# OpenHAB3 & NSPanel

Installation and configuration guide

Alf Pfeiffer, 2022-03-27

## Table of content

# 1. Overview

This documentation describes the installation steps for how to flash a Sonoff NSPanel with Tasmota firmware and then to connect it to a OpenHAB3 system. The setup also assumes you would like to get weather information on the start panel.

I've read (and reread) all the posts on this topic on the OpenHAB forum and my finding is that most people – just like me – get stuck on 1. Get NSPanel to "talk" to OpenHAB and 2. Configuring the panels (screens) in NSPanel. For quick answer on 1, check picture in chapter **Fel! Hittar inte referenskälla.**.

Components used for the setup:

- A Windows PC to do the work on
- Raspberry Pi (minimum 3, recommended 4)
- A USB Serial Adapter
- Some cables to connect the USB serial adapter to the circuit board of the NSPanel.
- Sonoff NSPanel EU
- OpenHABian (v1.7.2), components needed:
    - Binding: **MQTT Binding**
    - Binding: **OpenWeatherMap Binding**
    - Add-on: **JSONpath Transformation**
    - Add-on: **RegEx Transformation**
    - Automation: **Groovy Scripting**
- Mosquitto MQTT broker (included in OpenHABian)
- Openweathermap cloud service

## Disclaimer

**Use this documentation at your own risk!** The author assumes no responsibility of any mishaps resulting in your use of this documentation.

## Acknowledgements

m-home (Mike) – For his initiative and appreciated efforts to bring NSPanel to OpenHAB

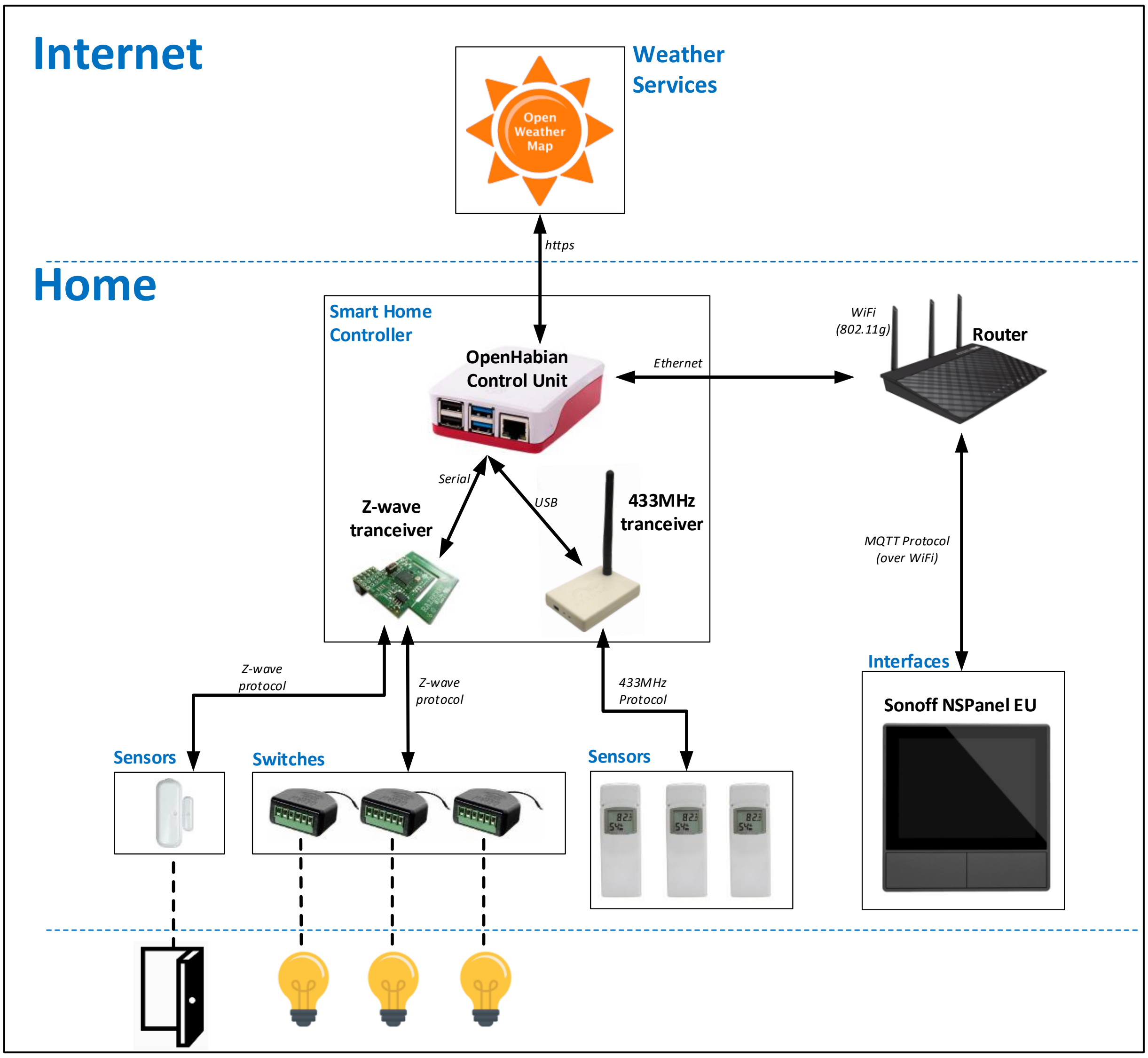
Blakadder – For creating a Tasmota firmware for NSPanel

