Companion — Mynah user guide

This module drives an **Analog Way LivePremier (Aquilon)** with lighting-desk command syntax, from a Stream Deck or any other Companion surface. Two ways in: type a command, or **build one out of key presses**.

The README covers installing the module. This is how to use both, and what the grammar will and will not accept.

> **Before you rely on this:** the module has been driven end to end in Companion 5.0.1. The socket it uses is the **same Web RCS WebSocket that Analog Way's own browser UI uses**, and it is **unauthenticated** — keep it on a trusted network, because anything that can reach it can put something on air.
>
> This module was built with AI assistance, directed and reviewed by a human author.

Connecting

Device address and port. **80** for a switcher, **3000** for the LivePremier simulator. The module reconnects on its own if the link drops.

The Command action

Verb first, then what it acts on:

Command	What happens
`Recall Screen 1 Memory 5`	Memory 5 into Screen 1's preview
`Recall Screen 1 Memory 5 Program`	Straight to program
`Recall Screen 1 Thru 4 Memory 7`	Four screens at once
`Recall Master 12`	A whole-desk memory
`Recall Screen 3 Layer 1 Memory 8`	A layer memory
`Store Screen 1 Memory 5`	Store program into memory 5
`Store Master 12 If Screen 1 + 3 Category Source`	A masked store
`Take Screen 1`	Transition preview to program
`Label Screen 1 Memory 5 "Wide Open"`	Name a memory
`Delete Screen 1 Memory 5`	Erase it

Ranges use `Thru`, `+` and `-`, so `Screen 1 Thru 8 - 5` is every screen from 1 to 8 except 5.

Keywords may be shortened to any unambiguous prefix, so `R Sc 1 Th 4 Me 7` is the same command as the long form — which matters when you are typing into a button field rather than a console.

The field supports Companion variables, so a command can be assembled at press time.

The two defaults that decide what reaches air

An unqualified `Recall` goes to Preview. An unqualified `Store` takes from Program — which is the device's own default, not an invention here.

So **getting to air always takes an explicit `Program` or a `Take`.** There is no command that puts something on screen by accident because you left a word off.

Ranges

Bank	Memories
Screen and Aux	1–1000
Master	1–500
Layer	1–50

Screens are 1–24, auxes 1–96, layers 1–128 plus `Native`.

Anything outside is refused rather than clamped, and the refusal is logged. A clamp would put memory 1000 on screen when you meant 10000 and typed a digit too many.

Every command names its own scope

Unlike the Mynah web tool, **a button has no sticky selection.** Every command must name its screen, and `Select` and `Clear` are refused for that reason.

This is the right trade for a surface: a button whose meaning depends on which button was pressed before it is a button that does the wrong thing after someone else walks past the desk.

The command builder

Build a command out of key presses instead of typing one.

Drag the **Command builder** presets onto a page: the numbered slot keys in order, then **Back**, **Home**, **More**, **Fire** and **Save**. Then **tell the connection how many slot keys you laid out** — a Companion module is never told how big a surface is, so it cannot work this out.

Press a slot and the faces change to the next set of choices: the verbs, then what to act on, then a list of screens, with a `123…` key to a keypad for anything longer or for a range.

The page never changes; the keys relabel themselves. That is deliberate — a module cannot change Companion's page, so the builder changes the *faces* instead. The upshot is that your muscle memory for where Fire is stays true all the way through.

Fire lights as soon as what you have built compiles, which is often before the last question is answered: `Take Screen 1` is finished the moment the screen lands, and the offer of Preview or Program is only an offer.

The keypad carries `Thru`, `+` and `-` **wherever the grammar takes a range, and hides them where it does not**. **Back** is an undo, one press at a time.

Save parks the line on one of twenty-four macro slots. Macros live in the connection's own config, so they survive a restart and travel with a Companion configuration export; each macro key labels itself.

What the builder deliberately leaves out

`Set` and `Label`.

`Label` **needs a quoted string**, and a control surface has no text entry. `Set` **— live layer control — needs the device's current buffer state**, which this module does not track, so the compiler would

refuse every one it produced. Offering either would be offering a key that cannot work.

Both are still available from the typed Command action.

Feedbacks

- **Memory is loaded on a screen** — the memory sits in one of that screen's preset buffers. It **does not distinguish preview from program**: the device reports buffers as A/B/C, and which one is preview differs between screens. Read it as "this memory is in play here".
- **Connected to the device**
- **Screen is selected in the vendor Web RCS** — what the browser UI has selected, which is useful when someone else is driving it.
- **Builder: what a slot is showing** — colours a slot key by what it holds, so an empty slot goes dark and a Delete goes red with nothing to configure.
- **Builder: the line would fire**
- **Builder: the list has more pages**
- **Macro: the slot holds a command**

Troubleshooting

Symptom	Cause
The builder's slot keys run out	The connection was not told how many you laid out.
A command was refused	Out of range — the log says which part. It refuses rather than clamping.
Nothing reached air	An unqualified `Recall` goes to preview. Add `Program`, or send a `Take`.
`Select` **does nothing**	It is refused by design; every command names its own scope.
`Set` **or** `Label` **missing from the builder**	Deliberate — see above. Use the typed Command action.
Fire lit before I finished	Also deliberate: the line already compiles.

See also

- README — installing, and the full action/feedback/variable list

- `companion/HELP.md` — the same material, in Companion's help panel
