DIY: timerPDM per ozono

Many ozone generators are without PDM (Pulse Duration Modulation) and therefore cannot regulate the ozone produced in the unit of time. Some inexpensive generators are even without timer. But both PDM and the timer are practically indispensable for the correct use of ozone in domestic sanitation. For further information see my introduction to ozone: https://github.com/msillano/Ozone-coronavirus-sonoff/blob/master/Ozono_Coronavirus_Sonoff_it.pdf and the ozone simulator:
https://github.com/msillano/Ozone-coronavirus-sonoff/tree/master/PROJECTS-DIY/simulOzone

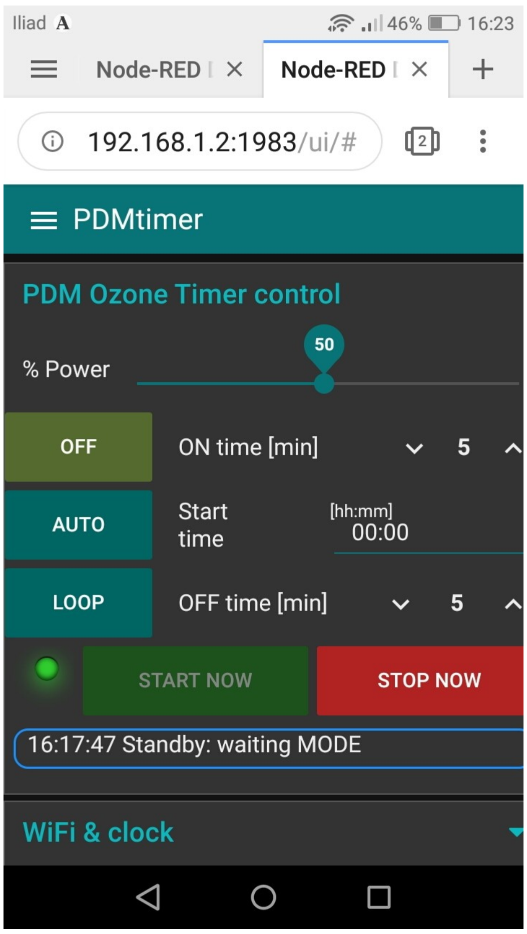
The solution in these cases is this DIY **timerPDM** *project: a PDM timer for ozone generators, designed and manufactured according to the state of the art, better than many timers present in even high-end ozonators.*

Ozone in air purify applications

The **timerPDM** allows a first control of the ozone produced by *PDM* (2% .. 100%) with a one minute cycle. Furthermore, in *LOOP* mode, it allows you to adjust the *ON* (1min..3h) and the *OFF* (1min..24h) times separately in 1 minute intervals.

Ozone in chock applications

In *AUTO* mode, in addition to the PDM control and the setting of the switch-on duration, it is possible to program the start of the ozonator (hh:mm). In addition, the button on the **timerPDM** allows manual start of *AUTO* mode with an initial delay of 1 minute to move away.



PDM Ozone Timer control

% Power 50

OFF | ON time [min] 5

AUTO | Start time [hh:mm] 00:00

LOOP | OFF time [min] 5

START NOW | STOP NOW

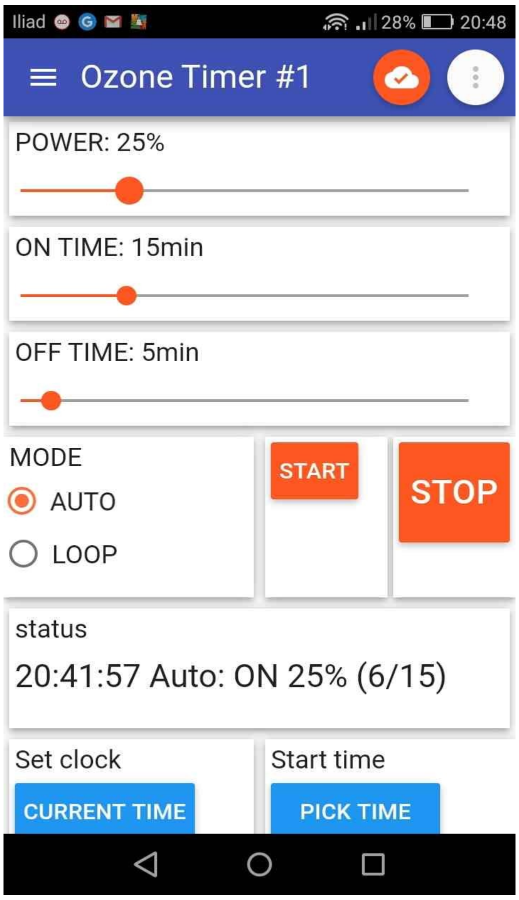
16:17:47 Standby: waiting MODE

WiFi & clock

This **timerPDM** can be controlled directly or remotely. To exchange data and commands with other external applications, it follows the IOT logic and the MQTT protocol.
For connectivity, it contains an STA (WIFI client) to connect to the internet via an existing WiFi network, an AP (WiFi Access Point) to allow smartphones and PCs to connect to the timer even in the absence of a network and router.
For operating logic, it contains an MQTT client and an MQTT server (broker), equipped with their own programming language (script).

Concluding in this project, the operating logic is entirely internal: an external MQTT server (mosquitto type) active 24/7 is not required. This **timerPDM** is autonomous, but at the same time compatible with external standard MQTT clients and brokers: therefore it can be used standalone or remotely controlled or even

integrated into more complex projects IOT, like smart homes.
The monitoring and basic configuration of **timerPDM** can be carried out with MQTT clients with dashboards, both on PC and Android smartphones. Two applications are available:



POWER: 25%

ON TIME: 15min

OFF TIME: 5min

MODE
[✓ AUTO]
LOOP

status
20:41:57 Auto: ON 25% (6/15)

Set clock

Start time

1. **timerPDM-red** for PC, using *node-red*

2. **PDM timer** for Android smartphones, using a standard app (*IoT MQTT Panel)*

*Although using updated and quality components, this **timerPDM** is very cheap: total cost less than € 10 (US $ 13)!*

All this is made possible by the happy union of two phenomenal products:

Sonoff S20 economic WIFI switch with an excellent form factor, power supply, EPS8266 processor and relay

esp_MQTT, a firmware for EPS8266 created by *Martin-ger,* which transforms Sonoff into a tank.

*Sonoff S20 is an extraordinary WiFi switch but has unspecialized performance for controlling ozonators. With esp_MQTT the operating logic of S20 can be reprogrammed, thus obtaining a functionally complete **timerPDM**.*

Performance:

- Power**: PDM** adjustment 2 %..100 %
- Switch-on: **ON time** (minutes)
- Switch-off: **OFF time** (minutes)
- Start time: **Start time** (hh:mm)
- *Operating modes*:

  0. **OFF:** safety standby.

  1. **AUTO:** timed start, only one cycle of ***ON time*** duration, for shock applications.
  turns on: automatically at ***Start time***, or **START** (immediate), **TOGGLE** (delayed) buttons.
  turns off: after ***ON time***. Or **STOP**, **TOGGLE** (immediate) buttons.

  2. **LOOP:** infinite cycle ***ON time*** (on) + ***OFF time*** (off) for air purify applications.
  begins: **START** (immediate), **TOGGLE** (delayed) buttons.
  ends: **STOP**, **TOGGLE** (immediate) buttons.

- The **timerPDM** has two LEDs:
  GREEN flashes in OFF mode, while it is permanently on in AUTO or LOOP mode

RED is on when the load (ozonator) is powered.

- In all ON periods, the load is modulated by the PDM, with a 60-second base. 100% equals always on.
- The maximum values of the ON and OFF times (in minutes) are defined in the UI and not in the timer. They can therefore be easily modified:
  Window: **ON time** max 6H, **OFF time** max 24H, in order to have daily cycles.
  Android: **ON time** max 1H, **OFF time** max 1H, for easier setting.

- In OFF mode, the **STOP** button updates all the values presented on the app screen, for example in case of a new connection.

- Button on Sonoff: **TOGGLE** function (AUTO and LOOP). When acting as *START* the action is delayed by 1 minute to allow the user to leave, while as *STOP* the action is immediate.
- In OFF mode, pressing the Sonoff **TOGGLE** button once switches to AUTO mode and the GREEN LED from flashing becomes steady. Pressing the button a second time is equivalent to a START (delayed 1 minute) which uses the stored parameters. A third press acts as a STOP (the GREEN led flashes).

Features:

✔ *Autonomous:* contains an MQTT server with operating logic (script).
✔ *Compatible:* can be used with standard MQTT clients and MQTT servers.
✔ *Autostart:* preserves status and configuration in flash RAM: in case of reset or blackout it restores the previous status autonomously.
✔ *NTP-client:* with access to the Internet, **timerPDM** uses NTP to get the exact time. The timezone command must be used for summer/winter time. In the absence of an NTP connection, the time can be set from the PC or smartphone clock.
✔ *Serial console:* for installation and debugging, via WiFi (telnet) or serial cable (COM)
✔ *OTA update:* the operating logic (script) can be updated via OTA (Over The Air), without moving or opening the timer.
✔ *MQTT client*: currently 2 applications for monitoring and configuration:
1. "timerPDM-red" for PC, (use node-red)
2. "PDM timer" for Android, (use IoT MQTT Panel).

