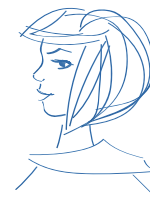


Blu2Light

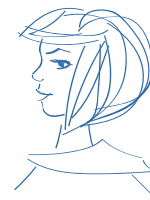


Technical Application Guide LAN Gateway Server Demo



Contents

1 Warranty, warnings, limitation (TODO)	3
1.1 License of the demo software	
2 Thermes used in this document.....	3
3 Introduction	3
3.1 Why Python™?	3
4 Installation and configuration of the VS demo software	3
4.1 Installation on Raspberry Pi	4
4.2 Installation on Windows server.....	4
4.3 Update of the VS demo software on Raspberry Pi.....	4
4.4 Configuration.....	5
4.5 Configuration page.....	7
4.6 Grafana Login.....	7
4.7 Data storage in the demo.....	7
4.8 Backup	7
4.9 Sample configuration of the actual working system.....	8
5 Updating the VS LAN Gateway firmware	14
5.1 General limitations of the update process	14
5.2 Update process with VS Update Tool for Windows.....	14
5.3 Update via web configuration page of VS demo server ..	15
6 Substitution or replacement of a Blu2Light LAN Gateway	17
7 Communication between the LAN Gateway and the server	18
7.1 The Blu2Light protocol	18
7.2 Available events from sensors.....	19
7.3 Used Cipher Suite.....	19
7.4 Ports.....	19
8 Commands.....	19
8.1 Ping	19
8.2 How to encode a B2L command	20
8.3 How to send a B2L frame.....	20
8.4 How to create a light control command	20
8.5 How to read the function group state.....	21
8.6 How to create a DALI tunnel	21
8.7 PMD commands.....	22
8.7.1 PMD initialization.....	22
8.7.2 PMD retrieve	22
8.7.3 PMD quit.....	23
8.7.4 Using PMD with the server demo	23
8.7.5 Adding the power measurements to Grafana	23
8.8 B2L Encryption method	24
9 Where to pull data	24
9.1 Read database (recommended)	24
9.2 When written to database	24
9.3 Retrieve events	24
9.4 Using B2L commands	24
10 Beaconing.....	25
11 Additional Information	25
11.1 WEB-API calls	25
11.1.1 /api/get_status	25
11.1.2 /api/pmd_init.....	25
11.1.3 /api/get_config.....	25
11.1.4 /api/get_rtc_time.....	25
11.1.5 /api/set_rtc_time	25
11.1.6 /api/set_config	26
11.1.7 /api/delete_system	26
11.1.8 /api/get_errors.....	27
11.1.9 /api/get_messages	27
11.1.10 /api/get_update_status.....	27
11.1.11 /api/upload	27
11.1.12 /api/update.....	27
11.1.13 /api/light_control.....	28
12 Troubleshooting	29
12.1 Bad CSRF token	29



1 Warranty, warnings, limitation (TODO)

VS does not provide any warranty. Also, the document may change without prior notice. Vossloh-Schwabe is not responsible for any kind of usage of the software, especially Vossloh-Schwabe is/does not:

- Provide warranty on data loss
- Care about necessary network security
- Provide data protection
- Provide bug-free code

The user of the documentation and the user of the final installation are responsible that all legal requirements in the country where this is used are fulfilled. Especially requirements on data protection, security have to be fulfilled.

1.1 License of the demo software

The demo software is provided according to the MITlicense. This is the granted license therefore:

Copyright (c) 2022 Vossloh-Schwabe Deutschland GmbH

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "software"), to deal in the software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software, and to permit persons to whom the software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2 Terms used in this document

VS	Vossloh-Schwabe Deutschland GmbH
PSK	Pre-shared key

3 Introduction

VS provides a LAN Gateway (187055) to connect a Blu2Light system to ethernet. This enables the integration of a Blu2Light system in various kinds of application.

To enable the integration Vossloh-Schwabe provides a demonstration written in Python™ running for example on a Raspberry Pi.

The demonstration is easy to setup and builds a basis for all different variants of implementation.

3.1 Why Python™?

The LAN Gateway software is written in Python™ because it's a common high-level interpreted programming language. One of its advantages is, that it runs on every common OS. It's also easy to read and it is open source.

4 Installation and configuration of the VS demo software

The VS demo software is intended to be installed on the stock Raspberry Pi OS. Please follow the instructions provided at <https://www.raspberrypi.com/software/> how to basically set up a new Raspberry Pi OS and gain SSH login.

The demo software will show:

- How to establish a connection between the server and the Gateway
- How to receive and decode incoming mesh events
- Store the incoming events in a database
- Visualize incoming data in Grafana
- Provide an easy web-based UI to control functional groups in a Blu2Light system
- Read out the systems PMD data (if provided by the DALI drivers)



4.1 Installation on Raspberry Pi

1. Download the latest version of "Raspberry Pi OS" or "Raspberry Pi OS lite" (32 Bit or 64 Bit) on an SD card. The instructions therefore and all required downloads can be found here: <https://www.raspberrypi.com/software/>.

Tip: for installation with "Raspberry Pi Imager" press **CTRL + SHIFT + X** for advanced settings like SSH or WIFI.

2. Copy the provided .zip file to the Raspberry Pi and to the desired installation path.

- Method 1: get www.vossloh-schwabe.com/en/products/light-management-indoor/blu2light-iot-devices/blu2light-gateway (file is located under the tab SAFTEY & INSTALLATION)
- Method 2: if you use "Raspberry Pi OS" with a graphical interface, you can download the zip file directly
- Method 3: use an USB drive
- Method 4: use other file transfer methods, like Samba or SCP

3. Extract the contents of the file with `unzip <filename>`

4. Go into the extracted folder and run `sudo chmod +x setup.sh` to make setup.sh executable.

5. Run `sudo ./setup.sh` to start the setup routine.

While the setup is running, you will be asked which database system you want to use (MariaDB or InfluxDB) and for the credentials and the database name, so that the database and the server can be configured.

Attention: If you use MariaDB-database, please do not use the user "root" as a username for the MariaDB-database. This will cause a malfunction of the VS LAN Gateway software after every start and the device will not work properly.

The setup script updates all packages on the Raspberry Pi and installs the necessary packages for the server.

6. After the setup has been run successfully, it will show you the hostname / IP of your devices. Note down this information carefully.

7. After the setup has finished, you can now enable the server service to start on every reboot of the Raspberry Pi by the the following command `sudo systemctl enable lan-gateway`

8. After that, start the server service the first time with `sudo systemctl start lan-gateway`

4.2 Installation on Windows server

1. Download and unzip the installation files.

2. Start PowerShell 7 (or higher) as administrator.

3. Type in & „[path]\setup.ps1" f.e.: & „T:\media converter\setup.ps1"

4.3 Update of the VS demo software on Raspberry Pi

1. Download the latest version of "Raspberry Pi OS" or "Raspberry Pi OS lite" (32 Bit or 64 Bit) on an SD card. The instructions therefore and all required downloads can be found here: <https://www.raspberrypi.com/software/>.

Tip: For installation with "Raspberry Pi Imager", press **CTRL + SHIFT + X** for advanced settings like SSH or WIFI.

2. Copy the provided .zip file to the Raspberry Pi and to the desired installation path.

- Method 1: get www.vossloh-schwabe.com/en/products/light-management-indoor/blu2light-iot-devices/blu2light-gateway (file is located under the tab SAFTEY & INSTALLATION)
- Method 2: If you use "Raspberry Pi OS" with a graphical interface, you can download the zip file directly
- Method 3: Use an USB drive.
- Method 4: Use another file transfer methods, like Samba or SCP.

3. Extract the contents of the file with `unzip <filename>`

4. Go into the extracted folder and run `sudo chmod +x update.sh` to make update.sh executable.

5. Run `sudo ./update.sh` to start the update routine.

The update script updates all packages on the Raspberry Pi and updates the necessary packages for the server.



6. After the update has been finished successfully, it will show you the hostname / IP of your devices. Note down this information carefully.

7. After the successful update, start the server service with `sudo systemctl start lan-gateway`

4.4 Configuration

1. All configuration of the server can be done via the web interface, which is accessible at `http://<IP or domain of the device>:31460/`

2. Before the LAN Gateway can connect to the server, a pre-shared key (PSK) must be generated via the web interface. For simplicity a QR-Code with the PSK is created, which can be scanned and copied (f.e. with the iPad camera app). Afterwards, the IP address, port and pre-shared key must be configured to the LAN Gateway via the LiNA Connect app. Therefore the Gateway must have been added to a system in the LiNA Connect app and the system must be in expert mode. When everything was configured correctly, LED1 on the LAN Gateway will light up green.

The pre-shared key is generated in the web interface of the raspberry pi (`http://<IP or domain of the device>:31460`).
The pre-shared key is used to encrypt the communication between the Raspberry Pi and the Blu2Light LAN Gateway.

The following screenshot shows the generating of the key. By clicking on the field "Generate PSK" a pre-shared key is generated randomly.

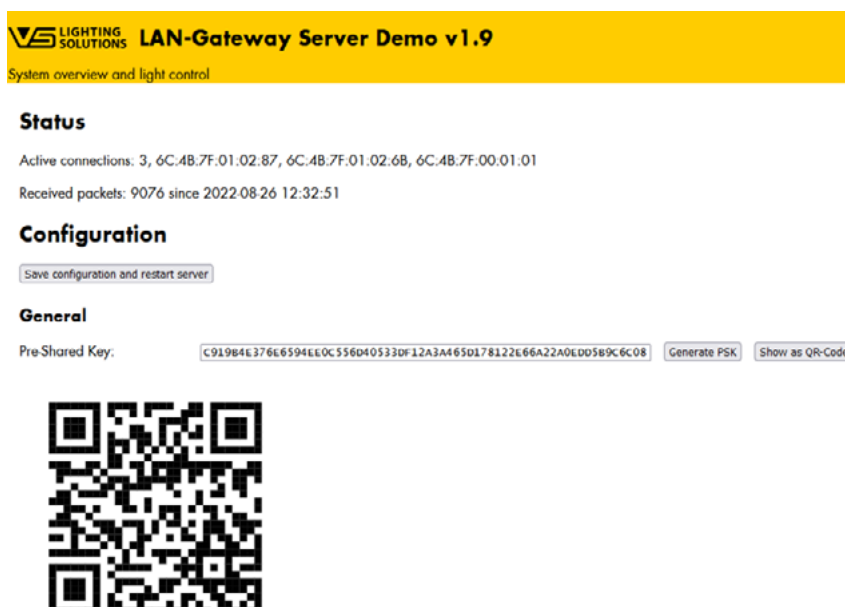


Figure 1: Generating the pre-shared key in the web interface of the Raspberry Pi

The key must now be copied into the appropriate field in the LiNA Connect app. For that, go into the network settings of the Gateway and choose "pre-shared key".

If an additional Blu2Light LAN Gateway will be used, a click on "Show as QR-Code" will open the suitable QR-Code for the current server, which has already been generated before. The code can then be assigned to the new Blu2Light LAN Gateway that shall be added to this server.

It is important to know, that only one "pre-shared key" exists or can exist for a server. At the end, all connected Blu2Light LAN Gateways must have the same "pre-shared key" assigned.

