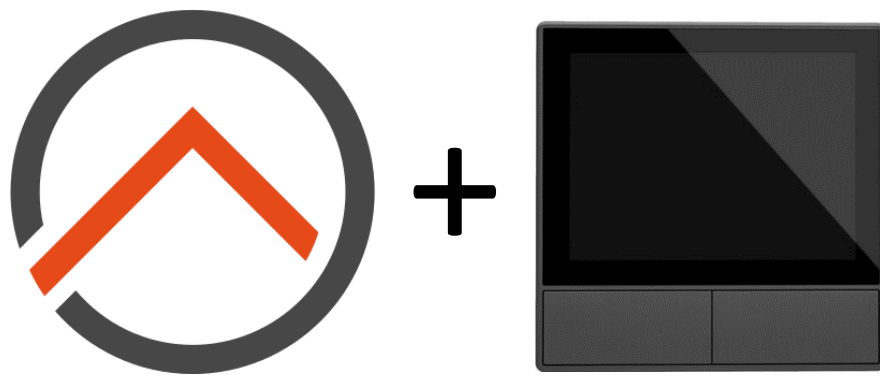


OpenHAB3 & NSPanel



Installation and configuration guide

Alf Pfeiffer, 2022-03-31

Table of content

1. Overview	4
Disclaimer	4
Acknowledgements	4
Hardware and Protocols.....	4
Documentation approach	5
2. Install and configure OpenWeatherMap	6
Links and references.....	6
Installation and configuration	6
3. Install and configure Mosquitto MQTT Broker	8
MQTT overview.....	8
Links and references.....	8
Installation and configuration	8
4. Flashing Sonoff NSPanel with Tasmota.....	11
Links and references.....	11
Preparations	11
Download Python	11
Install esptool	11
Download Flashing Script (ESP-Flasher)	11
Downloading new firmware for NSPanel	11
Ready to flash?	11
Flash Sonoff NSPanel firmware	12
5. Post configuration of Tasmota on NSPanel	14
Post configuration steps after flashing.....	14
6. Base setup of NSPanel-to-OpenHAB communication	16
Links and references.....	16
Preparations	16
Download an OpenHAB adopted “nxpanel.be”	16
Installation and configuration	16
Connecting NSPanel with OpenHAB.....	18
Enable logging!	19
Configure MQTT in NSPanel	19
Configuring MQTT in OpenHAB.....	20
How it works.....	23

7. Configuring the start panel	25
Configuration and installation	25
NXPanel Thing definition	25
NXPanel Item definitions	29
Validate that Items are linked	30
Configure rules that update primary panel	30
Push Weather information to NSPanel	30
Push Temperature(s) to NSPanel	31
8. Custom panel configuration	34
9. Appendix	35
Start Panel on NSPanel	35
Json templates	35
Examples	35
Tasmota Console Commands	35
Mikes Groovy script	35

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 6.

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.

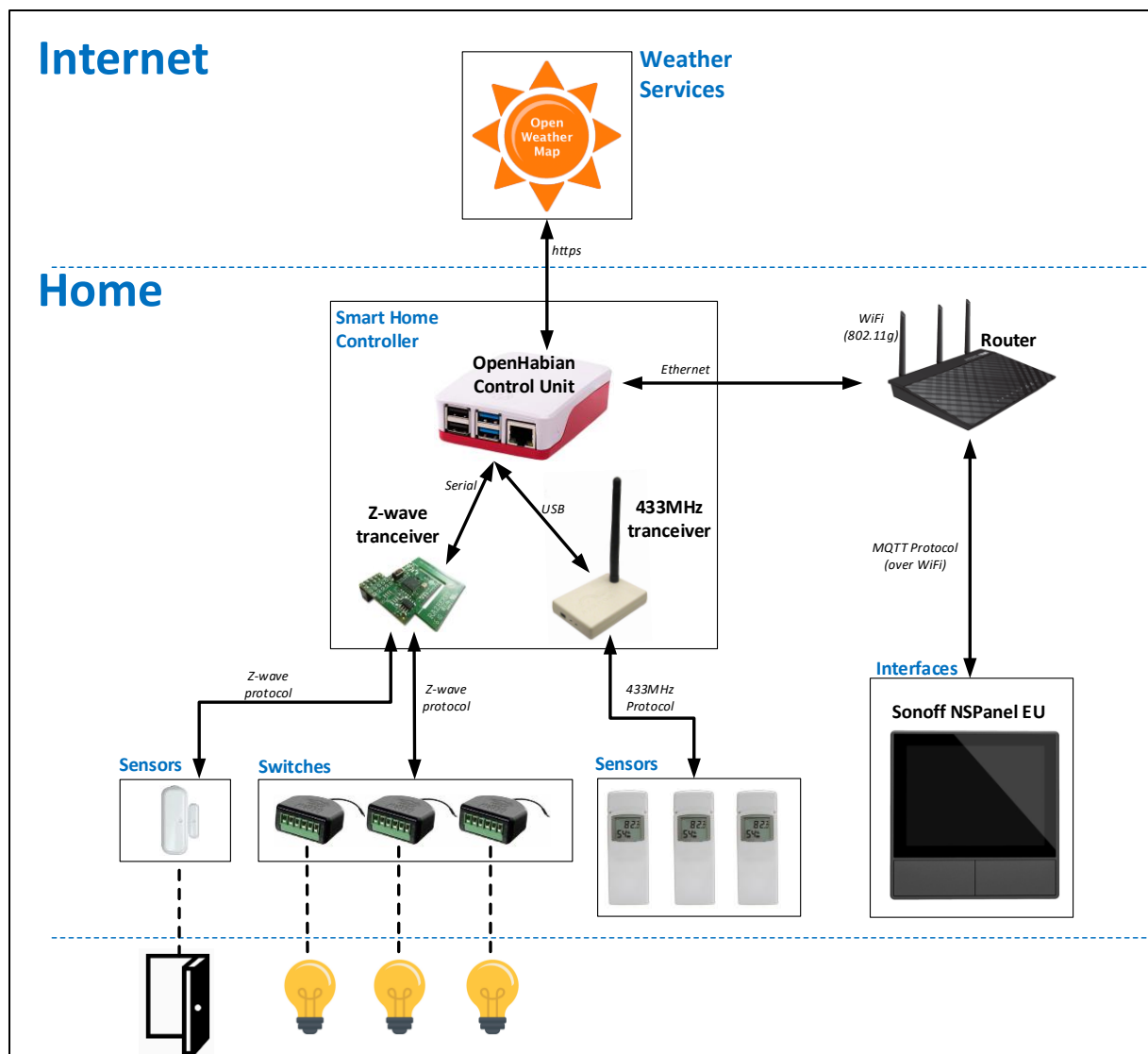
Acknowledgements

- [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)
- Configure the start panel
- 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:0bd084e6fe1ocal

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 [Map](#)

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.

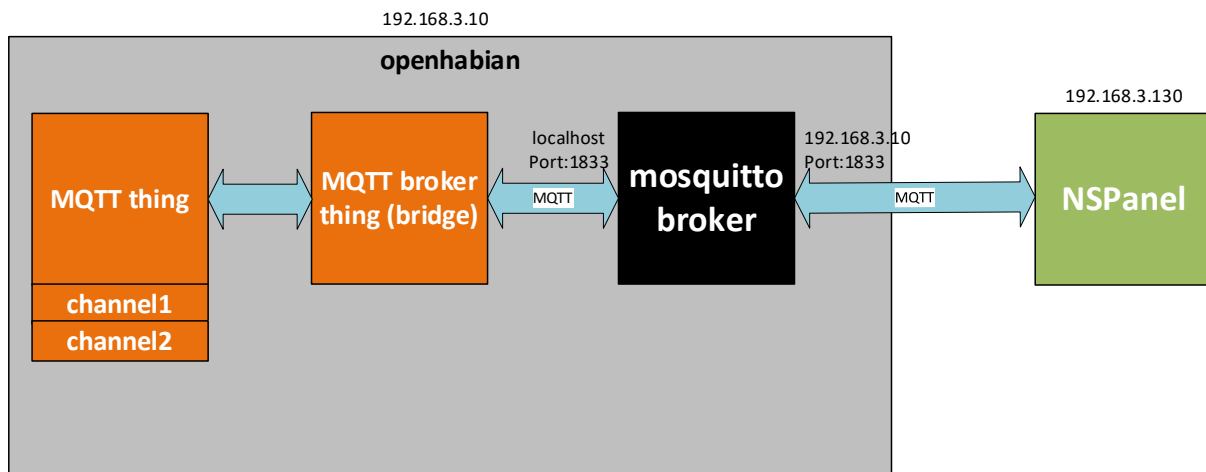
Links and references

- 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 6.

