

GodwinMix technical overview

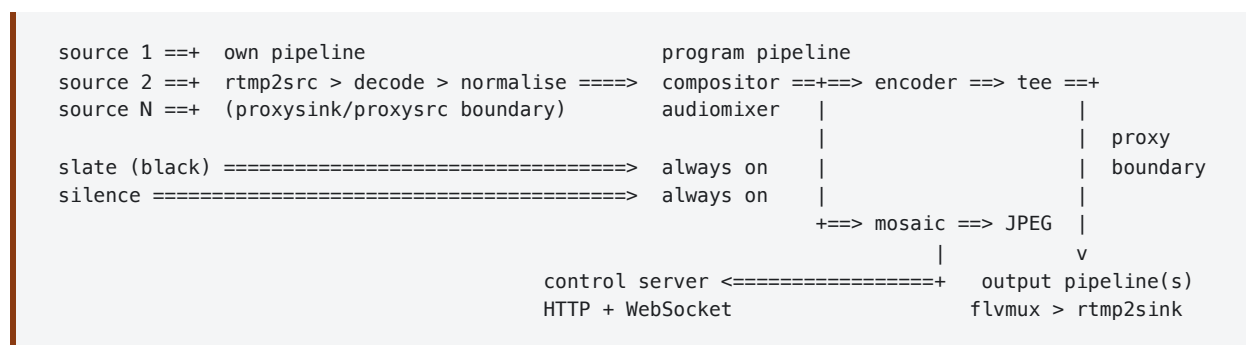
For your engineer: how GodwinMix is built, how it keeps the programme on air, how to drive and extend it, and what is still open.

OCTOBER 2026

The short version

One Rust binary on GStreamer, serving one HTTP port. The same binary is the headless server, the command line tool (`gmx` and `godwinmix`) and the MCP server. The web interface is a client of the same public API, and so is the desktop app. There is no private path a third party cannot use.

The pipeline



An RTMP output has to look like one unbroken stream: timestamps that only go forward, no gaps, codec parameters that never change. So the encoder is started once and runs until the broadcast ends. Everything that changes during the broadcast happens upstream of it, in raw video and raw audio. A take is a property change on a compositor pad, and it lands on the next frame. The encoder cannot tell anything happened, so nothing downstream reconnects.

Three isolation boundaries

Each of these is its own `GstPipeline`, because in GStreamer the pipeline is the unit of error propagation.

1. Every source. A camera that drops, errors or sends garbage cannot post a bus error into the programme pipeline.
2. Every output. A failing RTMP sink returns a flow error that would otherwise travel back through the tee and tear down the programme's streaming threads. Without this boundary, one dead CDN would take the whole broadcast with it.
3. The multiview. A problem in the preview encoder cannot touch the programme.

The force live compositor

The compositor and audiomixer are built with `force-live`, so they emit black and silence on schedule even with every input dead. A slate sits permanently at the bottom of the compositor's z order. Losing a source reveals black instead of freezing on the last frame.

The outage buffer and the proxy boundary

The outage buffer (`queue_secs` , 5 seconds of encoded data by default) lives on the programme side of the proxy boundary. That way it survives a reconnect that rebuilds the output pipeline. It leaks rather than blocks, so a slow destination cannot push back on the encoder that every other output shares.

