Finite Labs **Essentials**

Overview

> DISCLAIMER: This software is neither affiliated with nor endorsed by Control4.

Network Requests sends HTTP, raw TCP, UDP, and Wake-on-LAN commands to devices and services that have no Control4 driver. Requests are defined once, by name, in a dedicated Composer tab, and fired from programming with a single Send Request command. Request URLs, bodies, headers, and payloads can reference any Control4 variable, so the same named request can carry live values from the project.

Index

- System Requirements
- Features
- Installer Setup
  - DriverCentral Cloud Setup
  - Adding the Driver
  - Requests Tab
  - Driver Properties
    - Cloud Settings
    - Driver Settings
    - Webhook Settings
  - Driver Actions
- Programming
  - Request Types
  - Referencing Variables
  - Commands
  - Events and the Response Variable
  - Examples
- Developer Information
- Support
- Changelog

System Requirements

- Control4 OS 3.3+

Features

- Named requests managed from a dedicated Requests tab in Composer, with a searchable variable browser, insert at cursor, and per-request send history
- Fire any request from programming by name with the Send Request command
- HTTP and HTTPS with any method (GET, POST, PUT, PATCH, DELETE, HEAD), arbitrary custom headers, request bodies, and basic auth via the URL
- Raw TCP and UDP payloads as text (with `\r`, `\n`, `\t`, `\xNN` escapes) or hex bytes
- TCP requests can wait for the device's response
- Wake-on-LAN magic packets by MAC address
- Insert live Control4 variable values into URLs, bodies, headers, and payloads with `PARAM{}` tokens
- Each request fires a `<Name> Sent` or `<Name> Failed` event after every send; HTTP and response-waiting TCP requests also publish the last response body to a `<Name> Response` variable
- Inbound webhooks: external systems (cameras, iPhone Shortcuts, Home Assistant) call a URL to fire a `<Name> Received` event, with the request body published to a `<Name> Payload` variable and an optional shared key
- Requests, webhooks, and last results persist across driver restarts

Installer Setup

DriverCentral Cloud Setup

> If you already have the DriverCentral Cloud driver installed in your project you can continue to Adding the Driver.

This driver relies on the DriverCentral Cloud driver to manage licensing and automatic updates. If you are new to using DriverCentral you can refer to their Cloud Driver documentation for setting it up.