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 **MQTT broker thing (bridge)** which in turn is tied to your actual **NSPanel MQTT thing**. (IP's are of course mine, you will have others..). Once configured, the **MQTT broker thing** and **Mosquitto 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??**

The screenshot shows the MQTT Broker configuration page. The 'Channels' section at the top lists the broker with identifier 'mqtt.broker.mosquitto-sweden2'. The 'Information' section shows the 'Thing Type' as 'MQTT Broker'. The 'Configuration' section is expanded, showing various settings. The 'Broker Hostname/IP' is set to 'localhost'. The 'Broker Port' is empty. The 'Secure Connection' is disabled. The 'Quality of Service' is set to 'Exactly once (2)'. The 'Client ID' is empty. The 'Reconnect Time' is set to '60000'. The 'Heartbeat' is set to '60'. The 'Last Will Message', 'Last Will Topic', and 'Last Will QoS' are all empty. The 'Last Will Retain' is enabled. The 'Username' is set to 'openhabian' and the 'Password' is masked with dots.

- g.
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?

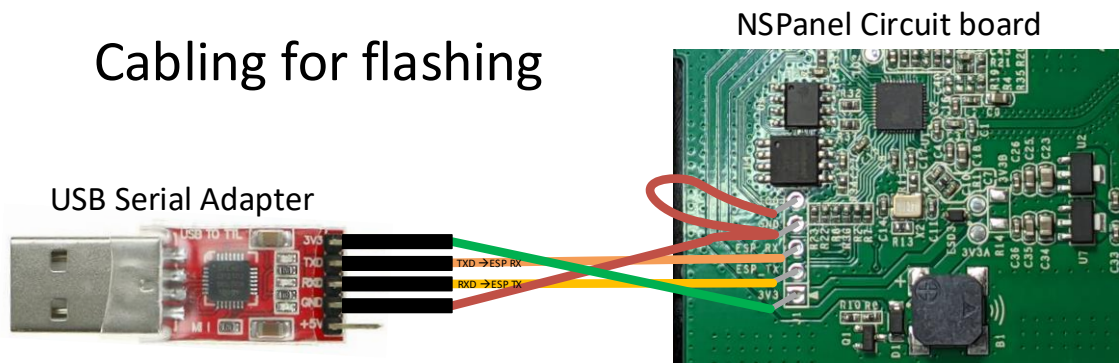
You should now have the following *files* to flash new firmware and do initial Tasmota config:

- 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:

```

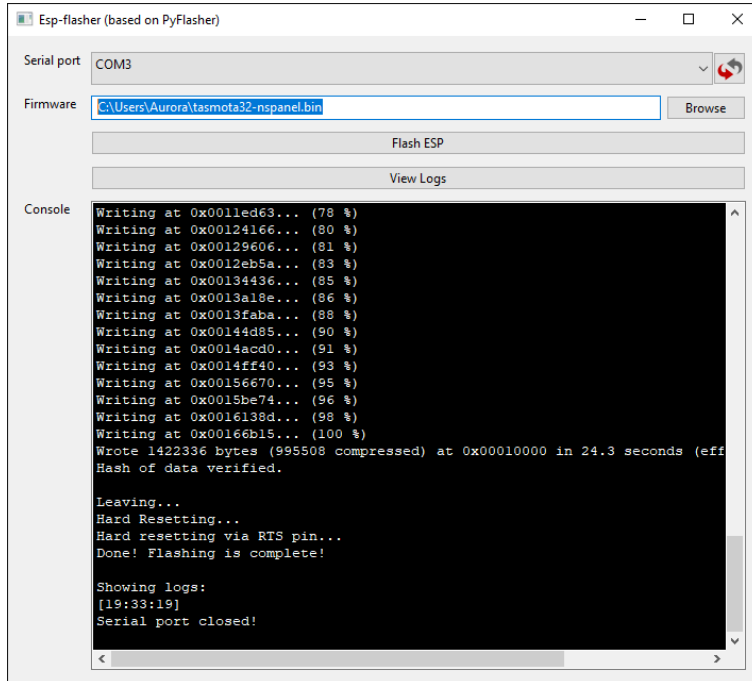
Kommandotolken
C:\Users\Aurora>esptool.py flash_id
esptool.py v2.2
Found 1 serial ports
Serial port COM3
Connecting...
Detecting chip type... Unsupported detection protocol, switching and trying again...
Detecting chip type... ESP32
Chip is ESP32-D0W0-V3 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, 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: 4816
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...
Detecting chip type... ESP32
Chip is ESP32-D0W0-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...

```

- a.
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:



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

5. Post configuration of Tasmota on NSPanel

Post configuration of Tasmota on NSPanel after flashing to make it ready for integration with OpenHAB.

Post configuration steps after flashing.

Steps are:

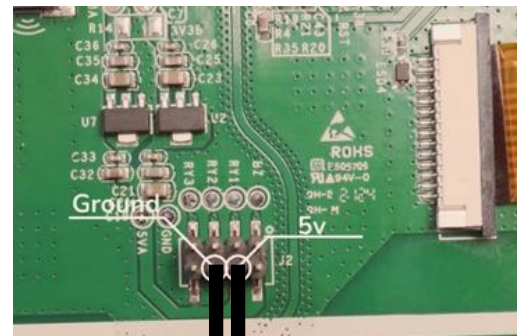
1. Unplug the 3.3V power (disconnect USB from serial adapter)
2. On NSPanel: Plug 5V + GND on two bottom middle pins:
3. On USB Serial Adapter: Plug 5V + GND

Cabling for post configuration

USB Serial Adapter



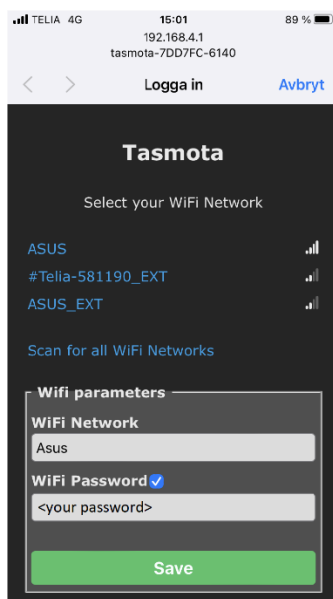
NSPanel Circuit board



4. Power your USB serial adapter by plugging it in on your PC
5. A WiFi hotspot should now appear called e.g., “Tasmota7DD7FC-6140” (or something similar)
6. Connect to the WiFi hotspot (used iPhone for this, didn't detect it on my PC..)



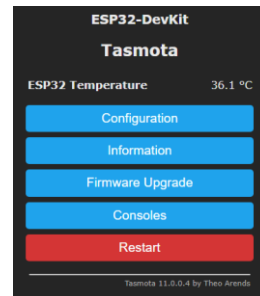
7. Put in WiFi SSID and password for you home WiFi and press “Save”:



and then this is shown:



8. The NSPanel will now connect to your WiFi



9. Browse to the IP that is displayed (192.168.3.121):

10. Do some initial configuration:

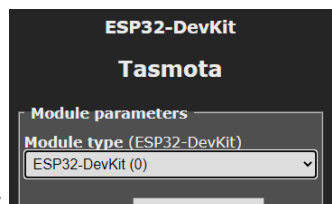
- a. Select: Configuration
- b. Select: Configure Other
- c. Replace Template string with:

```
{ "NAME": "NSPanel1", "GPIO": [0,0,0,0,3872,0,0,0,0,0,32,0,0,0,0,225,0,48,0,224,1,0,0,0,33,0,0,0,0,0,0,0,0,0,0,0,4736,0], "FLAG": 0, "BASE": 1, "CMND": "ADCParam 2,11200,10000,3950 | Sleep 0 | BuzzerPWM 1" }
```

- d. Select: “Save” (Tasmota now reboots)
- e. The screen should now come alive!

2. One final change

- a. Select: Configuration
- b. Select: Configure Module



- c. Select: ESP32-DevKit (0):
- d. Select: “Save” (Tasmota now reboots)

At this stage you now have a running NSPanel that is ready to be integrated to OpenHAB 😊.

If you used the instruction from the Tasmota, this is also the cut-off point where you use Mikes “[nxpanel.be](#)” file instead of installing the “[nspanel.be](#)” file described in the [Tasmota instruction](#).

6. Base setup of NSPanel-to-OpenHAB communication

This final step describes how the panel interface is adopted to work with OpenHAB. This is where the work from Mike comes into play. He has created a new “visual layout” of the panel (screen) which also supports several different panel types. The big advantage is that you with this change will be able to better adapt and extend the NSPanel to your home automation needs. I do not really understand how this actually “works”, just appreciate that it does and fits my purpose.

