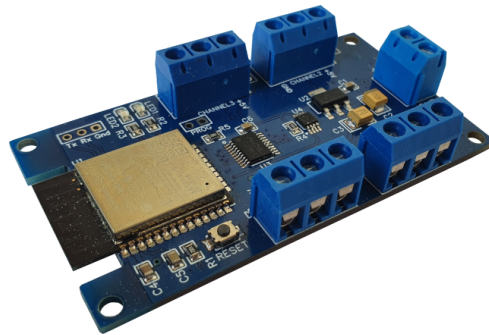




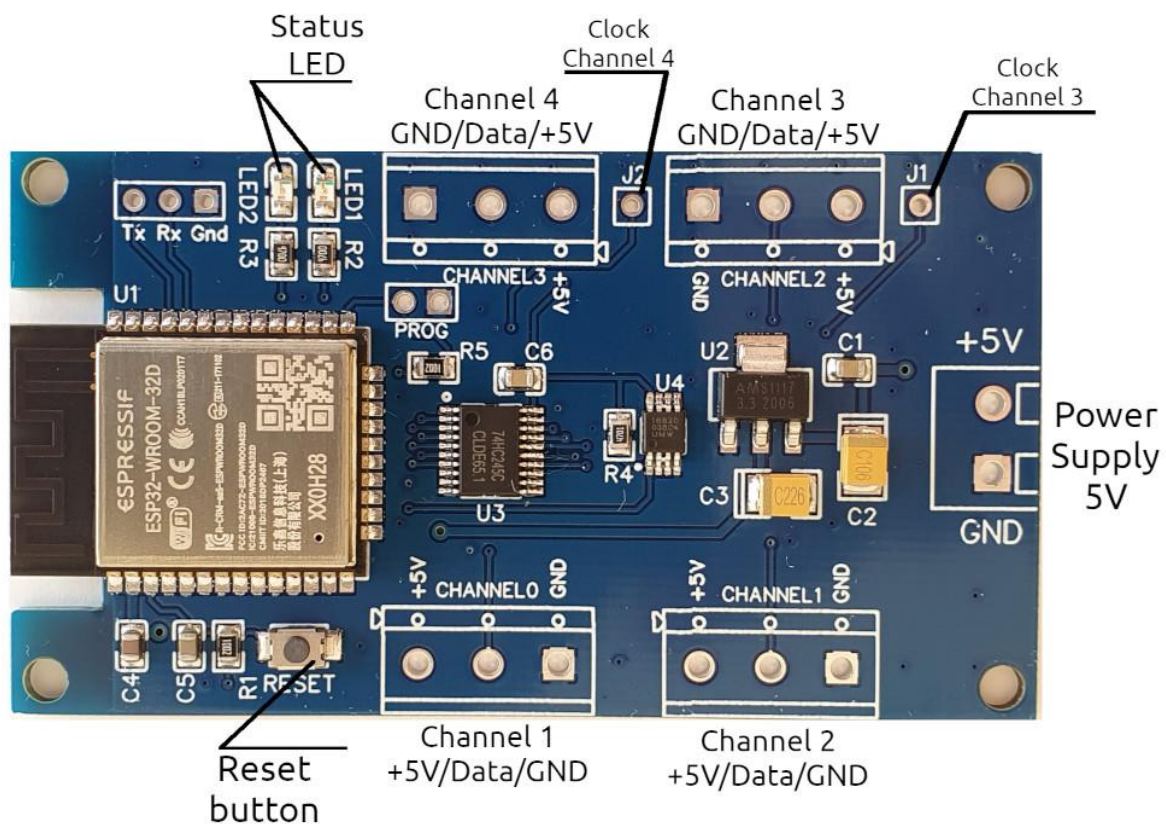
4 channel LED Controller

User Manual



Board	3
Connectors	3
Status LEDs	4
Reset button	5
Supported LED types	5
Web UI	5
First powering	5
Pages	6
WiFi	6
MQTT	7
Lights	7
Info	8
WiFi Configuration	9
MQTT	10
Configuration	10
MQTT Topics	10
Light control topics	11
Light settings topics	12
OTA control topics	12
Device control topic	14
Status topics	14
Attribute message topic	14
Examples	15
The structure of JSON data	17
JSON Data	17
Examples	18
Animation Effects	19
Parameters	19
Supported effects	20
Examples	23
Transition Effects	24
Parameters of transitions	24
Supported transitions	24
Examples	26
Pictures	27
Supported pictures	27
Custom pattern	28
Examples	29
Firmware	30
OTA	31
Factory Reset	32
Home Assistant Integration	33

Board



The image above shows the pixie board and all possible connections to the board.

