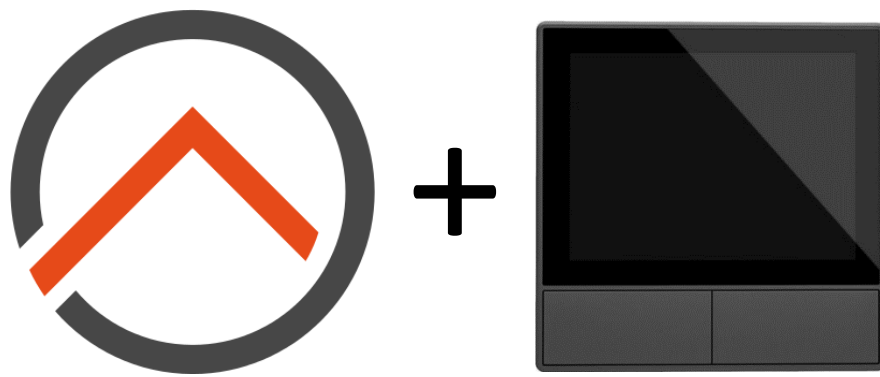


OpenHAB3 & NSPanel



Installation and configuration guide

Alf Pfeiffer, 2022-05-21

Table of content

1. Overview	5
Disclaimer	5
Acknowledgements	5
Hardware and Protocols.....	5
Documentation approach	6
2. Install and configure OpenWeatherMap	7
Links and references.....	7
Installation and configuration	7
3. Install and configure Mosquitto MQTT Broker	9
MQTT overview.....	9
Links and references.....	9
Installation and configuration	9
4. Flashing Sonoff NSPanel with Tasmota.....	12
Links and references.....	12
Preparations	12
Download Python	12
Install esptool	12
Download Flashing Script (ESP-Flasher)	12
Downloading new firmware for NSPanel	12
Ready to flash?	12
Flash Sonoff NSPanel firmware	13
5. Post configuration of Tasmota on NSPanel	15
Post configuration steps after flashing.....	15
6. Base setup of NSPanel-to-OpenHAB communication	17
Links and references.....	17
Preparations	17
Download an OpenHAB adopted “nxpanel.be”	17
Installation and configuration	17
Connecting NSPanel with OpenHAB.....	19
Enable logging!	20
Configure MQTT in NSPanel	20
Configuring MQTT in OpenHAB.....	21
How it works.....	23

7. Configuring the start panel	26
Configuration and installation	26
NXPanel Thing definition	26
NSPanel Buttons – If you don't use the relays	30
NXPanel Item definitions	30
Validate that Items are linked	31
Configure rules that update primary panel	31
Push Weather information to NSPanel	31
Push Temperature(s) to NSPanel	33
8. Custom panel configuration	35
Button Types	35
Warning Types	36
Icons	37
Panel Types	39
Panel Design – Work order	39
Panel Design – Example	39
Adopted Groovy Script	44
Button press responses	50
Create the channels for the buttons	51
Many channels listening on same topic	53
Link your items to the channels	53
Update your Groovy Script	53
Expected output	54
Dimmer press responses	55
Create a channel for the dimmer button	55
Create two dimmer items	56
Link your items to the channels	56
Create NXPanel Trigger Rule Dimmer	56
Groovy Script for Dimmer Panels	57
NXPanel Init	59
NXPanel Relays 1 and 2	62
Problem 1	62
Problem 2	62
Solution	63
9. Appendix	65
Examples	65

Tasmota Console Commands	65
Mikes Groovy script (original)	65

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.

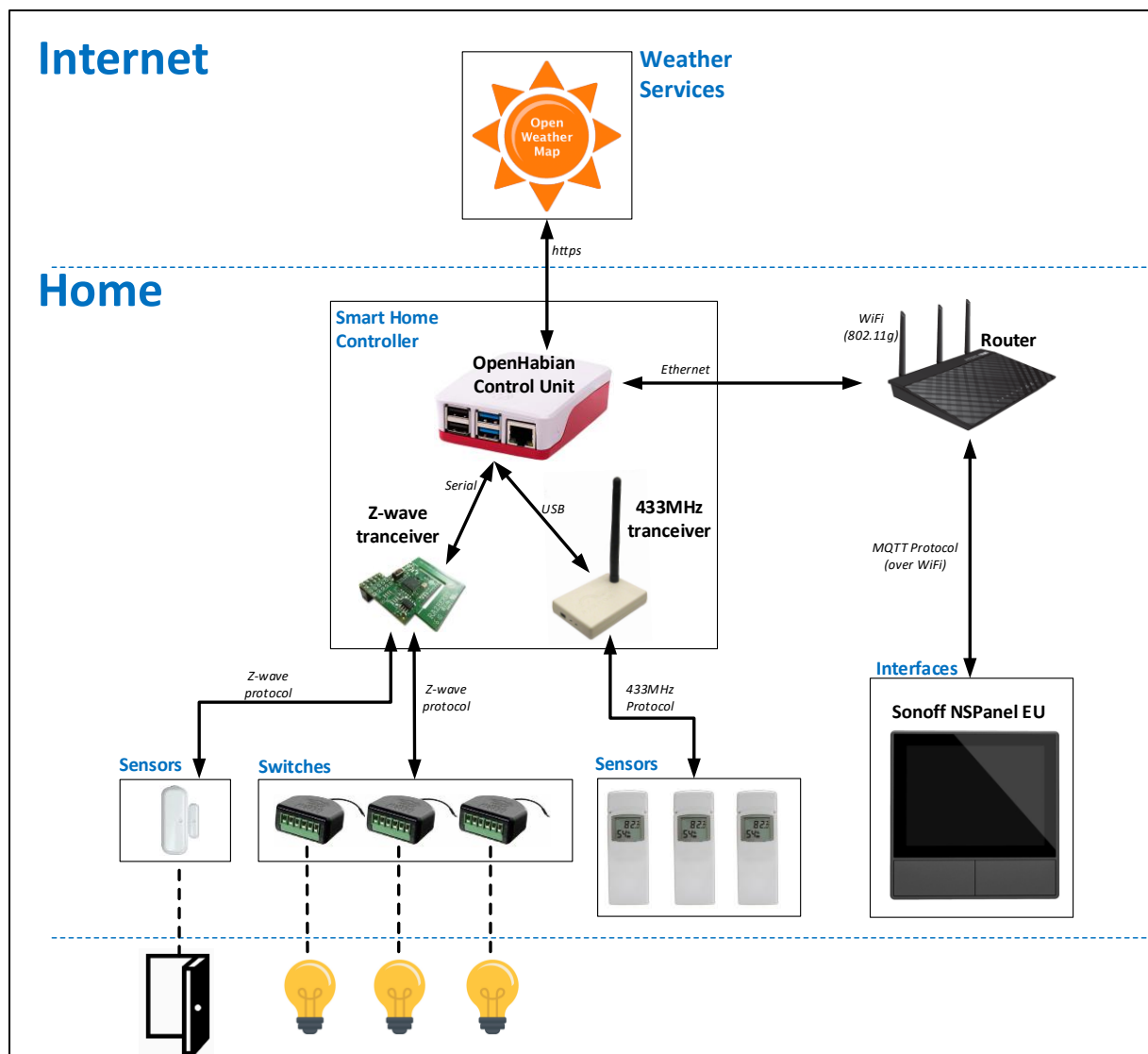
Acknowledgements

- [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 NXPanel 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...).

The screenshot shows a web interface for configuring a weather channel. At the top, there's a red header with the text "Local Weather and Forecast (One Call API)" and a "Channels" button. Below the header, the status is "ONLINE" with a green indicator. The main configuration area is divided into sections: "Identifier" (openweathermap:onecall:0bd084e6fe:local), "Label" (Local Weather and Forecast (One Call API)), "Location" (e.g. Kitchen), "Parent Bridge" (OpenWeatherMap Account), "Information" (Thing Type: One Call API Weather and Forecast), and "Configuration". The configuration section includes fields for "Location of Weather" (13.191482110834826,-59.63790893554688), "Number of Days" (2), "Number of Hours" (12), "Number of Minutes" (0), and "Number of Alerts" (0). Each field has a description and a "Map" button next to the location field.

Field	Value
Identifier	openweathermap:onecall:0bd084e6fe:local
Label	Local Weather and Forecast (One Call API)
Location	e.g. Kitchen
Parent Bridge	OpenWeatherMap Account
Thing Type	One Call API Weather and Forecast
Location of Weather	13.191482110834826,-59.63790893554688
Number of Days	2
Number of Hours	12
Number of Minutes	0
Number of Alerts	0

- 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.

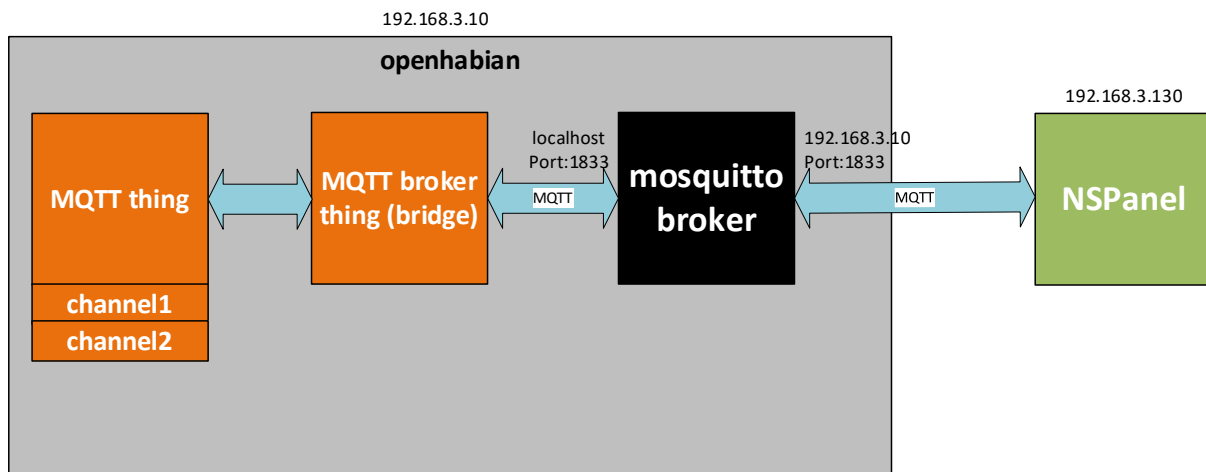
Links and references

- 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 NXPanel is described in chapter 6.

The picture below shows a generic MQTT setup for OpenHAB. The NXPanel 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: 'Broker Hostname/IP' is 'localhost', 'Broker Port' is empty, 'Secure Connection' is disabled, 'Quality of Service' is set to 'Exactly once (2)', 'Client ID' is empty, 'Reconnect Time' is '60000', 'Heartbeat' is '60', 'Last Will Message' is empty, 'Last Will Topic' is empty, 'Last Will QoS' is set to 'At most once (0)', 'Last Will Retain' is checked, 'Username' is 'openhabian', and '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 NXPanel. 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 NXPanel. 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

- Tasmota 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>
- Tasmota 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?

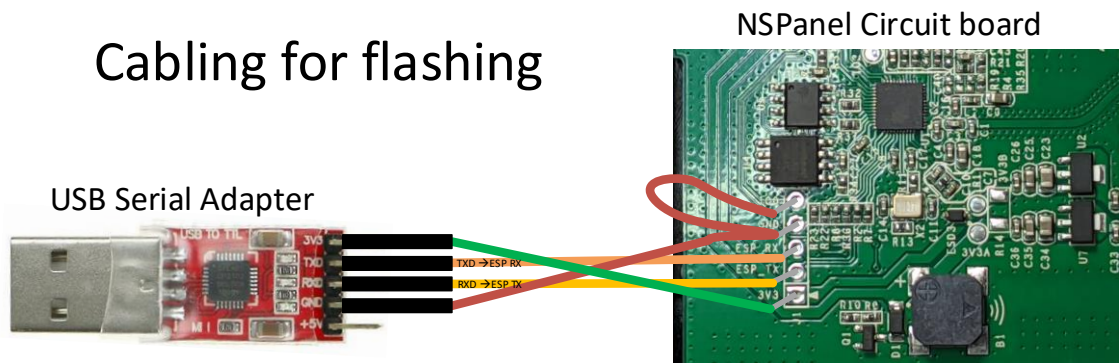
You should now have the following *files* to flash new firmware and do initial Tasmota config:

- 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:

```

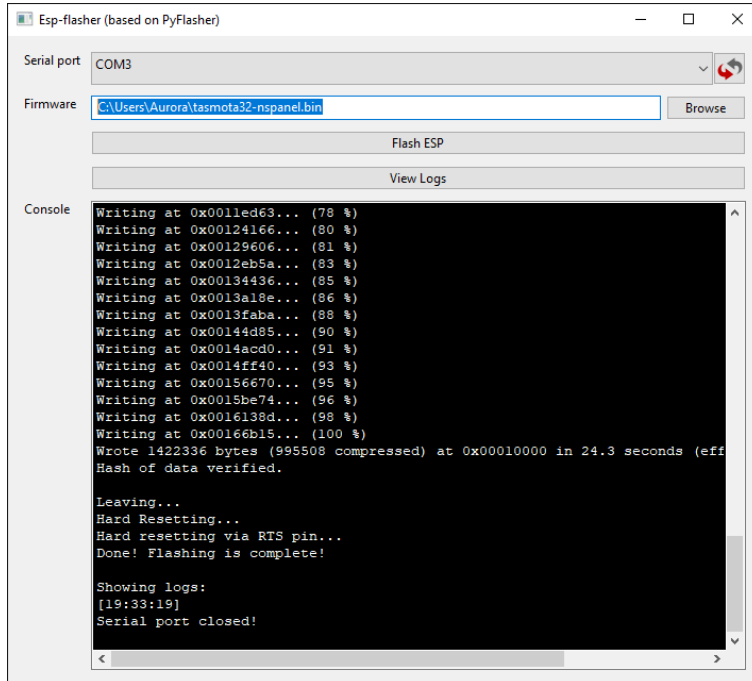
Kommandotolken
C:\Users\Aurora>esptool.py flash_id
esptool.py v2.2
Found 1 serial ports
Serial port COM3
Connecting...
Detecting chip type... Unsupported detection protocol, switching and trying again...
Detecting chip type... ESP32
Chip is ESP32-D0W0-V3 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz,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

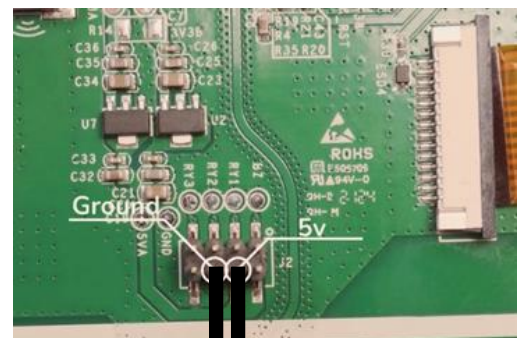
1. Unplug the 3.3V power (disconnect USB from serial adapter)
2. On NSPanel: Plug 5V + GND on two bottom middle pins:
3. On USB Serial Adapter: Plug 5V + GND

Cabling for post configuration

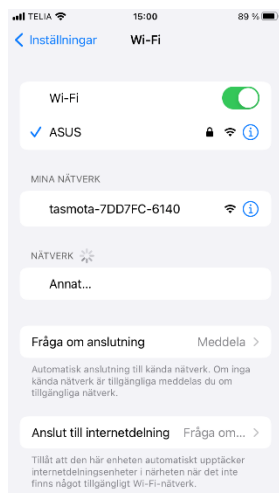
USB Serial Adapter



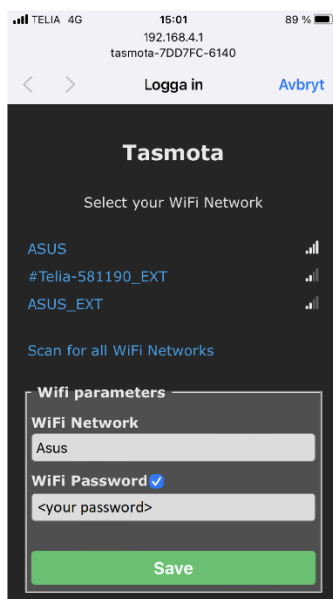
NSPanel Circuit board



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..)

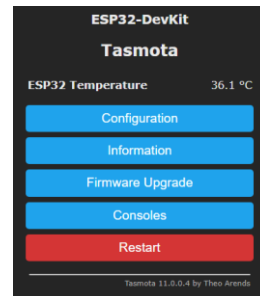


7. Put in WiFi SSID and password for you home WiFi and press “Save”:



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:

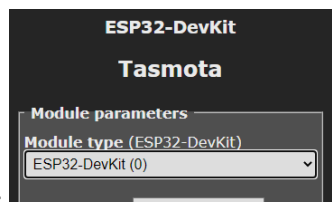
- a. Select: Configuration
- b. Select: Configure Other
- c. Replace Template string with:

```
{ "NAME": "NSPanel1", "GPIO": [0,0,0,0,3872,0,0,0,0,0,32,0,0,0,0,225,0,48,0,224,1,0,0,0,33,0,0,0,0,0,0,0,0,0,0,0,0,0,4736,0], "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](#)

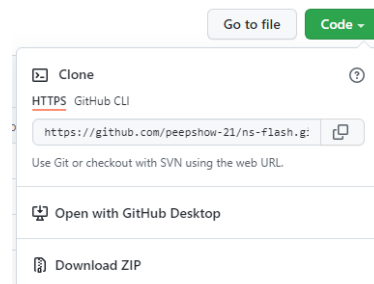
Preparations

Again, some preparations

Download an OpenHAB adopted “nxpanel.be”

Steps are:

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



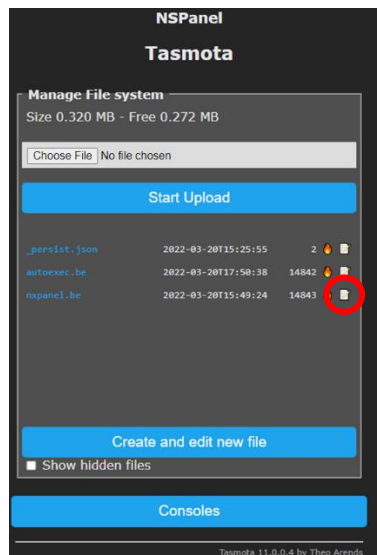
- Select: **Code**
- Select: **Download ZIP**
- You will download a file called “ns-flash-master.zip”
- Extract the file “ns-flash-master.zip\ns-flash-master\berry\nxpanel.be” from this zip and put it in a directory. (there might be other ways to do this, but this is what I did...)

You are now ready to replace the panel definition file.

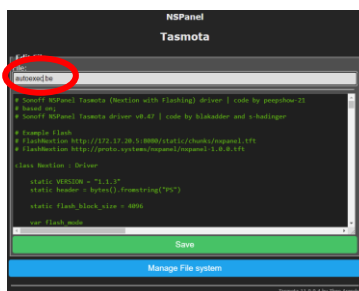
Installation and configuration

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:

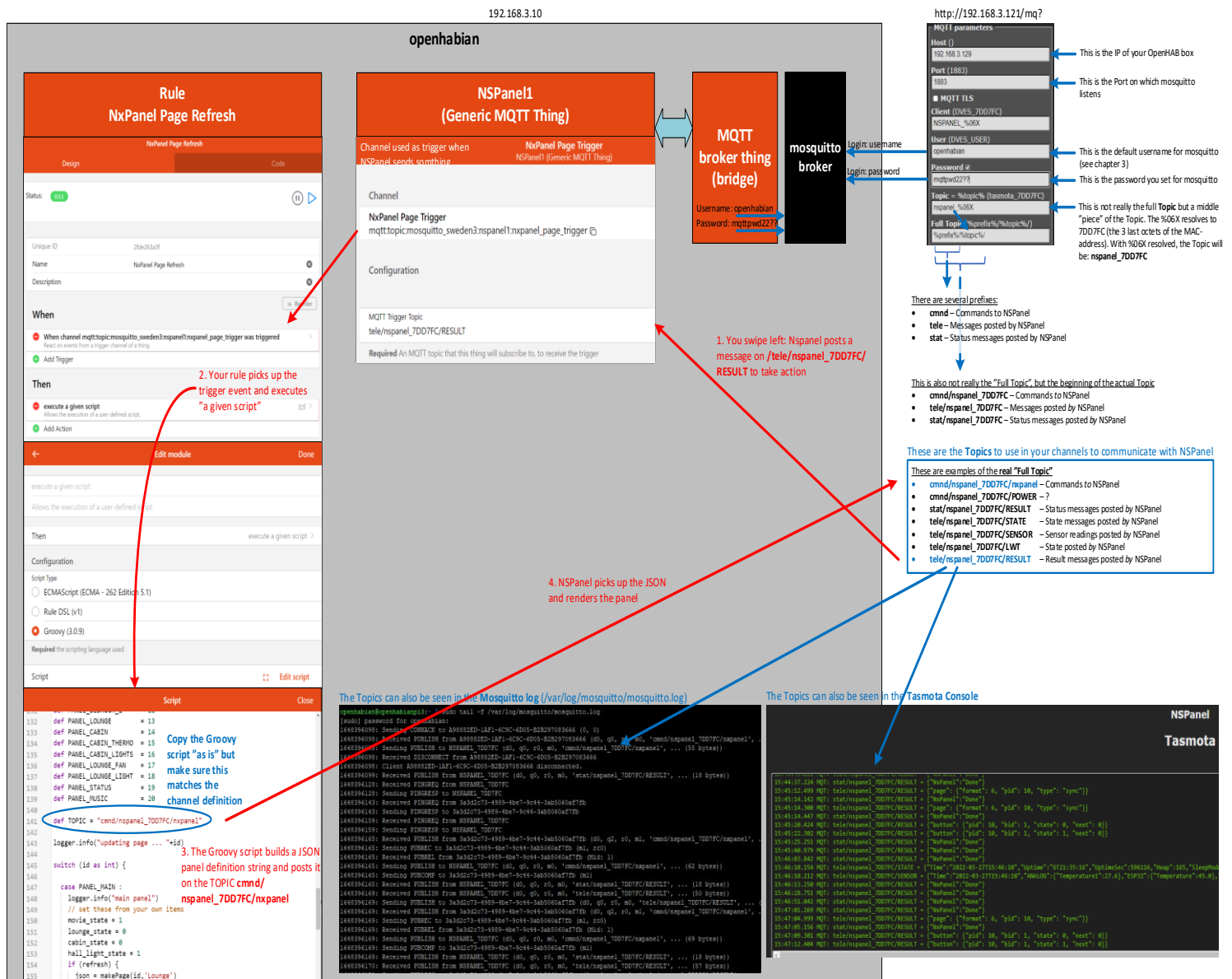


- b. After restart, the panel now looks like this:
- c. **This is a good place to be!** It's now time to connect the NSPanel to OpenHAB.

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!

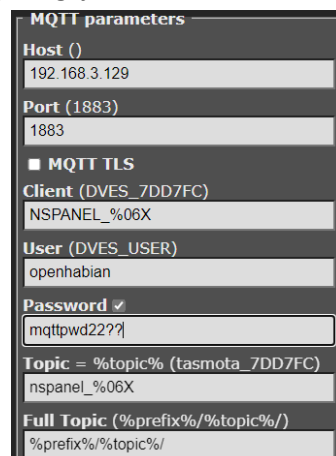
Before you begin configuring the connection, I suggest you prepare logging so you can monitor what's happening. Three logs of interest are:

- 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: **openhabian** (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:

- Host ()**: 192.168.3.129
- Port (1883)**: 1883
- MQTT TLS**: (checkbox is checked)
- Client (DVES_7DD7FC)**: NSPANEL_%06X
- User (DVES_USER)**: openhabian
- Password**: mqttpwd22?? (with a checkmark icon)
- 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:

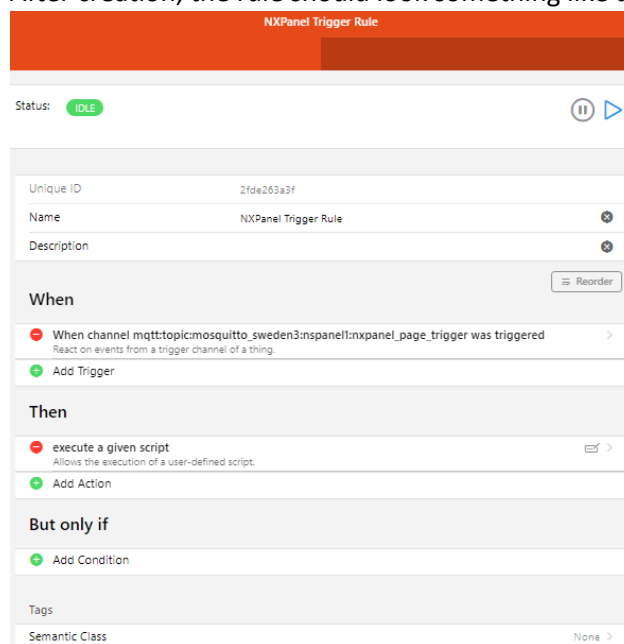
- Log on as admin in the OpenHAB web interface.
- Select: **Settings**
- To create the MQTT Thing for NSPanel
 - Select: **Things** and press "+"
 - Select: **MQTT Binding**
 - Select: **Generic MQTT Thing**
 - Enter a Label: **NSPanel1 (Generic MQTT Thing)**
 - Select Bridge: **MQTT Broker**
 - Select: **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:



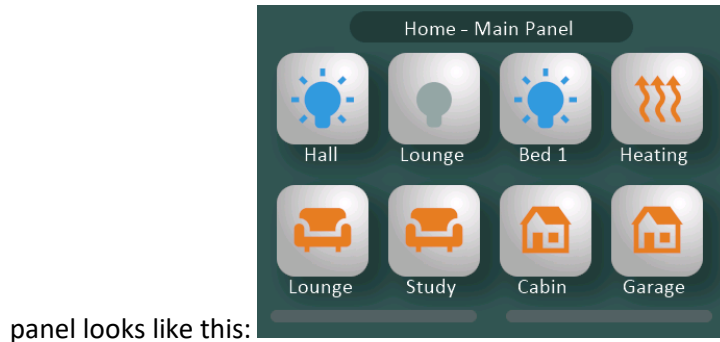
The screenshot displays the configuration page for a channel named 'NXPanel Page Trigger' under the 'NSPanel1 (Generic MQTT Thing)'. The 'Channel' section shows the identifier 'nxpanel_page_trigger' and the label 'NXPanel Page Trigger'. The 'Configuration' section, with 'Show advanced' checked, specifies the 'MQTT Trigger Topic' as 'tele/namespace_7DD7FC/RESULT'. A note indicates that the channel requires an MQTT topic subscription. The 'Transform Values' section is currently empty.