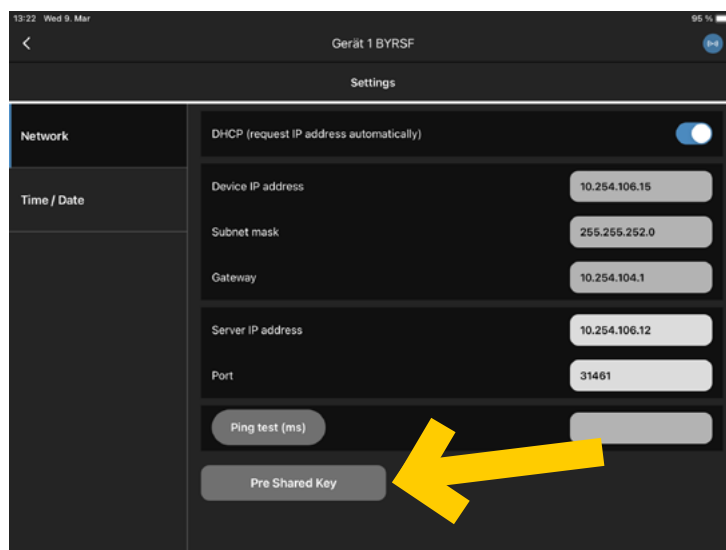


Figure 2: "pre-shared key" button for adding the generated pre-shared key from the raspberry pi.

Now select the QR symbol to scan the PSK from your screen and then press "save".

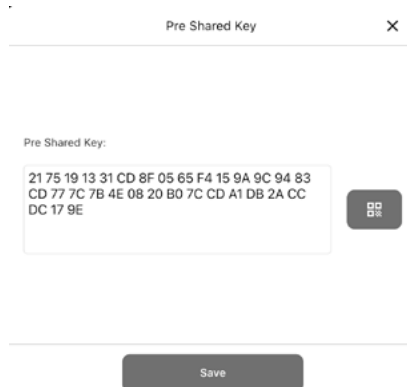
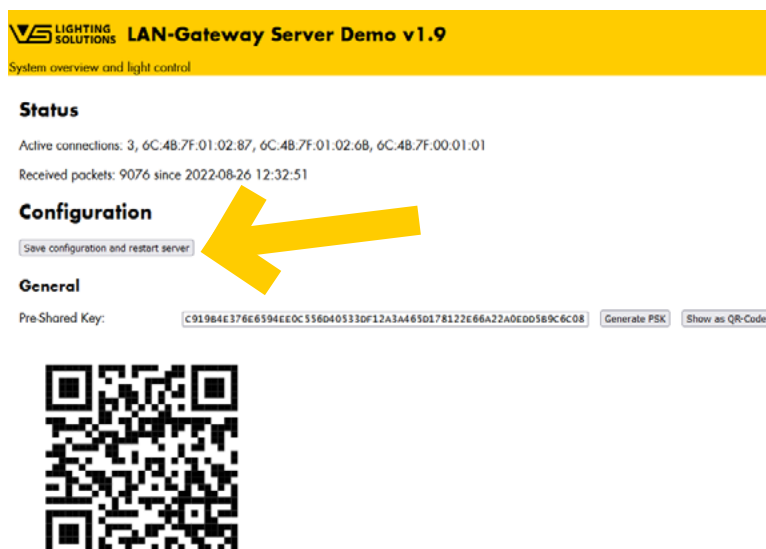


Figure 3: Add the PSK to the App

After the pre-shared key has been copied to LiNA Connect, it shall be saved in the web interface of the Raspberry Pi:





After this step has been done successfully, the LED 1 on the Blu2Light LAN Gateway should be green.
The pre-shared key will be visible in the web configuration. If needed at a later point it can be taken out of the following field:

General

Pre-Shared Key:

Server port: (default: 31461)

Figure 4: pre-shared key - visible in the web configuration

Additional Information: The used encryption method for the communication between Server and LAN Gateway is: TLSv1.2 ECDHE-PSK-CHACHA20-POLY1305.

4.5 Configuration page

The demo provides a configuration and control interface via web browser. Navigate to <http://<IP or domain of the device>:31460> to access the configuration

4.6 Grafana Login

The demo software will create a template of Grafana screens depending on your system. To access Grafana go to <http://<IP or domain of the device>:3000> and login with the default credentials *admin / admin*. You then must change your password.

4.7 Data storage in the demo

The configuration of the demo software is stored in the following location:

Raspberry Pi OS:

`/var/vs_lan_gateway_server/config.json`

Windows Server:

`C:/Users/All Users/.vs_lan_gateway_server/config.json`

If you have selected MariaDB, the database is located here:

Raspberry Pi OS:

`/var/lib/mysql`

Windows Server:

`C:/Users/All Users/scoop/persist/mariadb/data`

For more information about MariaDB: <https://mariadb.org/>

The pre-defined templates for Grafana are located here:

Raspberry Pi OS:

`/var/vs_lan_gateway_server/dashboards`

Windows Server:

`C:/Users/All Users/.vs_lan_gateway_server/dashboards`

Some additional data for Grafana are located here:

Raspberry Pi OS:

`/etc/grafana/provisioning/datasources`

`/etc/grafana/provisioning/dashboards`

Windows Server:

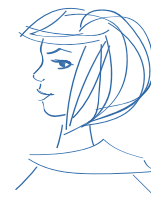
`C:/Users/All Users/scoop/persist/grafana/conf/provisioning/datasources`

`C:/Users/All Users/scoop/persist/grafana/conf/provisioning/dashboards`

For more information about Grafana, and how graphical panels are created: <https://grafana.com/>

4.8 Expected Database size

For every B2L device added to the system, the user should expect about 2-10 Mbyte of data per month.
Make sure that you have enough memory or add a cleaning function.

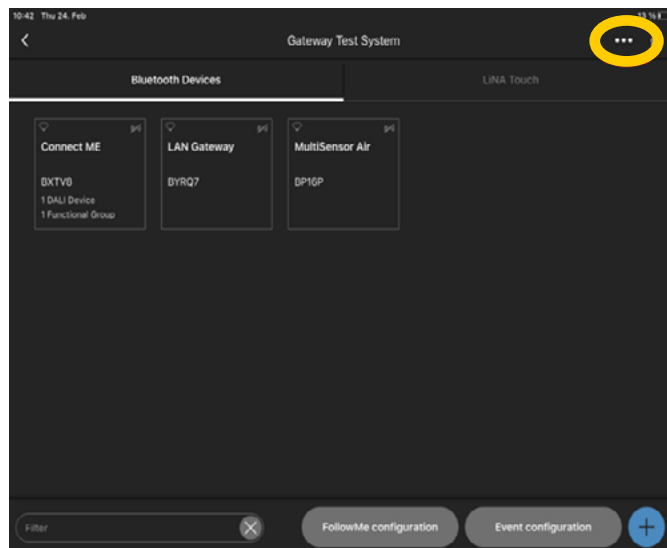


4.9 Backup

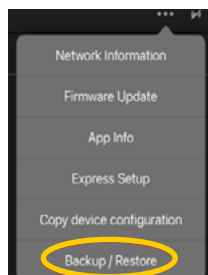
To prevent data loss, it is generally recommended to frequently backup your data. Beside the basic data from the OS, make sure the data locations mentioned in paragraph 4.7 are also backed up.

4.10 Sample configuration of the actual working system

To get the sample configuration running, a backup of the B2L system must have been created in the LiNA Connect app. For that go to the "... " menu in the system overview



Select "Backup/Restore"



Now go to "Create new backup/Export current configuration". If desired, give the system a new name. Now, a new backup on the tablet can be created, by tapping on "Create new backup", but to import the system to the Gateway web interface the system needs get exported to the device, where the web interface is opened. Therefore, go to "Export of system configuration" and then "Share". Now the backup can be up-loaded to the cloud or sent to an email address.

Now go back to the web interface and upload the backup with the MAC address of the Blu2Light LAN Gateway.



Figure 5: Import of the recorded backup-file in the web configuration

After clicking on "import file" a message shall appear, that the configuration has been imported successfully. If the message has been appeared, a login to Grafana ([http:// < IP or domain>: 3000](http://<IP or domain>: 3000)) can be done. Please keep in mind that the first login to Grafana is only successful with "admin" as user and as well as password.

You will be forced to change the password directly after the first login has been successful. After the password has been changed you will be forwarded to the welcome page.



The "imported system" shall be visible in the Grafana environment. When you login the first time you will have to click on the upper left corner on "manage". The system shall become visible:

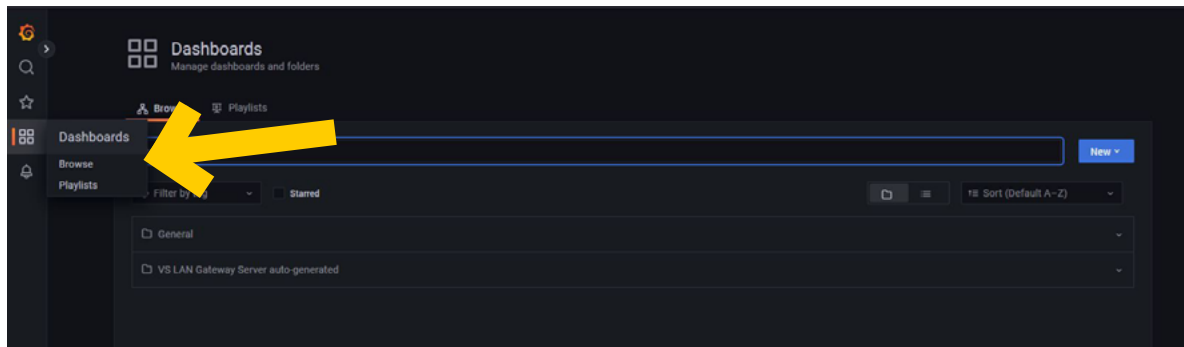


Figure 6: Dashboard-settings – the automatic generated dashboard shall be visible in the middle after a click on "Dashboards"

After the 2nd login, the dashboard should appear on the starting page in Grafana:

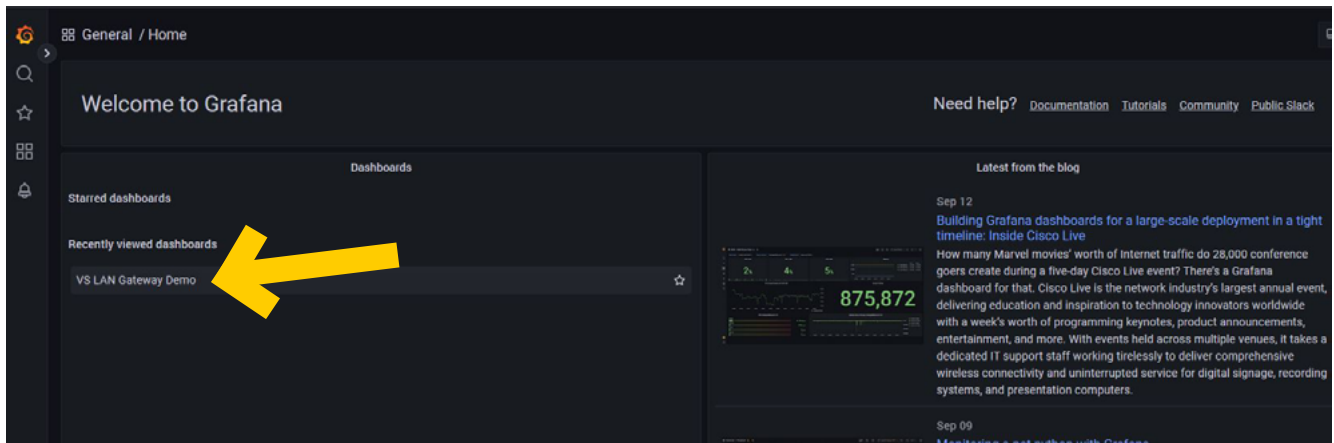


Figure 7: The imported „system“ shall be visible in Grafana after a successful import



