Kuando Busylight LoRa and The Things Network: Getting started

Table of Contents

The Things Network allows the addressing of the Busylight LoRa devices with https – requests.

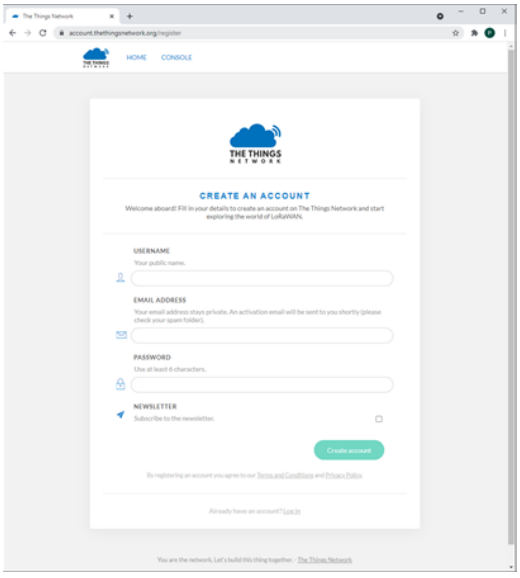
We will describe how to send commands to your LoRa Busylight in various ways.

For demonstration, we will use “The Things Network Community Edition”. Please note that the Community Edition may delay your Busylight control commands.

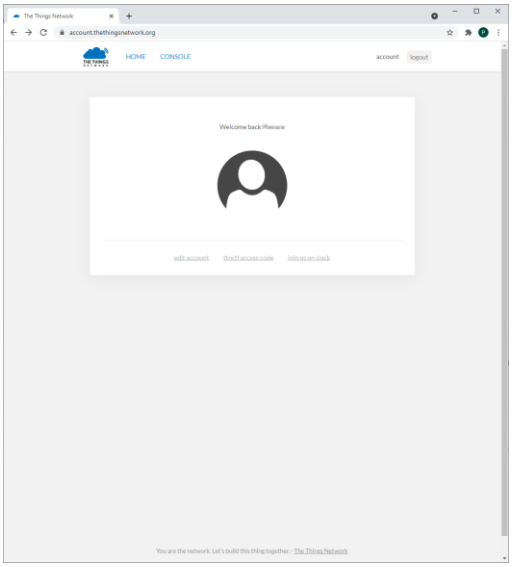
Create a TTN account and application

The first step is creating a user for TTN: Open a browser and go to
https://www.thethingsnetwork.org/

In the right upper corner, you will find a Button “Sign Up”.

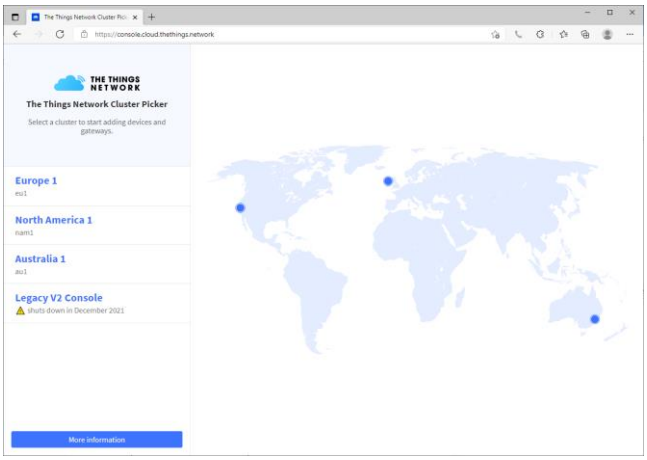


After filling the form, there will be an email validation. After Validation, you are logged in.

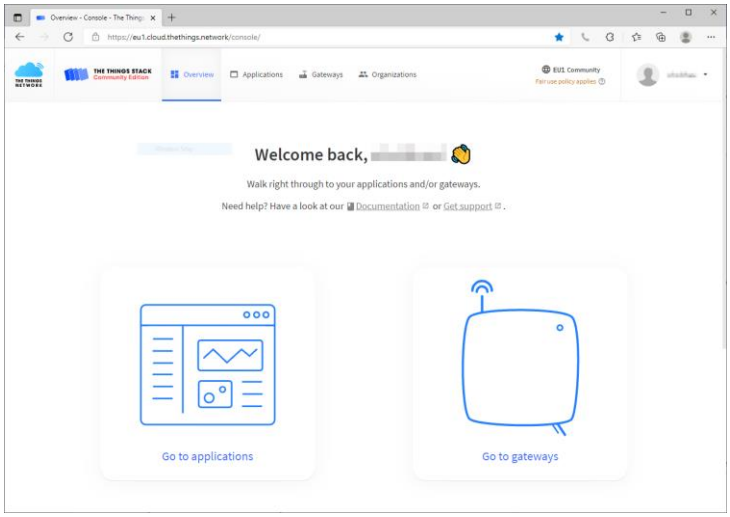


Please click on “Console”.

