BlackMatrix user guide

BlackMatrix is **a crosspoint router matrix for a fleet of Blackmagic ATEM switchers** — sources across the top, destinations down the side, one click to route.

It also **pretends to be a Blackmagic Videohub**, so hardware router panels, Companion and Blackmagic's own software can drive the same crosspoints — and, for a redundant rig, so a **media server can switch to its backup machine through it**.

> **Before you rely on this:** it was developed against a built-in simulated switcher fleet and a real TCP client speaking the Videohub protocol, then **checked against a real ATEM Mini Extreme ISO** — which *corrected* the routing rules rather than confirming them. The availability masks were probed on that hardware, and this app's matrix was served from it as a 29×39 router.
>
> **What has never happened is a real Videohub control panel driving it**, and no real Videohub has ever been one of its devices. Neither mobile app has run on a physical device. Prove it on your own kit before it goes anywhere near a show.
>
> This codebase was created with AI assistance, directed and reviewed by a human author.

The model: an ATEM has no router

So this treats **every bus that takes one source at a time** as a destination, grouped into sections:

Section	Destinations
Outputs	Aux 1..n
Program / Preview	Program and Preview per ME
Keyer sources	USK fill and key per ME, DSK fill and key
SuperSource	Box 1..n, art fill, art key
Multiview	Every multiviewer window

Which sources are legal on which destination is read from the switcher, not from a model table. Every ATEM input reports availability masks, and illegal cells are drawn hatched and refused server-side — so an aux output cannot be routed back onto an aux bus, and ME 1's output cannot be routed onto ME 1.

That is why a capture matters: see below.

Getting started

```
npm run dev:mock
```

serves the UI on http://localhost:8533 with three simulated switchers of deliberately different shapes and a Videohub server per switcher. No hardware, no network switchers, nothing to break — this is the right way to build a layout before you are in the room.

Against real switchers, write `blackmatrix.config.json` and `npm start`.

The **Devices** page adds, edits, reconnects and removes switchers and routers with **no config file and no restart**. **Scan network** sweeps the local /24s for both kinds — a switcher answers on UDP 9910, a router on TCP 9990, and neither answers the other's probe.

Capture a switcher before you lose it

```
npm run capture -- 192.168.10.240 --name "Stage"
```

The availability masks this whole project depends on exist only in the protocol. They are not in an ATEM autosave, and not in any file a switcher leaves behind. A capture puts the real shape of a real switcher on disk, and `"capture": "<file>"` on a device replays it with no hardware present.

So: capture every switcher you will ever have to build a show for, while you have it. See capture.md, including the opt-in probe that tests the masks against the hardware.

Device ids are permanent, deliberately

A device id **cannot be changed once made** — salvos, ties and label overrides are all keyed on it.

Removing a device **leaves salvos and ties that reference it alone** rather than rewriting them, and says which ones. An operator who pulls a switcher out for an hour should get their salvos back when it returns.

Videohub emulation ports are assigned from `basePort` upward and **written back to the config**, so they stay put as devices are added and removed. A panel is configured against a port number, and that number is a fact about the device rather than about where it sits in a list.

If something else on the host already listens on 9990 — Bitfocus Companion's Videohub panel surface does — the affected switcher logs the clash, starts without it, and everything else keeps working. Give it another port.

Driving it from a panel

Point any Videohub client at `<host>:<port for that switcher>`: Companion's `blackmagic-videohub` module, a Smart or Universal Videohub hardware panel, Blackmagic's Videohub Control software, or anything else speaking the protocol.

Input numbers are the switcher's sources in ATEM source-id order; output numbers are the destinations in the section order above. Both orderings are stable for a given switcher, so a panel's buttons keep meaning the same thing.

Full mapping and a worked telnet session: videohub.md.

Claims

A destination can be **claimed** — the flag on its row in the browser, or a lock from a panel. They are the same thing under different names: the app says claim, the Videohub protocol says lock. **Claims are held per IP address**, which is how a real Videohub does it, so a second panel on the same machine shares one and everyone else is refused. Shift-click the flag (or send `F` on the wire) to force one open.

It is enforced in one place, so a claim made from a panel refuses a route made from the browser, and the other way round.

A claim stops you as well — that is the point of it. Claim a row and it crosshatches: its crosspoints stop taking clicks, it cannot be staged into a take, and the flag is how you give it back. The Videohub rule is that whoever holds a lock may still route through it, and panels keep that, but the browser and the phone app sit at an address like anything else — so a claim that let the claimant route would stop nobody standing at the screen.

Salvos

A salvo is a named set of crosspoints **across the whole fleet** — one press sets them all.

Build one in the UI by capturing what destinations are currently taking (**Build new**, then `+` on each row you want), or write them into the config file.

Every crosspoint in a salvo is attempted, and the ones the switcher will not accept come back named rather than being silently dropped.

Salvos are this app's own idea — not part of the Videohub protocol — so they exist in the UI and the REST API only.

Redundant systems

A redundant media server rig is two machines playing the same show and a router downstream deciding which one reaches the screens. This can be that router, in either of the two shapes the industry uses.

The media server drives it. disguise's understudy sends matrix routing itself the moment it takes over a failed machine, and PIXERA fires a control action from its `System Lost` trigger. Both can point at the Videohub emulation. Because **an ATEM re-syncs its inputs, the main and backup feeds do not have to be genlocked to each other for a clean cut** — which an SDI router does require.

Or this app decides. A **failover watch** polls a machine — a TCP port, a URL, or a heartbeat it must be sent — and fires a salvo when it stops answering. It **will not fire before it has seen the machine working once**, it starts disarmed, it fires once, and it does not switch back unless you say what "back" means. What it fires is an ordinary salvo, so **the failover can be rehearsed by pressing Take on it.**

And a plain line protocol on TCP and UDP for anything that can send a string but not speak Videohub:

```
ROUTE 2 1          route output 2 to input 1
1*2!               the same thing, Extron style
SALVO backup       fire a salvo by name
3.                 fire the third salvo (Extron preset recall)
FAILOVER main      fire a watch's lost salvo
```

> **One lock trap, before you rely on any of it.** A refused route is answered with `ACK` and an unchanged status, as the Videohub spec requires — which means **a media server cannot see a refusal**. A locked destination is a failover that silently does not happen. `videohub.failoverClients` names the addresses whose routes walk through locks.

All of it, including how to fill in disguise's and PIXERA's fields, is in failover.md.

On a phone

Below 800px the grid is replaced by an **X-Y panel** — a destination list showing what each is taking, then the sources that destination will accept. Not a smaller grid, because a grid does not shrink; this is what a hardware panel does, for the same reason.

Preset is the default at that width, so a mis-tap stages instead of cutting.

That works in any phone browser. The native iOS app adds the one thing a browser cannot do — finding the server without typing an address — and is deliberately a shell: it does not speak the

ATEM or Videohub protocols, because a phone has no business holding a switcher connection open while the OS suspends it.

Names

Renaming a source renames the input on the switcher itself. Renaming a **destination** is local to this app — the ATEM has no name for "Aux 2".

If something is wrong

Symptom	Cause
A cell is hatched and refuses	That route is illegal on this switcher, read from its own availability masks.
A route from a media server silently did not happen	The destination is locked, and the protocol answers ACK either way. See the lock trap above.
A switcher started with no Videohub server	Port clash on 9990 — Companion's panel surface is the usual culprit. The log names it; give it another port.
A salvo lost its rows when I removed a device	It did not — they are left in place deliberately, and named.
The simulator's model numbers look wrong	Only entries marked as read off hardware carry real numbers. Everything else is an approximate shape; a capture replaces it with the truth.
