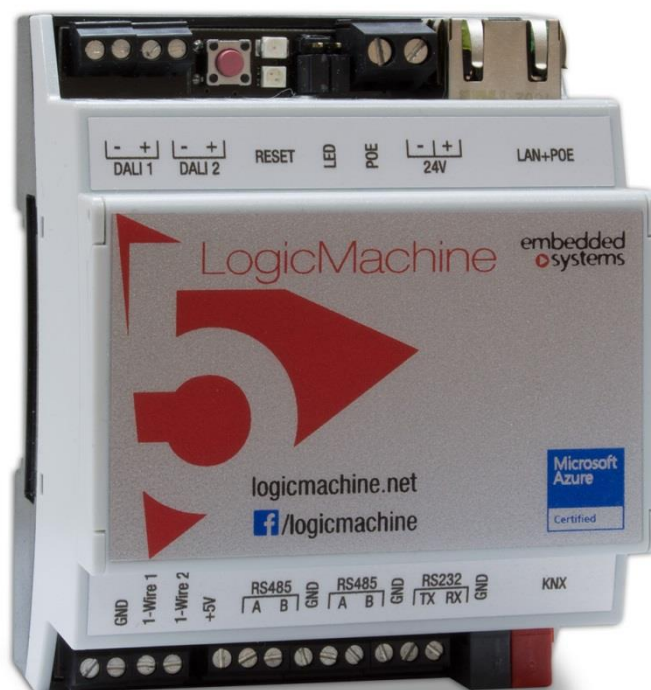


LogicMachine5 Power (LM5p-DW1)



Product Manual

Document Issue 1.0
December, 2016

Technical Support:
support@openrb.com

Copyright

Copyright © 2016 Embedded Systems SIA. All Rights Reserved.

Notice

Embedded Systems SIA., reserves the right to modify the information contained herein as necessary. Embedded Systems SIA assumes no responsibility for any errors which may appear in this document. Information in this document is provided solely to enable system and software implementers to use KNX/EIB LogicMachine product.

Trademarks

LogicMachine is a trademark of Embedded Systems SIA. All other names and trademarks are the property of their respective owners and are hereby acknowledged.

Introduction

LogicMachine (LM) is your easiest way to program complex logic in KNX/EIB, Modbus, BACnet, DALI, 1-wire networks. LM will enable you to efficiently customize building automation processes, easily delivering unlimited flexibility benefit to end users in a cost-effective way.

LM5 Power is an embedded platform with integrated Ethernet, USB, KNX/EIB, DALI, 1-wire, Serial interfaces. LM allows to use it as cross-standard gateway, logic engine, visualization platform, IP Router. Scripting templates provides user-friendly, flexible configuration interface and integration with cloud/web services, 3rd party devices. Via applying custom scripts LM can simultaneously act as thermostat, security panel, lighting controller, etc. LogicMachine application store and external app development possibility allows to extend device functionality and adjust to a specific market segment

LM5 Power has Power-over-Ethernet support. Further, LM5 Power is 3x more powerful than all previous LogicMachine versions due to more powerful CPU and RAM.

Technical support

Any faulty devices should be returned to Embedded Systems.

If there are any further technical questions concerning the product please contact our support, available Mon-Fri 9:00 – 17:00 GMT +02:00. Please write to support@openrb.com.

Firmware updates are available at www.openrb.com



Caution Security advice

The installation and assembly of electrical equipment may only be performed by skilled electrician. The devices must not be used in any relation with equipment that supports, directly or indirectly, human health or life or with application that can result danger of people, animals or real value

Mounting advice

The devices are supplied in operational status. The cables connections included can be clamped to the housing if required.

Electrical connection

The devices are constructed for the operation of protective low voltage (SELV). Grounding of device is not needed. When switching the power supply on or off, power surges must be avoided.

Contents

DEVICE SPECIFICATION	7
1. TERMINAL CONNECTION SCHEMES	9
2. STANDARDS SUPPORTED	16
3. QUICK STARTUP GUIDE	18
3.1. CONNECTION	18
3.2. DEFAULT LOGIN PARAMETERS	18
3.3. FACTORY DEFAULT	18
3.4. IP SETTINGS	18
3.5. DISCOVER LOGICMACHINE IP ADDRESS	20
3.6. FIRMWARE UPGRADE	22
3.7. LOGICMACHINE FOR KNX/EIB NETWORK CONFIGURATION MANAGEMENT WITH ETS	23
3.8. KNX AND IP ROUTER SETTINGS	24
3.9. QUICK GUIDE - MOSAIC APPLICATION FOR EASY VISUALIZATION	28
3.10. QUICK GUIDE - CREATE VISUALIZATION FOR IPAD/PC	32
4. GRAPHICAL USER INTERFACE LOGIN	41
4.1. <i>Customize background / Language</i>	42
4.2. <i>Find applications</i>	43
4.3. <i>Unlock the screen for sorting order and hiding apps</i>	44
4.4. <i>Admin mode: adding/removing/administering apps</i>	45
5. APPLICATION DEVELOPMENT	48
6. LOGICMACHINE CONFIGURATION	59
6.1. SCRIPTING.....	60
6.1.1. <i>Block programming</i>	60
6.1.2. <i>Block functions</i>	62
6.1.3. <i>Adding a new script</i>	64
6.1.4. <i>Event-based scripting</i>	66
6.1.5. <i>Resident scripting</i>	67
6.1.6. <i>Scheduled scripting</i>	67
6.1.7. <i>Script editor</i>	68
6.1.8. <i>Object functions</i>	69
6.1.9. <i>Returned object functions, group communication functions</i>	70
6.1.10. <i>Group communication functions</i>	71
6.1.11. <i>Object function examples</i>	71
6.1.12. <i>Data type functions, data types</i>	72
6.1.13. <i>Data types</i>	72
6.1.14. <i>Data storage function</i>	73
6.1.15. <i>Alert function</i>	74
6.1.16. <i>Log function</i>	75
6.1.17. <i>Scheduled scripting date/time format</i>	75
6.1.18. <i>Time function</i>	75
6.1.19. <i>Data Serialization</i>	76
6.1.20. <i>String functions</i>	76
6.1.21. <i>Input and output functions</i>	80
6.1.22. <i>Script control functions</i>	81
6.1.23. <i>JSON library</i>	81
6.1.24. <i>Conversion</i>	82
6.1.25. <i>Bit operators</i>	82
6.1.26. <i>Input and Output Facilities</i>	83
6.1.27. <i>Mathematical functions</i>	84
6.1.28. <i>Table manipulations</i>	86
6.1.29. <i>Operating system facilities</i>	87
6.1.30. <i>Extended function library</i>	89
6.1.31. <i>User libraries</i>	90
6.1.32. <i>Common functions</i>	91
6.1.33. <i>Start-up (init) script</i>	91
6.1.34. <i>Tools</i>	92

