Smart LED Christmas lights Documentation

Release 2.4.21.2

Pavol Babinčák

Dec 22, 2020

Contents

Contents:

CHAPTER 1

Smart LED Christmas lights

Unofficial documentation of *API reference*, protocol details and hardware of Twinkly - Smart Decoration LED lights for Christmas.

Official materials says:

> Twinkly is a LED light device that you can control via smartphone. It allows you to play with colouful and animated effects, or create new ones. Decoration lights, not suitable for household illumination.

Since its Kickstarter project in 2016 many products were introduced with varying properties and features. Most notably products released since September 2019 are identified as Generation II. Older products are since then referred as Generation I. Documentation has been created and tested only on some *hardware*.

Products could be further groupped by families for which firmware is released. Firmware on devices can be upgraded and sometimes new features are introduced. Documentation has been created for some of these *firmwares*.

1.1 Why?

My first Twinkly was 105 LEDs starter light set. That was the latest available model in 2017: TW105S-EU. As of December 2017 there are only two ways to control lights: mobile app on Android or iOS or hardware button on the cord.

Android application didn't work as advertised on my Xiaomi Redmi 3S phone. On first start it connected and disconnected in very fast pace (like every 1-2 seconds) to the hardware. I wasn't able to control anything at all. Later I wanted to connect it to my local WiFi network. But popup dialog that shouldn't have appear never did so.

Public API was promised around Christmas 2016 for next season. Later update from October 2016 it seems API won't be available any time soon:

> API for external control are on our dev check list, we definitely need some feedback from the community to understand which could be a proper core set to start with.

It turned out that application uses HTTP to control lights. I ended up with capturing network traffic and documented this private API. In the end I'm able to configure the device pretty easilly.

As of 2020 Twinkly devices can be controlled by Amazon Alexa and Google Assistant as well. Mobile application now requires an account to operate lights even locally. No sign of public API for local devices though. Therefore with my second device - Twinkly 210 RGB+W Wall I keep updating this documentation to be able to operate my devices locally and not rely on availability of manufacturer’s servers.

1.2 License

Documentation is available under MIT license.

CHAPTER 2

Guides

2.1 Hardware

There are wide variety of Twinkly products on the market with various functionality and properties. This page desribes hardware that this documentation has been tested with.

2.1.1 Generation I model TW105S-EU

It is a string of 105 RGB LED from 2017. Hardware consists of two circuit boards:

- Module ESP-01 with microcontroller ESP8266 by Espressif Systems.
- Custom-made LED driver module

API exposes these details:

- Product code: TW105SEUP06
- Product version: 2
- Hardware version: 6
- Flash size: 16
- LED Type: 6
- LED Version: 1
- Max Supported LEDs: 255
- Movie Capacity: 719
- Frame Rate: 25

Firmware belongs to family “D”. In this documentation the device is referred as:

- Hardware ID: 0033aaff
- MAC address 5c:cf:7f:33:aa:ff

2.1.2 Generation II model TWW210SPP-TEU

It is branded as Twinkly Wall and consists of 210 RGB+W LEDs. It is a matrix of 10 strings each consisting of 21 red, green, blue and white LEDs. Most likely the device is built on the top of ESP32 microcontroller by Espressif Systems.

API exposes these details:

- Product code: TWW210SPP
- Hardware version: 100
- Flash size: 64
- LED Type: 12
- Firmware family: G
- Bytes per LED: 4
- Max Supported LEDs: 1200
- Frame Rate: 11
- Movie Capacity: 992
- Wire type: 1

On the top of that device has Bluetooth for out of the band configuration of WiFi. In this documentation the device is referred as:

- Hardware ID: 1bb210
- MAC address 98:f4:ab:1b:b2:10

2.2 Firmware

This page desribes firmware that this documentation has been tested with. Firmware can be upgraded over the network.