If you click on the dashboard, the demo site shall be visible where the incoming data can be viewed:

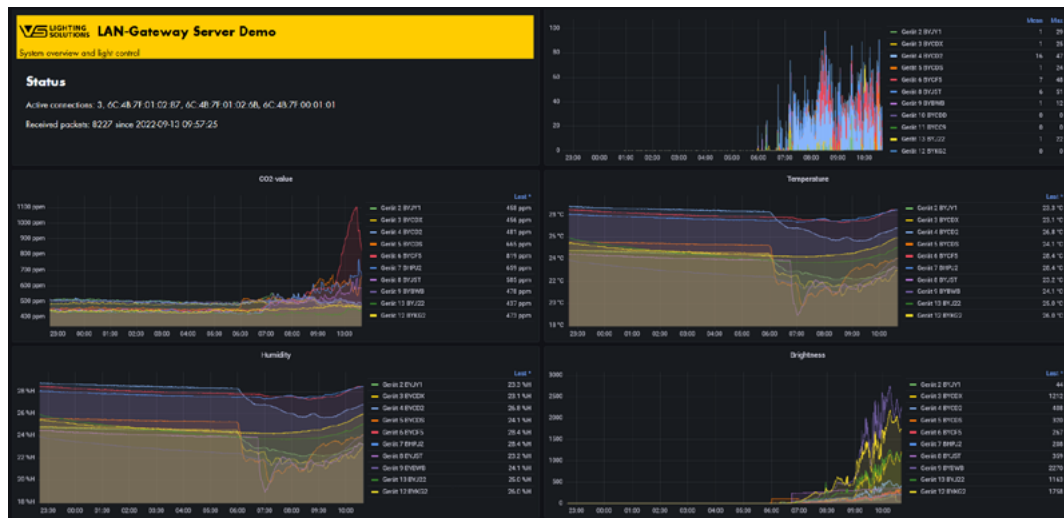


Figure 8: Example site of the imported system

You can adjust all settings add or delete a dashboard and its content as you wish and desire. To this end the „debug-mode“ in LiNA Connect must be enabled to access this feature in the settings of the Blu2Light Multisensor AIR:

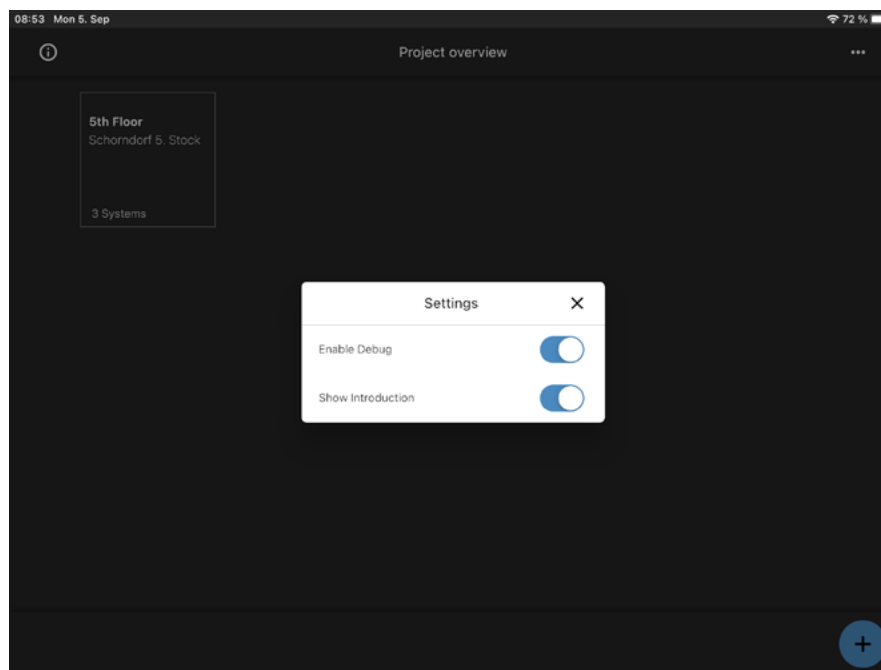


Figure 9: Enabling the „debug-mode“ in LiNA Connect



There is one important thing to prepare before you can see data like shown in the figure above. The interval of the incoming data must be set up at the corresponding device. The following example shows the necessary steps on a Blu2Light Multisensor AIR.

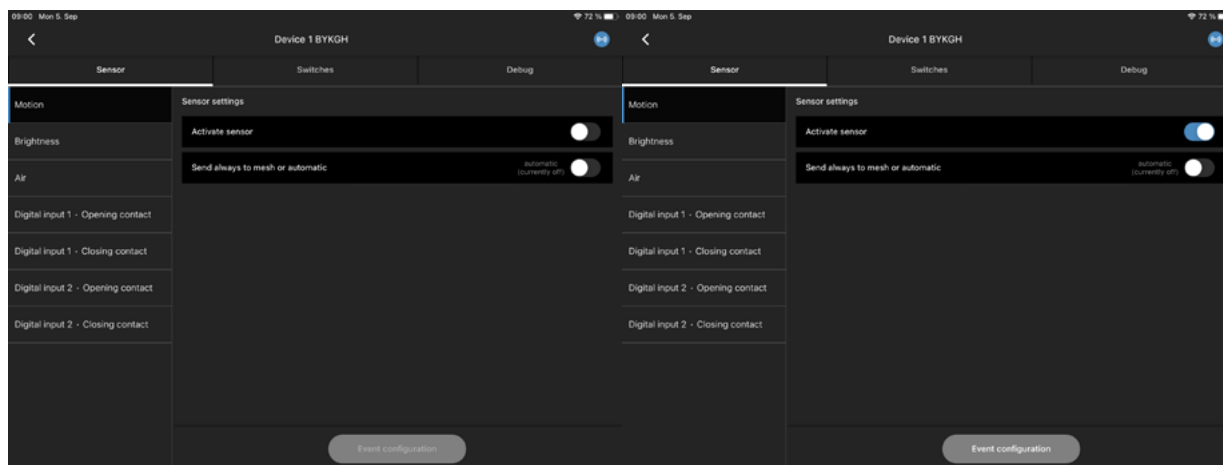


Figure 10: Step 1 (left) and step 2 (right) – activating the motion sensor

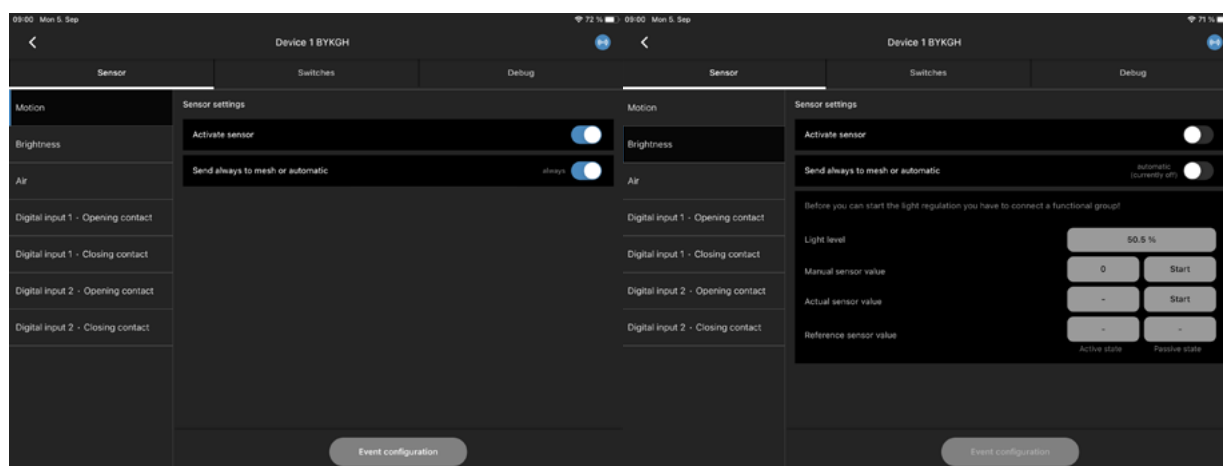


Figure 11: Step 3 (left) – activating “send always to mesh” of the motion sensor and step 4 (right) – brightness menu

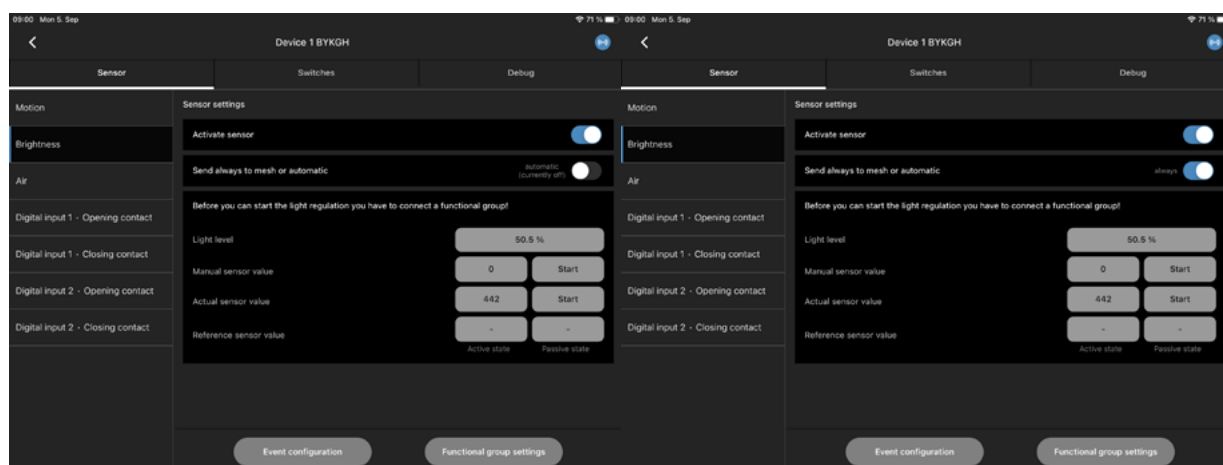


Figure 12: Step 5 (left) – activating the brightness sensor and step 6 (right) – activating “send always to mesh” of the brightness sensor

