Myro:Home Helper

HTTP API

Version 5.50

Controller status, decoded objects, typed control, integrations, graph data, and local discovery.

Myro Control, LLC | myrocontrol.com

Contents

About This API

- Controller families
- Capabilities
- Service Address
- API-Key Authentication
- Credentials by Route
- URL and JSON Rules
- Response Conventions

Endpoint Map

- Helper and Controller
- Typed Control and Audio
- Script Management
- Helper Services
- Local Integrations
- Graphing

Controller Reads

- Version
- Resolve the Controller ID
- Controller Status
- List Objects
- Get One Object
- Favorites
- Suggestions
- Timeline and Notes (5.50)
- Object Fields
- Recent Helper Log
- Synchronization
- Legacy Controller Property

Controller commands and security

Script Management

- Safe Authoring Workflow
- List Scripts
- AI Scripting Reference
- Get One Script
- Validate Source
- Save a Script
- Enable or Disable
- Reload Enabled Scripts

Legacy controller write routes

Helper Services

- Speech and Sound
- Push Notifications
- Logging

Lutron

- List Bridges
- List Devices
- List Pico Remotes
- Set Device Level
- Activate a LEAP Scene
- Tap an RA2 Button

Vestaboard and LaMetric

- Vestaboard
- LaMetric

Graphing API

- Graph Page
- Sensor List
- Graph Status for Web Control
- Sensor Data
- Panel Events (4.50)
- Browser Control (4.50)
- Ethernet Quality
- Connection Status

Mobile Discovery and Session Authentication

- Bonjour
- Identity
- Challenge
- Respond
- Session Inspection and Links
- Shared Rooms and Cameras
- Photo Library for PRO
- Home Photo for the Web Control
- Usage and Suggestions (5.50)

Internal Sonos Clip Transport

Request Examples

About This API

Myro:Home Helper exposes a local HTTP API for automation systems, MCP clients, diagnostic tools, and custom interfaces. This shared guide covers addresses, authentication, reads, script management and Helper services. Controller-specific command parameters and decoded panel fields are documented in the separate controller guides linked below.

This Markdown file is the source of truth for the bundled PDF. It is intentionally structured so an AI assistant can read it directly and generate requests without reverse-engineering example code.

Controller families

Helper 5.50 supports HAI / Leviton, NESS / ELK (preview), Savant (preview), and Control4 Link (preview), with different capabilities. Use the HAI and Leviton Controller Guide, NESS / ELK Controller Guide, Savant Controller Guide, or Control4 Controller Guide for the selected system.

For Savant, `GET /helper/{controllerID}/status` reports `controllerFamily: "Savant"`, `backend: "savant"`, native connection, `rooms`, `roomStates`, room/service counts, and capabilities. Savant service entries include `label`, `variantID`, `commandsAvailable` and optional `commands` / `commandsUpdatedAt`. `inventory` reports `available`, `refreshing`, optional `updatedAt` / `error`, `services` and `servicesWithCommands`. Source commands fail while current inventory or command discovery is unavailable. `capabilities.hap` means supported; `homeKit` separately reports `configured`, `enabled`, `bindingMatches`, `selectedActivities`, `running` and `phase` (`stopped`, `waitingForNetwork`, `running`, `failed`, `settingsUnavailable`) without pairing credentials. Unknown room fields remain omitted; check freshness and property timestamps. The scoped `/scripts/reference` includes Savant's contract and actual room/service IDs. Existing script-management routes can prepare Savant automation with the appropriate key permissions.

For Control4, `GET /helper/{controllerID}/status` reports `controllerFamily: "Control4"`, `backend: "control4"`, `protocol`, `connected`, `status`, `rooms`, `roomStates`, `inventory`, `homeKit` (bridge configuration and state, without pairing credentials), `objectCounts` (`Rooms`, `Activities`) and `capabilities`. Its `capabilities` are `rooms`, `av`, `scripting` and `hap` true, and `security`, `panelCommands`, `energyCost` and `ethernetQuality` false. Control4 has no panel objects: panel `/control`, `/security`, numbered objects, audio and Ethernet-quality routes do not apply. Since 5.50 a Control4 controller is offered for browser control, where its activities run and stop and its Timeline shows (see **Control4** under **Browser Control**), and to Myro:Home Pro for Apple TV. Unknown room fields are omitted rather than guessed. The scoped `/scripts/reference` includes the Control4 contract and actual room and activity IDs; use `MyroControl4` in reviewed scripts, as the Control4 Controller Guide describes. There is no public raw Control4 command route.

Authenticate HTTP clients with a Helper API key. Savant Local User credentials belong only in Helper's native Connection pane; Local User sign-in needs no cloud token or agent ID. Panel `/control`, `/security`, numbered objects, and legacy audio routes are not Savant media-command contracts. There is no public raw Savant command route. Since 5.50 a Savant controller is offered for browser control, where its activities run and stop (see **Savant** under **Browser Control**), and to Myro:Home Pro for Apple TV; use MyroSavant in reviewed scripts.

Capabilities

- Read controller identity, connection state, system troubles, phone-line state, synchronization timestamps, object counts, and Ethernet-quality facts.
- List all supported decoded objects or fetch one object by type and number.
- Read audio sources and their cached metadata, with an optional panel refresh request.
- Control configured objects using the selected controller's supported actions and parameters.
- Perform security actions using the selected controller's code-validation rules.
- Control configured Lutron LEAP/RA2, Vestaboard, and LaMetric devices.
- Read graph-ready sensor and Ethernet-quality history.
- Discover and authenticate Myro:Home mobile clients without publishing panel keys.
- Sign in a browser, or pair Myro:Home Pro for Apple TV with a six-digit code, for web control: shared favorites, doorbells, three days of home history, the Timeline and UniFi Protect camera events (browser-session routes).
- Read the Timeline from linked Myro:Home Pro apps (mobile-session routes).

Service Address

The Helper listens on TCP port `9393` by default:

```
http://HELPER_HOST:9393
```

`HELPER_HOST` is the IP address or Bonjour hostname of the Mac running Myro:Home Helper, for example `grouse-mac-mini.local`.

The server is intended for a trusted local network. It uses HTTP, not HTTPS. Do not expose port 9393 directly to the public Internet.

API-Key Authentication

Enable the API server and generate a key for a controller in the Helper's API tab. Most automation routes use this base URL:

```
http://HELPER_HOST:9393/API_KEY/api/v1
```

An API key is associated with one controller. Routes containing `{controllerID}` reject a key belonging to a different controller.

Every key starts with standard access: it can read status and scripts, validate script source, and control non-security devices. When generating a key, the person at the Helper Mac may independently grant either of these fixed capabilities:

- **Arm and disarm:** permits direct calls to `/security`. Code validation and zone bypass/restore rules follow the selected controller guide; do not assume a fallback user.
- **Remote script management:** permits `/scripts/save`, `/scripts/enable`, and `/scripts/reload`. This is code-execution authority: a managed script can use every local scripting API, including security control without a panel code.

Existing keys and keys without an explicit capability do not receive it. To change a key's capabilities, remove it and generate a new key.

Treat the API key as a credential. It appears in the URL, so avoid placing complete API URLs in public logs, screenshots, or source repositories.

Credentials by Route

Each route takes exactly one kind of credential:

Credential	How it is obtained	How it is sent	Routes
API key	Generated per controller in Helper's API tab	In the path: `/API_KEY/api/v1/...`	The keyed groups in **Endpoint Map**
Browser session	`POST /helper/web/session` with a Control Login or an API key for that controller, or **Pair by code** (an Apple TV)	`Authorization: Bearer` (or the `myro_session` cookie), always with `X-Myro-Origin`; the camera stream's WebSocket sends the cookie with a matching `Origin` instead	`/helper/web/...`; see **Browser Control**
Mobile session	`POST /helper/auth/challenge` and `/helper/auth/respond`, proving the panel keys (the Myro:Home apps)	`Authorization: Bearer`	`/helper/auth/whoami`, `/helper/auth/revoke`, `/helper/links`, `/helper/roomSync`, `/helper/cameras`, `/helper/syncStatus`, `/helper/rooms...`, `/helper/photos...`, `/helper/background` and `/helper/timeline...`
None			`/`, `/helper/control` and its files, `/helper/tablet`, `/helper/graphs...`, `/helper/ethernet`, `/helper/identity`, `/helper/auth/challenge`, `/helper/auth/respond`, `GET /helper/web/controllers`, and, with `X-Myro-Origin`, `GET` and `POST /helper/web/session`, `/helper/web/graphs/status` and the two `/helper/web/pair` routes

The credentials never stand in for one another: an API key does not open a browser or mobile route, a browser token (a paired Apple TV's included) does not open a mobile route, and a mobile token does not open a browser route. Most 5.50 features are browser-session only: changing favorites, doorbell marks, the three-day home history (`/helper/web/history`), the browser Timeline and its picture switches, Savant and Control4 activity Run and Stop, and every UniFi Protect route. On its keyed routes an API key reads only the shared favorites (`GET /helper/{controllerID}/favorites`), the household's usage-ranked suggestions (`GET /helper/{controllerID}/suggestions`, never the usage events themselves), the Timeline's moments and events without their pictures (`GET /helper/{controllerID}/timeline`) and the `doorbell` field on objects, and writes only notes and named moments into the Timeline (`POST /helper/{controllerID}/timeline/notes`; see **Timeline and Notes**); it can still sign in a browser session through `POST /helper/web/session`. `/helper/{controllerID}/history/{type}/{number}` is one sensor's graph history, not the home history. Linked Myro:Home Pro apps read the Timeline with their mobile session at `/helper/timeline` (see **Timeline for PRO**).

URL and JSON Rules

- Replace path placeholders such as `{controllerID}` and `{number}` without the braces.
- Percent-encode spaces and reserved characters in path values. `Auxiliary Sensor` becomes `Auxiliary%20Sensor`.
- Send POST bodies as UTF-8 JSON with `Content-Type: application/json`.
- Boolean inputs accept JSON booleans. The typed control endpoint also accepts common string and numeric forms.
- Temperatures use the controller's configured Fahrenheit or Celsius format.
- Object numbers are the Helper object numbers shown in Edit Objects, not database row IDs.

Response Conventions

New 4.07 routes return JSON and normally include `success`.

HTTP status	Meaning
`200`	Request was accepted or data was returned. Inspect `success` where present.
`400`	Invalid controller, action, field, or value.
`401`	Invalid API key, invalid session, or rejected security code.
`403`	The API key lacks the required security, script-management, or property capability.
`429`	Security, sign-in or event-refresh rate limit; follow the response's retry guidance.
`404`	Controller, object, integration, or resource was not found.
`503`	The controller is offline or did not acknowledge a command.

For typed controller routes, consult the controller guide for the meaning of `accepted: true`. It does not promise that a physical load changed; use a later object read when end-state confirmation matters.

Endpoint Map

All paths in the first five groups below are appended to the keyed base URL.

Helper and Controller

Method	Path	Purpose
GET	`/helper/version`	Helper version and build
GET	`/helper/getControllerID`	Controller ID associated with the API key
GET	`/helper/{controllerID}/status`	Complete decoded controller snapshot
GET	`/helper/{controllerID}/objects`	All decoded objects, optionally filtered by type
GET	`/helper/{controllerID}/object/{type}/{number}`	One decoded object
GET	`/helper/{controllerID}/favorites`	The web control's shared favorites (read-only)
GET	`/helper/{controllerID}/suggestions`	The household's "Right now" suggestions, ranked from usage (read-only; see **Usage and Suggestions**)
GET	`/helper/{controllerID}/timeline`	The Timeline's moments and events for the last 1 to 24 hours, without pictures (read-only; see **Timeline and Notes**)
POST	`/helper/{controllerID}/timeline/notes`	Write a note or a named moment into the Timeline (see **Timeline and Notes**)
GET, POST	`/helper/{controllerID}/events`	Cached panel events; POST starts a bounded panel download (see **Panel Events**)
GET	`/helper/{controllerID}/history/{type}/{number}`	One sensor's graph history, as `/helper/graphs/data/{type}/{number}`
GET	`/helper/{controllerID}/log`	Recent Helper log lines
GET	`/helper/{controllerID}/syncNow`	Start names and object synchronization
GET	`/helper/{controllerID}/syncClock`	Set the controller clock from the Mac
GET	`/helper/{controllerID}/controller/{controllerItem}`	Legacy single-property read

Typed Control and Audio

Method	Path	Purpose
POST	`/helper/{controllerID}/control`	Typed controller control actions
POST	`/helper/{controllerID}/security`	Code-validated security actions
GET	`/helper/{controllerID}/audio/sources`	Sources and cached metadata
GET	`/helper/{controllerID}/audio/source/{number}/metadata`	One source's metadata
GET	`/helper/{controllerID}/audio/{zone}/{cmd}/{value}`	Legacy audio-zone control
GET	`/helper/{controllerID}/unit/{number}/{value}`	Legacy unit control
GET	`/helper/{controllerID}/flag/{number}/{value}`	Legacy flag control
GET	`/helper/{controllerID}/output/{number}/{value}`	Legacy output control
GET	`/helper/{controllerID}/controller/setEnergyCost/{level}`	HAI / Leviton legacy energy-cost control; see its controller guide

Script Management

Method	Path	Purpose
GET	`/helper/{controllerID}/scripts`	List scripts and run metadata without source
GET	`/helper/{controllerID}/scripts/reference`	Current AI scripting guide and live device inventory
GET	`/helper/{controllerID}/scripts/{identifier}`	Read one script by ID or exact name
POST	`/helper/{controllerID}/scripts/validate`	Parse and lint source without executing it
POST	`/helper/{controllerID}/scripts/save`	Create or update a validated script (script-management key required)
POST	`/helper/{controllerID}/scripts/enable`	Enable or disable one script (script-management key required)
POST	`/helper/{controllerID}/scripts/reload`	Re-run all enabled scripts and rebuild handlers (script-management key required)

Helper Services

Method	Path	Purpose
GET	`/helper/beep`	System beep on the Helper Mac
GET	`/helper/alert/{message}`	Show an alert on the Helper Mac
GET	`/helper/setSpeechStartDelay/{number}`	Set initial speech delay
GET	`/helper/setSpeechDelay/{number}`	Set delay between speech items
GET	`/helper/speak/{message}`	Queue macOS speech; optional `?voice=`
GET	`/helper/voices`	Installed voices, the System voice and the voice in use
GET	`/helper/speakSound/{filename}`	Queue a sound in the speech sequence
GET	`/helper/playSound/{filename}`	Play a sound asynchronously
GET	`/helper/stopSound/{filename}`	Stop the current sound
GET	`/helper/{controllerID}/sendTestNotification`	Send a configured test notification
GET	`/helper/{controllerID}/sendNotification/{message}`	Send a push notification
GET	`/helper/{controllerID}/log/{message}`	Add a controller-scoped log message
GET	`/helper/logToExternalFile/{filename}/{message}`	Append to a file in the Logs folder

Local Integrations

Method	Path	Purpose
GET	`/helper/vestaboards`	List configured Vestaboards
POST	`/helper/vestaboard/send`	Send a Vestaboard message
GET	`/helper/lametrics`	List configured LaMetric devices
POST	`/helper/lametric/send`	Send a LaMetric notification
GET	`/helper/lutron/bridges`	List Lutron LEAP and RA2 bridges
GET	`/helper/lutron/devices`	List discovered Lutron devices
GET	`/helper/lutron/picos`	List LEAP Pico/keypad remotes and buttons
POST	`/helper/lutron/control`	Set a Lutron device level

Method	Path	Purpose
POST	`/helper/lutron/scene`	Activate a LEAP scene
POST	`/helper/lutron/button`	Tap an RA2 keypad/Pico component

Graphing

Graph routes are read-only and are available both at the server root and below the keyed base URL.

Method	Root path	Purpose
GET	`/helper/graphs`	Interactive graph page
GET	`/helper/graphs/sensors`	Sensors enabled for graph logging
GET	`/helper/graphs/data/{type}/{number}`	Parsed sensor CSV data
GET	`/helper/graphs/ethernet`	Ethernet-quality history and assessment
GET	`/helper/graphs/status`	Live connection state and active messages for the graph page
GET	`/helper/web/graphs/status`	Whether logging is on and any history exists (web control's empty state)
GET	`/helper/graphs/js/{file}`	Bundled graph JavaScript asset
GET	`/helper/graphs/img/{file}`	Bundled graph image asset
GET	`/helper/graphs/img/touch-icon.png`	The Helper's application icon as a 512 × 512 PNG on white, for a tablet's home screen (web control's `apple-touch-icon` and manifest icon); cached for a day

Controller Reads

Version

`GET /helper/version`

```
{
  "version": "5.50",
  "build": "2962"
}
```

Resolve the Controller ID

`GET /helper/getControllerID`

```
{
  "success": true,
  "controllerID": "953d8997-ab41-4aec-9f58-78235e36d46f"
}
```

Use this once when an integration has an API key but has not stored its controller GUID.

Controller Status

`GET /helper/{controllerID}/status` returns a coherent snapshot from Helper's controller store plus live connection-quality state. Common fields include identity, connection, firmware, troubles, sync timestamps and object counts. The `protocol` field is a free-text controller type; `controllerFamily` (string, "HAI / Leviton", "NESS / ELK", "Savant" or "Control4") is a stable label for integrations. Optional fields and solar/energy/battery meanings are described in the selected controller guide. Missing timestamps are JSON `null`.