Lewis Barclay – Especially this video which is the source for my flashing documentation (I actually suggest you use this for the flashing part and use my documentation only as a reference).

## Hardware and Protocols

The picture below shows a typical openhabian setup with a control unit connected to underlying hardware (switches, sensors, interfaces) and external services (openweathermap). The documentation will focus on the NSPanel setup and assume you have a running openhabian system (OpenHAB 3) and your other hardware is already configured and available in openhabian.

I also assume you are accustomed to OpenHAB and its concepts such as items, things, channels, etc.



## Documentation approach

The key aim in this documentation is to answer the question “what should I do” with a spice of “how does it work” whenever there is some understanding needed hampering the first question.

I’m also assuming that you want to display weather information on the panel.

This guide is covering the following steps:

- Install and configure openweathermap
- Install and configure Mosquitto MQTT broker
- Flashing Sonoff NSPanel with Tasmota
- Post configuration of Tasmota on NSPanel
- Base setup of NSPanel-to-OpenHAB communication (make NSPanel talk to openhab and customize the first screen)
- Custom panel configuration – The fun part where you design the layout and connect the control of your devices to NSPanel.

Each step is described in a separate chapter. Each chapter starts with links to sources and other relevant information.

# 2. Install and configure OpenWeatherMap

If you do not want weather information on the start panel or use another service, just skip this step.

OpenWeatherMap is a cloud service providing weather forecasts based on your location. There is an OpenWeatherMap binding that calls the OpenWeatherMap API making the setup and use in OpenHAB very straight forward.

## Links and references

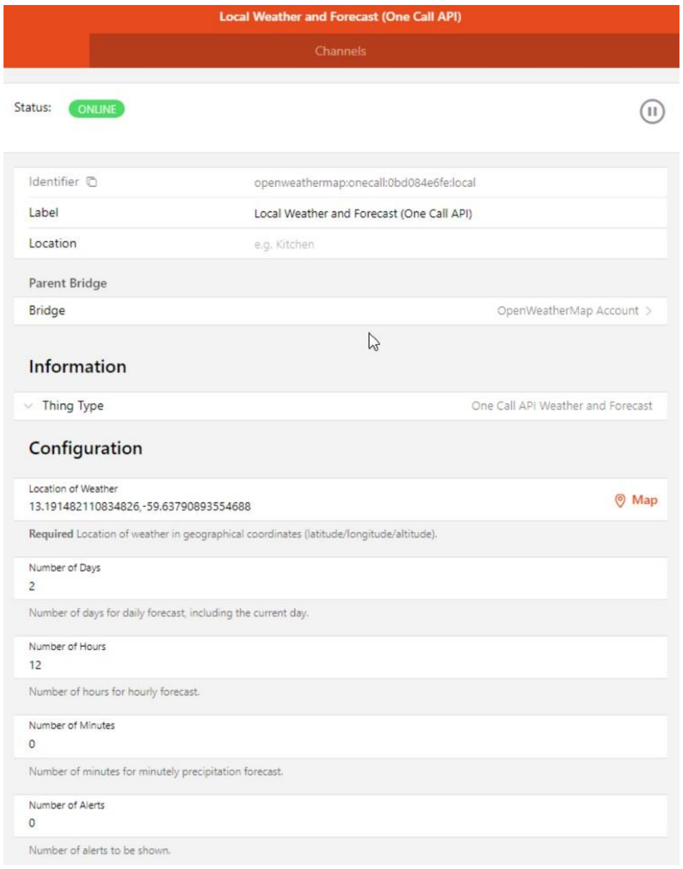
- Link to OpenWeatherMap service: https://openweathermap.org

## Installation and configuration

Very intuitive steps but describing this anyhow for completeness.