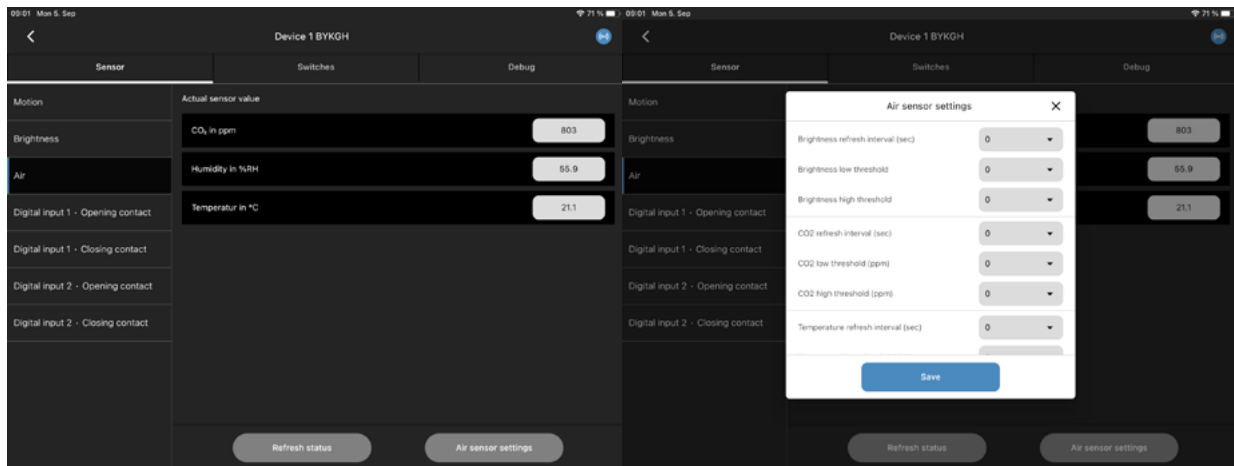


Figure 13: Step 6 (left) –and step 7 (right) – setting the data interval for data from Blu2Light Multisensor AIR

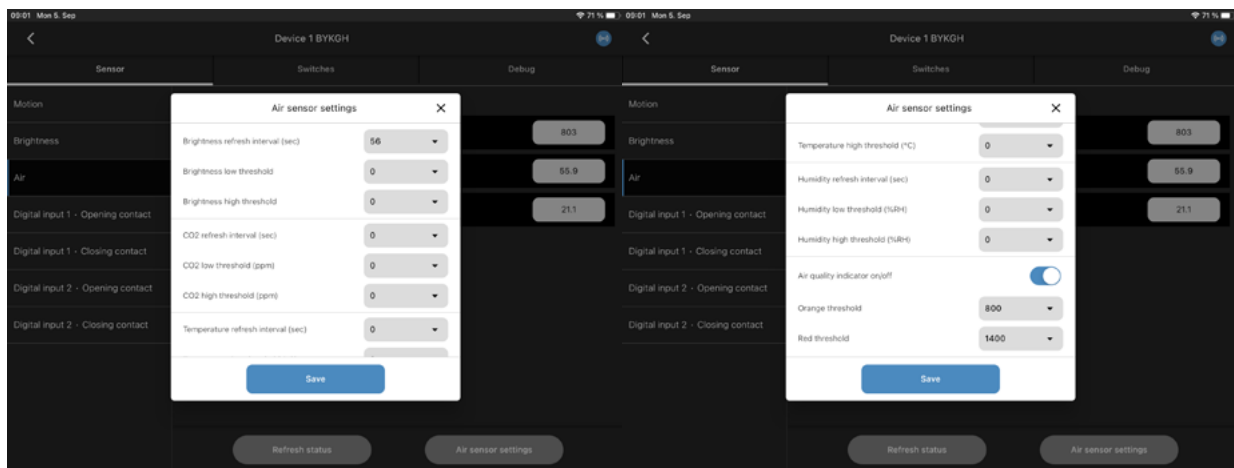


Figure 14: Step 7 (left) –and step 8 (right) – setting the data interval for data from Blu2Light Multisensor AIR

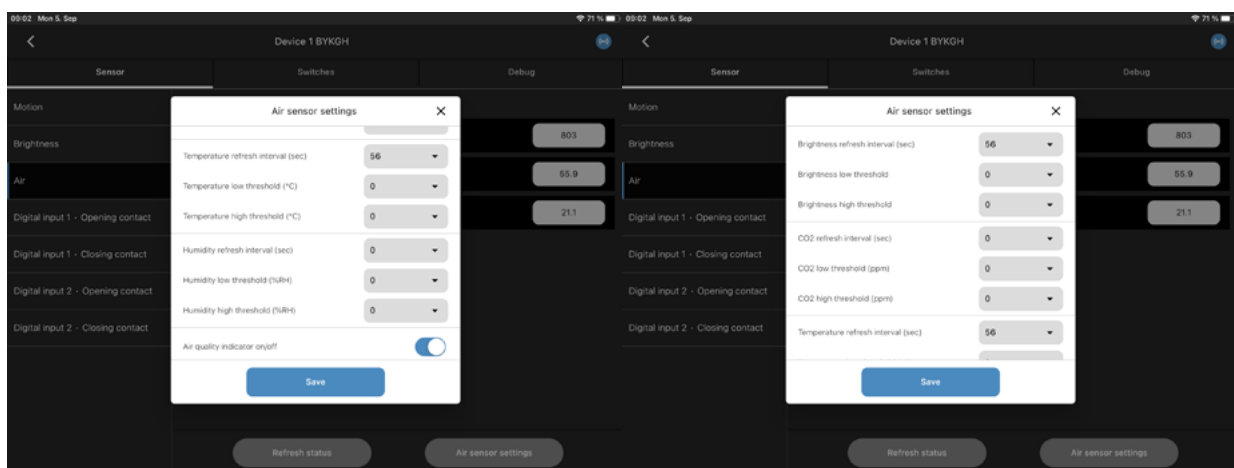
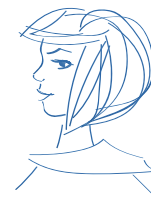


Figure 15: Step 9 (left) –and step 10 (right) – setting the data interval for data from Blu2Light Multisensor AIR

If the steps above have been configured, the data shall appear in Grafana within the configured time intervals. For a Blu2Light Multisensor XS, the brightness and motion sensor has only to be activated in the menu. As well as “send always to mesh”. The interval is fixed and cannot be changed.



5 Updating the VS LAN Gateway

The VS LAN Gateway has a firmware update function. You can use either the demo software provided by VS to update the device, or you can use a separate update program. The Gateway can be updated within the web configuration site. In the following, the general limitations of the update process are being described:

5.1 General limitations of the update process

- The device must be in the same subnet for the update process.
- If the device is powered via PoE and is supplying other devices in a daisy chain, note that the chained devices will be disconnected from PoE during the update process.
- If you use the Windows update tool you must provide a DHCP server in your network.
- Attention: Allow Windows firewall to release the port for an update.

5.2 Update process with VS Update Tool for Windows

For the update process you must fill in the IP of the device and the MAC Address. You can find the IP in the LINA-Connect App. The MAC address you can find on the device label.

The indication of the LEDs is as well shown by the horizontal and vertical bar in the software. After setting up the IP address and the MAC address of the Blu2Light LAN Gateway that shall be updated, the update can be started by clicking on the button "update Firmware" (yellow arrow):

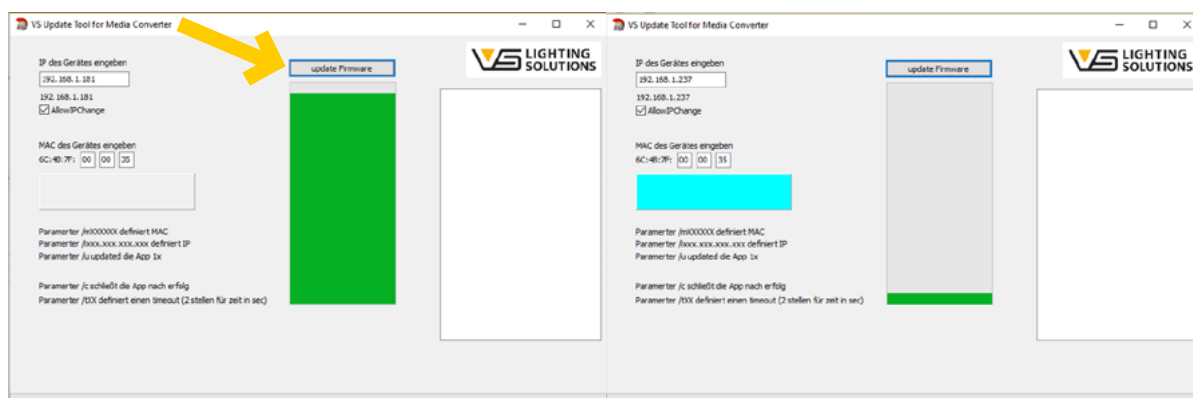


Figure 16: Update with the VS Update Tool for the Blu2Light LAN Gateway #1

If a timeout occurs somehow, the Update Tool will automatically make another try to load the firmware onto the device.

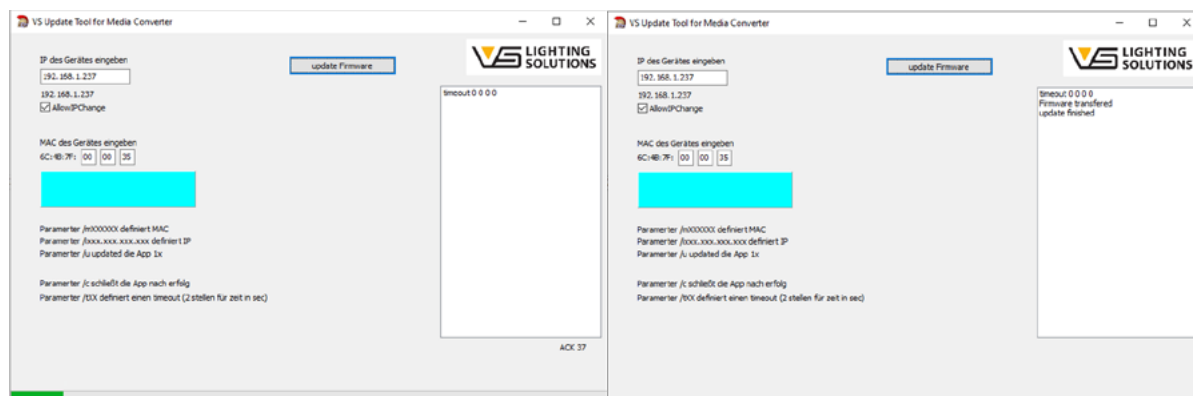


Figure 17: Update with the VS Update Tool for the Blu2Light LAN Gateway #1



5.3 Update via web configuration page of VS demo server

The device can as well be updated via the web configuration. The following steps describe the update process and show the LED indication in each corresponding mode.

The LED 1 and LED 2 of the Blu2Light LAN Gateway will show the following color while the LAN Gateway is in standard mode when it is receiving data from the mesh where it is commissioned to. LED 1 shines constantly green while LED 2 flashes green while data from the mesh is incoming:

LED 1	LED 2

By selecting the VS provided "bin-File" and putting the MAC address of the Blu2Light LAN Gateway in the field "MAC address" the update process will start by clicking on "Start update" (see the red arrow in the next picture):

Update LAN gateway

Select firmware update file firmwareMediaConverter.bin

MAC address

Figure 18: "Update LAN Gateway" mask in the web management of the Raspberry Pi

After a click on "Start update", the Gateway is being put into update mode. LED 1 will shine blue, while LED 2 will be off. The message "Restart gateway in update mode" will be shown as well:

Update LAN gateway

Select firmware update file firmwareMediaConverter.bin

MAC address

Restart gateway in update mode...



Figure 19: The Blu2Light LAN Gateway is put into "update mode"

LED 1 and LED 2 will indicate the following status:



LED 1	LED 2

LED 1 shines green and LED 2 will be dark for about 1.5 seconds.





LED 1	LED 2
	

LED 1 will be dark and LED2 shines green for about 10 seconds.

LED 1	LED 2
	

LED 1 will blink blue (1 Hz) and LED 2 will be dark for about 2 seconds.

