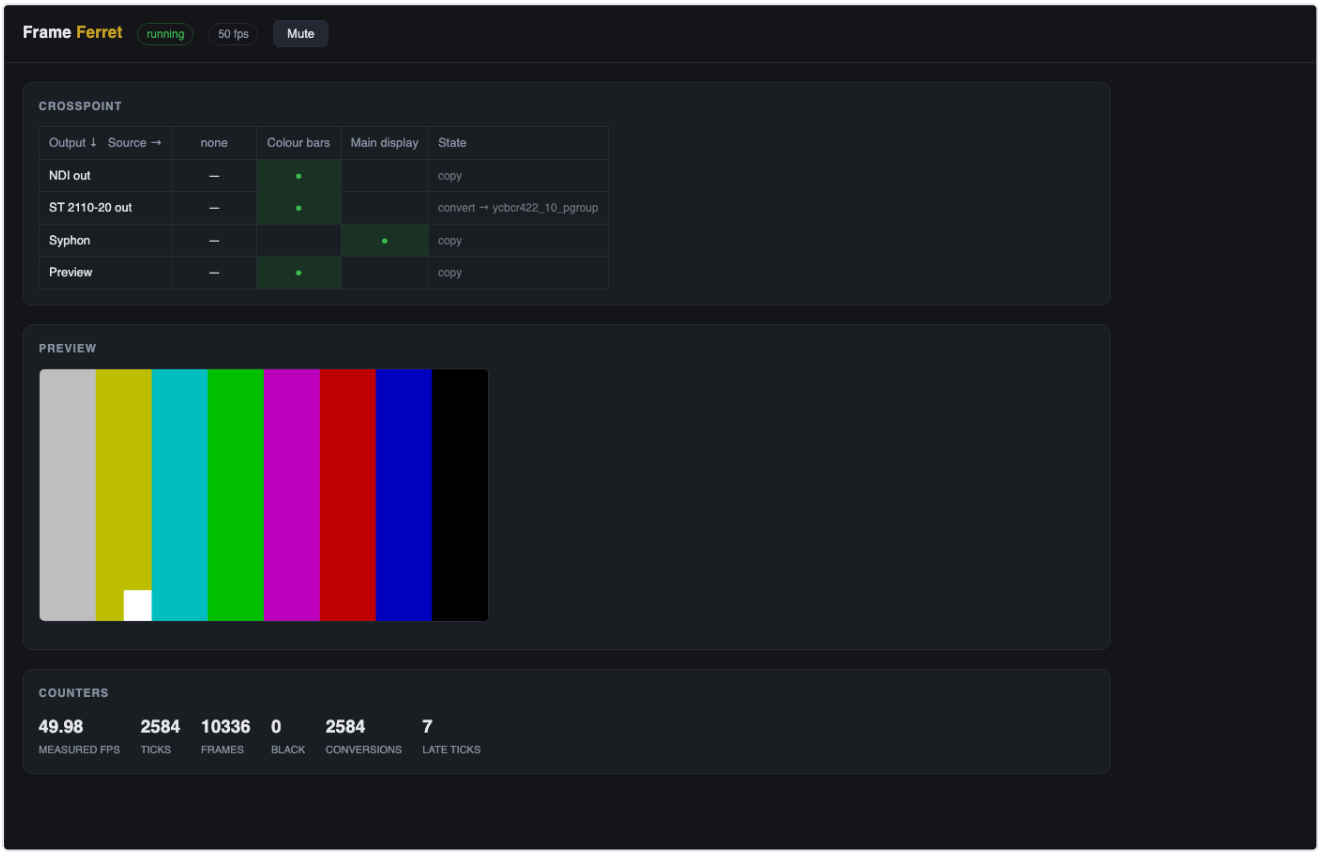
Frame Ferret user guide

Frame Ferret is **a software virtual capture card**. One application that is an NDI, OMT, SRT and ST 2110 endpoint — transmitting and receiving — over a chosen network interface, that can capture this machine's screen, and that presents itself to other software as a camera, a shared GPU surface, an SDI output or a web overlay.



The control page at 50 fps: colour bars routed to the NDI, ST 2110-20 and preview outputs and the main display to Syphon — the ST 2110 row converting to 4:2:2 10-bit on the way — with the preview and the tick, frame and conversion counters underneath.

The idea is that a protocol is just a port on a router. Anything can be a source, anything can be a sink, and the crosspoint in the middle does not care which is which.