- 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 channel: **nxpanel_page_trigger**
- 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** at the end of this document 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 `. (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



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

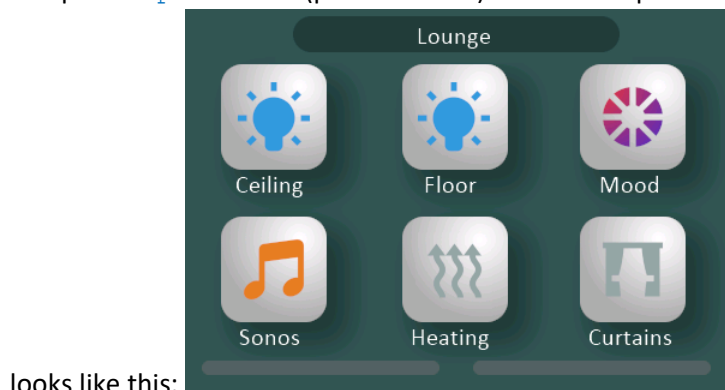
This is about what is sent between OpenHaB and NSPanel and assumes you now have a working connection.

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



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,
        "state": 5,
        "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,
        "state": 5,
        "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
      }
    ]
  }
}
```



```
        "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.

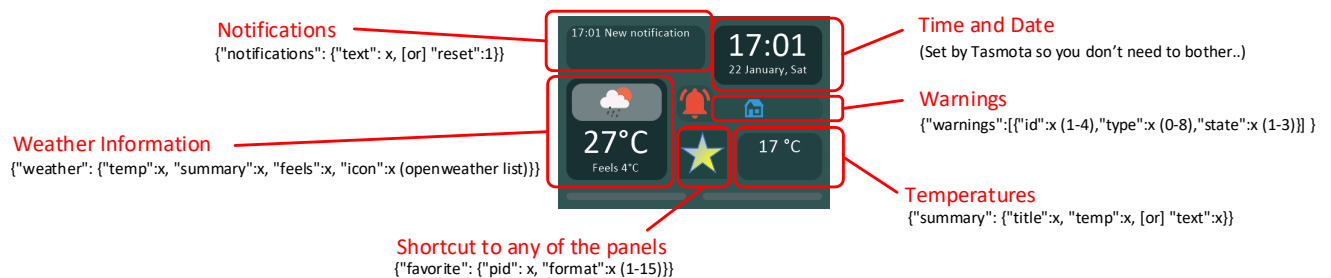
You possibly also want to configure the two physical buttons below the display. Two options are available:

1. Toggle the relays (if you want to use the relays to control something)
2. Only toggle the *status lights* above the buttons (if you are not using the relays but want to use the buttons to control some of your items).

Configuration and installation

Got 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

NXPanel1 (Generic MQTT Thing)

Channel

NXPanel Weather Command
 mqtt.topic:mosquito_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
 cmd/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: **nxpanel_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/Power1**
 - As QoS; Enter: **Exactly Once**
 - As **Incoming Value Transformations**, Enter:
REGEX:(.*POWER1.*)~JSONPATH:\$.POWER1
 - As **Outgoing Value Format**, Enter: **%s**
 - The channel for Switch1 should now look something like this:

NXPanel Switch1
NSPanel1 (Generic MQTT Thing)

Channel

NXPanel Switch1
mqtttopic:mosquitto_sweden3:nspanel1:nxpanel_switch1

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/Power1

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

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

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
%s

- 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/Power2**
 - As QoS; Enter: **Exactly Once**
 - As **Incoming Value Transformations**, Enter: **REGEX:(.*POWER2.*)~JSONPATH:\$.POWER2**
 - As **Outgoing Value Format**, Enter: **%s**
 - The channel for Switch2 should now look something like this:

NXPanel Switch2
 NSPanel1 (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/Power2

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

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

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 ~.

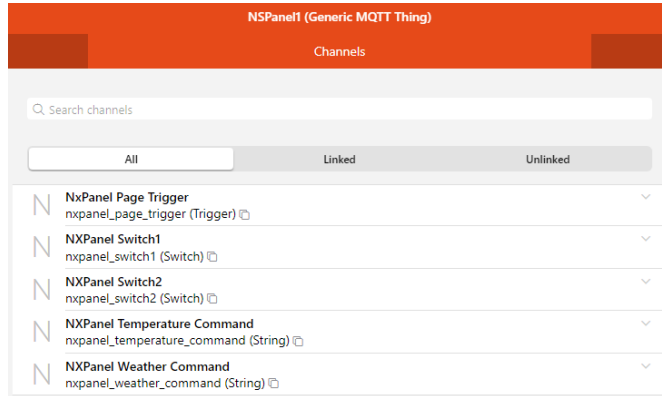
Outgoing Value Transformation

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
 %s

- 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:



NSPanel Buttons – If you don’t use the relays

If you want to use the physical buttons to control two of your items but *don’t want to toggle the relays* (they are kind of noisy..) you can still maintain the sync of the *on/off light* above each of your buttons with the following changes to the button channels.

NXPanel Switch1 – Alternative configuration:

- As MQTT Command Topic; Enter: [cmnd/nspanel_7DD7FC/nxpanel](#)
- As Custom On/Open Value; Enter: **1**
- As Custom Off/Closed Value; Enter: **0**
- As **Outgoing Value Format**, Enter: { "switches": { "switch1": %s } }

NXPanel Switch2 – Alternative configuration:

- As MQTT Command Topic; Enter: [cmnd/nspanel_7DD7FC/nxpanel](#)
- As Custom On/Open Value; Enter: **1**
- As Custom Off/Closed Value; Enter: **0**
- As **Outgoing Value Format**, Enter: { "switches": { "switch2": %s } }

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>"}

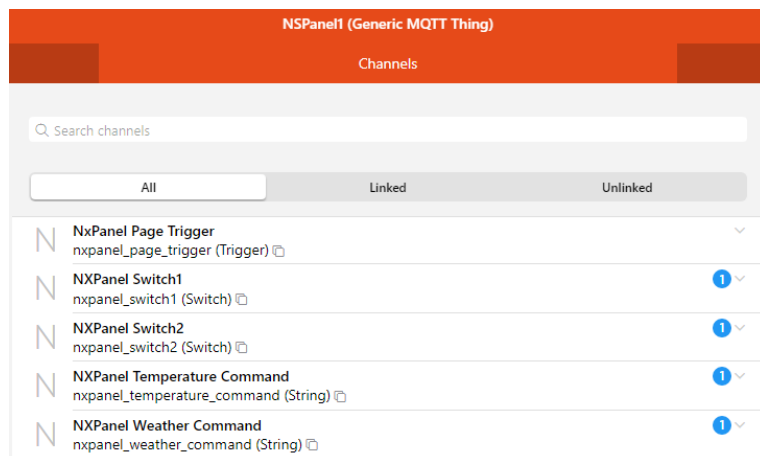
// Weather Forecast Items from OpenWeatherMap
Number:Temperature Forecast_Temp_1day {channel="openweathermap:onecall:0bd084e6fe:local:forecastTomorrow#day-temperature"}
Number:Temperature Forecast_Feels_1day {channel="openweathermap:onecall:0bd084e6fe:local:forecastTomorrow#apparent-day"}
String Forecast_WeatherIcon_1day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

The screenshot shows the configuration interface for an 'NXPanel Weather Rule'. At the top, the title 'NXPanel Weather Rule' is displayed in a red header. Below the title, the status is 'IDLE' with a green indicator. There are pause and play buttons. The configuration is divided into sections: 'When' (triggers) and 'Then' (actions). The 'When' section has a 'Reorder' button and lists three triggers: 'When Forecast_Temperature changed', 'When Forecast_Weather_Icon changed', and 'When Forecast_Feels_Temp changed'. Each trigger has a description: 'This triggers the rule if an item state has changed.' The 'Then' section has an 'Add Action' button and lists one action: 'execute a given script' with a description: 'Allows the execution of a user-defined script.' The 'But only if' section has an 'Add Condition' button.

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 “**Fel! Hittar inte referenskölla.**” 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\": \"Out %d°C\", \"text\": \"In %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 page for a rule titled "NXPanel Temperature Rule". At the top, the status is "IDLE" with a green indicator and pause/play buttons. Below this is a table with the following details:

Unique ID	d91452c50f
Name	NXPanel Temperature Rule
Description	

Below the table is a "Reorder" button. The rule is configured with the following sections:

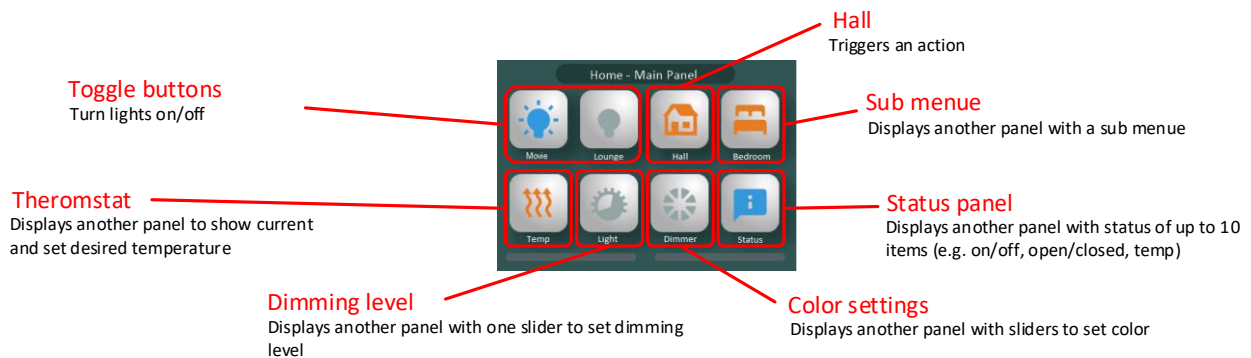
- When:** Contains two triggers:
 - When Current_Outdoor_Temp changed (This triggers the rule if an item state has changed.)
 - When Current_Indoor_Temp changed (This triggers the rule if an item state has changed.)
 There is an "Add Trigger" button at the bottom of this section.
- Then:** Contains one action:
 - execute a given script (Allows the execution of a user-defined script.)
 There is an "Add Action" button at the bottom of this section.
- But only if:** Contains one condition:
 - Add Condition

At this stage you have your first panel working!

8. Custom panel configuration

To configure the panels (sometimes also called “pages” or “screens”) you need to know the different artifacts that the NXPanel supports. This chapters starts with describing the available artifacts – button types and page types - before showing how these are configured and connected to your OpenHAB system.

The demo panels shown in the beginning of the post [NxPanel - Replacement firmware for Sonoff NSPanel](#) together with Mikes Groovy script are used as reference. As a teaser, the first panel generated by Mikes Groovy script is an eight-button panel configured as shown in in the pic below.



Button Types

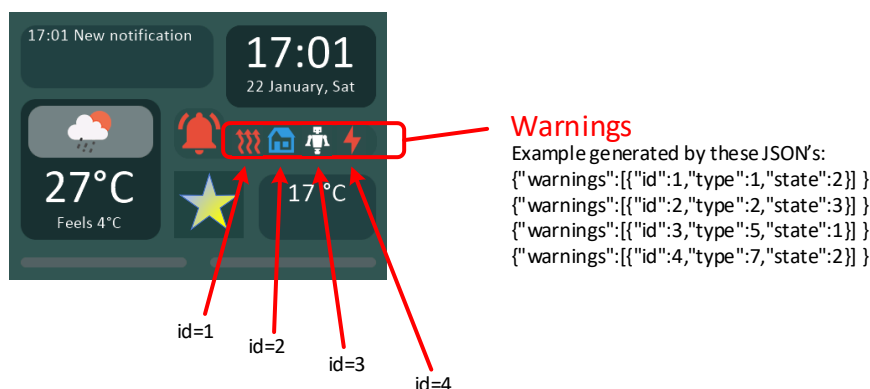
The NXPanel supports six different *button types*, and each type has a unique Id. The button *type* defines the *function* of the button. To keep the correlation with the demo pages, the table also includes the variable name used in Mikes Groovy script. The JSON's in the table are examples of what's sent *from* NXPanel when the button is pressed (the JSON sent *to* the NXPanel is covered in section Panel Types).

Variable Name	Button Id	Function
BUTTON_UNUSED	0	A placeholder button with no function. <u>JSON response example:</u> <Nothing is sent when pushed>
BUTTON_TOGGLE	1	When pressed, it toggles the button state to 1 (ON) or 0 (OFF) <u>JSON response example:</u> { "button": { "pid": 10, "bid": 1, "state": 1, "next": 0 } }
BUTTON_PUSH	2	When pressed, it just generates a JSON that it's been pressed to e.g., trigger an action or workflow that does not need a state change. <u>JSON response example:</u> { "button": { "pid": 10, "bid": 3, "state": 0, "next": 0 } }
BUTTON_DIMMER	3	Short press it toggles the button state to 1 (ON) or 0 (OFF). Long press renders a page with a toggle button and dimmer slider. <u>JSON response example, short press:</u> { "dimmer": { "pid": 18, "power": 0 } } <u>JSON Example, long press</u> { "page": { "format": 7, "pid": 18, "type": "refresh" } }

Variable Name	Button Id	Function
BUTTON_DIMMER_COLOR	4	<p>Short press it toggles the button state to 1 (ON) or 0 (OFF). Long press renders a page with a toggle button and dimmer slider.</p> <p><u>JSON response example, short press</u> {"dimmer":{"pid":16,"power":1}}</p> <p><u>JSON Example, long press</u> {"dimmer":{"pid":16,"power":0,"hsbcolor":"180,100,50"}}</p>
BUTTON_PAGE	10	<p>When pressed, it sends a request to render a new page.</p> <p><u>JSON response example:</u> {"page":{"format": 5, "pid": 11, "type": "sync"}}</p>














Warning Types

The warning field consists of four configurable locations for warning states. See example below.







