

Fixture HTTP API

A fixture is a group of consecutive DMX channels driven as one lamp. This document covers the `/api/fixtures` endpoints only; the rest of the HTTP API and the line based control protocol are in [README.md](#).

Contents

- [Before anything else](#)
- [What a fixture is](#)
- [Endpoints](#)
- [Setting values](#)
- [Errors](#)
- [Worked examples](#)

Before anything else

Authentication. Every endpoint needs HTTP Basic authentication with the system password. The user name is ignored, so `-u :password` is enough.

Content type. Requests with a body must send `Content-Type: application/json`.

8 bit DMX only. Fixtures assume one channel per DMX unit. In either 16 bit mode the two GET endpoints keep working so a patch can still be inspected, and everything that changes a fixture is answered with `400 fixtures need 8 bit dmx mode`. Nothing is lost by switching: fixtures stay patched and stored and work again as soon as the bus is back in 8 bit. Check `bitmode` in `GET /api/info` before offering an edit.

Success is `{"success": true}`, plus `"id"` when a fixture was created. **Failure** is a status code and a plain text body - not JSON.

What a fixture is

Types and footprints

Type	Channels	Order on the bus
<code>dim</code>	1	dimmer
<code>ww</code>	2	cold white, warm white
<code>rgb</code>	3	red, green, blue
<code>rgbw</code>	4	red, green, blue, dimmer
<code>rgbww</code>	5	red, green, blue, cold white, warm white

These five are the whole set. A fixture's type is fixed at creation - to change it, delete the fixture and create a new one.

Identity and address

`id` is assigned by the device at creation, is the lowest free one, and is what every endpoint addresses. Up to 512 fixtures can exist at once.

`address` is the 1 based DMX start channel. A fixture occupies `address` through `address + footprint - 1`, and those channels are reserved: a create or move that would overlap an existing fixture, or run past channel 512, is refused and nothing changes.

`patched` says whether the fixture holds channels. An unpatched fixture still exists and keeps its name and settings, but reports `address 0`, has no `channels` array, and refuses every value.

Name

Optional, and `""` means unnamed. At most **15 characters** of printable ASCII (0x20-0x7E). A comma, a double quote and a backslash are refused.

Brightness

A per fixture master, **1 to 100 percent**, that scales every channel of that fixture on its way to the bus. Setting a colour to full red at brightness 50 puts the red channel at 50 percent. It is not a channel of its own, it is a multiplier applied to all of them.

Two consequences worth knowing:

- **0 is refused.** The range starts at 1.
- **It is stored,** in NVS, and survives a reboot - unlike channel levels, which are whatever the bus is doing.

The `channels` array in `GET /api/fixtures/{id}` is reported with brightness divided back out, so it reads as the colour you asked for rather than the scaled level actually on the wire.

Temperature limits

`low_limit` and `high_limit` are kelvin and apply to `ww` and `rgbww` only. They default to **2000** and **6500**, must each be 1000-8000, and the low one must be strictly below the high one. They bound what `temperature` accepts, and `temperature` divides by their difference - which is why equal limits are refused rather than clamped.

Endpoints

Method	Path	Purpose
GET	<code>/api/fixtures</code>	List every fixture
POST	<code>/api/fixtures</code>	Create and patch one
GET	<code>/api/fixtures/{id}</code>	Read one, with live levels
PUT	<code>/api/fixtures/{id}</code>	Move and/or rename
DELETE	<code>/api/fixtures/{id}</code>	Remove and free its channels
PUT	<code>/api/fixtures/{id}/values</code>	Drive the light

GET /api/fixtures

Every fixture, ordered by id.

```
{ "fixtures": [
  { "id": 0, "type": "rgb", "footprint": 3, "patched": true, "address": 10,
    "brightness": 100, "name": "RGB Kitchen" },
  { "id": 1, "type": "ww", "footprint": 2, "patched": true, "address": 13,
    "brightness": 80, "name": "", "low_limit": 2700, "high_limit": 5000 } ] }
```

`low_limit` and `high_limit` appear for `ww` and `rgbww` only. `name` is "" when the fixture has none.

This response deliberately carries **no channel levels**: levels change as fast as the bus does, and this is the endpoint clients poll. Use the single fixture endpoint for levels.

A full universe of 512 fixtures is roughly 55 kB of JSON.

POST /api/fixtures

Creates a fixture and patches it in one step. Returns its new id.

Field	Required	Range	Default
<code>type</code>	yes	one of the five types	-
<code>address</code>	yes	1-512	-
<code>name</code>	no	see Name	""
<code>low_limit</code>	no	1000-8000, <code>ww</code> / <code>rgbww</code>	2000
<code>high_limit</code>	no	1000-8000, <code>ww</code> / <code>rgbww</code>	6500

```
curl -u :password -H "Content-Type: application/json" \
-X POST http://dmx/api/fixtures \
-d '{"type": "rgb", "address": 10, "name": "RGB Kitchen"}'
```

```
{ "success": true, "id": 0 }
```

If the address is taken or the fixture does not fit, nothing is created - the device rolls the half made fixture back out of the list itself.

GET /api/fixtures/{id}

One fixture, in the same shape the list uses plus a **channels** array: the current level of each of its channels in percent, in patch order.

```
{ "id": 0, "type": "rgbww", "footprint": 5, "patched": true, "address": 10,
  "brightness": 100, "name": "Living Room", "low_limit": 2700,
  "high_limit": 5000, "channels": [100, 40, 0, 50, 50] }
```

Levels written by anything else - Art-Net, a **setlevel** on those channels - are reported here too, because the bus is the single source of truth. They get the same brightness division as everything else, and the array cannot express more than full: at brightness 50 a channel forced past 128 on the wire still reads 100. Such a channel is clamped to that reported colour by the next brightness change, so what it does from then on follows the array, not the level it was forced to.

