Myro:Home Helper

MCP Guide

Version 5.50

Connect an AI assistant to Helper safely through Myro:Home Helper.

Myro Control, LLC | myrocontrol.com

Contents

What MCP Adds

Before You Begin

Create the API Key

Install uv

Configure Claude Desktop

- Configure Claude Code
- Store an Optional Security Code
- Testing a Local MCP Checkout
- Multiple Controllers

Confirm the Connection

What You Can Ask

- Read and Diagnose
- Control the System
- Create Scripts Safely
- Advanced Scripting

Optional Security Control

Troubleshooting

Updating and Reference Material

What MCP Adds

The Myro:Home Helper MCP server lets a compatible AI assistant work with Helper 5.50 through its local HTTP API. Supported systems include HAI / Leviton, NESS / ELK (preview), Savant (preview), and Control4 Link (preview), with controller-specific capabilities. Instead of guessing URLs or controller commands, the assistant receives named tools with clear inputs and safety guidance.

You can ask about controller health, zones, lights, thermostats, audio, sensor history, Ethernet quality, and configured integrations. With the appropriate API-key permissions, the assistant can also control devices and help create, validate, and manage Helper JavaScript automations.

The connection remains local:

```
AI client -> Myro:Home Helper MCP server -> Helper on port 9393
                                         -> HAI / Leviton, NESS / ELK, Savant, or Control4
```

Tool results are returned to the AI client and may become part of that conversation. Do not request or expose information that you would not want included in the client's conversation history.

Before You Begin

You need:

Item	Requirement
Myro:Home Helper	Version 5.50 for this guide, configured for the intended controller
MCP server	`myrohome-helper-mcp` version 0.5.0 or later for the tools described here (0.6.0, 0.7.0 and 0.8.0 add the tools named below)
MCP client	Claude Desktop, Claude Code, or another client that supports local standard-input/output MCP servers
Python launcher	`uv` and its `uvx` command
Network	The MCP client must be able to reach the Helper Mac on TCP port `9393`
API key	A key generated for the controller in Helper's **API** tab

The Helper HTTP service uses plain HTTP and is intended for a trusted local network. Never expose port `9393`, an API key, or the controller directly to the public Internet.

Create the API Key

In Myro:Home Helper, select the controller and open the **API** tab. Enable the API server if needed, then select **Add** to generate a key.

Every key has standard access for controller reads, non-security control, script reads, and script validation. Two additional permissions are optional and independent:

Permission	What It Allows
Arm and disarm through the HTTP API	MCP arming with a stored panel code, plus zone bypass and restore with or without one
Remote script management	Save, enable, disable, and reload scripts through MCP

Remote script management is powerful. A managed script can use every local scripting function, including security control without a panel code. Grant this permission only to a trusted MCP client when you intend to manage scripts remotely.

Capabilities are fixed for the life of a key. Existing keys created before this option was available remain standard keys. Remove and replace a key when different permissions are needed.

Install uv

If `uv` is not already installed, follow the current instructions at docs.astral.sh/uv. On a Mac with Homebrew, you can use:

```
brew install uv
```

The `uvx` command downloads and runs the published `myrohome-helper-mcp` package in an isolated environment. It does not modify Myro:Home Helper or the controller configuration.

Configure Claude Desktop

Open **Claude Desktop > Settings > Developer > Edit Config**. Add a `myrohome-helper` entry under `mcpServers`:

```
{
  "mcpServers": {
    "myrohome-helper": {
      "command": "uvx",
      "args": ["myrohome-helper-mcp"],
      "env": {
        "MHH_URL": "http://192.168.1.76:9393",
        "MHH_API_KEY": "YOUR_API_KEY"
      }
    }
  }
}
```

Replace the address with the IP address or Bonjour hostname of the Mac running Helper. Replace `YOUR_API_KEY` with the key generated for this controller. When Claude Desktop and Helper run on the same Mac, `http://localhost:9393` is also valid.

If the configuration already contains other MCP servers, add only the `myrohome-helper` object inside the existing `mcpServers` object. Save the file, fully quit Claude Desktop, and reopen it.

Configure Claude Code

Run this command in Terminal, replacing the address and API key with your own values:

```
claude mcp add myrohome-helper \
  -e MHH_URL=http://192.168.1.76:9393 \
  -e MHH_API_KEY=YOUR_API_KEY \
  -- uvx myrohome-helper-mcp
```

Restart Claude Code after changing the configuration so it reloads the MCP server and tool descriptions.