Ethernet ACK timeout meaning

`ackTimeouts` is cumulative since the Helper launched. An ACK timeout records a missed command confirmation; protocol details are in the controller guide. It is a missed confirmation, not proof that the command failed, the controller disconnected, or the panel Ethernet hardware is defective.

Use the graph endpoint's historical `assessment` when you need a recurrence level. Do not diagnose a panel from one timeout or from a lifetime total.

List Objects

`GET /helper/{controllerID}/objects` lists configured supported objects. Optional query `?type=Zone` matches a type case-insensitively; omit it for all supported types. The response contains `success`, `count` and `objects`. Consult your controller guide for available families and number mappings.

Get One Object

`GET /helper/{controllerID}/object/{type}/{number}`

Example:

```
curl "$BASE/helper/$CONTROLLER/object/Thermostat/2"
```

Returns `{"success":true,"object":{...}}`, or `404` when the object is absent.

Favorites

`GET /helper/{controllerID}/favorites` returns the objects starred in web control for this controller: `{"success":true,"favorites":["Unit:12","Zone:3"],"revision":4}`. Each id is the object's `type` and `number` joined by a colon, in the order they were starred. `revision` goes up by one on every change. The list is shared by every browser signed in to the controller; this route is read-only, for apps that show the same favorites.

Suggestions

`GET /helper/{controllerID}/suggestions?kind=web&home=1&at=2026-10-28T21:30:00-07:00&limit=6` returns the controller's "Right now" suggestions ranked from the household's usage, exactly the reply of `GET /helper/suggestions` described under **Usage and Suggestions** (5.50). An API key has no device of its own, so there is no `device` parameter (one sent is ignored), `personal` is 0 for every suggestion and the list has no recents: the reply is the household ranking plus the shared favorites (see **The smart list** there). `kind` is optional and defaults to `web`; `home` is `0`, `1` or omitted (unknown); `at` is an ISO 8601 time with offset, default now in the Helper's zone; `limit` is 1 to 24, default 6 (12 for `kind=tv`); `compose=0` returns the plain ranking (at most 12). The route is read-only and never returns usage events; an API key cannot read, upload or delete usage. A Helper older than 5.50 answers `404`.

Timeline and Notes (5.50)

`GET /helper/{controllerID}/timeline?hours=8` returns the controller's Timeline for the last `hours` (1 to 24, default 8, ending now): the same JSON as `GET /helper/web/timeline` under **Browser Control** (`version`, `cursor`, `window`, `available`, `gaps`, `headline`, `counts`, `current`, `moments`, `lanes`, `flags`, `events` and `eventsTruncated`) minus `settings`, each moment's `stills` and each event's `stillID`. An API key cannot fetch a picture, so none is named. There is no `end` or `after` parameter; every reply is the whole window. The route is read-only and replies with `Cache-Control: no-store`. A bad `hours` is `400 {"success": false, "code": "badParameter", "error": "hours must be 1 to 24."}`; a key that is not the controller's is `400 {"success": false, "code": "badRequest", "error": "Invalid Controller ID."}`. A Helper older than 5.50 answers `404`.

`POST /helper/{controllerID}/timeline/notes` writes a note, or a named moment, into the controller's three-day history with `source: "api"`, exactly as a script's `MyroTimeline.note` and `MyroTimeline.moment` do (the Scripting Guide's **MyroTimeline** section; the route and the scripting global share one validator). The body is a JSON object of at most 16 KB:

```
{"title": "Pool pump started", "kind": "power", "object": "Pool Pump", "room": "Backyard",
 "at": "2026-10-29T19:42:30Z", "importance": 0, "detail": "Started by the pool schedule"}
```

`title` is required, at most 80 characters. `kind` is one of the event kinds `lights`, `door`, `window`, `motion`, `security`, `climate`, `lock`, `audio`, `media`, `scene`, `power`, `system`, `doorbell` or `note` (the default; `moment` is not a note kind). `object` is a favorites id (`"Unit:12"`, `"Zone:7"`, `"Activity:<room>:<id>"`) or an object's name, accepted when it matches exactly one object and stored as the id with its `objectType`, `objectNumber` and room. `room` is a room name. `at` is ISO 8601 with or without fractional seconds (default now; it must be within the last 72 hours and not in the future). `importance` is `0` routine, `1` notable or `2` headline (default `1`). `detail` is at most 200 characters. Unknown fields are ignored.

Adding `"moment": {...}` writes a *named moment anchor* instead: an event of kind `moment` with importance 2, which the Timeline turns into a moment with exactly this title (never renamed, never merged into another moment's name), reaching ten minutes either side of its anchor. The `moment` object takes `symbol` (an SF Symbol name of at most 40 letters, digits and dots; default `sparkles`), `tone` (`info`, `good`, `warn` or `alert`; default `info`), `objects` (an array of ids or names the moment covers, at most 50, each of which must resolve; a lone string is accepted), and `start` and `end` (ISO 8601; default now and open; `at` stands in for a missing `start`). `kind` and `importance` are ignored for a moment; `detail` and `room` apply:

```
{"title": "Movie night", "detail": "Started from the couch",
 "moment": {"symbol": "film", "tone": "good", "objects": ["Unit:12", "Reading Lamp"]}}
```

The reply is `201 {"success": true, "id": "note-…" or "moment-…", "kind", "title", "time"}`. A missing, mistyped or out-of-range field is `400 {"success": false, "code": "badRequest", "error": "…"}` with one sentence saying what (`"The title is longer than 80 characters."`, `"No object matches \"Pool Pump\"."`, `"The time is in the future."`); nothing is written or counted. Each API key may write 120 notes and moments an hour; the next is `429 {"success": false, "code": "tooMany", "error", "retryAfter"}` with a `Retry-After` header, and a refused body never counts against the hour. The event is handed to Helper's Timeline queue and the reply does not wait for it; in the moment before that queue exists at launch the route answers `503 {"success": false, "code": "notReady"}`. Helper writes one log line per note, naming the controller, the remote host and the title, never the key.

A note never notifies anyone, never takes a picture and never sends a command. It appears in `GET /helper/{controllerID}/timeline`, in `/helper/web/timeline` and `/helper/timeline`, in a script's `MyroTimeline.events()` and, when its kind is named in `kinds`, in `/helper/web/history`. Helper keeps 1,000 script and API events per controller in the three-day store, oldest dropped first, apart from the 6,000 panel events. There is no route to change or delete a note; a script's `MyroTimeline.end(id)` closes an open moment written here as well. The scripting rules `mute`, `rename` and `mode` have no HTTP form.

Object Fields

Objects expose common fields such as `number`, `type`, `state`, `value`, `area`, `customName`, `hidden`, `room` and `updatedAt`. Decoded fields and unavailable values depend on the controller; use its guide and live inventory. Raw values remain available alongside decoded fields. A zone, flag or output marked "Treat as doorbell" in web control also carries `"doorbell": true` (5.50); the marks are listed and changed only through the browser route `/helper/web/doorbells` (see **Doorbells**).

On HAI and Leviton panels, thermostats report `holdMode` (`"off"`, `"hold"` or `"vacation"`) beside the 0/1 `hold` (Helper 5.01 and later); vacation hold cannot be set through the API. See the HAI guide.

On HAI and Leviton panels, access-control locks also report their reader's last access: granted or denied, the Omni user number (never a code) and when Helper received it. Helper requests reader status with lock status on every status refresh. Field names and user-number meanings are in the HAI guide.

Recent Helper Log

`GET /helper/{controllerID}/log?lines=100`

`lines` defaults to 100 and is clamped to 1 through 500.

```
{
  "success": true,
  "count": 2,
  "lines": [
    "2026-09-03 10:12:02 - Grouse - System - Connection established.",
    "2026-09-03 10:12:04 - Grouse - Sync - Object properties completed."
  ]
}
```

Synchronization

`GET /helper/{controllerID}/syncNow` starts names and object synchronization and returns immediately:

```
{"success":true,"message":"name and object synchronization started"}
```

`GET /helper/{controllerID}/syncClock` sets the controller clock from the Helper Mac.

Legacy Controller Property

`GET /helper/{controllerID}/controller/{controllerItem}` reads one allow-listed controller property. New integrations should prefer `/status`, which returns a consistent decoded snapshot in one request.

Controller commands and security

`POST /helper/{controllerID}/control` is the typed controller write endpoint. `POST /helper/{controllerID}/security` is the code-validated direct security endpoint. Audio reads use `GET /helper/{controllerID}/audio/sources`, `GET /helper/{controllerID}/audio/source/{number}/metadata` and the legacy `/helper/{controllerID}/audio/{zone}/{cmd}/{value}` route.

Use your controller guide for action parameters, object numbers, supported hardware, security-code rules, audio behavior and acceptance semantics. On HAI and Leviton panels, Helper 5.01 adds the `blink_upb` action, which blinks a UPB device for 30 seconds to find it; message `say`, `log` and `phone`; `keypad_beepers`; `radiora2_phantom`; `leviton_scene`; and all-door and timed unlock through `set_lock`.

- HAI and Leviton Controller Guide
- NESS / ELK Controller Guide

An API key must have direct security enabled to use `/security`; otherwise requests receive `403`. Script management is a separate, highly privileged capability because an enabled script can perform controller actions. A transport timeout leaves command outcome uncertain; read current state before deciding on another action.

Five rejected panel codes lock the direct security route for fifteen minutes and return `429` with `retryAfter` in seconds. A validation timeout or disconnection does not count as a rejected code. Every attempt is logged. Keep panel codes out of source control and send them only over a trusted local network.

Script Management

These authenticated routes let MCP clients and other trusted local tools manage the same per-controller scripts shown in the Helper's Scripts window. They use the retained `ScriptManager`; they do not write around handler ownership, timers, schedules, or Sonos phrase caching.

Script management is intentionally separate from the JavaScript runtime API. A running script cannot call these routes through a built-in `myro.saveScript()` function and cannot rewrite itself. The management client validates and stores source; the script executes only when enabled, manually run, scheduled, or triggered.

Listing scripts, reading source, and validating proposed source require any valid key. Saving, enabling, disabling, and reloading require a key generated with **remote script management** enabled. This permission authorizes arbitrary local automation code, including security-capable scripts. Never grant it to an untrusted client or expose the Helper API or MCP credentials directly to the public Internet.

In 4.51, `MyroSystem.runUserScript` also permits a managed script to launch a user-installed executable outside Helper's sandbox, but only after a local per-controller **User-Installed Scripts** opt-in in the Scripting window. This further increases the authority of script-management credentials. The HTTP API cannot enable that grant or install executables. Validation does not execute or verify the external program. See the Scripting Guide for installation, output limits, and timeout behavior.

There is no direct HTTP process-execution endpoint. A successful script save, enable, reload, or validation response does not confirm external-program completion. The JavaScript callback carries the process outcome; read the script log when diagnosing it. External-script arguments and output are not automatically logged. Do not expose secrets in arguments or add indiscriminate stdout/stderr logging. Timeout, disable, and permission revocation do not terminate a process that is already running; never automatically replay a physical operation after an uncertain outcome.

Safe Authoring Workflow

1. Read `/scripts/reference` and the relevant live device inventories.
2. Read the existing script and retain its `scriptID` when editing.
3. Validate the proposed source with `/scripts/validate`.
4. Save it disabled while reviewing or testing a new automation.
5. Enable it only after the user approves activation.

Saving with `enabled:true`, enabling a script, or reloading scripts may execute top-level JavaScript immediately. Top-level code can control physical equipment; event-handler bodies wait for their event.

List Scripts

`GET /helper/{controllerID}/scripts`

Returns metadata without source so listing remains compact:

```
{
  "success": true,
  "count": 1,
  "scripts": [
    {
      "scriptID": "D32AE430-703E-4E5D-8D1D-E63896298241",
      "name": "Office lights message",
      "enabled": false,
      "category": "automation",
      "runCount": 0,
      "lastRunStatus": ""
    }
  ]
}
```


AI Scripting Reference

`GET /helper/{controllerID}/scripts/reference?compact=true`

Returns `reference`, the shared Helper scripting guide plus the resolved controller family's explicit reference, followed by names from the selected controller and configured Sonos, Lutron, Pico, Vestaboard, and LaMetric integrations. This is the preferred context for AI script generation because it reflects the installed Helper version and the user's real names.

`compact` defaults to `true`. Use `compact=false` when the complete guide is needed.

Get One Script

`GET /helper/{controllerID}/scripts/{identifier}`

`identifier` can be the immutable `scriptID` or an exact case-insensitive name. Percent-encode names in the URL. Duplicate names are rejected; use `scriptID` for edits.

The response includes `script.code` plus metadata and last-run information.

Validate Source

`POST /helper/{controllerID}/scripts/validate`


```
{"code":"MyroSystem.log('validation only');"}
```


Validation parses and lints but never executes the source. It reports syntax errors, refused dynamic-code constructs, missing-device warnings, and potential performance concerns:

```
{
  "success": true,
  "validation": {
    "valid": true,
    "message": "Script is valid",
    "errors": [],
    "warnings": []
  }
}
```

Save a Script

`POST /helper/{controllerID}/scripts/save`

```
{
  "scriptID": "D32AE430-703E-4E5D-8D1D-E63896298241",
  "name": "Office lights message",
  "code": "ScriptEvents.onUnitChange('Office Lts', function(name, isOn, level) { /* ... */ });",
  "enabled": false,
  "category": "automation",
  "tags": "office,notification",
  "notes": "Created through MCP"
}
```

`name` and `code` are required. `scriptID` is recommended for updates. Without it, an exact unique name updates the existing script; otherwise a new script is created. New scripts default to disabled. For an existing script, omitting `enabled`, `category`, `tags`, or `notes` preserves that field.

The Helper validates before saving and rejects invalid source. The API limit is 500 KB per script. A key without remote script-management permission receives `403`; this decision is based on the key's fixed capability, not a fallible scan of the JavaScript text.

Enable or Disable

`POST /helper/{controllerID}/scripts/enable`

```
{"scriptID":"D32AE430-703E-4E5D-8D1D-E63896298241","enabled":true}
```

`name` can be used instead of `scriptID`. Invalid scripts cannot be enabled. Disabling unregisters handlers and cancels that script's schedules and timers; enabling executes its top-level body and registers handlers.

Reload Enabled Scripts

`POST /helper/{controllerID}/scripts/reload`

Use an empty JSON body: `{}`. The Helper reloads storage and re-runs every enabled script, including top-level control or security commands. A reload requires remote script-management permission, is not required after save or enable, and should be reserved for an explicit maintenance or recovery request.

Legacy controller write routes

Prefer typed control. Legacy compatibility routes include `/helper/{controllerID}/unit/{number}/{value}`, `/helper/{controllerID}/flag/{number}/{value}`, `/helper/{controllerID}/output/{number}/{value}`, and `/helper/{controllerID}/controller/setEnergyCost/{level}`. Consult the controller guide for support and semantics; a legacy URL does not make an unsupported panel feature available.

Helper Services

Speech and Sound

The following routes operate on the Helper Mac:

- `GET /helper/beep`
- `GET /helper/alert/{message}`
- `GET /helper/setSpeechStartDelay/{number}` - milliseconds before the first speech item
- `GET /helper/setSpeechDelay/{number}` - milliseconds between queued speech items
- `GET /helper/speak/{message}` - optional `?voice=` (identifier or name)
- `GET /helper/voices` - optional `?refresh=1`
- `GET /helper/speakSound/{filename}`
- `GET /helper/playSound/{filename}`
- `GET /helper/stopSound/{filename}`

Sound filenames are relative to the Helper's Scripts folder. Percent-encode messages and filenames placed in a URL path. Helper speaks with the Mac's System voice (System Settings ▸ Accessibility ▸ Spoken Content ▸ System voice); choose a Siri, Premium or Enhanced voice there for a more natural voice. Sounds queued with `speakSound` play between speech items unchanged.

Voice

Helper speaks with the Mac's System voice (System Settings ▸ Accessibility ▸ Spoken Content ▸ System voice); choose a Siri, Premium or Enhanced voice there for a more natural voice. The same voice is used by scripting speech, MCP's `speak` tool and Sonos announcements, and a change takes effect on the next message. A Siri System voice works because messages without `?voice=` are spoken by the Mac's own speech engine; a named voice is spoken by AVSpeechSynthesizer, which cannot use Siri voices. With a Siri System voice, Helper keeps the voice loaded in the background. macOS can take up to a minute to load a Siri voice for an app, so speech right after launch waits for it (up to 60 s, logged as `Speech: waiting for the System voice to load…`) and is then spoken in Siri; the Log window says when the System voice is ready. If the System voice cannot speak, Helper falls back to the best installed voice for the Mac's language (Premium, then Enhanced). Helper says Myro as "Mai-row" (MY-row).

`/helper/speak/{message}?voice=Ava` speaks one message in another voice. `voice` is an identifier (`com.apple.voice.premium.en-US.Ava`) or a name (`Ava`); percent-encode it. A voice that is not installed is not an error: the message is spoken in the System voice and the Log window says so once. The reply reports the voice that will be used, or `system` for the system default:

```
{"speak": "Front door is open", "delay": 0, "voice": "com.apple.voice.premium.en-US.Ava"}
```

If the voice produces no audio, the message is spoken with the Mac's speech synthesizer instead and a `Speech:` line is logged.