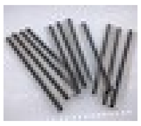
✔ *Power supply*: 110-240 AC
✔ *Stand-by consumption:* not measurable (<1 W)
✔ *Max load*: 240 V 10A / 300W (http://www.fanhar.com/en/products/21.html)

Software

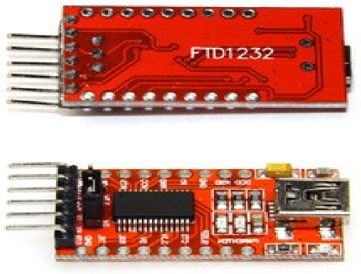
*I thank the authors of the following software used in the realization of the **timerPDM**:*

esp_MQTT	https://github.com/martin-ger/esp_mqtt
esptool	https://github.com/espressif/esptool
puTTY	https://putty.org/
TCP Telnet Terminal:	https://play.google.com/store/apps/details?id=project.telnettcpterminal&hl=en
Notepad++	https://notepad-plus-plus.org/
wpp_pampa	http://www.winpenpack.com/en/index.php
node-red	https://nodered.org/docs/getting-started/installation
IoT MQTT Panel	https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod&hl=en

Materials

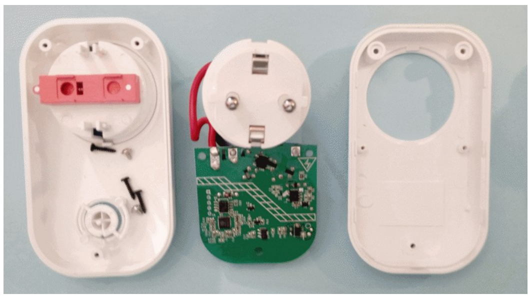
	Itead Sonoff S20 Smart Socket Plug Wifi https://it.aliexpress.com/item/4000382230184.html (Alternative: mod. S26, but has more difficult connections)	$ 8,52
	Cheaper alternative, without red LED, suitable to be mounted on the ozone generator case. Sonoff basic WiFi wireless switch https://www.itead.cc/smart-home/sonoff-wifi-wireless-switch.html	€ 4.85
	4/5 pin for c.s.	$ 0.10
	4 Female To Female Jumper Cable Dupont	$ 0.10
	FE-SP-HDR23-100/22 Optional inductive load filter (surge protector). Required for high loads.	£ 2.48+sp

A) Tools

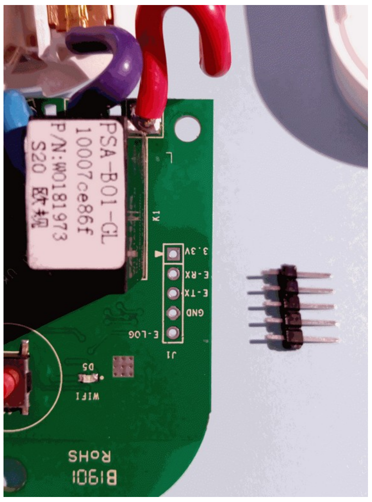
	Thin tip welder (I use PBLK 6 A1 Parkside)
	FTDI FT232RL USB to TTL Serial Converter Adapter Module 5V and 3.3V For Arduino (US$ 2.30)

Construction

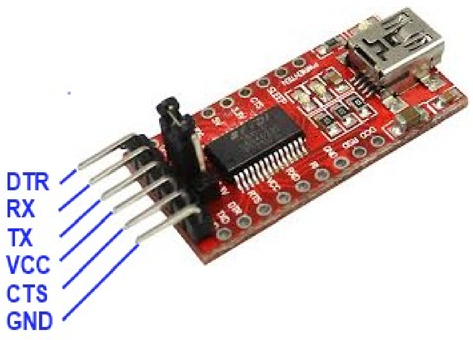
A) preparation of switch S20



1 **Remove the S20 card from the container.**
Three screws join the two shells of the container.
Two more screws block the printed circuit

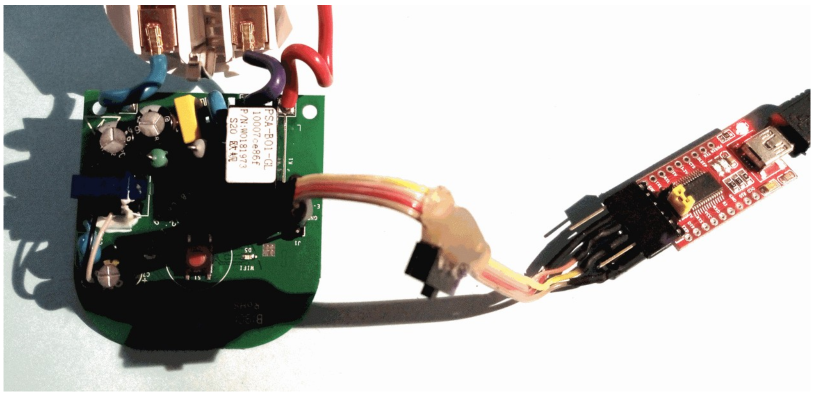


2 Solder a 4/5 pin connector into the Sonoff-S20 pitches (used to load the firmware)



3 Prepare the 4-wire connection FTDI ↔ Sonoff:

Place the jumper on FTDI on 3.3V (not on 5V, destroys S20).



Link:

FTDI	VCC	→	3.3	**S20**
	RX	→	TX	
	TX	→	RX	
	GND	→	GND	

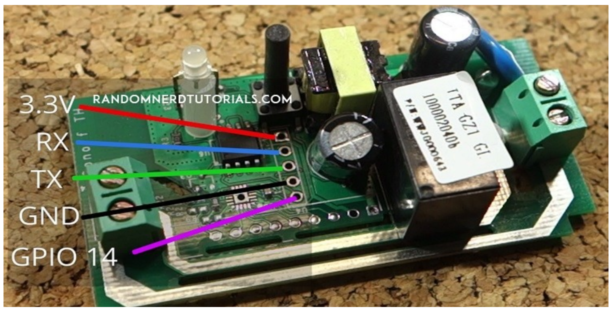
If connectors are used instead of flying cables, a switch on the VCC → 3.3 connection is useful

note: *for more complete instructions on the cable, see:* http://randomnerdtutorials.com/how-to-flash-a-custom-firmware-to-sonoff/

Using S26

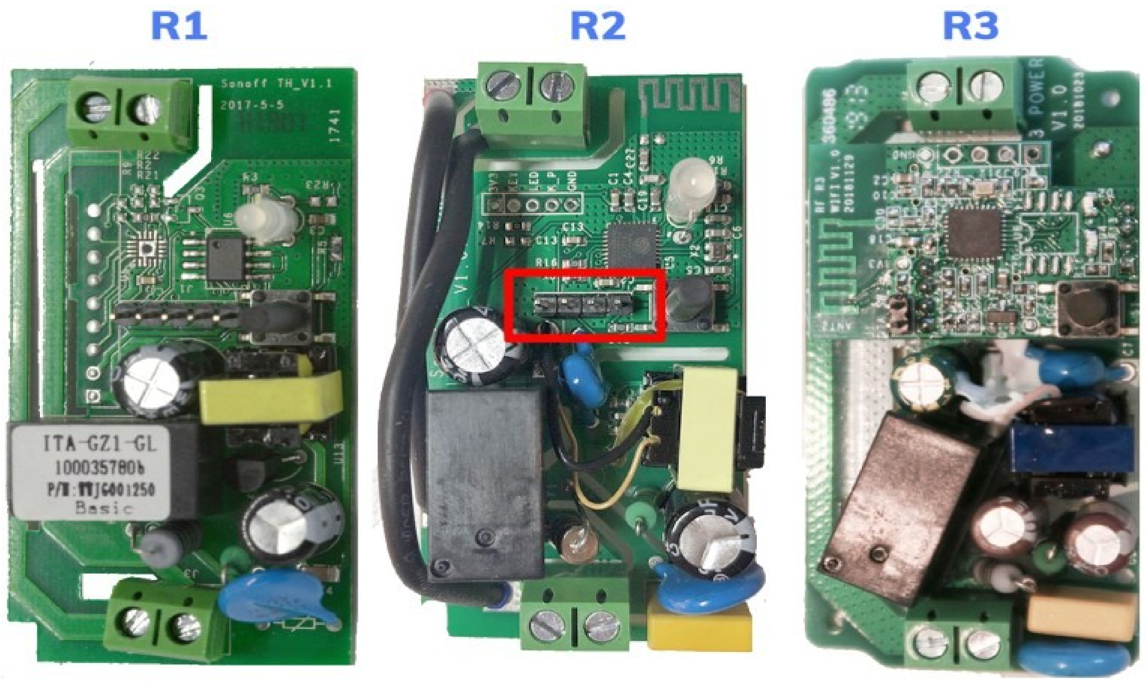
This model is similar to the S20 model, but smaller and does not use a connector. It is therefore necessary to solder 4 wires directly on the printout.. see: https://diyprojects.io/hack-sonoff-s26-wifi-smart-plug-tasmota-firmware-installation/#.XtaeYzozZPY

Using Sonoff-basic



There are 3 versions of Sonoff basic, but all of them use a 4 or 5 pin connector.