2.2.1 Firmware family "D"

Firmware consists of two files. I have seen following versions:

- 1.99.20
- 1.99.24
- 2.0.22 - adds MQTT support
- 2.1.0
- 2.3.5

2.2.2 Firmware family "G"

Firmware consists of one file. I have seen following versions:

- 2.4.21

2.3 Protocol details

This page describes hardware, modes of operation and some private protocols or algorithms used by Twinkly application.

2.3.1 Device name

Device name is used to announce SSID if it operates in AP mode, or to select device in the application. By default consists of prefix **Twinkly_** and uppercased unique identifier derived from MAC address. It can be read or changed by API.

2.3.2 Modes of network operation

Hardware works in two network modes:

- Access Point (AP)
- Station (STA)

AP mode is default - after factory reset. Broadcasts SSID made from *device name*. Server uses static IP address 192.168.4.1 and operates in network 192.168.4.0/24. Provides DHCP server for any device it joins the network.

To switch to STA mode hardware needs to be configured with SSID network to connect to and encrypted password. Rest is simple API call through TCP port 80 (HTTP).

Switch from STA mode back to AP mode is as easy as another API call.

http://41j.com/blog/2015/01/esp8266-access-mode-notes/

2.3.3 WiFi password encryption

1. Generate encryption key
   1. Use secret key: **supersecretkey!!**
   2. Get byte representation of MAC address of a server and repeat it to length of the secret key
   3. xor these two values
2. Encrypt
   1. Use password to access WiFi and pad it with zero bytes to length 64 bytes.
   2. Use rc4 to encrypt padded password with the *encryption key*
3. Encode

   Base64 encode encrypted string.

2.3.4 Discovery

This seems to be used to find all Twinkly devices on the network.

1. Application sends UDP broadcast to port 5555 with message **\x01discover** (first character is byte with hex representation 0x01).
2. Server responds back with following message:

- first four bytes are octets of IP address written in reverse - first byte is last octet of the IP adress, second second to last, . . .
- fifth and sixth byte forms string “OK”
- rest is string representing *device name* padded with zero byte.

2.3.5 Get and verify authentication token

Application uses TCP port 80 to get and verify authentication token. It is later used for some calls that require it.

1. Application generates challenge and sends it as part of login request.
2. Among other data server responds with authentication token
3. Application uses authentication_token in header of request to verify.

Only after this handshake authentication token can be used in other calls. Most of them require it.

2.3.6 Verification of challenge-response

As part of login process server sends not only authentication token but also challenge-response. Application may verify if it shares secret with server - maybe if it is genuine Twinkly device with following algorithm:

1. Generate encryption key
    1. Use secret key: **evenmoresecret!!**
    2. get byte representation of MAC address of a server and repeat it to length of the secret key
    3. xor these two values
2. Encrypt - use rc4 to encrypt challenge with the key
3. Generate hash digest - encrypted data with SHA1
4. Compare - hash digest must be same as challenge-response from server

2.3.7 Firmware update

Update sequence follows:

1. application sends first file to endpoint 0 over HTTP
2. server returns sha1sum of received file
3. application sends second file to endpoint 1 over HTTP
4. server returns sha1sum of received file
5. application calls update API with sha1sum of each stages.

2.3.8 LED effect operating modes

Hardware can operate in one of following modes:

- off - turns off lights
- demo - starts predefined sequence of effects that are changed after few seconds
- movie - plays last uploaded effect

- rt - receive effect in real time

First two are set just by API call.

2.3.9 Upload full movie LED effect

1. Application calls API to switch mode to movie
2. Application calls API movie/full with file sent as part of the request
3. Application calls config movie call with additional parameters of the movie

2.3.10 Movie file format

LED effect is called **movie**. It consists of **frames**. Each frame defines colour of each LED.

Movie file format is simple sequence of bytes. Three bytes in a row represent intensity of *red*, *green* and *blue* in this order. Each frame is defined just with number of LEDs times three. Frames don't have any separator. Definition of each frame starts from LED closer to LED driver/adapter.