Late sources

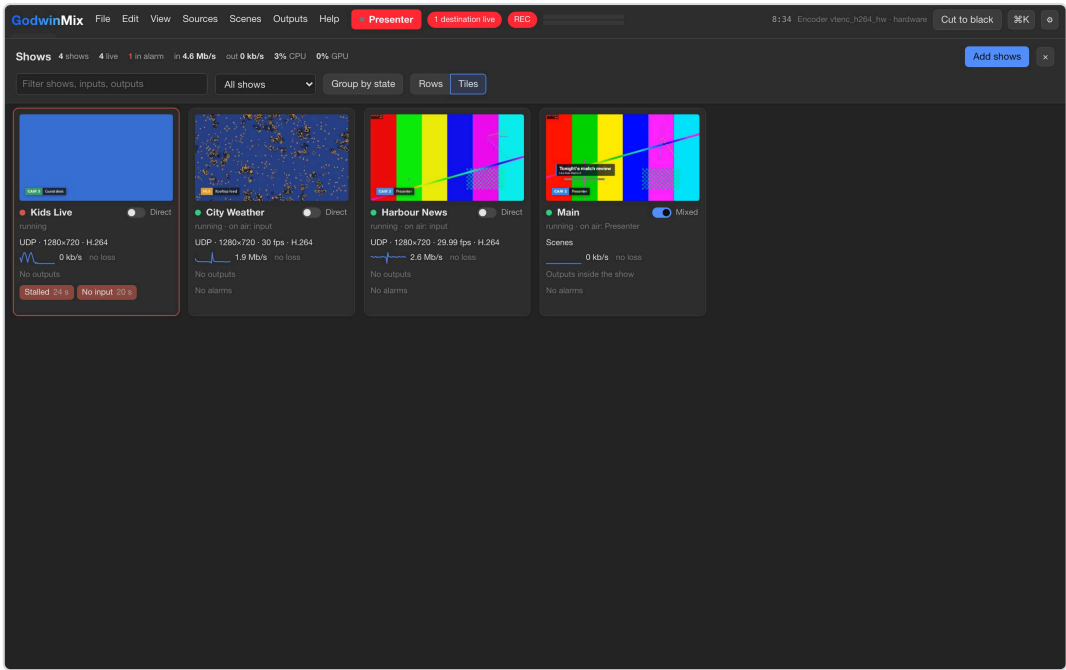
A source whose frames arrive late is pulled back onto time: if every frame waits more than 200 ms for two seconds, the mixer moves it back. Tests hold preview latency to budgets of 90 ms for a source tile, 120 ms for the programme tile and 90 ms for the Studio preview. With a camera that delivers its first 1.5 seconds of frames in a burst, the measured source tile delay fell from 1,649 to 1,708 ms down to 201 to 202 ms. This work is dated 6 October 2026, sits on a working branch, and has not yet run in the installed app. The programme compositor can still wait up to one second for a late input.

Nothing runs unless asked

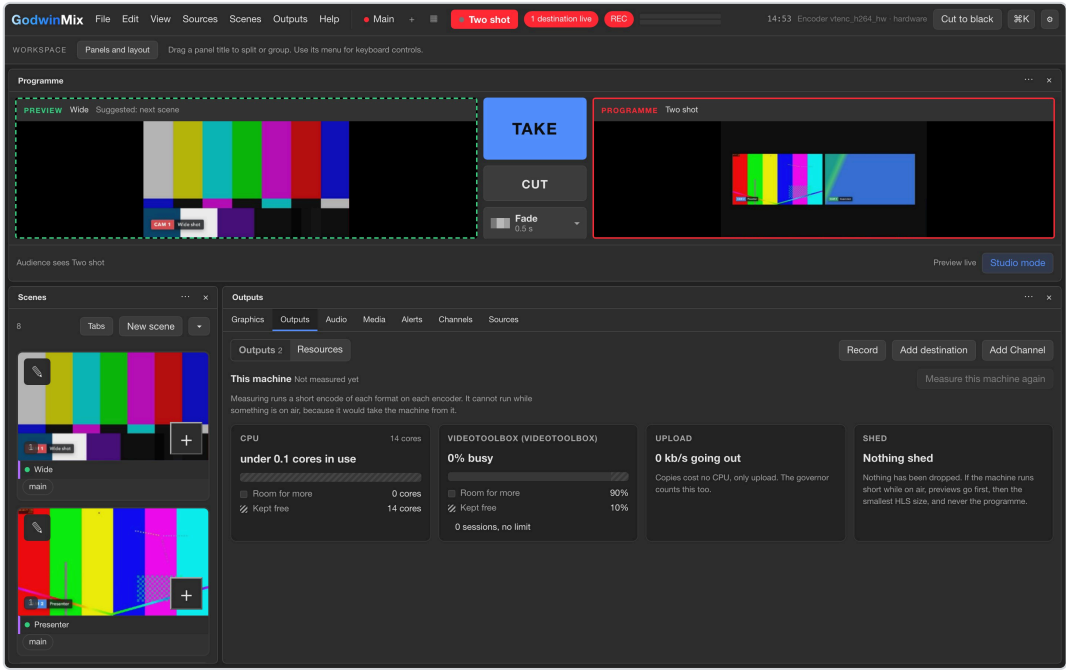
The multiview, thumbnails, preview, audio monitoring and the encoder start with the first client that needs them and stop with the last.