The basic as only one GREEN LED.



The Sonoff basic can be mounted on the ozone generator case:

B) firmware esp_MQTT (windows)

esp_MQTT does not have to be compiled: in addition to publishing the sources, *martin_ger* also publishes the 'bin' files ready to be loaded on Sonoff.

- Download and install esptool (https://github.com/espressif/esptool, necessary to copy the firmware. (Also requires python).
- Download from https://github.com/martin-ger/esp_mqtt/tree/master/firmware. the 2 'bin' files and put them in the same dir as esptool.py (in my case in `C:\Program Files (x86)\Python38-32\Scripts\`)

Disconnect Sonoff (S20 / S26 / basic) from the AC power supply!!!

- Connect FTDI via USB to the PC. Check the port number used (Control Panel, System, Device Management, Ports) In my case: COM6.
- Check that FTDI is 3.3 V and then connect Sonoff.
- Keep the button on Sonoff pressed before supplying power to Sonoff (use the switch on the FTDI → Sonoff connection, or, more inconvenient, disconnect and reconnect the cable VCC→ 3.3V). Release the button on Sonoff 1 or 2 seconds after applying voltage.

Use the following command to clean the Sonoff board (COM6 may change). I use a small BAT file (see `esptool/write-erase-flash.bat)`:

```
esptool.py-script.py  --port COM6 erase_flash
```

Then use the following command to copy the firmware to Sonoff-basic (COM6 can change). I use a small BAT file (see `esptool/write-esp-Sonoff.bat)`:

```
esptool.py-script.py --port COM6 write_flash -fs 1MB -fm dout 0x00000 0x00000.bin
0x10000 0x10000.bin
```

note: *The original Itead firmware can be restored with some complications (see* https://wiki.almeroth.com/doku.php?id=projects:sonoff *). But in general it is not necessary: in fact the martin-get "script.sonoff" script perfectly emulates the original functioning of Sonoff Basic (see –* https://github.com/martin-ger/esp_mqtt/tree/master/scripts*).*

note: F*irmware writing does not always succeed on the first try. Try several times.*
The problem is often attributable to the low current supplied by 3.3V FTDI. A simple solution (it worked for me) is to insert an electrolytic capacitor between VCC and GND (I used 100 μF). A more drastic solution is to use a separate 3.3 V power supply (e.g. a step-down 5 → 3.3). Another solution is to power the Sonoff board using 5V, connecting it to ground and to one side of the relay coil, the one with the (+) of the diode (black strip).

DO NOT POWER TO AC Sonoff when FTDI is connected !! *An oversight (a screwdriver falling on the circuit) and your PC (in the best case) is to be thrown away! But you could also risk your life !!*

note*: For more information, for Linux etc. see martin-ger: "Building and Flashing":*
https://github.com/martin-ger/esp_mqtt

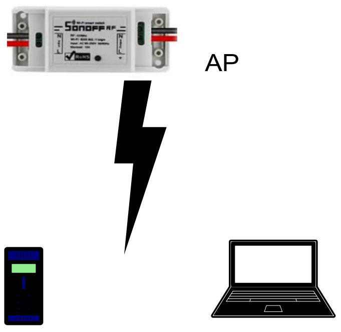
C) Network architectures

Standalone

The timer is used alone, controlling it with its button.

- Only AUTO mode can be used, immediate or timed.
  The first press on Sonoff button activates *AUTO*, the second *START* (after one minute)
- The three values: **ON time**, **PDM** and **Start time** must be previously programmed with one of the apps. They are memorized by the timer, even in the absence of power.

SoloAP



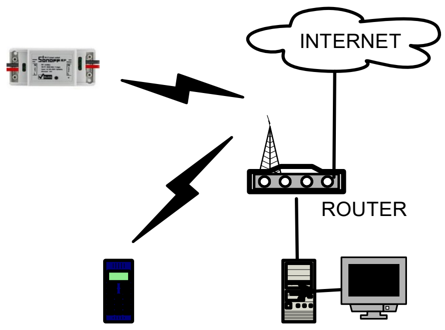
The timer is configured as AP, clients must have WiFi connected directly to the timer AP.

- SSID: "**timerPDM**" (or timerPDM01, timerPDM02 if more than one)
- Password: *none* (open)
- The current time must be set by an app.

Advantages:

- Portable, all apps can be used.
- The MQTT broker has the IP fixed 192.168.4.1, port 1883
- A smartphone without the SIM can be used as a dedicated remote control.

LanRouter



The timer is configured as STA (+ AP) and connects to the router's WiFi.

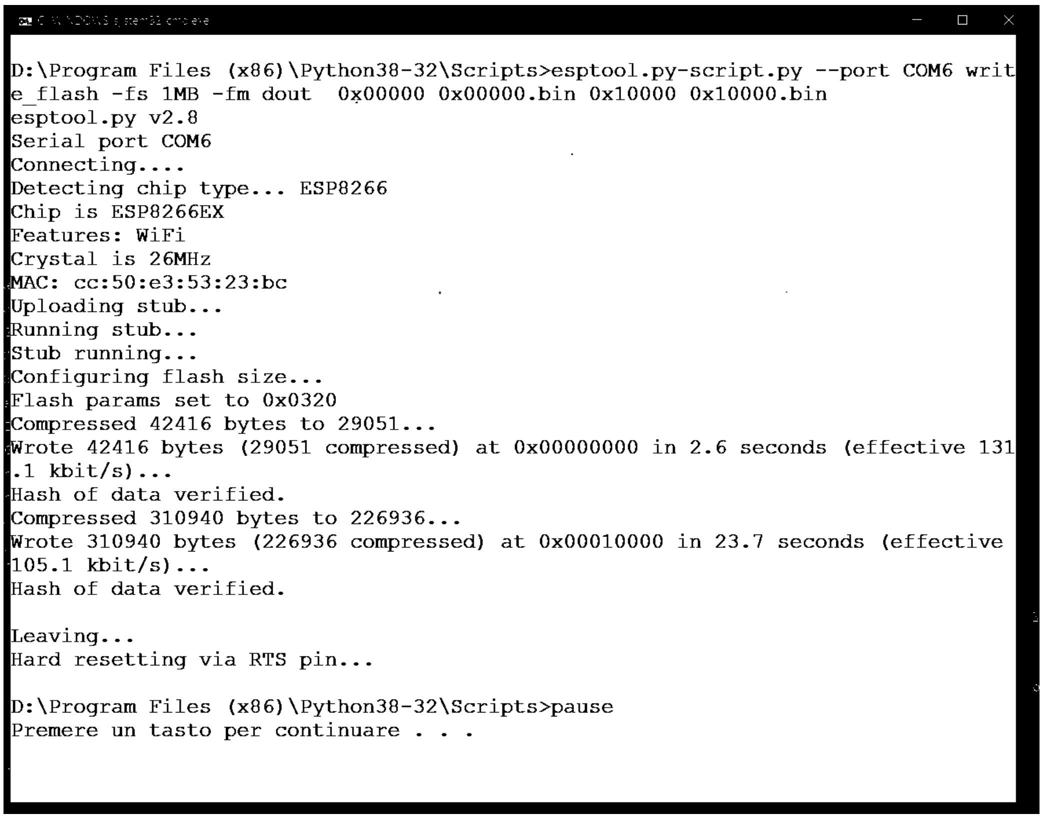
- All devices on the network can connect to the PDM timer
- The I.P. of the MQTT broker is given by the router, port 1883
- The AP function is active: it is possible to connect WiFi directly to the timer (as in the case of **soloAP**)

Advantages:

- Allows the OTA update of the script
- Greater WiFi coverage, also does not need WiFi in devices connected via LAN
- The timer can use an NTP server to get the exact time. Manage daylight saving time (DST) with timezone.
- If *node-red* and the "**timerPDM-red**" flow are active on a PC, smartphones or other devices on the network can use it by connecting to `http://<IP_pc_red>:<port_red>/ui`
- Access is local via WiFi/LAN, but can be made global by the Internet, using a free dynamic DNS service (example noip: https://www.noip.com/).

The initial configuration is **LanRouter**, necessary to use the OTA function and load the script, but through the apps, it is possible to reconfigure the timer (usually not necessary). To disable the **LanRouter** mode: with *ssid* set any name, but DO NOT set *pass:* this blocks the STA function and the empty search for a router that does not exist. To enable **LanRouter** mode or change the router: once **soloAP** is connected, use the *ssid* to set the name of the WiFi router, and use the *pass* to set the password. The connection is automatic.