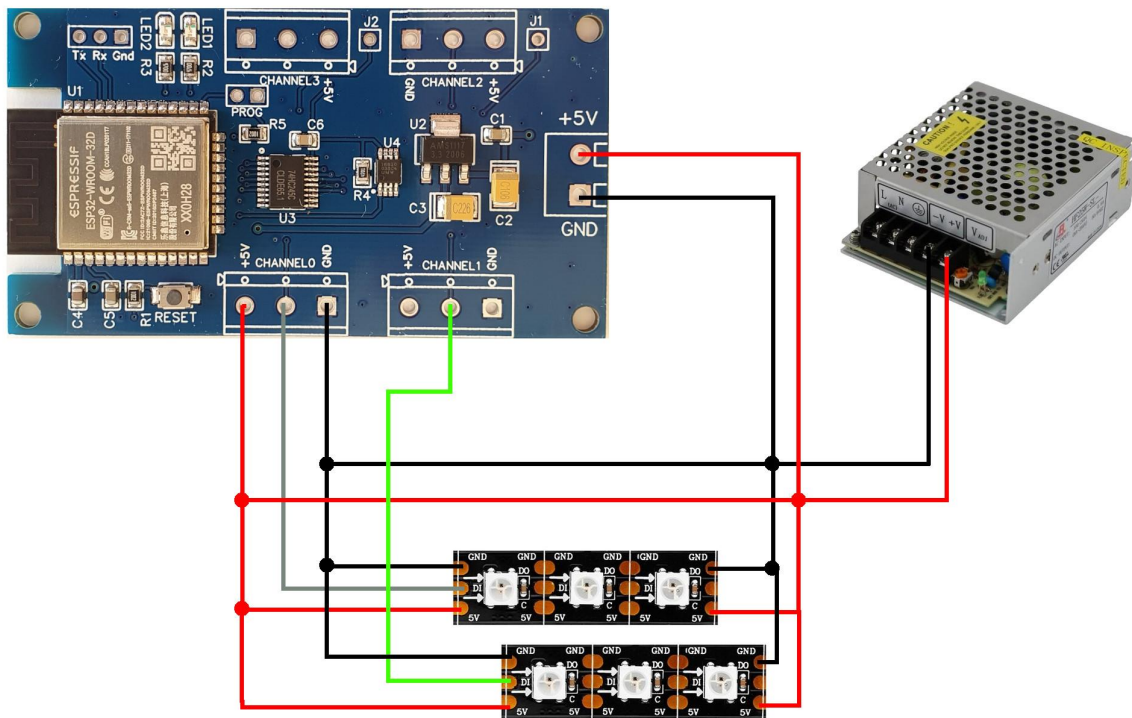
Connectors

The board has 4 independent hardware LED channels: All four channels can drive LED strips with one data line and only 2 channels can drive LED strips which require a clock signal (Connectors J1 and J2 on the board for channel 3 and channel 4 correspondingly)

IMPORTANT! The board can be only powered with a 5V power supply.

WARNING! DO NOT REVERSE POLARITY OF THE POWER SUPPLY. IT IS GOING TO DAMAGE THE BOARD !

If a long LED strip is connected to the board it should be supplied by a separate power line. Do not pull too much current through the output channel connector. Use connection shown in the picture below:



Status LEDs

The board has 2 status LEDs: a green one (sometimes a yellow LED) and a blue one. They indicate different states of the board:

Green LED	Blue LED	State
ON	ON	The board is booting
OFF	Blink every second	Connecting to the specified WiFi network
One short blink every 5 sec	OFF	Board connected to the specified WiFi Access Point
Two short blinks every 5 sec	OFF	Board connected to the specified WiFi Access Point and to the specified MQTT server
OFF	ON	The board has activated a WiFi Access Point
Flashing (700 ms ON/700 ms OFF)	Flashing (700 ms OFF/700 ms ON)	OTA update is active

Fast Flashing (100
ms ON/100 ms
OFF)

Fast Flashing (100
ms OFF/100 ms
ON)

Reset MCU to Factory settings

Reset button

The reset button on the board only restarts the controller. All settings won't be erased. After restart the board will restore its previous state.

Supported LED types

There are several LED types that the controller supports:

- WS2812 - value for a config json: "ws2812"
- WS2812b - value for a config json: "ws2812b"
- WS2812d - value for a config json: "ws2812d"
- WS2811 - value for a config json: "ws2811"
- SK6812 (RGB) - value for a config json: "sk6812_grb"
- SK6812 (RGBW) - value for a config json: "ws2812_grbw"
- APA102 - value for a config json: "apa102"

Those LED types which require an extra pin for clocking can be connected only to Channel2 or Channel3. (See the board pic) The pins J1 and J2 on the board are the clock signal for the channels 3 and 4 correspondingly. Every channel can drive up to 1000 LEDs. Please make sure that the power supply can provide enough current to drive all LEDs.

Web UI

A Pixie light controller can be controlled over WebUI that can be open in your browser. Every controller has a unique URL (web address) to access the UI, like <http://pixie-abcdef.local>. Please note, here -abcdef is a unique id of your device.

First powering

When a pixie device is powered for the first time it creates a WiFi Access Point Pixie_ABCDEF, where ABCDEF is a unique id of the device. The access point has a password which coincides with the Access Point name (SSID).

NOTE: Please keep in mind that the password is case sensitive and must coincide with the name of the access point.

Connect to the access point and open the address <http://192.168.4.1> in your browser. This will open the WebUI where the device can be configured and controlled.


Pages



The WebUI has four pages:



Pages


WiFi	WiFi configuration page. Here a connection to an AP can be set.
MQTT	MQTT configuration page. A connection to an MQTT broker can be configured here if there is a need to control the device over MQTT.
Lights	All light channels can be controlled on this page.
Info	Contains some common information about the controller.



WiFi



 WiFi MQTT Lights Info



 Lime2 



 Galaxy 



 Vardagsrum.I025



 Doorbell_16C653 



 NETGEAR-30 

 ORBI36 

 #Telia-98C388 

 Telia-40F827 

 #Telia-917E20 

 Telia-305505 

SSID

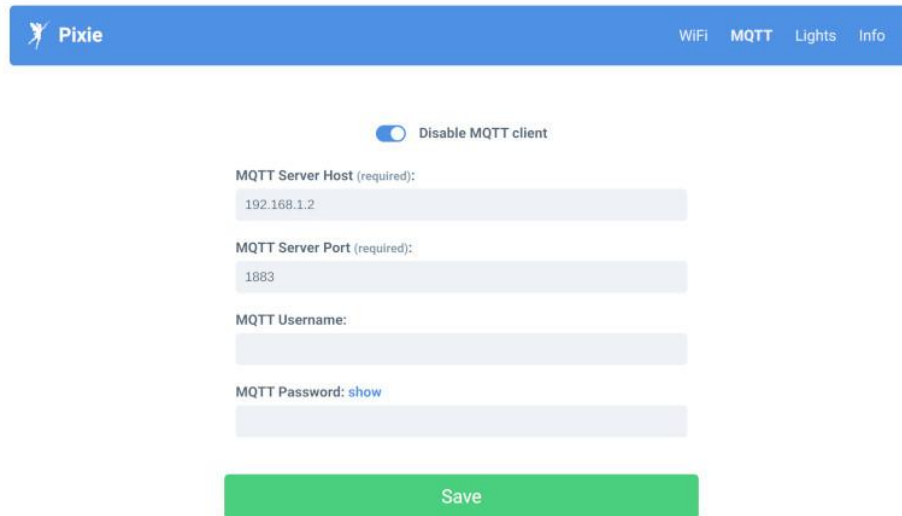
Password: [show](#)

Connect

Scan

Here a connection to a WiFi access point (AP) of your home network can be configured. When the page is open in your browser it initiates a scanning of available WiFi networks. When the list is ready, click on any WiFi access point, input the password and click the "Connect" button below. When the device successfully connects to your AP it shuts down its own access point and remains connected to the specified AP.