A station of shows

The process you start is a station that runs each show as its own process. A show that crashes restarts while the others stay on air. The station itself costs about 20 MiB. Direct shows copy one input to outputs without decoding, at 1 to 3 percent of one core for a 1080p 8 Mbit/s feed, and a scale test drives 200 live MPEG-TS feeds. A resource governor checks whether the machine can take on new work without costing a frame on air.



The monitoring wall lists every show on the station; a show whose input stalls is raised in red with how long it has been stalled.



The governor's view of CPU, hardware encoder and upload in use, and what is held back for the programme.

Hardware codecs

The same binary picks its codecs at startup and works with or without a GPU. Decode and encode are chosen separately, because a machine with a hardware decoder and no usable hardware encoder is a real configuration.

Hardware	Decode	Encode
NVIDIA	<code>nvh264dec</code>	<code>nvh264enc</code>
Intel or AMD (VA, Linux)	<code>vah264dec</code>	<code>vah264enc</code>
Windows	<code>d3d11h264dec</code>	<code>mfh264enc</code>
macOS	<code>vtdec_hw</code>	<code>vtenc_h264_hw</code>
Software	<code>avdec_h264</code>	<code>x264enc</code>

On the Intel test laptop, Windows picked Quick Sync by itself. Setting `hardware.decode` or `hardware.encode` to a specific backend makes startup fail loudly if that backend is missing, which is what you want on a server you control. Properties are set defensively: backends disagree about names, units and integer widths, and a property one of them lacks is a logged warning, not a crash.

One caution from the status record: on that Intel test laptop, the Quick Sync driver corrupts memory on about one show start in four. The station restarts the show when it happens.

RTMP client interoperability

GStreamer ships two RTMP client implementations and they do not work with the same servers. Against node-media-server, `rtmp2src` connects, is accepted, and then delivers nothing at all, with no error: 0 buffers where the `librtmp` based `rtmpsrc` delivered 250 from the same stream. `ffmpeg` reads that stream fine, so it is a client problem. Because the failure is silent, `rtmp_client = "auto"` starts with the modern client and switches once after six seconds without media. You can pin `rtmp2` or `librtmp` per source to skip the wait. The publish direction (`rtmp2sink`) works with both servers.

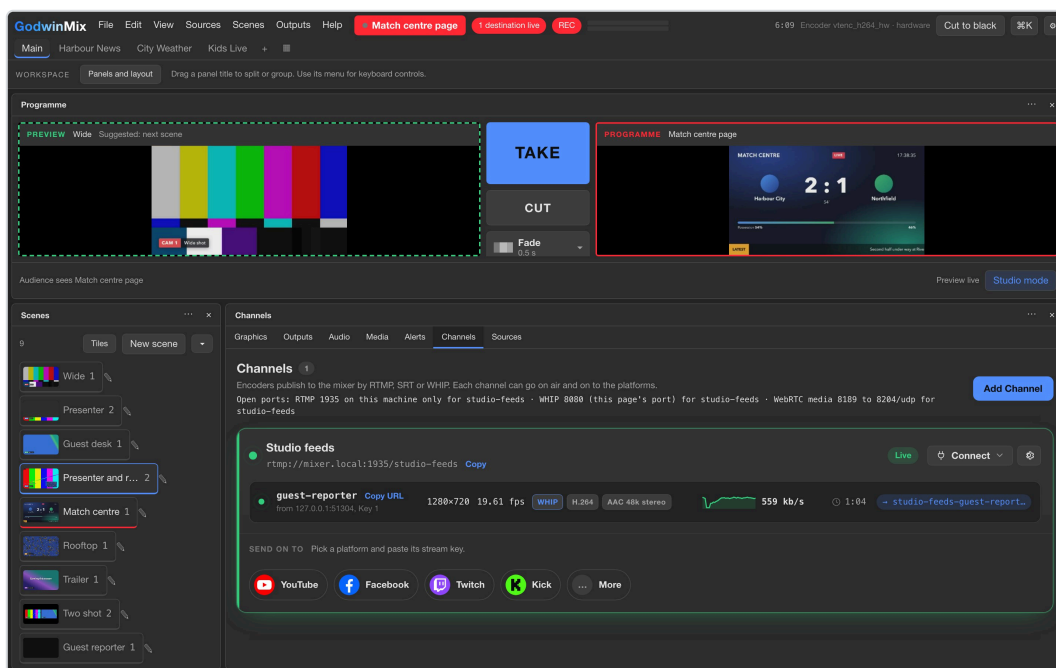
Sources and outputs, in brief

Sources are read from their URL:

URL	Opened as
<code>rtmp://</code> , <code>rtmps://</code>	RTMP
<code>https://host/stream.m3u8</code>	HLS
<code>https://host/manifest.mpd</code>	DASH
<code>rtsp://</code> , <code>srt://</code> , <code>udp://</code> , <code>rtp://</code>	a continuous stream
<code>web+https://host/page</code>	the page in Chromium (a CEF sidecar), with its audio
<code>exec:<command line></code>	whatever the process writes to stdout
a path, <code>file://</code> , an <code>https</code> clip	a file

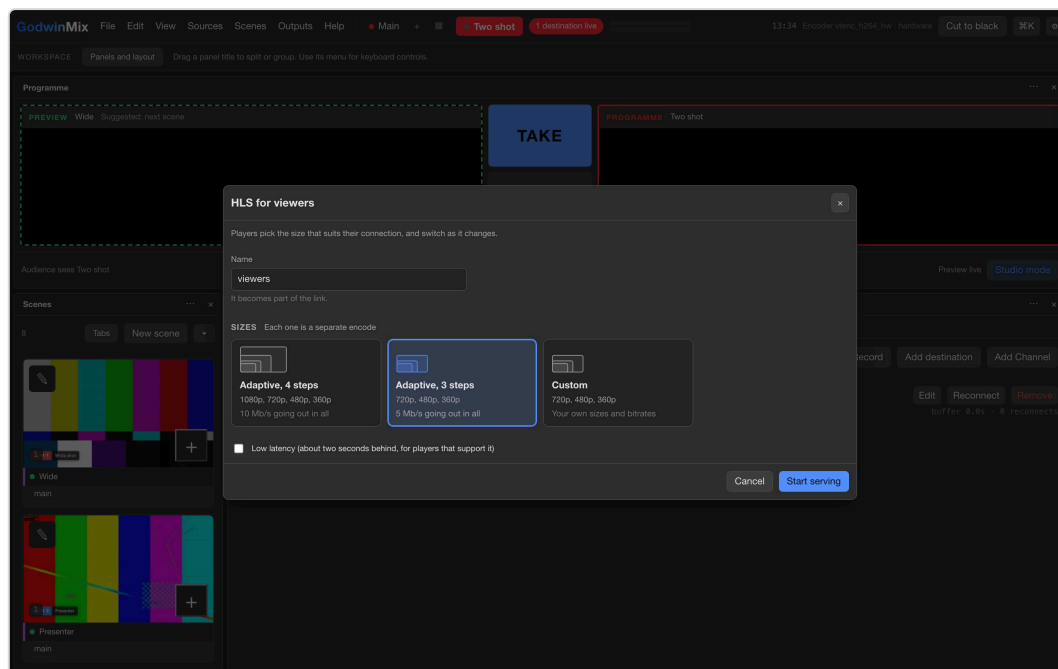
A web page with video inside can be superimposed: the mixer decodes the page's video on its own hardware decoder and the browser draws only the page around it. On a Mac, a page with three videos fell from about 130 to about 28 percent browser CPU this way. Superimpose cannot take over MSE or DRM players, and YouTube is MSE. Pages can be added with `transparent: true`.

Plugins add cameras, microphones, screens, NDI, IP cameras, DeckLink, MPEG-TS over UDP and RTP, Icecast and RIST. The built in `ingest` plugin is an RTMP server in pure Rust plus WHIP, so every publisher becomes a source; channels take RTMP, RTMPS, SRT and WHIP with one set of keys. The browser publisher page sends H.264 at up to 2.5 Mbps and 30 fps from a 1280x720 camera with Opus audio, and WHIP accepts VP8 from browsers that offer no H.264 and encodes it to H.264. Sources are expected to be H.264 and AAC; HEVC is handled through channels.



A browser guest live on a channel over WHIP, with its resolution, codecs and bitrate, and the source it became.