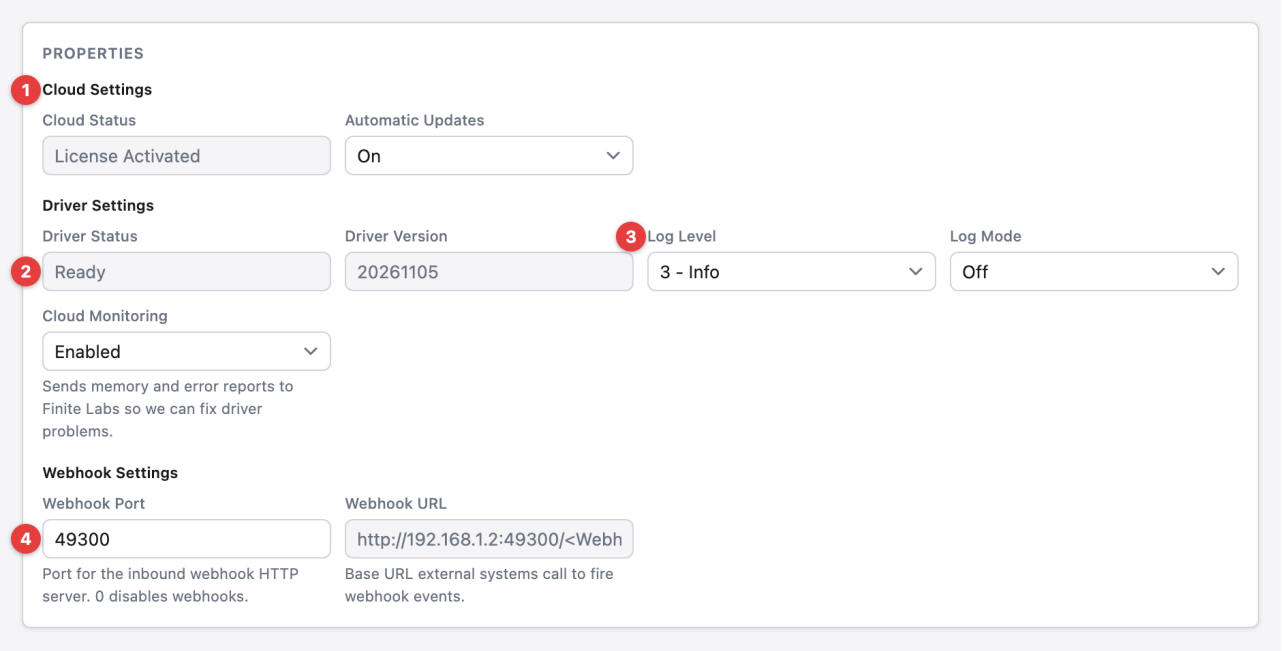
Adding the Driver

1. Download the latest `control4-finite-labs-essentials.zip` from DriverCentral.
2. Extract and install the `network_requests.c4z` driver.
3. Use the "Search" tab to find "Network Requests" and add it to your project.

Requests Tab

The Requests tab in Composer (select the driver, then the Requests tab) is the primary way to create and manage named requests.

Properties

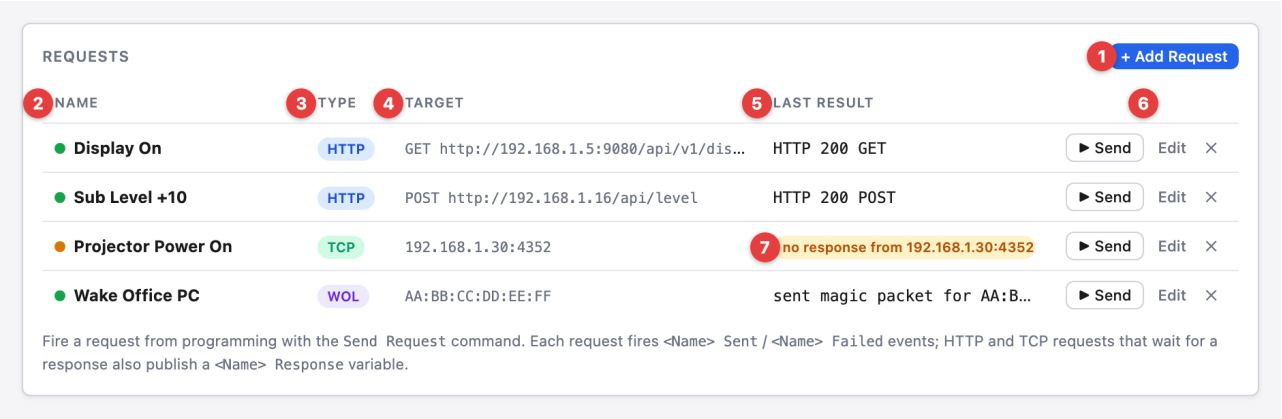


1. **Cloud Settings** - the driver's properties under the same headings as Composer's "Properties" tab.
2. **Driver Status** - grayed out, as every read-only property is: the driver sets it.
3. **Log Level** - a change applies at once, and shows in Composer's "Properties" tab too. A change made there shows here.
4. **Webhook Port** - applies when you leave the field. A value the driver can't use is refused, with the reason under the field, and the port stays as it was.

The Support Code is entered in Composer's "Properties" tab only.