6.2.	OBJECTS.....	94
6.2.1.	Object parameters	94
6.2.2.	RGB group object.....	95
6.2.3.	Object visualization parameters.....	98
6.2.4.	Change the object state.....	102
6.2.6.	Object control bar	102
6.2.7.	Filter objects.....	104
6.3.	OBJECT LOGS	105
6.3.1.	Export logs.....	106
6.4.	SCHEDULERS.....	108
6.4.1.	Add new scheduler.....	108
6.4.2.	Scheduler events	109
6.4.3.	Scheduler holidays.....	109
6.4.4.	Direct link.....	110
6.5.	TREND LOGS.....	110
6.5.1.	Add new trend log.....	111
6.5.2.	Direct link.....	111
6.5.3.	Trend logs functions	112
6.6.	SCENES	113
6.7.	VISUALIZATION STRUCTURE	114
6.7.1.	Levels / Plans.....	115
6.7.2.	Layouts / Widgets.....	117
6.8.	VISUALIZATION.....	121
6.8.1.	Plan editor.....	121
6.8.2.	Object	122
6.8.3.	Link.....	124
6.8.4.	Text Label.....	125
6.8.5.	Image.....	126
6.8.6.	Frame	127
6.8.7.	Gauge	129
6.8.8.	Camera	129
6.8.9.	Graph.....	131
6.9.	VIS.GRAPHICS	133
6.10.	UTILITIES	135
6.11.	USER ACCESS	139
6.12.	ALERTS	142
6.13.	ERROR LOG	143
6.14.	LOGS	143
7.	USER MODE VISUALIZATION	144
7.1.	CUSTOM DESIGN USERMODE VISUALIZATION	145
8.	TOUCH VISUALIZATION	146
9.	SYSTEM CONFIGURATION	147
9.1.	HOSTNAME.....	147
9.2.	CHANGING ADMIN PASSWORD	148
9.3.	PACKAGES.....	148
9.4.	UPGRADE FIRMWARE	148
9.5.	REBOOT LOGIC MACHINE	149
9.6.	SHUTDOWN LOGIC MACHINE.....	149
9.7.	INTERFACE CONFIGURATION	149
9.8.	KNX CONNECTION.....	151
9.9.	KNX STATISTICS.....	155
9.10.	BACNET SETTINGS.....	155
9.11.	BACNET OBJECTS	157
9.12.	HTTP SERVER.....	158
9.13.	FTP SERVER.....	158
9.14.	REMOTE SERVICES	159
9.15.	SYSTEM MONITORING	161
9.16.	REMOTE DIAGNOSTICS	162
9.17.	NTP CLIENT	162
9.18.	SYSTEM STATUS	163

9.19.	NETWORK UTILITIES	164
9.20.	SYSTEM LOG	164
9.21.	RUNNING PROCESSES	165
10.	USER MODE SCHEDULERS	166
10.1.	EVENTS	166
10.2.	HOLIDAYS	167
11.	TREND LOGS	168
12.	MODBUS RTU/TCP INTERCONNECTION WITH LM.....	171
12.1.	MODBUS DEVICE PROFILE	171
12.2.	READING MODBUS RTU COIL / REGISTER FROM THE INTERFACE.....	173
12.3.	RTU SCAN	173
12.4.	RTU SETTINGS	174
12.5.	ADDING MODBUS DEVICE	175
12.6.	PROGRAM ADDRESS FOR UIO20 MODBUS DEVICE	176
12.7.	MODBUS SLAVE EXAMPLES	176
13.	BACNET IP INTERCONNECTION WITH LM	183
13.1.	BACNET SERVER MODE: TRANSPARENT DATA TRANSFER TO BACNET NETWORK	183
13.2.	BACNET CLIENT MODE	184
14.	DALI CONFIGURATION.....	187
14.1.	DALI OBJECT MAPPING.....	188
14.2.	ACCESS DALI BUS FROM SCRIPTS.....	188
15.	DMX INTERCONNECTION WITH LM	196
16.	1-WIRE CONFIGURATION	202
17.	3G MODEM CONNECTION WITH LM	205
17.1.	EXAMPLES	207
17.2.	SEND SMS MESSAGES TO SPECIFIC SIM NUMBERS AFTER GROUP-READ OR GROUP-WRITE IS TRIGGERED 208	208
17.3.	SEND SMS MESSAGES WITHOUT 3G MODEM.....	208
18.	COMMUNICATION WITH RS232/RS485 SERIAL PORTS	210
19.	BLUETOOTH 4.0 INTEGRATION	213
21.	SIP SERVER ON LOGICMACHINE	216
22.	OBJECT VALUE EXPORT VIA XML	218
23.	ALERTS, ERRORS VALUES.....	220
24.	READ ALERTS RSS FEEDS FROM LOGICMACHINE	221
25.	OTHER EXAMPLES.....	222