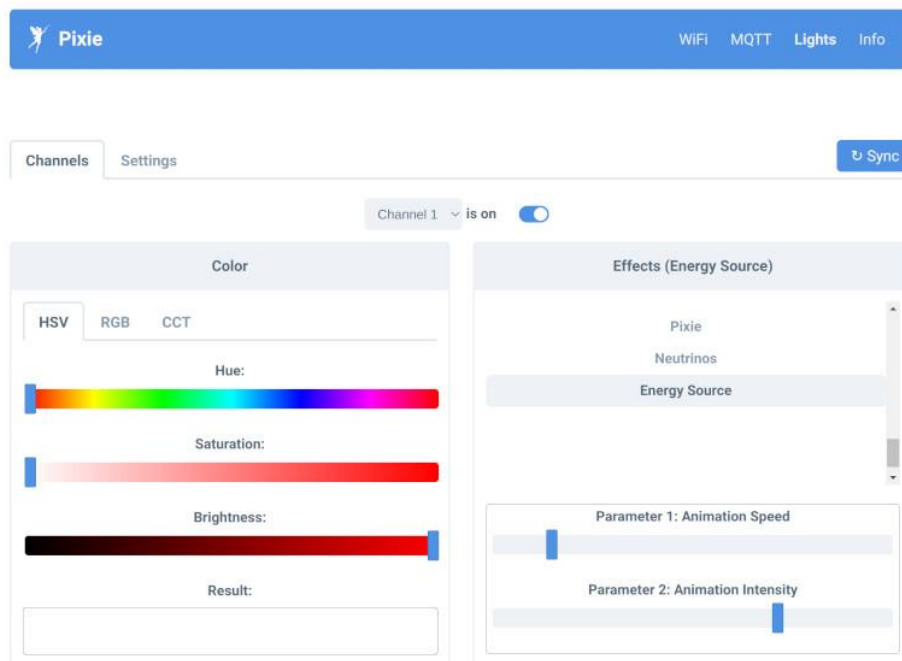
MQTT



The MQTT configuration interface features a blue header with the 'Pixie' logo and navigation links for 'WiFi', 'MQTT', 'Lights', and 'Info'. Below the header, there is a toggle switch labeled 'Disable MQTT client'. The main section contains four input fields: 'MQTT Server Host (required):' with the value '192.168.1.2', 'MQTT Server Port (required):' with the value '1883', 'MQTT Username:', and 'MQTT Password: show'. A green 'Save' button is positioned at the bottom of the form.


The inbuilt MQTT client can be configured on this page. Input the ip address/port (or a domain name) of your broker and login and password if required. Control the current connection status with [status LED](#) on the board or on the Info page.

Lights



The Lights configuration interface has a blue header with the 'Pixie' logo and navigation links for 'WiFi', 'MQTT', 'Lights', and 'Info'. Below the header, there are tabs for 'Channels' and 'Settings', and a 'Sync' button. The 'Channels' tab is active, showing 'Channel 1' with a dropdown arrow, 'is on', and a toggle switch. The main area is divided into two panels. The left panel, titled 'Color', has tabs for 'HSV', 'RGB', and 'CCT'. It includes sliders for 'Hue:', 'Saturation:', and 'Brightness:', a 'Result:' display, and a color bar. The right panel, titled 'Effects (Energy Source)', has a scrollable list with 'Pixie', 'Neutrinos', and 'Energy Source'. Below this are two sliders: 'Parameter 1: Animation Speed' and 'Parameter 2: Animation Intensity'.

On this page a user can set on/off any light channel, choose an animation effect or set a solid color on the LED strips connected to the device. It's also possible to configure what type of LED strips are connected to each channel and how many LEDs every strip has.

 Pixie

[WiFi](#) [MQTT](#) [Lights](#) [Info](#)

Channels

Settings

Sync

Channel 1

Number of LEDs60

Type of LEDsWS2812B (Color order: GRB; 3 colr

Channel 2

Number of LEDs60

Type of LEDSSK6812 + White (Color order: GRB

Channel 3

Number of LEDs60

Type of LEDsAPA102 (Same as SK9822)

Channel 4

Number of LEDs49

Type of LEDsWS2812B (Color order: GRB; 3 colr

Save

Info

Network Status

WiFi	Connected (Galaxy)
MQTT	Connected

Firmware

Firmware version	0.8.5
Available version	No information Check update

Device

Developer	IoT-Hus14
MCU	ESP32 (rev:1)
Flash size	16MB
MAC Address	98:F4:AB:98:E4:E4
IP Address	192.168.1.182
Uptime	1T00:44:40
Board Temperature	30.1C (86.18F)

Links

The device is powered by [FreeRTOS](#).

The UI is created with a CSS toolkit [Siimple](#).


Device [wiki](#).

[Reboot](#)


The info page contains some extra information about the device, connection statuses to WiFi and an MQTT broker, uptime, firmware version, OTA update and links to documentation.


WiFi Configuration


When the WiFi configuration page is open it starts scanning available WiFi networks. When the scan is done choose yours by clicking on it in the list of the WiFi networks and enter the password in the field below:


 Pixie


WiFiMQTTLightsInfo


 Lime2


 Vardagsrum.I025


 Galaxy


 Doorbell_16C653

 NETGEAR-30

 Telia-40F827

 Telia-305505

 #Telia-98C388

 Telia-EF1FC1

SSID

Lime2

Password:

4hj6h388fb8x76z

hide

Connect

Scan

Click the connect button and let the device connect to your WiFi network. When the device is connected, make sure your computer/phone is in the same WiFi network and then try to access the device by this URL: <http://pixie-abcdef.local> , where `abcdef` is the same unique id of the device. If the URL doesn't work, check which IP address the pixie device obtained from your WiFi router and open the WebUI with the following URL <http://192.168.1.123>. The address `192.168.1.123` is just an example here and you should use the one which your pixie device got in the WiFi network.

The connection status can be also controlled by the [status LED](#) on the board.

MQTT

Configuration

The inbuilt mqtt client can be configured in [Web UI](#) by specifying the mqtt server, a username and a password if they are required.

MQTT Topics

There are several topics which the controller subscribes to:

- Light control topics

- Light settings topics
- OTA control topics
- Device control topic

Light control topics

Pixie has four independent hardware light channels which can be controlled via MQTT messages separately. Every channel has three topics:

1. **Command topic** - an mqtt topic which the device receives [light control json data](#).
2. **Light state request topic** - if the controller receives an mqtt message with any payload to this topic, it reports the current state of the requested channel to the report state topic.
3. **Report state topic** - The device will report the current state of a channel when it was requested or when the light channel has changed its state (went on or off / changed a color or animation effect and so on).