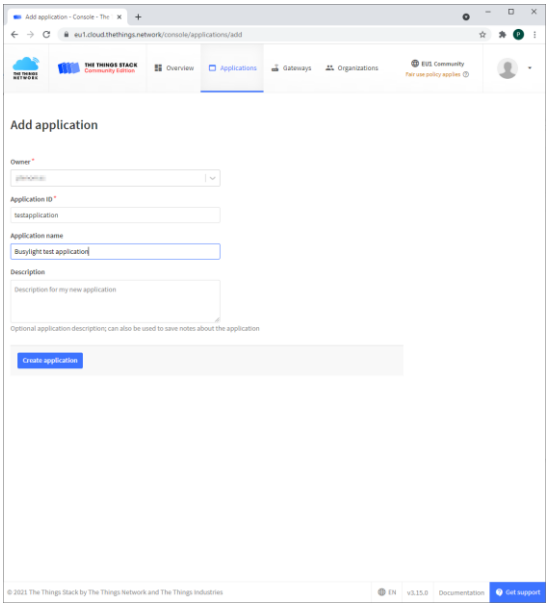
Now, please select the appropriate network cluster for you, typically the nearest.



After selecting, you enter the TTN Console:



Now, klick on Applications – Add Application



Please note the applicationid, you will need it later.

Registering you Gateway

Typically, the procedure to add a LoRa Gateway to TTN is very good described in the gateway documentation, and for the most common gateways as well in the TTN documentation.

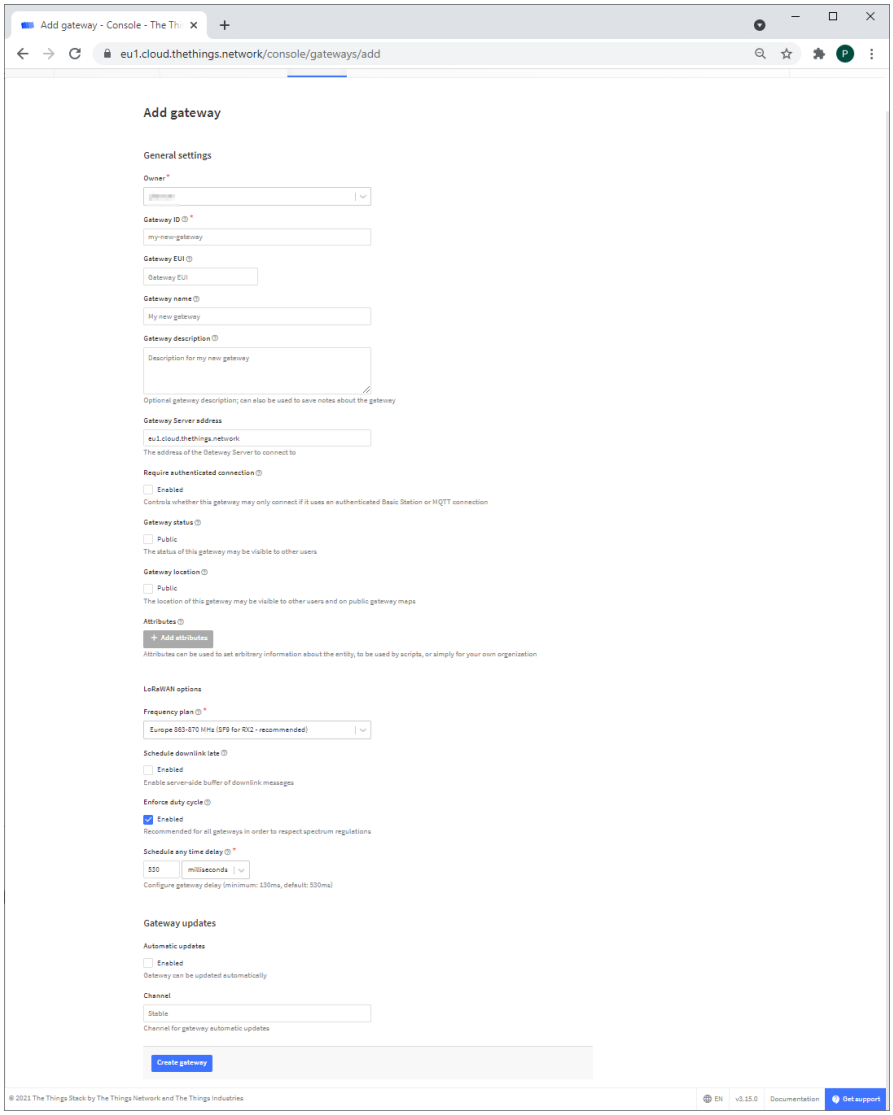
You can find the TTN documentation here:

https://www.thethingsnetwork.org/docs/devices-and-gateways/adding-gateways/

Here, we show a simple gateway using the Semtech UDP Packet Forwarder.

To register the gateway, you need to know the gateway EUID.

In the TTN console, lick on Gateways, then klick Add gateway.