The table below shows available Icons with their corresponding types and states.































Icon	Icon Description	Icon Type	Icon State	Icon Color	JSON
	blank	0	0	None	{ "warnings":[{"id":x,"type":0,"state":0}] }
	heat	1	1	White	{ "warnings":[{"id":x,"type":1,"state":1}] }
	heat	1	2	Red	{ "warnings":[{"id":x,"type":1,"state":2}] }
	heat	1	3	Blue	{ "warnings":[{"id":x,"type":1,"state":3}] }
	house	2	1	White	{ "warnings":[{"id":x,"type":2,"state":1}] }
	house	2	2	Red	{ "warnings":[{"id":x,"type":2,"state":2}] }
	house	2	3	Blue	{ "warnings":[{"id":x,"type":2,"state":3}] }
	light	3	1	White	{ "warnings":[{"id":x,"type":3,"state":1}] }
	light	3	2	Red	{ "warnings":[{"id":x,"type":3,"state":2}] }


Icon	Icon Description	Icon Type	Icon State	Icon Color	JSON
	light	3	3	Blue	{ "warnings":[{"id":x,"type":3,"state":3}] }
	plug	4	1	White	{ "warnings":[{"id":x,"type":4,"state":1}] }
	plug	4	2	Red	{ "warnings":[{"id":x,"type":4,"state":2}] }
	plug	4	3	Blue	{ "warnings":[{"id":x,"type":4,"state":3}] }
	robot	5	1	White	{ "warnings":[{"id":x,"type":5,"state":1}] }
	robot	5	2	Red	{ "warnings":[{"id":x,"type":5,"state":2}] }
	robot	5	3	Blue	{ "warnings":[{"id":x,"type":5,"state":3}] }
	speaker	6	1	White	{ "warnings":[{"id":x,"type":6,"state":1}] }
	speaker	6	2	Red	{ "warnings":[{"id":x,"type":6,"state":2}] }
	speaker	6	3	Blue	{ "warnings":[{"id":x,"type":6,"state":3}] }
	zap	7	1	White	{ "warnings":[{"id":x,"type":7,"state":1}] }
	zap	7	2	Red	{ "warnings":[{"id":x,"type":7,"state":2}] }
	zap	7	3	Blue	{ "warnings":[{"id":x,"type":7,"state":3}] }
N/A	dustbin	8	1	White	{ "warnings":[{"id":x,"type":8,"state":1}] }
N/A	dustbin	8	2	Red	{ "warnings":[{"id":x,"type":8,"state":2}] }
N/A	dustbin	8	3	Blue	{ "warnings":[{"id":x,"type":8,"state":3}] }

Icons

Icons below can be assigned to the buttons. Pictures for a few are missing. Also, haven't figured out how the color (state) is controlled, for toggle buttons this works by itself, but for other cases, e.g., buttons taking you to another panel, this is still a bit of a mystery for me.

Icon	Icon Description	Icon Type	Icon State	Comment
	blank	0		Blank
	bulb	1		Light1
	bulb	1		Light1a
	bulb	1		Light2

Icon	Icon Description	Icon Type	Icon State	Comment
	bulb	1		Light3
	dimmer	2		Dimmer1
	dimmer	2		Dimmer2
	dimmer	2		Dimmer3
	dimmer color	3		Rgb1
	dimmer color	3		Rgb2
	vaccum	4		Vaccum1
	vaccum	4		Vaccum2
	vaccum	4		Vaccum3
	bed	5		Bed1
	bed	5		Bed2
	bed	5		Bed3
	house	6		House1
	house	6		House2
	house	6		House3
	sofa	7		Sofa1
	sofa	7		Sofa2
	sofa	7		Sofa3
	bell	8		Bell1
	bell	8		Bell2
	bell	8		Bell3
	heat	9		Heat1
	heat	9		Heat2
	heat	9		Heat3
	curtains	10		Curtain1
	curtains	10		Curtain2
	curtains	10		Curtain3
	music	11		Music1
	music	11		Music2
	music	11		Music3
N/A	binary	12		
N/A	binary	12		
N/A	binary	12		
N/A	fan	13		
N/A	fan	13		
N/A	fan	13		
N/A	switch	14		
N/A	switch	14		

Icon	Icon Description	Icon Type	Icon State	Comment
N/A	switch	14		
N/A	talk	15		
N/A	talk	15		
N/A	talk	15		
N/A	Info	16		
	info	16		
N/A	info	16		

Panel Types

Designing panels is when this becomes really fun. I've chosen to use the demo panels as a reference. As these differ quite much from the panels generated by Mikes Groovy script, the panel definitions in the script – from now on called **Adopted Groovy Script** - are completely replaced. I've also found some small bug (I think) in the Groovy script that I've fixed (its about handling of the panel format which I haven't understood when to use or not).

The documentation principle I followed was to provide you an extensive example to facilitate understanding how this works and for you to steal ideas from when you design the panels to fit your needs.

Panel Design – Work order

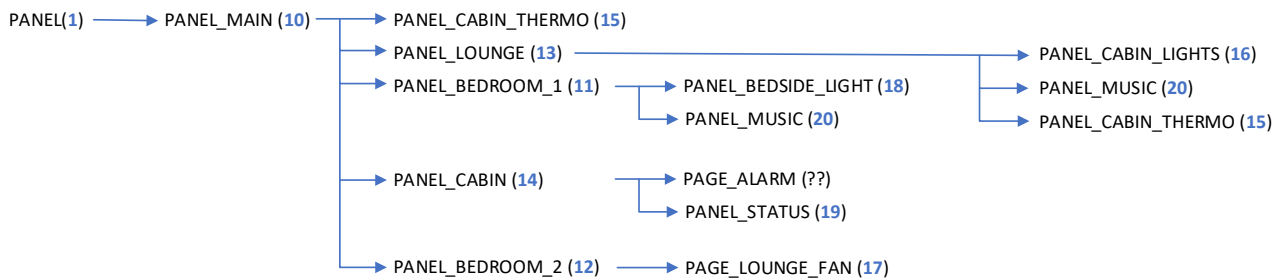
When designing panels, the basic work order is:

1. Make a drawing of the panels you want and the buttons of these panels
2. Make a connection diagram – How panels are calling each other
3. **Note!** Due to memory constraint, I advise you to use each button page only once. It will still work if you use e.g., a four-button panel twice, but it then the panel needs to be re-rendered each time it's called – an in comparison slow process (upside is that it's still A LOT more functionality and configuration options that the Sonoff stock firmware 😊).
4. Design/adopt the Groovy script. I suggest you don't hook up your items to the buttons before your panels are working, just simulate the items for now by creating variables for the item states and assign them some value (e.g., 0 or 1).
5. Create the channels for the buttons on your panels.
6. Link your items to the channels.
7. In the Groovy script, replace the simulated item states with the state/value of your actual items.

Panel Design – Example



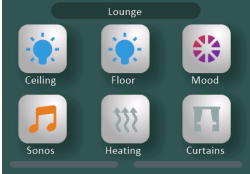
The NXPanel supports 15 different panel types, and as for buttons, each *panel type* has an unique number 1-12 (**green** in the table below). The *panel type* defines the *function* of the panel. The *panel Id* is something you decide yourself except for panel 1 (the home panel) and can be any number. These are marked in **blue** in the examples below. Again, variables match the *Adopted Groovy Script* and the demo panels and also examples of the different JSON's sent *to* and *from* NXPanel to control the panels and to configure the different buttons or fields. I suggest you look at the *Adopted Groovy Script* in parallel with the JSON examples in the table below.



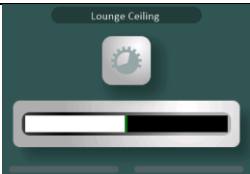
So now to the design example. Below is the connection diagram for the panels in this example. I've used the panel variable names in the diagram.



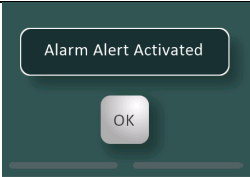






Again, variables match the *Adopted Groovy Script* and the demo panels and also examples of the different JSON's sent *to* and *from* NXPanel to control the panels and to configure the different buttons or fields. I suggest you look at the *Adopted Groovy Script* in parallel with the JSON examples in the table below.

Panel Type Name	Panel Type	Panel Id	JSON Examples	Demo Panel
PAGE_HOME	1	1	<p><u>Top left – Notifications posted, max 4 in a list that scrolls upwards</u> {"notifications": {"text": x, [or] "reset": 1}}</p> <p><u>Top right – Time and date maintained by Tasmota</u> {"clock": {"date": x, "hour": x, "min": x, "month": x, "weekday": x}}</p> <p><u>Bottom left – Weather information</u> {"weather": {"temp": x, "icon": x, "feels": x, [or] "summary": x}} {"weather": {"temp": 27, "icon": "04d", "feels": 4}} {"weather": {"temp": 27, "icon": "04d", "summary": "Humidity 65%"}}</p> <p><u>Bottom right – 2xTemperature or 1xTemperature + 1x Text</u> {"summary": {"title": "17°C", "temp": x, [or] "text": x}}</p> <p><u>Panel to show when you swipe left</u> {"start": {"pid": x, "format": x (1-15)}} {"start": {"pid": 10, "format": 6}} // Default Panel id 10, format is 8-button panel {"start": {"pid": 11, "format": 4}} // Panel id 11, format is 4-button panel</p> <p><u>Bottom middle – Panel to show when favorite button pressed</u> {"favorite": {"pid": x, "format": x (1-15)}}</p> <p><u>NXPanel default brightness and dimming levels</u> {"dim": {"low": n, "normal": n}}</p> <p><u>Status light of built in relays (unknowns how replays are controlled)</u> {"switches": {"switch1": x, "switch2": x}} // also called by tasmota. Note! This only toggles the status lights but not the relays. For operating the relays, see chapter 7: NXPanel Thing Definition.</p> <p><u>Middle right – Banner with warning and state symbols</u> {"warnings": [{"id": x (1-4), "type": x (0-7), "state": x (1-3)},...]}}</p>	
PAGE_2_BUTTON	2	12	<p><u>Refresh request from NXPanel</u> {"format": 2, "pid": 12, "type": "refresh"}}</p> <p><u>Refresh answer from OpenHAB</u> {"refresh": {"pid": 12, "name": "Teds bedroom", "format": 2, "buttons": [{"bid": 1, "label": "Xbox", "type": 1, "state": 1, "icon": 14}, {"bid": 2, "label": "Fan", "type": 3, "next": 17, "state": 1, "icon": 13}]}}}</p> <p><u>Sync request from NXPanel</u> {"page": {"format": 2, "pid": 12, "type": "sync"}}</p>	

Panel Type Name	Panel Type	Panel Id	JSON Examples	Demo Panel
			<p><u>Sync answer from OpenHAB</u></p> <pre>{ "sync": { "pid": 12, "buttons": [{ "bid": 1, "state": 1 }, { "bid": 2, "state": 1 }] } }</pre> <p><u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u></p> <pre>{ "button": { "pid": 12, "bid": 1, "state": 0, "next": 0 } } { "dimmer": { "pid": 17, "power": 0 } }</pre>	
PAGE_3_BUTTON	3	14	<p><u>Refresh request from NXPanel</u></p> <pre>{ "page": { "format": 3, "pid": 14, "type": "refresh" } }</pre> <p><u>Refresh answer from OpenHAB</u></p> <pre>{ "refresh": { "pid": 14, "name": "Hall Lightning", "format": 3, "buttons": [{ "bid": 1, "label": "Front", "type": 1, "state": 1, "icon": 1 }, { "bid": 2, "label": "Alarm", "type": 10, "next": 0, "state": 12, "icon": 8 }, { "bid": 3, "label": "Status", "type": 10, "next": 19, "state": 15, "icon": 16 }] } }</pre> <p><u>Sync request from NXPanel</u></p> <pre>{ "page": { "format": 3, "pid": 14, "type": "sync" } }</pre> <p><u>Sync answer from OpenHAB</u></p> <pre>{ "sync": { "pid": 14, "buttons": [{ "bid": 1, "state": 1 }] } }</pre> <p><u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u></p> <pre>{ "button": { "pid": 14, "bid": 1, "state": 0, "next": 0 } }</pre>	 <p>(Added buttons for Alarm and Status as these did not exist in the sample panels)</p>
PAGE_4_BUTTON	4	11	<p><u>Refresh request from NXPanel</u></p> <pre>{ "format": 4, "pid": 11, "type": "refresh" }</pre> <p><u>Refresh answer from OpenHAB</u></p> <pre>{ "refresh": { "pid": 11, "name": "Master Bedroom", "format": 4, "buttons": [{ "bid": 1, "label": "Cieling", "type": 1, "state": 0, "icon": 1 }, { "bid": 2, "label": "Bedside", "type": 3, "next": 18, "state": 1, "icon": 2 }, { "bid": 3, "label": "Music", "type": 10, "next": 20, "state": 13, "icon": 11 }, { "bid": 4, "label": "Curtains", "type": 1, "state": 1, "icon": 10 }] } }</pre> <p><u>Sync request from NXPanel</u></p> <pre>{ "page": { "format": 4, "pid": 11, "type": "sync" } }</pre> <p><u>Sync answer from OpenHAB</u></p> <pre>{ "sync": { "pid": 11, "buttons": [{ "bid": 1, "state": 0 }, { "bid": 2, "state": 1 }, { "bid": 4, "state": 1 }] } }</pre> <p><u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u></p> <pre>{ "dimmer": { "pid": 18, "power": 0 } } // Short press Dimmer</pre>	
PAGE_6_BUTTON	5	13	<p><u>Refresh request from NXPanel</u></p> <pre>{ "page": { "format": 5, "pid": 13, "type": "refresh" } }</pre> <p><u>Refresh answer from OpenHAB</u></p> <pre>{ "refresh": { "pid": 13, "name": "Lounge", "format": 5, "buttons": [{ "bid": 1, "label": "Cieling", "type": 1, "state": 1, "icon": 1 }, { "bid": 2, "label": "Floor", "type": 1, "state": 1, "icon": 1 }, { "bid": 3, "label": "Mood", "type": 4, "next": 16, "state": 1, "icon": 3 }, { "bid": 4, "label": "Sonos", "type": 10, "next": 20, "state": 13, "icon": 11 }, { "bid": 5, "label": "Heating", "type": 10, "next": 15, "state": 9, "icon": 9 }, { "bid": 6, "label": "Curtains", "type": 1, "state": 0, "icon": 10 }] } }</pre> <p><u>Sync request from NXPanel</u></p> <pre>{ "page": { "format": 5, "pid": 13, "type": "sync" } }</pre> <p><u>Sync answer from OpenHAB (buttons 1, 2, 3 & 6)</u></p>	