2.3.11 Real time LED operating mode

1. Application calls HTTP API to switch mode to rt
2. Then UDP packets are sent to a port 7777 of device. *Each packet represents single frame* that is immediately displayed. See bellow for format of the packets.
3. After some time without any UDP packets device switches back to movie mode.

2.3.12 Real time LED UDP packet format

Before packets are sent to a device application needs to login and verify authentication token. See above.

Each UDP has header:

- 1 byte *\x01* (byte with hex representation 0x01)
- 8 bytes Base 64 decoded authentication token
- 1 byte number of LED definitions in the frame

Then follows body of the frame similarly to movie file format - three bytes for each LED.

For my 105 LED each packet is 325 bytes long.

2.3.13 Scan for WiFi networks

Hardware can be used to scan for available WiFi networks and return some information about them. I haven't seen this call done by the application so I guess it can be used to find available channels or so.

1. Call network scan API
2. Wait a little bit
3. Call network results API

CHAPTER 3

API reference

3.1 Messaging API reference

New in firmware version 2.0.22.

3.1.1 Overview

Device sends messages with its states to a broker with MQTT. By default broker is set to mqtt.twinkly.com.

Last topic levels are always client ID which are by default derived from MAC address of the device as uppercased hexadecimal digits.

3.1.2 Status from device

Device publishes these messages to a broker.

Topic *xled/status/* followed by client ID

Messages

- *online*
  - *ip* - IP address. Added in firmware 2.1.0.
  - *ssid* - SSID to which device is connected. Added in firmware 2.1.0.
- *offline*

Example

Online (firmware 2.0.22):

```
{"status": "online"}
```

Online (since firmware 2.1.0):

```
{"status": "online", "ip": "192.168.4.1", "ssid": "home"}
```

Offline:

```
{"status": "offline"}
```

3.1.3 Application status from device

Device publishes these messages to a broker.

Topic

xled/appstatus/ followed by client ID

Messages

- *off*
- *movie*
- *collision*
- *rainbow*
- *twinkle*
- *snake*

Example

Rainbow:

```
{"appstatus": "rainbow"}
```

3.1.4 Parameters

Parameters published by a device to broker.

Topic

xled/params/ followed by client ID

Messages

brightness (number), brightness value in percent

filters (list), contains a string "brightness" if brightness is set.

Example

```
{"brightness": 50, "filters": ["brightness"]}
```

3.1.5 Command messages to device

Device listens to these messages.

Topic

xled/command/ followed by client ID

Messages

changeeffect each message switches to next of default effects or movie. Same as pressing the button on the device.

setmovie sets to uploaded movie effect

setcollision sets default effect "collision"

setrainbow sets default effect "rainbow"

setsnake sets default effect "snake"

settwinkle sets default effect "twinkle"

setwaves sets default effect "waves"

switchoff switches off the device

switchon switches on the device

setbrightnessdelta changes brightness by amount set by *value*. Parameter *value* is signed number.

setbrightness changes brightness to *value*. Parameter *value* is a number between 0..255 but only up to 100 has a dimming effect.

Example

Switch on:

```
{"command": "switchon"}
```

Lower brightness value by 20 percents:

```
{"command": "setbrightnessdelta", "value", -20}
```

3.2 RESTful API reference

3.2.1 Overview

Twinkly rest API is primary way to get information about the device, configure network and modes of the device. It is a HTTP 1.1 based API sent over TCP port 80.

This API is used by mobile applications. It haven't been made public yet so it may change at any time.

Request Authentication

Most API requests require valid authentication token. Except of:

- login
- gestalt
- fw version

If API requires authentication but valid token wasn't passed server returns HTTP status code 401 Unauthenticated and string *Invalid Token.* in the response body.

HTTP Responses

The HTTP response can be used to determine if the request was successful, and if not, whether the request should be retried.