`GET /helper/voices` lists what this Mac has. Voices are cached for five minutes; `?refresh=1` re-reads them after downloading one.

```
{
  "language": "en-US",
  "selected": "automatic",
  "automatic": {"identifier": "com.apple.voice.premium.en-US.Ava", "name": "Ava", "language": "en-
      US", "quality": "premium", "category": "premium", "group": "main"},
  "current": {"identifier": "com.apple.voice.premium.en-US.Ava", "name": "Ava", "language": "en-US",
      "quality": "premium", "category": "premium", "group": "main"},
  "voices": [
    {"identifier": "com.apple.voice.premium.en-US.Ava", "name": "Ava", "language": "en-US",
      "quality": "premium", "category": "premium", "group": "main"},
    {"identifier": "com.apple.voice.compact.en-US.Samantha", "name": "Samantha", "language": "en-
      US", "quality": "standard", "category": "standard", "group": "main"},
    {"identifier": "com.apple.voice.compact.de-DE.Anna", "name": "Anna", "language": "de-DE",
      "quality": "standard", "category": "standard", "group": "more"},
    {"identifier": "com.apple.speech.synthesis.voice.Bubbles", "name": "Bubbles", "language": "en-
      US", "quality": "standard", "category": "novelty", "group": "more"}
  ]
}
```


`selected` is `automatic` (Helper follows the System voice). `automatic` is the fallback voice used if the System voice cannot speak (a Siri System voice is reported here as the standard voice macOS offers apps), and `current` is the same unless a voice is chosen; either is `null` when there is none. `quality` is `siri`, `premium`, `enhanced` or `standard`, as macOS reports it. `category` is the same except that novelty voices — the sound-effect voices (Bubbles, Zarvox, Whisper, ...) and the character voices (Eddy, Flo, Grandma, Grandpa, Reed, Rocko, Sandy, Shelley) — are `novelty`. `group` is `main` for the Mac's language (novelty excluded) and `more` for everything else. `voices` is ordered: `main` best first (Siri, Premium, Enhanced, Standard; the Mac's region first), then each other language, then the Mac's language's novelty voices.

Push Notifications

- `GET /helper/{controllerID}/sendTestNotification`
- `GET /helper/{controllerID}/sendNotification/{message}`

Before using these routes, register each receiving device. In Helper, select the controller, enable the appropriate app under **Notifications > App**, and display the QR code under **Notifications > Configuration**. In the Myro:Home iOS or Android app, open that controller's settings, scroll to **Push Notifications**, tap the scanner icon, and scan Helper's QR code. Repeat for every device that should receive alerts, then use **Send Test Push Notification** in Helper to confirm delivery.

Logging

`GET /helper/{controllerID}/log/{message}` adds a controller-scoped System entry to the main Helper log.

`GET /helper/logToExternalFile/{filename}/{message}` appends text to a file below the Helper Logs directory. Use a simple relative filename. This legacy endpoint is intended for trusted local automation; it is not a general upload API.

Lutron

Bridge IDs are unique across configured LEAP and RA2 bridges. Responses identify the protocol.

List Bridges

`GET /helper/lutron/bridges`


```
{"success":true,"bridges":[{"id":1,"name":"Main Bridge","protocol":"LEAP"}]}
```


List Devices

`GET /helper/lutron/devices`

Optional query: `?bridgeID=1`.

LEAP devices use a resource path such as `/zone/1`. RA2 devices use an integration ID.

List Pico Remotes

`GET /helper/lutron/picos`

Returns discovered LEAP Pico/keypad remotes, including unassigned remotes that still produce scripting events. Each item includes its bridge ID, name, href, remote type, and button definitions.

Set Device Level

`POST /helper/lutron/control`

LEAP:


```
{"bridgeID":1,"deviceHref":"/zone/1","level":50}
```


RA2:


```
{"bridgeID":2,"integrationID":5,"level":50}
```


For compatibility, an RA2 integration ID can be supplied as a numeric `deviceHref` string.

Activate a LEAP Scene

`POST /helper/lutron/scene`


```
{"bridgeID":1,"sceneHref":"/virtualbutton/1"}
```

Tap an RA2 Button

`POST /helper/lutron/button`


```
{"bridgeID":2,"integrationID":6,"component":2}
```

Vestaboard and LaMetric

Vestaboard

`GET /helper/vestaboards` lists configured boards.

`POST /helper/vestaboard/send`:

```
{
  "id": 1,
  "lines": ["LINE ONE", "LINE TWO"],
  "persist": false,
  "timeout": 30
}
```

`board` can be used instead of `id` by legacy clients. With `persist:false`, the previous message is restored after the optional timeout.

Exactly one of three layouts is required:

Field	Layout
`lines`	One string per row, each centered horizontally, placed from the top. The original behaviour, unchanged.
`text`	Free text. The Helper wraps it to the board's width, honours `\n`, and centers the block vertically and horizontally for that board's size (Flagship 6x22, Note 3x15).
`characters`	An array of rows, each an array of integers 0-71, sent as an exact grid. Short rows and missing rows are padded; a code outside 0-71 returns an error.

```
{ "id": 1, "text": "Garage door has been open\nfor 20 minutes" }
```

LaMetric

`GET /helper/lametrics` lists configured devices.

`POST /helper/lametric/send`:

```
{
  "id": 1,
  "text": "Garage door open",
  "icon": "i73576",
  "sound": "",
  "priority": "warning",
  "cycles": 1
}
```

Priority is `info`, `warning`, or `critical`.

Graphing API

Graph routes are read-only and intentionally available without an API key at the server root. This permits a browser or Myro:Home app to open graphs without placing a controller-control credential in a shared URL.

Graph Page

`GET /helper/graphs?controller={controllerID}`

The root page at `http://HELPER_HOST:9393/` lists configured controllers and opens the browser control sign-in. Graphs remain available directly at the URL above, and from the control page.

Sensor List

`GET /helper/graphs/sensors?controller={controllerID}`

Returns thermostats and auxiliary sensors with CSV logging enabled, the controller temperature format, display name, and whether each sensor currently has data.

Graph Status for Web Control

`GET /helper/web/graphs/status?controller={controllerID}`

Used by web control to explain an empty Graphs view. It needs the same-origin header web control sends (`X-Myro-Origin`) but no sign-in, like the other graph reads.

```
{"controllerID": "...", "loggingEnabled": true, "loggableSensorCount": 20, "thermostatCount": 2,
        "sensorCount": 18, "loggedObjectCount": 20, "hasHistory": false}
```

- `loggableSensorCount`, `thermostatCount`, `sensorCount`: thermostats and auxiliary sensors known to Helper for the controller (ELK keypad and zone temperatures count as sensors).
- `loggingEnabled`: at least one of them is ticked in the Logging tab's Data Log Object Types list; `loggedObjectCount` is how many.
- `hasHistory`: at least one logged object has recorded data on disk.

Sensor Data

`GET /helper/graphs/data/{type}/{number}?controller={controllerID}`

Optional parameters:

- `from={ISO-8601}` and `to={ISO-8601}` select the date range. Include a timezone, for example `2026-09-01T00:00:00Z`.
- `since={ISO-8601}` is retained as a lower-bound alias when `from` is absent.
- `points={number}` requests a display budget, clamped to 96-4096 rows (default 1200).
- `removeOutliers=false` disables the normal temperature/humidity outlier filter.
- `runtimeBucket=auto|hour|day|week` selects runtime chart intervals. Auto uses hours for up to seven days, days for up to 180 days, and weeks for longer ranges. Responses are limited to 2,048 runtime

buckets; narrow the range or choose a coarser interval if needed.
- `timeZone={IANA identifier}` selects calendar boundaries, for example `America/Los_Angeles`. The default is the Helper Mac's timezone; the browser requests its own timezone. This does not reinterpret legacy CSV timestamps.

In 4.50 this endpoint incrementally indexes the live CSV and its history archives in a rebuildable SQLite cache. On a new append, the live file is archived at 8 MiB or a calendar-month boundary, then a new live file begins. Archives are compressed after validation and kept indefinitely by default. Existing large CSVs are preserved as recovery chunks rather than rewritten or truncated. Initial indexing returns `indexing: true` and `progress` (0-1); retry after a short delay. It does not return a partial index as a completed result.

Responses include `sampleCount`, `returnedCount`, `availableFrom`, `availableTo`, `stats`, `humidityStats`, `skippedRows`, `flaggedCount`, `data`, and `flagged`. Statistics cover the selected range after optional outlier filtering but **before** display reduction. `skippedRows` counts malformed source rows across the indexed file. Flag annotations are limited to 500; `flaggedCount` is the full count for the range.

The display reduction retains each bucket's endpoints and per-field extrema plus the first/last step or gap boundary, then fills the rest of the bucket's budget with evenly spaced readings. `summarizedTransitions: true` marks that some readings were left out; the graph draws stepped lines through what remains (each value holds until the next). Narrow the date range for every reading. `gapBefore` marks a silence longer than 2 hours, where the graph line breaks.

`runtime` reports heating, cooling, humidifying and dehumidifying seconds from consecutive, valid thermostat action reports no more than 2 hours apart: each reported action holds until the next report, and a longer silence is unknown. `observedSeconds` is covered time; `unknownSeconds` is not assumed idle. No period is extended from the final row to now. These are **reported-action interval estimates**, not metered equipment runtime. Missing logging, long unchanged periods and communication failures reduce coverage. Heating and cooling fields can overlap if both bits were reported. CSV times use the Helper Mac's local timezone; legacy rows have no offset, so historical DST ambiguity cannot be recovered.

`runtimeBuckets` contains interval rows with `from`, `to`, the four action durations, `offSeconds`, `observedSeconds`, and `unknownSeconds`. Boundaries use the returned `runtimeTimeZone`; daylight-saving days need not be 24 hours. `runtimeBucket` reports the resolved interval. `queryFrom` and `queryTo` delimit the requested coverage, including gaps outside recorded observations.

`archiveCount` reports the included archive sources. `archivePolicy` describes the local chunk/compression/retention settings; it is informational, not a remotely writable policy. `keepAll: true` is the default. `warnings` identifies unreadable sources: treat the result as incomplete, not as evidence of no activity. Archives remain the recoverable source data; SQLite and decompression scratch files are disposable caches. Compression bounds individual chunks, not total disk usage while keep-all retention is enabled.

The authenticated equivalent is `GET /{apiKey}/api/v1/helper/{controllerID}/history/{type}/{number}` with the same parameters. It requires a key scoped to that controller and returns the same payload.

Panel Events (4.50)

- `GET /{apiKey}/api/v1/helper/{controllerID}/events` reads the panel-event cache, newest first.
- `POST /{apiKey}/api/v1/helper/{controllerID}/events` with `{}` or `{"scope":"recent"}` starts a bounded, read-only download from the panel. Refreshes are coalesced and limited to one attempt per controller per 30 seconds. Recent reads request at most 100 records, stopping at a known record or the end of data.

- `{"scope":"all"}` explicitly reads past known records toward the panel's oldest available event. It reads one record at a time, with pacing, an eight-second per-record timeout, a 120-second overall limit and a 2,000-record ceiling. A disconnect ends the read without reconnecting or changing panel state. This cannot recover events the panel has already overwritten.

Optional read filters: `limit` (1-200, default 100), `offset`, `zone`, `area`, `eventType`, `month`, `day`. Up to 2000 records are retained per controller. This is not a guaranteed complete audit archive; activity beyond the panel's retained history or between widely separated refreshes may be unavailable.

Responses identify `source: "panel"`, `refreshing`, `events`, `total`, `offset`, `hasMore`, and any `refreshError`. `downloadScope` and `downloadedCount` describe the latest attempt. `panelEndReached` is true only after the panel reports end-of-data, not after reaching a limit or finding a cached record. Wait for `refreshing: false`, check errors and `panelEndReached`, and distinguish a completed request from a complete panel-history download. Each event includes the protocol event number/type/parameters, summary, optional zone/area/user attribution, and panel-local month/day/hour/minute.

New event downloads are limited to once every 30 seconds per controller; full (`all`) downloads additionally have a ten-minute cooldown. A rate-limited request returns `429` rather than presenting an old completed download as the new result. Reading or filtering the existing cache does not start another panel download.

The panel packet contains no year. `yearKnown` is false. `receivedAt` records when Helper downloaded the event, not when it happened. Do not infer the event year from it. These events are distinct from the `/log` endpoint's Helper diagnostics.

Browser Control (4.50)

`GET /helper/control?controller={controllerID}` opens the responsive controller interface. The page exchanges either a controller's Control Login username/password or a scoped Helper API key for a separate, in-memory browser bearer token using `POST /helper/web/session`. Tokens expire after two hours, or 15 minutes without a request, and are revoked on sign-out. Automatic status polling counts as activity. The token is also set as an `HttpOnly`, `SameSite=Strict` session cookie (`myro_session`, path `/helper/web`), so a reload or another tab in the same browser resumes the session through `GET /helper/web/session`, which answers `{"signedIn": false}` when there is no live session; page code cannot read the cookie, other sites cannot send it, and every browser request still needs the `X-Myro-Origin` header. The cookie lasts until the browser quits and is cleared on sign-out; it is not marked `Secure` because Helper serves plain HTTP. With `"remember": true` in the sign-in body (or `POST /helper/web/session/keep` with `{"remember": true}` or `false` later), the session is kept: it lasts 30 days from its last use instead of 2 hours / 15 minutes idle, survives Helper restarts, and its cookie gets the same 30 days (renewed about daily while in use). Helper saves kept sessions only as SHA-256 hashes of the token (and of the API key, for a key sign-in). A kept session ends on sign-out, **Sign Out All Browsers** in Helper, a changed or removed Control Login, or a revoked API key. `GET /helper/web/session` and the sign-in reply include `remember`. Revoking an API key invalidates its browser tokens. Saving or removing a Control Login invalidates its previous sessions without changing API keys.

Create one Control Login per controller locally under **API > Control Login**. Passwords require at least 15 characters and at most 1,024 UTF-8 bytes. Helper stores a random 32-byte salt and a PBKDF2-HMAC-SHA256 verifier with 600,000 iterations in a device-local Keychain item. It does not store the password or synchronize the item through iCloud. Password checks are rate-limited by source address and concurrent

verification is bounded. The login provides browser access only, not API-key, mobile-discovery, or script-management authority. Security control is off by default. The setting also applies to Apple TVs paired by code on that controller.

Browser routes require same-origin validation and a bearer token, never credentials in a URL. They expose cached snapshots, typed commands, security actions, panel events and sensor history. Security writes require a security-enabled login or API key, explicit confirmation, and a panel code of the controller's configured length every time, including bypass/restore. Existing API and mobile discovery tokens are not browser tokens. Command success means panel acknowledgement, not confirmation of the physical result. The UI holds supported requested values while awaiting a matching snapshot, then reverts to reported state on rejection or after 15 seconds without confirmation. It never retries a command automatically or optimistically changes security/lock status.