D) Configuration, first run



```
D:\Program Files (x86)\Python38-32\Scripts>esptool.py-script.py --port COM6 writ
e_flash -fs 1MB -fm dout  0x00000 0x00000.bin 0x10000 0x10000.bin
esptool.py v2.8
Serial port COM6
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: cc:50:e3:53:23:bc
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash params set to 0x0320
Compressed 42416 bytes to 29051...
Wrote 42416 bytes (29051 compressed) at 0x00000000 in 2.6 seconds (effective 131
.1 kbit/s)...
Hash of data verified.
Compressed 310940 bytes to 226936...
Wrote 310940 bytes (226936 compressed) at 0x00010000 in 23.7 seconds (effective
105.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

D:\Program Files (x86)\Python38-32\Scripts>pause
Premere un tasto per continuare . . .
```

When the messages inform us of the correct writing of the flash memory, switch Sonoff off and on again with the switch (*without* pressing the button), leaving FTDI connected.

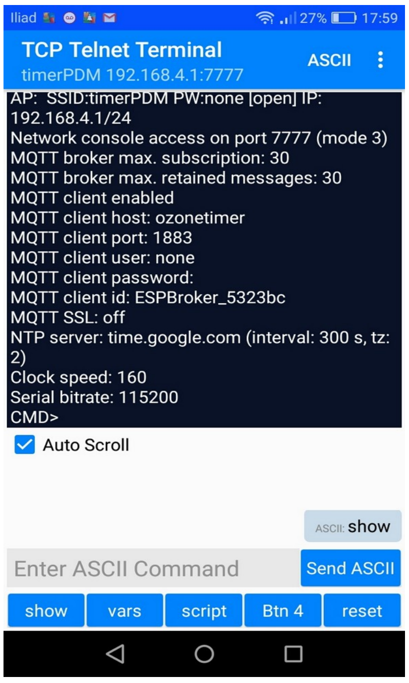
- *The initial basic configuration can be done with the serial console.*

  Use FTDI for the connection and the puTTY Window terminal (https://putty.org/) with the configuration: 'serial', COM6, 115200.
  Pressing [ENTER], the esp_MQTT prompt '**CMD>**' must appear, confirming that it is working correctly. Now we can give all the console commands accepted by esp_MQTT (for usable commands see https://github.com/martin-ger/esp_mqtt)

- *Alternative with Telnet*

  If you want to change the configuration, without using FTDI, for example with the **timerPDM** closed, you can use a laptop with WiFi, and connect the timer: AP "MyAP" (initial default) or "timerPDM" (after configuration).
  Now use *PuTTY*, previously installed on the laptop, to connect (Telnet, IP: 192.168.4.1, port: 7777).

  Or you can also use an Android smartphone and the *TCP Telnet Terminal* app (or other telnet client):
  https://play.google.com/store/apps/details?



```
AP:  SSID:timerPDM PW:none [open] IP:
192.168.4.1/24
Network console access on port 7777 (mode 3)
MQTT broker max. subscription: 30
MQTT broker max. retained messages: 30
MQTT client enabled
MQTT client host: ozonetimer
MQTT client port: 1883
MQTT client user: none
MQTT client password:
MQTT client id: ESPBroker_5323bc
MQTT SSL: off
NTP server: time.google.com (interval: 300 s, tz:
2)
Clock speed: 160
Serial bitrate: 115200
CMD>
```

id=project.telnettcpterminal&hl=en

For our purposes, we want to configure the **timerPDM** as STA and AP:

```
CMD>set system_output 2
CMD>set speed 160
CMD>set ssid <router_name>
CMD>set password <router_pass>
CMD>set ap_ssid timerPDM (or timerPDM01, timerPDM02)
CMD>set mqtt_host ozonetimer

CMD>save
CMD>reset
```

After the reset, **timerPDM** will be reachable as AP with the name timerPDM (for simplicity, without password, but if desired, a password can be configured with:

```
CMD> set ap_password <secret>).
```

We can permanently disconnect FDTI.
The Sonoff card is ready: the script with the operating logic will be loaded via OTA, so we can close the S20 switch with the screws.

note: *use an inductive load filter only for high absorption with low power factor. Connect it on the output terminals, in parallel to the load, or directly on the ozone generator: it increases the useful life of the relay.*
With my two ozone generators (11W, pf: 0.70 and 103W, pf: 0.95) the filter is not necessary.

E) Operating logic (script)

The "`timerPDM02.cde`" file is the script used for the **timerPDM**.

To create the script I used **Notepad ++** (https://notepad-plus-plus.org/), an excellent text editor for programmers. Note: *the script file must be ANSI encoded.*

To stay within the memory limits (4'000 bytes), in the script I tried to minimize the characters used:

1) reduced indentation
2) deleted the comments
3) limited debug 'println'
4) reduced the initial 'config' commands
5) used 2-3 letter names for variables.
6) reused the variable $tmp several times.
7) eliminated checks on received configuration values
8) eliminated all superfluous spaces, quotation marks [“] and brackets [(], [)]
9) used UNIX format for end of line characters

With these warnings, the script is approximately 3990 bytes, but it is not easy to read.

Customizations in the `timerPDM02.cde` script:

2	`%CMD> script http://192.168.178.23:88/www/sonoff/timerPMD02.cde`	OTA command: adapt to the URL used
4	`config ntp_server time.google.com`	used NTP server (internet)
5	`config ntp_timezone 1`	default timezone (can be changed from the app)
6	`config @6 /ozone/timerpdm/1`	change ID (1) in case of multiple **timerPDMs**

note: *In **SoloAP** mode, the **timerPDM** clock must be updated by the apps. In this case, the entered time is used directly as is, and **timezone** is automatically set to 0.*
*In **LanRouter** mode the **timerPDM** tries to use the **NTP** server defined in the script line 4. In this case, **timezone** must be updated, to display the exact local time and take into account the DST.*
note: *If the indicated time is "99:99:99" the **NTP** update has failed. In some routers or firewalls, access to port 123 (default for NTP) must be explicitly enabled. However, it is always possible to set the clock via the apps.*

The easiest way to load a script into ***timerPDM*** in OTA mode is to use a WEB server.

In development, it is convenient to have the local server on the main PC. In my case (Windows) I used the portable server "wpp_pampa" http://www.winpenpack.com/en/index.php contained in a USB stick "winPenPack". Otherwise, you can use any WAMP/LAMP server or even a remote server on the Internet, for example by downloading directly from Github. For Linux see https://github.com/martin-ger/esp_mqtt,

In any case, the **timerPDM** must be in **LanRouter** mode to reach the WEB server.

Using the remote console (telnet: **Putty** or **TCP Telnet Terminal**) give the command '

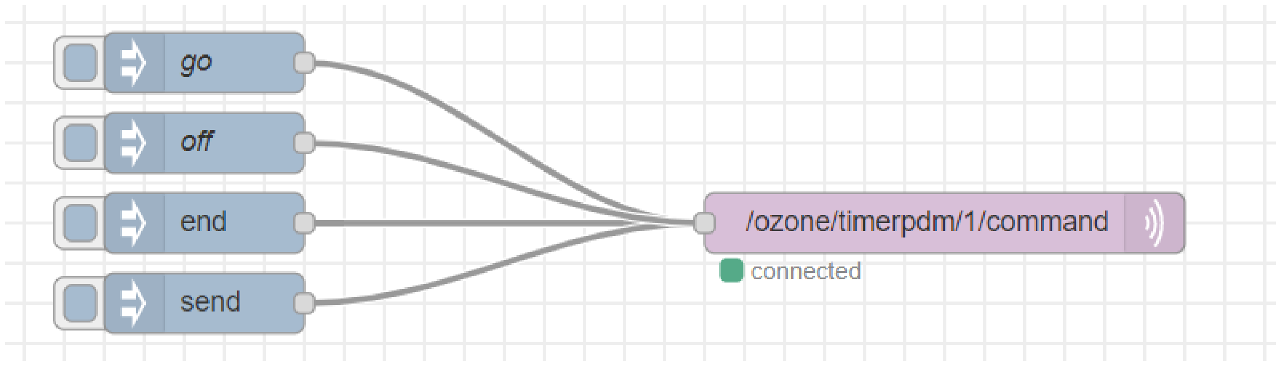
```
CMD> script <url_to_script>
```

For convenience the command to be used on the console is written in the second line of the file: modify it to adapt it to the configuration used, then use copy-paste.