200 Success The request was successful.

401 Unauthenticated Request requires authentication but authorization failed. Application didn't handle the request.

404 Not Found If a string *Resource not found.* is found in the response body check if API endpoint is implemented in the current firmware version.

Application responses

The API may return application status as *code* value of JSON. Returned will not necessarily "correspond" with the HTTP status code. For example, a HTTP status code 200 OK returned with an error application code indicates that the request successfully reached the server, but application cannot process the request.

1100 Ok

1101 Invalid argument value

1102 Error

1103 Error - value too long? Or missing required object key?

1104 Error - malformed JSON on input?

1105 Invalid argument key

1107 Ok?

1108 Ok?

1205 Error with firmware upgrade - SHA1SUM does not match

3.2.2 Login

Request access token.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/login

Parameters

Parameters as JSON object.

challenge Random 32 byte string encoded with base64.

Response

The response will be an object.

authentication_token Access token in format: 8 byte string base64 encoded. First authenticated API with this token must be Verify.

challenge-response 41 byte string ([0-9a-h])

code Application return code.

authentication_token_expires_in: integer. All the time 14400?

Example

Request:

```
POST /xled/v1/login HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
Content-Length: 61

{"challenge": "AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8="}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 155
Content-Type: application/json

{"authentication_token":"5jPe+ONhwUY=","authentication_token_expires_in":14400,
↪"challenge-response":"8d87f080947e343180da3f411df3997e3e9ae0cc","code":1000}
```

3.2.3 Verify

Verify the token retrieved by Login.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/verify

Parameters

Parameters as JSON object.

challenge-response (optional) value returned by login request.

Response

The response will be an object.

code Application return code.

Example

Request:

```
POST /xled/v1/verify HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 66

{"challenge-response": "8d87f080947e343180da3f411df3997e3e9ae0cc"}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13
Content-Type: application/json

{"code":1000}
```

3.2.4 Logout

Probably invalidate access token. Doesn't work.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/logout

Response

The response will be an object.

code Application return code.

Example

Request:

```
POST /xled/v1/logout HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 2

{}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13
Content-Type: application/json

{"code":1000}
```

3.2.5 Device details

Gets information detailed information about the device.

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/gestalt

Response

The response will be an object.

product_name (string) *Twinkly*

product_version (numeric string), e.g. "2"

hardware_version (numeric string), e.g. "6"

flash_size (number), e.g. 16

led_type (number), e.g. 6

led_version (string) "1"

product_code (string), e.g. "TW105SEUP06"

device_name (string), by default consists of *Twinkly_* prefix and uppercased *hw_id* (see bellow)

rssi (number), Received signal strength indication. Since firmware version: 2.1.0.

uptime (string) number as a string, e.g. "60"

hw_id (string), right three bytes of mac address encoded as hexadecimal digits prefixed with 00.

mac (string) MAC address as six groups of two hexadecimal digits separated by colons (:).

uuid (string) UUID of the device. Since firmware version: 2.0.22.

max_supported_led (number), e.g. 180

base_leds_number (number), e.g. 105

number_of_led (number), e.g. 105

led_profile (string) "RGB"

frame_rate (number), 25

movie_capacity (number), e.g. 719

copyright (string) "LEDWORKS 2017"

code Application return code.

Example

Request:

```
GET /xled/v1/gestalt HTTP/1.1
Host: 192.168.4.1
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 406
Content-Type: application/json

{"product_name":"Twinkly","product_version":"2","hardware_version":"6","flash_size
↪":16,"led_type":6,"led_version":"1","product_code":"TW105SEUP06","device_name":
↪"Twinkly_33AAFF","uptime":"60","hw_id":"0033aaff","mac":"5c:cf:7f:33:aa:ff","max_
↪supported_led":224,"base_leds_number":105,"number_of_led":105,"led_profile":"RGB",
↪"frame_rate":25,"movie_capacity":719,"copyright":"LEDWORKS 2017","code":1000}
```

3.2.6 Get device name

Gets device name

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/device_name

Response

The response will be an object.

name (string) Device name.

code Application return code.