Request List

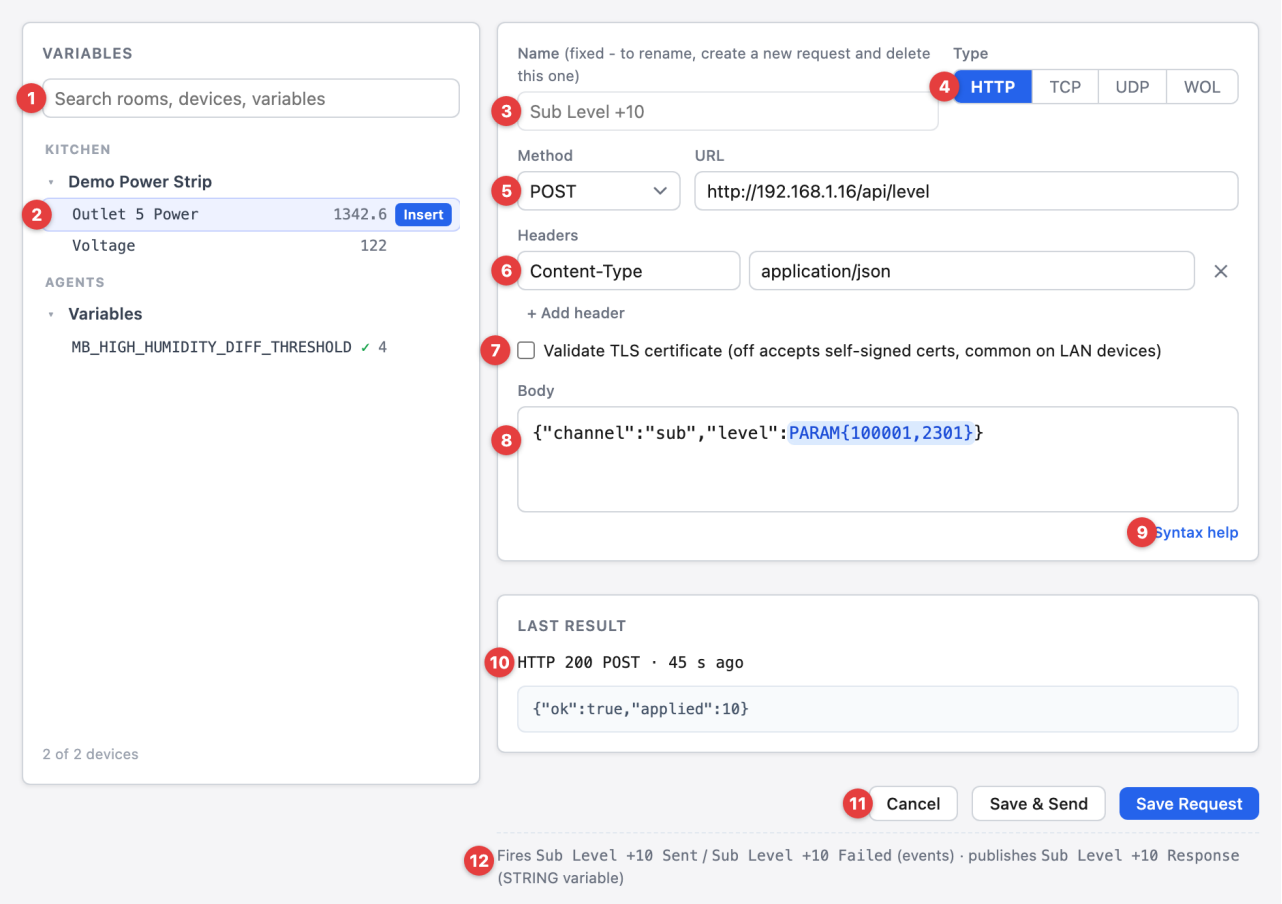
Every named request appears as a row with its type, target, and the result of its last send. Results update live as requests fire, whether from programming or the Send button on the row.



1. **+ Add Request** - opens an empty editor for a new request.
2. **Name** - the name programming uses to fire the request. The dot to its left shows the last outcome: green for a successful send, amber for a failure, and grey when the request has not been sent yet.
3. **Type** - HTTP, TCP, UDP, or WOL.
4. **Target** - where the request goes: method and URL for HTTP, `host:port` for TCP and UDP, and the MAC address for WOL.
5. **Last Result** - the outcome of the most recent send. Hovering a successful send shows the captured response and how long ago it was sent.
6. **Send / Edit / X** - send the request immediately, open it in the editor, or delete it along with its events and Response variable.
7. **Failed send** - failures show the error in place of the result, for example a refused connection, a timeout, or an unresolved `PARAM{}` reference.