After a **reset** the **timerPDM** is ready

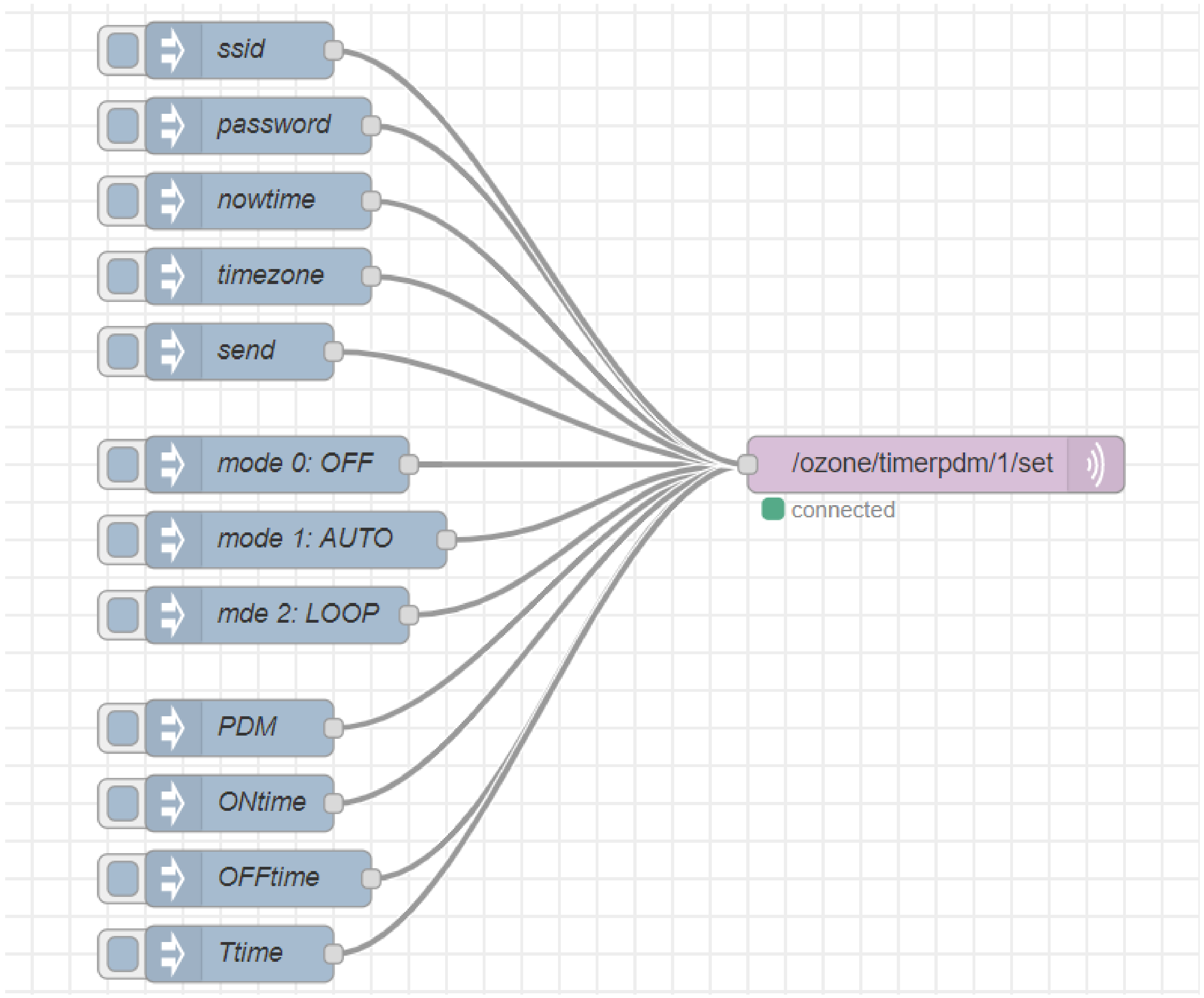
Reference: MQTT input messages

Syntax and semantics of MQTT messages executed by the **timerPDM**.



Subscribe comand: `/ozone/timerpdm/<num>/command`

`go`	: START not conditioned, only in AUTO and LOOP mode
`off`	: STOP not conditioned, keeps the current mode (AUTO or LOOP)
`end`	: STOP not conditioned, returns to OFF mode (standby)
`send`	: (refresh) no action, sends `status/output`



Subscribe Configuration: `/ozone/timerpdm/<num>/set`

`{"data":"ssid", "value":"<wifi-id>"}`	: name of the router to connect to for the internet. Block STA: only AP remains active.
`{"data":"pssw", "value":"<wifi-password>"}`	:password to access the router, activate STA
`{"data":"timezone", "value":<h>}`	:set timezone. <h> is added to the hour provided by NTP
`{"data":"nowtime", "value":"<HH:MM:SS>"}`	:missing NTP, adjust clock (and ptimezone = 0)
`send`	: (refresh) no action, sends `status/config`
`{"data":"mode", "value":0|1|2 }`	:operating mode OFF = 0, AUTO = 1, LOOP = 2
`{"data":"PDM", "value":<2..100> }`	:generator control, in% with respect to the max.
`{"data":"ONtime", "value":<min> }`	:ON duration, in minutes (AUTO and LOOP)
`{"data":"OFFtime", "value":<min> }`	:OFF duration, in minutes (LOOP only)
`{"data":"Ttime", "value":"<HH:MM>" }`	:switch-on time (AUTO only)

note: *The values sent are wired in the properties of the nodes, therefore these flows serve only as an example and test.*

Riferimento: MQTT messaggi output

Syntax and semantics of MQTT sent by the **timerPDM**.

Publish Status-Configuration: `/ozone/timerpdm/<num>/status/config`
(JSON, sent after each change and after `config/send` message):

```
{"mode":<0|1|2>,"Ontime":<min>,"OFFtime":<min>,"PDM":<%>,"Ttime":"HH:MM"}
```

Publish Status-output: `/ozone/timerpdm/<num>/status/output`
(JSON, sent after each command and after `command/send` message)

```
{"Time":"<timestamp>","mode":<0|1|2>,"PDM":<%>,"output":<0|1>,"count":<min>}
```

Publish Status-info: `/ozone/timerpdm/<num>/status/info`
(auto, every second, array: [0] string double escaped for display
[1] led status (0..2))

```
["<timestamp> <mode>: <output> [<PDM>% (<count>/<time>])",
 <led_status>]
```

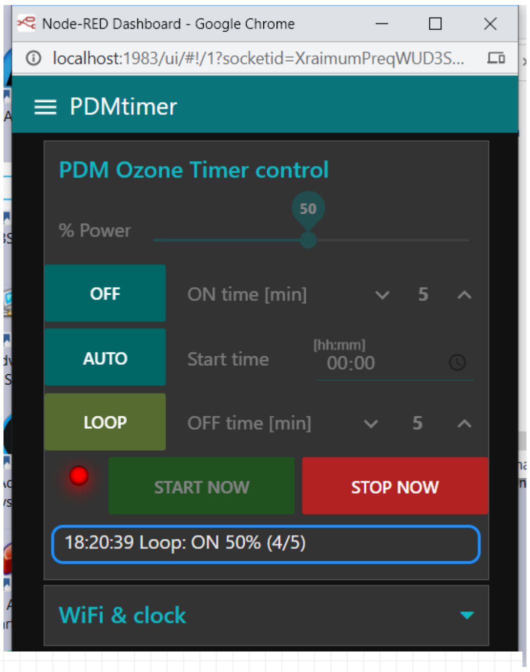
TimerPDM-red: node-red client

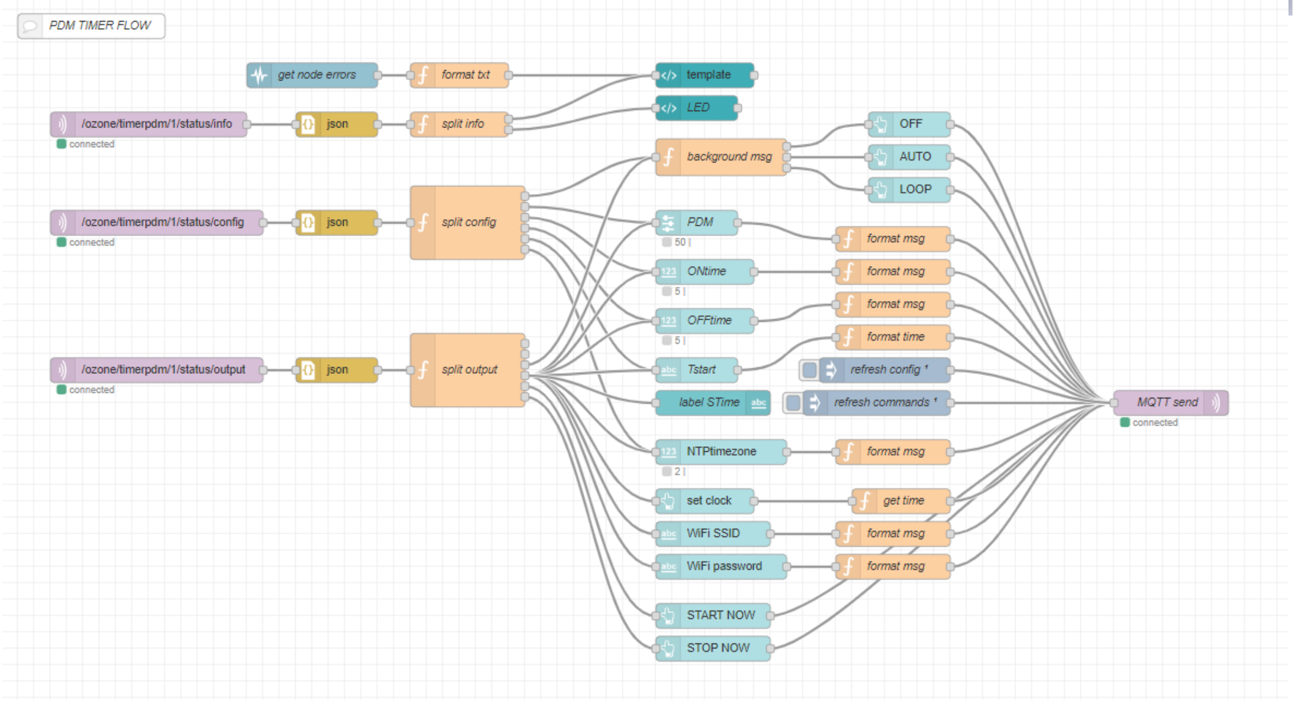
This app is the most complete, suitable for fixed applications in **lanRouter** mode, or for use with portable PCs. Since node-red acts like a WEB server, all devices connected to the network, including smartphones, can use it.