Example

Request:

```
GET /xled/v1/device_name HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 37
Content-Type: application/json

{"name":"Twinkly_33AAFF","code":1000}
```

3.2.7 Set device name

Sets device name

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/device_name

Parameters

Parameters as JSON object.

name (string) Desired device name. At most 32 characters.

Response

The response will be an object.

code Application return code. *1103* if too long.

Example

Request:

```
POST /xled/v1/device_name HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 26

{"name": "Twinkly_33AAFF"}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 37
Content-Type: application/json

{"name":"Twinkly_33AAFF","code":1000}
```

3.2.8 Get network status

Gets network mode operation.

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/network/status

Response

The response will be an object.

mode (enum) 1 or 2

station (object)

ap (object)

code Application return code.

Contents of object *station*:

ssid (string), SSID of a WiFi network to connect to

ip (string), IP address of the device

gw (string), IP address of the gateway

mask (string), subnet mask

status (integer), status of the network connection

Contents of object *ap*:

ssid (string), SSID of the device

channel (integer), channel number

ip (string), IP address

enc (integer)

Example

Request:

```
GET /xled/v1/network/status HTTP/1.1
Host: 192.168.1.2
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 187
Content-Type: application/json

{"mode":1,"station":{"ssid":"home","ip":"192.168.1.2","gw":"192.168.1.1","mask":"255.
↪255.255.0","status":5},"ap":{"ssid":"Twinkly_33AAFF","channel":11,"ip":"0.0.0.0",
↪"enc":0},"code":1000}
```

3.2.9 Set network status

Sets network mode operation.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/network/status

Parameters

Parameters as JSON object.

mode (enum) 1 or 2

station (object) if mode set to 1 this parameter provides additional details.

Station object parameters:

dhcp (integer) 1

ssid (string) SSID of a WiFi network

encpassword (string) encrypted password.

Response

The response will be an object.

code Application return code.

Example

Request to change network mode to client and connect to SSID "home" with password "Twinkly". Encoded with MAC address '5C:CF:7F:33:AA:FF':

```
POST /xled/v1/network/status HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 150

{"mode":1,"station":{"ssid":"home","encpassword":
↪"e4XXiiUhg4J1FnJEfUQ0BhIji2HGVk1NHU5vGCHfyclFdX6R8Nd9BSXVKS5nj2FXGU6SWy9CIzztfAvGgTGLUw==
↪","dhcp":1}}
```

(continues on next page)

(continued from previous page)

```
```

Request to change network mode to AP:

```
POST /xled/v1/network/status HTTP/1.1
Host: 192.168.1.100
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 10

{"mode":2}
```

3.2.10 Get timer

Gets time when lights should be turned on and time to turn them off.

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/timer

Response

The response will be an object.

time_now (integer) current time in seconds after midnight

time_on (number) time when to turn lights on in seconds after midnight. -1 if not set

time_off (number) time when to turn lights off in seconds after midnight. -1 if not set

Example

Request:

```
GET /xled/v1/timer HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 45
Content-Type: application/json

{"time_now":17083,"time_on":-1,"time_off":-1}
```

3.2.11 Set timer

Sets time when lights should be turned on and time to turn them off.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/timer

Parameters

Parameters as JSON object.

time_now (integer) current time in seconds after midnight

time_on (number) time when to turn lights on in seconds after midnight. -1 if not set

time_off (number) time when to turn lights off in seconds after midnight. -1 if not set

Example

Request to set current time to 2:00 AM, turn on lights at 1:00 AM and turn off at 4:00 AM:

```
POST /xled/v1/timer HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 51

{"time_now": 120, "time_on": 60, "time_off": 240}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13
Content-Type: application/json

{"code":1000}
```

3.2.12 Get LED operation mode

Gets current LED operation mode.

