REV 1

BALDR SMARTHOME

DOCUMENTATION

VI WA
BALDR.OS
Rev. 1

Content

1. Introduction and Usage

Baldr is a smart home project and represents an open source solution for controlling various IoT components. The current status is limited to LED control / lighting. The functions are constantly being developed.

2. Components

Baldr consists of different components. Essentially, the project is divided into µBaldr (microcontroller application), Baldr-UI and Baldr-Tools

2.1 µBaldr

µBaldr consists of different modules that perform different tasks. These are briefly explained under this section.

2.1.1 Modules

LightControl.py

LightControl.py takes over the control of the LEDs. All types of NeoPixel LEDs are supported. The number of possible color combinations is determined by the bytes-per-pixel (bpp). The value set here determines the length of the list that will be passed to the LEDs:

NeoPixel Module	bpp (value)
RGB	3 (255,255,255)
WW/CW	3 (255,255,255)
RGBW	4 (255,255,255,255)

The bpp-value is set in the config.json file.

Currently, the module supports the following functions for controlling NeoPixel-LEDs:

Line: LEDs are set by line animation (color)

- Adjustable are color, speed, starting point, distance, direction

Dim: Dimming of the LEDs (value 0... 100%) with adjustable speed

The last status is saved and automatically restored when the device is switched on again if the “Autostart” option is activated.

PicoWifi.py

The Wifi module connects the µBaldr device to the Wifi network. Connection options / access data are stored in the config file (config.json). If no connection has been established after 5 attempts, the device restarts. If the authentication fails, the connection attempt will be aborted immediately. The event is then logged into the log file (/log/Wifi.log).

logger.py

This module takes over the log function and saves log files in the /log directory for the respective modules.

mqtt_handler.py
The MQTT handler is responsible for communicating over the MQTT protocol and handles the connection/login and messages.

In addition, the module executes the OTA update when the corresponding command has been received.

The string contains a list of modules to be updated and must contain the base URL for the OTA update.

After a successful update, the client will be restarted.

PicoClient.py
This module essentially takes over control of the MQTT handler and the connection establishment. Incoming messages are passed to the MQTT handler.
The module also includes a **watchdog timer** that performs a connection check if no messages have been received after a certain period.

2.1.2 Configuration

The configuration is stored in the **/params** directory. JSON files are used for settings. There are 3 JSON files in this folder:

config.json
Configuration of the device. Includes LightControl Settings, MQTT-Config, Wifi-Config.

Can be created **via the** JsonConfigBuilder tool.

time_setting.json
This file contains settings for the NTP module.

status.json
Includes the current status (light-level, color)

3. Control via MQTT

The control of the µBaldr devices is implemented via the MQTT protocol. JSON strings are used for communication.

Communication takes place essentially via two topics.

device/order = Order topic for commands sent to the device

device/status = Status-Topic, which contain replies, and status messages

3.1 Structure and functions of the JSON string

The JSON string consists of several objects. sub_type and command must always be included. Depending on this, further parameters are required. See the following tables.

Example:

```
{
 “sub_type”: “LC”
 "command": "dim",
 "payload": 50,
 "speed": 5
}
```

=> The type "LC" (LightControl) is addressed here. With "command": "dim" the brightness (payload) is to be dimmed to 50%. The value "speed" describes the pause between the individual steps (1% steps) in milliseconds.

3.2 Subtypes / Module

3.2.1 LC

„command“: „line“

Object	Format	Value
payload	List or String / hex (nur bei bpp=3)	[255, 255, 255, 255] „ffffff“
format	String	"list" or "hex"
speed	Integer (higher = slower) *Pause between steps in milliseconds*	1... 100

With "hex" as the format, the payload must be provided as a string in hex format "ffffff". In case of "list", it must be provided as a list ([255,255,255] for RGB or [0,0,0,255] for RGBW).

"command": "dim"

Object	Format	Value
payload	Integer	0... 100
speed	Integer *Pause between steps in milliseconds*	1... 100

3.2.2 Admin

With this sub_type, it is possible to perform various admin functions.
The answers are given on the status topic.

Command	Parameters/payload	Format	Description
echo	-	-	Returns a string with the content "Pico_alive"
get_version	sub_system	String	Returns the version of the corresponding module
change_led_qty	new_value	Integer	Changes the number of LEDs
get_qty	-	-	Returns the number of LEDs
reboot	password	String	Performs a reset (password = "loki"
get_sysinfo	-	-	Returns the platform (e.g. ESP32)
set_GMT_wintertime	new_value	Boolean	Set the time to winter time/daylight saving time
get_update	module	List	Triggers an OTA update

3.2.3 Queries from the µClient to the MQTT Broker

3.2.3.1 Watchdog

The built-in watchdog function monitors the activity and thus the connection to the MQTT broker. Watchdog is triggered when the last message was more than 60s ago. It compares the timestamp of the last message with the current time. A cooldown time of 5s prevents Watchdog from being triggered too often in sequence.

```
if watchdog_last_chk and pico_time - watchdog_last_chk < cooldown:
        return True
```

When Watchdog is triggered, the µClient sends a message to the status topic with the content "echo". With a `client.wait_msg()` function, the program is interrupted until it receives a response on the order topic.