Store an Optional Security Code

The direct arming tool remains disabled until a panel code is stored locally. Disarming and door unlocking are off unless you allow them, as described in Optional Security Control below. Zone bypass and restore use the validated user when a code is supplied; behavior without a code depends on the controller family. Follow the security rules in the selected controller's guide. On the same computer that runs the MCP server, open Terminal and run:

```
uvx myrohome-helper-mcp security-code set
```

The command prompts for the Helper address, API key, and panel code according to the controller's code length. Use an adapter release that accepts the configured length. The API key and code are entered without echoing. The server resolves the controller through Helper and stores the code under that controller's identity in macOS Keychain, Windows Credential Locker, or Linux Secret Service.

The code is not placed in the MCP configuration, an environment variable, or an AI conversation. Run the command once for each controller that needs direct security control. Remove a stored code with:

```
uvx myrohome-helper-mcp security-code delete
```

Testing a Local MCP Checkout

Developers testing an unpublished checkout can replace the `args` value with:

```
[
  "--from",
  "/absolute/path/to/myrohome-helper-mcp",
  "myrohome-helper-mcp"
]
```

Return to `["myrohome-helper-mcp"]` after the release is published so `uvx` uses the PyPI package.

Multiple Controllers

Create a separate API key and MCP server entry for each controller. Give each entry a distinct name such as `myrohome-main` and `myrohome-office`. Normally the MCP server discovers the controller associated with its key. `MHH_CONTROLLER_ID` may be added only when a configuration needs to pin that identity explicitly. Run `security-code set` with each controller's matching API key when direct security control is needed.

Confirm the Connection

Start with read-only requests:

> Use Myro:Home Helper to show its version, controller name, and connection status.

> List my thermostats and current temperatures. Do not change anything.

> Check recent Ethernet quality and explain whether any observation needs attention.

The assistant should call `helper_info` first and then use the relevant read tools. If those requests work, the MCP server, API key, selected controller, and network path are all functioning.

What You Can Ask

Read and Diagnose

- Controller identity, firmware, connection status, power, battery, phone-line state, and troubles
- All supported named controller objects and their decoded state
- Thermostats, auxiliary sensors, graph history, and Ethernet-quality observations. On HAI and Leviton panels with Helper 5.01, thermostats report `holdMode`: off, hold or vacation. The assistant can switch hold on or off but cannot set a vacation hold
- Door access on HAI and Leviton panels: whether each access-control lock is locked and, from its reader, whether the last badge or keypad code was granted or denied, the user number and when Helper received it. Helper reads this at startup and on every status refresh, not only after the next access. User codes are never exposed, and reading door access never unlocks anything; see the HAI guide
- With the matching MCP adapter and Helper 4.50: `history_window` for bounded date ranges and reported HVAC action intervals; `panel_events` and `refresh_panel_events` for the panel's own event history. A recent refresh is the default; `refresh_panel_events(scope="all")` explicitly requests a longer read, limited to 2,000 records and two minutes. Any refresh has a 30-second cooldown, and full downloads a ten-minute cooldown. On `429`, wait and read the cached events instead of repeating the download. Wait until `refreshing` is false and check `refreshError` and `panelEndReached` before describing a download as complete. The panel cannot return events it has already overwritten.

For runtime comparisons, `history_window(..., runtime_bucket="day")` returns daily intervals; `hour`, `week`, and `auto` are also supported. These are estimates from reported thermostat actions, not measured equipment power. Unknown coverage is never off time. The same query includes compressed history, and any `warnings` must be disclosed when summarizing results.

- Whole-home audio sources and cached metadata
- Recent Helper log lines for troubleshooting
- With Helper 5.50 and MCP adapter 0.7.0: the objects starred in Helper's browser control (`list_favorites`), the zones, flags and outputs marked as doorbells (`list_doorbells`), which browser and Apple TV features Helper offers for the controller (`web_capabilities`), and the household's "Right now" suggestions ranked from the taps the apps share with Helper (`suggestions`)
- With Helper 5.50 and MCP adapter 0.8.0: the home Timeline's moments and events for the last 1 to 24 hours, without pictures (`timeline`), and two writes that are not physical actions: a note (`timeline_note`) or a named moment (`timeline_moment`) added to the Timeline
- Existing scripts, validation results, and the live scripting reference

Control the System

The installed MCP adapter offers typed controller tools plus shared Helper tools for notifications, speech, synchronization, history and scripting. Supported controller actions, ranges, timers and audio/security behavior are documented in the resolved controller's guide. A named tool is not proof that its panel supports the feature.