Device specification

Types of product

LogiMachine5 Power LM5p-DW1

Standards and norms compliance

EMC: EN61000-6-1

EN61000-6-3

PCT Certificate

Technical data:

Power supply: 12V-30V DC on terminal connectors or
12V-30V DC Passive Power-over-Ethernet

Power consumption: 1.3W

Interface:

KNX/EIB TP1	1
10BaseT/100BaseTX	1
RS-485	1
RS-485/RS-232	1
(switchable in software – full-duplex=RS232, half-duplex=RS485)	
DALI master	2 (up to DALI 128 ballasts in total)
1-wire	2

Connections:

KNX bus	Bus Connection Terminal 0.8mm ²
Power supply	Screw, 1.5 mm ²
Serial	Screw, 1 mm ²
DALI	Screw, 1 mm ²
1-wire	Screw, 1 mm ²

Operating elements

LED	1 – CPU load 1 - Activity
-----	------------------------------

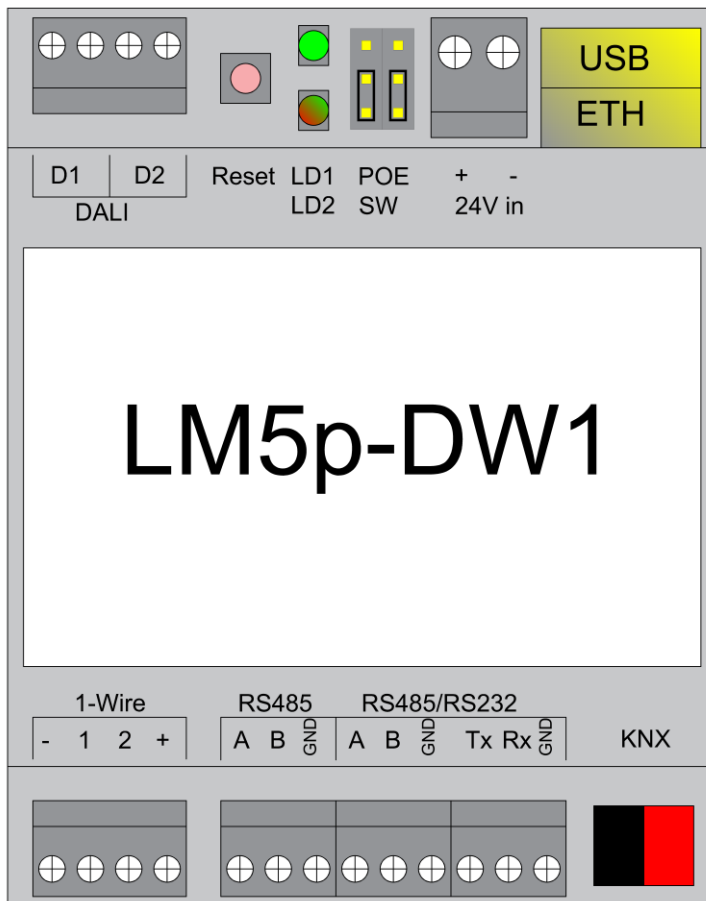
Enclosure:

Material:	Polyamide
Color:	Gray
Dimensions:	71(W)x90(H)x61(L) mm

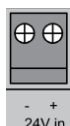
Usage temperature:	0C ... +45C
Storage temperature:	-15C ... +55C
Net Weight:	119g
Gross Weight:	137g
Warranty:	2 years
Relative Humidity:	10...95 % without condensation

LogicMachine5 Power contains:

- Embedded board with preinstalled software
- Plastic DIN-rail case

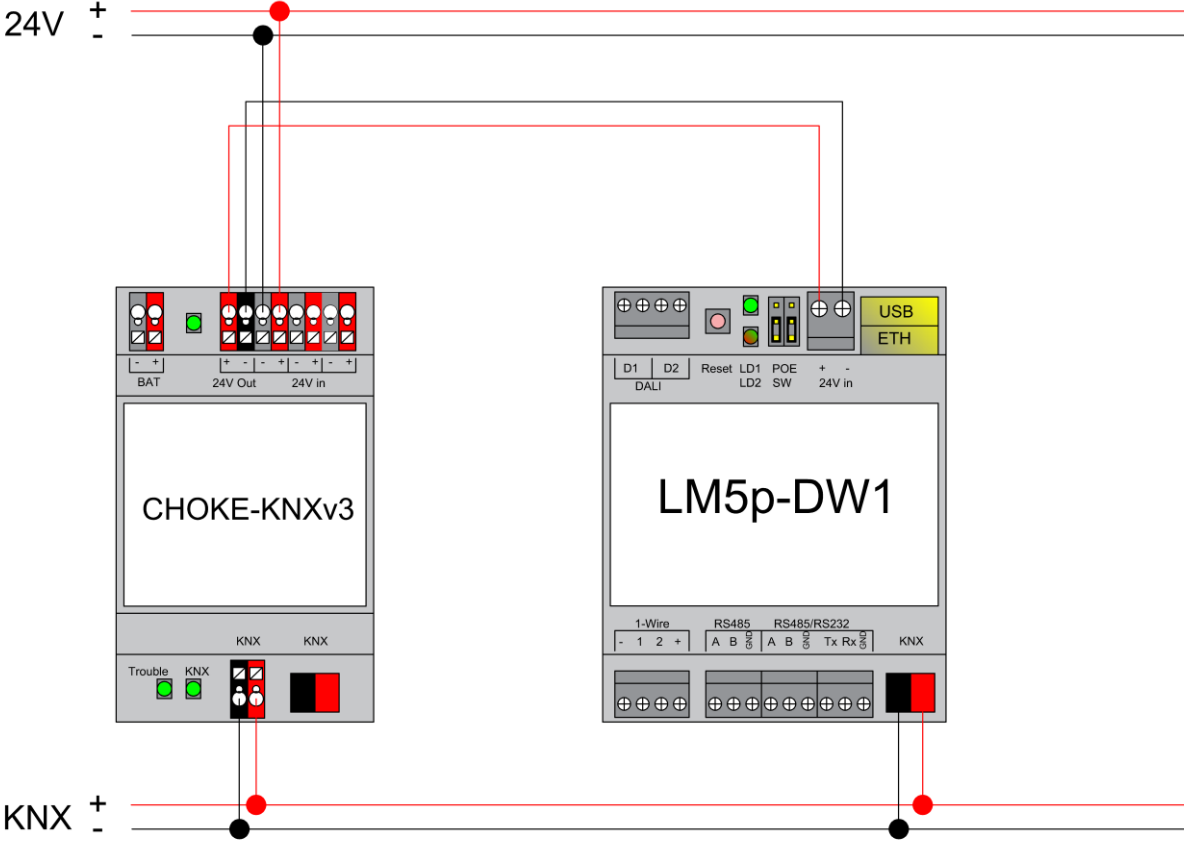


Note! In first batch of LM5 devices, the 24V DC clamp powering ports are placed vice versa rather than showed on diagrams!



1. Terminal connection schemes

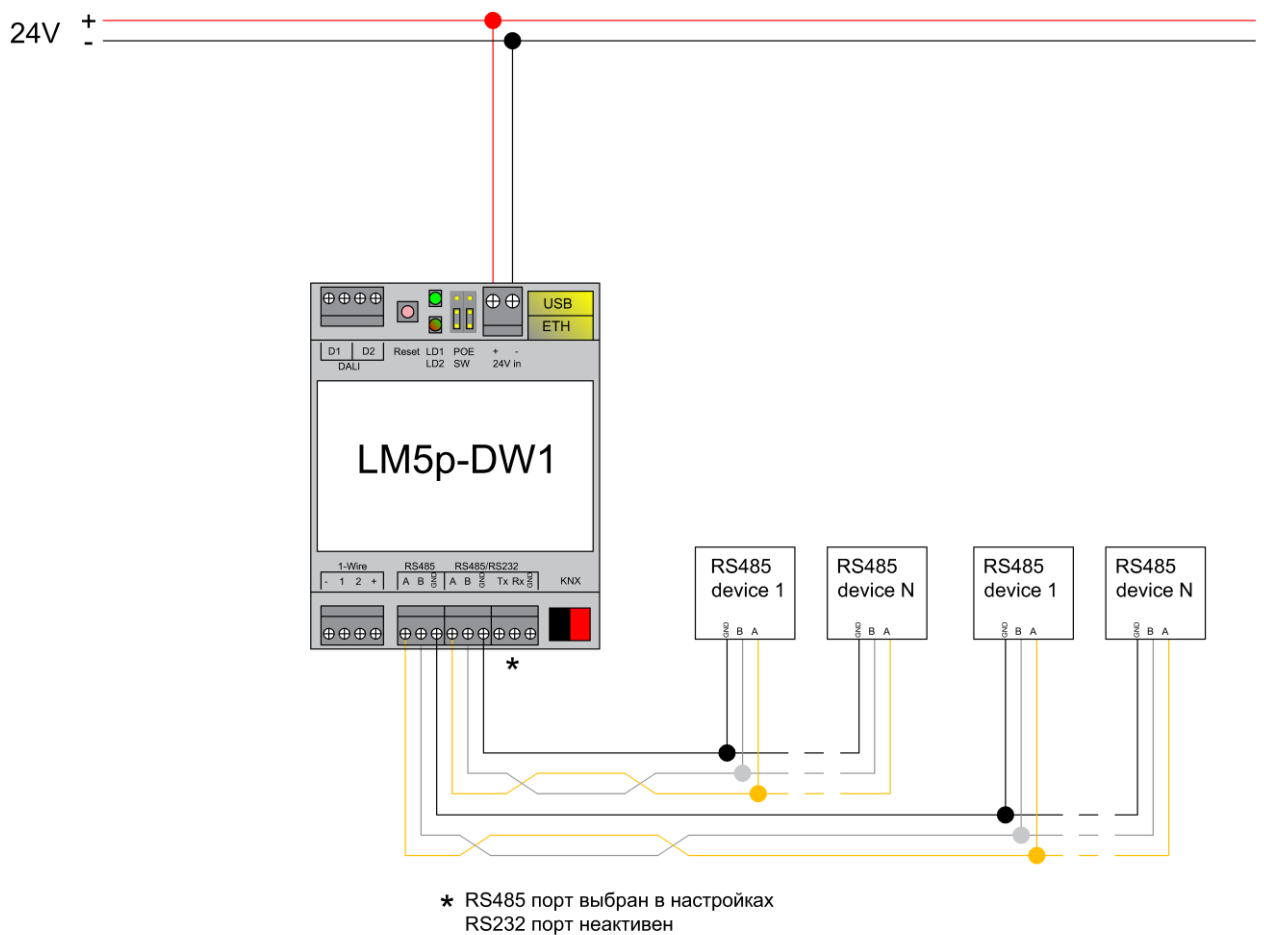
KNX connection



RS-485 connection

There can be used max two RS-485 on LM5p Power. First one is definitive, second one is software switchable – either it works as RS-485 or as RS-232 :