Channel 0:

- Command topic: `pixie_abcdef/channel0/set`
- Light state request topic `pixie_abcdef/channel0/get`
- Report state topic: `pixie_abcdef/channel0`

Channel 1:

- Command topic: `pixie_abcdef/channel1/set`
- Light state request topic `pixie_abcdef/channel1/get`
- Report state topic: `pixie_abcdef/channel1`

Channel 2:

- Command topic: `pixie_abcdef/channel2/set`
- Light state request topic `pixie_abcdef/channel2/get`
- Report state topic: `pixie_abcdef/channel2`

Channel 3:

- Command topic: `pixie_abcdef/channel3/set`
- Light state request topic `pixie_abcdef/channel3/get`
- Report state topic: `pixie_abcdef/channel3`

All topics include a unique id of the controller which looks like a postfix `_abcdef` in the topics.

Besides all these topics listed above it is possible to control all four light channels simultaneously and send the same command to all channels. The topics look like:

- Command topic: `pixie_abcdef/channel/set` - will set the same command to all four channels.
- Light state request topic `pixie_abcdef/channel/get` - the device will report the current state of all channels separately.

Light settings topics

All light channels can be configured via MQTT. There are, similar to the light control topics, three different topics to set / request / report the current settings:

- `pixie_abcdef/settings/set` - set a new configuration.
- `pixie_abcdef/settings/get` - request the current configuration.
- `pixie_abcdef/settings` - the device reports the current configuration on a request.

As a payload of a configuration message a json document must be sent. The structure of the json is shown below:

```
{
  "channel0": {
    "active":1,
    "num_leds":80,
    "type":"ws2811"
  },
  "channel1": {
    "active":1,
    "num_leds":260,
    "type":"ws2812b"
  },
  "channel2": {
    "active":1,
    "num_leds":60,
    "type":"apa102"
  },
  "channel3": {
    "active":1,
    "num_leds":30,
    "type":"sk6812_grbw"
  }
}
```

Here, `active` - the value must be always 1; `num_leds` - number of LEDs connected to the channel; `type` - type of LEDs, must be one of the supported types. Please see the valid values in the [supported LED types](#).

NOTE: When the controller receives new settings successfully it will reboot to apply the changes.

OTA control topics

Availability an Over The Air (OTA) update can be requested over an mqtt message by sending any payload to the topic:

```
pixie_abcdef/ota/check
```

The device will respond the following MQTT message: Topic:

pixie_abcdef/ota

Payload:

```
{
  "ota_state": "start",
  "ota_type": "check",
  "result": 1,
  "message": "Check on an available OTA update has started"
}
```

This operation can take a little time for the controller to connect to the remote server and get a reply. In case if an ota request was done successfully this message will be sent to the topic pixie_abcdef/ota :

```
{
  "ota_state": "end",
  "ota_type": "check",
  "result": 1,
  "running_version": "1.0.0",
  "remote_version": "1.0.1",
  "message": "Success"
}
```

If a request fails, this json will be sent instead:

```
{
  "ota_state": "end",
  "ota_type": "check",
  "result": 0,
  "message": "Failed to check an available update"
}
```

The update itself can be performed by sending any payload to the topic:

pixie_abcdef/ota/perform

With this the device will respond with a json document first to say that an update has started:

```
{
  "ota_state": "start",
  "ota_type": "update",
  "result": 1,
  "message": "OTA update has started"
}
```

If the update is done successfully the following json document is sent:

```
{
  "ota_state": "end",
  "ota_type": "update",
  "result": 1,
  "message": "Success"
}
```

```
}
```

And the device restarts automatically to boot to the new firmware. If something went wrong this json will be sent instead:

```
{
  "ota_state": "end",
  "ota_type": "update",
  "result": 0,
  "message": "<Problem description>"
}
```

Device control topic

The only action, which can currently be done is to reboot the device. This can be achieved by sending any mqtt message to the topic

`pixie_abcdef/reboot`

This will restart the device immediately.

Status topics

When the controller gets connected to an mqtt server it notifies about this with an mqtt message:

<code>pixie_abcdef/status</code>	Topic where the controller reports its status.
<code>online</code>	Payload when the device is connected.
<code>offline</code>	Payload when the device is disconnected.

These messages have a set retain flag.

Attribute message topic

Every 10 minutes the controller sends an attribute message to the topic

`pixie_abcdef/attributes`

to report some extra information. The payload is a json document in the following format:

```
{
  "firmware_version": "1.0.0",
  "mac": "XX:YY:ZZ:AB:CD:EF",
}
```

```

    "ip_addr": "192.168.1.107",
    "uptime": "0T00:10:00",
    "url": "http://pixie-abcdef.local",
    "board_temperature": 27.6
}

```

Here:

- `firmware_version` - the current version of the running firmware.
- `mac` - MAC address of the device.
- `ip_addr` - IP address which the device got in the network.
- `uptime` - this holds time showing how long the controller is running. The format: DaysTHours:Minutes:Seconds.
- `url` - This is the URL of Web UI.
- `board_temperature` - A board temperature. Please, note, this is optional and some versions of the controllers do not have a temperature sensor on board. Here the temperature is always in Celsius. This might be needed if the controller is located in a closed box where it's required to control the temperature.

An attribute message can be requested at any time by sending any message to the following topic:

```
pixie_abcdef/attributes/get
```

This might be helpful if there is an interest for the board temperature which an attribute message carries.

Examples

Here is an example of typical communication with a pixie controller over MQTT:

A user configures the MQTT client of a Pixie device. The device gets connected to the server and immediately reports about its status and states of all channels

```

pixie_abcdef/status` -> online
pixie_abcdef/channel0` -> {"state":"OFF"}
pixie_abcdef/channel1` -> {"state":"OFF"}
pixie_abcdef/channel2` -> {"state":"OFF"}
pixie_abcdef/channel3` -> {"state":"OFF"}
pixie_abcdef/attributes` -> { "firmware_version":"0.8.2",
    "mac":"11:22:33:AB:CD:EF", "ip_addr":"192.168.1.12", "uptime":"0T00:00:01",
    "url":"http://pixie-abcdef.local"}

```

The user sets a solid red color to the channel 0 and the "Rainbow" effect to the channel 2

```

pixie_abcdef/channel0/set` -> {"state":"on","color":{"r":255,"g":0,"b":0}}
pixie_abcdef/channel2/set` -> {"state":"on","effect":"Rainbow","brightness":120}

```

The device receives and applies the commands and reports the state back

```

pixie_abcdef/channel0` -> {"state":"on","color":{"r":255,"g":0,"b":0}}