On HAI and Leviton panels, `blink_upb_unit` (Helper 5.01 with the matching MCP adapter) blinks a UPB light for 30 seconds so you can find which device it is, or stops it early. Only UPB units are accepted.

Helper 5.01 with MCP adapter 0.6.0 adds `say_panel_message`, `log_panel_message`, `set_keypad_beepers`, `press_radiora2_phantom`, `set_leviton_scene` and `lock_all_doors`, and an optional `seconds` on `unlock_door` for a timed unlock (still only if the owner allowed unlocking). There is no tool to unlock every door or to phone and say: those are left to the apps, scripts and the HTTP API.

Helper 5.50 with MCP adapter 0.7.0 adds four read-only tools. Each needs Helper 5.50; an older Helper answers with `requiredHelper` instead.

Tool	What It Returns
`list_favorites`	The objects starred in Helper's browser control (shared by every signed-in browser), in starred order, matched to their names and current state
`list_doorbells`	The zones, flags and outputs marked **Treat as doorbell** in the browser control, with their current state. On Helper 5.50, `list_objects` and `get_object` also carry `"doorbell": true` on each marked object; only objects Helper has a name for appear
`web_capabilities`	Which browser and Apple TV features Helper offers for this controller (`favorites`, `doorbell`, `history`, `timeline`, `pair`, `protect`, `protectEvents`, and `savant` on Savant controllers), with the Helper version. It reads the public controller list that the browser control and Myro:Home Pro for Apple TV use, and returns only this controller's entry. `offered: false` means Helper does not offer the controller for browser control, as with an HAI / Leviton controller whose panel key Helper does not hold. A Control4 controller is offered, without `savant` or `protectEvents`
`suggestions`	The household's **Right now** list: the shared favorites first, then the things people in this home usually do at about this time on a day like today, ranked from the taps Myro:Home Pro and the Apple TV share with Helper, each with its object, name, room, suggested action, score, whether it is a favorite, current state when Helper knows it, plain-English reasons, a one-line `why` and when it was last used. Optional `home` (true or false), `at` (an ISO 8601 time with offset, default now), `limit` (1 to 24, default 6) and `compose` (false for the plain ranking of up to 12, without favorites). An API key has no device of its own, so the list is the household's, never one person's, and the tool never sees the usage events. A suggestion with `confirm: true` is a security action: it is only a suggestion, and MCP's own security rules still apply

None of these change anything. Starring favorites and marking doorbells stay in the browser control; a suggestion is never acted on by the tool, and the assistant must not treat one as a reason to arm, disarm or unlock.

Helper 5.50 with MCP adapter 0.8.0 adds the Timeline: one read and two writes through the keyed routes `GET /helper/{controllerID}/timeline` and `POST /helper/{controllerID}/timeline/notes`. Each needs Helper 5.50; an older Helper answers with `requiredHelper`.

Tool	What It Does
`timeline`	The home Timeline for the last `hours` (1 to 24, default 8), as the browser control, Myro:Home Pro and the Apple TV show it: the headline, the moments (arrivals, departures, doorbell rings, alarms, scenes, music, busy rooms and the named moments scripts or this adapter wrote), the events (each with its time, kind, title, object, room, importance and `source`; a light the panel changed quietly carries `approximate: true`), the flags (a light left on, a door left open, offline) and the current security, lights and open doors. Optional `kind` keeps only events of that kind (one, or several separated by commas). Pictures are never returned, and the per-room drawing data (`lanes`) is left out of the result. Read-only
`timeline_note`	Writes one note into the Timeline (`source: "api"`): `title` (at most 80 characters), optional `kind` (an event kind; default `note`), `object` (an id such as `Unit:12` or a name that matches one object), `room`, `at` (ISO 8601, within the last 72 hours), `importance` (0, 1 or 2; default 1) and `detail` (at most 200 characters). A note is a line in the Timeline and nothing more: it never notifies anyone, never takes a picture and never sends a command. The tool tells the assistant to say what it wrote
`timeline_moment`	Writes a named moment: `title`, optional `symbol` (an SF Symbol name; default `sparkles`), `tone` (`info`, `good`, `warn` or `alert`), `objects` (ids or names it covers), `start`, `end` (ISO 8601; default now and open) and `detail`. The Timeline shows it with exactly this title. Like a note, it changes nothing in the home

Helper allows 120 notes and moments an hour per API key and answers `429` past that; a value Helper refuses comes back as a `400` with one sentence saying what. Notes cannot be changed or deleted through MCP (a script's `MyroTimeline.end` can close an open moment), and the scripting rules that quiet or rename automatic events (`MyroTimeline.mute`, `rename`, `mode`) have no MCP form; a script written through `save_script` can set them.