LED 1	LED 2
	

LED will shine constantly blue and LED 2 will be dark for about 1 second.

The update process starts when LED 1 becomes dark and when LED 2 will constantly shine green. The whole progress will be shown down in the update windows in the web configuration:

Update LAN gateway



Select firmware update file firmwareMediaConverter.bin

MAC address

Transmitting frame 1 of 371...

Figure 20: Frame 1 of 371 is being transmitted

While the update process takes place, the LED 2 on the Blu2Light LAN Gateway will shine green. LED 1 will stay dark:

LED 1	LED 2
	

The update process will take around approximately 60 seconds – LED 1 will be dark and LED 2 will be constantly green depending on the firmware size that is being flashed.

When the update process has been successfully finished, the following message "Update finished" will appear down next to the button "Start update":

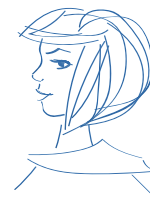
Update LAN gateway

Select firmware update file firmwareMediaConverter.bin



MAC address

Update finished.



Figure 21: The Blu2Light LAN Gateway is put into bootloader-mode





After a successful update, LED 1 will be constantly blue and LED 2 will be off. The Blu2Light LAN Gateway will restart and after a short yellow phase of LED 1 (Blu2Light LAN Gateway has a "IP address"), the LED 1 will shine constantly, while LED 2 will flash when data is incoming:

LED 1	LED 2
	

LED 1 will be constantly blue and LED 2 will be off for about 8 seconds.

LED 1	LED 2
	

LED 1 constantly yellow and LED 2 will be off for about 0,5 seconds

LED 1	LED 2
	

The Blu2Light LAN Gateway will be back in normal operation mode (LED 1 constantly green and LED 2 flashing when data from the mesh arrives).

6 Substitution or replacement of a Blu2Light LAN Gateway

If it is necessary to replace an already full configured Blu2Light LAN Gateway, the following steps shall be done:

1. Delete the "old" Blu2Light LAN Gateway in the corresponding system in LiNA Connect.
2. In the next step, make a commissioning of the new Blu2Light LAN Gateway into the corresponding system.
3. Configure the new Blu2Light LAN Gateway with the same parameters as IP-Address, Subnet/Mask, Server IP Address and Port of the "old" Blu2Light LAN Gateway.
4. Create a new "pre-shared key" or take the old key and put it into the corresponding field in the App *LiNA Connect* at the Blu2Light LAN Gateway options.
5. Make a new copy of the current system in the option menu of Blu2Light LiNA Connect ("Backup and Restore").
6. Import the "Backup and Restore" configuration-file and import this file into the configuration of the Raspberry Pi. Navigate to <http://<IP or domain of the device>:31460> to access the configuration like shown in the following screenshot:



Figure 22: Import of the configuration file of the corresponding system in the web configuration

7. The LED1 on the Blu2light LAN Gateway shall be green then and the configuration of the replaced or substituted Blu2light LAN Gateway has been successfully completed.



7 Communication between the LAN Gateway and the server

The communication is based on a SSL/PSK encrypted connection. The server acts as a socket for several Gateways to connect to. All incoming traffic on the Bluetooth side is forwarded to the ethernet socket.

7.1 The Blu2Light protocol

The communication frames follow the Blu2Light protocol.

Group	Offset	Size	Field	Value	Description
-	0	1	Sync Byte	0x42 = 66 = 'B'	Serial sync byte; always 0x42
Header	1	1	Address Length = AL	0xnn	
	2	2	Block Count = BC	0xnnnn	Data Length (DL) is $BC * 8 - PL - 2$
	4	2	Packet Type	0xnnnn	See Appendix F
	6	1	Packet Version	0xnn	
	7	2	Manufacturer	0xnnnn	0 = according open standard 1 = VS
	9	1	CRC Header	0xnn	
Address	10	1	Address Status	0xnn = 0b0f0edcba	Bit field for present Address data (1 = present): a = Source ID field b = Destination ID field c = Destination Type field d = Packet ID field e = Encryption Type field (0=unencrypted) f = Timestamp field
		2	Source ID	0xnnnn	ID of the source node
		2	Destination ID	0xnnnn	ID of destination node
		1	Destination Type	0xnn	0 = Message between BT/MWAYfirmware 1 = Mesh-Message
		1	Packet ID	0xnn	Necessary to identify response frame
		1	Encryption Type	0xnn	0 = NodekeyID 1 = SystemkeyID N= UserkeyID#N-2; (N=2..251)
		4	Timestamp	0xnnnnnnnn	
		1	CRC Address	0xnn	
Data (encrypted)		1	PL	0xnn	Length of Padding Data
		DL	Data	...	$DL = BC * 8 - PL - 2$
		1	CRC of Data	0xnn	CRC of decrypted Data
		PL	Padding data 0x00	...	Add Padding to reach $DL + PL + 2 = BC * 8$
		1	CRC Data	0xnn	
		1	CRC of CRCs	0xnn	CRC of CRC Header & CRC Address & CRC Data



7.2 Available events from sensors

ET_SENSE_MOVEMENT (Movement events; MultiSensor XS, XL, Air)

EventNumber	Input number	Count of movement events in measurement interval
6 (1 Byte)	(1 Byte)	(1 Byte)

ET_SENSE_BRIGHTNESS (Brightness events; MultiSensor XS, XL, Air)

EventNumber	Input number	Current brightness value	Target brightness value
7 (1 Byte)	(1 Byte)	(2 Byte, big endian)	(2 Byte, big endian)

ET_SENSE_AIR (Brightness events; MultiSensor Air)

EventNumber	Sensor ID	Measurement value		
251 (1 Byte)	[1;3] (1 Byte)	(4 Byte, float, little		

7.3 Used Cipher Suite

IANA name: TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256
OpenSSL name: ECDHE-PSK-CHACHA20-POLY1305
GnuTLS name: TLS_ECDHE_PSK_CHACHA20_POLY1305
Hex code: 0xCC, 0xAC
TLS Version(s): TLS1.2
Protocol: Transport Layer Security (TLS)
Key Exchange: Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)
Authentication: Pre-Shared Key (PSK)
Encryption: ChaCha stream cipher and Poly1305 authenticator (CHACHA20 POLY1305)
Hash: Secure Hash Algorithm 256 (SHA256)
Included in RFC: RFC 7905

7.4 Ports

TCP 31460 for web interface
TCP 31461 for Gateway connection (configurable)
UDP 31462 for Gateway update
TCP 3000 for Grafana

8 Commands

8.1 Ping

The ping command is used to check the communication, if the LAN Gateway is connected and if the keys are correct. To ping the Gateway simply use the command ping without any parameters.

Example:

ping()

Example description:

Ping command will be sent and gateway will answer.



8.2 How to encode a B2L command

To encode a B2L command use *build_enc_frame*.

Command parameters:

<i>build_enc_frame</i> (<i>data</i> : bytearray,	#data to be sent
	<i>source_id</i> : int,	#who sent the package
	<i>destination_id</i> : int,	#package destination
	<i>packet_id</i> : int,	#ID to identify response frame
	<i>key</i> : UUID)	# the key to encrypt with

Range of values:

<i>data</i> :	command to be encoded (example: <i>set_fg_state</i>)
<i>source_id</i> :	0 to 0xFFFF
<i>destination_id</i> :	0 to 0xFFFF
	0 = broadcast
<i>packet_id</i> :	0 to 0xFF
<i>key</i> :	key as UUID

Example:

```
build_enc_frame(set_fg_state(...), 0, data['targetId'], 99, sys[1].net_key)))
```

Example description:

A frame with the command *set_fg_state* will be created. **Source ID is 0** and **target ID** will be added from an array. **Packet ID is 99** and the **key** will also be added from an array.

8.3 How to send a B2L frame

Use *config.send_queue.put* to send a B2L frame. *config.send_queue* is a Queue object, to send data over ethernet, to the LAN Gateway

Command parameters:

```
config.send_queue.put((socket_id, data))
```

Range of values:

<i>socket_id</i> :	The socket ID of the gateway to which the data should be sent
<i>data</i> :	the data to send

Example:

```
config.send_queue.put((sys[0], build_enc_frame(X)))
```

Example description:

The **encoded B2L frame** will be added to the **send queue**, with **socket ID** added from an array.

8.4 How to create a light control command

To create a light control command, *set_fg_state* is used. With *build_enc_frame* the frame gets encoded into a B2L command. Use *config.send_queue.put* to send the frame.

Command parameters:

<i>set_fg_state</i> (<i>FGNumber</i> : int,	#function group number
	<i>newState</i> : FGStates,	#state of function group
	<i>sceneNum</i> : int,	#scene number
	<i>lightLevel</i> : int,	#DALI light level
	<i>param</i> : int = 0)	#?



Range of values:

FGNumber: 0 to 15
FGStates: STATE_MANUAL = 0
STATE_AUTO_ACTIVE = 1
STATE_AUTO_PASSIVE = 2
STATE_AUTO_BASIC = 3
STATE_AUTO_OFF = 4
STATE_SEQUENCE = 5
STATE_KEEP_CURRENT = 255
sceneNum: 0 to 63
lightLevel: 0 to 254

Example:

```
set_fg_state(1, STATE_MANUAL, 2, 200)
```

Example description:

The light command will set **function group 1** to **DALI level 200** in **scene 2**. The selected state will be **manual**.

8.5 How to read the function group state

To read the function group state, `get_fg_state` is used. With `build_enc_frame` the frame gets encoded into a B2L command. Use `config.send_queue.put` to send the frame.

Command parameters:

`get_fg_state(FGNumber: int)` #function group number

Range of values:

FGNumber: 0 to 15

Example:

```
get_fg_state(FGNumber=0)
```

Example description:

Function group 0 will be read.

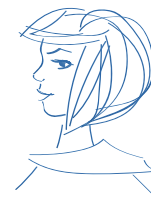
8.6 How to create a DALI tunnel

The DALI tunnel is used to send DALI commands through a B2L device directly to a DALI driver (for example to read out parameters). To create a DALI command use the function `dali_tunnel`. With `build_enc_frame` the frame gets encoded into a B2L command. Use `config.send_queue.put` to send the frame.

WARNING: Do not use the DALI tunnel to change any DALI parameters, because this may cause the B2L system to not work correctly.

Command parameters:

<code>dali_tunnel(dali_cmd: int,</code>	#DALI command (DALI standard)
<code> dtr0: int = None,</code>	#dtr0 register
<code> dtr1: int = None,</code>	#dtr1 register
<code> dtr2: int = None,</code>	#dtr2 register
<code> edtx: int = None,</code>	#enable device type
<code> dri_addr: int = 0xFE,</code>	#DALI address byte (DALI standard)
<code> repetition: int = 0,</code>	#send multiple times
<code> prio: int = 0,</code>	#priority
<code> answer: int = 0,</code>	#answer required
<code> repeat: int = 0)</code>	#command must be sent twice (DALI standard)



Range of values:

dali_cmd:	0 to 255 (0xFF)
dtr0:	0 to 255 (0xFF)
dtr1:	0 to 255 (0xFF)
dtr2:	0 to 255 (0xFF)
edtx:	0 to 255 (0xFF)
dri_addr:	0 to 255 (0xFF) (YAAA AAAS) Y: short or group, A: address bits, S: standard or DAPC
repetition:	0 to 64
prio:	0: high 1: low
answer:	0: no 1: yes
repeat:	0: no repetition (default) 1: repetition

Example:

dali_tunnel(dali_cmd=0xC5, dtr0=0, dtr1=202, dri_addr=0b0000 0011, repetition=1, answer=1)

Example description:

The **DALI command 0xC5** (READ MEMORY LOCATION) is send **twice**, with the parameters **DTR0=0** and **DTR1=202**, addressed with **short address 1**. An **answer is required**, and **no device type** is set. This example will read byte 0 and 1 of memory bank 202.