```

```
pixie_abcdef/channel2` -> {"state":"on","effect":"Rainbow","brightness":120}
```

The controller periodically sends an attribute message (every 10 minutes)

```
pixie_abcdef/attributes` -> {"firmware_version":"1.0.0",  
"mac":"XX:YY:ZZ:AB:CD:EF", "ip_addr":"192.168.1.107", "uptime":"0T00:10:00", "u  
rl":"http://pixie-abcdef.local"}
```

The user connects a new LED strip to the port 4 with 200 WS2812B LEDs and configures the channel to the strip:

```
pixie_abcdef/settings/set` ->  
{"channel3":{"active":1,"num_leds":200,"type":"ws2812b"}}
```

To apply the new settings the device restarts and reports its state again:

```
pixie_abcdef/status` -> offline  
...  
pixie_abcdef/status` -> online  
pixie_abcdef/channel0` -> {"state":"ON","color":{"r":255,"g":0,"b":0}}  
pixie_abcdef/channel1` -> {"state":"OFF"}  
pixie_abcdef/channel2` -> {"state":"ON","effect":"Rainbow","brightness":120}  
pixie_abcdef/channel3` -> {"state":"OFF"}  
pixie_abcdef/attributes` -> { "firmware_version":"0.8.2",  
"mac":"11:22:33:AB:CD:EF", "ip_addr":"192.168.1.12", "uptime":"0T00:00:01",  
"url":"http://pixie-abcdef.local"}
```

Now the user wants to turn on the channel 3 with a transition "SinIn" in 3 minutes, so a json the user sends is:

```
pixie_abcdef/channel3/set` ->  
{"state":"on","transition":180,"transition_name":"SinIn","color":  
{"r":255,"g":128,"b":0}}
```

When the device receives the MQTT message it runs the requested transition to the color #ff8000 and again reports the applied command:

```
pixie_abcdef/channel3` -> {"state":"on","color": {"r":255,"g":128,"b":0}}
```

At this point the user wants to check if there is any available OTA update. They send an mqtt message to the topic (the payload doesn't matter)

```
pixie_abcdef/ota/check -> 1
```

Almost immediately the device responds with a message:

```
pixie_abcdef/ota ->  
{"ota_state":"start","ota_type":"check","result":1,"message":"Check on an  
available OTA update has started"}
```

In a few seconds the devices responds with a message:

```
pixie_abcdef/ota ->
```



```
{"ota_state":"end","ota_type":"check","result":1,"running_version":
"1.0.1","remote_version": "1.0.1","message":"Success"}
```

Which means the controller has the latest firmware.

The user wants to request the current state of all light channels, they send the following mqtt message:

```
pixie_abcdef/channel/get -> 1
```

Here it doesn't matter what the payload is. The device responds by sending the state of every light channel:

```
pixie_abcdef/channel0 -> {"state":"ON","color":{"r":255,"g":0,"b":0}}
pixie_abcdef/channel1 -> {"state":"OFF"}
pixie_abcdef/channel2 -> {"state":"ON","effect":"Rainbow","brightness":120}
pixie_abcdef/channel3 -> {"state":"ON","color": {"r":255,"g":128,"b":0}}
```

Eventually the user switches off all the lights by sending this message:

```
pixie_abcdef/channel/set -> {"state":"OFF"}
```

and the controller switches all LED strips off and sends a response:

```
pixie_abcdef/channel0 -> {"state":"OFF"}
pixie_abcdef/channel1 -> {"state":"OFF"}
pixie_abcdef/channel2 -> {"state":"OFF"}
pixie_abcdef/channel3 -> {"state":"OFF"}
```

The structure of JSON data

JSON Data

The device can be controlled over [MQTT](#) by receiving a structured JSON document. This part describes the structure of JSON data which can be sent to the controller.

```
{
  "state": "ON",
  "color": {
    "r": 0,
    "g": 128,
    "b": 255,
  },
  "effect": "ColorLoop",
  "brightness": 255,
  "parameter1": 130,
  "parameter2": 255,
  "transition": 10,
  "transition_name": "Fade",
}
```

```

"picture": "Rainbow",
"picture_pattern": [
  {
    "leds":20,
    "c":{"r":255,"g":0,"b":0}
  },
  {
    "leds":20,
    "c":{"r":255,"g":255,"b":0}
  },
  {
    "leds":20,
    "c":{"r":0,"g":0,"b":0}
  }
],
"white_value": 210
}

```

Keys:

- **state** : string - takes only two values: "ON" and "OFF". If the value is "ON" - the device goes on and reads other parameters in the document. If the value is "OFF", the device ignores all attributes in the document and switches off all LEDs.
- **color** : Object - The color object supports only three components of any color: Red, Green, Blue which have corresponding keys in the JSON: "r", "g" and "b". A valid value for each color component is an integer number in the range 0..255.
- **effect** : string - takes predefined animation names as a string. Please check more details [here](#) . If any effect is specified, it is applied immediately with no any extra effect such as a transition, for instance.
- **brightness** : integer - a valid value is an integer number in the range 0..255.
- **parameter1** : integer - is used as an input parameter for some animation/transition/picture effects. Please check all details about valid numbers in [animation effects](#), [transitions](#) and [pictures](#).
- **parameter2** : integer - is used as an input parameter for some animation/transition/picture effects. Please check all details about valid numbers in [animation effects](#), [transitions](#) and [pictures](#).
- **transition** : integer - time of the specified transition in **transition_name** in seconds.
- **transition_name** : string - a transition name - one time animation applied when the LED strip changes its state. Please, see more details on [the transition page](#).
- **picture** : string - is a static light effect applied to an LED strip. It does not change in time as an "effect" does but it takes **color**, **parameter1** and **parameter2** as input values. Please, check more details on [the picture effects page](#).
- **picture_pattern** : array - defines a custom picture. Please, see all details on [the picture effects page](#).
- **white_value** : integer - this parameter is only valid if LEDs with a separate white channel are used (like SK6812 with RGBW LEDs). A valid value is in the range 0..255.