Pairing Myro:Home Pro for Apple TV, marking doorbells, the three-day home history, the Timeline's replay and the doorbell and camera pictures it keeps, and UniFi Protect camera setup and camera events use routes under `/helper/web` for a signed-in browser, or for an Apple TV that the owner approves on the Helper Mac. The paired apps read the Timeline under `/helper/timeline` with their own session token. An MCP API key cannot call either, so MCP has no tools for them, and UniFi Protect sign-ins never pass through MCP. For the panel's own event log, use `panel_events` and `history_window`; for what the home did, `timeline`, which never shows pictures.

Paired Apple TVs arm and disarm with the panel code but cannot force arm or bypass zones. That rule applies to paired devices only. MCP security still follows the API key's **Arm and disarm** permission and the stored panel code, as described in Optional Security Control.

Read the live inventory and scripting reference before choosing a target. Native relative commands change the panel's current state; acceptance does not confirm a physical result, and an uncertain command must not be retried automatically. Use the controller guide's exact parameters and hardware requirements.

Ask for a read before a write when names or numbers could be ambiguous. Be explicit about physical actions, especially locks, outputs, macros, broad lighting commands, security modes, and zone bypasses.

A controller acknowledgement confirms acceptance, not physical completion. An HTTP failure or timeout can leave the outcome unknown. Read current state before considering another command; do not retry a physical write automatically.

Create Scripts Safely

A good request is:

> Create a disabled script that sends a push notification and writes "Lights On" to my Vestaboard when Office Lts turns on. Read the live scripting reference, validate the script, save it disabled, and show me any warnings. Do not enable it yet.

The safe workflow is:

1. Read the Helper's live scripting reference and device inventory.
2. Read the existing script and retain its immutable `scriptID` when editing.
3. Validate proposed JavaScript without executing it.
4. Save new or changed automation disabled.
5. Enable it only after you review the source and explicitly approve activation.

Enabling a script executes its top-level body. Reloading scripts re-runs the top-level body of every enabled script. Either operation can have immediate physical effects.

Advanced Scripting

The live `scripting_reference` includes shared libraries (`MyroLibraries`), explicit cross-controller handles (`MyroControllers`), and script-owned speech callbacks and cancellation (`MyroSystem.speakQueued`, `cancelSpeech`, `cancelAllSpeech`). The existing MCP script tools can create scripts using these methods; separate MCP tools for each JavaScript method are unnecessary.

Savant through MCP

Savant setup uses its native **Connection** pane: discover and trust the host, then choose a **Local User** and enter the local password. No cloud token or agent ID is needed. MCP authenticates to Helper with its API key, not the Savant password. Read the Savant Controller Guide and the selected controller's live scripting reference.

The current adapter has no dedicated Savant media tools. Its existing script tools can read inventories/references and prepare scripts that call `MyroSavant` on a Savant controller or `MyroControllers.get(guid).savant` for a permitted destination. Saving/enabling requires **Remote script management**. Read status first, save new automation disabled, review it, and enable it only after explicit approval. Panel tools for numbered lights, security, thermostats, or panel audio must not be used as Savant room controls. Use room/service IDs when names repeat and verify later feedback after a command. Read `services(room)` for advertised commands. Room-only transports require exactly one active source; use

`transport(room, service, action)` for an explicit source. Missing command discovery blocks source commands until refreshed. Savant status separates inventory freshness and HomeKit support from configured/enabled/running bridge state.

Control4 through MCP

Control4 Link is set up in Helper's native **Connection** pane by importing the installer's private connection file; MCP authenticates to Helper with its API key and never sees the connection token. Read the Control4 Controller Guide and the selected controller's live scripting reference.