After the steps in this chapter, you will have:

- A new panel layout installed (Mikes)
- Base communication between NSPanel and OpenHAB setup established
- Customized the primary panel with on your OpenHAB items (temperature and weather)

Links and references

- 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](#)

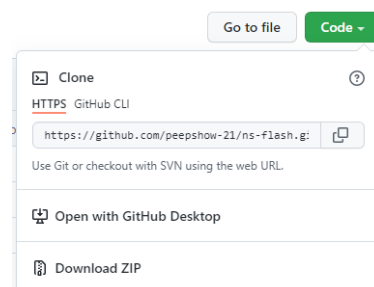
Preparations

Again, some preparations

Download an OpenHAB adopted “nxpanel.be”

Steps are:

- Download nxpanel.be from here: [GitHub - peepshow-21/ns-flash](#)



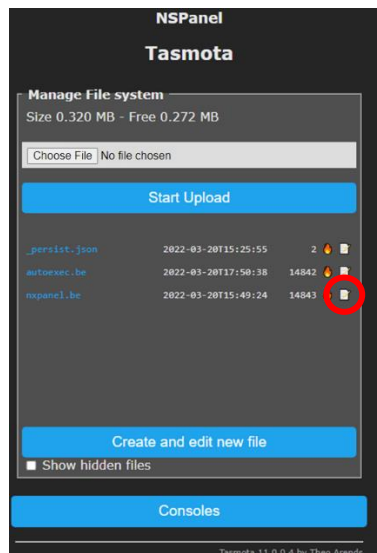
- Select: **Code**
- Select: **Download ZIP**
- You will download a file called “ns-flash-master.zip”
- Extract the file “ns-flash-master.zip\ns-flash-master\berry\nxpanel.be” from this zip and put it in a directory. (there might be other ways to do this, but this is what I did...)

You are now ready to replace the panel definition file.

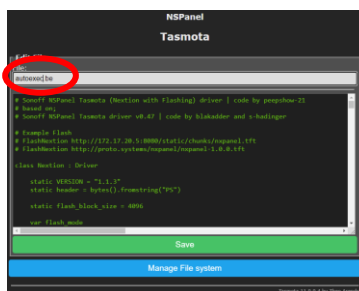
Installation and configuration

Next step is to install the new interface of NSPanel. Instead of using the “nspanel.be” file according to the Tasmota installation instruction, use the “nxpanel.be” file (see "Download an OpenHAB adopted “nxpanel.be”).

1. Browse to the IP-address of your NSPanel
2. The Tasmota web interface is now shown
3. Select: **Consoles**
4. Select: **Manage File System**
5. Select: **Choose File**
6. Browse to where you stored the file and Select: **nxpanel.be**



7. Select: **edit-icon** for nxpanel.be



8. Rename the file to: **autoexec.be**

9. Select: **Save**

10. Select: **Consoles**

11. Select: **Main menu**

12. Select: **Restart**

13. Select: **Consoles**

14. Select: **Console**

15. Type: **InstallNxPanel**

a. NSPanel now starts flashing the “nxpanel-latest.tft” downloaded from this site: [Index of](#)



[/nxpanel \(proto.systems\)](#), screen looks like this:

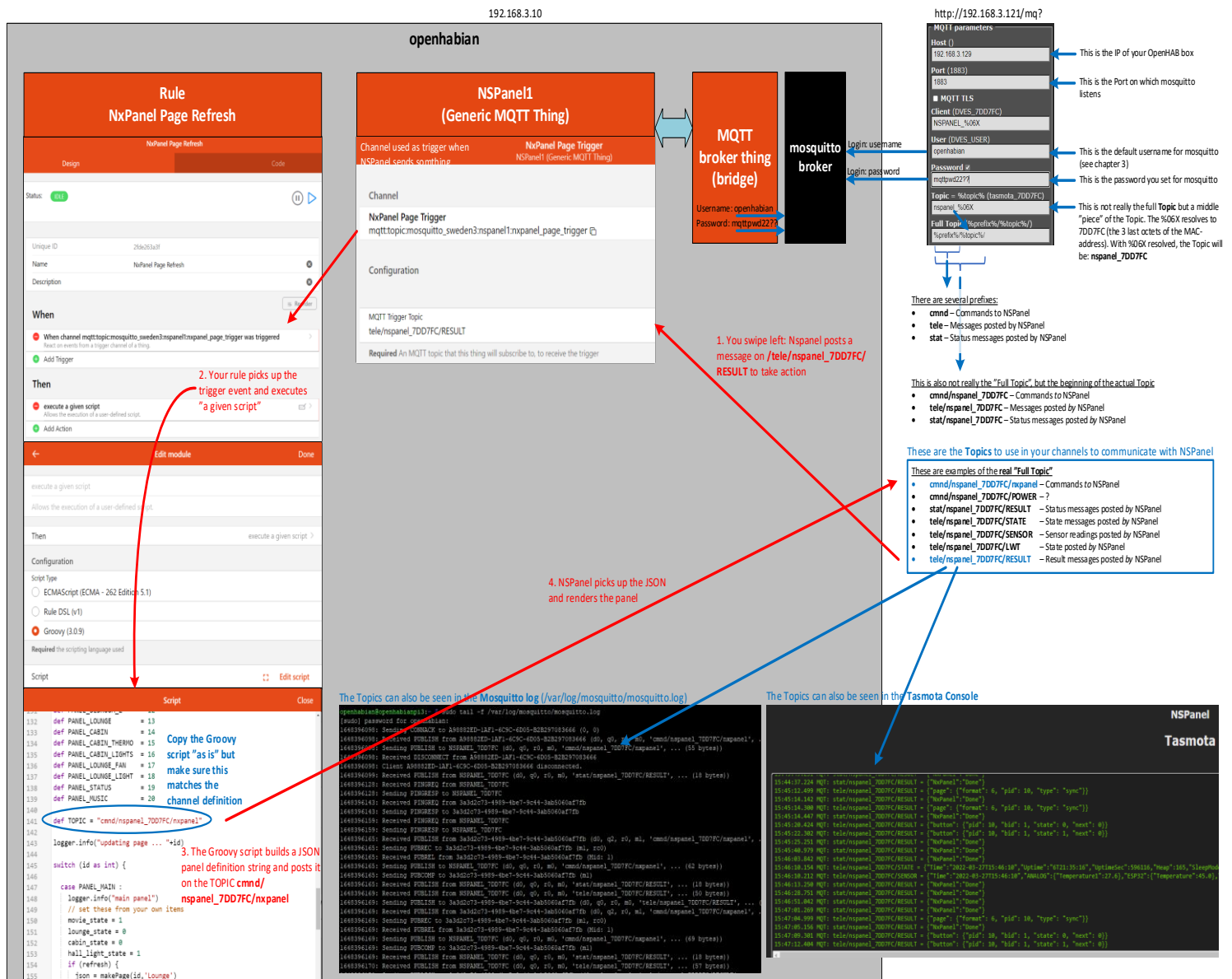


- b. After restart, the panel now looks like this:
- c. **This is a good place to be!** It's now time to connect the NSPanel to OpenHAB.

Connecting NSPanel with OpenHAB

To facilitate the understanding how this is all connected see picture below. Details of how to configure this will follow in the next sections. Legend:

- **Blue:** Configuration stuff
- **Red:** Execution flow



Enable logging!

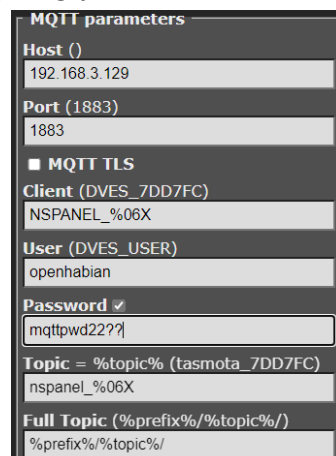
Before you begin configuring the connection, I suggest you prepare logging so you can monitor what's happening. Three logs of interest are:

- The normal **OpenHAB log** (frontail) available here: <http://<your-openhab-IP>:9001>
- The **mosquitto broker log** will full logging enabled (see end of section "Installation and configuration" in Chapter 3). To look at the log:
 - Log on your openhab with putty (or any other ssh client)
 - Run the command: **sudo tail -f /var/log/mosquitto/mosquitto.log**
- The NSPanel Console log available here: <http://<your-NSPanel-IP>/cs?>
 - Select: **Consoles**
 - Select: **Console**
 - Enter command: **weblog 4**
 - This turns on extended logging (to reset to normal, enter command: **weblog 2**)

Configure MQTT in NSPanel

This is where we configure the MQTT settings to start talking to the mosquito broker in OpenHAB.