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/led/mode

Response

The response will be an object.

code Application return code.

mode (string) mode of operation.

Mode can be one of:

- *off* - lights are turned off
- *demo* - in demo mode
- *movie* - plays predefined or uploaded effect
- *rt* - receive effect in real time

Example

Request:

```
GET /xled/v1/led/mode HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 28
Content-Type: application/json

{"mode":"movie","code":1000}
```

3.2.13 Set LED operation mode

Changes LED operation mode.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/led/mode

Parameters

Parameters as JSON object.

mode (string) mode of operation.

Mode can be one of:

- *off* - turns off lights
- *demo* - starts predefined sequence of effects that are changed after few seconds
- *movie* - plays predefined or uploaded effect

- *rt* - receive effect in real time

Response

The response will be an object.

code Application return code.

Example

Request:

```
POST /xled/v1/led/mode HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
Content-Length: 15

{"mode":"demo"}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13
Content-Type: application/json

{"code":1000}
```

3.2.14 Upload full movie

Effect is received in body of the request with Content-Type application/octet-stream. If mode is *movie* it starts playing this effect.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/led/movie/full

Response

The response will be an object.

code Application return code.

frames_number (integer) number of received frames

3.2.15 Get LED effects

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/led/effects

Response

The response will be an object.

code Application return code.

effects_number (integer), e.g. 5

Example

Request:

```
GET /xled/v1/led/effects HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 32
Content-Type: application/json

{"effects_number":5,"code":1000}
```

3.2.16 Get current LED effect

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/led/effects/current

Response

The response will be an object.

code Application return code.

effect_id (integer), e.g. 0

Example

Request:

```
GET /xled/v1/led/effect/current HTTP/1.1
Host: 192.168.4.1
Content-Type: application/json
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 27
Content-Type: application/json

{"effect_id":0,"code":1000}
```

3.2.17 Get LED config

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/led/config

Response

The response will be an object.

strings Array of objects

code Application return code.

Item of strings array is object:

first_led_id (integer), e.g. 0

length (integer), e.g. 105

Example

Request:

```
GET /xled/v1/led/config HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 57
Content-Type: application/json

{"strings":[{"first_led_id":0,"length":105}],"code":1000}
```

3.2.18 Set LED config

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/led/config

Parameters

Parameters as JSON object.

strings Array of objects

Item of strings array is object:

first_led_id (integer), e.g. 0

length (integer), e.g. 105

Response

The response will be an object.

code Application return code.

Example

Request:

```
POST /xled/v1/led/config HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
Content-Type: application/json
Content-Length: 45

{"strings":[{"first_led_id":0,"length":100}]}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13
Content-Type: application/json

{"code":1000}
```

3.2.19 Get LED movie config

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/led/movie/config

Response

The response will be an object.

frame_delay (integer)

leds_number (integer) seems to be total number of LEDs to use

loop_type (integer), e.g. 0

frames_number (integer)

sync (object)

code Application return code.

Contents of object *sync*:

mode (string), e.g. "none"

slave_id (string), e.g. ""

master_id (string), e.g. ""

Example

Request:

```
GET /xled/v1/led/movie/config HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 134
Content-Type: application/json

{"frame_delay":40,"leds_number":105,"loop_type":0,"frames_number":325,"sync":{"mode":
↪"none","slave_id":"","master_id":""},"code":1000}
```

3.2.20 Set LED movie config

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/led/movie/config

Parameters

Parameters as JSON object.

frame_delay (integer)

leds_number (integer) seems to be total number of LEDs to use

frames_number (integer)

Response

The response will be an object.

code Application return code.

3.2.21 Get current brightness

Gets the current brightness level.

Since firmware version: 2.3.5.