Browser Route	Purpose
`GET /helper/web/controllers`	Public controller picker, the same names already shown at `/`. Each entry is `{"id", "name", "family", "capabilities", "fingerprint"}` (`fingerprint` is the controller's public identity, the one in its Bonjour record, or its id when it has no panel keys; the browser seeds the rooms' fallback-art assignment with it, 5.50), `family` being `"HAI / Leviton"`, `"NESS / ELK"`, `"Savant"` or `"Control4"`; the reply also carries `helperVersion` (the Helper's marketing version). ELK, Savant and Control4 controllers are listed (they have no panel key; sign in with an API key or Control Login). See **Capabilities** below
`POST /helper/web/session`	JSON `controller` plus either `username` and `password`, or `key`. The reply (and `GET /helper/web/session`) includes `controllerFamily`, `"HAI / Leviton"`, `"NESS / ELK"`, `"Savant"` or `"Control4"`; web control hides its Ethernet Quality view for NESS / ELK
`POST /helper/web/pair/start`	No session. JSON `{"controller", "deviceName"}` asks to pair a device (an Apple TV) by code. Replies `{"pairID", "code", "expires", "status": "pending"}`; see **Pair by code**
`GET /helper/web/pair/{pairID}`	No session. `{"status", "expires"}`, `status` being `pending`, `approved`, `denied` or `expired`; once approved, also the session fields `token`, `controller`, `securityEnabled` and `controllerFamily`
`POST /helper/web/logout`	Revoke the presenting browser token
`GET /helper/web/snapshot`	Cached controller, objects, audio metadata, and `favorites` with `favoritesRevision`
`GET /helper/web/favorites`	The controller's shared favorites: `{"favorites": [ids], "revision": n}`
`POST /helper/web/favorites`	Replace the list with JSON `{"favorites": [ids]}`; replies with the stored list

Browser Route	Purpose
`POST /helper/web/favorites/toggle`	JSON `{"id": "Unit:12"}` adds the id at the end, or removes it if present; replies with the new list
`GET /helper/web/doorbells`	The objects marked "Treat as doorbell": `{"items": [{"type": "Zone", "number": 7}], "revision": n}`
`POST /helper/web/doorbells`	Replace the list with JSON `{"items": [{"type", "number"}]}`; replies with the stored list. See **Doorbells**
`GET /helper/web/history`	The last three days at home, newest first: `?since={ISO 8601}&kinds=a,b&limit=200&protect=all`. See **History**
`POST /helper/web/usage`	A paired Apple TV's own taps, the same body as `POST /helper/usage` with the device taken from the pairing; a browser, tablet or password-signed-in TV session gets `403 deviceRestricted` (5.50). See **Usage and Suggestions**
`GET /helper/web/suggestions`	The "Right now" row for this session: `?home=0
`GET /helper/web/usage/summary`	Full sign-in only (a tablet or paired device gets `403 deviceRestricted`): the usage events kept per device for the Settings panel (5.50)
`POST /helper/web/usage/clear`	Full sign-in only: body `{}`; clears the controller's whole usage store, the Settings panel's **Clear usage history** (5.50)
`GET /helper/web/timeline`	The home timeline for a window: headline, moments, room lanes, flags and events. `?hours=8&end={ISO 8601}&after={cursor}`. See **Timeline**
`GET /helper/web/timeline/state`	The house at one moment, as snapshot objects: `?at={ISO 8601}`. See **Timeline**
`GET /helper/web/timeline/replay`	A window's state changes to replay on the device: `?hours=8&end={ISO 8601}&since={ISO 8601}`. See **Timeline**
`GET /helper/web/timeline/stills/{id}`	A Timeline picture (JPEG; a doorbell or a UniFi Protect camera detection), by an id a moment lists. See **Pictures**
`GET` and `POST /helper/web/timeline/settings`	Whether doorbell pictures are kept, and whether camera detections may keep pictures: `{"stills": true, "cameraStills": false}`; `POST` takes either or both. See **Timeline**
`GET /helper/web/changes`	Authenticated, bounded long-poll change notifications
`POST /helper/web/control`	Existing typed control JSON. For a Savant or Control4 controller: `{"action": "run_activity" or "stop_activity", "id", "room"?}`; see **Savant** and **Control4**
`POST /helper/web/security`	Security JSON plus explicit `code` and `confirmed: true`. `404` for a Savant or Control4 controller

Browser Route	Purpose
`GET /helper/web/events`	Cached panel activity and filters. `404` (GET and POST) for a Savant or Control4 controller
`POST /helper/web/events`	Bounded panel-history refresh
`GET /helper/web/history/{type}/{number}`	The authenticated controller's bounded history
`GET /helper/web/qr?text={text}`	PNG QR code of `text` (1 to 1,024 UTF-8 bytes) for the signed-in page; `400` otherwise

Except for the public picker, all browser API requests send `X-Myro-Origin` equal to the page origin, matched against the request Host. Any Origin header must also match. After sign-in, send `Authorization: Bearer {token}`. POST bodies are JSON and limited to 16 KiB at this application boundary. These browser tokens cannot be used as mobile discovery tokens or as API keys. No CORS access is enabled.

Snapshots include an opaque `revision`. Pass it as `since` to `/helper/web/changes?since={revision}` using the same authorization and origin headers. The request completes on a coalesced controller-cache change or after 25 seconds, returning `revision`, `changed` and `timeline` (the controller's timeline cursor as a decimal string; see **Timeline**). It does not block a server worker or poll the panel. Authorization is checked again before responding; expired or revoked credentials receive 401. At most one wait per session and 64 waits across the server are held. Clients should cancel when hidden or signed out, back off on errors, and keep periodic snapshots as a fallback. Use a timeout longer than 25 seconds. A revision is not an action acknowledgement or proof of physical completion.

Favorites. Stars in web control are kept by Helper per controller, so every browser and device signed in to it shows the same list. Ids are the object's `type` and `number` joined by a colon (`"Unit:12"`). A list holds at most 200 ids of 1 to 64 characters; anything else is rejected with `400`, and duplicates are dropped. A change wakes `/helper/web/changes`, so other open pages show it with their next snapshot. Pages from Helper versions that did not keep favorites stored them in the browser; the page copies such a list to Helper once, when Helper's list has never been set, then removes the browser copy.

Capabilities. Each `GET /helper/web/controllers` entry lists what this Helper's web API offers for it, so a client can show a feature only when the Helper it reached has it: `favorites` (shared favorites), `protect` (adding UniFi Protect cameras), `pair` (pair by code), `history` (`/helper/web/history`), `doorbell` (`/helper/web/doorbells`), `timeline` (`/helper/web/timeline` and its routes) on HAI / Leviton, NESS / ELK, Savant and Control4 controllers, `protectEvents` (UniFi Protect camera events: the `/helper/web/protect` routes, `keep` and the camera pictures switch; 5.50) on HAI / Leviton, NESS / ELK and Savant controllers (not Control4), `suggestions` (the usage and "Right now" routes under **Usage and Suggestions**; 5.50), and `savant` on Savant controllers only. Older Helpers send no `capabilities`; treat a missing list as none of these.

Pair by code. A device that cannot comfortably type a sign-in (an Apple TV) sends `POST /helper/web/pair/start` with `{"controller": "{controllerID}", "deviceName": "Living Room"}` (1 to 64 characters, no control characters; same `X-Myro-Origin`, JSON and 16 KiB rules as the other browser routes, no session). The reply is `{"pairID", "code", "expires", "status": "pending"}`: a six-digit `code` the device shows, and an `expires` five minutes away. The Mac shows **Apple TV “Living Room” wants access to {controller}.** with the same code and **Allow** / **Deny**; an unanswered request is denied when it expires. Each address may start five requests a minute, and at most three requests wait on the Mac at once (`429` otherwise); an unknown controller is `404`. The device polls `GET /helper/web/pair/{pairID}`: `{"status": "pending" | "approved" | "denied" | "expired", "expires"}`. Once approved, and only for

the origin that started the request, the reply also carries `token`, `controller`, `securityEnabled` and `controllerFamily`, with `expires` then being the token's, exactly like `POST /helper/web/session`; that reply can be collected for 60 seconds after approval, after which the request reads `expired`. An unknown or forgotten `pairID` reads `{"status": "expired"}`. The token is a kept (30-day sliding) session labelled with the device name, bound to the requesting origin like any browser token and sent as `Authorization: Bearer`. It stays valid while the controller is offered for web control, allows security actions only when the controller's Control Login has security enabled (`securityEnabled` reports this, and a controller with only API keys gives paired devices no security), and ends on **API > Control Login > Paired Devices > Revoke** (its next request gets `401`) or **Sign Out All Browsers**. Helper keeps at most 16 paired devices across its controllers, apart from the 32 kept browser sessions, so browser sign-ins never push out a paired device; pairing a 17th drops the least recently used. A paired token opens every other browser route (snapshot, control, favorites, doorbells, history, Timeline, cameras and UniFi Protect) like a Control Login session, but it is a browser token, not a mobile session: it does not open `/helper/timeline` or the other mobile routes. A paired device may arm and disarm through `POST /helper/web/security`, but the panel code is still required, the per-controller wrong-code lockout (5 wrong codes, 15 minutes) still applies, and zone bypass and restore, and a force arm (`"force": true`), answer `403` without reaching the panel. Turning security off in Control Login applies to paired devices on their next request. Helper logs each decision without the code.

Doorbells. Any zone, flag or output can be marked "Treat as doorbell" (the switch on its card in web control). Helper keeps the list per controller like favorites; `POST /helper/web/doorbells` replaces it with `{"items": [{"type": "Zone" | "Flag" | "Output", "number": 1–65535}]}` (at most 64; anything else is `400`; duplicates are dropped) and replies `{"success": true, "items", "revision"}`. Snapshot objects that are marked carry `"doorbell": true`. A press is a zone becoming not ready or tripped, a flag going from 0 to any value, or an output turning on, as reported by the panel's change notifications (status replies after a connect or refresh never count). Each press adds a `doorbell` event to the history and wakes `/helper/web/changes`.

History. `GET /helper/web/history` returns `{"events": [{"id", "time", "kind", "title", "room"?, "objectType"?, "objectNumber"?, "cameraID"?}], "retainedDays": 3}`, newest first. `kind` is `doorbell`, `lights` (a unit switching on or off; title "{name} on" or "{name} off"), `security` (arming, disarming and alarms from the panel log), `door` or `motion` (a trip on a door or motion zone in the panel log). Panel-log entries use the summary the Events page shows and appear once the panel log has been read (`POST /helper/web/events`). `time` is ISO 8601; panel-log times come from the panel's clock, with the year inferred. `since` returns events at or after that time (dedupe by `id`), `kinds` is a comma-separated filter, `limit` is 1 to 2,000 (default 200); a bad value is `400`. `objectType` and `objectNumber` are the snapshot's own `type` and `number`. `room` is the object's room name, or the one web room holding it. `cameraID` is a snapshot `cameras[].id`, present only when the web room holding the object shows exactly one camera. Helper keeps three days and at most 6,000 events per controller (a reply holds at most 2,000); new events wake `/helper/web/changes`.

Without `kinds`, `/history` returns only those five kinds. The timeline adds `window`, `climate`, `lock`, `audio`, `media`, `scene`, `power` and `system`, and since 5.50 `note` and `moment`, which only scripts (`MyroTimeline`) and the keyed notes route write (see **Timeline and Notes**); ask for them by name in `kinds`. Lights events come from Helper's state record: on and off edges only, at most 30 an hour per unit. Since 5.50 a Unit or Output edge the panel did not announce, which Helper first saw in a status poll, is an event too, with the poll's time and `approximate: true`, once the connection has been up for two minutes and the previous state was known (never the first sighting or a reconnect burst); zones, areas, thermostats, locks and audio stay live-only. Doors, motion and arming now also come live from the panel's change notifications; a panel-log record that describes a live event (the same area and direction, or the same zone, within 3 minutes) adds "by {user}" to it instead of a second event. Events may carry these optional fields, each only when present: `seq`

(the event's timeline cursor), `end` (a door closed, an activity stopped), `count` (motion trips folded in), `detail` ("by User 3", "from 68°"), `objectID`, `area`, `source` (`hai`, `elk`, `savant`, `control4`, `panelLog`, `helper`, `protect`, or since 5.50 `script` for a note a script wrote and `api` for one written through `POST /helper/{controllerID}/timeline/notes`), `importance` (0 routine to 3 urgent), `symbol` (an SF Symbol name, when Helper has a specific one), `detections` (UniFi Protect only), `stillID` (a kept picture), `rev` (bumped when the event changes after it was stored), `approximate` (5.50; `true` only, on a light or output edge learned from a status poll: the time is the poll's, not the change's, and the browser prefixes it with "~"), and on a `moment` anchor `tone` (`info`, `good`, `warn` or `alert`) and `objects` (the ids it covers). UniFi Protect camera events (`source: "protect"`, see **UniFi Protect camera events** below) are `doorbell` and `motion` events, so they are among the five kinds; `/history` leaves out those with an `importance` below 2 (vehicles, animals, packages and plain motion) unless the request adds `protect=all`, so a short list shows people and rings. The timeline routes always carry every event.

Timeline. For controllers whose entry lists `timeline`, Helper records the state of every object it knows (in the same shape as `/helper/web/snapshot` objects) for the last 24 hours, and derives a readable timeline from it. Times are ISO 8601 UTC with whole seconds; lists run oldest first. All three routes take the session, `X-Myro-Origin` and Bearer token like the others, reply `Cache-Control: no-store`, answer `401` without a session, `404 {"error", "code": "notAvailable"}` for a controller without a timeline (every 5.50 family has one), and `400 {"error", "code": "badParameter"}` for a bad parameter.

`GET /helper/web/timeline?hours=8&end={ISO 8601}&after={cursor}`: `hours` 1 to 24 (default 8), `end` defaults to now and is clamped to it, and a window that would start before the retained range starts at `available.start`. `after` limits `events` only to events newer than that cursor or changed since it; everything else is always complete. The reply:

```
{"version": 1, "controllerID", "cursor": "1761400000123",
 "window": {"start", "end", "hours", "bucketSeconds", "bucketCount": 96, "timeZone"},
 "available": {"start", "retainedHours": 24},
 "gaps": [{"start", "end" | null, "reason": "offline" | "helperStopped"}],
 "headline": {"text": "Evening at home. Home at 6:31 PM. Doorbell rang at 7:42 PM.",
              "highlights": [{"text", "momentID" | null, "eventIDs": []}]},
 "counts": {"events", "moments", "flags", "byKind": {"lights": 7}},
 "current": {"security": [{"area", "name", "mode", "modeName"}], "lightsOn", "openDoors"},
 "moments": [{"id": "m-4c1e9a0b2d77", "kind", "title", "summary", "start", "end", "ongoing",
"importance",
              "tone": "info" | "ok" | "warn" | "bad", "symbol", "icon", "rooms": [{"id" | null,
"name"}],
              "eventIDs": [], "objectIDs"?: ["Zone:12"], "stills": [{"id": "s-8f2c1d9e0a3b4c5d",
"cameraID", "time"}],
              "flagIDs": []}],
 "lanes": [{"id": "room:{roomID}" | "home", "roomID" | null, "name", "levels": [96 x 0-4], "light":
[96 x 0-100],
            "activity": [96 counts], "lightMinutes", "events"}],
 "flags": [{"id", "kind", "severity": "notice" | "warning", "title", "since", "active", "objectType"
| null,
            "objectNumber" | null, "objectID" | null, "roomID" | null, "momentID" | null}],
 "events": [history events, plus "roomID" when one web room holds the object],
 "eventsTruncated": false,
 "settings": {"stills": true, "cameraStills": false}}
```

`cursor` is a decimal string (it grows with time, in milliseconds); pass it back as `after`, and compare it with the `timeline` field of `/helper/web/changes` to know when to fetch. Moment `kind` is `arrival` ("Arrived home"), `departure` ("Left home"), `goodnight`, `morning`, `doorbell`, `alarm`, `media`, `music`, `scene`, `security`, `activity` ("Kitchen busy") or, since 5.50, `moment` (a named moment a script or the keyed notes route wrote, with exactly its title, `symbol` and `tone`, gathering the events ten minutes either side of its anchor or until its `end`; see **Timeline and Notes**); a moment keeps its `id` while it grows. `symbol` is an SF Symbol name, `icon` a Lucide name. Lanes follow the web rooms' order, then rooms known only by name, with "Whole home" (areas, security, system) last; `light` is the average level of the room's lights per bucket, `activity` counts doors, motion, scenes and media, and `levels` combines them. Flag `kind` is `lightLeftOn` (6 hours, or 30 minutes while every area is armed Away or Vacation), `doorLeftOpen` (15 minutes), `openedWhileArmed`, `tempOffSetpoint` (4° for 45 minutes), `accessDenied`, `chatter` or `offline` (5 minutes or more). A moment's optional `objectIDs` lists the distinct object keys of its events (`objectID`, or `type:number`), present only when every event has one, so a client that hides objects can drop a moment whose keys are all hidden. `settings.cameraStills` (5.50) reports the camera pictures switch next to `settings.stills`; see **Pictures**. `events` holds at most 2,000 (the newest; `eventsTruncated` is then true). The headline is English.

Timeline for PRO (5.50). Paired apps read the same timeline with the controller-bound `Authorization: Bearer SESSION_TOKEN` they use for `/helper/rooms` and `/helper/cameras` (see the mobile routes under **Shared Rooms and Cameras** below), at `GET /helper/timeline`, `/helper/timeline/state`, `/helper/timeline/replay` and `/helper/timeline/stills/{id}`. They run the same code as the browser routes with the controller taken from the token (there is no controller parameter; a `controller` query is ignored), so the parameters, limits, JSON, error codes (`400 badParameter`, `400 outOfRange` on `/state`,

`404 notAvailable` for a controller without a timeline, `404 notFound` for a picture) and the `Cache-Control` headers (`no-store`, and `private, no-store` on a picture) are identical to **Timeline** and **Pictures** below. Differences: a missing or unknown token is `401 {"error", "code": "unauthorized"}`; the routes are read only, so there is no app route for `/helper/web/timeline/settings` (both picture switches stay in the browser; `settings.stills` and `settings.cameraStills` in the timeline reply report them); `since` on `/helper/timeline/replay` works as on the browser route. A browser or paired Apple TV token is not accepted here. The `X-Myro-Sync-Review` header is not needed. There is no `capabilities` list for apps: a client detects support with `GET /helper/timeline?hours=1`, where `200` means the timeline is there and a `404` without a `code` means this Helper predates it, and `404` with `code: "notAvailable"` means the controller has no timeline (no 5.50 controller family answers it). There is no push either: every timeline reply carries `cursor`, which grows whenever anything the timeline records changes, so an app fetches on appearing and then re-fetches while the screen is visible (replies are shared for 15 seconds, so polling every 15 to 30 seconds is cheap), passing the last `cursor` as `after` to receive only new or changed `events` (everything else is always complete) and skipping a rebuild of its own when the cursor is unchanged. `GET /helper/web/changes` stays browser only.