1. Browse to the IP-address of your NSPanel
2. The Tasmota web interface is now shown
3. Select: **Configuration**
4. Select: **Configure MQTT**
5. Enter Host: **<IP of your OpenHAB>**
6. Enter Client: **NSPANEL_%06X** (don't actually think this is used somewhere)
7. Enter User: **openhbian** (the default user for Mosquitto)
8. Tick the box to the left of Password
9. Enter Password: **mqttpwd22??** (Must match the one you entered when installing Mosquitto)
10. Enter the Topic: **nspanel_%06X** (you can use anything, just make sure this matches everywhere)



The screenshot shows the 'MQTT parameters' configuration form. The fields are as follows:

Field	Value
Host ()	192.168.3.129
Port (1883)	1883
■ MQTT TLS	
Client (DVES_7DD7FC)	NSPANEL_%06X
User (DVES_USER)	openhbian
Password <input checked="" type="checkbox"/>	mqttpwd22??
Topic = %topic% (tasmota_7DD7FC)	nspanel_%06X
Full Topic (%prefix%/%topic%/)	%prefix%/%topic%/

11. When done the screen looks something like this:
12. Select: **Save**
13. After reboot, entries should now start to show in /var/log/mosquitto/mosquitto.log
14. This is good. Your NSPanel has successfully logged into your mosquito broker. But nothing will happen as no one is listening in OpenHAB yet...

Configuring MQTT in OpenHAB

This is where we configure the MQTT settings in OpenHAB to be able to 1. Send commands to NSPanel and 2. To listen what NSPanel is posting to us. We will also create a rule that uses the template Groovy script from Mike just to get us started on getting our custom panels in place to confirm communication back and forth is working.

In short, we will configure:

- A **Generic MQTT thing** representing our NSPanel
- Two **channels** for the above thing, one for receiving messages and one for sending commands to the NSPanel.
- One **rule** that triggers on received messages from NSPanel and sends commands back

Steps are:

- Log on as admin in the OpenHAB web interface.
- Select: **Settings**
- To create the MQTT Thing for NSPanel
 - Select: **Things** and press "+"
 - Select: **MQTT Binding**
 - Select: **Generic MQTT Thing**
 - Enter a Label: **NSPanel1 (Generic MQTT Thing)**
 - Select Bridge: **MQTT Broker**
 - Select: **Save** (top right corner)
- To create a trigger channel for the above thing:
 - On the Things Menu, Select: **NSPanel1 (Generic MQTT Thing)**
 - Select: **Channels** (top middle)
 - Select: **Add Channel**
 - As Channel Identifier; Enter: **nxpanel_page_trigger**
 - As Label; Enter: **NXPanel Page Trigger**
 - Select: **Trigger**
 - Tick: **Show Advanced**
 - As MQTT Command Topic; Enter: **tele/namespace_7DD7FC/RESULT**
 - As QoS; Enter: **Exactly Once**
 - Select: **Done** (top right)
 - After creation, the channel should look something like this:



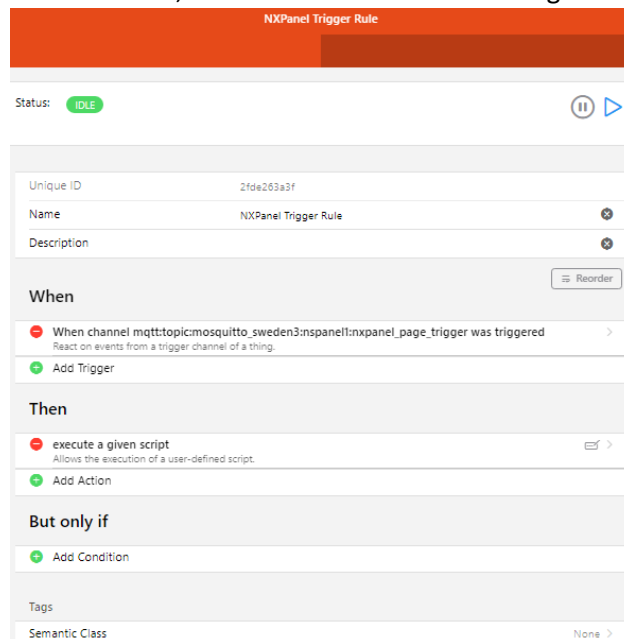
The screenshot displays the configuration page for a channel named 'NXPanel Page Trigger' under the 'NSPanel1 (Generic MQTT Thing)'. The 'Channel' section shows the identifier 'nxpanel_page_trigger' and the label 'NXPanel Page Trigger'. The 'Configuration' section, with 'Show advanced' checked, specifies the 'MQTT Trigger Topic' as 'tele/namespace_7DD7FC/RESULT'. A note indicates that an MQTT topic subscription is required. The 'Transform Values' section is also present, explaining that these parameters allow for altering received values before use in triggers.

- Select: **Save** (top right) to update the NXPanel1 thing with the new channel
- Only the rule left to configure
 - Select: **Settings**
 - Select: **Rules** and press "+"

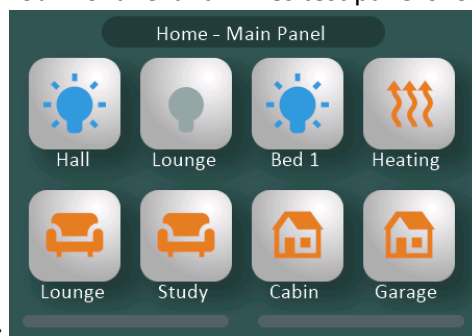
- As Name Enter: **NXPanel Trigger Rule**
- Select: **Add Trigger**
- Select: **Thing Event**
- Select: **NSPanel1 (Generic MQTT Thing)**
- Select: **A trigger channel fired**
- Select: **Done** (top right)
- Select: **Add Action**
- Select: **Run Script**
- Select: **Groovy** (remember to have installed the Groovy Automation)

Cut and Paste Mikes default Grovy script, you can either pick it from section “Mikes Groovy script” in the

- **Appendix** or from the [community post](#)
- **Important #1!** After adding the script code: **Go to line 141** (the one that says `def TOPIC = "cmd/nxpanel/nxpanel"`) and replace the Topic with **"cmd/nspanel_7DD7FC/nxpanel"**. If you don't, the script will post the response on the wrong Topic.
- **Important #2!** After adding the script code: **Go to line 48** (the one that says `def mqtt = actions.get("mqtt", "mqtt:broker:mqtt_broker")`) and replace the last part of the id with the **<name you gave your MQTT brooker>**. (In my example this part is "mosquitte-sweden2", see chapter 3). Will not work without this change.
- Select: **Save (Ctrl-S)** (top right corner).
- After creation, the rule should look something like this:



- **Done!** Swipe left on Your NSPanel and Mikes test panel should now be displayed, the first



panel looks like this:

If this does not work:

1. Check in your logs that the "topics" are all correctly matched in all places (you will most probably have another topic compared with the one I put in as example as this is based on the MAC address on my NSPanel).
2. Check if you get the following message in the OpenHAB (frontail) log when you swipe left: "Demo page rules called". This means that the rule is triggered through the channel **NXPanel Trigger** which in turn means that the read topic is correct. If the demo page is not displayed this means

that the response the script posts does not succeed. Check that the topic in the script matches the one in the channel definition of **NXPanel Command**.

How it works

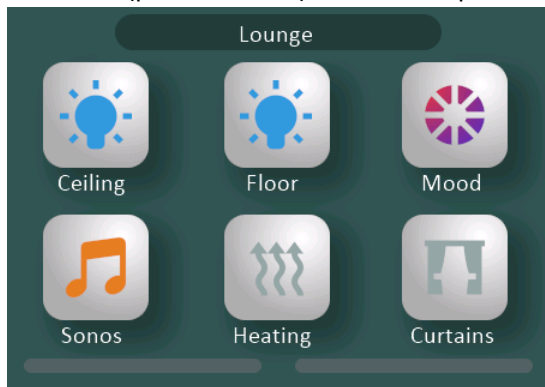
This is about what is sent between OpenHaB and NSPanel and assumes you now have a working connection.

When you swipe left, the NSPanel posts the JSON in blue to OpenHAB.

```
2022-03-27 21:02:48.967 [INFO ] [openhab.event.ChannelTriggeredEvent ] -  
mqtt:topic:mosquitto_sweden3:nspanell:nxpanel_page_trigger triggered {"page": {"format": 6, "pid": 10,  
"type": "sync"}}  
2022-03-27 21:02:48.974 [INFO ] [org.openhab.core.automation.nspanel ] - Demo page rules called  
2022-03-27 21:02:48.978 [INFO ] [org.openhab.core.automation.nspanel ] - updating page ... 10  
2022-03-27 21:02:48.981 [INFO ] [org.openhab.core.automation.nspanel ] - main panel  
2022-03-27 21:02:48.988 [INFO ] [org.openhab.core.automation.nspanel ] - rule done
```

The rule you created is triggered and the action for the rule is to run the Groovy script.