An unpatched fixture holds no channels and leaves the array out entirely rather than sending zeroes that are not true.

PUT /api/fixtures/{id}

Moves a fixture, renames it, or both.

```
{ "address": 20 }
{ "name": "RGB Kitchen" }
{ "address": 20, "name": "RGB Kitchen" }
```

Both fields are checked before either is applied. A rename on its own needs no **address**; send **"name": ""** to clear the name. A move that is refused leaves the fixture on its old address.

The type cannot be changed here.

DELETE /api/fixtures/{id}

Removes the fixture and frees its channels.

The channels are **not** blacked out - nothing writes them down, so they stay at the level they were on until something else drives them. Patch another fixture over them and it inherits those levels.

PUT /api/fixtures/{id}/values

See [Setting values](#).

Setting values

One endpoint drives the light. Every field is optional and any combination can be sent in a single request.

Field	Type	Range	Applies to
brightness	number	1-100	every type
rate	number	0-65535 ms	every type
off	boolean	true acts, false is a no-op	every type
on	boolean	true acts, false is a no-op	every type
dimmer	number	0-100	dim, rgbw
color	object	r, g, b each 0-100	rgb, rgbw, rgbww
hsv	object	h 0-360, s 0-100, v 0-100	rgb, rgbw, rgbww

Field	Type	Range	Applies to
white	object	wc , ww each 0-100	ww , rgbww
temperature	number	the fixture's own limits, in kelvin	ww , rgbww

Everything is a percentage, not a DMX level. color is 0-100 per component, not 0-255.

rate is a fade time in milliseconds

rate is how long the move takes, in **ms**, not a level per second. It applies to every field in the same request. Omit it, or send 0, and the change is instant.

rate on its own does nothing - it is a modifier on the other fields, not a command.

on and off

off sends every channel of the fixture to 0, **on** sends them to 100. They act only when **true**; sending **false** does nothing. Both honour **rate**, and both ignore brightness - **on** is every channel at full, whatever the fixture is scaled to.

Order matters

Fields are applied in this order: **brightness**, **off**, **on**, **dimmer**, **color**, **hsv**, **white**, **temperature**.

Brightness goes first on purpose, so a request carrying both brightness and a colour ends at that colour scaled by the *new* brightness. Sending **color** and **hsv** together is allowed but pointless - **hsv** is applied second and wins.

Nothing is applied unless all of it can be

The whole request is validated against the fixture's type and its kelvin limits before any of it is written. A request that is refused changes nothing - which matters because brightness persists to NVS, so a half applied request would otherwise leave a stored change behind while telling the caller it failed.

The fixture has to be patched

An unpatched fixture has no channels on the bus, so every value is answered with 400 **fixture does not exist or is not patched**.

Errors

The body of a failure is plain text, not JSON.

Status	Body	Means
401	-	Missing or wrong password
400	fixtures need 8 bit dmx mode	Bus is in a 16 bit mode
400	unsupported type	type is not one of the five
400	invalid address	address missing or outside 1-512
400	invalid limits	A limit is not a 16 bit number
400	create failed, check limits	Limits out of range, low not below high, or 512 fixtures exist
400	address taken or fixture does not fit	Overlaps another fixture or runs past 512
400	invalid name	Too long, or a banned character
400	rename failed	The device refused a name it had accepted
400	fixture does not exist or is not patched	Values sent to a fixture with no channels
400	<i>the name of a field</i>	Out of range, or not something this type has
404	no such fixture	No fixture with that id
404	no such resource	Unknown path under a fixture

A rejected value field answers with **just the field name** - temperature, color, brightness - so a client can point at the control that was wrong.

Worked examples

Create an RGB fixture at channel 10, named:

```
curl -u :password -H "Content-Type: application/json" \
  -X POST http://dmx/api/fixtures \
  -d '{"type":"rgb","address":10,"name":"RGB Kitchen"}'
# {"success": true, "id": 0}
```

Fade it to red over two seconds at half brightness:

```
curl -u :password -H "Content-Type: application/json" \
  -X PUT http://dmx/api/fixtures/0/values \
  -d '{"brightness":50,"color":{"r":100,"g":0,"b":0},"rate":2000}'
```

Read back what the bus is doing:

```
curl -u :password http://dmx/api/fixtures/0
# {"id":0,...,"channels":[100,0,0]}
```

The channels read 100/0/0, not 50/0/0: brightness is divided back out.

A tunable white fixture with its own limits, warmed over one second:

```
curl -u :password -H "Content-Type: application/json" \
  -X POST http://dmx/api/fixtures \
  -d '{"type":"ww","address":20,"name":"Bedside Lamp",
      "low_limit":2200,"high_limit":5000}'

curl -u :password -H "Content-Type: application/json" \
  -X PUT http://dmx/api/fixtures/1/values \
  -d '{"temperature":2700,"rate":1000}'
```

Asking for 6000 K on that fixture is refused, because it is outside its own limits:

```
curl -u :password -i -H "Content-Type: application/json" \
  -X PUT http://dmx/api/fixtures/1/values -d '{"temperature":6000}'
# HTTP/1.1 400 Bad Request
# temperature
```

Blackout, move and rename, then remove:

```
curl -u :password -H "Content-Type: application/json" \
  -X PUT http://dmx/api/fixtures/0/values -d '{"off":true,"rate":500}'

curl -u :password -H "Content-Type: application/json" \
  -X PUT http://dmx/api/fixtures/0 -d '{"address":40,"name":"Hall Ceiling"}'

curl -u :password -X DELETE http://dmx/api/fixtures/0
```