Pictures. Helper keeps two kinds of Timeline picture, each with its own switch and budget, so camera detections can never evict, delay or rate-limit a doorbell picture:

	Doorbell pictures	Camera pictures (5.50)
Taken for	A ring of a zone, flag or output marked as a doorbell (a JPEG from the one camera in its room, only when exactly one camera is there), and a UniFi Protect doorbell ring	A UniFi Protect detection other than a ring, on a camera whose **Keep pictures** choice is on
Switch	`settings.stills`, on until turned off	`settings.cameraStills`, off until turned on, **and** the camera's `pictures` (see **UniFi Protect**), off until turned on
Kept	24 hours	8 hours
Limits	200 KB each, 200 and 50 MB in all, 30 an hour, one a minute per camera	200 KB each, 300 and 40 MB in all, 30 an hour, 6 an hour and one a minute per camera

Nothing else takes a picture: not alarms, not doors, not panel motion. For a UniFi Protect event Helper keeps Protect's own picture of that event (asked for from 2 seconds after it starts or ends, for about 10 seconds), or one live snapshot from Protect when there is none, and never opens the camera's stream for it. The one exception is a panel doorbell's ring merged with a Protect ring: when Protect's picture does not come (no thumbnail and no snapshot, a refused or expired Protect session, or no connection), Helper takes that ring's picture from the room camera's stream, as it would without Protect. The oldest picture of a kind goes first when its limits are reached. A moment's `stills` lists the pictures kept for its events (`id`, `cameraID`, `time`), whichever kind they are, and the event carries the same id as `stillID`; both are left out once the picture is gone (the events are kept three days). Pictures sit in Helper's Application Support folder, readable by the Mac user only. `GET /helper/web/timeline/settings` returns `{"stills": true, "cameraStills": false}`; `POST` takes `{"stills": true | false}`, `{"cameraStills": true | false}` or both, and returns the same shape. Anything else is `400 {"error", "code": "badParameter"}`. Turning a switch off deletes only that kind's pictures kept so far. A camera whose `pictures` choice goes off, that is hidden or that leaves the camera list loses its camera pictures at once, and a picture still being fetched from it is not kept. Any

signed-in session can read and change both, and a change wakes `/helper/web/changes`. `GET /helper/web/timeline/stills/{id}` returns the JPEG (`Content-Type: image/jpeg`, `Cache-Control: private, no-store`), `404 {"code": "notFound"}` when the picture is pruned, unknown or another controller's, and `401` without a session. Fetch it with the session headers into a blob URL, as camera snapshots are; an `<img src>` cannot send `X-Myro-Origin`. Savant homes have no panel doorbell, so they keep doorbell pictures only from a linked UniFi Protect doorbell. Removing a controller deletes its pictures and both settings.

UniFi Protect camera events (5.50). When the owner keeps a UniFi Protect sign-in (see **UniFi Protect** under **Shared Rooms and Cameras**), Helper listens to the console and records the detections the owner picked for each camera in Helper's camera list. No new event kinds are used: a Protect ring is a `doorbell` event and every other detection a `motion` event, with `source: "protect"`. A detection event looks like this:

```
{"id": "protect-6531a8f2009c4e03e4000412", "time": "2026-10-27T18:42:10Z", "kind": "motion",
 "title": "Person at Front Door", "room": "Entry", "roomID": "room-3",
 "objectType": "Camera", "objectID": "Camera:4f6c2b8e-1d3a-4c5b-9e7f-0a1b2c3d4e5f",
 "cameraID": "4f6c2b8e-1d3a-4c5b-9e7f-0a1b2c3d4e5f", "seq": 1793212930123,
 "end": "2026-10-27T18:46:31Z", "count": 3, "detail": "Person and package", "source": "protect",
 "importance": 2, "symbol": "figure.walk", "detections": ["person", "package"],
 "stillID": "s-8f2c1d9e0a3b4c5d", "rev": 3}
```

- `cameraID` is the Helper camera id (the snapshot's `cameras[].id`, lowercase), and `objectID` is `"Camera:" + cameraID` on every Protect event, so one hide covers a camera's events and its notes. `objectType` is `"Camera"` and there is no `objectNumber`. `room` and `roomID` come from the web room holding the camera; a camera in no room falls in the "Whole home" lane.
- `detections` lists what Protect saw, highest first: `person`, `package`, `vehicle`, `animal`, `motion` (a ring is `ring`). The highest sets `title`, `symbol` and `importance`, and `detail` names them all when there is more than one:

Detection	title	symbol	importance
`ring`	"{camera name} rang" (`kind: "doorbell"`)	`bell.fill`	3
`person`	"Person at {place}"	`figure.walk`	2
`package`	"Package at {place}"	`shippingbox.fill`	1
`vehicle`	"Vehicle in {place}"	`car.fill`	1
`animal`	"Animal at {place}"	`pawprint.fill`	1
`motion`	"Motion at {place}"	`figure.walk.motion`	0

`{place}` is the camera's name without a trailing "camera", "cam" or "doorbell". `symbol` is new on events in 5.50 and appears only where Helper has a specific one; a client without it falls back by `kind`.

- Detections close together on one camera fold into one event (`count`, `end`; `end` is absent while Protect still reports it open): at most 6 new detection events an hour per camera and 30 per controller, after which new detections fold into the camera's newest event. A detection with a higher `importance` than the event

it would join opens a new one. Rings are never folded with detections; a second ring within 30 seconds folds into the first. `rev` grows with every change, so fetch with `after` to see updates.

- A ring that a panel doorbell (a zone, flag or output marked as a doorbell) also reports within 30 seconds, in the same room or linked by the camera's `twin` choice, is **one** event: the panel's (its own `id`, `title`, `objectType`, `objectNumber` and implied `objectID` such as `"Zone:12"`, and no `source`), with `cameraID`, `importance: 3`, `symbol: "bell.fill"`, `detections: ["ring"]` and Protect's `stillID` added.
- Moments: a Protect ring anchors a `doorbell` moment like any doorbell, and that moment also takes the camera's detections from 2 minutes before the ring. Detections left over make an `activity` moment per camera only when a person was seen ("Person at Front Door", `summary` "Seen 4 times."), growing for up to 30 minutes with the same `id`, with `importance` at most 1. Vehicles, animals, packages and plain motion never make a moment of their own. Protect events never start, end or cancel *Arrived home*, *Left home*, *Good night* or *Good morning*, never join a busy-room chain and do not count in the headline's quiet or busy wording. Their pictures appear in `stills` exactly as doorbell pictures do.
- Flags: a camera with more than 30 detections in an hour gets one `chatter` note a day (`"Driveway busy: 34 detections in an hour"`, `severity: "notice"`, `active: false`, `objectType: "Camera"`, `objectID: "Camera:{cameraID}"`). A client that shows a single note should not pick this one when no flag is active; it is for the notes list. While a kept console is stopped for a reason the owner must fix, the timeline carries an active `offline` flag `"UniFi Protect camera events paused: sign in again in Helper"`, or `"Can't reach UniFi Protect since 18:02"` after an hour without an answer, with `objectType: "Protect"`, `objectID: "Protect:{consoleID}"` and no `objectNumber`.
- Helper keeps at most 1,500 Protect events per controller, apart from the 6,000 others, so busy cameras never push panel history out.

`GET /helper/web/timeline/state?at={ISO 8601}` returns the house at that moment: `{"at", "cursor", "inGap": {"start", "end", "reason"} | null, "objects": [snapshot objects], "unknown": ["Zone:40"]}`. Objects use `/helper/web/snapshot`'s `type`, `number` and field names, with today's names and rooms, and without the live-only fields (`updatedAt`, entry and exit timers, `lastAccessAt`). `unknown` lists objects that exist today but had no recorded state then. An `at` before the retained range is `400 {"code": "outOfRange", "available": {"start", "end"}}`; one in the future is clamped to now.

`GET /helper/web/timeline/replay?hours=8&end={ISO 8601}&since={ISO 8601}` returns `{"window": {"start", "end", "hours", "timeZone"}, "cursor", "base": [snapshot objects at window.start], "changes": [{"seq", "time", "offset", "id", "correction"?, "object" | "gone": true}], "gaps", "truncated"}`. Each change carries the whole object after it, so a client replaces by id (`type:number`); `offset` is seconds from `window.start`. `correction: true` marks a value that came from a status refresh rather than a live change (true state, never an event). At most 10,000 changes: with more, the newest are kept, `truncated` is true and `window.start` moves to the first one. The optional `since` sends only the changes at or after that time, with an empty `base` and `since` echoed in the reply; a client already holding the earlier ones drops its own changes from `since` on and appends these.

Savant. A Helper that runs only a Savant system lists it in `GET /helper/web/controllers` with `family: "Savant"`, and it signs in the same way (Control Login or API key for that controller). Its snapshot's `objects` are the Savant activities: `{"type": "Activity", "id": "{service id}", "number": 0, "name", "displayName", "room", "roomName", "kind", "active": true | false | null}`. `room` is the id of the web room (as in `rooms`) whose saved Savant source is this activity, or the Savant room's own id when no single web room holds it; `roomName` is the Savant room's name; `active` is null until Savant reports the room's state. `POST /helper/web/control` with `{"action": "run_activity", "id"}` starts an activity and `{"action": "stop_activity", "id"}` turns it off; add `"room"` (either room id from the snapshot) when the

same service id is in more than one room. The reply is `{"success": true, "action", "status": "sent"}`, meaning sent, not confirmed (watch the snapshot's `active`); an unknown activity, or one in several rooms sent without `room`, is `400`; a disconnected host, unknown state or busy activity is `503` with the reason. Security and panel events return `404` "not available" for Savant. The timeline records each activity going on and off ("Living Room: Apple TV on", a `media` event whose `end` is set when it stops after a minute or more) and keeps the activities' state for the replay and state routes, with the ids `Activity:{room}:{id}`, so the same service in two rooms stays two objects. Unknown (`active: null`, for example while Savant is disconnected) never creates an event. A Savant timeline has no doors, lights, security or flags; its moments are media and, in a busy room, activity.

Control4. A Control4 Link controller (preview) is listed in `GET /helper/web/controllers` with `family: "Control4"` and the capabilities above without `savant` or `protectEvents`, and it signs in the same way (Control Login or API key for that controller); an Apple TV can pair with it. Its snapshot's `objects` are the Watch and Listen activities in the same shape as Savant's, plus the room's reported state: `{"type": "Activity", "id": "{activity id}", "number": 0, "name", "displayName", "room", "roomName", "kind": "Watch" | "Listen", "active": true | false | null, "power", "volume", "muted"}`. `id` is the connector's activity id (for example `watch:345`); `room` is always the Control4 room id and `roomName` its name; `active`, `power`, `volume` and `muted` are null while the room is not reporting or Control4 Link is disconnected. `POST /helper/web/control` takes the same `{"action": "run_activity" | "stop_activity", "id", "room"?}` as Savant (`room` is the Control4 room id, needed when the same activity id is in more than one room); `stop_activity` turns the activity off only while it is the one playing in that room. The reply is `{"success": true, "action", "status": "sent"}`, meaning sent, not confirmed; any other action, a missing id, an unknown activity, or one in several rooms sent without `room` is `400`; no imported connection, a disconnected connector, unknown room state or a busy activity is `503` with the reason. Security and panel events return `404` for Control4. The timeline uses Savant's events, moments and lanes unchanged, with `source: "control4"` and the ids `Activity:{room}:{id}` (for example `Activity:12:watch:345`, which has a second colon: build the id, never split it): an activity going on is a `media` event ("Living Room: Watch Apple TV on", `end` set when it stops after a minute or more), a quick stop is an "off" event, and switching a room's source is one activity stopping and another starting. Volume and mute changes never create events; they are kept on the activity objects for the replay and state routes. Unknown state never creates an event, and a disconnection shows as an offline gap. A Control4 timeline has no doors, lights, security, flags or UniFi Protect events. Linked Myro:Home Pro apps cannot read it at `/helper/timeline`: that mobile session proves a panel key, which a Control4 controller does not have.

Open on another device. The Mac's own browser opens web control at `localhost`, which on a phone means the phone. When the sign-in reply or `GET /helper/web/session` answers a request that comes from the Mac itself (loopback or one of the Mac's own addresses), it also carries `otherDevices`: `{"link": "http://{LAN address}:9393/helper/control?controller={controllerID}", "more": [{"label", "kind", "link"}], "hostName": "{Mac}.local", "hostLink": "http://{Mac}.local:9393/helper/control?controller={controllerID}"}`. `hostName` is the Mac's Bonjour name (the one Helper advertises) and `hostLink` is the same page by name, which keeps working if the LAN address changes; the page shows it as a second row with its own Copy button. `link` uses the Mac's first private home-network address (10/8, 172.16/12, 192.168/16, Ethernet or Wi-Fi first) and is omitted when there is none; `more` lists any other addresses, with `kind` `lan`, `other` or `tailscale` (100.64/10, for use away from home). Requests from other devices never get `otherDevices`. The page shows the link, a Copy button and a QR code from `GET /helper/web/qr`, which needs the same session and `X-Myro-Origin` header as the other browser routes. The links carry no key or token. **View > Copy Web Control Link for Other Devices** in Helper copies the same `link`.

Tablet layout and the wall tablet address (5.50). The web control page and its files need no credential and are served from the Helper bundle with the same headers as `/helper/control` (`Cache-Control: no-cache`, `Referrer-Policy: no-referrer`, `X-Content-Type-Options: nosniff` and the page's `Content-Security-Policy`, which allows only same-origin scripts and styles):

Page or file	Purpose
`GET /helper/control`	The web control page. `?controller={controllerID}` preselects a controller, `?layout=tablet` or `?layout=standard` forces a layout, and `?kiosk=1` adds the wall-tablet behaviour
`GET /helper/tablet`	The same page at a second address, always in the tablet layout and with the wall-tablet behaviour; takes `?controller=` like `/helper/control`
`GET /helper/control/tablet.css`, `/helper/control/tablet.js`	The tablet layout: a trailing rail and two panes, scoped to the tablet layout so the standard layout is unchanged
`GET /helper/control/room-palettes.js`	The shared room fallback-art library (the same 18 palettes and assignment rule as Myro:Home Pro) that the Rooms view draws for a room without a photo, using the room's stored `palette` or choosing one the way Pro does (5.50)
`GET /helper/control/kiosk.js`	The wall-tablet behaviour: ambient dimming, overnight and watchdog reloads, the forced Keep sign-in and the low-power display
`GET /helper/control/manifest.webmanifest`	The web app manifest (`application/manifest+json`) for Add to Home Screen: scope `/helper/`, standalone display, the touch icon above

The page decides its layout in the browser before first paint: `?layout=` first, then the browser's own Settings choice (auto, tablet or standard, kept in that browser's local storage, never a credential), then, for auto, a landscape screen whose main pointer is touch and whose window is at least 1000 × 600. Neither the layout nor the wall-tablet behaviour changes what the server allows. There is no separate tablet credential and no new authenticated route: a page at `/helper/tablet` signs in through `POST /helper/web/session` with a Control Login or an API key like any browser and has exactly that session's rights, `GET /helper/auth/whoami` is unchanged and remains a mobile-session route, and a tablet session ends the ways any kept browser session does (sign-out, **Sign Out All Browsers**, a changed or removed Control Login, or a revoked API key). On the wall tablet address the page turns on Keep with `POST /helper/web/session/keep` `{"remember": true}` while signed in, and before a scheduled or watchdog reload it checks that Helper answers with `GET /helper/web/controllers` (sent without cookies); it hides sign-out and the home-photo and Timeline-picture settings on screen, but the session can still use every browser route its sign-in allows. Kept sessions from wall tablets count toward the 32 kept browser sessions, unlike paired devices.

Control embeds Graphs and Ethernet Quality, and shows them before sign-in as those public read-only pages do; it asks for sign-in the first time another view is chosen. No bearer token is passed to the embedded pages. Navigating among views keeps the session, and a reload resumes it from the session cookie. The standalone graph URLs and their existing access rules are unchanged.

The bundled server uses HTTP. Passwords, keys, tokens and panel codes are not encrypted in transit. Use only a trusted local network or trusted encrypted VPN; never forward port 9393 to the internet. Browser sessions do not change the existing read-only graph/discovery access policy. The saved System/Light/Dark

preference contains no credentials.

`GET /helper/ethernet?controller={controllerID}` opens the dedicated Ethernet Quality page. Sensor Graphs remain at `/helper/graphs`; the Ethernet JSON endpoint remains `/helper/graphs/ethernet`. Bonjour HTTP links now open Web Control, while mobile discovery's `graphsPath` remains a direct graphs link.

Tablet sign-in (5.50). A wall tablet can hold a restricted sign-in of its own instead of a full browser session. It is enrolled from the tablet itself, from a full browser sign-in, and the Mac's **Control Login > Paired Devices** list shows and revokes it next to the paired Apple TVs. All three routes need `X-Myro-Origin`, take and return JSON, and answer with `Cache-Control: no-store`. "Full sign-in" means a browser session that is not a tablet; a tablet gets `403 {"code": "tabletRestricted"}` on those routes.

Route	Credential	Purpose
`POST /helper/web/tablet/enroll`	Full sign-in	Body `{"name": "Kitchen iPad"}` (1 to 64 characters). Turns this browser's sign-in into a tablet session: the sign-in ends, `myro_session` is cleared and the `myro_tablet` cookie is set in the same reply. Returns `success`, `token`, `expires`, `controller`, `tablet: true`, `securityEnabled`, `controllerFamily` and the new `device` row. `400 badParameter` for a bad name; `401` without a session.
`GET /helper/web/tablet/devices`	Any browser session, tablet included	`{"devices": [...]}`, most recently used first: `id` (64 hex, a hash that cannot sign in), `name`, `kind` (`"tablet"` or `"tv"`), `created` (tablets only), `lastUsed`, `current` (true only for the tablet making the request).
`POST /helper/web/tablet/revoke`	Full sign-in	Body `{"id": "..."}`. Ends that tablet or paired Apple TV at once, in memory and on disk, and returns the remaining list. `400 badParameter`, `404 notFound`.