Outputs: RTMP and RTMPS (added and removed live), SRT, RIST, UDP and multicast, an RTSP server, WHIP, WHEP playback under half a second behind, NDI, Icecast, and HLS, LL-HLS and DASH served from the control port with nothing written to disk. Rendition presets include `youtube-1080p30`, `youtube-720p30`, `facebook-720p30`, `twitch-1080p60`, `twitch-720p30`, `audio-only-aac`, `abr-ladder-4`, `abr-ladder-3` and `copy`. Recording goes to fragmented MP4 (a crash still leaves a playable file) or Matroska.



Serving HLS from the mixer: a 4 or 3 step adaptive ladder or custom sizes, with an optional low latency mode.

The API and the compatibility promise

Everything the interface does is one call on the HTTP API. `protocol.json` lists 215 methods. A WebSocket at `/rpc` gives a snapshot and then deltas. Calls can carry idempotency keys (honoured for 24 hours) and a `dry_run` flag, and refusals name the next step to take. `POST /api/golive` sets the page, the destination and the take in one call.

The web interface, `gmxctl`, the MCP server and the desktop app all use this same public API. There is one protocol for every client.

The promise: `api_level 1` is frozen for breaking changes. Additions bump the level. Breaking changes move only at an announced major version, at most once a year, and the previous level is supported for a year after that.

Generated clients exist in TypeScript, Python and Rust, built from `protocol.json`.

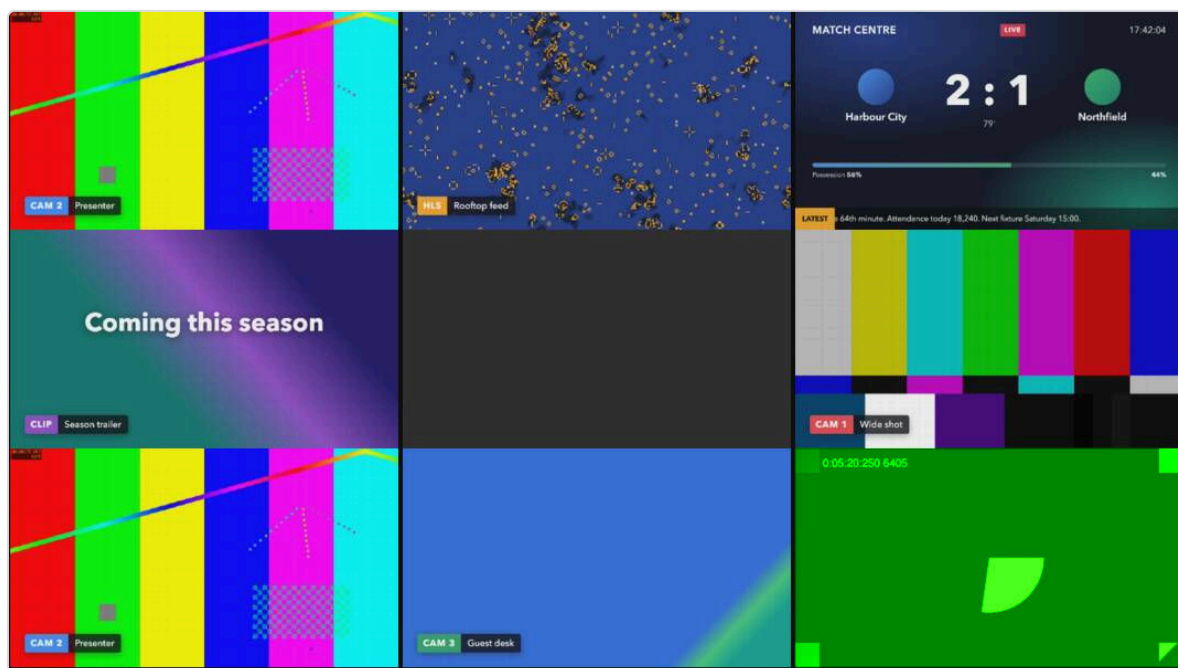
```
gmx ctl status

$ gmx ctl status
program : cam2
backend : vtenc_h264_hw (hardware)
source  : roof      live      http://127.0.0.1:8888/...
source  : scoreboard live      web+http://127.0.0.1:8090/...
source  : trailer    live      ...
source  : lower-third-glass live ...
source  : cam1       live      rtmp://127.0.0.1:1936/...
source  : cam2       live      rtmp://127.0.0.1:1936/...
source  : cam3       live      rtmp://127.0.0.1:1936/...
source  : studio-feeds-guest-reporter live ...
output  : recording-muwm27u4 live      0 reconnects
output  : primary    live      0 reconnects
```

`gmx ctl status` from a terminal: what is on air, every source and every output, one line each.