8.7 PMD commands

PMD (power metering and monitoring device) commands are used to monitor different parameters of DALI devices, like power or energy consumption. Therefore the DALI device has to support device type 49, 50, 51 and/or 52.

8.7.1 PMD initialization

The command *pmd_init_start* is used to search for all devices, which are compatible with PMD. Therefore, the list of all DALI devices in the B2L system must be added (4.8). This function will activate the *pmd_init_handler*, which will receive the answer.

Command parameters:

pmd_init_start() -> None

Example:

pmd_init_start()

Example description:

PMD will be initialized.

8.7.2 PMD retrieve

The command *pmd_retrieve_start* is used to read out the PMD parameters. This function will activate the *pmd_retrieve_handler*, which will receive the answer.

Command parameters:

pmd_retrieve_start() -> None

Example:

pmd_retrieve_start()

Example description:

All PMD parameters will be read.



8.7.3 PMD quit

The command `pmd_init_quit` is used, to abort a PMD search after timeout.

Command parameters:

`pmd_init_quit()` -> None

Example:

`pmd_init_quit()`

Example description:

The running PMD search will be aborted.

8.7.4 Using PMD with the Server-Demo

In the LAN Gateway server demo you can find a section named "Power measurement (only for drivers with PMD)".

Power measurement (only for drivers with PMD)

Enable power measurement ☒

Power measurement interval seconds (default: 30)

By pressing the button "Initialize PMD" the `pmd_init_start` function gets triggered. As mentioned before, therefore a B2L system has to be added. While the PMD initialization is running "`pmd_init: PMD initialization running...`" is displayed in the status section. When the initialization is done, the check mark for "Enable power measurement" can be set. By enabling the power measurement, every 30 seconds the current PMD parameters will be read from the compatible drivers. This interval can be changed in the "Power measurement interval" text field. After changing the configuration click "Save configuration and restart server".

8.7.5 Adding the power measurements to Grafana

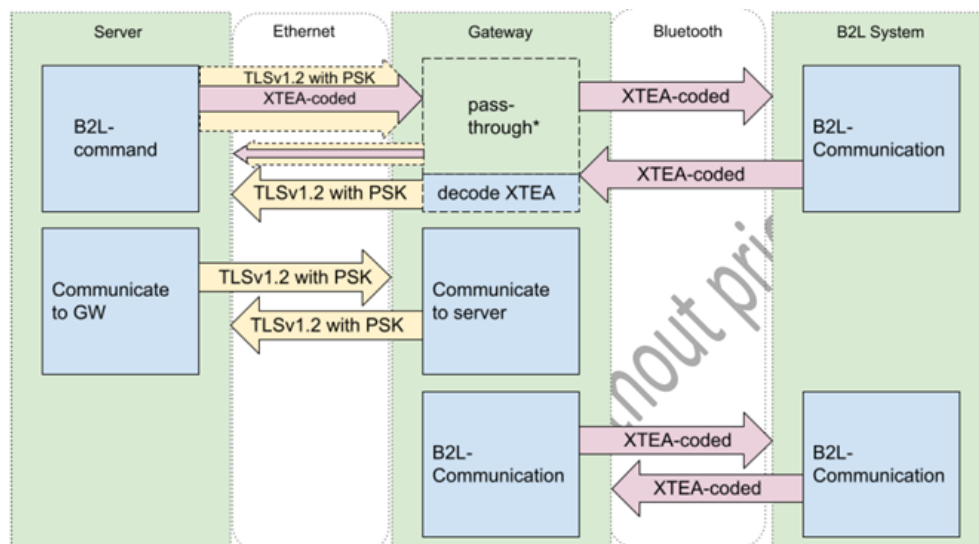
On every new start of Grafana, it will be checked, if any PMD devices are initialized. If this is the case, it will automatically be shown, in the Grafana dashboard.





8.8 B2L Encryption method

The data transfer in the B2L mesh is encrypted with XTEA (eXtended Tiny Encryption Algorithm). To ensure, that the B2L network is safe, the commands for the B2L network must be sent XTEA encrypted, from the server.



*Answers to server commands will get passed through to the server. B2L-events will get decoded in the gateway.

9 Where to pull data

If you need data from your B2L system, there are several different ways to get them:

9.1 Read database (recommended)

Movement events from the different Blu2Light MultiSensors, as well as measurements from the "Blu2Light MultiSensor AIR" are directly stored into the database and can be read from there. If PMD is initialized (8.7), this data will also be stored to the database.

9.2 When written to database

Another way to get data is by modifying a function, that writes to the database. For example, a modified DBHelper based on the functions *MariaDBHelper* (*maria_db.py*) or *InfluxDBHelper* (*influx_db.py*) can be created for that. Which DBHelper will be used can be modified in *_main_.py*.

9.3 Retrieve events

The third possible way to get data, is in the event handler. Therefore, the function *parse_event* (*b2l_parser.py*) can be modified to also pull data.

9.4 Using B2L commands

The fourth possible way is by using B2L commands. How to create B2L events can be found in 8.2.



10 Beaconsing

With the Beaconsing functionality of all Blu2Light nodes, including LAN Gateway, the user can configure advertising messages, up to 31 Bytes, which will be sent out periodically. This can be a URL for example. Common profiles for Bluetooth Beacons are iBeacon by Apple or Eddystone by Google. But also custom advertising messages are supported by Blu2Light, as long as these are below 31 Bytes long.

For more information, read these documentations below provided by our Partner M-Way:

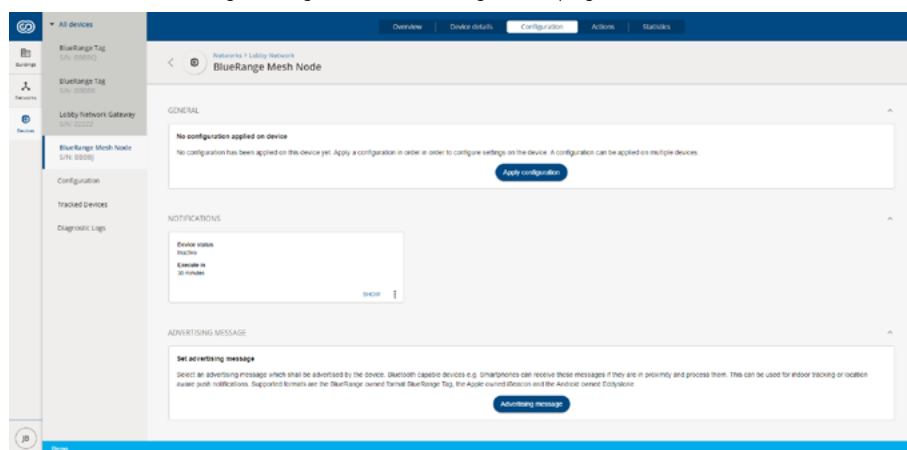
Overview

In the early days, when our Mesh Nodes were still called SmartBeacons, we were one of the first companies to provide a platform for managing and monitoring a huge number of beacons. We have kept these possibilities but nowadays, this is only a small subset of what our platform can do. It allows you to remotely configure beaconing messages such as iBeacon, Eddystone or our custom BlueRange Tag message.

Note: Only BlueRange Mesh Nodes (most of them) support this feature and you cannot assign beaconing messages to BlueRange Tags as they are usually running with small batteries.

Configuration

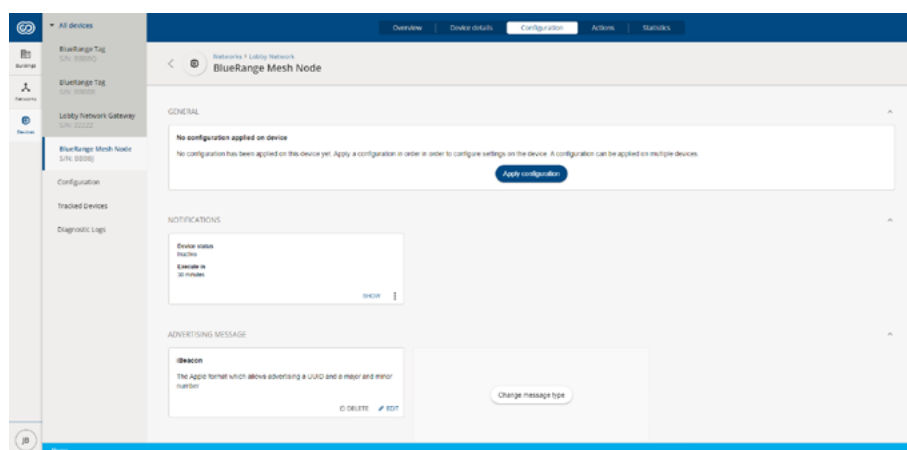
You can add beaconing messages from the Configuration page of a Mesh Node.



Once you choose to add a beaconing message, you will be prompted for the kind of message that you want to assign. Continue to read below depending on the message type you want to use.

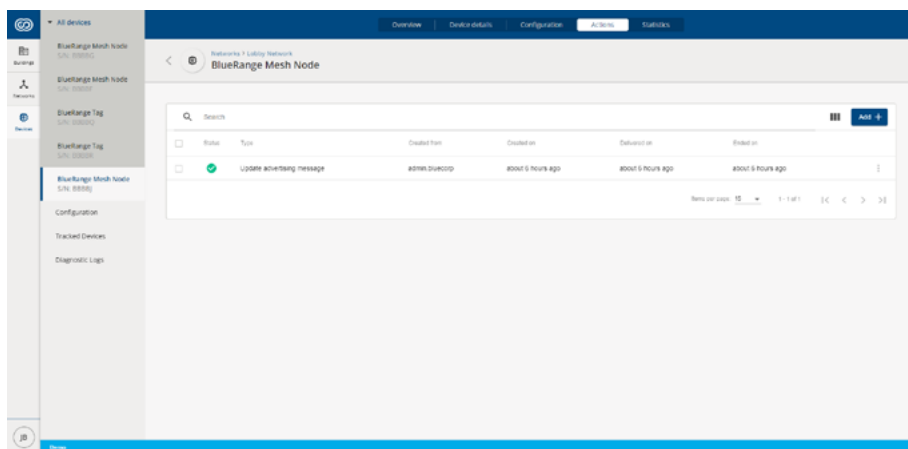
Tip: We are currently only supporting a single beacon message per mesh node, so make sure to not select a mesh node that already has a message configured, otherwise you need to remove that message first before being able to assign a new one.

You are able to see the currently configured Beaconing Messages if you open one of your mesh nodes and go to the Configuration page.