note: *Data storage on DB has not been implemented, but if desired, it is simple to add it to the flow..*

Node-red can be installed on any *PC* (Win / Linux) and also on *Raspberry pi 3* if you want a dedicated low-cost computer (for node-red installation, see https://nodered.org/docs/getting-started/installation)
If not installed by default, import the '`node-red-dashboard`' library.

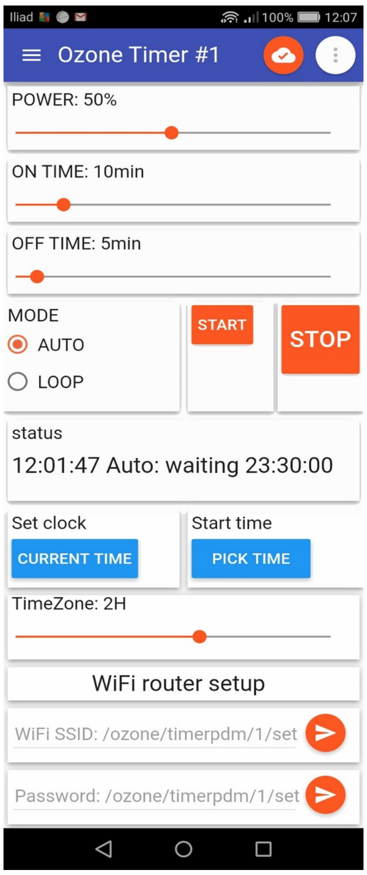
- Copy the contents of the `node-red/timerPDM.json` file to the clipboard and import it into a new flow in *node-red*.
- Configure access to the timer **MQTT broker**: `<IP_timer>:1883`





For convenience in the node-red dir there are also two 'BAT' scripts that I use in a Windows environment to launch *node-red*. These scripts allow you to have numerous flows at the same time, using different ports and paths (port: 1980, 1981 ...)
In my case the dir that contains the timerPDM flows is `D:/node-red/flow-1983/`, obviously you have to modify the BAT and HTML files to adapt them to the paths you use. The main files are:
`start-red-3ui-ozone.bat`, to launch only the user interface (uses `dashboard-1983.html`) and `start-red-debug3.bat`, to launch *node-red*.
Run *node-red* using `start-red-debug3.bat`, and then import the `timerPDM.json` file into a new flow: it will be automatically saved in dir: `D:/node-red/flow-1983/`.

PDM timer: Android client


Iliad 100% 12:07

Ozone Timer #1

POWER: 50%

ON TIME: 10min

OFF TIME: 5min

MODE
- [✓ AUTO]
- LOOP

START | STOP

status
12:01:47 Auto: waiting 23:30:00

Set clock: CURRENT TIME

Start time: PICK TIME

TimeZone: 2H

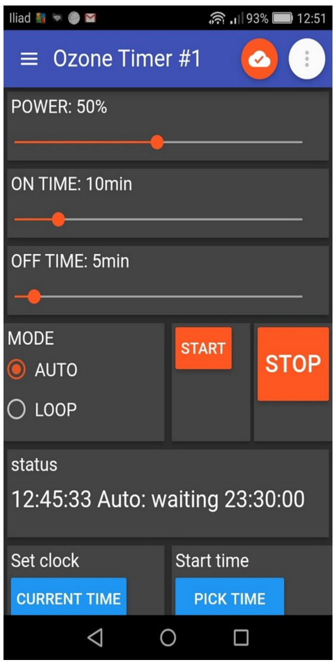
WiFi router setup

WiFi SSID: /ozone/timerpdm/1/set

Password: /ozone/timerpdm/1/set

This simple client allows you to control the timer using a WiFi connected Android smartphone, both in **soloAP** and in **lanRouter** mode. Convenient and above all portable solution, it acts as a remote control, allowing you to activate the ozone generator from a safe area.

For this client, I chose the free *IoT MQTT Dashboard* https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard because it is rich in inputs and aesthetically pleasing, but there are also other alternatives. You can search *Google Play*. The important thing is that the application you choose manages JSON payload.


Iliad 93% 12:51

Ozone Timer #1

POWER: 50%

ON TIME: 10min

OFF TIME: 5min

MODE
- [✓ AUTO]
- LOOP

START | STOP

status
12:45:33 Auto: waiting 23:30:00

Set clock: CURRENT TIME

Start time: PICK TIME

We first create the **soloAP** version.

Unfortunately, it is not possible to import a dashboard into *IoT MQTT Dashboard* from a file: the **PDM timer** dashboard must be created interactively.

Refer to the *IoT MQTT Dashboard* help, and the following figures and comments:

---

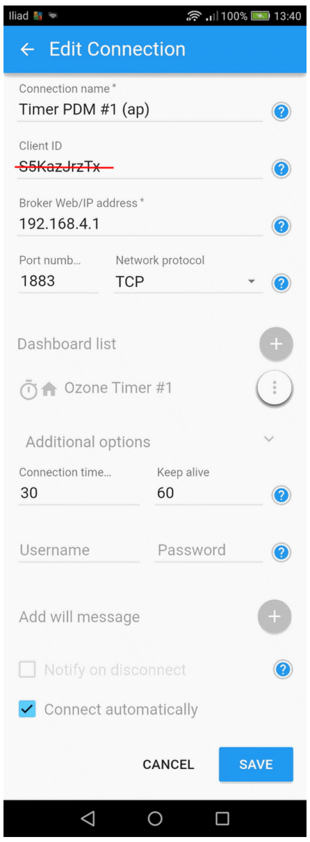
1) *Create a new connection*
Red plus (+) button.

2) *Edit Connection* (see img)
- Leave Client ID blank: it is assigned automatically
- Default IP (**soloAP**): 192.168.4.1, TCP protocol

nota: *in* ***soloAP*** *mode, the WiFi of the smartphone must be connected to the timer AP. SSID =* ***timerPDM***.

3) *Edit Dashboard* (see img)
- Il "Dashboard prefix topic" simplifies the subsequent definition of the panels: `/ozone/timerpdm/1/`
- "lock panels": activate it only when finish.

4) ***Add the following panels (red button):***


Iliad 100% 13:40

Edit Connection

Connection name *: Timer PDM #1 (ap)

Client ID: ~~S5KazJrzTx~~

Broker Web/IP address *: 192.168.4.1

Port numb...: 1883

Network protocol: TCP

Dashboard list: Ozone Timer #1

Additional options

Connection time...: 30

Keep alive: 60

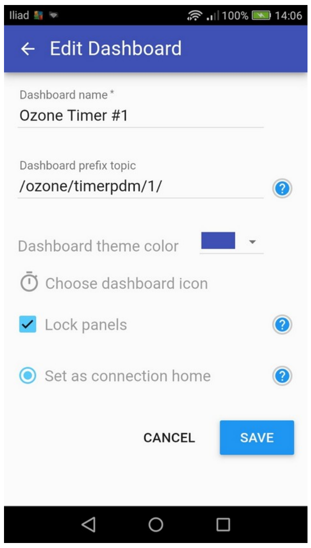
Username

Password

Add will message

- [ ] Notify on disconnect
- [x] Connect automatically

CANCEL | SAVE


Iliad 100% 14:06

Edit Dashboard

Dashboard name *: Ozone Timer #1

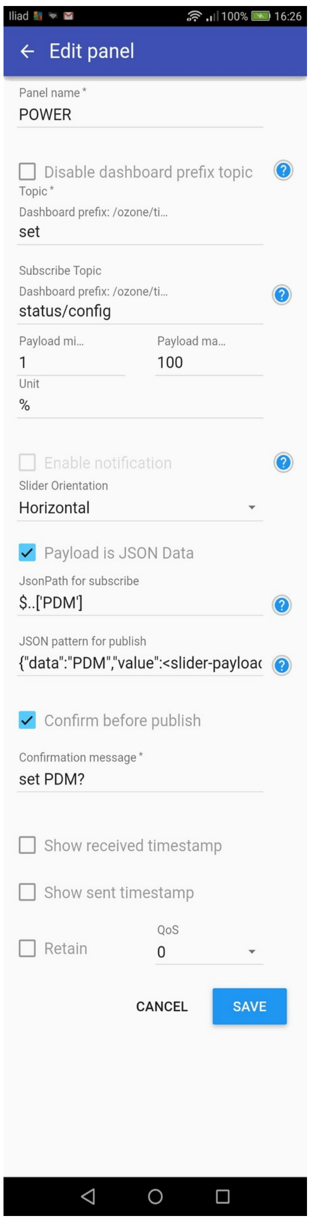
Dashboard prefix topic: /ozone/timerpdm/1/