The agent surface

godwinmix mcp serves MCP over stdio or Streamable HTTP. Tool profiles are budgeted in CI: minimal is 6 tools (5,922 bytes), standard is 14 (18,244 bytes), headend is 11 (16,785 bytes). Every profile has `call_tool`, which runs any tool by name. `GET /api/v1/agent/state` returns what is on air, each source's state and a motion figure from 0.0 to 1.0, in 228 bytes at two sources and 823 at sixteen. The snapshot sheet puts every source and the programme in one labelled image; a 320 by 180 still is 84 tokens to a vision model, against 2,691 for 1080p. Safety rules in the core apply to every caller: a minimum shot length (zero by default for people), a rate limit (120 takes a minute by default) and an ITU-R BT.1702-3 flash guard. A token marked as an agent is held to eight seconds a shot and twelve takes a minute whatever the config says, and can only be made stricter. A refusal carries the rule and how long is left. The project's agent evals passed 34 of 34 cases, three runs each, on 17 September 2026.



The snapshot sheet an agent reads: every feed source and the programme in one labelled image.

Plugins

Plugins are separate processes with a manifest. They speak JSON lines on stdio, and media crosses by unix file descriptor, shared memory or a pipe. A whole source plugin fits in under 200 lines of standard library Python.

There are 19 first party plugins in the repository today, plus two shared libraries. The status page's figure of fourteen dates from v0.2.0.

Plugin	Purpose
audio-device	microphone, line input or sound card
camera	USB or built in camera
screen	a monitor or region
file-record	programme to a file
decklink	SDI and HDMI through Blackmagic DeckLink
director	automatic director, by rules or a model
icecast	programme sound to Icecast; radio in
ingest	RTMP server in pure Rust, WHIP ingest, channels
ipcam	IP cameras, ONVIF discovery
min-hold	refuse a take too soon after the last (WASM)
ndi	NDI in and out
ograf	OGraf graphics host
osc	OSC control, tally and programme out
rtsp	serve the programme over RTSP
srt	receive SRT
tally	TSL UMD v5 tally lamps
udp	MPEG-TS over UDP and RTP
whip	WHIP out, WHEP in
wipe	a worked example transition

Installers carry every first party plugin that runs on that system. Device plugins install on first launch and the rest on first use, with nothing downloaded. Sources can also run on another machine (a node) over mutual TLS. SDK templates exist in Go, Node, Python, Rust, shell and WASM.

Some plugin abilities are built but not yet wired into the core: plugin device listings, plugin tools over MCP, and non source plugins on nodes.

The WASM tier

A service or transition plugin can run as a WebAssembly component inside the core process instead of as a separate process. It is sandboxed, held to a fuel allowance and a deadline on every call, and touches no media by design. The `min-hold` plugin is an example.

Presets and themes

Six presets ship: `default`, `church`, `classroom`, `esports`, `broadcast` and `headless-agent`. A preset is itself a plugin, and `gmx preset apply <name>` writes the config, scenes and layout. Some presets name plugins that do not exist yet (`replay`, `companion`, `browser`, `rtmp`); the presets README says this is deliberate.

Themes are one CSS file each. Five ship: `dark`, `light`, `high contrast`, `system` and `touch`. Third party panels can be added to the interface.

Embedding the engine

`godwinmix-core` is the same engine as a Rust library, with no HTTP server and no argument parser. You call `gststreamer::init()`, build a `Config` (every field has a default), and drive the mixer from your own code.

Custom branded builds

A separate build template turns a preset plus your branding into installers under your own product name, built by GitHub Actions from a pinned GodwinMix tag. It refuses to bundle a GPL codec, so `openh264` and `SVT-AV1` are present as the fallbacks a closed build encodes with.