Request Editor

Add Request and Edit both open the editor in place of the list. The fields below the Type selector are the ones the selected type needs; the screenshot shows an HTTP request.

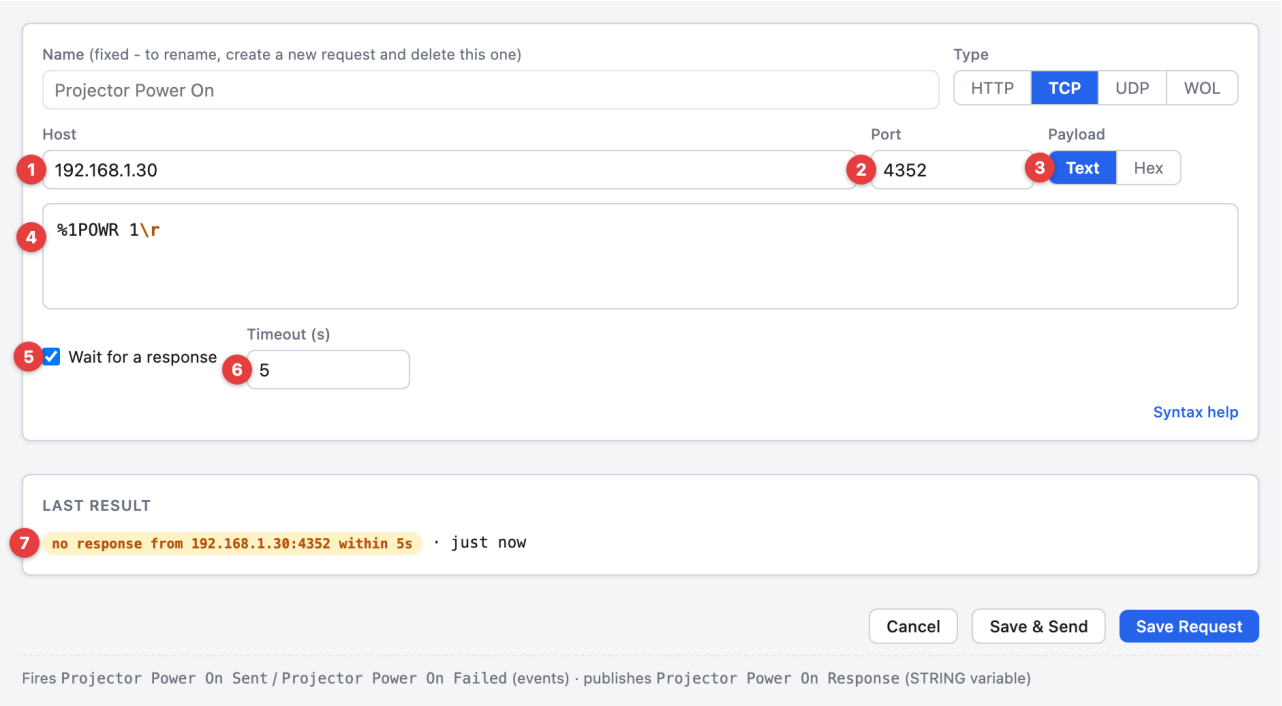


1. **Search** - filters the variable browser by room, device, agent, or variable name. Matching devices expand automatically.
2. **Variable** - click a device or agent to load its variables with their current values, then click a variable to insert its `PARAM{}` token into the field you last edited. A green check marks variables the request already references.
3. **Name** - the name used by programming, by the events, and by the Response variable. It is fixed once the request has been saved: to rename, create a new request and delete the old one. Names must be unique.
4. **Type** - switches between HTTP, TCP, UDP, and WOL.
5. **Method / URL** - the HTTP method and the destination. For basic auth, put the user name and password in the URL ahead of the host.
6. **Headers** - optional request headers. **+ Add header** adds a row and the X removes one; rows left without a name are dropped when saving.
7. **Validate TLS certificate** - off by default, so self-signed HTTPS endpoints on the LAN work without extra setup. Turn it on when the certificate matters.
8. **Body** - the request body. `PARAM{}` tokens are highlighted and replaced with the variables' current values every time the request is sent.
9. **Syntax help** - a reference for tokens, text and hex payloads, and the HTTP specifics, without leaving the editor.

