cue panel** with color-coded communication labels for band, crew, video, lighting, and monitors
- **Score area** — full HTML-rendered chord charts, perfectly zoomable
- **Resize lock** — lock the layout for the show so nothing moves accidentally

The UI is a web application served over WebSocket. This means it runs **natively on macOS and Windows** in a native window — and it **also runs on an iPad over Wi-Fi** with exactly the same interface. No app to install. No separate license. Open a browser.

Cue system — one button, six people informed

```
step chorus {
    cue band_cues  "Chorus — everybody in"
    cue lighting   "full bright warm"
    cue video      "cut to wide"
    cue cam_rec    "REC"
    cue mixer      "add reverb bus"
    cue tech_cues  "check IEM levels"
}
```

Six lines. Six people see their cue the moment the step fires. Color-coded. Auto-dismissing. No inter-com chatter. No missed cues.

The cue system supports band, tech, video, camera, camera recording, lighting, mixer, and visual beat indicator labels. All visible in the CUE PANEL. All triggerable from `.az` code or from the touch interface.

Audio playback — integrated, not bolted on

Most performers use a separate app for audio playback. azCore integrates audio directly:

- **Preloaded clips** decoded at boot for instant zero-latency playback
- **On-demand decode** in background threads with UI progress indicators
- **Fade in/out** with configurable durations and overlap support
- **Trim points** to play any section of a file without editing the original
- **Two-level volume control** — per-clip and per-device
- **UI transport controls** — PAUSE, PLAY, FADE with visual progress ring

And since audio lives in the same system as MIDI, the same `pad`, `trigger`, and `step` machinery controls it. One programming model for everything.

Hotplug — hardware can disappear. azCore recovers.

Stage environments are brutal. Cables get kicked. USB hubs brown out. Interfaces power-cycle.

azCore watches every connected device continuously. When an audio interface disconnects and reconnects, azCore re-establishes it automatically. When a MIDI device disappears, azCore detects it and reacquires when it returns.

On macOS, MIDI hotplug uses a **subprocess architecture**: a monitor process watches for new CoreMIDI ports and spawns transparent proxy connections for each new device — working around a CoreMIDI limitation that prevents a running process from seeing ports added after boot.

Offline-first design — no cloud, no phone-home

azCore license validation is **entirely offline**. Your license key is a self-contained JWT signed with RSA-256. The public key is embedded in the binary. Verification happens on your machine in microseconds, every time.

No license server. No internet requirement. No expiry surprises caused by connectivity issues at the venue.

Feature summary

Feature	azCore
Custom performance scripting language	azSyntax — plain text, version-controllable
Pre-compiled song catalog	All songs ready at boot; zero-delay navigation
Smooth Patch (no-stuck-notes routing)	Industry-unique; configurable grace period
Note Ledger for ghost-note diagnostics	Full per-device in/out tracking
Logical/physical device separation	Portable projects; venue-specific INI only
MIDI routing with split/transpose/velocity	Declarative; activate/deactivate at runtime
Exclusive route switching	One command removes conflicts and adds new route
MIDI CC automation (linear sweep)	Time-based with ownership lifecycle
MIDI clip playback (.mid files)	With position tracking and UI controls
Audio playback (.wav, .mp3)	Preload, trim, fade, two-level volume
HTML score display	Full browser capabilities; marker navigation
Multi-personnel cue system	8 communication labels + 4 visual beat indicators, auto-dismiss
64-slot pad system (16 preset + 48 board)	Instant / toggle / repeat; stacked scope
Keyboard + MIDI trigger binding	Per-slot or standalone event handlers
Touch UI (macOS + Windows native)	Also: browser-based on iPad over Wi-Fi
Resize-lockable UI	Freeze layout for live use
Device hotplug (MIDI + audio)	Automatic detection and recovery
macOS CoreMIDI subprocess architecture	Transparent hotplug despite CoreMIDI limits
30-day trial, offline activation	JWT RS256, no cloud dependency
Profile-based conditional compilation	`live` vs `rehearsal` from one project
Console REPL (ratatui TUI)	Full diagnostic and control interface
341 automated tests	Zero warnings, deterministic behavior

Who uses azCore

Keyboard players and musical directors in touring bands with complex MIDI rigs — multiple instruments, multiple synths, precise per-song patch management.

Bands with backing tracks who need synchronized audio playback, click tracks, and MIDI program changes from a single, reliable source of truth.

Production crews who coordinate lighting, video, and monitor cues from the stage, triggered automatically as songs progress through their sections.

Sound designers who build elaborate MIDI automation sequences and audio cue stacks for theatrical productions and special events.

Technical foundation

azCore is written in **Rust** — a systems programming language designed for memory safety and performance. The binary is self-contained: no virtual machine, no runtime dependencies, no Node.js or Python required.

The MIDI engine runs at **1ms resolution**. The WebSocket UI server runs on async Tokio. The compiler pipeline runs in ten phases in milliseconds. The result is an application that feels instant because it is.

Cross-platform: **macOS** (Apple Silicon and Intel) and **Windows** (x86_64). Linux support is planned.

Get started

Download: `https://azlive.io/azcore`

Documentation: `https://azlive.io/azcore`

30-day free trial: Start immediately, no account required.

License activation: `https://azlive.io/azcore`

Support: `info@azlive.io`