- Get API key from OpenWeatherMap
  - Browse to https://openweathermap.org and create an account
  - Select: **API keys**
  - Select: **Generate**
  - API Key: **y2)uc2a7cae3d54037563f30r2e0637cp** (**example**; you will get another key)
  - This key will be entered in the OpenWeatherMap account item next step.
- Configure Your OpenHAB
  - Install: **OpenWeatherMap binding**
  - Select: **Settings**
  - Select: **Things** and press ”+”
  - Select: **OpenWeatherMap Binding**
  - Select: **OpenWeatherMap Account** (this is just to store your API key)
  - Enter your API key: **y2)uc2a7cae3d54037563f30r2e0637cp**
  - Select: **Save** (top right)
  - It takes a while - hour(s) - for your API key being registered and provisioned to be usable, so the status of this thing will be **red** until this has happened – so no alarm.
  - Next step is to create the **Local Weather and Forecas (One Call API)** thing which will be the one you actually will be using
  - Select: **Things** and press ”+”
  - Select: **OpenWeatherMap Binding**
  - Select: **Local Weather and Forecast (One Call API)**
    - As Bridge; Select: **OpenWeatherMap Account**
    - As Location of Weather; Enter: **<your coordinates>**
    - As Number of Days; Enter: **2** (2=today and tomorrow. You can of course change this but as the NSPanel has only one small piece of the primary display for weather forecasts. I was primarily interested in tomorrow’s weather. So this reduces the number of channels in the created item to what

I'm interested in – will be a lot anyway...).



Local Weather and Forecast (One Call API)

Channels

Status: ONLINE

| | |
|---|---|
| Identifier | openweathermap:onecall:0bd084e6fe:local |
| Label | Local Weather and Forecast (One Call API) |
| Location | e.g. Kitchen |

Parent Bridge

| | |
|---|---|
| Bridge | OpenWeatherMap Account |

Information

| | |
|---|---|
| Thing Type | One Call API Weather and Forecast |

Configuration

Location of Weather
13.191482110834826,-59.63790893554688

**Required** Location of weather in geographical coordinates (latitude/longitude/altitude).

Number of Days
2

Number of days for daily forecast, including the current day.

Number of Hours
12

Number of hours for hourly forecast.

Number of Minutes
0

Number of minutes for minutely precipitation forecast.

Number of Alerts
0

Number of alerts to be shown.

  - Select: **Save** (top right)
  - Also this thing will also have a status of red until your API key is provisioned, so don't worry...
- This concludes the preparations.

# 3. Install and configure Mosquitto MQTT Broker

## MQTT overview

MQTT is a standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth.