You can always check the status of the message if you go to the Actions page. For each change of the advertising message, an Update Advertising Message action will be created. Once this has been marked as complete, the device has successfully accepted the beaconing message and is now broadcasting it.



iBeacon Messages

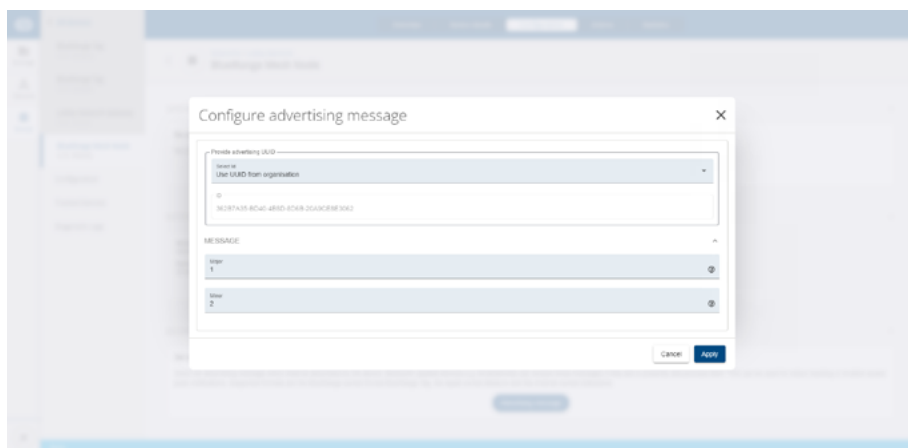
The iBeacon format consists of a 128 bit UUID, a 2 byte identifier called major and another 2 byte identifier called minor. As the UUID is a really long number with 5 undecillion possible values, it is almost impossible to randomly pick a number that has been used by somebody else on earth. It is therefore possible to randomly generate it with the (almost) guarantee that it is unique. When putting iBeacons in multiple buildings, one option is to use the major identifier to specify the building and the minor identifier to have a unique id for each beacon in that building.

With BlueRange you can either specify these values yourself or you can choose to use the platform generated values for either your organization or your network. If you decide to leave the major and minor empty, these will also be automatically assigned based on the UUID you selected.

Here are the rules for automatic generation:

- **UUID:** If you do not change the default UUID, the UUID of your organization will be used.
- **Major:** Each of your networks has a fixed id when you create it. This id will be used for all mesh nodes, if you leave the field empty.
- **Minor:** Each of your mesh nodes has a unique node id in its network. Each mesh node will use this unique id if you do not specify another value.

After assigning the message(s) the values will be generated for all mesh nodes and will be stored. They will not change if you do not delete or change them manually.

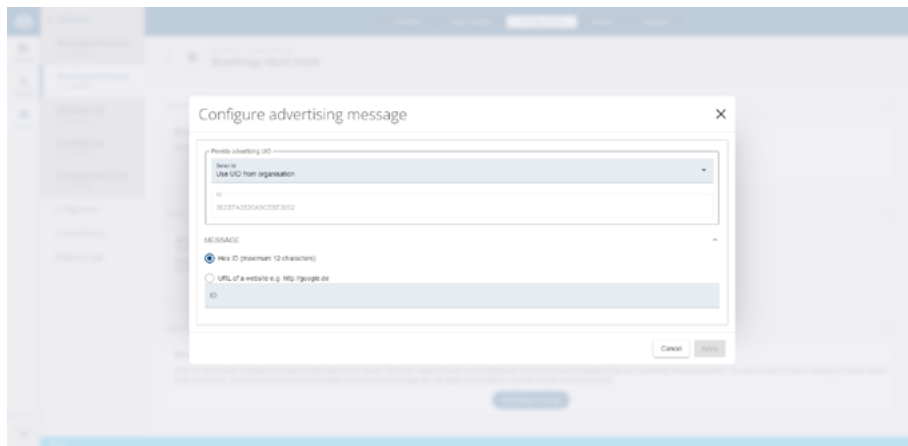




Eddystone UID Message

The Eddystone UID format is similar to the iBeacon format and also tries to specify a format that uniquely identifies a beacon. Similar to the iBeacon message, you can either specify all values or let the platform generate them. Here are the rules:

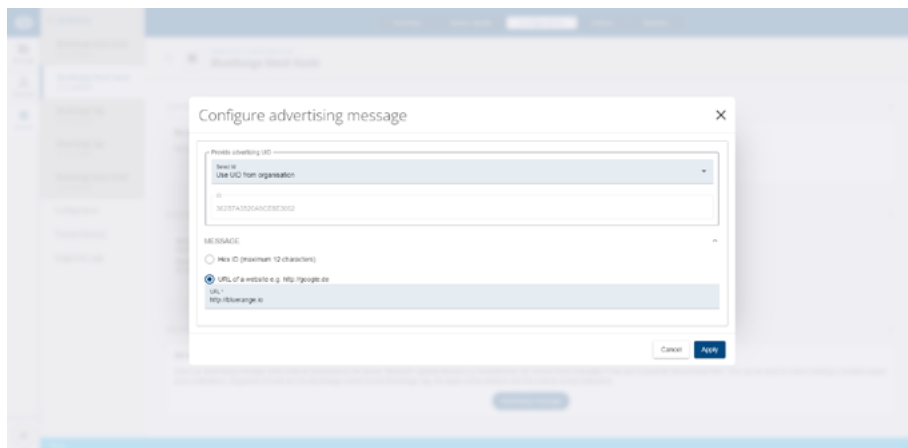
- **UID:** The UID is similar to a UUID but shorter. It is still big enough so that it is very unlikely that you would randomly pick the same UID as another person. Leave the field prefilled to use the UID generated from your organization UUID.
- **Identifier:** The identifier consists of 6 bytes that you can freely assign to contain either a mesh node id, a building id, product id or a combination of these. If you leave it empty, it will be generated from the network id and node id automatically and will be unique for each mesh node.



Eddystone URL Message

The Eddystone URL message can be used to let one or many mesh nodes advertise an internet address. The URL is encoded in a special format to make it as short as possible. You can of course also use an URL shortener and use the shortened URL in the beaconing message.

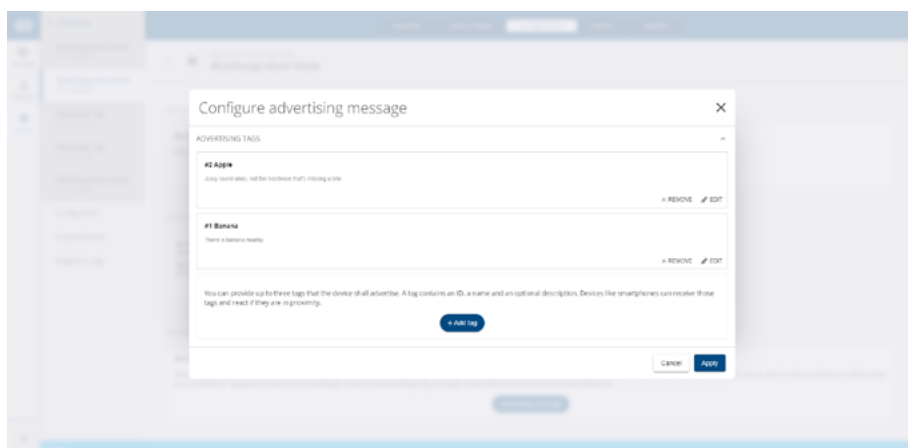
- **URL:** Enter the URL of the page that you want to advertise, this is used for all selected mesh nodes.



BlueRange Tag Messages

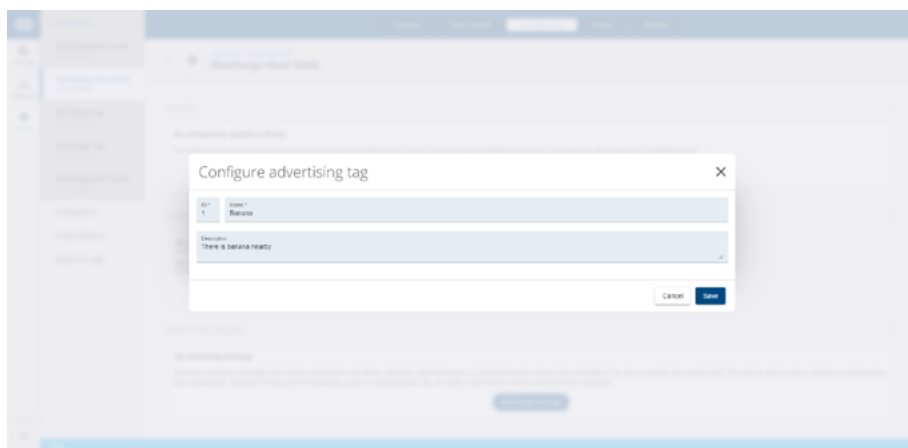
BlueRange Tag messages are a nifty feature to create smartphone applications that do not need internet access while still being able to re-configure the type of products advertised by a mesh node. The tags can be either hardcoded in the smartphone app or they can be retrieved from our REST endpoints and can be cached in the app. By being able to reconfigure the tags for all your mesh nodes, you can always remotely reconfigure everything without updating the smartphone app or relying on the internet connection of the user.

Once you decide to apply a BlueRange Tag message, you will have the ability to create and select a number of advertising tags.



Through the Add Tag button, you will be able to select all previously created advertising tags or create new ones. The tags can be used organization wide for all of your devices once you apply the BlueRange tag message.

Creating a new advertising tag requires a name and an id. This id will then be stored in the final beaconing message and can be picked up by any beacon scanning application. In this example, the beaconing message would state that there are apples and bananas nearby.



After your tag was created, you can add it to the message. Up to three tags can be selected for each Mesh Node. You can always change the tags for any of the mesh nodes and the BlueRange Tag Message will be updated accordingly.

Purpose

The BeaconsModule with ModuleId 1 allows the user to configure a number of advertising/beaconing messages to be sent periodically. It is possible to send iBeacon messages, Eddystone messages and other custom advertising messages without the need to update the firmware. Using FruityMesh in conjunction with a MeshGateway, a fully managed beacon infrastructure can be established.

The BeaconsModule is only intended for beaconing messages that can subsequently be configured over the mesh. Take a look at the AdvertisingController documentation for how to schedule advertising messages in your implementation.

Functionality

The BeaconsModule is currently configured by setting its persistent configuration. At the moment, it allows one message to be broadcast at a time and doesn't cycle through different messages. Message cycling can however be easily implemented since the mesh framework already supports this.

Add an advertising job

action [nodeld] adv add [Hex/Base64-String up to 31 byte] {requestHandle}

The node with the given nodeld will advertise the given advertising message on success together with all other advertising messages that were either configured through a different module or through the BeaconsModule. Note that only valid broadcast messages are actually advertised. Others are rejected by our HAL.



Example (adding an iBeacon message)

action 123 adv add 02:01:06:1A:FF:4C:00:02:15:F0:01:8B:9B:75:09:4C:31:A9:05:1A:27:D3:9C:00:3C:EA:60:00:32:81:00 13

The response acknowledges the receipt with this command:

```
{
  "type": "adv_add_response",
  "nodeId": 123,
  "module": 1,
  "requestHandle": 13,
  "code": 0
}
```

where "code" can have the following values:

Code	Name	Description
0	SUCCESS	The node successfully advertises the data and stored it in its persistent storage.
1	FULL	Too many advertisings have been stored in the node. To clear the advertisings you can reenroll the node.
2	RECORD_STORAGE_ERROR	The node was unable to persist the adv data. It is advertising it only until the next reboot.

Set an advertising job

action [nodeId] adv set [slot] [Hex/Base64-String up to 31 byte] {requestHandle}

The node with the given nodeId will advertise the given advertising message on success together with all other advertising messages that were either configured through a different module or through the BeaconsingModule. Note that only valid broadcast messages are actually advertised. Others are rejected by our HAL.