3.2.3.2 Online and offline notification

The µClient sends "online" on the /status topic after a successful connection establishment and "offline" when the client terminates the connection.

3.2.4 OTA Update Feature

The admin command "get_update" can be used to trigger an OTA update to update the µBaldr software. This requires the base URL and a list of modules to be updated.

```
{
"sub_type": "admin",
"command": "get_update",
"module": [
"MODUL1.py",
"MODUL2.py",
],
"base_url": "OTA-BASE-URL"
}
```

After the update has been successfully executed, a status message is sent on the /status topic and then a reboot. Errors are logged in OTA.log.

4. Raspberry Pi as an MQTT broker and UI

This section describes how to set up the MQTT broker and the GUI for Baldr-Smarthome.

An MQTT broker is needed to reliably transmit the messages and control the devices.

The setup here is done with a Raspberry Pi 5 8GB under Ubuntu 24.04LTS. For the desktop version, at least 4GB of RAM should be available. RaspbianOS can also be installed (For RaspberryPI-specific features and models <4).

4.1 Setting up MQTT brokers

To make it easier to work, the superuser password can be removed:

```
$ sudo passwd -d root
```

4.1.1 Installing Mosquitto

First, the system should be updated:

```
$ su
# apt update
# apt upgrade -y
```

Then install Mosquitto, activate autostart and then start:

```
# apt install mosquitto mosquitto-clients -y
# systemctl enable mosquito
# systemctl start mosquito
```

4.1.2 Set up user and password (optional)

If the broker is to work password-protected, create a user as follows:

```
# mosquitto_passwd -c /etc/mosquitto/passwd <NEW_USER>
```

Then add these lines into /etc/mosquitto/mosquitto.conf:

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

And restart Mosquitto

```
# systemctl restart mosquitto
```

In some cases, Mosquitto can then no longer be started. The reason for this is that the password file has no permissions. This can be fixed by giving the file write and read access for all users:

```
# chmod 666 /etc/mosquitto/passwd
```

4.2 Create NodeRED Dashboard (optional)

4.2.1 NodeRed via App-Center

Under Ubuntu 24.04LTS, NodeRED can be installed via the App Center. In this case, you can skip to the setup NodeRED part.

4.2.2 NodeRED via Docker

On some distributions (Raspbian Bullseye) NodeRED cannot be installed natively. Docker can be used as an alternative

4.2.2.1

To install Docker automatically, according to the system, run the following script:

```
$curl-sSl https://get.docker.com | sh
```

Then add the user to the Docker group and perform a reboot:

```
$ su
# usermod -aG docker $USER
# systemctl enable docker
# docker volume create portainer_data
# docker run -d -p 9000:9000 -p 8000:8000 \
  --name=portainer --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data \
  Portainer/Portainer-CE
# reboot
```

Note: *$USER must be replaced by the username!*
The web GUI can then be accessed at http://<IP address>:1880

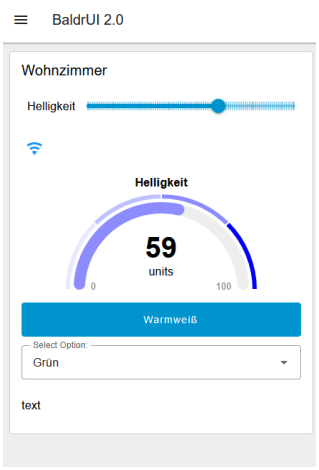
4.2.2.2 Creating Node-RED Containers

After reboot, create Node-RED containers with autostart:

```
$ docker run -d -p 1880:1880 \
  -v node_red_data:/data \
  --restart unless-stopped \
  --name mynodered \
  nodered/node-red
```

Note: *The "\" characters are only needed if the input is multi-line.*

4.2.3 NodeRED Dashboard 2.0 einrichten



After installation, Node-RED can be accessed via port 1880 in the web browser:

http://<IP Address>:1880

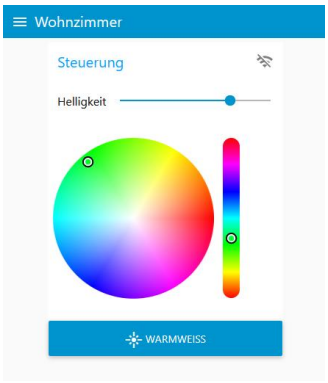
In Node-RED, then install the dashboard via the web browser:

1. Menü > Manage palette > install
2. Search for @flowfuse/node-red-dashboard
3. Installing Node

The dashboard can then be accessed at the following address:

http://<IP-Address>:1880\dashboard

Afterwards, the flows can be imported from the BaldrUI or your own flows can be created.



Note: Currently, the Dashboard 2.0 does not include a color picker and is therefore not supported. To use Color Picker, the "old" Dashboard 1.0 must be used for the time being.

On the left is an example of Dashboard 1.0 with Color Picker.

4.2.3 Importing BaldrUI Flows (Optional)

Optionally, the BaldrUI flows can be imported and customized. Will provide it in the future.

As already mentioned, there is no color picker (native) in Dashboard 2.0 yet.