Examples

The examples below help to understand how to use a JSON document sent over MQTT to control the lights. Let's say we want to turn on the LED strip in the red color. This JSON can be sent over MQTT to the controller:

```
{
  "state": "on",
  "color": { "r":255, "g":0, "b":0 }
}
```

To switch off the strip, simply send the following JSON:

```
{
  "state": "off"
}
```

If the LED strip has LEDs with a separate white channel we can light the white LEDs with JSON:

```
{
  "state": "on",
  "white_value": 130
}
```

To adjust brightness of the strip we need to set it in the document like this:

```
{
  "state": "on",
  "color": { "r":255, "g":0, "b":0 },
  "brightness": 120
}
```

The same result can be also achieved by reducing brightness of the red channel only:

```
{
  "state": "on",
  "color": { "r":120, "g":0, "b":0 }
}
```

Animation Effects

Animation - is a dynamic light effect which is "displayed" by an LED strip. It always changes in time. There are several predefined animations listed in a table below. An animation can be set on an LED strip if the json key `effect` is presented. (See json light control data)

Parameters

All animations take input parameters: `parameter1`, `parameter2`, `brightness` and some animations take `color` as an input parameter as well:

- `parameter1` - usually adjusts an animation speed and defines how fast an animation flows. The higher value the faster an animation flows. The valid range is `0..255`. The

default value is 0. If this parameter is not presented in a JSON, the default value will be taken.

- **parameter2** - usually adjusts an animation intensity. The higher value the more intensively animation is. The valid range is 0..255. The default value is 0. If this parameter is not presented in a JSON, the default value will be taken.
- **brightness** - defines max brightness of LEDs on the strip. It works the same way as brightness affects a color on the LED strip. The higher value is the brighter animation is and the higher current your LED strip draws from the power supply. The valid range is 0..255. The default value is 255. If this parameter is not presented in a JSON, the default value will be taken.
- **color** - some animations can change their color depending on the set color in the json or the color which was already set before. Please, take a look at the examples below to see how a color can be set along with an animation effect. The default value is the white color or color which was set last time before.

Supported effects

Animation name	Description
ColorLoop	Input parameters: parameter1, parameter2, brightness
Rainbow	Input parameters: parameter1, parameter2, brightness
DoubleRainbow	Input parameters: parameter1, parameter2, brightness
RainbowChase	Input parameters: parameter1, parameter2, brightness
RunningLights	Input parameters: parameter1, parameter2, brightness
TwoColors	Input parameters: parameter1, parameter2, brightness
ThreeColors	Input parameters: parameter1, parameter2, brightness
ColorPeaks	Input parameters: parameter1, parameter2, brightness
Sparks	Input parameters: parameter1, parameter2, brightness
Comet	Input parameters: parameter1, parameter2, brightness, color
CometWithParticles	Input parameters: parameter1, parameter2, brightness
RandomComets	Input parameters: parameter1, parameter2, brightness
ColorFadings	Input parameters: parameter1, parameter2, brightness

Sparkles	Input parameters: parameter1, parameter2, brightness, color
SparklesOnColor	Input parameters: parameter1, parameter2, brightness, color
SparklesOnColorLoop	Input parameters: parameter1, parameter2, brightness
DarkSparklesOnColor	Input parameters: parameter1, parameter2, brightness, color
RainbowWipesUp	Input parameters: parameter1, parameter2, brightness
RainbowWipesDown	Input parameters: parameter1, parameter2, brightness
ColorWipes	Input parameters: parameter1, parameter2, brightness
Chain	Input parameters: parameter1, parameter2, brightness, color
BrokenLamp	Input parameters: parameter1, parameter2, brightness, color
FastPixels	Input parameters: parameter1, parameter2, brightness
BrightStripes	Input parameters: parameter1, parameter2, brightness, color
DarkStripes	Input parameters: parameter1, parameter2, brightness, color
MulticolorBurst	Input parameters: parameter1, parameter2, brightness
OneColorBurst	Input parameters: parameter1, parameter2, brightness, color
RandomColorBurst	Input parameters: parameter1, parameter2, brightness
ColorPendulum	Input parameters: parameter1, parameter2, brightness
RainbowScan	Input parameters: parameter1, parameter2, brightness
DoubleRainbowScan	Input parameters: parameter1, parameter2, brightness
RandomColor	Input parameters: parameter1, parameter2, brightness
ChaseDown	Input parameters: parameter1, parameter2, brightness, color
ChaseUp	Input parameters: parameter1, parameter2, brightness, color

Chase2Down	Input parameters: parameter1, parameter2, brightness, color
Chase2Up	Input parameters: parameter1, parameter2, brightness, color
SporadicMeteors	Input parameters: parameter1, parameter2, brightness, color
Dots	Input parameters: parameter1, parameter2, brightness
RGBScanner	Input parameters: parameter1, parameter2, brightness
Twinkles	Input parameters: parameter1, parameter2, brightness
PixelQueue	Input parameters: parameter1, parameter2, brightness
PeriodicMeteorsUp	Input parameters: parameter1, parameter2, brightness
PeriodicMeteorsDown	Input parameters: parameter1, parameter2, brightness
Flicker	Input parameters: parameter1, parameter2, brightness, color
MultiplePixelQueues	Input parameters: parameter1, parameter2, brightness, color
MultipleColorPixelQueues	Input parameters: parameter1, parameter2, brightness
Fireworks	Input parameters: parameter1, parameter2, brightness
Fireworks2	Input parameters: parameter1, parameter2, brightness
Strobe	Input parameters: parameter1, parameter2, brightness, color
BigSparks	Input parameters: parameter1, parameter2, brightness
RunAndLightUp	Input parameters: parameter1, parameter2, brightness, color
Moths	Input parameters: parameter1, parameter2, brightness
Breathe	Input parameters: parameter1, parameter2, brightness, color
Pixie	Input parameters: parameter1, parameter2, brightness

Neutrinos	Input parameters: parameter1, parameter2, brightness, color
Emitter	Input parameters: parameter1, parameter2, brightness
BlackHole	Input parameters: parameter1, parameter2, brightness
ColorRunsUp	Input parameters: parameter1, parameter2, brightness
ColorRunsDown	Input parameters: parameter1, parameter2, brightness
LightBars	Input parameters: parameter1, parameter2, brightness
Paintbrush	Input parameters: parameter1, parameter2, brightness, color
Lightning	Input parameters: parameter1, parameter2, brightness
Noise	Input parameters: parameter1, parameter2, brightness
Particles	Input parameters: parameter1, parameter2, brightness
Curtains	Input parameters: parameter1, parameter2, brightness, color
Scanner	Input parameters: parameter1, parameter2, brightness, color

NOTE: Any animation name must be sent to the device exactly how they are listed in the table above. No spaces in the names. A JSON document will be ignored if it has an unsupported animation name.