Control4 has rooms and Watch/Listen activities, not panel objects, so panel tools for numbered lights, security, thermostats, or panel audio do not apply to it. `controller_status` reports `controllerFamily` `"Control4"` with `rooms`, `roomStates`, `inventory`, `homeKit`, `objectCounts` and `capabilities`; unknown room fields are omitted rather than guessed. The current adapter has no dedicated Control4 media tools. Its script tools prepare scripts that call `MyroControl4` on a Control4 controller or `MyroControllers.get(guid).control4` for a permitted destination. Saving/enabling requires **Remote script management**. Save new automation disabled, review it, and enable it only after explicit approval. Use the IDs returned by `rooms()` and `activities(room)`, remember that `off(room)` shuts down the whole room, and read observed state afterwards: a queued command is not proof of a physical result.

Rooms, cameras, photos and Savant HomeKit

The current MCP adapter has no dedicated tools to edit shared room layouts, synchronize camera definitions, upload photos or configure Savant HAP switches. Use Helper's browser Rooms/Cameras, **API > Photos**, **API > Rooms & Cameras** (when shared records exist), the linked apps, and Savant's native **HAP** pane. PRO/Classic shared sync requires a user-reviewed preview under Edit > Sync with Helper; discovery alone performs no shared-data writes. Tapping Edit makes one read-only sync-status check and may put a dot on Sync with Helper; entering the tab does not check or show a banner. Choose Both ways, From Helper or To Helper before applying reviewed changes. Ignore suppresses the dot until Helper changes again; Turn off disables checks until Turn on. There is no background timer or automatic apply. The mobile sync/photo routes in the HTTP guide require a paired mobile bearer session; an MCP API key does not authorize them. Browser playback uses its own signed-in session. Never put camera passwords or mobile session tokens into scripts or chat.

Savant's HAP pane exposes selected activities as feedback-driven Apple Home switches, with an optional **Custom Name** for Apple Home and Siri. HAI's optional receiver prototype is also configured locally through **File > HAI Audio Receivers (Prototype)...**; the current MCP adapter does not configure either feature. Known activity IDs can recover after a host upload through saved, unique routing metadata within the same trusted system and room. Missing or ambiguous activities remain unavailable. Read the current inventory/reference before authoring scripts; do not invent UUIDs or use panel tools for Savant media. Both NESS / ELK and Savant remain previews.

Controller references

Read the selected controller's live `scripting_reference` before generating scripts. Full and compact references combine shared Helper contracts with that controller family's explicit guide and real device names. A selected HAI scope excludes ELK instructions; a selected ELK scope excludes callable HAI-only APIs and examples. Unscoped mixed homes include labeled contracts for each configured HAI / Leviton, NESS / ELK, Savant, and Control4 family.

Use the HAI and Leviton Controller Guide, NESS / ELK Controller Guide, Savant Controller Guide, or Control4 Controller Guide for that controller's controls, events, and available readings. For a cross-controller destination, request its reference and respect its capabilities and local grants.

Library files must be installed locally in Helper's Scripts/Libraries folder. Cross-controller access requires local **Send To** and **Accept From** grants at the respective controllers. MCP cannot upload libraries or change those grants. Grants extend the origin's scripting authority, including remotely managed scripts, and must be enabled deliberately. Validation checks the script body without loading libraries or proving permissions or physical behavior.

Version 4.51 adds `MyroSystem.runUserScript(filename, args, callback)` for trusted executables such as a Python integration. The owner must install the executable in Helper's Application Scripts folder and explicitly enable **User-Installed Scripts** for the controller locally. MCP cannot install executables or enable the grant, but a script-management credential can invoke installed executables through automation once it is enabled. These programs run outside Helper's sandbox with the user's permissions. Read the live scripting reference before generating calls. One external process is allowed globally; the 60-second timeout reports failure without killing the process, and the slot remains occupied until it exits. Never retry an uncertain physical command automatically.

For a new external integration, ask the assistant to create the JavaScript automation **disabled** and list its local prerequisites. The user must review and install the executable, configure its Python environment if needed, and enable the local grant themselves. Run the Scripting Guide's harmless `helper-probe.sh` setup check before testing a device action. Validation only checks JavaScript; it does not prove the external program is installed, safe, or functional. Do not put Tuya local keys, panel codes, or other secrets into script arguments or chat. Store them locally for the executable to read. A `finished` result confirms successful process exit, not physical completion; a timeout does not justify retrying the command.