Dashboard theme color

Choose dashboard icon

- [x] Lock panels
- [✓ Set as connection home]

CANCEL | SAVE


Edit panel

Panel name *
POWER

☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
set

Subscribe Topic
Dashboard prefix: /ozone/ti...
status/config

Payload mi...	Payload ma...
1	100

Unit
%

☐ Enable notification
Slider Orientation
Horizontal

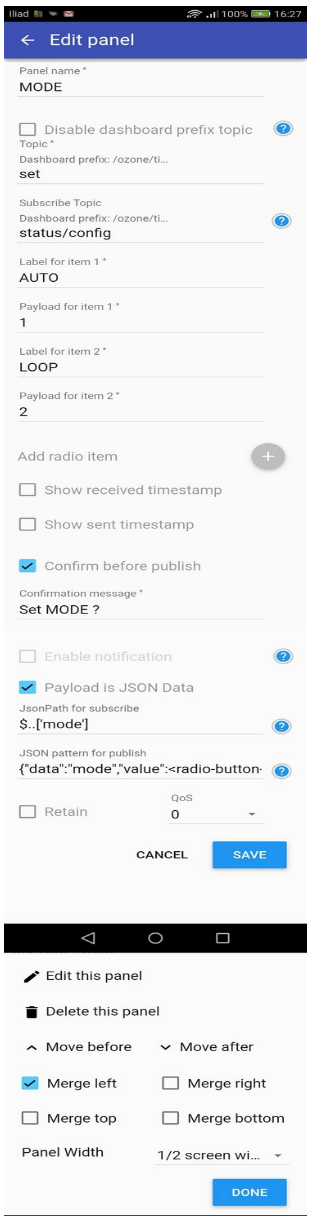
☑ Payload is JSON Data
JsonPath for subscribe
$..['PDM']

JSON pattern for publish
{"data":"PDM","value":<slider-payloac

☑ Confirm before publish
Confirmation message *
set PDM?

☐ Show received timestamp
☐ Show sent timestamp
☐ Retain   QoS 0

CANCEL SAVE


Edit panel

Panel name *
MODE

☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
set

Subscribe Topic
Dashboard prefix: /ozone/ti...
status/config

Label for item 1 *
AUTO
Payload for item 1 *
1
Label for item 2 *
LOOP
Payload for item 2 *
2

Add radio item
☐ Show received timestamp
☐ Show sent timestamp
☑ Confirm before publish
Confirmation message *
Set MODE ?

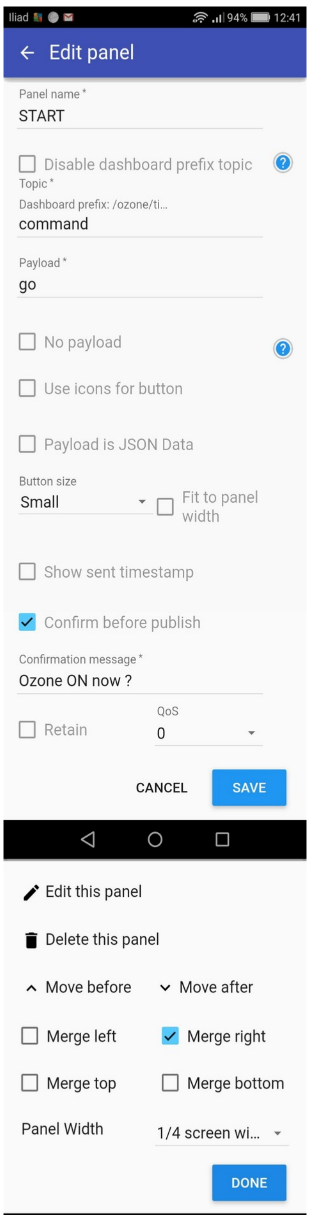
☐ Enable notification
☑ Payload is JSON Data
JsonPath for subscribe
$..['mode']
JSON pattern for publish
{"data":"mode","value":<radio-button-

☐ Retain   QoS 0

CANCEL SAVE

Edit this panel
Delete this panel
Move before   Move after
☑ Merge left   ☐ Merge right
☐ Merge top   ☐ Merge bottom
Panel Width   1/2 screen wi...

DONE


Edit panel

Panel name *
START

☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
command

Payload *
go

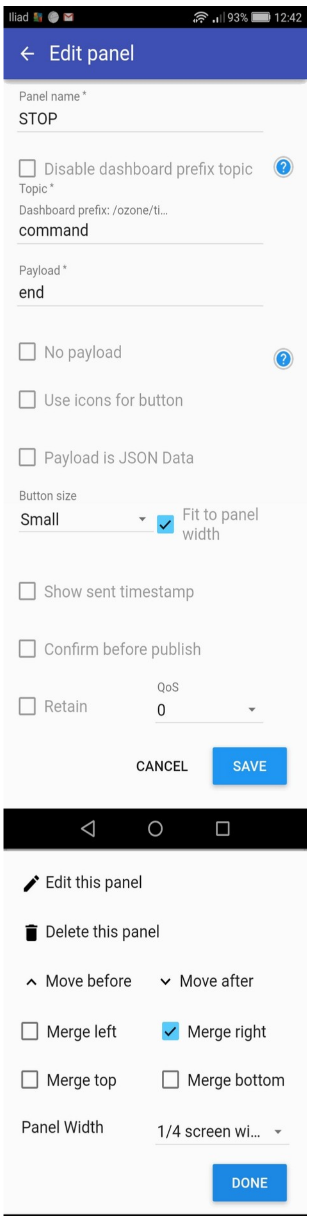
☐ No payload
☐ Use icons for button
☐ Payload is JSON Data
Button size
Small   ☐ Fit to panel width
☐ Show sent timestamp
☑ Confirm before publish
Confirmation message *
Ozone ON now ?

☐ Retain   QoS 0

CANCEL SAVE

Edit this panel
Delete this panel
Move before   Move after
☐ Merge left   ☑ Merge right
☐ Merge top   ☐ Merge bottom
Panel Width   1/4 screen wi...

DONE


Edit panel

Panel name *
STOP

☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
command

Payload *
end

☐ No payload
☐ Use icons for button
☐ Payload is JSON Data
Button size
Small   ☑ Fit to panel width
☐ Show sent timestamp
☐ Confirm before publish
☐ Retain   QoS 0

CANCEL SAVE

Edit this panel
Delete this panel
Move before   Move after
☐ Merge left   ☑ Merge right
☐ Merge top   ☐ Merge bottom
Panel Width   1/4 screen wi...

DONE

4.1 *slider POWER (see img)*

Topic: set
Subscribe topic: status/config
min, max, unit: 1, 100, %
JsonPhat for subscribe: $..['PDM']
JSON pattern for publish: {"data":"PDM","value":<slider-payload>}
confirm: set PDM?

4.2 *slider ON TIME (similar to POWER)*

Topic: set
Subscribe topic: status/config
min, max, unit: 1, 60, min
JsonPhat for subscribe: $..['ONtime']
JSON pattern for publish: {"data":"ONtime","value":<slider-payload>}
confirm: set ON time?

4.3 *slider OFF TIME (similar to POWER)*

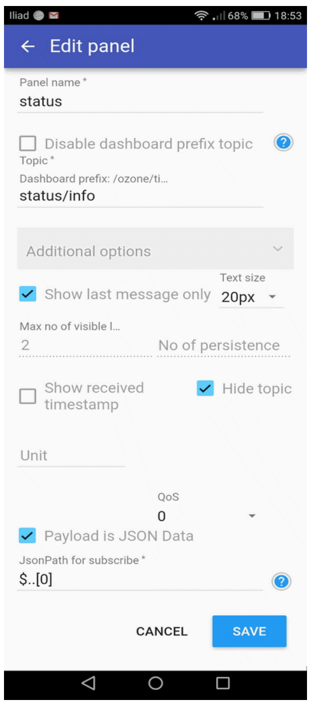
Topic: set
Subscribe topic: status/config

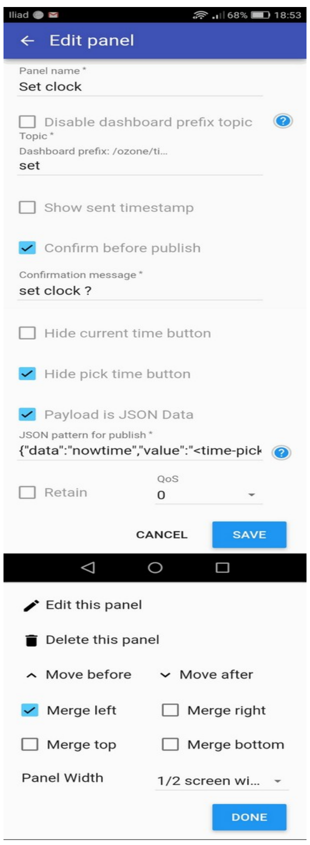
min, max, unit: 1, 60, min
JsonPhat for subscribe: $..[‘OFFtime’]
JSON pattern for publish {“data”:”OFFtime”,”value”:<slider-payload>}
confirm: set OFF time?