Examples

The examples below help to understand how to use animation effects. The first example runs the "ColorLoop" effect with a half brightness. parameter1 is specified as 180 and parameter2 is not presented. The default value - zero will be taken in this case.

```
{
  "state": "on",
  "effect": "ColorLoop",
  "parameter1": 180,
  "brightness": 128
}
```

Let's say we set a solid color on the LED strip with the following json:

```
{
  "state": "on",
```

```
"color": { "r":255, "g":0, "b":0 }  
}
```

After that we decided to set an animation effect which takes a color as an input parameter:

```
{  
  "state": "on",  
  "effect": "Flicker",  
  "parameter1": 180,  
  "parameter2": 128  
}
```

Here we didn't specify any color in the JSON. The previous color, which we set before, will be taken as an input parameter or we can explicitly say in which color we want to see this effect:

```
{  
  "state": "on",  
  "effect": "Flicker",  
  "parameter1": 180,  
  "parameter2": 128,  
  "color": { "r":0, "g":0, "b":255 }  
}
```

Transition Effects

Transitions is a one time animation which is applied when an LED strip changes its state. For example, when the LED strip goes off, on or simply changes a color. There are two json keys in a [json light control data](#) which are required to run a transition: `transition_name` and `transition`. The `transition_name` key holds the name of the predefined transition to run (see the supported names below) and the `transition` key specifies the **amount of seconds** which a transition takes. The default transition is "Fade" and is used when the key `transition` is specified. Some transitions are suitable to only turn on an LED strip, some are to turn off and some of them are good to go with any action. Any transition can be interrupted at any time by sending another command to the device. If a transition gets interrupted the end state will be applied immediately.

Parameters of transitions

Some transitions support parameters to adjust animation. All parameters can be passed to the transition over the json keys `parameter1` and `parameter2` in the [json document](#).

Supported transitions

Transition name	Description
-----------------	-------------

Fade	This is the default transition. If the key <code>transition_name</code> is omitted but <code>transition</code> is specified the "Fade" transition effect is applied.
Fold	Doesn't take any parameters except time in the key transition.
Unfold	Doesn't take any parameters except time in the key transition.
Roll	Accepts a value of the key <code>parameter1</code> to adjust animation. The range of the valid values is <code>0..255</code> .
Unroll	Accepts a value of the key <code>parameter1</code> to adjust animation. The range of the valid values is <code>0..255</code> .
Dots	This transition fills the strip with dots with a fixed distance between two lit LEDs. Each gap between two lit LEDs is filled sequentially (Unlike the "Curtains" transition). The <code>parameter1</code> defines the distance between two lit LEDs. It accepts a value in the range <code>0..255</code>
FadeOut	This is a special transition which can be used to only switch off the LEDs. The difference between the "Fade" transition is that "Fade" can only switch all LEDs off from a solid color. "FadeOut" can fade all LEDs gradually even if all LEDs have different colors without changing their colors. This transition is good to use to switch off the LED strip after an animation effect or a picture , for instance.
SinIn	This transition is more suitable to turn the LED strip on. With switching off it will reach the end point (all LEDs are off) immediately so no transition will be applied. To turn the LEDs off the transition "SinOut" shall be used instead.
SinOut	This transition is more suitable to turn the LED strip off. With switching on it will reach the end point (all LEDs are on) immediately so no transition will be applied. To turn the LEDs on the transition "SinIn" shall be used instead.
Paintbrush	The transition takes <code>parameter1</code> as an input parameter as well as <code>transition</code> itself which holds the time in seconds which the transition takes.

Curtains

This is similar to "Dots" but it fills all gaps simultaneously unlike "Dots" which does it sequentially. As an input takes parameter1 in the range 0..255.

NOTE: The device is case sensitive to all transition names. All names must be passed exactly how they are presented in the table above.

Examples

Here are some examples which can help to better understand how transitions work. First, let's say the LED strip is on and we want to switch it off by fading all LEDs down slowly in 10 seconds. In this case we need to send the following json document over [mqtt](#) as a payload shown below:

```
{
  "state": "off",
  "transition": 10
}
```

If we want to switch the lights on with another transition, for instance, "Unroll", a json document shown below shall be sent to the controller:

```
{
  "state": "on",
  "transition_name": "Unroll",
  "transition": 10,
  "color": {"r": 255, "g": 17, "b": 190}
}
```

If we want to dim the lights slowly in 15 seconds from the maximum brightness to half a json document must be sent to the device:

```
{
  "state": "on",
  "transition_name": "Fade",
  "transition": 15,
  "brightness": 128
}
```

Another example shows how to slowly change a color of LED strip in 5 seconds:

```
{
  "state": "on",
  "transition_name": "Fade",
  "transition": 5,
  "color": {"r": 0, "g": 0, "b": 255}
}
```

With the example below we can emulate a sunrise on a long LED strip which lasts for 10 minutes:

```
{
```

```

    "state": "on",
    "transition_name": "SinIn",
    "transition": 600,
    "color": {"r": 255, "g": 128, "b": 10}
}

```

NOTE: The key state must be always specified to let the controller understand the next state.

NOTE: If a transition name is specified with the key transition_name but transition is not it will take a default value 0 (zero) which makes any transition invisible therefore transition must be always provided if a transition is required.

Pictures

Pictures - is a static pattern which is "displayed" by an LED strip. It never changes in time like [animations](#). There are several predefined pictures listed in a table below. A picture can be set on an LED strip if the json key picture is presented. (See [json light control data](#))

Supported pictures

Picture name	Description
Rainbow	It displays a rainbow over the entire strip. The value of parameter1 defines a color in the beginning of the displayed rainbow. The valid values are in the range 0..255.
Rainbow2	It displays two rainbows over the entire strip. The value of parameter1 defines a color in the beginning of the displayed rainbow. The valid values are in the range 0..255.
Rainbow3	It displays three rainbows over the entire strip. The value of parameter1 defines a color in the beginning of the displayed rainbow. The valid values are in the range 0..255.
Rainbow4	It displays four rainbows over the entire strip. The value of parameter1 defines a color in the beginning of the displayed rainbow. The valid values are in the range 0..255.
Dots	It lights up every Nth LED on a strip in a certain color. If the parameter parameter1 is specified it defines a distance between two lit LEDs as a percentage of all LEDs on the strip. This means that when parameter1 takes its max value (equal 255) only two LEDs on the edges of the strip will be on. If you need to have a certain distance (counted in amount of LEDs) between two lit LEDs, the parameter parameter2 should be used

instead. The max LEDs between two lit LEDs, in this case, are 255 as the max value of parameter2.