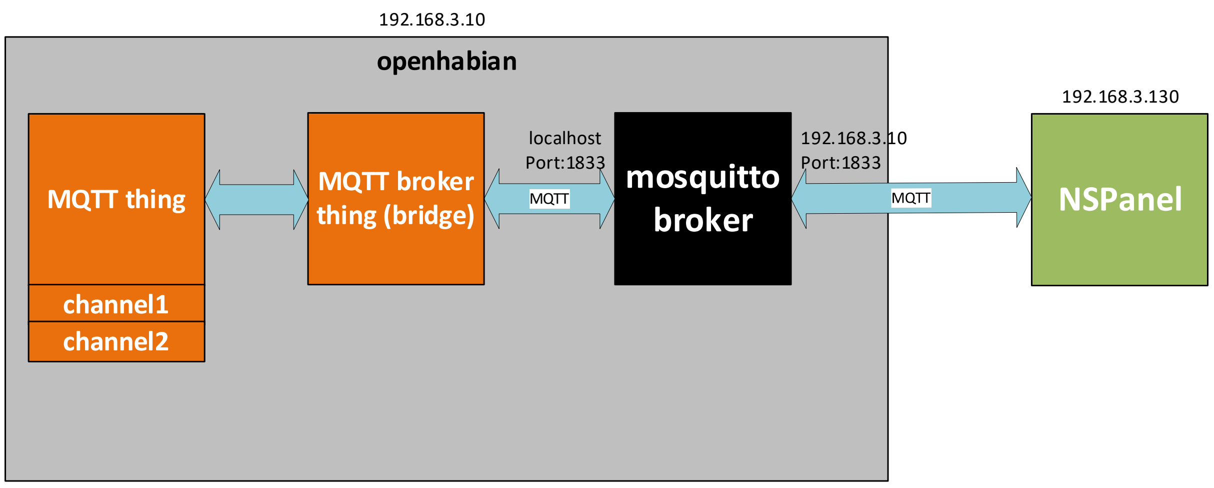
## Links and references

- General MQTT overview https://www.instructables.com/MQTT-on-Openhab-3-Tutorial/

## Installation and configuration

This chapter will only cover the basic MQTT setup. The actual integration of OpenHAB with NSPanel is described in chapter **Fel! Hittar inte referenskälla.**.

The picture below shows a generic MQTT setup for OpenHAB. The NSPanel device will communicate with the **Mosquitto broker** which in turn communicates with the thing **MQTT broker thing** (bride) which in turn is tied to your actual **NSPanel MQTT thing**. (IP' are of course mine, you will have others..). Once configured, the **MQTT broker thing** and **Mosquito broker** do not need to be touched anymore and will support most of your MQTT use cases 😊.



1. Install Mosquitto – This is a "MQTT broker" coming with the openhabian image, steps are:
   a. Log on your openhab with putty (or any other ssh client)
   b. Run command: **sudo openhabian-config**
   c. Select: **20 Optional Components**
   d. Select: **23 Mosquitto**
   e. Username will be **openhabian** (**Note!** remember this, username and password needs to be entered in both the **NSPanel device** and the **MQTT broker thing** bridge)
   f. Enter the password: **mqttpwd22??**
   g. The Mosquitto broker will now start and listen for traffic on port 1883
2. Base configuration of the MQTT broker thing (bridge)
   a. Log on as admin in the OpenHAB web interface. First we need to install some required components:
      i. Select: **Settings** in the menu
      ii. Select: **addons** and install "JSONpath Transformation" (This is needed to do JSON transformations in a Channel definition)
      iii. Select: **addons** and install "RegEx Transformation" (This is needed to do regex-selections on a JSON response in a Channel definition)

iv. Select: **bindings** and install ”MQTT Binding”

b. Select: **Things** and press ”+”

c. Select: **MQTT Broker** (this is just a bridge between your MQTT things and the Mosquitto broker)

d. Select: **Add manually**

e. Select: **MQTT Brooker**

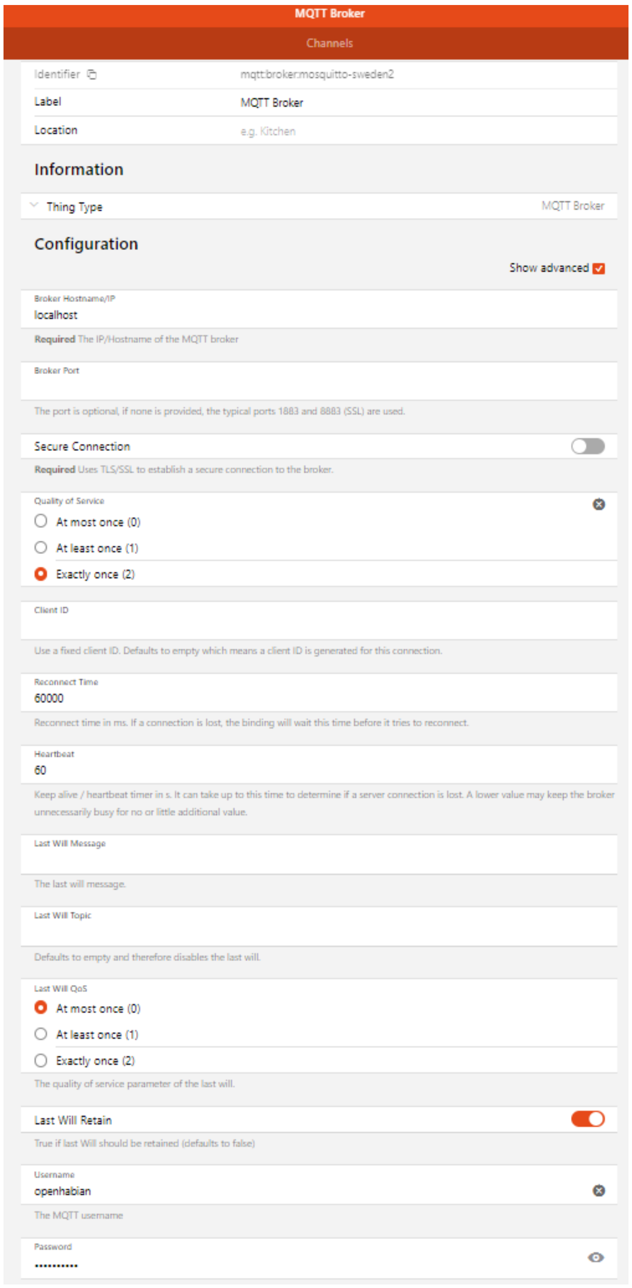
f. Enter:

i. Broker Hostname/IP: **localhost**

ii. Quality of Service: **Exactly Once**

iii. Username: **openhabian**

iv. Password: **mqttpwd22??**

g. 

MQTT Broker

Channels

| | |
|---|---|
| Identifier | mqtt:broker:mosquitto-sweden2 |
| Label | MQTT Broker |
| Location | e.g. Kitchen |

Information

Thing Type — MQTT Broker

Configuration

Show advanced [✓]

Broker Hostname/IP: localhost
Required The IP/Hostname of the MQTT broker

Broker Port:
The port is optional, if none is provided, the typical ports 1883 and 8883 (SSL) are used.

Secure Connection: off
Required Uses TLS/SSL to establish a secure connection to the broker.

Quality of Service: At most once (0) / At least once (1) / [✓ Exactly once (2)]

Client ID:
Use a fixed client ID. Defaults to empty which means a client ID is generated for this connection.

Reconnect Time: 60000
Reconnect time in ms. If a connection is lost, the binding will wait this time before it tries to reconnect.

Heartbeat: 60
Keep alive / heartbeat timer in s. It can take up to this time to determine if a server connection is lost. A lower value may keep the broker unnecessarily busy for no or little additional value.

Last Will Message:
The last will message.

Last Will Topic:
Defaults to empty and therefore disables the last will.

Last Will QoS: [✓ At most once (0)] / At least once (1) / Exactly once (2)
The quality of service parameter of the last will.

Last Will Retain: on
True if last Will should be retained (defaults to false)

Username: openhabian
The MQTT username

Password: ••••••••••

3. Finally configure extended logging for the mosquitto broker. You will need this to see the JSON’s sent from the NSPanel. This is done by creating a configuration file for the Mosquitto broker, steps are:

a. Log on your openhab with putty (or any other ssh client)

b. Run the command: **sudo echo “log_type all” >>/etc/mosquitto/conf.d/local.conf**
c. Run the command: **sudo service mosquitto reload**
d. The mosquitto service now reloads the configuration files and starts extended logging. This really helps in later steps when you need to see what is happening between openhab and NSPanel. Once all configuration is done and everything works, delete the file again and reissue the “reload” command above.

# 4. Flashing Sonoff NSPanel with Tasmota

This step is effectively replacing the stock firmware that came with NSPanel and thus voiding your warranty, so you do this on your own risk.

## Links and references

- Tasmoto windows binary for flashing ESP firmware: Releases · Jason2866/ESP_Flasher · GitHub
- Tasmota firmware for NSPanel: https://github.com/tasmota/install/raw/main/firmware/unofficial/tasmota32-nspanel.bin
- Tasmoto NSPanel Documentation: Sonoff NSPanel Touch Display Switch (E32-MSW-NX) Configuration for Tasmota (blakadder.com)
- Server/location hosting latest nxpanel.tft definition: Index of /nxpanel (proto.systems)
- Location of “nxpanel.be”, the panel definition file adapted for OpenHAB: ns-flash/berry at master · peepshow-21/ns-flash · GitHub

## Preparations

Preparations consist of downloading and installing flashing tools and flash images

### Download Python

Download latest version of Python from here: Download Python | Python.org

- Tick the checkbox for “Add Python to PATH” before install

### Install esptool

The **esptool.py** is a python script that can check if you have connection with the controller in NSPanel through the serial USB adapter. You can also use the script to make a backup of the existing firmware.