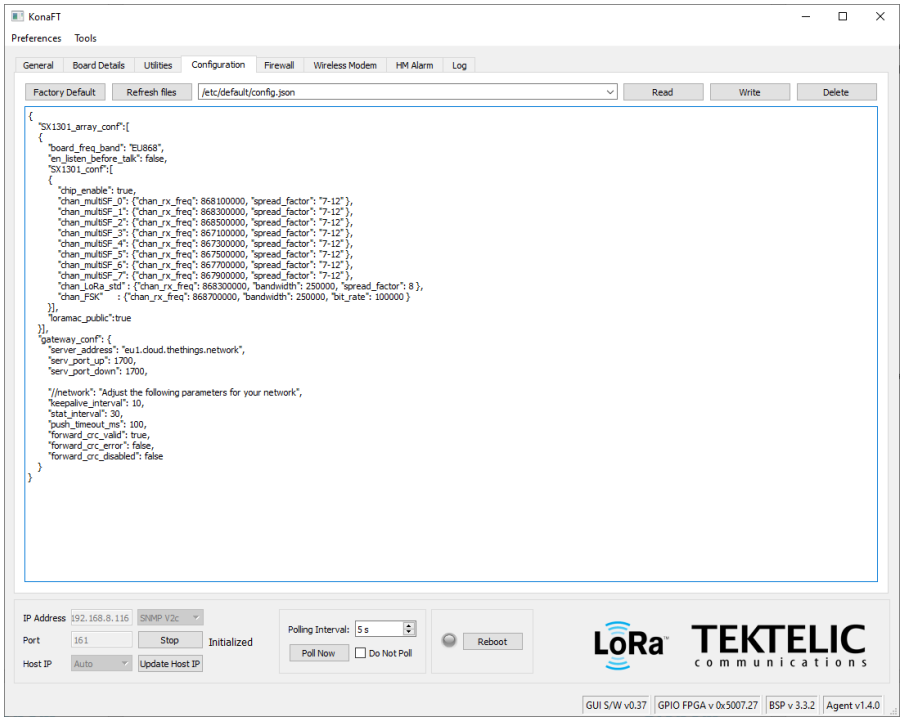
Please select the appropriate values for Frequencies for your country.

In the gateway configuration, you need to enter the server address which is shown here in the registration form.

Example: For a Tektelic gateway, you need to enter the server FQDN in the file /etc/default/config.json.

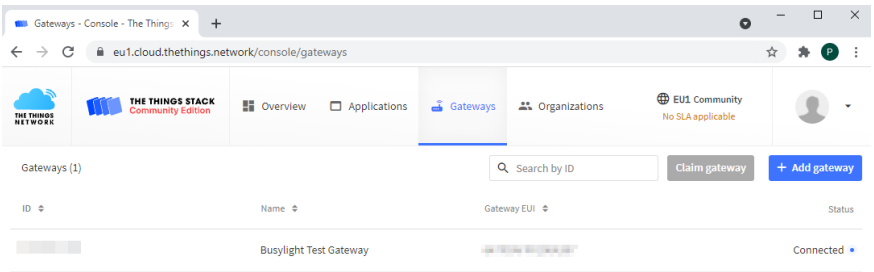
The server address needs to be adjusted.

Here you see an example for the European TTN community cloud:



After changing and writing the content of the file, you need to restart the packet forwarder (or the gateway).

If everything is done correctly, you can see the gateway in status connected.



Registering your Busylight

For registering your Busylight device, you need these information:

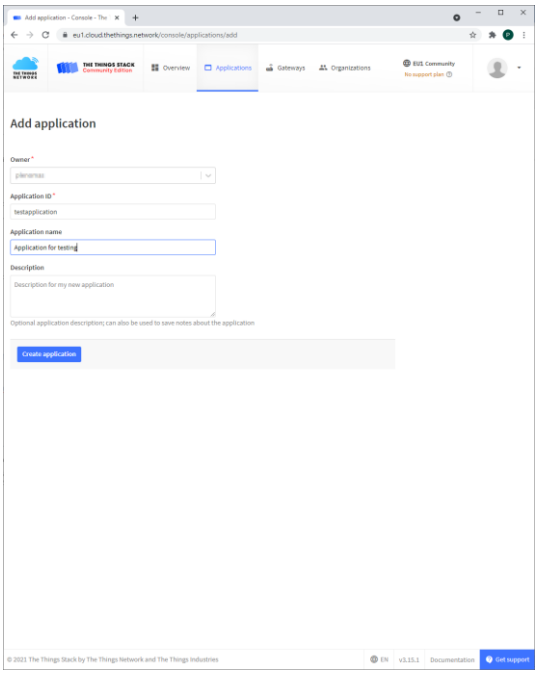
DeviceEUI (8 Byte Hex)

AppEUI (8 Byte Hex)