HTTP request

GET /xled/v1/led/out/brightness

Response

The response will be an object.

code Application return code.

mode (string) one of "enabled" or "disabled".

value (integer) brightness level in range of 0..255

Mode string displays if the dimming is applied. The led shines at full brightness regardless of what value is set if the *mode* is *disabled*. Brightness level value seems to represent percent so 0 is dark and maximum meaningful value is 100. Greater values doesn't seem to have any effect.

Example

Request:

```
GET /xled/v1/led/out/brightness HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 37
Content-Type: application/json

{"value":"100,"mode":"enabled","code":1000}
```

3.2.22 Set brightness

Since firmware version: 2.3.5.

HTTP request

POST /xled/v1/led/out/brightness

Parameters

Parameters as JSON object.

mode**:** (string) one of "enabled", "disabled"

type**:** (string) always "A"

value**:** (integer) brightness level in range of 0..255

When *mode* is "disabled" no dimming is applied and the led works at full brightness. It is not necessary to submit all the parameters, basically it would work if only *value* or *mode* is supplied. *type* parameter can be omitted, and the only value seen on the wire was "A". Brightness level value seems to represent percent so 0 is dark and maximum meaningful value is 100. Greater values doesn't seem to have any effect.

Response

The response will be an object.

code Application return code.

Example

Set the brightness level to 10%:

Request:

```
POST /xled/v1/led/out/brightness HTTP/1.1
Host: 192.168.4.1
X-Auth-Token: 5jPe+ONhwUY=
Content-Type: application/json
Content-Length: 45

{"mode":"enabled","type": "A","value": "100"}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13

{"code":1000}
```

3.2.23 Get firmware version

Note: no authentication needed.

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/fw/version

Response

The response will be an object.

code Application return code.

version (string)

Example

Request:

```
GET /xled/v1/fw/version HTTP/1.1
Host: 192.168.4.1
Accept: */*
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 33
Content-Type: application/json

{"version":"1.99.24","code":1000}
```

3.2.24 Update firmware

Probably initiates firmware update.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/fw/update

Parameters

Parameters as JSON object.

checksum (object)

Checksum object parameters:

stage0_sha1sum (string) SHA1 digest of first stage

stage1_sha1sum (string) SHA1 digest of second stage

Response

The response will be an object.

code Application return code.

Example

Request:

```
POST /xled/v1/fw/update HTTP/1.1
X-Auth-Token: 5jPe+ONhwUY=
Content-Type: application/json
Content-Length: 134
Host: 192.168.4.1

{"checksum":{"stage0_sha1sum":"1c705292285a1a5b8558f7b39abd22c5550606b5","stage1_
↪sha1sum":"ac691b8d4563dcdbb3f837bf3db2ebf56fe77fbe"}}
```

Response:

```
HTTP/1.1 200 Ok
Connection: close
Content-Length: 13
Content-Type: application/json

{"code":1000}
```

3.2.25 Upload first stage of firmware

First stage of firmware is uploaded in body of the request with Content-Type application/octet-stream.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/fw/0/update

Response

The response will be an object.

code Application return code.

sha1sum SHA1 digest of uploaded firmware.

3.2.26 Upload second stage of firmware

Second stage of firmware is uploaded in body of the request with Content-Type application/octet-stream.

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/fw/1/update

Response

The response will be an object.

code Application return code.

sha1sum SHA1 digest of uploaded firmware.

3.2.27 Initiate WiFi network scan

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/network/scan

Response

The response will be an object.

code Application return code.

3.2.28 Get results of WiFi network scan

Since firmware version 1.99.20.

HTTP request

GET /xled/v1/network/scan_results

Response

The response will be an object.

code Application return code.

networks Array of objects

Item of networks array is object:

ssid (string)

mac (string)

rssi (number) negative number

channel (integer)

enc One of numbers 0 (Open), 1 (WEP), 2 (WPA-PSK), 3 (WPA2-PSK), 4 (WPA-PSK + WPA2-PSK), 5 (WPA2-EAP).

Response seems to correspond with AT command CWLAP.