The piece: `"pid": 10` (pid = Panel ID) tells the script to render the panel with ID 10. This panel



looks like this:

The script posts the following JSON in response to NSPanel which renders the panel:

```
{ "refresh": { "pid": 10, "name": "Lounge", "6buttons": [ { "bid": 1, "label": "Movie", "type": 1, "state": 1, "icon": 1 }, { "bid": 2, "label": "Lounge", "type": 1, "state": 0, "icon": 1 }, { "bid": 3, "label": "Hall", "type": 2, "icon": 6 }, { "bid": 4, "label": "Bedroom", "type": 10, "next": 11, "state": 5, "icon": 5 }, { "bid": 5, "label": "Temp", "type": 10, "next": 15, "state": 9, "icon": 9 }, { "bid": 6, "label": "Light", "type": 3, "next": 18, "state": 1, "icon": 2 }, { "bid": 7, "label": "Dimmer", "type": 4, "next": 16, "state": 0, "icon": 3 }, { "bid": 8, "label": "Status", "type": 10, "next": 19, "state": 15, "icon": 16 } ] }
```

Or the same in formatted, a bit more readable form:

```
{  
  "refresh": {  
    "pid": 10,  
    "name": "Lounge",  
    "6buttons": [  
      {  
        "bid": 1,  
        "label": "Movie",  
        "type": 1,  
        "state": 1,  
        "icon": 1  
      },  
      {  
        "bid": 2,  
        "label": "Lounge",  
        "type": 1,  
        "state": 0,  
        "icon": 1  
      }  
    ]  
  }  
}
```

```

    },
    {
      "bid":3,
      "label":"Hall",
      "type":2,
      "icon":6
    },
    {
      "bid":4,
      "label":"Bedroom",
      "type":10,
      "next":11,
      "state":5,
      "icon":5
    },
    {
      "bid":5,
      "label":"Temp",
      "type":10,
      "next":15,
      "state":9,
      "icon":9
    },
    {
      "bid":6,
      "label":"Light",
      "type":3,
      "next":18,
      "state":1,
      "icon":2
    },
    {
      "bid":7,
      "label":"Dimmer",
      "type":4,
      "next":16,
      "state":0,
      "icon":3
    },
    {
      "bid":8,
      "label":"Status",
      "type":10,
      "next":19,
      "state":15,
      "icon":16
    }
  ]
}

```

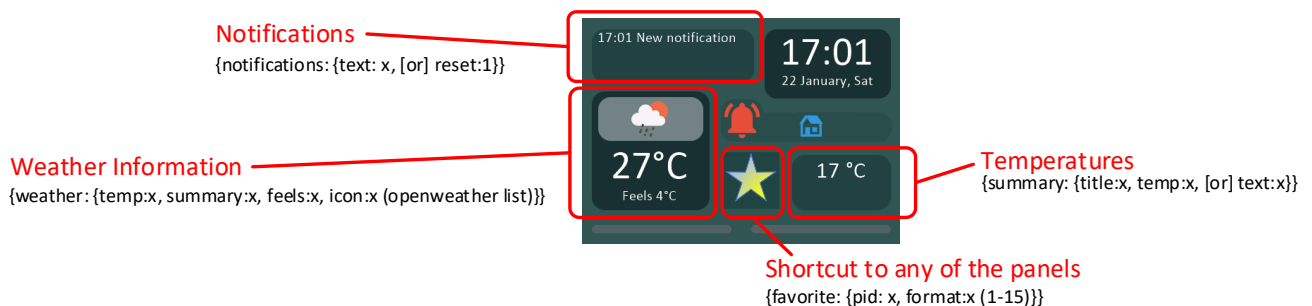

7. Configuring the start panel

The start panel is shown after reboot and is basically only used to display some key information elements, e.g., Weather forecast, temperatures, and notifications. The start panel also has a shortcut to one of the underlying panels.

Configuration and installation

Gotten this far, the first thing you typically will start out with is to update the information on the start panel.

The picture below shows the different areas and the corresponding JSON that updates the information in these areas.



If you want to give it a try, do the following:

- Log on your openhab with putty (or any other ssh client)
- Run the command: **mosquitto_pub -u openhabian -P mqttpwd22?? -t cmnd/nspanel_7DD7FC/nxpanel -m '{"summary": {"title":"Out 32°C", "text":"In 29°C"}}'**

NXPanel Thing definition

You should already have a working NXPanel Thing with one trigger channel. To make the first panel work, we now need to create four more Channels.

- On the Things Menu, Select: **NSPanel1 (Generic MQTT Thing)**
 - Select: **Channels** (top middle)
 - Select: **Add Channel**
 - As Channel Identifier; Enter: **nxpanel_weather_command**
 - As Label; Enter: **NXPanel Weather Command**
 - Select: **Text Value**
 - Tick: **Show Advanced**
 - As MQTT Command Topic; Enter: **cmnd/nspanel_7DD7FC/nxpanel**
 - As QoS; Enter: **Exactly Once**
 - Select: **Done** (top right)

- After creation, the channel should look like this:

NXPanel Weather Command
NSPanel1 (Generic MQTT Thing)

Channel

NXPanel Weather Command
mqtttopicmosquito_sweden3:nspanel1:nxpanel_weather_command

Configuration Show advanced ☒

MQTT State Topic

An MQTT topic that this thing will subscribe to, to receive the state. This can be left empty, the channel will be state-less command-only channel.

MQTT Command Topic
cmdnd/nspanel_7DD7FC/nxpanel

An MQTT topic that this thing will send a command to. If not set, this will be a read-only switch.

QoS

☐ At most once (best effort delivery "fire and forget")

☐ At least once (guaranteed that a message will be delivered at least once)

☒ Exactly once (guarantees that each message is received only once by the counterpart)

MQTT QoS of this channel (0, 1, 2). Default is QoS of the broker connection.

Retained ☐

The value will be published to the command topic as retained message. A retained value stays on the broker and can even be seen by MQTT clients that are subscribing at a later point in time.

Is Command ☐

If the received MQTT value should not only update the state of linked items, but command them, enable this option.

Allowed States

If your MQTT topic is limited to a set of one or more specific commands or specific states, define those states here. Separate multiple states with commas. An example for a light bulb state set: ON,DIMMED,OFF

Transform Values

These configuration parameters allow you to alter a value before it is published to MQTT or before a received value is assigned to an item.

Incoming Value Transformations

Applies transformations to an incoming MQTT topic value. A transformation example for a received JSON would be "JSONPATH:\$device.status.temperature" for a json {device: {status: { temperature: 23.2 }}}. You can chain transformations by separating them with the intersection character ^.

Outgoing Value Transformation

Applies a transformation before publishing a MQTT topic value. Transformations are specialised in extracting a value, but some transformations like the MAP one could be useful.

Outgoing Value Format

%s

- We continue to create channels for **NSPanel1 (Generic MQTT Thing)**:
- Select: **Channels** (top middle)
- Select: **Add Channel**
- As Channel Identifier; Enter: **nspanel_temperature_command**
- As Label; Enter: **NXPanel Temperature Command**
- Select: **Text Value**
- Tick: **Show Advanced**
- As MQTT Command Topic; Enter: **cmdnd/nspanel_7DD7FC/nxpanel**
- As QoS; Enter: **Exactly Once**
- Select: **Done** (top right)
- This channel is basically the same and just used to send information to NSPanel. The reason for having two is that we now can link two different items to each of the channels. The Items will retain the last value sent to NSPanel if this needs to be sent again, e.g., after NSPanel losing power.

- Next is to create a channel for **left** button on NSPanel
 - Select: **Channels** (top middle)
 - Select: **Add Channel**
 - As Channel Identifier; Enter: **nxpanel_switch1**
 - As Label; Enter: **NXPanel Switch1**
 - Select: **On/Off Switch**
 - Tick: **Show Advanced**
 - As MQTT State Topic; Enter: **stat/nspanel_7DD7FC/RESULT**
 - As MQTT Command Topic; Enter: **cmdnd/nspanel_7DD7FC/nxpanel**
 - As QoS; Enter: **Exactly Once**
 - As Custom On/Open Value; Enter: **1**
 - As Custom Off/Closed Value; Enter: **0**
 - As **Incoming Value Transformations**, Enter:
REGEX:(.*POWER1.*)JSONPATH:\$.POWER1
 - As **Outgoing Value Format**, Enter: **{ "switches": { "switch1": %s } }**
 - The channel for Switch1 should now look something like this:

NXPanel Switch1
NSPanel1 (Generic MQTT Thing)

Configuration Show advanced ☒

MQTT State Topic
stat/nspanel_7DD7FC/RESULT

An MQTT topic that this thing will subscribe to, to receive the state. This can be left empty; the channel will be state-less command-only channel.

MQTT Command Topic
cmdnd/nspanel_7DD7FC/nxpanel

An MQTT topic that this thing will send a command to. If not set, this will be a read-only switch.