Example (setting an iBeacon message)

action 123 adv set 0 02:01:06:1A:FF:4C:00:02:15:F0:01:8B:9B:75:09:4C:31:A9:05:1A:27:D3:9C:00:3C:EA:60:00:32:81:00 13

The response acknowledges the receipt with this command:

```
{
  "type": "adv_set_response",
  "nodeId": 123,
  "module": 1,
  "requestHandle": 13,
  "code": 0
}
```

where "code" can have the following values:

Code	Name	Description
0	SUCCESS	The node successfully advertises the data and stored it in its persistent storage.
1	SLOT_OUT_OF_RANGE	The given slot is outside the range of possible slots.
2	RECORD_STORAGE_ERROR	The node was unable to persist the adv data. It is advertising it only until the next reboot.

Remove an advertising job

action [nodeId] adv remove [slot] {requestHandle}

Clears an advertisement slot. The node with the given nodeId will no longer advertise data in this slot. If this slot is already cleared, the command silently succeeds.

Example

action 123 adv remove 0 13

The response acknowledges the receipt with this command:

```
{
  "type": "adv_set_response",
  "nodeId": 123,
  "module": 1,
  "requestHandle": 13,
  "code": 0
}
```



where "code" can have the following values:

Code	Name	Description
0	SUCCESS	The node successfully advertises the data and stored it in its persistent storage.
1	SLOT_OUT_OF_RANGE	The given slot is outside the range of possible slots.
2	RECORD_STORAGE_ERROR	The node was unable to persist the adv data. It will continue to advertise the old message after a reboot.

Link to the M-Way Informations:
<https://www.bluerange.io/docs/fruitymesh/BeaconingModule.html>

10.1 How to set up a Beacon message in our Web-UI

The easiest way to set up a beaconing/advertising message is the Gateway Demo Software.
This can be found in "System overview and light control":



Insert the Beacon Message and the click "Set Beacon", to write the message to the node.
Click "Remove Beacon", if you want to delete the Beacon Message from the node.
The Beacon Message can only be set or deleted, but not read.
There are 2 example Beacon messages selectable by pressing the triangle sign at the text box.

10.2 How to set up a Beacon message using the Web-API

A beacon can be set or removed via Web-API by using `/api/light_control` and the commands `sb` (set beacon) or `rb` (remove beacon). More information can be found in chapter 11.1.13.



Additional information

11.1 WEB-API calls

All API calls are handled in webconfig.py. In this file additional calls should be added.

11.1.1 /api/get_status

Read out active connections and how many packages were received.

Parameter: none

Output: JSON: connections, received_packets (since) f.e.: {"connections": ["XX:XX:XX:XX:XX:XX"],
"received_packets": "399 since 2022-07-26 17:17:07"}

11.1.2 /api/pmd_init

(login required)

Initialize and start PMD (see 8.7).

Parameter: none

Output: none (or error)

11.1.3 /api/get_config

Read out config data.

Parameter: system: setting systems mac adress f.e.: /api/get_config?system= XX:XX:XX:XX:XX:XX

Output:

- Pre-shared key for connection between server and Gateway (see 4.3)
- Config of Maria Database (enabled, host, port, username, password, dbname)
- Config of Influx Database (enabled, host, port, username, password, dbname)
- Config of PMD (enabled, interval)

Example (out): {"mariadb": {"enabled": true, "port": "", "username": "pi", "password": "pwd123", "dbname": "maria", "host": ""},
"web": {"username": "pi", "password": "pbkdf2:sha256:260000\$i3lanlaCOSO5tLzD\$96198848c7b55225a2344924b70c6cb39826b73c7e0b19b12845bf3a013954b"}, "influxdb": {"enabled": false, "port": "", "username": "", "password": "", "dbname": "", "host": ""}, "psk": "A5A387DADD5B219E51ECF65B1973A8295F9430C186C9EB006375C5656CE32701", "system": ["6C:4B:7F:01:02:43"], "pmd": {"enabled": false, "interval": ""}}

11.1.4 /api/get_rtc_time

Read out RTC time of Gateway.

Parameter: system: setting systems mac adress f.e.: /api/get_config?system= XX:XX:XX:XX:XX:XX

Output: RTC Time f.e.: {"msgType": "GetRTCTime", "time": "2022-07-27 16:56:26"}

11.1.5 /api/set_rtc_time

Set RTC Time. Uses local time of the server.

Parameter: system: setting systems mac adress f.e.: /api/get_config?system= XX:XX:XX:XX:XX:XX

Output: none (or error)



11.1.6 /api/set_config

(login required)

Set all Configurations (f.e. PSK, port and database) and restarts the server software.

Methods: POST

Parameter:

- csrf - CSRF-Token
- psk - Pre-shared key for connection between server and Gateway (see 4.3)
- port - Server port
- mariadb - Config Maria Database (enabled, host, port, username, password, dbname)
- influxdb - Config Influx Database (enabled, host, port, username, password, dbname)
- pmd - Config PMD (enabled, interval)

Example:

```
data = {
  port: 31461,                                     (default: 31461)
  psk: A5A387DADD5B219E51ECF65B1973A8295F9430C186C9EB006375C5656CE32701,
  mariadb: {
    enabled: true,
    port: 3306,                                     (default: 3306)
    username: pi,
    password: pwd123,
    dbname: maria,
    host: 127.0.0.1                                 (default: localhost)
  },
  influxdb: {
    enabled: false,
    port: ,                                         (default: 8086)
    username: ,
    password: ,
    dbname: ,
    host:                                           (default: localhost)
  }
};
data.pmd = {
  enabled: true,
  interval: 30                                     (default: 30)
};

fetch('/api/light_control?csrf=' + document.getElementById('csrf-token').value, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
});
```

Output: none (or error)

11.1.7 /api/delete_system

(login required)

Delete a system.

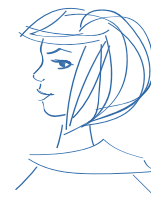
Methods: GET

Parameter: csrf,

System: setting systems mac adress

f.e.: /api/delete_system?csrf=XXXX&system= XX:XX:XX:XX:XX:XX

Output: none (or error)



11.1.8 /api/get_errors

Gets errors from Server.

Parameter: none

Output: list of errors, f.e.: {"mariadb": null, "server": null}

11.1.9 /api/get_messages

(login required)

Get the list of messages for the web interface.

Methods: GET

Parameter: none

Output: list of messages, if no message available: []

11.1.10 /api/get_update_status

Get the status of a running update.

Parameter: none

Output: update status or none if no update running f.e.: *Transmitting frame 42 of 69...*

11.1.11 /api/upload

(login required)

Upload a *.b2lssystem file of the connected B2L system to get the connected devices.

Methods: POST

Parameter: csrf,
file (*.b2lssystem),
macaddr

Example:

```
let b2lssystem = document.getElementById("b2lssystem-file").files[0];
let formData = new FormData();

formData.append("file", b2lssystem);
formData.append("csrf", document.getElementById('csrf-token').value);
formData.append("macaddr", document.querySelector('macaddress').value);
fetch('/api/upload', {method: "POST", body: formData});
```

Output: none (or error)

11.1.12 /api/update

(login required)

Upload an update file for the Gateway (.bin).

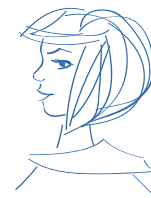
Methods: POST

Parameter: csrf,
file (*.bin),
macaddr

Example:

```
let update = document.getElementById("update-file").files[0];
let formData = new FormData();

formData.append("file", update);
formData.append("csrf", document.getElementById('csrf-token').value);
formData.append("macaddr", document.querySelector('macaddress').value);
fetch('/api/upload', {method: "POST", body: formData});
```

Output: none (or error)

11.1.13 /api/light_control

(login required)

Light control API

Methods: GET - Get values (default)
POST - Set values

Parameter (GET): csrf - CSRF-Token
syst - System (str)
node - Selected node (int)
lum - Selected functional group (FG) of the node (int)
cmd - Command (int) (see below)

Commands (GET): 41 - get_version_init
80 - get_fg_state_init
89 - get_gps_position_init

Example (GET): `fetch('/api/light_control?csrf=' + document.getElementById('csrf-token').value + '&system=XX:XX:XX:XX:XX' + '&node=' + ids[2] + '&lum=' + ids[3] + '&cmd=80')`

Parameter (POST): csrf - CSRF-Token
Data - Command (sfgs, eu, sb, rb) and parameter (see below)

Data (sfgs): command - sfgs (set FG state), eu (energy update), sb(set beacon), rb (remove beacon)
(see 8.2 and 8.4) lightLevel - DALI value (int)
FGNumber - Selected function group (int)
newState - New FG-state (manual: 0, auto (active: 1/passive: 2/basic: 3/off: 4), keep current state: 255) (int)
sceneNum - Selected Scene Number
targetId - ID of the Target Node (also named destination_id) (int)
system - Selected System (str)

Example (sfgs):

```
data = {
  command: 'sfgs',
  system: ids[1],
  targetId: Number(ids[2]),
  FGNumber: Number(ids[3]),
  newState: Number(state),
  sceneNum: Number(scene),
  lightLevel: Number(lum),
};
fetch('/api/light_control?csrf=' + document.getElementById('csrf-token').value, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
});
```

Data (eu): command - eu (Energy Update / Set Emergency)
energyUpdate - Energy Update Value (signed 8-bit value, range -100 to 100, unit: %, negative value will increase brightness, positive will reduce)
system - Selected System (str)

Example (eu):

```
data = {
  command: 'eu',
};
```



```
        system:                ids[1],
        energyUpdate:          Number(eu)
    };
    fetch('/api/light_control?csrf=' + document.getElementById('csrf-token').value, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(data),
    });
```

Data (sb):

command	- sb (set beacon)
slot	- slot is a parameter to send more than one advertising message per Node. Currently only one slot (0) is available
beacon_msg	- The message that will be sent by the Node (see 10)
targetId	- ID of the Target Node (also named destination_id) (int)
system	- Selected System (str)

Example (sb):

```
data = {
    command:                'sb',
    system:                  ids[1],
    targetId:                Number(ids[2]),
    slot:                    0,
    beacon_msg:              '02:01:06:03:03:AA:FE:0D:16:AA:FE:10:00:01:67:6F:6F:67:6C:65:00'
};
fetch('/api/light_control?csrf=' + document.getElementById('csrf-token').value, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify(data),
});
```

Data (rb):

command	- rb (remove beacon)
slot	- slot is a parameter for more than one advertising message per Node. Currently only one slot (0) is available
targetId	- ID of the Target Node (also named destination_id) (int)
system	- Selected System (str)

Example (rb):

```
data = {
    command:                'rb',
    system:                  ids[1],
    targetId:                Number(ids[2]),
    slot:                    0,
};
fetch('/api/light_control?csrf=' + document.getElementById('csrf-token').value, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify(data),
});
```



12 Troubleshooting

12.1 Bad CSRF token

The CSRF token is stored in the client browser and on the server. It's needed to authorize the communication between the web interface and the server. "*Bad CSRF Token*" means that the token stored in the client browser is outdated.

How to fix: Refresh the web page (f.e. press "F5"-key)

12.2 LED 1 red

The red LED shows, that the Gateway doesn't have an IP and something is wrong with the network connection.

How to fix: Check the ethernet cable and check the router settings.