Companion — Presentation Commander Server user guide

This module drives a Presentation Commander **Master Server** from a Stream Deck, or any other Bitfocus Companion surface: **route outputs, recall scenes, blackout, send stage notes and step slides**.

The README covers installing the module and pointing it at the server. This is how to build a surface with it, and what to be careful with.

What this drives

This changes what an audience sees. A button press here routes live outputs on a running Master Server. There is no confirmation step and no undo.

Two things to know before you build a page:

- **The Master Server's automation API has no authentication.** It binds `127.0.0.1` by default for exactly that reason. If you've tunnelled or proxied it so Companion can reach it from another machine, anything else that can reach it can drive your show too.
- **Some of the things this module reports are up to 3 seconds stale**, and some can be wrong while the connection is down (Feedbacks).

Connecting

Two config fields: **Master Server host** (default `127.0.0.1`) and **Automation API port** (default `9700`).

The default is loopback because that's where the server listens. **If Companion runs on a different machine**, the server's API is not reachable directly — see the README's section on that; it needs a tunnel or an authenticating proxy rather than a config change.

When it's connected the instance shows OK and the `connection_status` variable reads `Connected`.

The buttons you can build

Action	What it does
Route Output	send an output to a scene or a single source
Blackout Output	send an output to nothing
Recall Scene to Output	put a scene on one output
Send Note to Stage	push a message to the presenter
Next Slide / **Previous Slide**	step a Client Node's deck

All the dropdowns fill themselves from the live server — you pick real output, scene, source and client names, not generated ids. They refresh automatically as the server's list changes.

Worth knowing:

- **Route with "— Unrouted —" is the same as Blackout.** Both exist because they read differently on a button; the command sent is identical.
- **Recall Scene affects one output**, not the whole rig. It is not a global preset recall.
- **Send Note replaces the previous message.** There's one slot, not a queue. Send an empty message to clear it.
- **If the server had no outputs when the module last refreshed**, a new button's Output dropdown will default to blank. Re-open the dialog once the server is up.

Slide buttons can succeed without moving anything

Next/Previous Slide are forwarded to the Master Server, which passes them to the Client Node — **if that client is connected.** If it isn't, **the server steps its own internal counter and reports success anyway.**

This module cannot tell the difference. The button will behave exactly as if it worked while the projector doesn't move.

Put the "Client Node is online" feedback on those buttons. It's the only indication available that the command has somewhere to go.

Feedbacks, and where they lie

Two feedbacks, both go green by default:

- **Output is routed to a specific source/scene** — the button lights when that output is currently on that target.
- **Client Node is online** — lights while that client is connected.

They keep showing the last known state when the server is unreachable

If the connection drops, the module marks itself failed and sets `connection_status` to `Disconnected` — **but the routing and client feedbacks keep displaying whatever they last saw.**

So a green "on air" button can be green while the module has no idea what's actually on air.

Put `$(<instance>:connection_status)` somewhere visible on every page that has routing buttons. That variable is the honest one.

Everything lags by up to 3 seconds

The module **polls** the server every 3 seconds; there's no live push. A route changed from the server's own UI, or by another surface, won't light the corresponding button until the next poll.

Don't treat a button that hasn't lit yet as a failed command.

Variables

Variable	Shows
`connection_status`	`Connected` / `Disconnected` — **the one to trust**
`client_count`	how many Client Nodes are online
`routed_<output>`	the scene or source name on that output, or `Unrouted`

`routed_*` variables only exist **after the first successful poll**. On a server that has never been reachable, they aren't defined at all — which is why a fresh install against a stopped server shows nothing rather than showing `Unrouted`.

Building a surface that fails safe

1. `connection_status` **on every page** that has routing buttons (Feedbacks).
2. **"Client Node is online" on every slide button** (the buttons).
3. `routed_<output>` **as button text** on your route buttons — it reads back the actual routed name, so a stale feedback and a stale variable at least agree with each other.
4. **Keep Blackout adjacent to Route** on the same page. It's the fastest way out of a wrong route.
5. Remember the 3-second poll before pressing a button a second time (Feedbacks).

Troubleshooting

Symptom	Cause
Instance shows Connection Failure	Server not running, or not reachable at that host/port. Default is loopback-only (Connecting).
Companion is on another machine and can't connect	The API binds `127.0.0.1`. Needs a tunnel or proxy, not a config change (Connecting).
Dropdowns are empty	No successful poll yet. They fill from live server state (the buttons).
A new button's Output field is blank	The server had no outputs when the module last refreshed (the buttons).
Button lit green but nothing is on air	The connection may be down — feedbacks keep the last known state (Feedbacks). Check `connection_status`.
Slide button works but the deck doesn't move	The Client Node is offline and the server simulated it (the buttons).
A change I made on the server didn't light the button	Up to 3 seconds of poll lag (Feedbacks).
`routed_*` **variables don't exist**	No successful poll has ever happened (Variables).
Send Note replaced my previous message	One slot, not a queue (the buttons).
A Stream Deck button silently stopped working after a server update	The three repos share a protocol kept in sync by hand — this module is the one people forget. See DEVELOPING.md.

See also

- API.md — every action, feedback, variable and the poll loop
- DEVELOPING.md — the three-repo protocol
- README — setup, remote Companion, installing