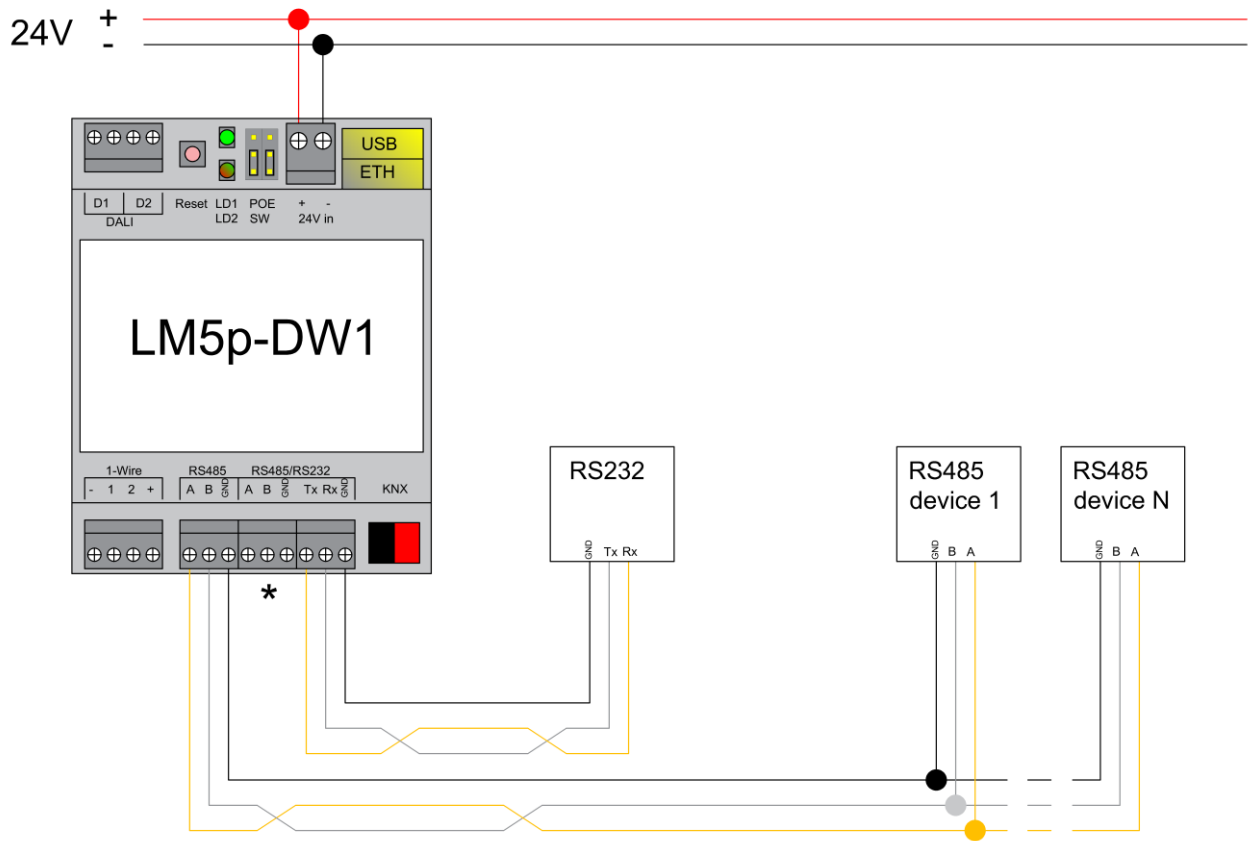
- If it is set up as full-duplex it will operate as RS-232 and respective TX/RX/GND screw terminals should be used
- If it is set up as half-duplex (*) it will operate as RS-485 and respective A/B/GND screw terminals should be used



*RS-485 is chosen in this case, RS-232 is not activated

RS-232 connection

If second serial port is set as full-duplex in LogicMachine configuration, it will operate as RS-232 and respective TX/RX/GND screw terminals should be used.



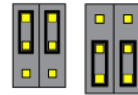
* RS232 порт выбран в настройках
RS485 порт неактивен

*RS-232 is chosen in this case, RS-485 is not activated

Powering

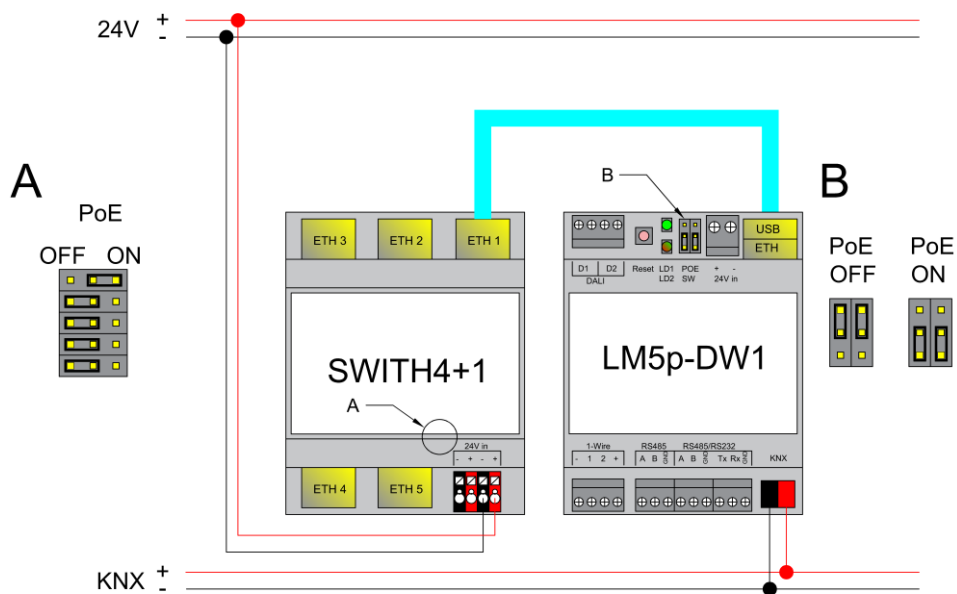
LM5 supports two powering modes:

- regular powering over screw terminals (Jumpers up or down)
- passive PoE powering over 24V DC (Jumpers down)



Please note that there are two PoE types of PoE switches/adapters – passive and active (802.3af). In passive mode 4 Ethernet cable wires are used for data and 4 are used for power. In active PoE mode data and power goes together.

Note! In first batch of LM5 devices, the 24V DC clamp powering ports are placed vice versa rather than showed on diagrams!



Passive PoE switch

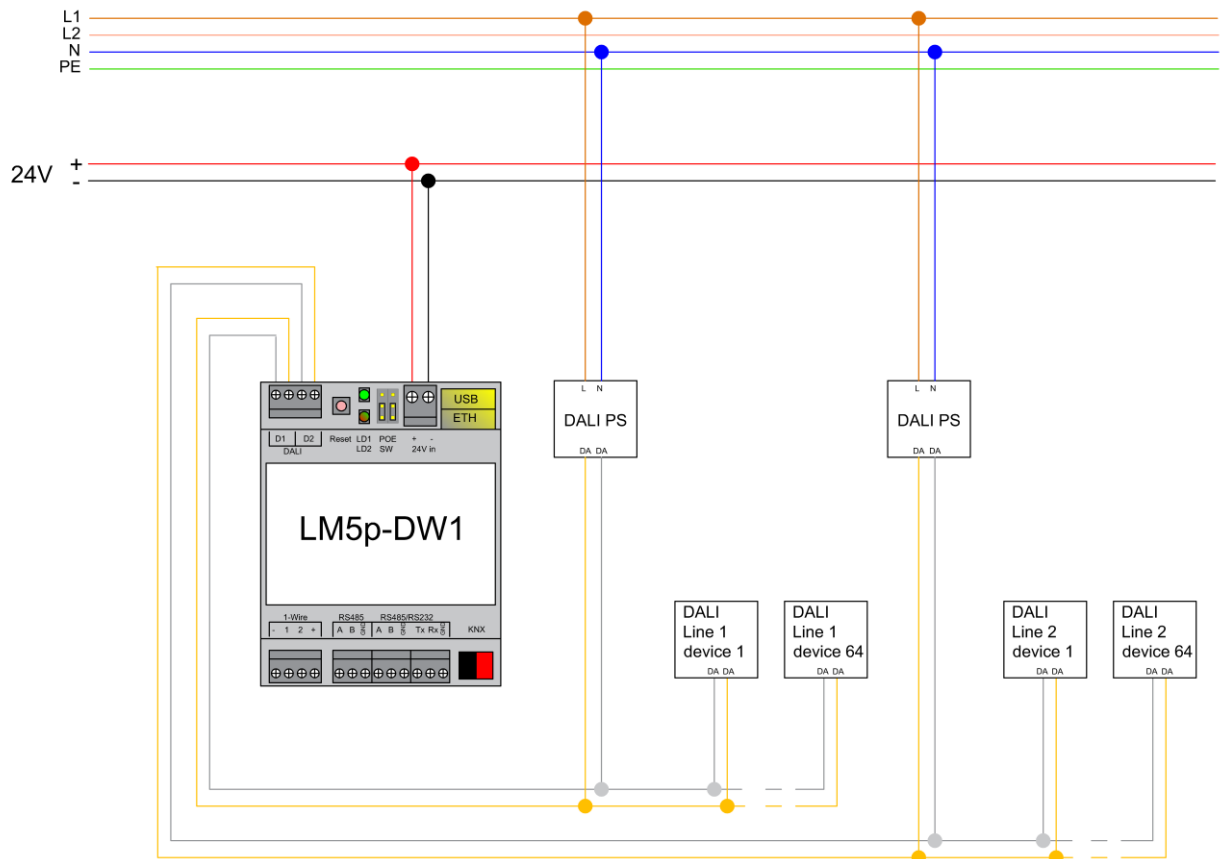


Passive PoE adapters



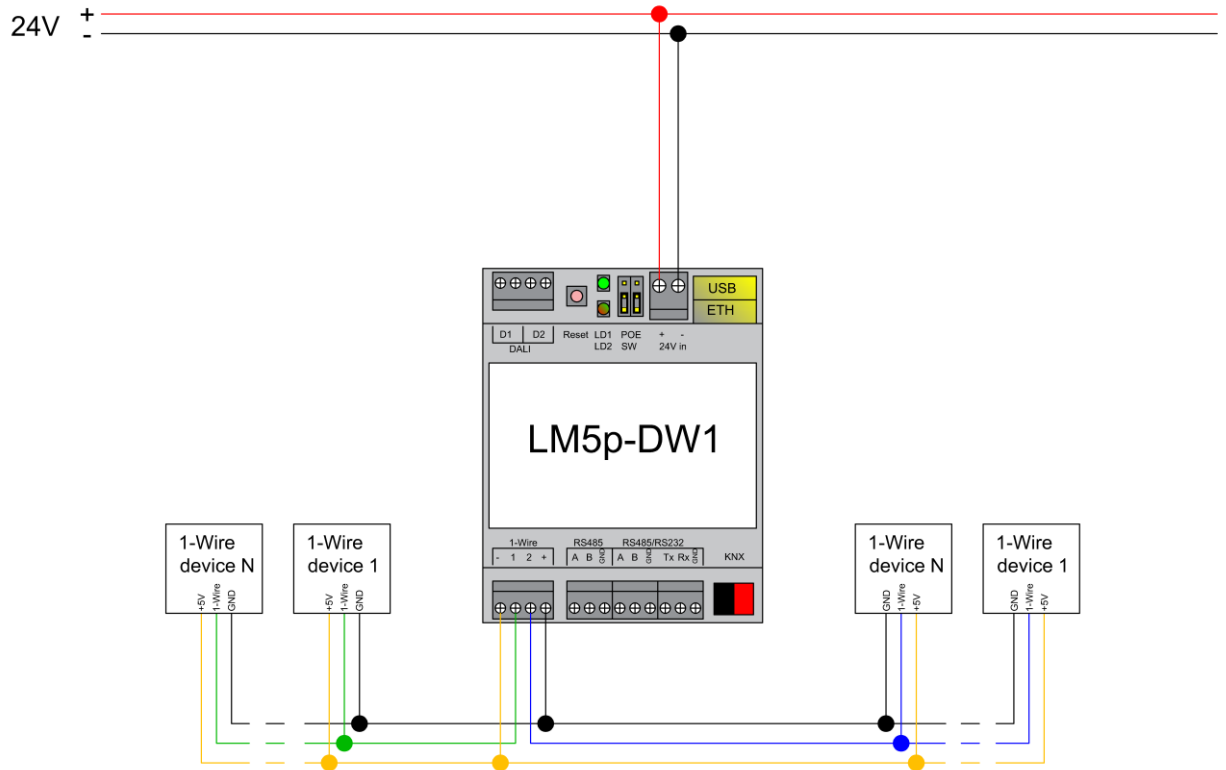
DALI connection

There are two DALI master interfaces in this device. You can connect up to 64 DALI ballasts to each master interface.

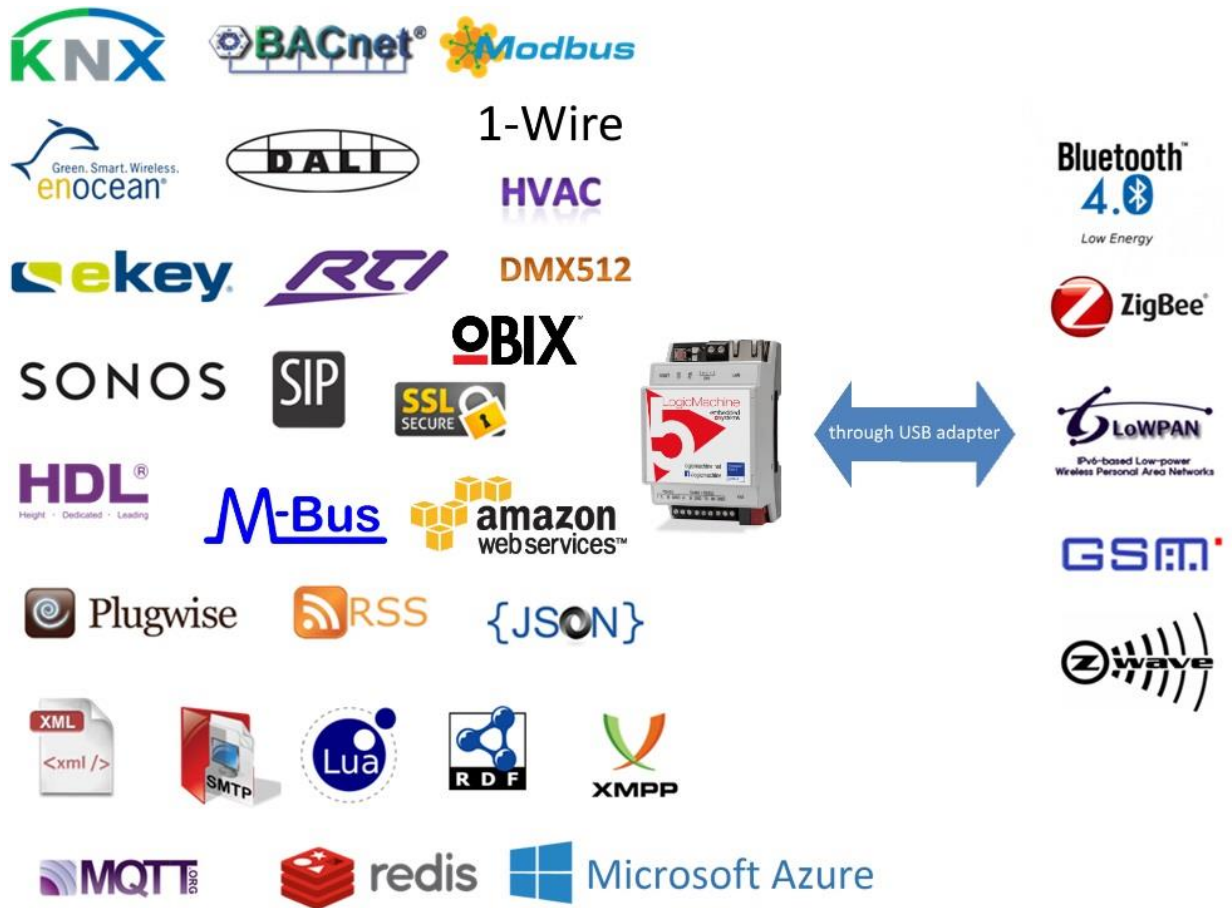


1-wire connection

There are two 1-wire interfaces in this device.