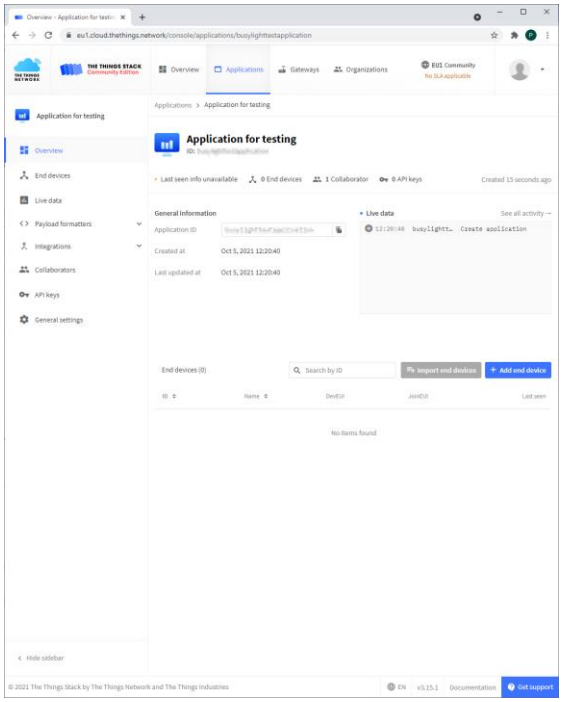
AppKey (16 Byte Hex)

If you have already created an application inside TTN, you can skip the next step.

If you do not have an application, please create one using the “+ Add Application Button.”

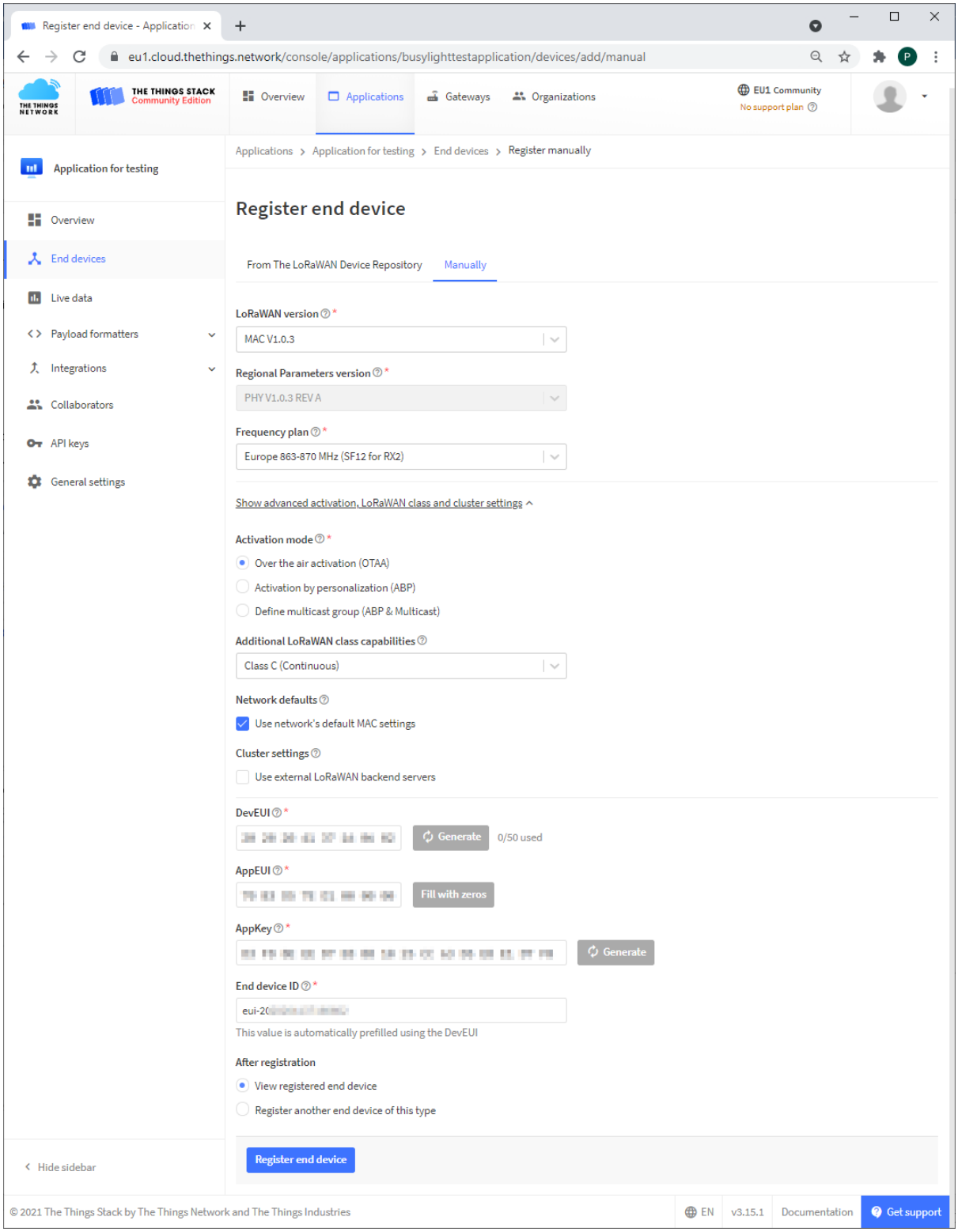


Please enter the application by clicking it.