What a tablet session can do: every `GET` a browser session can (snapshot, changes, favorites, doorbells, history, Timeline, stills, suggestions, graphs, cameras, photos, background); `POST /helper/web/control` and Savant activities; `POST /helper/web/security` `set_mode` with the panel code, typed each time. Everything else that writes is refused by the Helper with `403 deviceRestricted`: `force`, bypass and restore, favorites and doorbell marks, Timeline settings, photo upload, Protect sign-in, pairing, usage upload and clearing, and any Helper-wide setting. `GET /helper/web/session` answers `tablet: true` and the `device` row for a tablet; `POST /helper/web/session/keep` changes nothing (a tablet is always kept) and `POST /helper/web/logout` ends it. Tablets do not count against the kept-session cap; at most 16 tablets are kept per controller. Wrong panel codes lock the tablet's own session as well as the controller.

Ethernet Quality

`GET /helper/graphs/ethernet?controller={controllerID}`

For a NESS / ELK controller this returns `404` with `{"error":"Ethernet Quality is not available for NESS / ELK controllers"}`, because its signals (Omni-Link CRC errors, link drops, connection failures) do not exist on that protocol. The `ethernetQuality` object is likewise omitted from an ELK controller's status snapshot; `connected` stays.

Optional `since={ISO-8601}` limits returned samples.


```
{
  "hasData": true,
  "controllerGUID": "953d8997-ab41-4aec-9f58-78235e36d46f",
  "assessment": {
    "level": "low",
    "reason": "1 isolated connection episode in the last 7 days.",
    "episodes24Hours": 1,
    "episodes7Days": 1,
    "disruptiveIncidents24Hours": 1,
    "disruptiveIncidents7Days": 1,
    "decayedLoad": 0.99,
    "velocity": null,
    "sharedNetworkEpisodes": 0,
    "networkDownEpisodes": 0,
    "incidents24Hours": 1,
    "incidents7Days": 1
  },
  "totals": {
    "crcErrors": 0,
    "linkDrops": 1,
    "halfOpen": 0,
    "connectionFailures": 1,
    "ackTimeouts": 0,
    "incidentRecords": 2,
    "incidentEpisodes": 1,
    "samples": 78
  },
  "data": []
}
```


The assessment describes recent controller-link recurrence, not the health of the Ethernet chip:

- `low` - isolated or no recent link incidents.
- `medium` - a recurring pattern that merits observation.
- `high` - a dense or sustained pattern that merits investigation.

Since 4.10 the level is computed from **episodes**, not rows. Rows written within 120 seconds of each other -- an unexpected disconnect and the failed reconnect that follows it -- are one episode, so a single glitch cannot read as a recurring pattern. `incidentRecords` still counts rows; `incidentEpisodes` counts episodes, and `incidents24Hours` / `incidents7Days` now count episodes as their names always implied.

Episodes fade: each one contributes `0.5 ^ (age / 48 h)` to `decayedLoad`, halved again if it was ACK timeouts only, so trouble from weeks ago no longer colours the level. `velocity` is the last 24 hours' episode count divided by the prior six days' daily mean (`null` with no prior week). One or two episodes, whenever they happened, stay `low`. `medium` needs recurrence -- a sustained load of about three recent episodes, or a week whose frequency is rising -- and `high` needs three disruptive episodes in a day or a heavy sustained load. `reason` is one sentence explaining which of these applied.

Episodes during a known whole-Mac outage (`networkDownEpisodes`), or while independently configured local bridges were also down (`sharedNetworkEpisodes`), never count against the controller link. `disruptive...` excludes ACK-only episodes and counts CRC errors, unexpected drops, half-open sessions, and connection failures.

The `totals` object also reports corroborated, inconclusive, and network-down incident context, disconnected baselines, and interface observations when available. None is a hardware diagnosis by itself.

Connection Status

`GET /helper/graphs/status?controller={controllerID}`

A small, uncached status read the graph page polls every ten seconds so it can say whether the data on screen is live, and show any messages the controller is currently displaying. It shares the graph routes' access rules and reads only cached state, so it is cheap to call.


```
{
  "controllerID": "953d8997-ab41-4aec-9f58-78235e36d46f",
  "name": "Grouse",
  "connected": true,
  "connecting": false,
  "secondsSinceLastMessage": 4,
  "messages": [],
  "canClearMessages": false
}
```


`connecting` is true while the Helper is re-establishing a dropped connection. `secondsSinceLastMessage` is `null` until the first message from the panel arrives after launch. `messages` describes active stored messages where the controller supports them. See the controller guide; an empty list is not evidence that every panel supports stored messages.

`canClearMessages` describes the keyed page's control access. Clearing stored messages also requires controller support; use that controller guide's message contract. An unkeyed graph page offers no message control.

Mobile Discovery and Session Authentication

These unkeyed routes support Myro:Home mobile apps discovering a Helper that manages the same panel. They are documented for interoperability but are separate from the API-key automation surface.

Bonjour

The Helper publishes one `_myrohome-helper._tcp` record per controller and a compatible `_http._tcp` record. TXT data can include `helperID`, `version`, `apiPort`, fingerprint scheme, controller fingerprint, controller display name, and hostname.

The fingerprint is a one-way identifier derived from panel key material. Panel keys are never published.

Identity

`GET /helper/identity`

Returns the Helper ID, version, build, API port, fingerprint scheme, and managed controller fingerprints. Controller names are intentionally omitted from this unkeyed response.

Challenge

`POST /helper/auth/challenge`

```
{"fingerprint":"CONTROLLER_FINGERPRINT"}
```

Returns a single-use challenge ID, nonce, proof scheme, and expiration interval.

Respond

`POST /helper/auth/respond`

```
{
  "challengeID": "CHALLENGE_ID",
  "proof": "PROOF",
  "client": "Myro:Home Pro"
}
```

The proof is computed from the panel keys and nonce according to the advertised scheme. A successful response returns a bearer token scoped to one controller. Challenges are single use whether verification succeeds or fails.

Session Inspection and Links

Send the bearer token in the `Authorization` header:

```
Authorization: Bearer SESSION_TOKEN
```

- `GET /helper/auth/whoami` returns the token's controller, fingerprint, and Helper ID.
- `GET /helper/links` returns a controller-scoped `graphsPath` after successful authentication, along with `addresses` and `apiPort` (added in 4.08). `addresses` lists every non-loopback IPv4 address this Mac answers on — the LAN address, plus any VPN overlay address such as a Tailscale `100.64/10` one. Bonjour only ever advertises the LAN address, so a client that stores this list while it is on the local network can still reach the Helper from elsewhere, instead of reporting it missing when only multicast is missing. The list is returned here rather than on the unauthenticated `/helper/identity` because it describes where the machine can be reached; a caller must first prove it holds the panel key. It also returns `controllerAddress` (string) and `controllerPort` (integer): the private LAN address and port Helper is actually connected to (the live connection's peer, or a literal private IP configured while disconnected). It is omitted for outside addresses (public IPs, DDNS names, Tailscale `100.64/10`), loopback, and ELK controllers, so a phone never learns an address that cannot help it at home. Apps always dial their configured (outside) address first and fall back to this LAN address only when that dial cannot connect, so they still connect at home behind double NAT with no added latency elsewhere. Both keys are omitted when there is no such private LAN address; treat them as optional.
- `POST /helper/auth/revoke` invalidates the current session.

Shared Rooms and Cameras

Browser Rooms/Cameras and controller-scoped synchronization with linked PRO/Classic apps are available for supported HAI/ELK homes. These are shared Helper catalogs, separate from panel programming commands. Browser room activity control is limited to saved Savant assignments resolved against a configured Helper installation.

PRO and Classic use **Edit > Sync with Helper** in Rooms or Cameras to review shared changes. Finding or reconnecting to Helper does not import, upload or remove anything, including controller photos; there is no periodic room/camera reconciliation. Tapping **Edit** makes one read-only check of `GET /helper/syncStatus`; if Helper's camera, room or photo revision differs from what this device last synced or ignored, the **Sync with Helper** button shows a badge. There is no banner, no check on entering the tab and nothing periodic. The review sheet has a direction control: **Both ways** (three-way merge; each conflicting room, camera or photo needs a choice of **Use Helper** or **Use this device**, default **Skip**, which keeps both versions), **From Helper** (only this device changes; Helper wins conflicts) and **To Helper** (only Helper changes; this device wins conflicts). The review lists additions, updates, removals, assigned controls and photo changes before **Apply**. Conflicts never create automatic copies. If the local or shared data changes after review, review again; a race during Apply stops instead of retrying with new revisions. Some already-reviewed stages can have completed before a later stage fails, so review the remaining changes again.

Ignore means "until Helper changes again": the device remembers the ignored generations and the badge returns only after a newer one. **Turn off** and **Turn on** are in the sheet. Opening review is read-only. Controls appear only with an authenticated Helper for the selected controller, with no subscription requirement. Existing local data remains usable while Helper is unavailable. Live controller feedback is independent of layout sync.

Identity is derived, never matched by name. Panel rooms are `room-<panel number>` (number below 1000) on every device and in Helper. App-created rooms are `room-<roomGUID lowercase>` with numbers from 1000 up. Panel cameras are `setup-<number>`; app cameras use their lowercase `cameraGUID`. Helper rejects any

other room ID form with 400. Panel rooms are never deleted: a delete mutation for `room-<int>` sets `webHidden: true` and keeps the layout. App rooms get tombstones, so another device cannot silently resurrect them. Helper refuses a camera create whose normalized connection (scheme, host, port, path, username; password excluded) already exists, returning 409 with that mutation ID in `conflicts`; the client adopts the existing record. Favorites, local visibility and display order stay on each device. Classic does not create new empty cards for rooms containing only HomeKit/Savant controls; those assignments remain in Helper. Layout revision (`revision`) and photo revision are independent: assigning or replacing a photo changes only `photoRevision`. A room without a photo may carry a `palette`: an id from the shared fallback-art library that Myro:Home Pro 3.10 assigns (lowercase letters, digits and dashes, up to 64 characters; Helper keeps unknown ids as text) so every device and Helper's browser draw the same gradient for that room. Like the photo it lives outside the layout: set it only with `POST /helper/rooms/{roomID}/palette`, never in a roomSync mutation, where it is ignored and the stored value kept. Writing it changes the catalog `photoRevision` (not the room's own `photoRevision`), so clients re-read the room list; an invalid id is `422`. Myro:Home for Android 1.4.1 or later takes part in this shared room/camera sync too. Helper rejects the older automatic room/camera sync protocol with HTTP 426, and `POST /helper/rooms` (whole-list replacement) returns 410 Gone; update PRO and Classic on every participating iOS device before sharing. The catalog file is version 2; a version 1 file is migrated on first read, keeping panel-numbered and UUID rooms with photos and dropping hash-identified rooms.

Matching room and camera records establish the device's sync baseline even when the review says **Up to date** and no Apply is needed. A new room carries its photo on the first approved sync, with one photo upload per room. Photo transfers use the reviewed local image and Helper photo revision; changing either before the transfer completes requires another review rather than overwriting a newer local choice or uploading an unreviewed image.

Mobile routes below require the existing controller-bound `Authorization: Bearer SESSION_TOKEN` from Helper pairing. Ordinary API keys and browser Control Logins do not authorize them. There is no caller-supplied controller GUID: the session selects the controller. Camera definitions may include addresses and credentials; treat their replies as private and never paste them into an AI prompt or log.

Route	Contract
`GET /helper/roomSync`	Version 1 room records, controllerGUID, layout `revision`, `photoRevision` and conflicts
`POST /helper/roomSync`	Revision-checked room create/edit/delete mutations
`GET /helper/cameras`	Version 1 camera records, controllerGUID, catalog revision and conflicts
`GET /helper/syncStatus`	`{"version":1,"controllerGUID":…,"cameraRevision":…,"roomRevision":…,"photoRevision":…}`; apps call it once when the user taps Edit
`GET /helper/timeline`	The home timeline for a window, exactly as `GET /helper/web/timeline`: `?hours=8&end={ISO 8601}&after={cursor}` (5.50; read only). See **Timeline for PRO**
`GET /helper/timeline/state`	The house at one moment, exactly as `GET /helper/web/timeline/state`: `?at={ISO 8601}` (5.50)

Route	Contract
`GET /helper/timeline/replay`	A window's state changes to replay, exactly as `GET /helper/web/timeline/replay` : `?hours=8&end={ISO 8601}&since={ISO 8601}` (5.50)
`GET /helper/timeline/stills/{id}`	A Timeline picture (JPEG; doorbell or camera), by an id a moment lists (5.50)
`POST /helper/usage`	Upload this device's usage events, 1 to 500 per batch: `{"version":1,"device":{"id","kind","name"?},"events":[…]}` ; replies `{"accepted","duplicates","retained":{"days":90,"events":N}}` (5.50). See **Usage and Suggestions**
`GET /helper/usage`	`?device={id}&since={ISO 8601}` : that device's own events, newest first, at most 2,000 (restore after a reinstall); without `device` , the household's events for a client that ranks locally (5.50)
`GET /helper/suggestions`	`?device={id}&kind={kind}&home=0
`DELETE /helper/usage`	`?device={id}` : forget one device's events; the app's **Share usage with Helper** switch calls it when turned off (5.50)
`POST /helper/cameras`	Revision-checked camera create/edit/delete mutations
`GET /helper/rooms`	Current room definitions, memberships, assigned photo IDs/revisions and, for rooms without a photo, the fallback `palette` id (5.50)
`POST /helper/rooms`	Legacy whole-list replacement; returns `410 Gone` . Use roomSync
`POST /helper/rooms/{roomID}/photo`	Raw image body; archives a normalized master in Photos and assigns it to the room
`POST /helper/rooms/{roomID}/resetPhoto`	JSON `basePhotoRevision` ; clear the assignment while retaining archived photos
`POST /helper/rooms/{roomID}/palette`	JSON `{"palette": "navy-coral"}` (null clears); the fallback art id for a room without a photo. Replies with the room record (5.50)

Reviewed mobile clients send `X-Myro-Sync-Review: 1` for GET/POST `/helper/cameras` and `/helper/roomSync` , `/helper/syncStatus` , mobile room photo/reset writes, and controller-photo POST `/helper/photos` . Missing/unknown markers receive HTTP 426 before catalog seeding or reads/writes. This is a protocol compatibility barrier, not authentication or proof of user consent; mobile bearer authorization is still required. Photo-library browsing and browser routes retain their existing contracts.

Room and camera sync requests use `Content-Type: application/json` and `{"version":1,"mutations":[...],"baseRevision":"CATALOG_REVISION"}` . Native POSTs require the top-level `baseRevision` returned by the latest corresponding snapshot. Helper compares it atomically under the catalog lock, rejecting a stale batch with HTTP 409 before any writes; this also prevents two initial devices creating different shared IDs for the same room. An identical retry after a lost response remains idempotent. Older Helpers without the room

snapshot's catalog `revision` must be updated before reviewed sync. Each mutation also supplies its own `id`, `baseRevision`, and `room` or `camera`. A null room/camera deletes that record; a new record has a null base revision. Read the current records first and send their exact revisions for updates. Room definitions contain `id`, `number`, `name`, and `items`; each item identifies a control by `type` and `number`, or a camera with `type: "Camera"` and `cameraID`. Optional `webHidden: true` on a room or camera hides it from Helper browser views outside edit mode. Native clients preserve this metadata without changing their per-device visibility. Photos have separate assignment/revision fields. Optional item `source` metadata preserves external controls: `backend` (`homekit` or `savant`), stable `id`, and display `name`. Savant sources also carry `installation` (`HOST_UID_UPPERCASE|homeID`), `roomID`, `serviceID` and, when known, the exact `component`, `logicalComponent`, `variantID`, `serviceType` route. External items use `number: 0`; their identity is backend plus source ID, never a device-local object number. HomeKit IDs combine accessory and service identity; no HomeKit credentials or state are shared. Older panel/camera assignments remain valid. Clients must preserve source metadata when renaming rooms or editing other assignments.

Successful sync returns the current snapshot. A `409` sync response also contains the current snapshot and conflicting record IDs; a conflicting batch is rejected without applying its mutations. Reconcile those records and review the latest version before submitting a fresh mutation. Do not blindly overwrite or repeat the entire batch. Deletions retain tombstones so an older device does not silently recreate removed records. Layout and photo revisions are separate. Mobile room-photo upload sends the last known photo revision in `X-Myro-Photo-Base`; a `409` photo conflict keeps the incoming image in Photos without replacing the newer room assignment. An identical already-applied change is safe to retry.