Panel Type Name	Panel Type	Panel Id	JSON Examples	Demo Panel
			<pre>{ "sync": { "pid": 13, "buttons": [{ "bid": 1, "state": 1 }, { "bid": 2, "state": 1 }, { "bid": 3, "state": 1 }, { "bid": 6, "state": 0 }] } }</pre> <p>Commands sent from NXPanel by pressing buttons/using sliders on panel</p> <pre>{ "button": { "pid": 13, "bid": 1, "state": 0, "next": 0 }, "dimmer": { "pid": 16, "power": 0 } }</pre>	
PAGE_8_BUTTON	6	10	<p><u>Refresh request from NXPanel</u></p> <pre>{ "page": { "format": 6, "pid": 10, "type": "refresh" } }</pre> <p><u>Refresh answer from OpenHAB</u></p> <pre>{ "refresh": { "pid": 10, "name": "Home - Main Panel", "format": 6, "buttons": [{ "bid": 1, "label": "Hall", "type": 1, "state": 0, "icon": 1 }, { "bid": 2, "label": "Lounge", "type": 1, "state": 0, "icon": 1 }, { "bid": 3, "label": "Bed 1", "type": 2, "icon": 1 }, { "bid": 4, "label": "Heating", "type": 10, "next": 15, "state": 9, "icon": 9 }, { "bid": 5, "label": "Lounge", "type": 10, "next": 13, "state": 5, "icon": 7 }, { "bid": 6, "label": "Study", "type": 10, "next": 11, "state": 4, "icon": 7 }, { "bid": 7, "label": "Cabin", "type": 10, "next": 14, "state": 3, "icon": 6 }, { "bid": 8, "label": "Garage", "type": 10, "next": 12, "state": 2, "icon": 6 }] } }</pre> <p><u>Sync request from NXPanel</u></p> <pre>{ "format": 6, "pid": 10, "type": "sync" }</pre> <p><u>Sync answer from OpenHAB</u></p> <pre>{ "sync": { "pid": 10, "buttons": [{ "bid": 1, "state": 0 }, { "bid": 2, "state": 0 }] } }</pre> <p>Commands sent from pressing buttons/sliders on panel</p> <pre>{ "button": { "pid": 10, "bid": 1, "state": 0, "next": 0 } }</pre>	
PAGE_DIMMER	7	17	<p><u>Refresh request from NXPanel</u></p> <pre>{ "page": { "format": 7, "pid": 17, "type": "refresh" } }</pre> <p><u>Refresh answer from OpenHAB</u></p> <pre>{ "refresh": { "pid": 17, "name": "Lounge Fan", "power": "ON", "min": 1, "max": 4, "icon": 13, "dimmer": 3 } }</pre> <p><u>Sync request from NXPanel</u></p> <p>N/A</p> <p><u>Sync answer from OpenHAB</u></p> <p>N/A</p> <p>Commands sent from NXPanel by pressing buttons/using sliders on panel</p> <pre>{ "dimmer": { "pid": 17, "power": 0, "dimmer": 2 } }</pre>	
PAGE_DIMMER	7	18	<p><u>Refresh request from NXPanel</u></p> <pre>{ "page": { "format": 7, "pid": 18, "type": "refresh" } }</pre> <p><u>Refresh answer from OpenHAB</u></p> <pre>{ "refresh": { "pid": 18, "name": "Lounge Light", "power": "ON", "dimmer": 30 } }</pre> <p><u>Sync request from NXPanel</u></p> <p>N/A</p> <p><u>Sync answer from OpenHAB</u></p> <p>N/A</p> <p>Commands sent from NXPanel by pressing buttons/using sliders on panel</p> <pre>{ "dimmer": { "pid": 18, "power": 1 } }</pre> <p>// sent when pressing on "calling" page</p> <pre>{ "dimmer": { "pid": 18, "power": 1, "dimmer": 66 } }</pre> <p>// sent from this panel</p>	

Panel Type Name	Panel Type	Panel Id	JSON Examples	Demo Panel
PAGE_DIMMER_COLOR	8	16	<u>Refresh request from NXPanel</u> <pre>{"page": {"format": 8, "pid": 16, "type": "refresh"}}</pre> <u>Refresh answer from OpenHAB</u> <pre>{"refresh":{"pid":16,"name":"Bedroom mood light", "power":ON, "hsbcolor":"10,100,50"}}</pre> <u>Sync request from NXPanel</u> <u>Sync answer from OpenHAB</u> <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u> <pre>{"dimmer":{"pid":16,"power":1,"hsbcolor":"180,100,68"}}</pre>	
PAGE_THERMOS_TAT	9	15	<u>Refresh request from NXPanel</u> <pre>{"page": {"format": 9, "pid": 15, "type": "refresh"}}</pre> <u>Refresh answer from OpenHAB</u> <pre>{"refresh":{"pid":15,"name":"Cabin", "therm":{"set":14,"temp":0,"heat":1,"state":0"}}</pre> <u>Sync request from NXPanel</u> <u>Sync answer from OpenHAB</u> <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u> <pre>{"therm":{"pid": 15, "set": 14, "state": 1}} // Toggle heating</pre> <pre>{"therm":{"pid": 15, "set": 15, "state": 0}} // Increase temp</pre>	
PAGE_ALERT_1	10	??	<u>Refresh request from NXPanel</u> <u>Refresh answer from OpenHAB</u> <u>Sync request from NXPanel</u> <u>Sync answer from OpenHAB</u> <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u>	
PAGE_ALERT_2	11	??	??	??
PAGE_ALARM	12	??	<u>Refresh request from NXPanel</u> <u>Refresh answer from OpenHAB</u> <u>Sync request from NXPanel</u> <u>Sync answer from OpenHAB</u> <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u> <pre>{"alarm":{"code":"1234","type":1}} // Enter code</pre>	
PAGE_MEDIA	13	20	<u>Refresh request from NXPanel</u> <pre>{"page": {"format": 13, "pid": 20, "type": "refresh"}}</pre> <u>Refresh answer from OpenHAB</u> <pre>{"refresh":{"pid":20,"name":"Music Room", "artist":"New Order", "album":"Power, Corruption & Lies", "track":"Blue Monday", "volume":70}}</pre> <u>Sync request from NXPanel</u> N/A (will always ask for refresh) <u>Sync answer from OpenHAB</u> N/A <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u> <pre>{"media":{"pid":20,"volume":80}} // Press volume</pre> <pre>{"media":{"pid":20,"action":"play","volume":70}} // Press Play</pre> <pre>{"media":{"pid":20,"action":"pause","volume":70}} // Press Pause</pre> <pre>{"media":{"pid":20,"action":"next","volume":70}} // Press Next</pre>	

Panel Type Name	Panel Type	Panel Id	JSON Examples	Demo Panel
PAGE_PLAYLIST	14		<u>Refresh request from NXPanel</u> <u>Refresh answer from OpenHAB</u> <u>Sync request from NXPanel</u> <u>Sync answer from OpenHAB</u> <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u>	
PAGE_STATUS	15	19	<u>Refresh request from NXPanel</u> {"page": {"format": 15, "pid": 19, "type": "refresh"}} <u>Refresh answer from OpenHAB</u> {"refresh":{"pid":19,"name":"System Status", "status":[{"id":1,"text":"Gate";,"value":"Open","color":2}, {"id":2,"text":"Window";,"value":"Shut","color":3}, {"id":5,"text":"Room Temp";,"value":"20°C"}]}} <u>Sync request from NXPanel</u> N/A (will always ask for refresh) <u>Sync answer from OpenHAB</u> N/A <u>Commands sent from NXPanel by pressing buttons/using sliders on panel</u> N/A	

Adopted Groovy Script

With Mikes Groovy script as model, the below script has been adopted to match the demo panels. The *Adopted Groovy Script* below renders all the demo panels (with the exception of the Alarm panels where I miss examples myself).

```

/*Imports and global definitions */
import org.slf4j.LoggerFactory
def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

/*-----
 * Custom Configurations
 * Note1! Replace MQTT channel with that of your mosquitto broker
 * Note2! Set TOPIC to the value of your NSPanel
 * (This is where the JSON string built by this script is posted)
 *-----*/
def mqtt = actions.get("mqtt","mqtt:broker:mosquitto_sweden3")
def TOPIC = "cmd/nspanel_7DD7FC/nxpanel"

/* Definition of Panel types (page types) for available layouts
 * NOTE! This is not the same as the PanelID (id) parameter but instead one of
 * 15 predefined "panel designs"
 */
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

/* Definition of button types */
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

/* Definition of icon types */
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

// Get the JSON string when event has triggered
def str = event.getEvent()

logger.info("NXPanel Trigger - Script triggered by event: <"+str+">")

// Return if the string {"page": is not found in the JSON
//This basically means that all other events posted on this channel are dropped
if (str.indexOf('{"page":')!=0) {
    logger.info("NXPanel Trigger - Not a page event, returning...")
    return
}

/*-----
 * Utility functions - start
 *-----*/

/* Build a JSON string segment for a button, parameters are:
 *   bid   - Button id (bid==1 is the first button on a page)
 *   label - Label under the button
 *   type  - button type
 *   icon  - Button icon to use [or null]
 *   state - Button state [or null]
 *   next  - Panel ID of next panel [or null]
 */
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
}

```

```

/* Build a JSON string segment for a button page, parameters are:
 * pid - Panel id
 * name - JSON with all buttons on this panel
 * format - Page type
 */
def makePage(pid,name,format) {
    var str = new StringBuilder("{\"refresh\":'")
    str<<{"pid":'<<pid<<',"name":"'<<name<<',"format":"'<<format<<',"buttons:['
    //str<<{"pid":'<<pid<<',"name":"'<<name<<',"
    return str
}

/* Build a JSON string segment for a dimmer page, parameters are:
 * pid - Panel id
 * name - JSON with all buttons on this panel
 */
def makePage2(pid,name) {
    var str = new StringBuilder("{\"refresh\":'")
    str<<{"pid":'<<pid<<',"name":"'<<name<<',"
    return str
}

/* Build a JSON string header for a sync for a page
 * pid - Panel id
 */
def makeEmptySync(pid) {
    var str = new StringBuilder("{\"sync:'")
    str<<{"pid":'<<pid<<'}}'
    return str
}

/* Build a JSON string header for a page refresh (just update one page), parameters are:
 * pid - Panel id
 */
def makeEmptyRefresh(pid) {
    var str = new StringBuilder("{\"refresh\":'")
    str<<{"pid":'<<pid<<'}}'
    return str
}

/* Build a JSON string for a button state sync (just update state of button)
 * pid - Panel id
 * bid - Button id
 * state - Button state
 */
def makeSyncButtonStart(pid,bid,state) {
    var str = new StringBuilder("{\"sync:'")
    str<<{"pid":'<<pid
    str<<',"buttons":[{"bid":'<<bid<<',"state":'<<state<<'}}'
    return str
}

/* Build a JSON string for a button state sync (just update state of button)
 * bid - Button id
 * state - Button state
 */
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)
 */
logger.info("NXPanel Trigger - Analyzing id and format..")