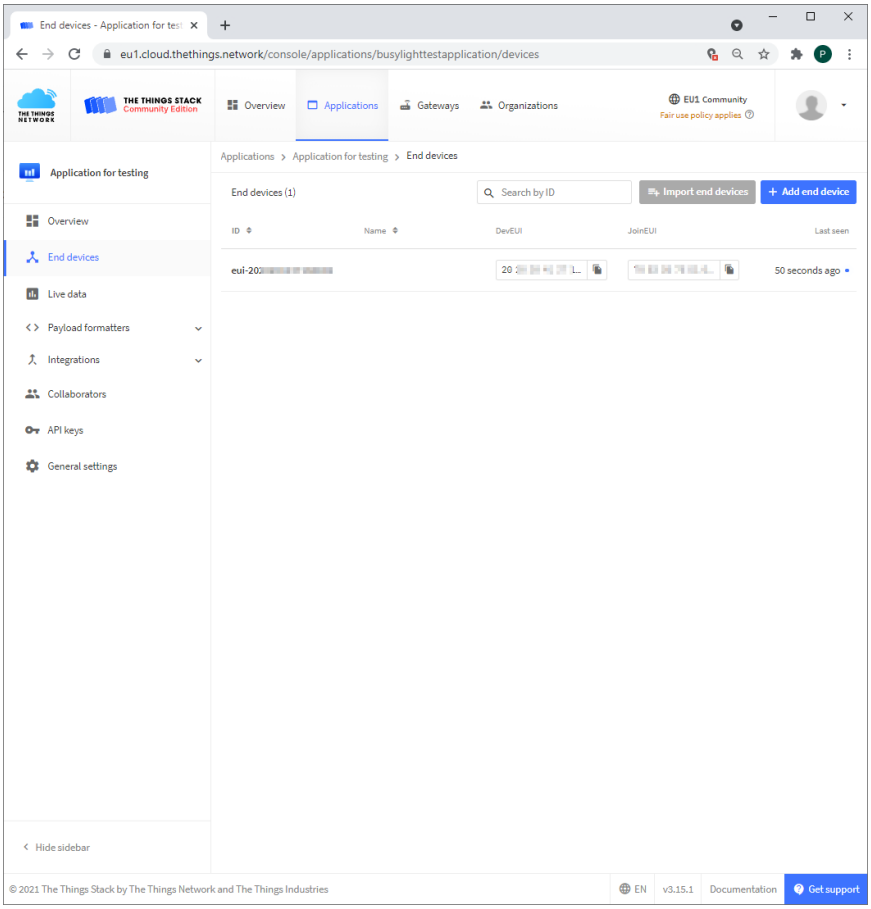


Please click “+ Add end device” now and click on “Manually”.

Please fill the form like this, using the appropriate frequency plan for your location:

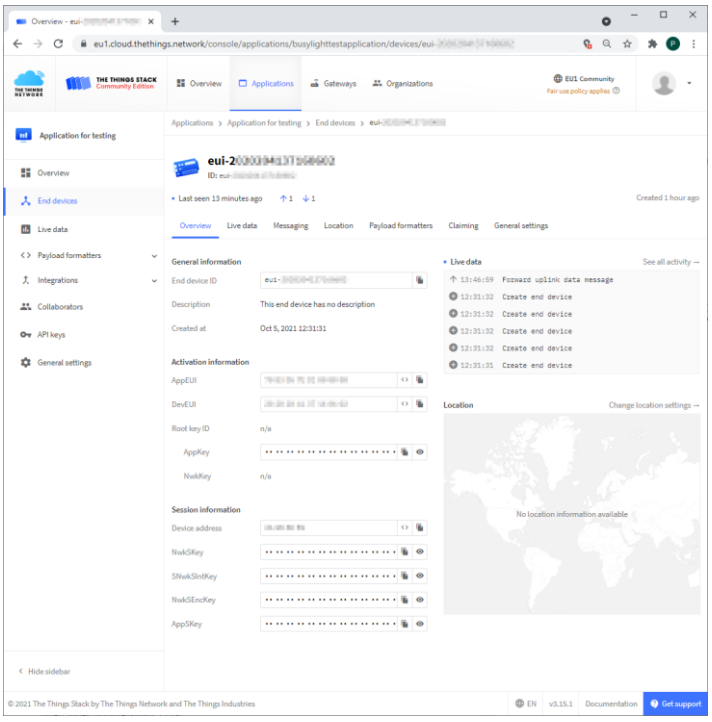


After registering and everything OK, the device will be shown as connected:

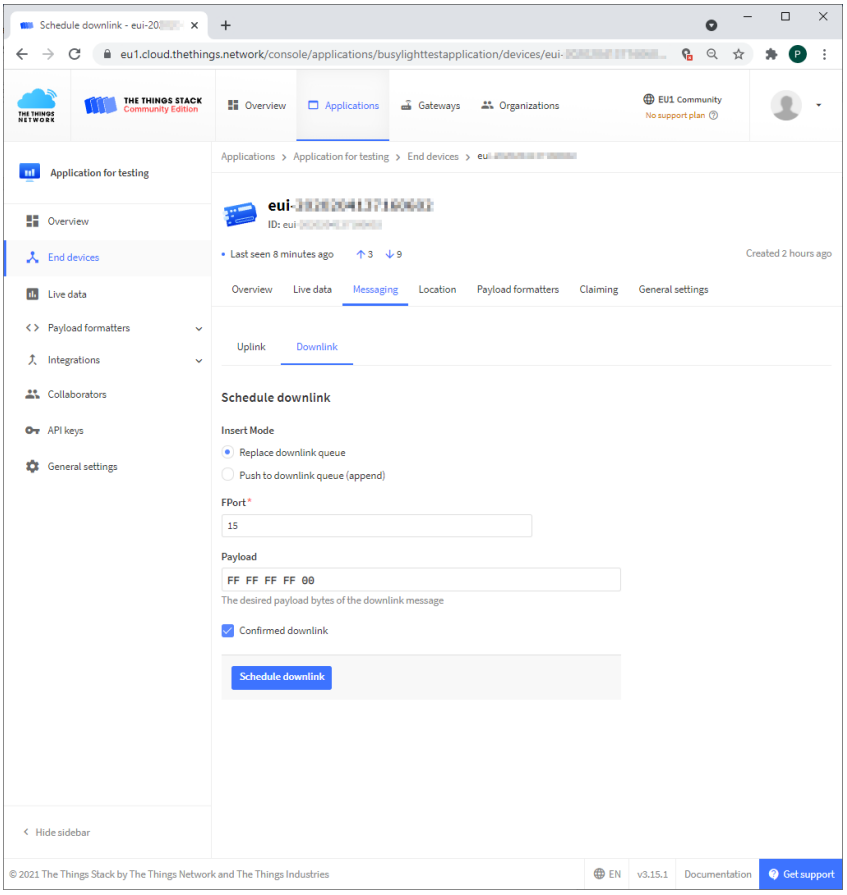


Testing the Busylight

You can test your connected Busylight by sending the downlink payload from the TTN console. To do that, please open the device details page.



Here, please klick on Messaging – Downlink:



The Fport needs to be set to 15. They Payload needs to be specified in Hex notation. Please have a look to the chaper about the hardware payload format.

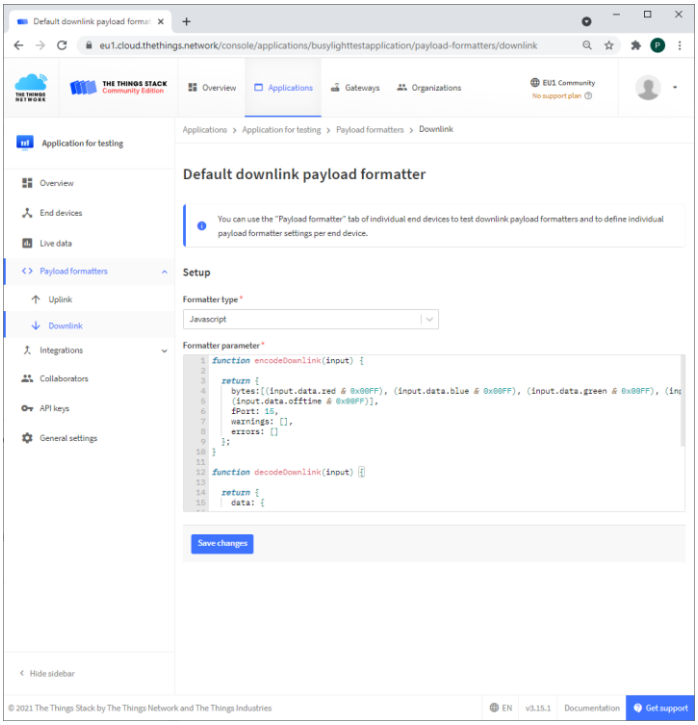
In this case, the Busylight will be solid white.

TTN Payload formatter for the Busylight

A Payload formatter gives you the ability to send human-readable json strings to control the Busylight instead of the raw hardware bytes as described in the Busylight LoRa Hardware Payload format chapter.

To insert Payload formatter, open the application and klick on the “Payload Formatter – Downlink” menu.

Please select “Javascript” as the Formatter Type and enter the source into the Formatter parameter field.