QoS

- ☐ At most once (best effort delivery "fire and forget")
- ☐ At least once (guaranteed that a message will be delivered at least once)
- ☒ Exactly once (guarantees that each message is received only once by the counterpart)

MQTT QoS of this channel (0, 1, 2). Default is QoS of the broker connection.

Retained ☐

The value will be published to the command topic as retained message. A retained value stays on the broker and can even be seen by MQTT clients that are subscribing at a later point in time.

Is Command ☐

If the received MQTT value should not only update the state of linked items, but command them, enable this option.

Custom On/Open Value
1

A number (like 1, 10) or a string (like "enabled") that is additionally recognised as on/open state. You can use this parameter for a second keyword, next to ON (OPEN) respectively on a Contact.

Custom Off/Closed Value
0

A number (like 0, -10) or a string (like "disabled") that is additionally recognised as off/closed state. You can use this parameter for a second keyword, next to OFF (CLOSED) respectively on a Contact.

Transform Values

These configuration parameters allow you to alter a value before it is published to MQTT or before a received value is assigned to an item.

Incoming Value Transformations
REGEX:(.*POWER1.*)JSONPATH:\$.POWER1

Applies transformations to an incoming MQTT topic value. A transformation example for a received JSON would be "/JSONPATH:\$device.status.temperature" for a json {device: {status: {temperature: 23.2}}}. You can chain transformations by separating them with the intersection character ^.

Outgoing Value Transformation

Applies a transformation before publishing a MQTT topic value. Transformations are specialised in extracting a value, but some transformations like the MAP one could be useful.

Outgoing Value Format
{ "switches": { "switch1": %s } }

Format a value before it is published to the MQTT broker. The default is to just pass the channel/item state. If you want to apply a prefix, say "MYCOLOR:", you would use "MYCOLOR:%s". If you want to adjust the precision of a number to for example 4 digits, you would use "%.4f".

- Select: **Done** (top right)
- Next is to create a channel for **right** button on NSPanel
 - Select: **Channels** (top middle)
 - Select: **Add Channel**

- As Channel Identifier; Enter: **nxpanel_switch2**
- As Label; Enter: **NXPanel Switch2**
- Select: **On/Off Switch**
- Tick: **Show Advanced**
- As MQTT State Topic; Enter: **stat/nspanel_7DD7FC/RESULT**
- As MQTT Command Topic; Enter: **cmdnd/nspanel_7DD7FC/nxpanel**
- As QoS; Enter: **Exactly Once**
- As Custom On/Open Value; Enter: **1**
- As Custom Off/Closed Value; Enter: **0**
- As **Incoming Value Transformations**, Enter:
REGEX:(.*POWER2.*)~JSONPATH:\$.POWER2
- As **Outgoing Value Format**, Enter: **{ "switches": { "switch2": %s } }**
- The channel for Switch2 should now look something like this:

NXPanel Switch2
NXPanel1 (Generic MQTT Thing)

Channel
NXPanel Switch2
mqtttopic:mosquitto_sweden3:nspanel1:nxpanel_switch2

Configuration Show advanced

MQTT State Topic
stat/nspanel_7DD7FC/RESULT
An MQTT topic that this thing will subscribe to, to receive the state. This can be left empty; the channel will be state-less command-only channel.

MQTT Command Topic
cmdnd/nspanel_7DD7FC/nxpanel
An MQTT topic that this thing will send a command to. If not set, this will be a read-only switch.

QoS
☐ At most once (best effort delivery "fire and forget")
☐ At least once (guaranteed that a message will be delivered at least once)
☒ Exactly once (guarantees that each message is received only once by the counterpart)
 MQTT QoS of this channel (0, 1, 2). Default is QoS of the broker connection.

Retained ☐

The value will be published to the command topic as retained message. A retained value stays on the broker and can even be seen by MQTT clients that are subscribing at a later point in time.

Is Command ☐

If the received MQTT value should not only update the state of linked items, but command them, enable this option.

Custom On/Open Value
1
A number (like 1, 10) or a string (like "enabled") that is additionally recognised as on/open state. You can use this parameter for a second keyword, next to ON (ORPN respectively) on a Contact).

Custom Off/Closed Value
0
A number (like 0, -10) or a string (like "disabled") that is additionally recognised as off/closed state. You can use this parameter for a second keyword, next to OFF (CLOSED respectively) on a Contact).

Transform Values
These configuration parameters allow you to alter a value before it is published to MQTT or before a received value is assigned to an item.

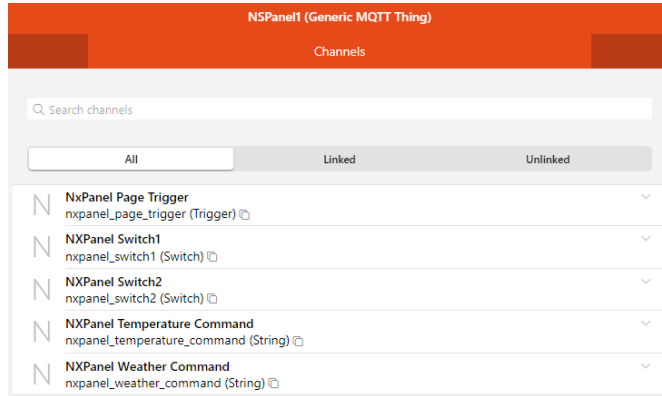
Incoming Value Transformations
REGEX:(.*POWER2.*)~JSONPATH:\$.POWER2
Applies transformations to an incoming MQTT topic value. A transformation example for a received JSON would be "JSONPATH:\$.device.status.temperature" for a json {device: {status: {temperature: 23.2 }}}. You can chain transformations by separating them with the intersection character ~.

Outgoing Value Transformation
Applies a transformation before publishing a MQTT topic value. Transformations are specialised in extracting a value, but some transformations like the MAP one could be useful.

Outgoing Value Format
{ "switches": { "switch2": %s } }
Format a value before it is published to the MQTT broker. The default is to just pass the channel/item state. If you want to apply a prefix, say "MYCOLOR:", you would use "MYCOLOR:%s". If you want to adjust the precision of a number to for example 4 digits, you would use

- Finally, select: **Save** (top right) to update the NXPanel1 thing with the four new channels

- You should now have a NSPanel1 (General MQTT Thing) with channels as this:



NXPanel Item definitions

Next step is to add the Items we want to link to the channels so that we can get automatic updates of weather forecast, temperature(s) and manage the status of the two physical switches on the NSPanel. You will need the following **Items**:

- Current_Outdoor_Temp → From your outdoor thermometer
- Current_Indoor_Temp → From your indoor thermometer
- Forecast_Temp_1day → From OpenWeatherMap
- Forecast_Feels_1day → From OpenWeatherMap
- Forecast_WeatherIcon_1day → From OpenWeatherMap
- nxpanel_weather_command → Holds JSON string generated by “NXPanel Weather Rule”
- nxpanel_temperature_command → Holds JSON string generated by “NXPanel Temperature Rule”

Classic item-file config for the above would look something like this.

Note! I kept the channel definitions for my items so you can match this with the definition of the Channels for the **NSPanel1 (Generic MQTT thing)** in the previous section – Your channels will be different.

```
// NXPanel - Command Items (to send JSON commands to NXPanel)
String nxpanel_weather_command {channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_weather_command"}
String nxpanel_temperature_command {channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_temperature_command"}

// Your thermometer Items
Number:Temperature Current_Outdoor_Temp {channel="<your channel to Outdoor thermometer>"}
Number:Temperature Current_Indoor_Temp {channel="<your channel to Outdoor thermometer>"}

// Weather Forecast Items from OpenWeatherMap
Number:Temperature Forecast_Temp_1day {channel="openweathermap:onecall:0bd084e6fe:local:forecastTomorrow#day-temperature"}
Number:Temperature Forecast_Feels_1day {channel="openweathermap:onecall:0bd084e6fe:local:forecastTomorrow#apparent-day"}
String Forecast_WeatherIcon_1day {channel="openweathermap:onecall:0bd084e6fe:local:forecastTomorrow#icon-id"}

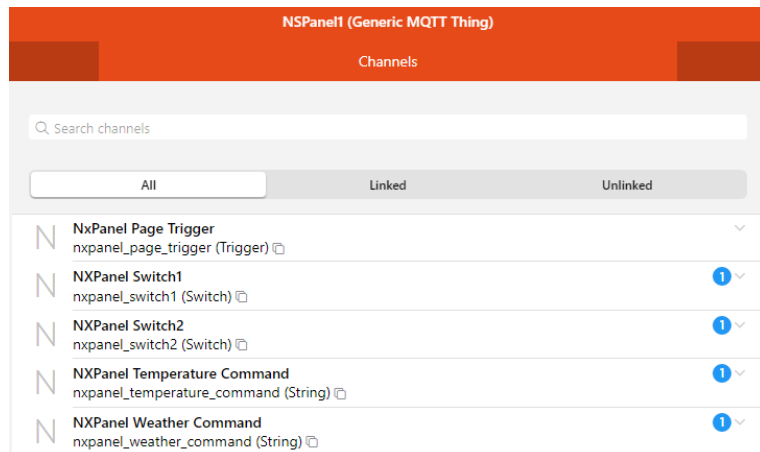
// NXPanel - Buttons
```

```
Switch nxpanel_sw1 "NXPanel - Switch {channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_switch1"}
Switch nxpanel_sw2 "NXPanel - Switch {channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_switch2"}
```

Note! Make sure the channels match the channels in your setup

Validate that Items are linked

After creating these items, make sure they are correctly linked to your channels. The NXPanel1 thing should now look like this – Note the blue dots representing the linked items.