```

```

// str contains the JSON string.
// Extract the panel id (var id) and the panel type (var format) from str
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
// Uncomment if you want to do a full refresh every time
//refresh = 1 > 0

logger.info("NXPanel Trigger - id="+id+", format="+format)

// Empty variable to contain the return JSON
var json

// This is YOUR design. Define the panels and their corresponding panel id
// (This basically maps all the panels you have designed)
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_BEDSIDE_LIGHT = 18
def PANEL_STATUS = 19
def PANEL_MUSIC = 20

// Just send a message to the log that your received a post from NSPanel
logger.info("NXPanel Trigger - Updating page ... "+id)

// Check wich of your panels NSPanel wants you to process
switch (id as int) {

    case PANEL_MAIN :
        logger.info("NXPanel Trigger - PANEL_MAIN")
        // set these from your own items
        //movie_state = 1
        hall_state = ir.getItem("P10B1_Hall").state==ON?1:0
        //lounge_state = 1
        lounge_state = ir.getItem("P10B2_Lounge").state==ON?1:0

        if (refresh) {
            // NSPanel has asked you to render the entire panel
            // Define the layout of your panel
            json = makePage(id,'Home - Main Panel',format)
            json<<makeButton(1,"Hall",BUTTON_TOGGLE,ICON_BULB,hall_state)
            json<<makeButton(2,"Lounge",BUTTON_TOGGLE,ICON_BULB,lounge_state)
            json<<makeButton(3,"Bed 1",BUTTON_PUSH,ICON_BULB)
            json<<makeButton(4,"Heating",BUTTON_PAGE,ICON_HEAT,PAGE_THERMOSTAT,PANEL_CABIN_THERMO)
            json<<makeButton(5,"Lounge",BUTTON_PAGE,ICON_SOFA,PAGE_6_BUTTON,PANEL_LOUNGE)
            json<<makeButton(6,"Study",BUTTON_PAGE,ICON_SOFA,PAGE_4_BUTTON,PANEL_BEDROOM_1)
            json<<makeButton(7,"Cabin",BUTTON_PAGE,ICON_HOUSE,PAGE_3_BUTTON,PANEL_CABIN)
            json<<makeButton(8,"Garage",BUTTON_PAGE,ICON_HOUSE,PAGE_2_BUTTON,PANEL_BEDROOM_2)
            json<<""]}"
        } else {
            // NSPanel has asked you to just update the states of your buttons on this panel
            json = makeSyncButtonStart(id,1,hall_state)
            json<<addSyncButton(2,lounge_state)
            json<<""]}"
        }
        logger.info("Panel: 10, Sending JSON:"+json.toString())
        mqtt.publishMQTT(TOPIC, json.toString())
        break

    case PANEL_BEDROOM_1 :

```

```

logger.info("NXPanel Trigger - PANEL_BEDROOM_1")
// set these from your own items
cieling_state = 0
//cieling_state = ir.getItem("P11B1_Ceiling").state==ON?1:0
bedside_state = 1
//bedside_state = ir.getItem("P18_Bedside_Light").state==ON?1:0
curtains_state = 1
if (refresh) {
    // NSPanel has asked you to render the entire panel
    // Define the layout of your panel
    json = makePage(id, 'Master Bedroom', format)
    json<<makeButton(1, "Cieling", BUTTON_TOGGLE, ICON_BULB, cieling_state)

json<<makeButton(2, "Bedside", BUTTON_DIMMER, ICON_DIMMER, bedside_state, PANEL_BEDSIDE_LIGHT)
    json<<makeButton(3, "Music", BUTTON_PAGE, ICON_MUSIC, PAGE_MEDIA, PANEL_MUSIC)
    json<<makeButton(4, "Curtains", BUTTON_TOGGLE, ICON_CURTAINS, curtains_state)
    json<<"]]}"
} else {
    // NSPanel has asked you to just update the states of your buttons on this panel
    json = makeSyncButtonStart(id, 1, cieling_state)
    json<<addSyncButton(2, bedside_state)
    json<<addSyncButton(4, curtains_state)
    json<<"]]}"
}
logger.info("Panel: "+PANEL_BEDROOM_1+", Sending JSON:"+json.toString())
mqtt.publishMQTT(TOPIC, json.toString())
break
case PANEL_BEDROOM_2 :
logger.info("NXPanel Trigger - PANEL_BEDROOM_2")
// set these from your own items
xbox_state = 1
fan_state = 1
if (refresh) {
    // NSPanel has asked you to render the entire panel
    // Define the layout of your panel
    json = makePage(id, 'Teds bedroom', format)
    json<<makeButton(1, "Xbox", BUTTON_TOGGLE, ICON_SWITCH, xbox_state)
    json<<makeButton(2, "Fan", BUTTON_DIMMER, ICON_FAN, fan_state, PANEL_LOUNGE_FAN)
    json<<"]]}"
} else {
    // NSPanel has asked you to just update the states of your buttons on this panel
    json = makeSyncButtonStart(id, 1, xbox_state)
    json<<addSyncButton(2, fan_state)
    json<<"]]}"
}
logger.info("Panel: "+PANEL_BEDROOM_2+", Sending JSON:"+json.toString())
mqtt.publishMQTT(TOPIC, json.toString())
break
case PANEL_LOUNGE :
// set these from your own items
cieling_state = 1
floor_state = 1
mood_state = 1
curtains_state = 0
if (refresh) {
    // NSPanel has asked you to render the entire panel
    // Define the layout of your panel
    json = makePage(id, 'Lounge', format)
    json<<makeButton(1, "Cieling", BUTTON_TOGGLE, ICON_BULB, cieling_state)
    json<<makeButton(2, "Floor", BUTTON_TOGGLE, ICON_BULB, floor_state)

json<<makeButton(3, "Mood", BUTTON_DIMMER_COLOR, ICON_DIMMER_COLOR, mood_state, PANEL_CABIN_LIGHTS)
    json<<makeButton(4, "Sonos", BUTTON_PAGE, ICON_MUSIC, PAGE_MEDIA, PANEL_MUSIC)
    json<<makeButton(5, "Heating", BUTTON_PAGE, ICON_HEAT, PAGE_THERMOSTAT, PANEL_CABIN_THERMO)
    json<<makeButton(6, "Curtains", BUTTON_TOGGLE, ICON_CURTAINS, curtains_state)
    json<<"]]}"
} else {
    // NSPanel has asked you to just update the states of your buttons on this panel
    json = makeSyncButtonStart(id, 1, cieling_state)
    json<<addSyncButton(2, floor_state)
    json<<addSyncButton(3, mood_state)

```



```

        json<<addSyncButton(6,curtains_state)
        json<<"]]}"
    }
    logger.info("Panel: "+PANEL_LOUNGE+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_CABIN :
    logger.info("NXPanel Trigger - PANEL_CABIN")
    // set these from your own items
    front_state = 1
    if (refresh) {
        // NSPanel has asked you to render the entire panel
        // Define the layout of your panel
        json = makePage(id,'Hall Lightning',format)
        json<<makeButton(1,"Front",BUTTON_TOGGLE,ICON_BULB,front_state)
        json<<makeButton(2,"Alarm",BUTTON_PAGE,ICON_BELL,PAGE_ALARM,NONE)
        json<<makeButton(3,"Status",BUTTON_PAGE,ICON_INFO,PAGE_STATUS,PANEL_STATUS)
        json<<"]]}"
    } else {
        // NSPanel has asked you to just update the states of your buttons on this panel
        json = makeSyncButtonStart(id,1,front_state)
        json<<"]]}"
    }
    logger.info("Panel: "+PANEL_CABIN+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_CABIN_THERMO :
    // set these from your own items
    var heater = 1
    var auto = 0
    var temp = 0
    var set = 14
    json = makePage2(id,'Cabin')
    json<<"therm":{"
    json<<"set":'<<set<<',"temp":'<<temp<<',"heat":'<<heater<<',"state":'<<auto<<'"
    json<<"}}}"
    logger.info("Panel: "+PANEL_CABIN_THERMO+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_CABIN_LIGHTS :
    // Color Dimmer Panel: This panel needs to refresh each time it's called

    // set these from your own items
    bedroom_mood_state = 1
    bedroom_mood_hsbcolor "10,100,50"
    json = makePage2(id,'Bedroom mood light')
    json<<"power":'<<bedroom_mood_state<<',"hsbcolor":'<<"10,100,50"'
    json<<"}}}"
    logger.info("Panel: "+PANEL_CABIN_LIGHTS+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_LOUNGE_FAN :
    // Dimmer Panel: This panel needs to refresh each time it's called

    // set these from your own items
    fan_state = ON
    fan_setting = 3
    json = makePage2(id,'Lounge Fan')

    json<<"power":'<<fan_state<<',"min":'<<1<<',"max":'<<4<<',"icon":'<<ICON_FAN<<',"dimmer":'<<fan_setting
    json<<"}}}"
    logger.info("Panel: "+PANEL_LOUNGE_FAN+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_BEDSIDE_LIGHT :
    // Dimmer Panel: This panel needs to refresh each time it's called
    logger.info("NXPanel Trigger - Processing PANEL_KÖKSBOARD_DIMMER..")
    // set these from your own items
    bedside_state = ir.getItem("P18_Bedside_Light").state==ON?1:0
    bedside_dimlevel = ir.getItem("P18_Bedside_Light_Dimlevel").state

```

```

// Build JSON
json = makePage2(id,'Bedside Light')
json<<"power":'<<bedside_state<<',"dimmer":'<<bedside_dimlevel
json<<""})"
logger.info("Panel: "+PANEL_BEDSIDE_LIGHT+", Sending JSON:"+json.toString())
mqtt.publishMQTT(TOPIC, json.toString())
break
case PANEL_STATUS :
    json = makePage(id,'System Status',format)
    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<<"]}]'
    logger.info("Panel: "+PANEL_STATUS+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
case PANEL_MUSIC :
    json = makePage(id,'Sonos Player',format)
    // set these from your own items
    json<<"artist":'<<"New Order"'<<',"album":'<<"Movement"'<<',"track":'<<"Power
Play"'<<',"volume":'<<70
    json<<""})"
    logger.info("Panel: "+PANEL_MUSIC+", Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break
default :
    logger.info("unknown page!")
    break
}

logger.info("rule done")

```

Button press responses

So far, it's all been around panel design but the point of all this is of course to have things to happen when buttons are pressed, or sliders are moved. This section covers the art of connecting button presses to state/value changes of your OpenHAB items. There are several ways to do this, two of them are:

1. Button presses are picked up by channels in turn operating on items.
2. Groovy script deciphers the JSONs from NXPanel and operates on items

This documentation describes the former method (just a matter of my taste).

Note! Whichever of the two methods used, remember that *every* JSON sent from NXPanel requires a response (refresh or sync).

At this stage it is assumed that steps 1-4 in the Panel Design Work Order are completed. So this section will cover remaining steps:

5. Create the channels for the buttons on your panels.
6. Link your items to the channels.
7. In the Groovy script, replace the simulated item states with the state/value of your actual items.

Also Assumed is that you at this stage know how to add channels to your **NSPanel1 (Generic MQTT Thing)** so I use the short hand version of what you need to configure. The setup is for the first two buttons and shows the generic pattern you need to repeat for all of your buttons.

Create the channels for the buttons

First channel is for the first button on panel 10, definition is:

```
- id: nxpanel_p10b1_hall
  channelTypeUID: mqtt:switch
  label: NXPanel P10B1 Hall
  description: ""
  configuration:
    postCommand: true
    qos: 2
    formatBeforePublish: '{"sync":{"pid":10,"bid":1,"state":%s}}'
    commandTopic: cmdnd/nspanel_7DD7FC/nxpanel
    stateTopic: tele/nspanel_7DD7FC/RESULT
    transformationPattern: 'REGEX:(.*\\"pid\\": 10.*\\"bid\\": 1.*)\nJSONPATH:$.button.state'
    off: "0"
    on: "1"
```

The corresponding screen dump is:

NXPanel P10B1 Hall
NSPanel1 (Generic MQTT Thing)

Channel

NXPanel P10B1 Hall
mqtttopic:mosquitto_sweden3:nspanel1:nxpanel_p10b1_hall

Configuration

Show advanced ☒

MQTT State Topic

tele/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:(.*\\"pid\\": 10.*\\"bid\\": 1.*)\nJSONPATH:\$.button.state

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

{\"sync\":{\"pid\":10,\"bid\":1,\"state\":%s}}

Second channel is for the second button on panel 10, definition is:

```
- id: nxpanel_p10b2_lounge
  channelTypeUID: mqtt:switch
  label: NXPanel P10B2 Lounge
  description: ""
  configuration:
    postCommand: true
    qos: 2
    formatBeforePublish: '{"sync":{"pid":10,"bid":2,"state":%s}}'
    commandTopic: cmdnd/nspanel_7DD7FC/nxpanel
    stateTopic: tele/nspanel_7DD7FC/RESULT
    transformationPattern: 'REGEX:(.*\\"pid\\": 10.*\\"bid\\": 2.*)\nJSONPATH:$.button.state'
    off: "0"
    on: "1"
```

The corresponding screen dump is:

NXPanel P10B2 Lounge

NSPanel1 (Generic MQTT Thing)

Channel

NXPanel P10B2 Lounge

mqtttopicmosquitto_sweden3:nspanel1:nxpanel_p10b2_lounge

Configuration

Show advanced

MQTT State Topic

tele/nspanel7DD7FC/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/nspanel7DD7FC/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:(.*\\"pid\\": 10.*\\"bid\\": 2.*)\nJSONPATH:\$.button.state

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

{ "sync": { "pid": 10, "bid": 2, "state": %s } }

Many channels listening on same topic

As all of the channels you set up will listen to the *same topic simultaneously*, the task for the *transformationPattern* is to uniquely match the incoming JSON so that only the channel intended for a specific button will pick up the state.

The transformationPattern consists of two parts:

- Part 1 is a regular expression (REGEX) that uniquely matches a JSON to one specific channel
- Part 2 takes this matched JSON and using JSONPATH; extracts the value

So, with an example JSON string of: {"button": {"pid": 10, "bid": 2, "state": 0, "next": 0}}

1. The REGEX: (.*)"pid": 10.*"bid": 2.*) will match this string
2. And the sign \cap
3. Will send the matched string to the JSONPATH which extract the state: 0

After picking up the state, NXPanel will of course wait for a sync. As the state of the Item that is linked to this channel changes, it triggers the *resulting state* to be sent back formatted according to *formatBeforePublish*: {"sync":{"pid":10,"bid":2,"state":%s}}. The loop is now closed. NXPanel sent a button changed JSON and OpenHAB sent a sync back.

Link your items to the channels

Add a secondary channel to your items and add the [profile="follow"] at the end, see example below.