10. **Last Result** - the outcome of the last send, with the captured response body underneath it when there was one.
11. **Cancel / Save & Send / Save Request** - discard the changes, save and fire the request immediately (the result lands in Last Result), or just save it.
12. **Publish line** - the events this request owns once it is saved, plus the Response variable for HTTP and response-waiting TCP requests.

TCP, UDP, and WOL Requests

Selecting TCP or UDP replaces the HTTP fields with a raw payload form. The variables browser works the same way and is left out of the screenshot below.



1. **Host** - the address to connect to.
2. **Port** - the TCP or UDP port.
3. **Text / Hex** - how the payload is read. Text decodes the escapes `\r`, `\n`, `\t`, `\0`, `\\`, and `\xNN`, and substitutes `PARAM{}` tokens. Hex sends raw bytes: whitespace, commas, and `0x` prefixes are ignored, so `02 41 54 0d`, `0x02,0x41,0x54,0x0d`, and `0241540d` all send the same four bytes. Tokens are not substituted in hex payloads.
4. **Payload** - the bytes to send. Escapes and tokens are highlighted.
5. **Wait for a response** - TCP only. The driver holds the connection open until the device answers and publishes the answer to the Response variable. UDP is always fire and forget.
6. **Timeout (s)** - how long to wait for the connection, and then for that response, before the send fails.
7. **Last Result** - here, a send that timed out, which fired the `<Name> Failed` event.

WOL requests take a single **MAC address** field and broadcast a Wake-on-LAN magic packet to it.

Outputs

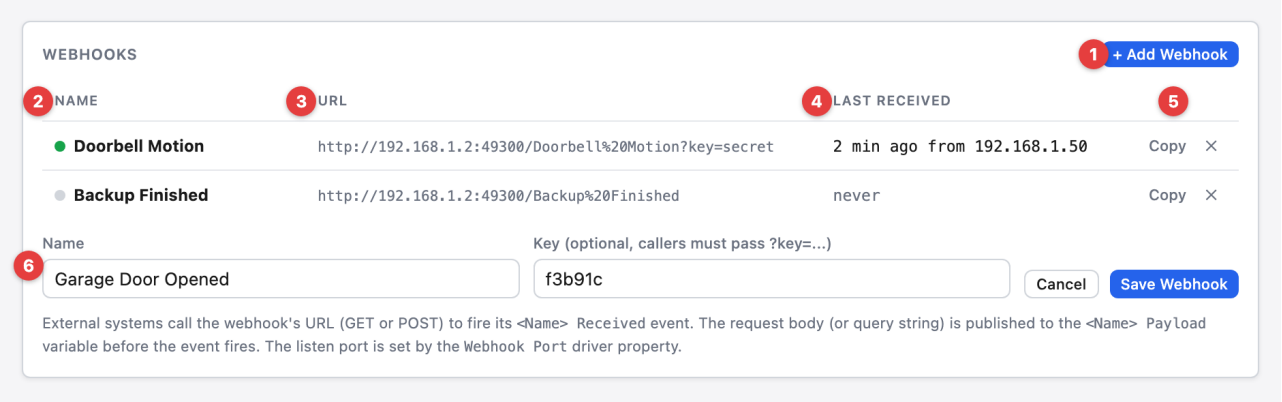
Saving a request named `Projector Power On` creates:

- An event `Projector Power On Sent` that fires after every successful send
- An event `Projector Power On Failed` that fires when a send fails
- For HTTP and response-waiting TCP requests, a read-only variable `Projector Power On Response` holding the last response body

Request names are permanent: programming, events, and the Response variable all reference the request by name, so the editor does not allow renaming. To rename, create a new request and delete the old one, then update any programming that referenced it. Names must be unique - saving a new request with a name that is already taken is rejected. Deleting a request removes its events and variable.