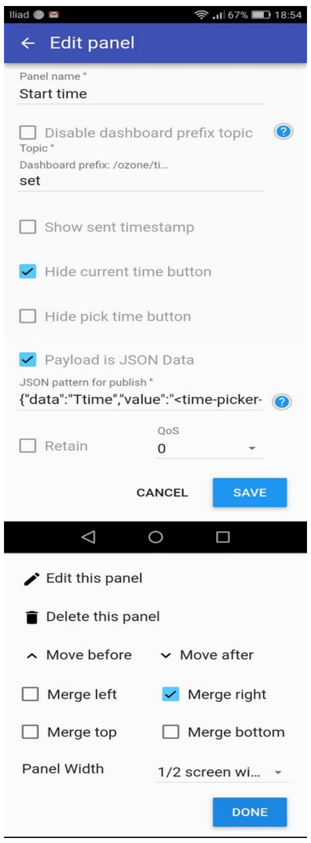
4.4 *Radio.button MODE (see img)*
Topic: set
Subscribe topic: status/config
JsonPhat for subscribe: $..[‘mode’]
JSON pattern for publish:
{“data”:”mode”,”value”:<radio-button-payload>}
dimension, ½; to the left

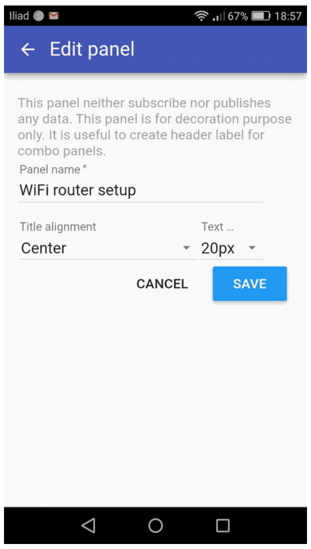
4.5 *Button START (see img)*
Topic: command
Payload: go
confirm: Ozone ON now?
small, *dimension: ¼; to the right*

4.6 *Button END (see img)*
Topic: command
Payload: go
medium, *dimensioni: ¼;to the right*


Iliad 68% 18:53
Edit panel
Panel name *
status
☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
status/info
Additional options
Text size
☑ Show last message only 20px
Max no of visible l...
2
No of persistence
☐ Show received timestamp ☑ Hide topic
Unit
QoS
0
☑ Payload is JSON Data
JsonPath for subscribe *
$..[0]
CANCEL SAVE


Iliad 68% 18:53
Edit panel
Panel name *
Set clock
☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
set
☐ Show sent timestamp
☑ Confirm before publish
Confirmation message *
set clock ?
☐ Hide current time button
☑ Hide pick time button
☑ Payload is JSON Data
JSON pattern for publish *
{"data":"nowtime","value":"<time-pick
☐ Retain
QoS
0
CANCEL SAVE
Edit this panel
Delete this panel
Move before Move after
☑ Merge left ☐ Merge right
☐ Merge top ☐ Merge bottom
Panel Width 1/2 screen wi...
DONE


Iliad 67% 18:54
Edit panel
Panel name *
Start time
☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
set
☐ Show sent timestamp
☑ Hide current time button
☐ Hide pick time button
☑ Payload is JSON Data
JSON pattern for publish *
{"data":"Ttime","value":"<time-picker-
☐ Retain
QoS
0
CANCEL SAVE
Edit this panel
Delete this panel
Move before Move after
☐ Merge left ☑ Merge right
☐ Merge top ☐ Merge bottom
Panel Width 1/2 screen wi...
DONE


Iliad 67% 18:57
Edit panel
This panel neither subscribe nor publishes any data. This panel is for decoration purpose only. It is useful to create header label for combo panels.
Panel name *
WiFi router setup
Title alignment
Center
Text ...
20px
CANCEL SAVE

4.7 *Text Log: status (see img)*
Topic: status/info
JsonPhat for subscribe: $..[0]

4.8 Time Picker *Set clock (see img)*
Topic: set

Payload is JSON data: {“data”:”nowtime”,”value”:“<time-picker-payload>”}
dimension, ½; to the left

4.9 Time Picker *Start time (see img)*
Topic: set
Payload is JSON data:
{“data”:”Ttime”,”value”:“<time-picker-payload>”}
dimension, ½; to the right

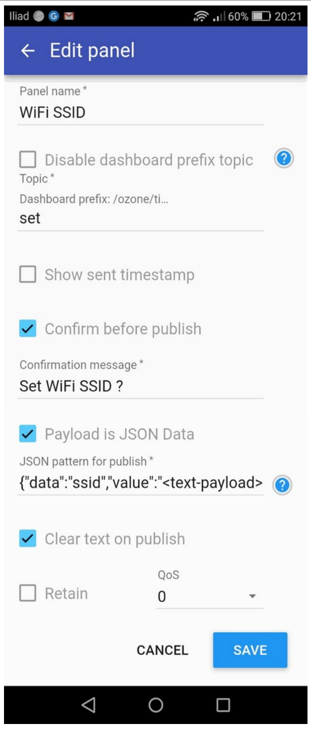
4.10 Layout Decorator *(see img)*

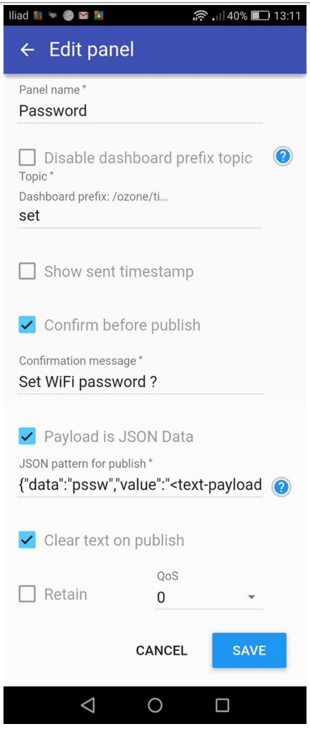
- *usato come separatore*

4.11 Text Input *WiFDi SSID (see img)*
Topic: set
Payload is JSON data:
{“data”:”ssid”,”value”:“<text-payload>”}
confirm: set WiFi SSID?

4.12 Text Input *Password (see img)*
Topic: set
Payload is JSON data:
{“data”:”pssw”,”value”:“<text-payload>”}
confirm: set WiFi password?

5) These 12 panels form the **timerPDM** Android app. When everything works you can lock the panels in the Edit Dashboard.

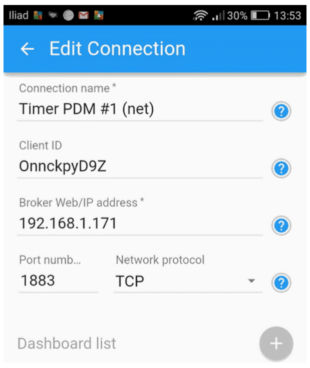

Edit panel
Panel name *
WiFi SSID
☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
set
☐ Show sent timestamp
☑ Confirm before publish
Confirmation message *
Set WiFi SSID ?
☑ Payload is JSON Data
JSON pattern for publish *
{"data":"ssid","value":"<text-payload>
☑ Clear text on publish
☐ Retain
QoS
0
CANCEL SAVE


Edit panel
Panel name *
Password
☐ Disable dashboard prefix topic
Topic *
Dashboard prefix: /ozone/ti...
set
☐ Show sent timestamp
☑ Confirm before publish
Confirmation message *
Set WiFi password ?
☑ Payload is JSON Data
JSON pattern for publish *
{"data":"pssw","value":"<text-payload
☑ Clear text on publish
☐ Retain
QoS
0
CANCEL SAVE

The solution proposed here is the simplest: it can be customized in infinite ways.

If you want an app on a smartphone even for **lanRouter** mode, you don't have to start over, but you can use the "clone connection" command which duplicates a connection. Then in Edit Connection make the following changes:

1. change the Connection name
2. delete the Client ID: a new one will be generated
3. change Broker Web / IP address, using the IP assigned by the router


Edit Connection
Connection name *
Timer PDM #1 (net)
Client ID
OnnckpyD9Z
Broker Web/IP address *
192.168.1.171
Port numb...
1883
Network protocol
TCP
Dashboard list

Note: *I think there are apps with the function of MQTT + dashboard also for O.S. other than Android, they must handle JSON payloads.*