Here is the complete formatter:

```
function encodeDownlink(input) {

  return {
    bytes:[(input.data.red & 0x00FF), (input.data.blue & 0x00FF), (input.data.green
& 0x00FF), (input.data.ontime & 0x00FF),
    (input.data.offtime & 0x00FF)],
    fPort: 15,
    warnings: [],
    errors: []
  };
}

function decodeDownlink(input) {

  return {
    data: {
      red: input.bytes[0],
      green: input.bytes[2],
      blue: input.bytes[1],
      ontime: input.bytes[3],
      offtime: input.bytes[4]
    },
    warnings: [],
    errors: []
  }
}
```

For decoding the uplink messages, you can use this uplink decoder:

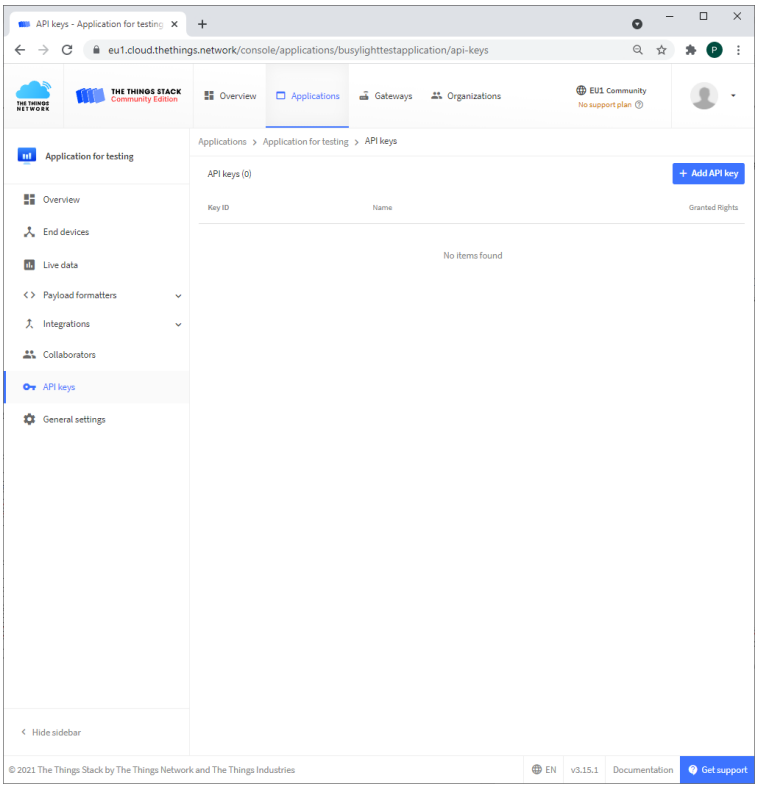
```
function decodeUplink(input) {
  if (input.bytes.length == 24)
  {
  return {
    data: {
      RSSI: byteArrayToLong(input.bytes, 0),
      SNR: byteArrayToLong(input.bytes, 4),
      messages_received: byteArrayToLong(input.bytes, 8),
      messages_send: byteArrayToLong(input.bytes, 12),
      lastcolor_red: input.bytes[16],
      lastcolor_blue: input.bytes[17],
      lastcolor_green: input.bytes[18],
      lastcolor_ontime: input.bytes[19],
      lastcolor_offtime: input.bytes[20],
      sw_rev: input.bytes[21],
      hw_rev: input.bytes[22],
      adr_state: input.bytes[23]
    },
    warnings: [],
    errors: []
  };
  }
  else
  {
    return {data: {
      bytes: input.bytes,
    },
    warnings: [],
    errors: []
  }
  }
}


byteArrayToLong = function(/*byte[]*/byteArray, /*int*/from) {
    return byteArray[from] | (byteArray[from+1] << 8) | (byteArray[from+2] << 16) |
(byteArray[from+3] << 24);
};
```

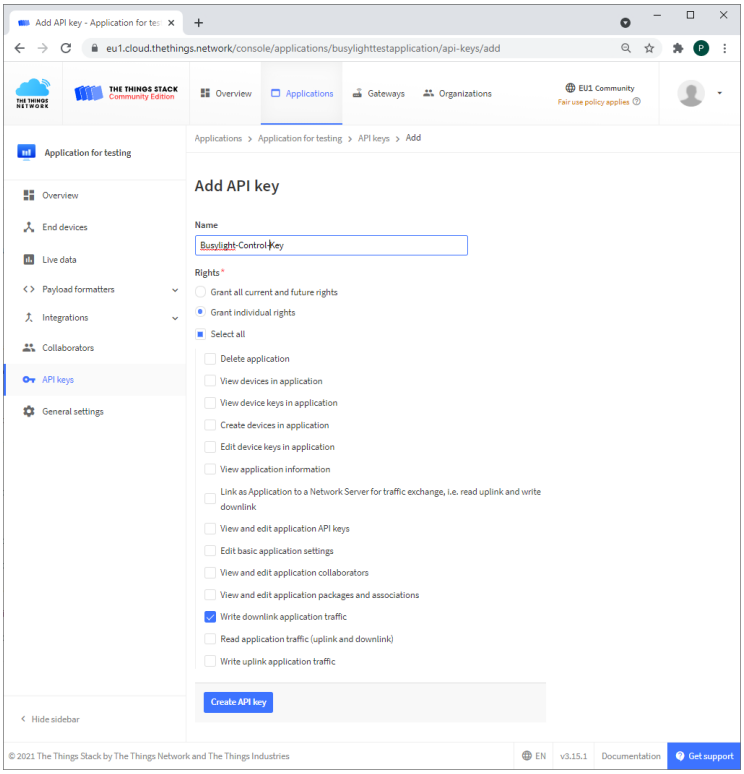
Controlling the Busylight with http Requests

If you plan to control the Busylight with http/https requests, you need to create an API Key.