Browser routes use the signed-in browser session and its origin/request checks, not a mobile bearer or a raw Savant password:

Route	Contract
`POST /helper/web/rooms`	One revision-checked room mutation from the room editor
`POST /helper/web/rooms/{roomID}/activity`	JSON `identity` (`savant:` plus saved source ID) and Boolean `on`; starts/stops only that saved room activity
`GET /helper/web/cameras/{cameraID}`	Selected camera configuration for the signed-in editor
`POST /helper/web/cameras`	One revision-checked camera mutation; a UniFi Protect create may add `protect: {consoleID, protectID, picks, pictures}` to link it for camera events (5.50)
`POST /helper/web/protect/cameras`	JSON `server`, `username`, `password` and optional `keep` for a UniFi OS console; signs in and lists its connected Protect cameras with `added`, `isDoorbell`, `detections`, `enabledInProtect` and `defaults`. With `keep: true` (https only) Helper saves the sign-in in the Keychain and listens for camera events; the reply then carries `connection` (5.50)
`GET /helper/web/protect`	The kept UniFi Protect consoles (`consoles`: state, message, server, username, never the password), the Camera events list (`cameras`: links, detections and the owner's `picks`, `pictures`, `twin`), `doorbells` for the twin choice, `cameraStills` and `keychainLocked` (5.50)

Route	Contract
`POST /helper/web/protect/consoles/{consoleID}/disconnect`	Body `{}`; stops camera events from that console and deletes its saved sign-in. Camera links and choices stay. Replies like `GET /helper/web/protect`, plus `signInDeleted` (5.50)
`POST /helper/web/protect/cameras/{cameraID}/detections`	Any of `picks` (from `person`, `vehicle`, `animal`, `package`, `motion`, `ring`), `pictures` (Boolean) and `twin` (a `doorbells` id, `"none"` or null); replies with that camera's entry (5.50)
`GET /helper/web/cameras/{cameraID}/snapshot`	Authenticated image snapshot
`GET /helper/web/cameras/{cameraID}/stream`	Same-origin authenticated WebSocket playback; optional `compatibility=h264` or `mjpeg`, and `audio=1` with `h264` to transcode AAC on request
`POST /helper/web/rooms/{roomID}/photo`	Upload and assign a room image from the browser
`POST /helper/web/rooms/{roomID}/assignPhoto`	JSON `photoID` from this controller's library; null restores the default
`GET /helper/web/photos`	This controller's photo manifest for the browser selector
`GET /helper/web/photos/{photoID}/thumbnail`	Browser-session JPEG preview
`GET /helper/web/photos/{photoID}/image`	Browser-session normalized master

UniFi Protect (5.50). The Add camera view can list a UniFi Protect console's cameras the way PRO does. `POST /helper/web/protect/cameras` takes `{"server":"https://192.168.1.1","username":"LOCAL_USER","password":"…","keep":false}` (a bare address is taken as `https://`; only a scheme, host and optional port are accepted; `keep` is optional). Use a local user, or a UniFi account without two-step verification. Helper posts the login to `{server}/api/auth/login`, reads `{server}/proxy/protect/api/bootstrap` with the session cookie the console returned (`UOS_TOKEN` or `TOKEN`) and its CSRF token, and replies `{"cameras":[{"id","name","state","width","height","rtsp","added","url","parameters","channels":[{"id","name","width","height","fps","bitrate","rtspAlias","url"}],"isDoorbell","detections","enabledInProtect","defaults"}],"connection"?}`. Only connected, adopted, managed cameras are listed (PRO's filter); an empty list means none were found. `url` is `rtsp://{host}:7447/{rtspAlias}` for channel 0 (High), or the best channel with RTSP turned on; `rtsp: false` means no channel has RTSP enabled. `parameters` is PRO's channel list, `{"channels":[{"alias":…,"h":…,"id":0,"w":…},…]}`, which lets Apple TV pick the medium channel. `added` is true when a camera in this controller's catalog already plays that Protect camera, however it was entered (this picker, PRO's `rtsps://user:pass@host:7441/alias` form, or a typed URL), matched by the console's host names and the stream alias. `detections` lists what the camera can detect (`person`, `vehicle`, `animal`, `package`, `motion`, and `ring` on a doorbell), `enabledInProtect` the smart detections turned on in Protect, and `defaults` what the picker pre-ticks (`person`, plus `ring` on a doorbell). The browser then creates each ticked camera through `POST /helper/web/cameras` with `cameraType: 20`, that `url`, `rtspPort: "7447"` and that `parameters`, and no username or password (Protect's RTSP streams need none). The browser keeps one camera `id` per listed camera, so retrying after a lost reply resends the identical record and is accepted once. A Protect camera added in PRO but never synced to Helper is not marked `added`, so sync PRO's cameras to Helper before adding them here. Without `keep`, the login is used for this request only and never stored. Helper never logs the password, the session

cookie or the CSRF token, accepts the console's self-signed certificate only within this request's own connections, follows no redirects and allows 15 seconds per console request. Errors are JSON `{"error", "code"}`: `400` (`invalidServer`, `missingLogin`, `localServer` for an address on this Mac: loopback or `localhost` in any spelling, or Helper's own port 9393 on any host; `httpsRequired` for `keep` over `http`), `403` (`signIn` when the console refuses the sign-in, `noProtectAccess` when the account signs in but UniFi OS gives it no UniFi Protect access, `mfaRequired` when the account needs two-step verification; never `401`, which belongs to the browser session), `429` (`rateLimited`, or `lockedOut` when the console locked the account after too many failed sign-ins; unlock it in UniFi OS or wait), `404 notAvailable` for `keep` on a controller without `protectEvents`, and `502` (`unreachable`, `notProtect`, `unexpected`) when the console cannot be reached, is not running Protect or returns no camera list.

Keeping the sign-in for camera events. With `keep: true` (https only) and a successful sign-in, Helper saves `{server, username, password}` in a device-local Keychain item for this controller and console, and nowhere else: never in a file, the log, a reply or a browser. The console is identified by the bootstrap's own id (so the same console typed as an IP or a host name is one console; keeping it again replaces the item), at most four per controller. Helper also records how the console's certificate was trusted: when it passes system trust for that host, later connections require that again; otherwise (the usual self-signed console) Helper records the SHA-256 of its public key and later accepts a certificate that passes system trust or has that same key. Any other certificate stops the listener before the password is sent. The sign-in's session is handed to the listener, which keeps one session and one updates connection per console, pings it every minute, reconnects with backoff (2 seconds rising to 5 minutes), catches up the events of a gap shorter than 10 minutes, survives sleep, wake and network changes, and stops on Disconnect, controller removal and quit. The reply's `connection` is that console's entry (below); a failed save, or a console that signs in without returning a session Helper can keep, still returns the cameras with `connection.state: "error"` and a `message`, and never removes a sign-in that already works. `connection` is also present without `keep` when this controller already keeps that console.

Sign-in policy. Helper never retries a refused sign-in. A sign-in the console rejects (wrong or changed password), a lockout (`AUTHENTICATION_FAILED_LIMIT_REACHED` or `429`), an account that needs two-step verification, a changed certificate, a saved sign-in Helper cannot read, or more than six automatic sign-ins within an hour stops that console's listener, and the stop is remembered across restarts until the owner signs in again. A sign-in the console did not complete (a `5xx`, a timeout after the request was sent, a reply without a session) is retried once, no sooner than an hour later and only after the console answers, and a second failure stops it. Helper checks that the console answers before every automatic sign-in, so an unreachable console costs no attempts. An account without access to Protect (`notPermitted`) or a console without Protect (`notProtect`) is checked again hourly with the existing session and is never treated as a refused password.

`GET /helper/web/protect` returns the session controller's kept consoles and its Camera events list:

```
{"consoles": [{"id": "p-3f2a9c0d1b2e", "server": "https://192.168.1.1", "username": "danny",
               "state": "connected", "message": "Listening for camera events.", "since": "2026-10-
27T18:02:11Z",
               "stopped": false, "lastEvent": "2026-10-27T18:42:10Z" | null, "linkedCameras": 3}],
 "cameras": [{"id": "4f6c2b8e-…", "name": "Front Door", "console": "p-3f2a9c0d1b2e" | null,
"protectID": "65…" | null,
              "linked": "added" | "matched" | "duplicate" | null, "isDoorbell": true, "online": true
| null, "hidden": false,
              "detections": ["person", "vehicle", "animal", "package", "motion", "ring"] | null,
              "enabledInProtect": ["person", "vehicle", "package"] | null, "defaults": ["person",
"ring"],
              "picks": ["person", "ring"], "pictures": false, "twin": "Zone:12" | "none" | null}],
 "doorbells": [{"id": "Zone:12", "name": "Front Door Bell"}],
 "cameraStills": false, "keychainLocked": false}
```

`state` is `connecting`, `connected`, `reconnecting`, `unreachable` (no answer for an hour), `asleep`, `consoleError` (one more sign-in an hour later), `notPermitted`, `notProtect`, or one of the stopped states `signInNeeded`, `lockedOut`, `mfaRequired`, `certificateChanged`, `sessionUnstable`, `credentialMissing`; `stopped` is true for those (and for a second `consoleError`), and `message` is plain English ready to show. `keychainLocked` is true while the Mac's login keychain is locked, so Helper cannot read the kept sign-ins yet (only for a controller that has kept one; a controller that never connected UniFi Protect always reads false); it retries on unlock, wake and every 15 minutes. The password is never returned. `cameras` lists every type 20 camera and any camera linked to a kept console. `linked` is `added` (linked when added from the picker), `matched` (an existing camera Helper linked by its stream) or `duplicate` (another Helper camera plays the same Protect camera, so this one records nothing). For a kept console that is stopped, Helper has no fresh facts, so `detections`, `enabledInProtect` and `online` are null. `picks` are the detections that become events (an empty list, the default for matched cameras, records nothing), `pictures` the camera's **Keep pictures** choice, and `twin` the panel doorbell a Protect doorbell's ring is the same press as (null for automatic, by room; `"none"` to never merge). Hidden cameras (`hidden: true`) record nothing whatever their picks. `doorbells` lists the marked doorbells a `twin` can name.

`POST /helper/web/protect/consoles/{consoleID}/disconnect` with `{}` stops that console's listener at once, deletes its Keychain item and replies like `GET /helper/web/protect` without it, plus `signInDeleted`; camera links and choices stay, so signing in again restores them. `signInDeleted: false` means the Mac's login keychain was locked: the console stays disconnected and is never signed in to again, and Helper deletes the saved sign-in when the keychain unlocks. `POST /helper/web/protect/cameras/{cameraID}/detections` takes any of `{"picks": ["person", "ring"], "pictures": false, "twin": "Zone:12"}` (at least one; `picks` a subset of the six names without duplicates, `[]` allowed; `pictures` a JSON Boolean; `twin` a `doorbells` id, `"none"` or null), saves them for any camera in Helper, linked or not, wakes `/helper/web/changes` and replies with that camera's `cameras` entry. `POST /helper/web/cameras` may carry `"protect": {"consoleID", "protectID", "picks", "pictures"}` next to `version` and `mutations`; it is honored only for a single create of a `cameraType: 20` camera whose `url` plays that Protect camera on a console this controller keeps, and that no other Helper camera already plays. The camera is saved either way. These routes answer `400 badParameter` for a malformed id or body, `404 notFound` for an unknown camera or console, `404 notAvailable` on a controller without `protectEvents`, and `500 storage` when Helper cannot save the choices.

Browser snapshot room items include `webHidden` and, for external cards, `source`, `available`, optional `active`, and `unavailableReason`. Normal room views omit hidden controls, HomeKit assignments and unavailable Savant activities; edit mode retains them with explanations. Browser room writes accept optional `webHiddenItems`, an array of assigned item identities for the single room mutation. These Helper-only visibility choices persist independently of native membership, so PRO/Classic sync cannot erase them. Omitting the field preserves existing choices; an empty array clears them. Unknown or duplicate identities are rejected. Snapshot rooms also include `webVisibilityRevision`; browser writes send it as `baseWebVisibilityRevision`. Visibility has its own revision so hiding a control cannot conflict with a native room rename. Layout and visibility still commit atomically; a stale competing visibility save returns 409. Savant availability requires a unique configured installation, a unique saved activity route, current feedback and discovered PowerOn/PowerOff commands. If exact route metadata exists, it can recover a regenerated service UUID; missing/ambiguous routes never fall back to display names. The activity route rechecks the signed-in controller's current room membership, resolves the destination internally and checks its connection/binding on the transport queue. It accepts no destination controller, host or raw command. A room assignment delegates only its selected activity to that room's authorized browser users. Paired-client sync introduces source routes; browser editors can reuse existing shared sources but cannot invent or retarget them. A new/changed browser source route returns 403 and must first be synced from paired PRO. Success means queued, not confirmed; the browser waits for Savant feedback and does not replay commands. Unavailable activity control returns `503`. Savant feedback wakes browsers whose shared rooms include Savant activities. Updated PRO exports this metadata; Classic preserves unresolved assignments through room edits.

The Rooms and Cameras pages initially hide management actions. Their sliders edit toggle reveals Add/Edit and room-photo actions; the checkmark exits edit mode. Switching edit mode retains decoded photos/snapshots; camera playback follows the same visibility and decoder limits. Hidden items remain visible in edit mode so they can be restored. The containing view region has no page-wide focus outline; individual interactive controls retain keyboard focus indicators.

The browser camera page is a live wall. Every visible card plays live video, up to six at once per browser (decoders, not sockets, are the scarce resource); further visible cards show paced snapshots (about every ten seconds, at most two requests in flight). Off-screen cards retain their last poster without polling. Leaving the view, hiding the page or signing out closes every stream. **Watch live** moves the card's own player into a larger viewer without reconnecting, adds mute (only when the stream carries audio), a format selector (Auto, H.264, MJPEG), fullscreen and a stats line (format, codec, size, latency behind live, bitrate, reconnects); closing hands the stream back to the card.

The player never shows the browser's native video controls; a status pill reports LIVE, Connecting, Reconnecting, Stalled, Busy, Offline or Unsupported. It reconnects without limit using exponential backoff with jitter (1 s to 30 s), retries at once when the network or page returns, reconnects when decoded playback stalls for eight seconds, or an open socket delivers no media for sixteen seconds, treats media-buffer errors as recoverable, and steps down a source ladder (native MSE, then `compatibility=h264`, then `mjpeg`) when the browser cannot decode a format, remembering that choice per controller, camera and camera revision for a day. Playback is kept within about half a second of the live edge.

When Helper is already serving its maximum number of tunnels (32) the stream route completes the WebSocket handshake and sends `{"type":"error","value":"busy"}` before closing, so the player shows Busy and retries, instead of a bare `503`. Helper writes its own owner PID and the go2rtc child's PID into its private workspace, replaces the child if it exits, restarts it after the Mac wakes from sleep (every camera TCP session is dead by then) and, at launch, terminates a matching bundled `myro-go2rtc` orphan only after

confirming its recorded owner is dead and the child has no live parent. Active Helper instances and unknown workspaces are preserved. Stream registration runs off the connection queue, so one slow camera no longer delays other viewers or snapshots. The existing routes and authorization rules are unchanged.

Browser stream setup uses the normal session cookie with a verified WebSocket Origin. Playback rechecks access and the camera definition; sign-out or a configuration change invalidates the old stream. Viewing a camera does not give the browser its camera password. Invalid catalog bodies return `400`, missing authorization `401`, missing resources `404`, revision conflicts `409`, and unexpected internal failures `500`. JSON catalog replies use `Cache-Control: private, no-store`.

Photo Library for PRO

The Helper Mac also offers **API > Rooms & Cameras** for the selected controller, using the same revision-checked catalog as the browser. **Open Editor...** (also **Window > Rooms & Cameras...**) opens a resizable native window: rooms and cameras on the left; for a room, the name, web visibility, photo (library, upload or default) and a searchable list of the controller's controls with assign and show-in-web checkboxes; for a camera, every connection field and a snapshot preview. Panel rooms offer Hide/Show, app rooms and cameras offer Remove with confirmation. Saves use the record's revision; a change made elsewhere shows "changed elsewhere" and nothing is overwritten. The pane's Rename and Remove shortcuts remain. API > Photos > Open Photo Manager... (also Window > Photo Manager...) opens a separate, resizable library window with import, rename, replace, remove and exact-duplicate cleanup. Each controller has its own window, updated by the same library notifications as the Photos tab. None of this triggers automatic device sync; PRO and Classic pick up changes at their next reviewed sync.

Manage reusable room and home photos on the Helper Mac under **API > Photos**, with the intended controller selected in the sidebar. Import several images at once; rename, replace, or remove a selected photo. Libraries are separate for each controller. Removing a controller from Helper removes its library.

For a Savant controller, Helper automatically downloads each available room photo once after local sign-in and successful room discovery. Reconnects and app restarts reuse saved photos without requesting those images again; newly discovered rooms get their first download. Explicitly choosing **Refresh Rooms** downloads existing photos again to check for updates. Original-size images are requested from the configured Savant host; Helper stores the normalized JPEG masters described below. Room IDs identify each source, so reconnecting or renaming a room does not create duplicate entries. A local photo rename is retained; replacing or removing an automatically imported photo prevents later automatic imports from overwriting that choice. Manually imported photos are independent, even if their names match. Missing images are skipped, and unavailable or removed Savant rooms do not delete previously saved photos. These Savant downloads require no Helper API key.

PRO uses the existing mobile bearer session for these routes. API keys and browser Control Logins are not accepted. The session identifies the Helper controller; its GUID may differ from a mobile device's own controller record GUID.