3.2.29 Set LED driver parameters

Since firmware version 1.99.20.

HTTP request

POST /xled/v1/led/driver_params

3.2.30 Reset LED

HTTP request

GET /xled/v1/led/reset

Response

The response will be an object.

code Application return code.

3.2.31 Get MQTT configuration

Since firmware version: 2.0.22

HTTP request

GET /xled/v1/mqtt/config

Response

The response will be an object.

code Application return code.

broker_host (string), hostname of broker. By default *mqtt.twinkly.com*.

broker_port (integer), destination port of broker. By default “1883”.

client_id (string), by default hex string of length 12 derived from MAC address of the device as uppercased hexadecimal digits.

encryption_key_set (bool), by default “False”

keep_alive_interval (integer), by default “180”.

user (string), by default “twinkly_noauth”

3.2.32 Set MQTT configuration

Since firmware version: 2.0.22

HTTP request

POST /xled/v1/mqtt/config

Parameters

Parameters as JSON object.

broker_host (string), hostname of broker

broker_port (integer), destination port of broker

client_id (string)

encryption_key (string), length exactly 16 characters?

keep_alive_interval cannot be set?

user (string)

Response

The response will be an object.

code Application return code.

CHAPTER 4

The Contributor Guide

4.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/scrool/xled-docs/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

xled could always use more documentation, whether as part of the official xled docs or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/scrool/xled-docs/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Credits

4.2.1 Development Lead

- Pavol Babinčák <scroolik@gmail.com>

4.2.2 Contributors

- Paul Webster (@PaulWebster)
- Artem Ignatyev (@timon)

4.3 History

4.3.1 2.4.21.2 (2020-12-22)

- Add new REST API endpoints Get and set LED config
- Add new REST endpoints: get LED effects and current LED effects
- Improve application response description and add 1205
- More consistent info about first firmware supporting MQTT
- Document response 404 Not found
- Specify initial firmware versions for all REST API endpoints
- Move Logout earlier in the list of rest APIs
- Rename section Change LED operation mode -> Set LED operation mode
- Fix typos
- Shorten main headings
- Restructure rest api

4.3.2 2.4.21.1 (2020-12-07)

- Use tab for indentation consistently
- Add corresponding GET calls and one more example
- Rename firmware 2.0.22-mqtt to 2.0.22.

4.3.3 2.4.21.0 (2020-12-07)

- Introduce generation II and improve introduction

4.3.4 2.3.5.0 (2020-12-03)

- Brightness can be set and read through MQTT
- Fix typo in filename: msqtt_api.rst -> mqtt_api.rst
- Brightness is available since firmware 2.3.5
- Remove Cookiecutter_credits - it wasn't used in this project
- Brightness is most likely percent so improve its description
- Improve LED brightness REST endpoint
- Contribute myself to the list of authors
- Describe led brightness REST endpoint

4.3.5 2.1.0.2 (2018-12-15)

- gestalt returns rssi

4.3.6 2.1.0.1 (2018-12-09)

- Updated MQTT status API in new firmware

4.3.7 2.0.22.1 (2018-12-05)

- Initial description on MQTT API

4.3.8 1.99.24.11 (2018-11-27)

- Fix links in README

4.3.9 1.99.24.10 (2018-11-27)

- New return value - 1104

4.3.10 1.99.24.9 (2018-11-21)

- Describe private API for uploading movies.

4.3.11 1.99.24.8 (2018-11-18)

- Fix typos.
- Fix formatting.
- In private API repeat warning about private API.
- Remove uncertainty about real time mode.

4.3.12 1.99.24.7 (2018-11-12)

- Minor formatting improvements.

4.3.13 1.99.24.6 (2018-01-02)

- Protocol details about realtime and full movie upload LED modes.

4.3.14 1.99.24.5 (2018-01-02)

- Improve documentation for contributors.

4.3.15 1.99.24.4 (2017-12-17)

- First public documentation for private API.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search