> **Before you rely on this:** a great deal works and a great deal is not built yet, and the difference matters more here than in most projects. **NDI and OMT both work in both directions**, verified against separate implementations. **SRT send and receive are verified end to end**, and ST 2110-20 is verified against GStreamer's implementation in both directions. **DeckLink output has run on a real Duo 2** — 2531 frames with zero late and zero dropped by the card's own count, though the pixels on the wire have not been checked.
>
> **Not started:** the UVC virtual camera, the virtual display, DeckLink capture, Spout, HTML output, the tray launcher, and ST 2110 audio and ancillary. Windows and Linux build and self-test in CI but **have never been run interactively, and no hardware path exists on either.**
>
> This codebase was created with AI assistance, directed and reviewed by a human author.

The invariant

Every sink produces a frame, every tick. A sink with nothing routed to it goes black and **keeps running** rather than stopping.

That has been observed rather than asserted: over one 9049-tick run, 4507 frames delivered plus 4542 black is exactly 9049 — one action per sink per tick, **zero late ticks, 49.99 fps measured**. Unrouting a sink from the browser turned its output black and kept it running at 50 fps, reporting `no source routed`.

Running it

```
frame-ferret run
```

serves the control page. The crosspoint is clickable: sources down one axis, sinks along the other.

```
frame-ferret interfaces     # what NICs this machine has
frame-ferret kinds          # what node types this build has
frame-ferret selftest
```

Bind the interface explicitly. A stream on the wrong NIC is diagnosed on site as a network fault, which is an expensive hour. Every node takes an interface.

> **NDI is the exception, and it is not one Frame Ferret can fix.** NDI's own C API has no interface parameter at all, so Frame Ferret **warns** rather than pretending the setting applied.

What works today

NDI send and receive	Verified against an independent implementation, both directions.
OMT send and receive	Verified against oxbow and macOS's own `dns-sd`.
SRT send and receive	Verified end to end against an independent Rust SRT stack.
ST 2110-20 send and receive	Verified against GStreamer's RFC 4175 implementation, within 2 code values.
DeckLink output	Run on a real Duo 2 at 1080p50 in v210.
Screen, window, application and region capture (macOS)	Built and run. A region of interest is a source rect, so a crop costs nothing.
Syphon output (macOS)	Verified against Resolume's Syphon — a different implementation from the one vendored here.

One measurement worth understanding before you report a bug

Over an NDI round trip, **saturated blue comes back 178 rather than 191.**

That is NDI's own compression, not this code — an independent sender through the same round trip lands on exactly the same 178. **The control experiment mattered**: the number alone would have looked like a bug in the pixel conversion.

OMT does much better through the same test — all eight bars within 2 code values — because it carried BGRA rather than compressing it. That difference is the protocols, and it is the reason to choose between them.

SRT needs a codec, and that is why ffmpeg is a subprocess

NDI and OMT carry **raw frames**, so a node is a thin wrapper over the SDK. **SRT carries an MPEG transport stream of *compressed* video**, so it needs a codec.

Frame Ferret runs **ffmpeg as a subprocess** rather than linking it:

- **You point at the install you already have** — the node's own setting, `$FERRET_FFMPEG`, `$PATH`, or the usual locations, in that order. **An explicit path that is wrong is an error, never a quiet fallback to some other ffmpeg.**
- **No ABI coupling.** libavcodec's structs change between major versions; mirroring them by hand would be far more fragile than NDI, OMT or libsrt, which all expose deliberately flat C ABIs.

- **Licensing stays simple.** A separate process over a pipe raises no question an MIT codebase needs to answer.

Hardware encoders are preferred where present.

What is not built

Designed and documented, but **not implemented**: the **UVC virtual camera** (it needs an embedded provisioning profile the release harness does not yet produce), the **virtual display** (no public macOS API exists), **DeckLink capture**, **Spout**, **HTML output**, the **tray launcher**, and **ST 2110-30 audio and -40 ancillary**.

Those are absent from the UI rather than present and failing.

If something is wrong

Symptom	Cause
A stream is on the wrong NIC	Every node takes an interface — except NDI, whose API has no such parameter. Check for the warning.
A sink is black but running	Nothing is routed to it. That is the invariant, not a fault.
Blue reads 178 over NDI	NDI's compression. Independent senders land on the same number.
SRT does nothing	It needs an ffmpeg. Check the resolution order above; a wrong explicit path is an error rather than a silent fallback.
A node type is missing	Run `frame-ferret kinds`. Several are designed but not built — see above.
It does nothing on Windows or Linux	Both build and self-test in CI and have never been run interactively. No hardware path exists on either.