To create a key, open the application and click on “API Keys”.

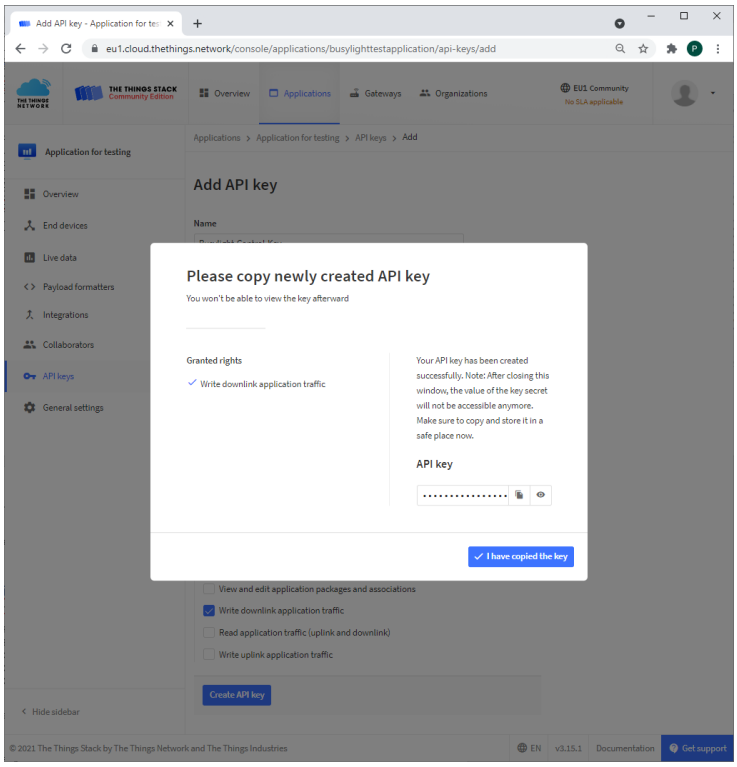


Then klick on “+ Add API Key”.



Give a reasonable name and grant the right to write downlink traffic.

After saving, you need to copy the key for your application. Please be aware that you cannot access it a second time!!



You are now ready to write control your Busylight with a HTML request. Please change the yellow text parts to your needs.

You need to send a POST request to this URI:

https://your_ttn_server/api/v3/as/applications/your_application_id/devices/your_device_id/down/push

You send this Body:

```
{
   "downlinks": [{
  "decoded_payload": {
          "red": 0,
          "green": 0,
          "blue": 255,
          "ontime": 255,
          "offtime": 0
      },
      "f_port": 15
      }]
  }
```

And you need to send these Headers:

```
Authorization: Bearer your_api_key
Content-Type: application/json
User-Agent: busylight/v1
```

Here is a PowerShell example that switches a Busylight to solid blue:

```
$cmd2 = '{
 "downlinks": [{
"decoded_payload": {
        "red": 0,
        "green": 0,
        "blue": 255,
        "ontime": 255,
        "offtime": 0
    },
    "f_port": 15
    }]
}'

Invoke-WebRequest -Method 'POST' `
    -Headers @{"Authorization" = "Bearer NNSXS.[redacted]FA"; `
              "Content-Type" = "application/json"; `
             "User-Agent" = "busylight/v1"` 
              } `
    -Body $cmd2 `
    -Uri https://eu1.cloud.thethings.network/api/v3/as/applications/[redacted]/devices/eui-[redacted]/down/push
```

Busylight LoRa Hardware Payload format

The Busylight expects a 5-byte binary payload for switching the colors.

Byte 0: Red Color intensity (0..255)
Byte 1: Blue Color intensity (0..255)
Byte 2: Green Color intensity (0..255)
Byte 3: On Steps (0..255)
Byte 4: Off Steps (0..255)

Example for blue static light:

Byte[0]=0
Byte[1]=255
Byte[2]=0
Byte[3]=255
Byte[4]=0

The Hex form will be: 00FF00FF00

For TTN https operating, if using the frm_payload to specify the payload for the end device, the byte array needs to be send as a base64 encoded string.

When using the payload formatter, you can specify the values using a json string:

```
{
   "downlinks": [{
  "decoded_payload": {
          "red": 0,
          "green": 0,
          "blue": 255,
          "ontime": 255,
          "offtime": 0
      },
      "f_port": 15
      }]
  }
```