Configure rules that update primary panel

Final thing to configure to get the first page & buttons working are the rules that trigger the updates if temperature or weather forecast changes.

Push Weather information to NSPanel

- First rule is to push the weather information whenever this gets updates from OpenWeatherMap. Steps are:
 - Select: **Settings**
 - Select: **Rules** and press "+"
 - As Name Enter: **NXPanel Weather Rule**
- Three triggers to be added, first is:
 - Select: **Add Trigger**
 - Select: **Item Event**
 - Select: **Forecast_WeatherIcon_1day** (tick "show non-semantic" if you do not use semantic models)
 - Tick: **changed**
 - Select: **Done** (top right)
- Second trigger is:
 - Select: **Add Trigger**
 - Select: **Item Event**
 - Select: **Forecast_Feels_1day**
 - Tick: **changed**
 - Select: **Done** (top right)
- Third trigger is:
 - Select: **Add Trigger**
 - Select: **Item Event**
 - Select: **Forecast_Temp_1day**
 - Tick: **changed**

- Select: **Done** (top right)
- Finally, add what happens if any of the above triggers fire:
 - Select: **Add Action**
 - Select: **Run Script**
 - Select: **Groovy**
 - Select: **Edit**
 - Cut and Paste this:

```
import org.slf4j.LoggerFactory
def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

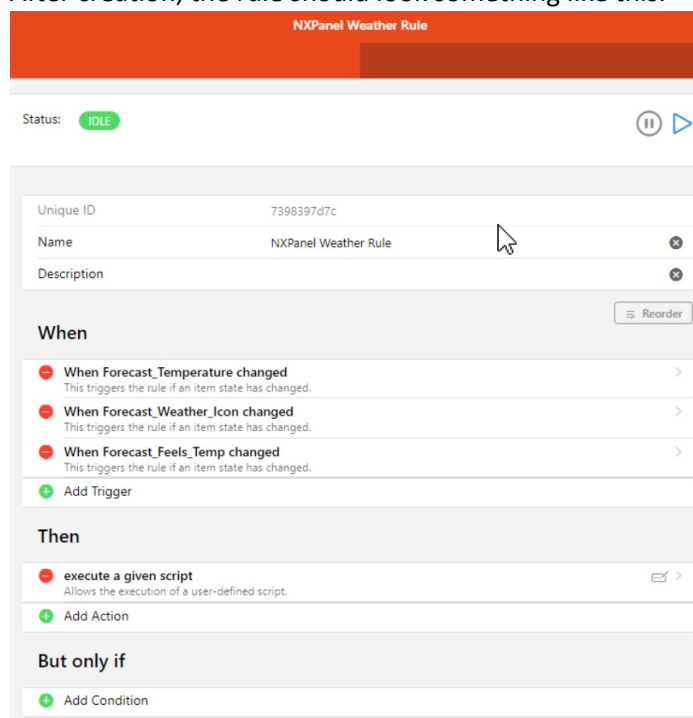
def weather = ir.getItem("Forecast_WeatherIcon_1day").state.toString()
def forecastTemp = ir.getItem("Forecast_Temperature_1day ").state.intValue()
def forecastFeelsTemp = ir.getItem("Forecast_Feels_1day ").state.intValue()

def json = String.format(
    "{ \"weather\": { \"temp\": %d, \"icon\": \"%s\", \"feels\": %d } }",
    forecastTemp, weather, forecastFeelsTemp)

events.sendCommand("nxpanel_weather_command", json)

logger.info("nxpanel_weather_command: "+json)
```

- Select: **Save (Ctrl-S)** (top right corner).
- After creation, the rule should look something like this:



Push Temperature(s) to NSPanel

- Second rule is to push the temperature updates. You can actually just send one temperature value, check the JSON string syntax in section “Start Panel on NSPanel” in the Appendix), but I liked to have both in- and outdoor temps on the panel.
 - Select: **Settings**
 - Select: **Rules** and press “+”
 - As Name Enter: **NXPanel Temperature Rule**
- Two triggers to be added for this, first is:

- Select: **Add Trigger**
- Select: **Item Event**
- Select: **Current_Outdoor_Temp** (tick “show non-semantic” if you do not use semantic models)
- Tick: **changed**
- Select: **Done** (top right)
- Second trigger is:
 - Select: **Add Trigger**
 - Select: **Item Event**
 - Select: **Current_Outdoor_Temp**
 - Tick: **changed**
 - Select: **Done** (top right)
- Now add what happens if any of the above triggers fire:
 - Select: **Add Action**
 - Select: **Run Script**
 - Select: **Groovy**
 - Select: **Edit**
 - Cut and Paste this:

```
import org.slf4j.LoggerFactory
def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

def Current_Outdoor_Temp = ir.getItem("Current_Outdoor_Temp").state.intValue()
def Current_Indoor_Temp = ir.getItem("Current_Indoor_Temp").state.intValue()

def json = String.format(
    "{ \"summary\": { \"title\": \"Ute %d°C\", \"text\": \"Inne %d°C\" } }",
    Current_Outdoor_Temp, Current_Indoor_Temp)

events.sendCommand("nxpanel_temperature_command", json)

logger.info("nxpanel_temperature_command: "+json)
```

- Select: **Save (Ctrl-S)** (top right corner).
- After creation, the rule should look something like this:

The screenshot shows the configuration of a rule named "NXPanel Temperature Rule". At the top, the status is "IDLE". Below this, a table lists the rule's details: Unique ID (d91452c50f), Name (NXPanel Temperature Rule), and Description. The rule is configured with two triggers under the "When" section: "When Current_Outdoor_Temp changed" and "When Current_Indoor_Temp changed". Under the "Then" section, there is one action: "execute a given script". The "But only if" section is currently empty.

At this stage you have your first panel working!

8. Custom panel configuration

To be continued – Haven't figured this out myself yet, so you need to be patient.

9. Appendix

Start Panel on NSPanel

Json templates

- {start: {pid: x, format:x (1-15)}}
- {favorite: {pid: x, format:x (1-15)}}
- {dim: {low:n, normal:n}}
- {notifications: {text: x, [or] reset:1}}
- {weather: {temp:x, summary:x, feels:x, icon:x (openweather list)}}
- {clock: {date:x, hour:x, min:x, month:x, weekday:x}} // called by tasmotoa
- {switches: {switch1:x, switch2: x}} // called by tasmotoa
- {summary: {title:x, temp:x, [or] text:x}}
- {warnings: [{id:x (1-4) , type:x (0-7) , state:x (1-3)},...]}

Examples

Command in openhabian:

Set brightness levels

- mosquitto_pub -u openhabian -P mqttPWD22?? -t cmd/nspanel_7DD7FC/nxpanel -m '{"dim":{"low":10, "normal":80}}'

Set temperatures:

- mosquitto_pub -u openhabian -P mqttPWD22?? -t cmd/nspanel_7DD7FC/nxpanel -m '{"summary":{"title":"Our 32°C", "text":"In 25°C"}}'