Route	Result
`GET /helper/photos`	Versioned manifest identifying the authenticated controller and its photos
`POST /helper/photos?name=PHOTO_NAME&requestID=UUID`	Add an image to the authenticated controller's library; body is the raw image, with an `image/*` Content-Type

Route	Result
`GET /helper/photos/{photoID}/thumbnail`	Small JPEG preview, at most 400 pixels on its longest side
`GET /helper/photos/{photoID}/image`	Normalized master JPEG at the imported photo's full resolution

Example list response:

```
{
  "version": 1,
  "controllerGUID": "HELPER_CONTROLLER_GUID",
  "photos": [{
    "id": "PHOTO_UUID",
    "name": "Theater",
    "width": 3072,
    "height": 2304,
    "revision": "REVISION_UUID",
    "thumbnailPath": "/helper/photos/PHOTO_UUID/thumbnail",
    "imagePath": "/helper/photos/PHOTO_UUID/image"
  }]
}
```

Photo IDs remain stable on rename and replacement. Replacement generates a new revision; image responses include a rendition-specific `ETag`. All photo replies use `Cache-Control: private, no-store`. Missing or expired bearer tokens receive `401`; missing, invalid, or another controller's photo ID receives `404`. Clients handle a `404` for the list route on older Helpers by showing that the photo library requires an updated Helper. A storage read failure receives `500`, not an empty library.

Import accepts ImageIO-supported images under 40 MB, up to 60 megapixels and 16,384 pixels on either side. Helper applies rotation and writes new JPEG masters and previews without the source's GPS, camera, or other metadata. Masters are stored persistently in Application Support rather than an evictable cache. The master is reencoded at high JPEG quality; it is not a byte-for-byte original. Rename, replacement, and removal remain local to Helper. The upload route adds a new photo and never replaces an existing photo by name.

For uploads, use the same mobile bearer session as downloads. Helper derives the destination controller from that session and verifies the controller still exists with the same pairing fingerprint. Caller-supplied controller IDs cannot change the destination. The photo name must contain 1–120 characters after trimming, with no control characters. URL-encode the name and use a fresh UUID `requestID` for each new photo. A successful reply is `200` with `{"version":1,"controllerGUID":"HELPER_CONTROLLER_GUID","photo":{...}}`, where `photo` has the same fields as a manifest entry.

Retrying the same name and image with the same `requestID` returns the original entry, including after Helper restarts. New uploads containing exact existing image bytes reuse the existing photo ID/name/revision even with a new request ID or name. Each request receipt remains idempotent and enforces its own content-conflict/removal checks. Explicit photo replacement retains its chosen ID. **API > Photos > Combine Exact Duplicates…** previews older identical stored masters before consolidation; old image IDs and revisions remain addressable, including room references and independent Savant sources. No API payload fields or authentication requirements change. Reusing that ID for different content receives `409`; retrying an upload

removed in Helper receives `404` with code `photo_removed` and does not recreate it. Missing names, invalid IDs, or an empty body receive `400`; oversized images receive `413`; unsupported Content-Types or invalid image content receive `415`. A deleted or re-keyed controller rejects the stale session with `401`, or an in-progress import invalidated by deletion receives `409`. Storage failures receive `500`.

Choosing **From Helper** in PRO explicitly downloads a copy into the device's existing photo storage. PRO continues to use that local copy when Helper is unavailable. For photos chosen as independent library copies, importing, replacing, or removing a Helper photo does not automatically overwrite or delete those device copies. Shared room-photo assignments use the separate revision-aware room sync described above. Users can select the photo again to copy a newer version. This library is separate from the single web-control background described below.

When adding a photo from the device library, camera, or clipboard, PRO offers **Save a copy to Helper** on the crop screen when a paired Helper is available. The photo still saves on the device first. The optional Helper sheet lets the user name the cropped photo and choose a destination if several Helpers serve that controller. Closing the sheet or an upload failure leaves the device copy in place. Older Helpers without the upload route show an update message. Images selected from **From Helper** do not offer to upload another copy back.

Home Photo for the Web Control

The web control shows the controller's photo from the Myro:Home apps behind its screens. The first app to connect for a controller that has no photo sends one; after that, apps leave it alone so different phones do not replace each other. To change it, remove it in Helper (**API > Control Login**) or choose **Change photo** in the web control's Settings; the next app to connect then sends its own.

- `POST /helper/auth/respond` includes `hasWebBackground` (true or false), so an app knows without asking whether to send its photo.
- `GET /helper/background` returns `{"controllerGUID", "hasImage", "updated"}` for the session's controller.
- `POST /helper/background` with the photo as the request body (`Content-Type: image/jpeg`, `image/png`, `image/heic` and similar, up to 12 MB) stores it only when none is set: `{"stored": true}`, or `{"stored": false, "reason": ...}` when one already is. `413` when too large, `415` when the body is not an image.

Helper decodes every photo and saves a fresh JPEG, at most 2,400 pixels on its longest side, rotated upright, with no location, camera or other metadata.

The web control reads and changes it with its browser session:

- `GET /helper/web/background` returns the photo as `image/jpeg`, or `404` when there is none.
- `POST /helper/web/background` with an image body replaces the photo (an explicit choice in Settings).
- `POST /helper/web/background/remove` with `{}` removes it.

Sessions expire and are bounded in memory. Clients should rediscover and reauthenticate when a token is rejected.

Usage and Suggestions (5.50)

Helper 5.50, Myro:Home Pro 3.10 and Myro:Home Pro for Apple TV 2.0 share one usage scheme; Myro:Home and Myro:Home for Android follow with the same one. **Usage is what a person did in an app**: a tap that sent a command. Schedules, switch presses, HomeKit and state echoes are never usage. Each app keeps its own log and ranks it offline; Helper is the household store and ranker, so the feature works without a Helper and gets better with one. From the log Helper and the apps build a short "Right now" list: the things this person, on this device, usually does at about this time of a day like today. **Suggestions never act.** They are one-tap shortcuts, like Recents; a security suggestion carries `confirm: true` and goes through the normal code or biometric path. There is no model: decayed counts in half-hour buckets, and every suggestion carries human reasons.

Everything stays local. Logs live on the device and in `Application Support/MyroHomeHelper/Usage/<controller guid>.json` on the Helper Mac, written atomically; nothing goes to a cloud. The apps have a **Share usage with Helper** switch (on by default once a Helper is paired); turning it off deletes that device's events from Helper. The browser's Settings has **Clear usage history**, which empties the controller's store. Helper keeps 90 days and at most 50,000 events per controller, pruning on every write, and removes the file with the controller. Shared screens are read-only: a browser or wall-tablet session gets suggestions and never uploads. An API key reads the household ranking only (see **Suggestions** under **Controller Reads**), never the events.

Identities. `object` is the favorites id form: `"<Type>:<number>"` for numbered panel objects with the type's spaces removed (`"Unit:12"`, `"Button:3"`, `"Thermostat:1"`, `"Area:1"`, `"Zone:7"`, `"AudioZone:2"`, `"Flag:4"`, `"Output:2"`, `"UserSetting:3"`, `"Lutron:5"`), `"Activity:<service id>"` for a Savant activity, `"Activity:<room id>:<activity id>"` for a Control4 activity (its `watch:<source>` id is unique only within a room; the room id is the activity object's `room` on the snapshot), and `"HomeKit:<accessory uuid>"` for HomeKit-only items, which Helper stores as text without interpreting. Locks keep two distinct spellings: `"AccessControl:<n>"` for an "Access Control" object (the lock the apps command) and `"Lock:<n>"` for an "Access Control Lock". Rooms, cameras and access-control readers are never usage objects. `device.id` is a UUID the app makes once per install and keeps (in the Keychain on Apple platforms, so it survives a reinstall; it is not secret and not the vendor id). A Watch relays through its phone with `kind: "watch"` and the phone's id suffixed `.watch`; an Apple TV uses its paired-device id. `device.kind` is one of `iphone`, `ipad`, `mac`, `watch`, `tv`, `tablet`, `android` or `web`.

The event. Logged when the command is *sent*, never for state echoes, refreshes or HomeKit observers:

```
{"id":"<uuid>","object":"Unit:12","at":"2026-10-28T21:34:10-
        07:00","action":"on","value":100,"home":true}
```

`id` is a client UUID; Helper dedupes on `(device.id, id)`. `at` is ISO 8601 **with the device's UTC offset**: the bucket is local time, and the offset carries it. `action` is one of `on`, `off`, `level`, `open`, `close`, `stop`, `run`, `arm`, `disarm`, `bypass`, `restore`, `lock`, `unlock`, `set`, `play`, `pause`, `volume` or `other`; `value` is an optional number (a level, setpoint or volume). `home` is `true`, `false` or `null` from the app's LAN-subnet check at the moment of the tap (`null` is unknown). A device keeps 90 days and at most 20,000 events, oldest dropped; `useCount` and `usedLast` stay as derived values and keep feeding today's Recent.

Routes. All are per controller like room sync (the session or key selects the controller); `X-Myro-Sync-Review: 1` is not required.

Route	Credential	Contract
`POST /helper/usage`	Mobile bearer	`{"version":1,"device":{"id","kind","name"?},"events":[…]}`, 1 to 500 events. Reply `{"accepted":n,"duplicates":n,"retained":{"days":90,"events":N}}`. `413` above 500 events, `400` for a malformed event, `422` for a bad `object` or `kind`.
`POST /helper/web/usage`	Paired Apple TV session	The same body and reply, but the device is the pairing's: the TV may omit `device` (or send it; Helper ignores it), `kind` is `tv` and `name` the paired name, so the events are kept under the paired-device id the device list shows. A browser, tablet or password-signed-in TV session gets `403 deviceRestricted`.
`GET /helper/usage?device={id}&since={ISO 8601}&limit={n}`	Mobile bearer	That device's own events (restore after a reinstall), newest first, at most 2,000, as `{"device"?,"since"?,"limit","count","events":[…]}`. Without `device`, the household's events for a client that ranks locally.
`GET /helper/suggestions?device={id}&kind={kind}&home=0	1&at={ISO 8601}&limit={n}&compose=0	1`
`GET /helper/web/suggestions?…`	Browser, tablet or paired Apple TV	The same reply. The `device` query is ignored: a paired Apple TV is its own device with `kind=tv`, an enrolled tablet its own with `kind=tablet`, and a sign-in (a password-signed-in TV included) gets the household ranking as `web`, or the `kind` it asks for. Never `403`: a session that can read the snapshot can read suggestions.
`GET /helper/{controllerID}/suggestions?…`	API key	The same reply, household only (no `device`); see **Suggestions** under **Controller Reads**.
`DELETE /helper/usage?device={id}`	Mobile bearer	Forget one device's events; replies `{"success":true,"device","removed":n}`. The app's **Share usage with Helper** switch calls this when turned off.

Route	Credential	Contract
`GET /helper/web/usage/summary`	Browser, full sign-in	Event counts per device for the Settings panel: `{"retained":{"days":90,"events":N,"cap":50000},"devices":[{"id","kind","name"?,"events","lastEvent"?,"lastUpload"?}]}`, most recent upload first. A tablet or paired device gets `403 deviceRestricted`.
`POST /helper/web/usage/clear`	Browser, full sign-in	Body `{}`; clears the controller's whole store, logs it, and replies `success: true` with the empty summary. A tablet or paired device gets `403 deviceRestricted`.

The reply. `name` and `room` come from Helper's object table (apps use their own names) and are omitted for an object Helper does not know; `roomID` is the web room id of an activity (absent for panel objects, and omitted for a Savant service that sits in more than one room, where it would be a guess); `value` is the action's usual value when its events carried one (the arm mode for `arm`, the level for `level`, the setting for `set`: the heaviest value among the dominant action's events, the larger on a tie; a recent tile carries its event's value); `state` is omitted when Helper does not know the object's state (it reads the Timeline ledger at `at`, else the current status). `favorite` and `why` are present in a composed list only; `lastUsed` is omitted for a favorite nobody has used yet.


```
{"at":"2026-10-28T21:30:00-07:00","bucket":43,"dayClass":"weekday","home":true,
 "evidence":{"device":412,"household":1890,"devices":3},
 "suggestions":[
  {"object":"Unit:12","name":"Reading lamp","room":"Bedroom","action":"on","score":4.31,
   "personal":3.9,"household":5.2,"confirm":false,
   "state":{"on":false,"level":0},
   "favorite":true,"why":["favorite","usual"],
   "reasons":["usually around now","most weekday evenings","on this device"],
   "lastUsed":"2026-10-27T21:41:03-07:00"}
 ]}
```


The smart list. Unless `compose=0`, the list is one "Right now" row of favorites, usual and recent items, composed by Helper so a client only renders it (Myro:Home Pro for Apple TV replaces its Favorites and Recents sections with it; an app ranking offline composes the same way with its own favorites and recents). It starts from the ranked suggestions below. The controller's shared favorites (the browser's starred list, in its order) are always present: a favorite that is also usual gets `+1.0` on its score; a favorite with no usage score sits after every scored item, in the favorites' own order, with score 0 and an `action` that is the opposite of its state (`off` for a light that is on, `disarm` for an armed area, `unlock` for a locked door, `stop` for a running activity) or the type's usual verb (`run` for a button, `set` for a thermostat, `other` for a status tile such as a zone); favorites are never demoted by the state check (`state` still says what is on). Recents fill the tail: objects this device used in the last 3 days that are not already listed, newest first, with score 0 and `reasons: ["used in the last 3 days"]`; a sign-in or API key has no device and so no recents. Each item carries `favorite` and `why`, a subset of `["favorite", "usual", "recent"]` in that order ("usual" means it passed

the evidence gate for this time). Ordering is by composed score, then the favorites' order, then `lastUsed`. Hidden objects (the row's hidden flag or an item hidden in a web room), rooms and cameras are never listed, even as favorites. Favorites and hidden items are matched to usage ids by their type with spaces removed (`Audio Zone:2` is `AudioZone:2`, an access-control lock is `Lock:1`) and an activity by its service id (`Activity:<room>:<id>` is `Activity:<id>`).

Scoring, identical on Helper and on every device so a list ranks the same offline. The bucket `b` is the half-hour of the local day (0 to 47), the day class `d` is weekday or weekend, and `h` is the home flag. Each event weighs `w = 0.5 ^ (ageDays / 14)`. An event matches when it has the same `d` and the same `h` (an event with `home: null` matches either) and its bucket is within one of the requested bucket, wrapping at midnight; it contributes `w` in the same bucket and `w / 2` in a neighbouring one. `personal` sums the requesting device's matching events and `household` sums every device's (the requesting device included). The `score` blends them: `0.7 × personal + 0.3 × household` once the device has 20 events in total, otherwise `0.3 × personal + 0.7 × household`, so a new device leans on the house. An object needs at least 3 matching events across personal and household or it is left out. Its suggested `action` is the one with the highest summed weight in that bucket; when the object's current state already satisfies it (on or off, armed, locked, running, open) the score is multiplied by 0.2, Helper reading its Timeline ledger and status and the apps their own panel state. Hidden objects and rooms are excluded. Security actions (`arm`, `disarm`, `unlock`, `bypass`) stay in the list flagged `confirm: true`. The top `limit` are returned, ties broken by the latest use. The shared fixture `Tools/Tests/UsageRankerFixture.json` (events plus the expected ranking for three `at` values) is what every platform's test replays.

Errors. Every error is `{"error", "code"}`: `401 unauthorized` for a missing or unknown token or session, `400 badParameter` for a malformed `at`, `home`, `kind`, `limit`, `compose`, `device` or `since`, `400 badRequest` for a body that is not the upload JSON, `400 invalidEvent` for an event without an id, a time with its offset or a known action (or an empty `events`), `400 version` for a `version` other than 1, `413 tooMany` above 500 events or 1 MB of JSON, `422 invalidObject` for an object that is not `<Type>:<id>` or is a room, `422 invalidKind` for an unknown `device.kind`, `403 deviceRestricted` for an upload from anything but a paired Apple TV and for a summary or clear from a tablet or paired device, `404 notAvailable` for a controller Helper keeps no usage for (one it has removed), `500 storage` when the history cannot be read or written, and a `404` without a `code` from a Helper older than 5.50. JSON replies use `Cache-Control: private, no-store`. A paired Apple TV detects support with `GET /helper/web/suggestions?limit=1` or the `suggestions` capability.

Internal Sonos Clip Transport

`HEAD /helper/sonos/clip/{token}` and `GET /helper/sonos/clip/{token}` are capability URLs generated by the Helper for a Sonos speaker to fetch a temporary announcement clip. They support byte ranges and are not general file-serving endpoints.

The random token is held only in memory, maps to one staged clip, and expires after use. Clients should not construct, cache, or enumerate these URLs.

Request Examples

Set these shell variables without committing the values:

```
HOST="grouse-mac-mini.local"
KEY="YOUR_API_KEY"
CONTROLLER="953d8997-ab41-4aec-9f58-78235e36d46f"
BASE="http://$HOST:9393/$KEY/api/v1"
```

Read one thermostat:

```
curl --fail-with-body \
  "$BASE/helper/$CONTROLLER/object/Thermostat/2"
```

Set a light to 40 percent:

```
curl --fail-with-body \
  -H "Content-Type: application/json" \
  -d '{"action":"set_unit","number":35,"value":40}' \
  "$BASE/helper/$CONTROLLER/control"
```

Read recent Ethernet history without an API key:

```
curl --fail-with-body \
  "http://$HOST:9393/helper/graphs/ethernet?controller=$CONTROLLER"
```

Myro Control, LLC - myrocontrol.com