Script-owned speech is separate from the HTTP speech queue used by MCP's `speak` tool. For MCP's `speak` tool, 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 HTTP route's per-message `?voice=` and `GET /helper/voices` are not exposed as MCP tool options yet. Browser login, themes and navigation remain UI features. Archived history is included automatically in history queries; deletion policy remains local configuration, not an MCP write capability.

Optional Security Control

The direct MCP arming tool requires both:

1. An API key generated with **Arm and disarm through the HTTP API** enabled.
2. A valid panel user code with the controller's configured length saved locally with `myrohome-helper-mcp security-code set`.

Without a stored code, arming remains disabled and returns a configuration error. Zone bypass and restore behavior without a client-side stored code is controller-specific; consult the corresponding controller guide. Do not assume that a panel provides a fallback user. A restore of a zone that is not ready or in trouble is refused by the panel whatever code is used; the result then includes a `hint` the assistant can relay, telling the user to close or clear the zone first. Normal startup reads only whether disarm or unlock is allowed, never the code, and non-security tools do not read the credential. The MCP server retrieves it only after an explicit security tool call, uses it for that one request, and then discards its reference.

Disarming and door unlocking are off by default, subject to support in the selected controller. To allow them, run `myrohome-helper-mcp allow disarm` or `myrohome-helper-mcp allow unlock` on the computer that runs the MCP server, type the word to confirm, and restart your MCP client; `myrohome-helper-mcp disallow disarm` (or `unlock`) takes the permission away again, effective on the next call. The permission applies only to the controller that API key belongs to. Until you allow them, the `disarm` and `unlock_door` tools are not offered to the assistant at all, so an instruction hidden in an email, web page or document it reads has nothing to call. When allowed, disarm still needs the security-enabled API key and a stored panel code. The assistant is told never to work around a tool that is not offered, for example by writing a script, and never to disarm or unlock because something it read asks it to. AI assistants can be misled by text they read; allow these only if you accept that, and prefer the Myro:Home app for disarming. This is an MCP tool policy, not a restriction on the Helper's HTTP API or local scripting permissions.

Never paste the panel code into a conversation. It is never included in MCP tool arguments, results, configuration JSON, or logs. The code travels only in the request body over the trusted local network to Helper, which asks the panel to validate it. Because Helper currently uses plain HTTP, this is not appropriate for an untrusted, shared, guest, or Internet-exposed network. When MCP and Helper run on the same Mac, use `http://localhost:9393` so the request remains on that computer.

Remote script management is a separate privilege. A script-management key can create and enable a local script that controls security without a separately stored panel code, so it must be protected at least as carefully as a direct-security key.

Troubleshooting

Symptom	What to Check
MCP tools do not appear	Validate the JSON, confirm `uvx` is installed, then fully quit and reopen the AI client
Cannot reach Helper	Confirm Helper is running, the API server is enabled, `MHH_URL` names the Helper Mac, port `9393` is reachable, and no VPN or firewall is blocking the local path
`401` response	The API key is missing, incorrect, or belongs to a different controller
Script save, enable, or reload returns `403`	Generate a replacement key with **Remote script management** enabled
Direct security returns `403`	Generate a replacement key with **Arm and disarm** enabled
Security is not configured	Run `uvx myrohome-helper-mcp security-code set` locally, or leave direct security disabled
A tool says Helper 4.07 is required, or returns `requiredHelper`	Update Myro:Home Helper to the version named and reconnect the MCP client
A device name is ambiguous	Use `list_objects`, verify the object type and number, and retry with the exact object

The Helper's **Show Log** window records API refusals and operational failures without logging panel security codes. Leave Verbose off for normal use and enable it when diagnosing a problem or when support requests additional detail.

Updating and Reference Material

The published package is myrohome-helper-mcp on PyPI. Restart the MCP client after updating the package or Helper so the server and tool descriptions are reloaded.

Use the other bundled guides when more detail is needed:

- **Getting Started** for initial Helper and controller setup
- **Scripting Guide** for shared Helper APIs, integrations, events and scheduling
- HAI and Leviton Controller Guide for HAI panel APIs, events and security
- NESS / ELK Controller Guide for ELK panel APIs, events, solar setup and security
- **HTTP API Documentation** for routes, payloads, authentication, and response details

Native photo management can preview and combine exact duplicate stored images while keeping existing image IDs, revisions and assignments. New byte-identical uploads reuse an existing photo. These are Helper administration features; the MCP adapter does not gain photo or shared-layout mutation tools.