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

```
Switch P10B1_Hall "Hall light" {channel="<channel to the Item you want to control>",
channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_p10b1_hall" [profile="follow"]}
Switch P10B2_Lounge "Lounge light" {channel="<channel to the Item you want to control>",
channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_p10b2_lounge" [profile="follow"]}
```

Update your Groovy Script

Finally, update the Groovy script, see example for Panel 10 below.

- Fetch the Item state (marked in blue)
- For a Panel sync: Add the Item state in the return JSON (marked in green)

```
case PANEL_MAIN :
    logger.info("main panel")
    // set these from your own items
    //movie_state = 1
    Hall_state = ir.getItem("P10B1_Hall").state==ON?1:0
    //lounge_state = 1
    lounge_state = ir.getItem("P10B2_Lounge").state==ON?1:0

    if (refresh) {
        // NSPanel has asked you to render the entire panel
        // Define the layout of your panel
        json = makePage(id, 'Home - Main Panel', format)
        json<<makeButton(1, "Hall", BUTTON_TOGGLE, ICON_BULB, hall_state)
        json<<makeButton(2, "Lounge", BUTTON_TOGGLE, ICON_BULB, lounge_state)
        json<<makeButton(3, "Bed 1", BUTTON_PUSH, ICON_BULB)
        json<<makeButton(4, "Heating", BUTTON_PAGE, ICON_HEAT, PAGE_THERMOSTAT, PANEL_CABIN_THERMO)
        json<<makeButton(5, "Lounge", BUTTON_PAGE, ICON_SOFA, PAGE_6_BUTTON, PANEL_LOUNGE)
```

```

        json<<makeButton(6,"Study",BUTTON_PAGE,ICON_SOFA,PAGE_4_BUTTON,PANEL_BEDROOM_1)
        json<<makeButton(7,"Cabin",BUTTON_PAGE,ICON_HOUSE,PAGE_3_BUTTON,PANEL_CABIN)
        json<<makeButton(8,"Garage",BUTTON_PAGE,ICON_HOUSE,PAGE_2_BUTTON,PANEL_BEDROOM_2)
        json<<""]}"
    } else {
        // NSPanel has asked you to just update the states of your buttons on this panel
        json = makeSyncButtonStart(id,1,hall_state)
        json<<addSyncButton(2,lounge_state)
        json<<""]}"
    }
    logger.info("Panel: 10, Sending JSON:"+json.toString())
    mqtt.publishMQTT(TOPIC, json.toString())
    break

// set these from your own items
//movie_state = 1
Hall_state = ir.getItem("P10B1_Hall").state==ON?1:0
//lounge_state = 1
lounge_state = ir.getItem("P10B2_Lounge").state==ON?1:0

```

Expected output

In the log you would see something like below when you toggle button 1 on page 10.

Note that both the Groovy script is triggered (but does nothing) and that the Channel for Panel10/Button1 is triggered and the Item “P10B1_Hall” follows in the state changes.

```

2022-04-18 11:06:31.954 [INFO ] [openhhab.event.ChannelTriggeredEvent ] -
mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_page_trigger triggered {"button": {"pid": 10, "bid": 1,
"state": 1, "next": 0}}
==> /var/log/openhab/openhab.log <==
2022-04-18 11:06:31.959 [INFO ] [org.openhab.core.automation.nspanel ] - Demo page rules called
==> /var/log/openhab/events.log <==
2022-04-18 11:06:31.976 [INFO ] [openhhab.event.ItemCommandEvent ] - Item 'P10B1_Hall' received
command ON
2022-04-18 11:06:32.082 [INFO ] [openhhab.event.ItemStateChangedEvent ] - Item 'P10B1_Hall' changed from
OFF to ON
2022-04-18 11:06:36.211 [INFO ] [openhhab.event.ChannelTriggeredEvent ] -
mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_page_trigger triggered {"button": {"pid": 10, "bid": 1,
"state": 0, "next": 0}}
==> /var/log/openhab/openhab.log <==
2022-04-18 11:06:36.214 [INFO ] [org.openhab.core.automation.nspanel ] - Demo page rules called
==> /var/log/openhab/events.log <==
2022-04-18 11:06:36.218 [INFO ] [openhhab.event.ItemCommandEvent ] - Item 'P10B1_Hall' received
command OFF
2022-04-18 11:06:36.286 [INFO ] [openhhab.event.ItemStateChangedEvent ] - Item 'P10B1_Hall' changed from
ON to OFF

```

What can happen is that neither your Groovy script or none of your channels pick up the JSON sent by NXPanel at a button press. And NXPanel will wait forever for the expected sync – nothing happens and you get a small red light at the bottom of the panel.

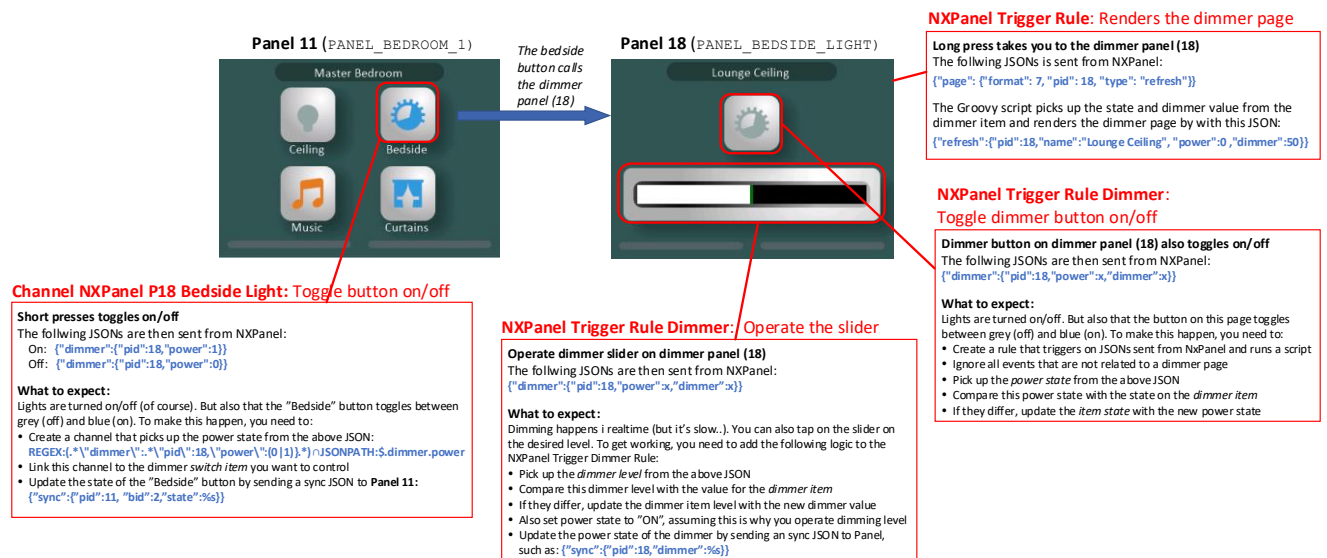
- Possible problem 1: The REGEX does not match the JSON
- Possible problem 2: The channel is not successfully linked to an item

Tip to help you to sort out the first problem;

- Checkout your REGEX with the JSON here: <https://regex101.com>
- Checkout your JSONPATH with the JSON here: <http://jsonpath.com>

Dimmer press responses

The dimmer panel type is special and has a couple of other use cases, e.g., controlling curtains, fans, basically anything that has needs a percentage in addition to the on/off state. It is also special as the on/off state can be controlled from the panel *hosting* the dimmer button, but also from the dimmer panel itself which in turn features both on/off stat and has a slider. This makes the implementation a bit more complicated, see pic below for an overview.



Steps are:

1. Create one channel (to be linked to the **Dimmer Switch Item** below)
2. Create *two* dimmer items for your Dimmer
 - a. One **Dimmer Switch Item**
 - b. One **Dimmer Item**
3. Link the channel to the Dimmer Switch Item
4. Create a **NXPanel Trigger Rule Dimmer** with Groovy script

Create a channel for the dimmer button

The purpose of this channel is to control your dimmer when you toggle on/off on the panel *hosting* the dimmer button (Panel 11 in this example) - You do this by a short press on the Dimmer button.

And here comes a trick, your response (the sync to NXPanel) should *not* be sent to the *same* panel as the incoming JSON:

1. JSON sent from NXPanel: {"dimmer":{"pid":18,"power":0}}
2. JSON response from channel: {"sync":{"pid":10,"bid":2,"power":1|0}}

Note! All incoming JSONs will contain the "pid 18" so no question about which dimmer panel is in question. But depending on which panels is toggling on/off, the response should in this example be sent to either Panel 11 (the panel hosting the button) or Panel 18 (the dimmer panel itself).

- JSON sent from the hosting panel (11) contain *only* the **on/off state**
- JSON sent from the dimmer panel (18) contain *both* the **on/off state** and the **dimmer value**

The channel definition is:

```
- id: nxpanel_p18_bedside_light
  channelTypeUID: mqtt:switch
  label: NXPanel P18 Bedside Light
  description: ""
  configuration:
    postCommand: true
    formatBeforePublish: '{"sync":{"pid":10,"power":%s}}'
    qos: 2
    commandTopic: cmdnd/nspanel_7DD7FC/nxpanel
    stateTopic: tele/nspanel_7DD7FC/RESULT
    transformationPattern:
      REGEX: (.*)"dimmer\":.*"pid\:18,\"power\":(0|1)}.*)\nJSONPATH:$.dimmer.power
    off: "0"
    on: "1"
```

Create two dimmer items

Normally you might only use one Dimmer item for your dimmers, but when you want to control the dimmer from NXPanel, it's useful to also have a separate **Dimmer Switch Item**, see item definition example below.

```
Switch P18_Bedside_Light "P18 Bedside Light Switch" {
channel="<channel to the Dimmer Item you want to control>"
Dimmer P18_Bedside_Light_Dimlevel "P18 Bedside Light Dimlevel" {
channel="<channel to the Dimmer Item you want to control>"
```

Link your items to the channels

As for any buttons, add a secondary channel to your **Dimmer Switch Items** and add the [profile="follow"] at the end, see example below.

```
Switch P18_Bedside_Light "P18 Bedside Light Switch" {
channel="<channel to the Dimmer Item you want to control>",
channel="mqtt:topic:mosquitto_sweden3:nspanel1:nxpanel_p18_bedside_light" [profile="follow"]}
```

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

Create NXPanel Trigger Rule Dimmer

This rule is basically a copy of the “NXPanel Trigger Rule” and listens on the same channel, only change is the Groovy script executed when the channel is triggered.

- Create the rule:
 - 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 the Groovy script in the next section below.
 - **Important!** After adding the script code, edit **line 11 and 12** to match your TOPIC and Mosquitto broker channel ID, the script will not work without this change.

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

Groovy Script for Dimmer Panels

The JSONs coming from dimmer panels structurally differ quite much from JSONs coming from button panels. I therefore choose to put this logic in a separate Groovy script and intention is that this script can process any Dimmer Panels you are using. Example of a JSONs sent when you toggle on/off or operate the slider on the *dimmer panel* are:

1. JSON sent from NXPanel: `{"dimmer":{"pid":18,"power":0,"dimmer":56}}`

The Groovy script looks like:

```
/*Imports and global definitions */
import org.slf4j.LoggerFactory
def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

/*-----
 * Custom Configurations
 * Note1! Replace MQTT channel with that of your mosquitto broker
 * Note2! Set TOPIC to the value of your NSPanel
 * (This is where the JSON string built by this script is posted)
 *-----*/
def mqtt = actions.get("mqtt", "mqtt:broker:mosquitto_sweden3")
def TOPIC = "cmd/nspanel_7DD7FC/nxpanel"

// Get the JSON string when event has triggered
def str = event.getEvent()

logger.info("NXPanel Trigger Dimmer - event: <"+str+">")

// Return if the string "dimmer": is not found twice in the JSON
// This basically means that all other events posted on this channel are dropped
if (str.count("dimmer")<2) {
    logger.info("NXPanel Trigger Dimmer - Not a dimmer panel event, returning...")
}
```

```

    return
}

// Get data from the dimmer message, str contains the JSON string.
// Extract the panel id, power state and dimmer level from str
var pid = (str=~/"pid":(\d+)/)[0][1]
var power = (str=~/"power":(\d+)/)[0][1]
var dimmer = (str=~/"dimmer":(\d+)/)[0][1]

logger.info("NXPanel Trigger Dimmer - id=<"+pid+">, power=<"+power+">,
dimmer=<"+dimmer+">")

// Empty variable to contain the return JSON
var json = ""

// Define the panels and their corresponding panel id as in the trigger rule
// (Here you map all the panels you have designed with dimmer type)
def PANEL_BEDSIDE_LIGHT = 18

// Just send a message to the log that your received a post from NSPanel
logger.info("Processing dimming for page ... "+pid)

// Check wich of your panels NSPanel wants you to process
switch (pid as int) {
    case PANEL_BEDSIDE_LIGHT :
        // Dimmer event. Need to figure out what changed
        logger.info("NXPanel Trigger Dimmer - Processing PANEL_BEDSIDE_LIGHT..")
        // set these from your own items
        bedside_state = ir.getItem("P18_Bedside_Light").state==ON?1:0.toString()
        bedside_dimlevel = ir.getItem("P18_Bedside_Light_Dimlevel").state.toString()

        // Check if power state has changed
        if (power!=bedside_state) {
            if (power=="1") {
                events.sendCommand("P18_Bedside_Light","ON")
                logger.info("NXPanel Trigger Dimmer - Item P18_Bedside_Light: ON")
            } else {
                events.sendCommand("P18_Bedside_Light","OFF")
                logger.info("NXPanel Trigger Dimmer - Item P18_Bedside_Light: OFF")
            }
        }
        // Check if dimmer level has changed
        if (dimmer!=bedside_dimlevel) {
            // Assume you want to turn on light when changing dimming level
            events.sendCommand("P18_Bedside_Light","ON")
            events.sendCommand("bedside_dimlevel",dimmer)
            // As power was turned on, update the button state on NXPanel
            json<<'{"sync":{"pid":'+pid+',"power":1}}'
            mqtt.publishMQTT(TOPIC, json.toString())
            logger.info("NXPanel Trigger Dimmer - Item bedside_dimlevel: "+dimmer)
        }

        break
    default :
        logger.info("NXPanel Trigger Dimmer - Unknown dimmer page!")
        break
}

logger.info("NXPanel Trigger Dimmer - Script ended")

```

NXPanel Init

The setup below shows how to populate the first screen when NXPanel starts – to avoid waiting for an item updates to trigger the update. The setup below also sets the timezone and configures daylight saving. Steps are to first add a trigger channel and then define the rule to be triggered.