Webhooks

The Webhooks card manages inbound HTTP endpoints: an external system calls a URL and the driver fires an event.



1. **+ Add Webhook** - reveals the form at the bottom of the card.
2. **Name** - names the webhook's event and payload variable, and forms the last part of its URL. The dot shows whether it has ever been called.
3. **URL** - the address external systems call, built from the controller's address and the `Webhook Port` property. It answers GET and POST.
4. **Last Received** - when the webhook was last called, and by whom. Hovering shows the payload that arrived.
5. **Copy / X** - put the full URL, key included, on the clipboard, or delete the webhook along with its event and payload variable.

6. **Name / Key** - the new webhook's name and its optional key. With a key set, callers must append `?key=<value>`; calls with a missing or wrong key are rejected with HTTP 403 and fire nothing.

Adding a webhook named `Doorbell Motion` creates:

- A URL like `http://<controller>:<port>/Doorbell%20Motion`, answering GET and POST
- An event `Doorbell Motion Received` that fires on every accepted call
- A read-only variable `Doorbell Motion Payload` holding the request body (or query string), updated before the event fires

Driver Properties

Cloud Settings

Cloud Status (read-only)

Displays the DriverCentral cloud license status.

Automatic Updates [Off | *On*]

Enables or disables automatic driver updates via DriverCentral. Changing it on any item of this driver changes it on all of them.

Driver Settings

Driver Status (read-only)

Displays the current status of the driver.

Driver Version (read-only)

Displays the current version of the driver.

Log Level [0 - Fatal | 1 - Error | 2 - Warning | *3 - Info* | 4 - Debug]

Sets the logging level. Default is `3 - Info`. While a Support Code has Composer's Lua tab open, it also offers `5 - Trace` and `6 - Ultra`.

Log Mode [*Off* | Print | Log | Print and Log]

Sets the logging mode. Default is `Off`.

Support Code

Leave it blank unless Finite Labs support gives you a code: for an hour, it opens Composer's Lua tab to this driver and adds `5 - Trace` and `6 - Ultra` to Log Level.

Cloud Monitoring [Disabled | *Enabled*]

When Enabled, the driver sends Finite Labs a short report once an hour, and a report of each error it hits, at most 10 a minute. We use them to find and fix problems in the driver. Set it to Disabled to send nothing.

A report holds the controller's MAC address and OS version, the driver's name, version, device ID and DriverCentral license status, how much memory the driver uses, and the sizes of its largest internal tables. An error report also holds the error message and where in the driver it happened. Before a message is sent, anything in it that looks like a password, token, key, network address or email address is removed.

Webhook Settings

Webhook Port [0 - 65535]

Sets the port the webhook server listens on. Default is `49300`. `0` disables webhooks.

Webhook URL (read-only)

Displays the base URL external systems call to fire webhook events, `Disabled` when the port is `0`, or the reason the server can't listen on the port.

Driver Actions

Reset Driver

Deletes every request along with its events and response variable, and clears all persisted state.

It also turns Cloud Monitoring back on, as a fresh install has it.

Parameters:

- **Are You Sure?** [***No*** | Yes] - Confirmation to reset the driver.

Diagnostics Snapshot

Prints a report in the Lua output, and sends it to Finite Labs only while Cloud Monitoring is Enabled.

Programming

Request Types

- **HTTP**: any method, HTTPS supported, custom headers, request body, and basic auth encoded in the URL. The response body is captured. TLS certificate validation is off by default (opt in per request), so self-signed LAN endpoints work without extra setup.
- **TCP**: opens a connection, sends the payload, and either disconnects immediately or waits for the response before disconnecting. The request's timeout bounds both the connection and the wait.
- **UDP**: sends the payload as a single datagram, fire and forget. Broadcast addresses are not supported.
- **WOL**: broadcasts a standard Wake-on-LAN magic packet for the configured MAC address. The target device must have Wake-on-LAN enabled.