Tasmota Console Commands

UK Timezones

TimeZone 99

TimeDST 0,0,3,1,1,60

TimeSTD 0,0,10,1,2,0

CET Timezone

TimeZone 99

TimeDST 0,0,3,1,1,60

TimeSTD 0,0,10,1,2,0

weblog 2 - Normal logging

weblog 4 - Extended logging

restart 1 - Restarts the panel

Mikes Groovy script

```
import org.slf4j.LoggerFactory
```

```

def PAGE_HOME = 1
def PAGE_2_BUTTON = 2
def PAGE_3_BUTTON = 3
def PAGE_4_BUTTON = 4
def PAGE_6_BUTTON = 5
def PAGE_8_BUTTON = 6
def PAGE_DIMMER = 7
def PAGE_DIMMER_COLOR = 8
def PAGE_THERMOSTAT = 9
def PAGE_ALERT_1 = 10
def PAGE_ALERT_2 = 11
def PAGE_ALARM = 12
def PAGE_MEDIA = 13
def PAGE_PLAYLIST = 14
def PAGE_STATUS = 15

def BUTTON_UNUSED = 0
def BUTTON_TOGGLE = 1
def BUTTON_PUSH = 2
def BUTTON_DIMMER = 3
def BUTTON_DIMMER_COLOR = 4
def BUTTON_PAGE = 10

def ICON_BLANK = 0
def ICON_BULB = 1
def ICON_DIMMER = 2
def ICON_DIMMER_COLOR = 3
def ICON_VACUUM = 4
def ICON_BED = 5
def ICON_HOUSE = 6
def ICON_SOFA = 7
def ICON_BELL = 8
def ICON_HEAT = 9
def ICON_CURTAINS = 10
def ICON_MUSIC = 11
def ICON_BINARY = 12
def ICON_FAN = 13
def ICON_SWITCH = 14
def ICON_TALK = 15
def ICON_INFO = 16

def NONE = 0

def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

def mqtt = actions.get("mqtt", "mqtt:broker:mqtt_broker")

def str = event.getEvent()

logger.info("Demo page rules called")

if (str.indexOf('"page":') != 0) {
    return
}

/*
 * Utility functions - start
 */

def makeButton(bid, label, type, icon=null, state=null, next=null) {
    var str = "<<((bid==1)?'':', '")
    str<< '{"bid":'<<bid<<', "label":'<<label<<', "type":'<<type
    if (next!=null) {
        str<<', "next":'<<next
    }
    if (state!=null) {
        str<<', "state":'<<state
    }
    if (icon!=null) {
        str<<', "icon":'<<icon
    }
}

```

```

    str<<'}'
    return str
}

def makePage(pid,name) {
    var str = new StringBuilder('{"refresh":')
    str<<'{"pid":'<<pid<<',"name":"'<<name<<',"
    return str
}

def makeEmptySync(pid) {
    var str = new StringBuilder('{"sync":')
    str<<'{"pid":'<<pid<<'}}'
    return str
}

def makeEmptyRefresh(pid) {
    var str = new StringBuilder('{"refresh":')
    str<<'{"pid":'<<pid<<'}}'
    return str
}

def makeSyncButtonStart(pid,bid,state) {
    var str = new StringBuilder('{"sync":')
    str<<'{"pid":'<<pid
    str<<',"buttons":[{"bid":'<<bid<<',"state":'<<state<<'}'
    return str
}

def addSyncButton(bid,state) {
    var str = ',{"bid":'<<bid<<',"state":'<<state<<'}'
    return str
}

/*
 * Utility functions - end
 */

/*
 * Get data from the page message
 * (would be good to use JsonSluper here but currently can't access)
 */

var i = str.indexOf("\"pid\"")
var i2 = str.indexOf(", ",i+7)
var id = str.substring(i+7,i2)
i = str.indexOf("\"format\"")
i2 = str.indexOf(", ",i+10)
var format = str.substring(i+10,i2)

// check if a full refresh or just a status update
var refresh = str.indexOf("refresh")>0

var json

def PANEL_MAIN          = 10
def PANEL_BEDROOM_1     = 11
def PANEL_BEDROOM_2     = 12
def PANEL_LOUNGE        = 13
def PANEL_CABIN         = 14
def PANEL_CABIN_THERMO  = 15
def PANEL_CABIN_LIGHTS  = 16
def PANEL_LOUNGE_FAN    = 17
def PANEL_LOUNGE_LIGHT  = 18
def PANEL_STATUS        = 19
def PANEL_MUSIC         = 20

def TOPIC = "cmd/nspanel/nxpanel"

logger.info("updating page ... "+id)

```

```

switch (id as int) {

case PANEL_MAIN :
    logger.info("main panel")
    // set these from your own items
    movie_state = 1
    lounge_state = 0
    cabin_state = 0
    hall_light_state = 1
    if (refresh) {
        json = makePage(id, 'Lounge')
        json<<format<<'buttons:['
        json<<makeButton(1, "Movie", BUTTON_TOGGLE, ICON_BULB, movie_state)
        json<<makeButton(2, "Lounge", BUTTON_TOGGLE, ICON_BULB, lounge_state)
        json<<makeButton(3, "Hall", BUTTON_PUSH, ICON_HOUSE)
        json<<makeButton(4, "Bedroom", BUTTON_PAGE, ICON_BED, PAGE_6_BUTTON, PANEL_BEDROOM_1)
        json<<makeButton(5, "Temp", BUTTON_PAGE, ICON_HEAT, PAGE_THERMOSTAT, PANEL_CABIN_THERMO)

json<<makeButton(6, "Light", BUTTON_DIMMER, ICON_DIMMER, hall_light_state, PANEL_LOUNGE_LIGHT)

json<<makeButton(7, "Dimmer", BUTTON_DIMMER_COLOR, ICON_DIMMER_COLOR, cabin_state, PANEL_CABIN_LIGHTS)

        json<<makeButton(8, "Status", BUTTON_PAGE, ICON_INFO, PAGE_STATUS, PANEL_STATUS)
        json<<"]]}"
    } else {
        json = makeSyncButtonStart(id, 1, movie_state)
        json<<addSyncButton(2, lounge_state)
        json<<"]]}"
    }
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_BEDROOM_1 :
    // set these from your own items
    fan_state = 1
    if (refresh) {
        json = makePage(id, 'Bedroom 1')
        json<<format<<'buttons:['
        json<<makeButton(1, "A", BUTTON_PUSH, ICON_HOUSE)
        json<<makeButton(2, "Fan", BUTTON_DIMMER, ICON_FAN, fan_state, PANEL_LOUNGE_FAN)
        json<<makeButton(3, "C", BUTTON_PUSH, ICON_SOFA)
        json<<makeButton(4, "Music", BUTTON_PAGE, ICON_MUSIC, PAGE_MEDIA, PANEL_MUSIC)
        json<<makeButton(5, "D", BUTTON_PUSH, ICON_TALK)
        json<<makeButton(6, "Alarm", BUTTON_PAGE, ICON_BELL, PAGE_ALARM, NONE)
        json<<"]]}"
    } else {
        json = makeEmptySync(id)
    }
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_BEDROOM_2 :
    json = makeEmptySync(id)
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_LOUNGE :
    json = makeEmptySync(id)
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_CABIN :
    json = makePage(id, 'Cabin')
    json<<"]]}"
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_CABIN_THERMO :
    // set these from your own items
    var heater = 1
    var auto = 0
    var temp = 15
    var set = 21
    json = makePage(id, 'Cabin')
    json<<format<<'{"therm":{"
    json<<"set":'<<set<<',"temp":'<<temp<<',"heat":'<<heater<<',"state":'<<auto<<'"}'

```

```

        json<<"}}"
        mqtt.publishMQTT(TOPIC, json.toString())
        break
    case PANEL_CABIN_LIGHTS :
        json = makePage(id, 'Cabin Lights')
        json<<"power":'<<ON<<', "hsbcolor":'<<"10,100,50"'
        json<<"}}"
        mqtt.publishMQTT(TOPIC, json.toString())
        break
    case PANEL_LOUNGE_FAN :
        // set these from your own items
        fan_state = ON
        fan_setting = 3
        json = makePage(id, 'Lounge Fan')

json<<"power":'<<fan_state<<', "min":'<<1<<', "max":'<<4<<', "icon":'<<ICON_FAN<<', "dimmer":'<<fan_setting
        json<<"}}"
        mqtt.publishMQTT(TOPIC, json.toString())
        break
    case PANEL_LOUNGE_LIGHT :
        json = makePage(id, 'Lounge Light')
        json<<"power":'<<ON<<', "dimmer":'<<30
        json<<"}}"
        mqtt.publishMQTT(TOPIC, json.toString())
        break
    case PANEL_STATUS :
        json = makePage(id, 'System Status')
        json<<"status":['
        json<<{"id":'<<1<<', "text":'<<"Gate":'<<', "value":'<<"Open"'<<', "color":'<<2<<}'
        json<<','
        json<<{"id":'<<2<<', "text":'<<"Window":'<<', "value":'<<', "Shut"'<<', "color":'<<3<<}'
        json<<','
        json<<{"id":'<<5<<', "text":'<<"Room Temp":'<<', "value":'<<', "20°C"'<<}'
        json<<']}]'
        mqtt.publishMQTT(TOPIC, json.toString())
        break
    case PANEL_MUSIC :
        json = makePage(id, 'Sonos Player')
        // set these from your own items
        json<<"artist":'<<"New Order"'<<', "album":'<<"Movement"'<<', "track":'<<"Power
Play"'<<', "volume":'<<70
        json<<"}}"
        mqtt.publishMQTT(TOPIC, json.toString())
        break
    default :
        logger.info("unknown page!")
        break
}

logger.info("rule done")

```