To install esptool do the following:

- On your PC, Start a **cmd** window (console window)
- Enter: pip install esptool

Detailed instructions available here: How to Install Esptool on Windows 10 - CyberBlogSpot

### Download Flashing Script (ESP-Flasher)

ESP-Flasher is a flashing tool that writes a flash image to a device using a USB serial adapter.

- Download ESPflasher from here: GitHub - Jason2866/ESP_Flasher: Tasmota Flasher for ESP8266 and ESP32
- The actual binary for windows is called “ESP-Flasher-Windows-x64.exe” and available here: Releases · Jason2866/ESP_Flasher · GitHub

### Downloading new firmware for NSPanel

Firmware from Blackadder for NSPanel (firmware file is called "tasmota32-nspanel.bin")

- Go to this link: https://github.com/blakadder/nspanel
- Download **tasmota32-nspanel.bin** by downloading the entire Code file as zip and then copy this file from the zip into a folder on your PC.

### Ready to flash?

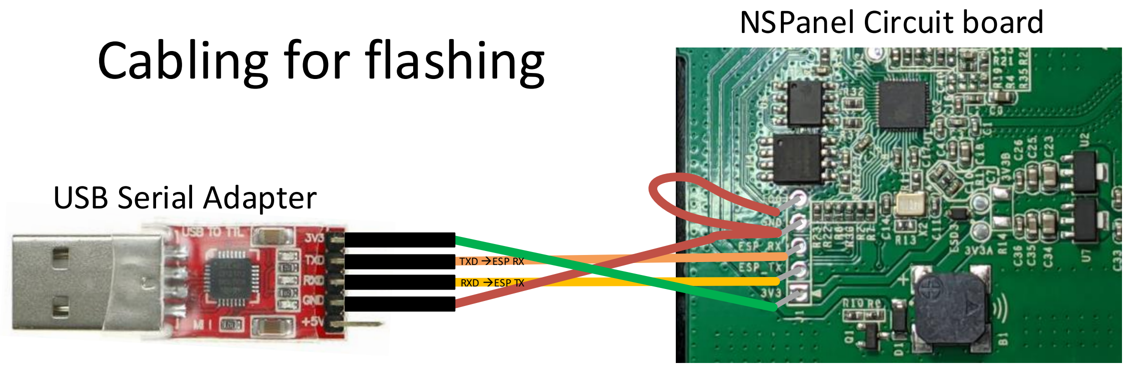
You should now have the following *files* to flash new firmware and do initial Tasmota config:

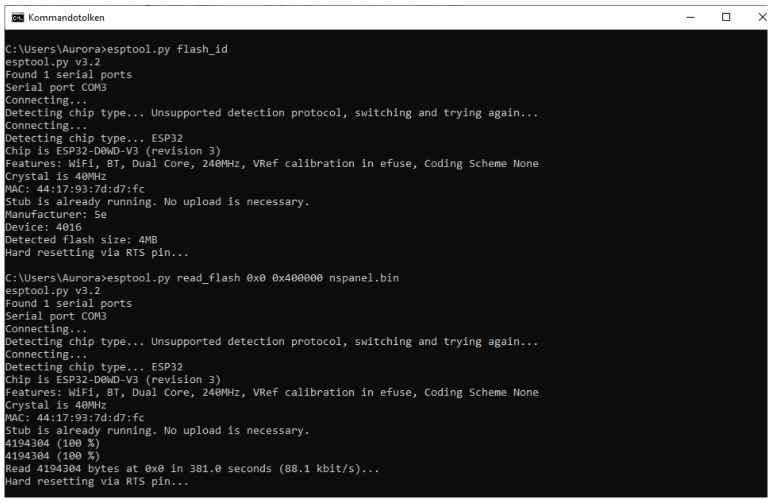
- ESP-Flasher-Windows-x64.exe
- Tasmota32-nspanel.bin

## Flash Sonoff NSPanel firmware

This step describes preparations and flashing of NSPanel firmware to Tasmota.