- To create the channel, on the Things Menu, Select: **NSPanel1 (Generic MQTT Thing)**
 - Select: **Channels** (top middle)
 - Select: **Add Channel**
 - As Channel Identifier; Enter: **nxpanel_init**
 - As Label; Enter: **NXPanel Init**
 - Select: **Trigger**
 - As MQTT Trigger Topic; Enter: **tele/nspanel_7DD7FC/LWT**
 - Top right, select: **Done**
- Only the rule left to configure
 - Select: **Settings**
 - Select: **Rules** and press "+"
 - As Name Enter: **NXPanel Init Rule**
 - Select: **Add Trigger**
 - Select: **Thing Event**
 - Select: **NSPanel1 (Generic MQTT Thing)**
 - Select: **A trigger channel fired**
 - Select: **nxpanel_init**
 - Select: **Done** (top right)
 - Select: **Add Action**
 - Select: **Run Script**
 - Select: **Groovy**
 - Select: **Groovy**
 - Select: **Edit**

- Cut and Paste this:

```

/*Imports and global definitions */
import org.slf4j.LoggerFactory
def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

/*-----
 * Custom Configurations
 * Note1! Replace MQTT channel with that of your mosquitto broker
 * Note2! Set TOPIC to the value of your NSPanel
 * (This is where the JSON string built by this script is posted)
 *-----*/
def mqtt = actions.get("mqtt","mqtt:broker:mosquitto-sweden2")
def TOPIC = "cmnd/nspanel_7DD7FC/nxpanel"

logger.info("nxpanel starting, updating first panel...")

// Get weather info from Items
def weather = ir.getItem("Forecast_Weather_Icon").state.toString()
def forecastTemp = ir.getItem("Forecast_Temperature").state.intValue()
//def forecastFeelsTemp = ir.getItem("Forecast_Feels_Temp").state.intValue()
def forecastCondition = ir.getItem("Forecast_Condition").state.toString()

// Build JSON string to display weather info on first panel
def json = String.format(
    "{ \"weather\": { \"temp\": %d, \"icon\": \"%s\", \"summary\": \"%s\" } }",
    forecastTemp, weather, forecastCondition)

// Update weather item with json
events.sendCommand("nxpanel_weather_command",json)
logger.info("nxpanel_weather_command: "+json)

// Get temperature from Items
def Current_Outdoor_Temp = ir.getItem("UteTemp").state.intValue()
def Current_Indoor_Temp = ir.getItem("VrumTemp").state.intValue()

// Build JSON string to set temperature on first panel
json = String.format(
    "{ \"summary\": { \"title\": \"Ute %d°C\", \"text\": \"Inne %d°C\" } }",
    Current_Outdoor_Temp, Current_Indoor_Temp)

// Update temperature item with json
events.sendCommand("nxpanel_temperature_command",json)
logger.info("nxpanel_temperature_command: "+json)

// Set Timezone
json = "{\"Timezone\":\"99\"}"
mqtt.publishMQTT(TOPIC,"{\"Timezone\":\"99\"}")
logger.info("nxpanel_temperature_command: {\"Timezone\":\"99\"}")

// Set summer daylight saving for CET (Central European Time) +120 minutes
mqtt.publishMQTT(TOPIC,"{\"TimeDst\":{\"Hemisphere\":\"0\",\"Week\":\"0\",\"Month\":\"3\",\"Day\
\":1,\"Hour\":\"1\",\"Offset\":\"120\"}}")
logger.info("nxpanel_TimeDst_command:
{\"TimeDst\":{\"Hemisphere\":\"0\",\"Week\":\"0\",\"Month\":\"3\",\"Day\":\"1\",\"Hour\":\"1\",\"Offset
\":\"120\"}}")

// Set winter daylight saving for CET (Central European Time) +60 minutes
mqtt.publishMQTT(TOPIC,"{\"TimeStd\":{\"Hemisphere\":\"0\",\"Week\":\"0\",\"Month\":\"10,\"Da
y\":\"1\",\"Hour\":\"2\",\"Offset\":\"60\"}}")
logger.info("nxpanel_TimeStd_command:
{\"TimeStd\":{\"Hemisphere\":\"0\",\"Week\":\"0\",\"Month\":\"10\",\"Day\":\"1\",\"Hour\":\"2\",\"Offse
t\":\"60\"}}")

```

- **Important #1!** Remember to update the Mosquitto channel and topic to match your values
- Select: **Save (Ctrl-S)** (top right corner).

- After creation, the rule should look something like this:

NXPanel Init Rule

Status: IDLE ⏸ ▶

Unique IDa97c8d7952

NameNXPanel Init Rule✕

Description✕

When

⋮ Reorder

✖ When channel mqtt:topic:mosquitto-sweden2:9d462764a0:nxpanel_init was triggered

React on events from a trigger channel of a thing.

➕ Add Trigger

Then

✖ execute a given script

Allows the execution of a user-defined script.

➕ Add Action

But only if

➕ Add Condition

Tags

Semantic ClassNone >

NXPanel Relays 1 and 2

This section describes some side effects that you need to be aware of for two different use cases for the two physical buttons on the NXPanel.

Problem 1

This is not a BIG problem but pretty annoying when it happens, in my case - NXPanel setup and working, button 1 connected to “all lights” and button 2 to “TV & Amplifier” and while playing with the config, I needed to reboot NXPanel which resulted in all lights + TV & Amp being turned off (=family not happy..).

The problem occurs when you have a config where you don’t want use the relays but instead just let the physical button control your item synchronized with the on/off light above the button.

The channel is then configured as below (simplified):

- MQTT State Topic: `stat/nspanel_7DD7FC/RESULT`
- MQTT Command Topic; Enter: `cmnd/nspanel_7DD7FC/nxpanel`
- Incoming Value Transformations: `REGEX:(.*POWER2.*)∩JSONPATH:$.POWER2`
- Custom On/Open Value; Enter: `1`
- Custom Off/Closed Value; Enter: `0`
- Outgoing Value Format: `{ "switches": { "switch1": %s } }`

Scenario: let’s assume both relays are off and you turn ON an item with button 2

As your item turns ON, you send the JSON to `{"switches":{"switch1":0,"switch2":1}}` to turn on the light above button 2. However, the *relay 2 state* is still OFF.

All looks good and you keep the state if your item in sync with the light, not bothering about the relay state.

The problem occurs if NXPanel restarts or reboots. NXPanel automatically saves the state of it’s *relays* in persistent memory so after reboot, NXPanel will restore the state to its previous value and automatically post the restored state of the relay `{"POWER2":"OFF"}` on the state topic to inform any subscribers.

The *relay state* (OFF) now differs from the *state of your item* (ON). As the channel *commands* the item, an OFF is now sent to the item – which is probably not what you want.

Problem 2

This is also not a big problem and only applicable when you control items (such as an TV) that do not have an explicit ON of OFF signal (same infrared command just toggles between on/off).

In this case the channel is configured to use the relays and to control the state of an item:

- MQTT State Topic: `stat/nspanel_7DD7FC/RESULT`
- MQTT Command Topic: `cmnd/nspanel_7DD7FC/Power2`
- Incoming Value Transformations: `REGEX:(.*POWER2.*)∩JSONPATH:$.POWER2`
- Outgoing Value Format: `%s`

Scenario: Let's assume both relays are off. You can now turn on your item with either the physical button or with the OpenHAB user interface.

When you push the physical button 2 (right) on NXPanel, everything works as it should. The channel picks up the ON and commands the item to turn ON. The problem arises when you control the item with the OpenHAB interface. As you switch your item ON, your channel also sends a simple ON (note: without { }) to the topic **cmnd/nspanel_7DD7FC/Power2**. The relay will toggle to ON and "POWER2" will get the value of ON. Tasmota then also automatically send a JSON to itself {"switches":{"switch1":0,"switch2":1}} to turn on the light above button 2. So far, all good, the item is in sync with the light above the physical button.

However... now Tasmota also confirms that the state of the relay has been turned ON by posting {"POWER2":"ON"} on the state topic. The channel picks this up and *commands* the item to turn ON. This is not problem with e.g. lights, if you send ON to something that is already ON, nothing happens. If you send ON to an item connected to a TV (where ON and OFF are the same), it will turn the TV OFF – Annoying.

Solution

To fix this, the solution below instead uses a trigger channel, a Groovy script and a normal DSL rule. The Groovy script compares incoming events with the current state and if they are the same, just returns. Steps are:

- To create the channel, on the Things Menu, Select: **NSPanel1 (Generic MQTT Thing)**
 - Select: **Channels** (top middle)
 - Select: **Add Channel**
 - As Channel Identifier; Enter: **nspanel_switch2_trigger**
 - As Label; Enter: **NXPanel Switch2 Trigger**
 - Select: **Trigger**
 - As MQTT Trigger Topic; Enter: **stat/nspanel_7DD7FC/RESULT**
 - As incoming Value Transformations; Enter:
REGEX:(.*POWER2.*)∩JSONPATH:\$.POWER2
 - Top right, select: **Done**
- Next, configure the rule:
 - Select: **Settings**
 - Select: **Rules** and press "+"
 - As Name Enter: **NXPanel Switch2 Trigger**
 - Select: **Add Trigger**
 - Select: **Thing Event**
 - Select: **NSPanel1 (Generic MQTT Thing)**
 - Select: **A trigger channel fired**
 - Select: **nspanel_switch2_trigger**
 - Select: **Done** (top right)
 - Select: **Add Action**
 - Select: **Run Script**
 - Select: **Groovy**
 - Select: **Groovy**
 - Select: **Edit**

- Cut and Paste this:

```
/*Imports and global definitions */
import org.slf4j.LoggerFactory
def logger = LoggerFactory.getLogger("org.openhab.core.automation.nspanel")

def TvandAmpState = ir.getItem("TvandAmpCtrl").state.toString()

// Get the JSON string when event has triggered
def str = event.getEvent()

logger.info("NXPanel Trigger Switch2 - event: <"+str+">")

//return

if (TvandAmpState == str) {
    logger.info("NXPanel Trigger Switch2 - event: No state change, returning...")
    return
} else {
    logger.info("NXPanel Trigger Switch2 - event: State change to: "+str)
    events.sendCommand("TvandAmpCtrl",str)
}

logger.info("NXPanel Trigger Switch2 - Script ended")
```

- **Note #1!** Remember to replace the item in this example (TvandAmpCtrl) with your item name.
- Select: **Save (Ctrl-S)** (top right corner).
- As the channel is no longer linked with the item, you also need to send item changes to NXPanel by some other means. I implemented this with the below (classic) DSL rule sending MQTT commands directly to NXPanel.

```
// Proxy item to control two devices with one switch (TV and Amplifier)
rule "TvandAmp Control"
when
    Item TvandAmpCtrl received command
then
    logInfo("myLog", "TvandAmpCtrl received command " + receivedCommand)

    val mqttActions = getActions("mqtt", "mqtt:broker:mosquitto-sweden2")
    logInfo("myLog", "TvandAmpCtrl: Power relay 2, sent command: "+TvandAmpCtrl.state)
    mqttActions.publishMQTT("cmdnd/nspanel_7DD7FC/Power2", receivedCommand.toString())

    if (receivedCommand==ON) {
        logInfo("myLog", "TvandAmpCtrl: Turning on TV and Amp")
        TvCtrl.sendCommand(ON)
        AmpCtrl.sendCommand(ON)
    }
    else {
        logInfo("myLog", "TvandAmpCtrl: Turning off TV and Amp")
        TvCtrl.sendCommand(OFF)
        AmpCtrl.sendCommand(OFF)
    }
end
```

- **Note!** As usual, replace the Mosquitto channel and command topic with your value.

That's it for now folks..

9. Appendix

Examples

Command in openhabian:

Set brightness levels

- `mosquitto_pub -u openhabian -P mqtttwd22?? -t cmdnd/nspanel_7DD7FC/nxpanel -m '{"dim":{"low":10, "normal":80}}'`

Set temperatures:

- `mosquitto_pub -u openhabian -P mqtttwd22?? -t cmdnd/nspanel_7DD7FC/nxpanel -m '{"summary":{"title":"Our 32°C", "text":"In 25°C"}}'`

Tasmota Console Commands

You write these comands in the Tasmota Console

UK Timezones:

TimeZone 99

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

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

CET Timezone

TimeZone 99

TimeDST 0,0,3,1,1,120

TimeSTD 0,0,10,1,2,60

weblog 2 - Normal logging
weblog 4 - Extended logging
restart 1 - Restarts the panel

You can find more of the tasmota commands here: <https://tasmota.github.io/docs/Commands>

Mikes Groovy script (original)

```
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")

```