Deployment

Way	What it is
Docker, headless	<code>ghcr.io/psmux/godwinmix:0.2.1</code> , with a compose file that adds <code>mediamtx</code> . No screen needed.
systemd	A unit file for a plain Linux server.
Desktop app	Installers for Windows (setup <code>.exe</code> or <code>.msi</code>), macOS on Apple Silicon (<code>.dmg</code>), and Linux (<code>.deb</code> and AppImage).
From source	Rust plus GStreamer with development headers on any of the three systems.

To reach it from outside the machine you put it behind a reverse proxy with TLS. Guests joining from another network need it served over https. The desktop app keeps other devices out by default; turning that on binds the network and still demands a token.

Footprint

Measured on an Apple M4 Pro with 24 GB on 14 September 2026:

Measure	Result
Core binary	8.4 MB
Idle memory, no sources, multiview off	58.0 MB
Cold start to first encoded programme frame	0.15 s
Two 720p30 live sources with programme encode	0.70 cores with hardware encoding, 0.88 software
Added memory per file source	34.7 MB

The reference machines that matter most for small venues (Raspberry Pi 4 and 5, an Intel N100, a laptop with no GPU) have not been measured yet. The README goes further and says no footprint numbers are published; the M4 Pro record above exists, but it does not tell you about your machine. Try it on yours.

Installer sizes from the release rehearsal of 6 October 2026: Windows setup 195 MB, `.msi` 271 MB (budget 300 MB), `.dmg` 224 MB, AppImage 448 MB, `.deb` 441 MB. The trimmed GStreamer inside measures 154.6 MB on Windows, 146.4 MB on Linux and 109.3 MB on macOS, against a 160 MB budget. The stock GStreamer runtime installer is 527 MB, for comparison.

Reliability records

Date	What	Result
17 September 2026	First hosted Linux soak, 60 minutes	720 rounds, no panic, no unanswered call. Missed one 51 ms stall; memory grew, attributed to GStreamer buffer pools.
19 September 2026	macOS integrated run	36 rounds, 0.05 percent memory growth, threads and descriptors flat. The strict timing gate failed at 34.028 ms against 34 ms.
Up to 5 October 2026	Installed desktop app on Windows	Ran 16.5 hours without a fault, then under heavy outside load grew to 12 GB and died. Cause found (an unbounded camera pipe) and fixed in 0.2.1. Stall restarts now back off up to 160 seconds.
5 October 2026	Two phones joining a full show	Aborted the mixer. Fixed; both phones then held live for three minutes, twice.
6 October 2026	Transitions under heavy load	19 of 19 runs passed after fixes.

The README speaks of 123 unit tests that build real GStreamer pipelines with no mocks. That count predates a lot of later work, so we do not quote it as current.

Open items

From the status page, in the order the project ranks them:

1. The mixer command loop could wedge after hours with an output reconnecting to a server that is not there. Two wedges have been found and fixed. Every call that waits on the mixer thread now has a five second deadline, and `/api/status` and `/metrics` answer 503 naming the command holding the loop instead of hanging. Whether the original overnight wedge is fully gone needs the long soak to finish.
2. CI on all three systems. Many fixes landed; further CI failures are being fixed on a working branch with a draft pull request, dated 6 October 2026.
3. A core once exited silently during mosaic teardown. Three causes have been found and fixed; two ten minute soaks then ran clean.
4. Alpha graphics keying to black. This item is out of date: the October work and the graphics reference say transparency now works, on Windows so far.
5. Smaller gaps: only sources can be placed on a node; moving a source between nodes is a remove and add; no template for a new graphic plugin; the composer's picker does not offer graphics.

Other limits worth planning around:

- No game capture on Windows, not in the first year. Screen capture is a desktop or a region.
- One picture shape at a time. No vertical and horizontal output together.
- A web page with H.264 or AAC inside needs a codec enabled browser build on macOS and Windows; superimpose is the way round it, except for MSE and DRM players such as YouTube.
- A superimposed page whose browser crashes takes about 14 seconds to return; other sources about 2.
- The preview mosaic carries no sound; meters stand in.
- The Linux installer still lacks `cmamux`, `livesync`, the WHIP and WHEP elements and `dav1ddec`. Windows cannot open WebP pictures.
- Moving graphics, transparency and transition packs have run on Windows only. A stinger's sound is dropped. Transparent items draw above every opaque item regardless of stack order.
- Guest links carry the channel key until invites exist.
- No plugin marketplace yet.