1. Connect your USB serial adapter to NSPanel (NOTE! **Make sure you to connect 3.3V** and **NOT 5V**. The serial adapter below has two pins, one for 3.3V and one for 5V. Other serial adapters might have a jumper to set 3.3V)



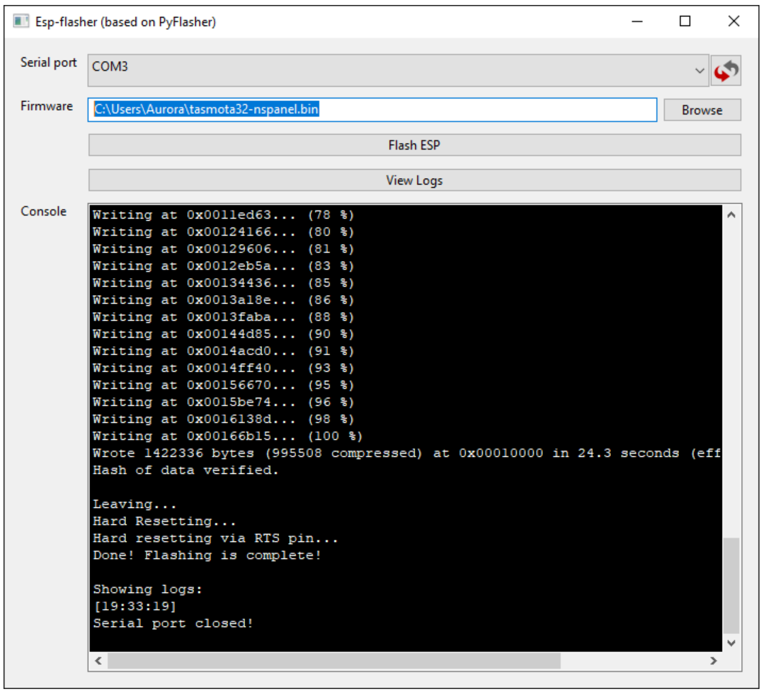
2. On your PC: Open a command window (cmd)
3. Check connection with serial port on chip
   a. Type: **esptool.py flash_id**
   b. You should get a response as shown in the screen shot below.
4. Make a backup of current firmware:
   a. Type: **esptool.py read_flash 0x0 0x400000 nspanel.bin**
5. When done, it looks something like this:
   a. 

```
C:\Users\Aurora>esptool.py flash_id
esptool.py v3.2
Found 1 serial ports
Serial port COM3
Connecting...
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting...
Detecting chip type... ESP32
Chip is ESP32-D0WD-V3 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 44:17:93:7d:d7:fc
Stub is already running. No upload is necessary.
Manufacturer: 5e
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...

C:\Users\Aurora>esptool.py read_flash 0x0 0x400000 nspanel.bin
esptool.py v3.2
Found 1 serial ports
Serial port COM3
Connecting...
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting...
Detecting chip type... ESP32
Chip is ESP32-D0WD-V3 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 44:17:93:7d:d7:fc
Stub is already running. No upload is necessary.
4194304 (100 %)
4194304 (100 %)
Read 4194304 bytes at 0x0 in 381.0 seconds (88.1 kbit/s)...
Hard resetting via RTS pin...
```

6. Flash now firmware with ESP-Flasher
   a. Type: ESP-Flasher-Windows-x64.exe
   b. Select: COM-port in the dropdown (should be only one = USB Serial adapter
   c. Select: Browse
   d. Go to the location of the firmware
   e. Select: the new firmware (tasmota32-nspanel.bin)
   f. Select: Flash ESP

7. When done, it will look something like:



COM3

C:\Users\Aurora\tasmota32-nspanel.bin

```
Writing at 0x0011ed63... (78 %)
Writing at 0x00124166... (80 %)
Writing at 0x00129606... (81 %)
Writing at 0x0012eb5a... (83 %)
Writing at 0x00134436... (85 %)
Writing at 0x0013a18e... (86 %)
Writing at 0x0013faba... (88 %)
Writing at 0x00144d85... (90 %)
Writing at 0x0014acd0... (91 %)
Writing at 0x0014ff40... (93 %)
Writing at 0x00156670... (95 %)
Writing at 0x0015be74... (96 %)
Writing at 0x0016138d... (98 %)
Writing at 0x00166b15... (100 %)
Wrote 1422336 bytes (995508 compressed) at 0x00010000 in 24.3 seconds (eff
Hash of data verified.

Leaving...
Hard Resetting...
Hard resetting via RTS pin...
Done! Flashing is complete!

Showing logs:
[19:33:19]
Serial port closed!
```

One critical thing done 👍, next step is now to connect the NSPanel to your WiFi and do base configuration.