Referencing Variables

Reference a variable anywhere in a URL, body, header value, or text payload with a `PARAM{}` token:

```
PARAM{DEVICE_ID,VARIABLE_ID}
```

`DEVICE_ID` is the numeric id of the device that owns the variable. `VARIABLE_ID` is either the numeric id of the variable or its name (matched exactly, case-sensitive), for example `PARAM{96,1002}` or `PARAM{96,Humidity}`. The token is replaced with the variable's current value each time the request is sent.

You rarely need to build a token by hand: click a variable in the Requests tab browser and its token is inserted into the focused field. If a referenced variable cannot be found when the request fires, the send fails and the `<Name> Failed` event fires.

Tokens are not substituted in hex payloads.

Commands

Send Request

Send a named request now. The Request dropdown lists every request defined in the Requests tab. Sends are asynchronous: programming continues immediately and the `<Name> Sent` / `<Name> Failed` event fires when the send completes.

Events and the Response Variable

Each request owns a `<Name> Sent` event, a `<Name> Failed` event, and, for HTTP and response-waiting TCP requests, a read-only `<Name> Response` variable.

- `<Name> Sent` fires after every successful send. For HTTP this means the server responded (any status code, including errors like 404 - check the response variable when the status matters).
- `<Name> Failed` fires when the send itself fails: connection refused, host unreachable, a TCP connection or response that doesn't come within the timeout, or an unresolved `PARAM{}` reference.
- `<Name> Response` updates before the `Sent` event fires, so programming triggered by the event can read the fresh response.

Responses are capped at 8 KB.

Webhook Events

Each webhook owns a `<Name> Received` event and a read-only `<Name> Payload` variable. When an external system calls the webhook's URL:

1. The request body (or, for calls without a body, the query string) is published to `<Name> Payload`, capped at 8 KB.
2. The `<Name> Received` event fires.
3. The caller gets a `200 {"ok":true}` JSON response.

Program off the `<Name> Received` event; read `<Name> Payload` for the data. Pair with the Variable Expressions driver to parse values out of a JSON or query-string payload.

Webhooks are unauthenticated LAN endpoints unless a key is set. For anything that triggers meaningful automation, set a key and treat the full URL as a secret.

Examples

Turn on a display (HTTP GET):

1. In the Requests tab, add an HTTP request named `Display On` with URL `http://192.168.1.5:9080/api/v1/display/on`.
2. In programming, fire it with `Send Request` -> `Display On`.

Set a subwoofer level from a variable (HTTP POST with a JSON body):

```
URL:  http://192.168.1.16/api/level
Body: {"channel":"sub","level":PARAM{100001,2500}}
```

PJLink projector power on (TCP with a response):

```
Host: 192.168.1.30   Port: 4352
Payload: %1POWR 1\r
Wait for a response: enabled
```

The projector's `OK` lands in `Projector Power On Response`.

Wake a computer: add a WOL request with the computer's MAC address and fire it from any scene or schedule.

Developer Information



Copyright © 2026 Finite Labs LLC

All information contained herein is, and remains the property of Finite Labs LLC and its suppliers, if any. The intellectual and technical concepts contained herein are proprietary to Finite Labs LLC and its suppliers and may be covered by U.S. and Foreign Patents, patents in process, and are protected by trade secret or copyright law. Dissemination of this information or reproduction of this material is strictly forbidden unless prior written permission is obtained from Finite Labs LLC. For the latest information, please visit https://drivercentral.io/platforms/control4-drivers/utility/finite-labs-essentials

Support

If you have any questions or issues integrating these drivers with Control4, you can contact us at driver-support@finitelabs.com or call/text us at +1 (949) 371-5805.

Changelog

v20261112 - 2026-11-12

Added

- Added the Device Programmer, Light Relay, Network Requests, Sensor Aggregator, Sensor Multiplexer and Variable Expressions drivers to DriverCentral.