Stripes

This draws stripes on the controlled LED strip. The parameter1 defines the length of every stripe and the parameter2 defines the gap between two stripes. If both parameters are equal zero, for instance, it will light up every second LED: The minimum length (when parameter1=0) of the stripe is 1 pixel and the minimum gap (when parameter2=0) is 1 pixel as well.

ProgressBar

This draws a progress bar on the LED strip. The entire strip is taken as 100%. The parameter1 defines a value of the progress. The valid range is 0..100. Higher values will be taken as 100. The parameter2 defines the direction of the progress bar. If parameter2=0, it goes up (begins from the side where the signal wire comes in). A non zero value will start the progress bar from the other side. color and brightness define the color of the progress bar.

Noise

It fills the strip with a random color in every pixel. As an input the effect takes parameter1 and brightness. parameter1 changes the saturation of all colors from 30% (when parameter1=255) to 100% (when parameter1=0). brightness is in the range 0..255

Pattern

This picture allows a user to specify their own pattern to be drawn on an LED strip. It requires an extra key in the json document - picture_pattern which holds the custom pattern itself. Please, see the description below.

NOTE: The device is case sensitive to all transition names. All names must be passed exactly how they are presented in the table above.

Custom pattern

A custom static picture can be defined in a json document sent to the controller. For this purpose two json keys must be provided: picture and picture_pattern. For a custom picture the parameter picture must take the value Pattern and picture_pattern shall be an array of intervals of LEDs in a certain color. Currently 20 intervals are supported to be parsed. Please, see a json example below:

```
{
  "state": "on",
  "brightness": 128,
  "picture": "Pattern",
  "picture_pattern": [
    {
      "leds": 20,
      "c": {"r": 255, "g": 0, "b": 0}
    }
  ]
}
```

```

    },
    {
      "leds":20,
      "c":{"r":255,"g":255,"b":0}
    },
    {
      "leds":20,
      "c":{"r":0,"g":0,"b":0}
    }
  ]
}

```

Every interval must contain:

```

{
  "leds":17,
  "c":{"r":200,"g":0,"b":150}
}

```

leds - amount of LEDs which an interval takes

c - the color to be displayed on the interval. No extra keys are supported in an interval.

Examples

The examples below help to understand how to use pictures. For instance to draw a static rainbow on an LED strip, we simply send the following json document to the controller over [MQTT](#):

```

{
  "state":"on",
  "picture":"Rainbow",
  "parameter1":150,
  "brightness":128
}

```

Light up every 6th led in a blue color with brightness 80% :

```

{
  "state":"on",
  "picture":"Dots",
  "parameter2":5,
  "color":{"r":0,"g":0,"b":255},
  "brightness":204
}

```

The example below shows how to fill an LED strip with 3 different colors. If the strip has 60 LEDs, the json document will look like:

```

{

```

```

"state": "on",
"picture": "Pattern",
"picture_pattern": [
  {
    "leds": 20,
    "c": {"r": 255, "g": 0, "b": 0}
  },
  {
    "leds": 20,
    "c": {"r": 0, "g": 255, "b": 0}
  },
  {
    "leds": 20,
    "c": {"r": 0, "g": 0, "b": 255}
  }
]
}

```

Or if we want to display only 2 yellow pixels in the middle, we can send the following json to the controller:

```

{
  "state": "on",
  "picture": "Pattern",
  "picture_pattern": [
    {
      "leds": 29,
      "c": {"r": 0, "g": 0, "b": 0}
    },
    {
      "leds": 2,
      "c": {"r": 255, "g": 255, "b": 0}
    },
    {
      "leds": 29,
      "c": {"r": 0, "g": 0, "b": 0}
    }
  ]
}

```

Firmware

Every device has a certain version of installed software. A pixie device reports its version in an attribute MQTT message (please see [MQTT page](#) for the details). Another way to check the current version is on the [WebUI](#) on the info page.

Network Status

WiFi	Connected (Galaxy)
MQTT	Connected

Firmware

Firmware version	0.8.5
Available version	No information <button>Check update</button>

Device

Developer	IoT-Hus14
MCU	ESP32 (rev:1)
Flash size	16MB
MAC Address	98:F4:AB:98:E4:E4
IP Address	192.168.1.182
Uptime	1T00:44:40
Board Temperature	30.1C (86.18F)

Links

The device is powered by [FreeRTOS](#).

The UI is created with a CSS toolkit [Siimple](#).

Device [wiki](#).

Reboot

OTA

If there is an update available on the server the firmware can be updated over the air. In this case the device must be connected to a WiFi network with internet access.

Network Status

WiFi	Connected (Galaxy)
MQTT	Connected

Firmware

Firmware version	0.8.5
Available version	0.9.0

[Check update](#)[Update](#)

Device

Developer	IoT-Hus14
MCU	ESP32 (rev:1)
Flash size	16MB
MAC Address	98:F4:AB:F4:E4:AB
IP Address	192.168.1.182
Uptime	1T00:44:40
Board Temperature	30.1C (86.18F)

Links

The device is powered by [FreeRTOS](#).

The UI is created with a CSS toolkit [Siimple](#).

Device [wiki](#).

[Reboot](#)

To check if any update is available, open the info page and click the ["Check update"](#) button. It will show an additional button ["Update"](#) to install a newer version of the software if there is any available on the server. The device will download the available firmware and install it. After restart it will run the latest firmware. Another way to do it is over MQTT. Please check the details on the [MQTT page](#).

Factory Reset

The board can be reset to factory settings. It will erase all settings done before and set the factory version of software to boot. To perform a factory reset, restart your board with the reset button on the board 10 times in a row. The interval between restarts must not exceed 2 seconds and at the same time the interval must not be too short to let the controller start booting. The best way to perform that:

- Hit the reset button
- The board will turn on both status LEDs for 100 ms to boot
- Wait when LEDs will go off and hit the reset button again.
- Repeat this 10 times.
- After 10 restarts the status LEDs will start alternately flashing quickly. This means the controller is erasing all data. In a couple of seconds the device will boot again and will configure a WiFi Access Point.

Home Assistant Integration

The Pixie controller can be integrated in Home Assistant quite easily with a custom component. The integration provides almost full functionality for a pixie device: set colors, animation effects, picture effects, transition effects, select entities of available pictures and animations, sensors of the board temperature and uptime.

Detailed information about the integration can be found at the integration Github page:

<https://github.com/iothus14/Pixie-Home-Assistant-Integration>