2. Standards supported



LogicMachine is compatible with the following standards:

- KNX/EIB TP, KNXnet/IP
- Modbus TCP, Modbus RTU Client/Server
- BACnet IP, Client/Server
- GSM (through USB modem) for sending SMS notifications and controlling the installation by receiving SMS commands.
- Bluetooth 3.0 and 4.0 (through USB modem)
- DMX512 (in the box, through RS485)
- DALI
- 1-Wire
- AllJoyn
- Ekey biometrical access systems (RS485)
- HVAC systems can be controller through RS485/Ethernet interface by using scripting
- SMTP/Email, SSL
- SIP
- XML (export object values, alerts or errors; integration with Fidelio)
- RSS (read Error or Alert tab content)
- JSON, XMPP
- MQTT
- REDIS
- etc.

The system is made so that each of the standards can be used with each other, so LogicMachine can act as BACnet to DALI gateway or Modbus to GSM etc.

3. Quick startup guide

3.1. Connection

- Mount the device on DIN rail
- Connect the KNX bus cable
- Connect 24V power supply to the device (red pole to 24V+, grey pole to GND)
- Connect Ethernet cable coming from the PC

3.2. Default login parameters

Login name	admin
Password	admin
IP address	192.168.0.10
Network mask	255.255.255.0

The device can be accessed by opening web browser (Chrome, Firefox, Safari are supported) and entering IP of the device [HTTP://IP](http://IP)

Secure access to the device is available via [HTTPS://IP:Port](https://IP:Port)

3.3. Factory default

You can either reboot the device by pressing RESET button or reset the configuration to factory defaults:

- *Press and hold for <10 sec* – reboot the device
- *Press and hold for >10 sec* – reset networking with IP to factory default
- *Press and hold for >10 sec and again press and hold for >10 sec* – full reset of configuration to factory defaults

For more info please see here: <http://openrb.com/discover-ip-of-logic-machine-or-streaming-player/>

3.4. IP settings

In *System configuration* → *Network* → *Interfaces* window click on the specific interface to change the IP settings.

Interface eth0	
Protocol	Static IP
IP address	192.168.1.12
Network mask	255.255.255.0
Gateway IP	192.168.1.100
DNS server 1	8.8.8.8
DNS server 2	8.8.4.4
MTU	

OK Cancel

- **Protocol**– specific protocol used for addressing
 - **Static IP** – static IP address. By default 192.168.0.10
 - **DHCP** – use DHCP protocol to get IP configuration.
 - **Current IP**– the IP address got from DHCP server. This field appears only if the IP address is given otherwise it's hidden.
- **Network mask** – network mask. By default 255.255.255.0 (/24)
- **Gateway IP** – gateway IP address
- **DNS server** – DNS server IP address
- **MTU**– maximum transmission unit, the largest size of the packet which could be passed in the communication protocol. By default 1500

When changes are done, the following icon appears in the top-right corner. This should be applied changes to take effect.

 Apply changes

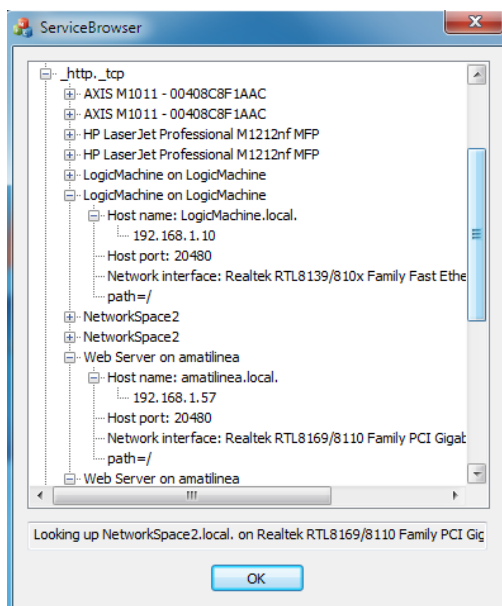
3.5. Discover LogicMachine IP address

LM has built-in zeroconf utility by default, so using the following applications you can find out the IP:

- Windows PC – *ServiceBrowser*
- Linux PC – *Avahi*
- Android – *ZeroConf Browser*
- iOS – *Discovery*

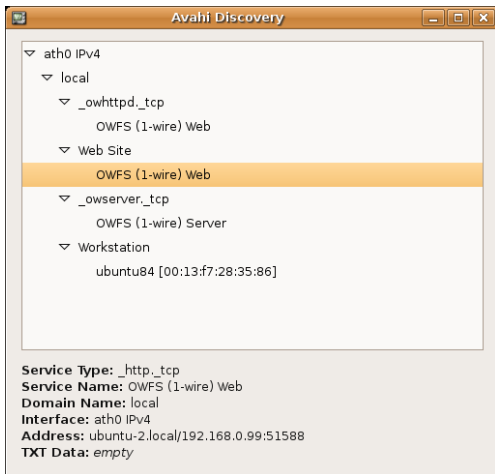
Windows PC

Easiest way is by using the utility **ServiceBrowser** which can be downloaded here: <http://marknelson.us/2011/10/25/dns-service-discovery-on-windows/>



Linux PC

The utility called **Avahi**, can be downloaded here: www.avahi.org



Android

The freely available app called **ZeroConf Browser**, can be downloaded in *Play Store*:

<https://play.google.com/store/apps/details?id=com.grokket.android.bonjour&hl=en>



iOS/Mac OS

The freely available app called **Discovery**, can be downloaded in *App Store*:

<https://itunes.apple.com/en/app/discovery-bonjour-browser/id305441017?mt=8>



For iPad install the iPhone/iPod version of the utility.



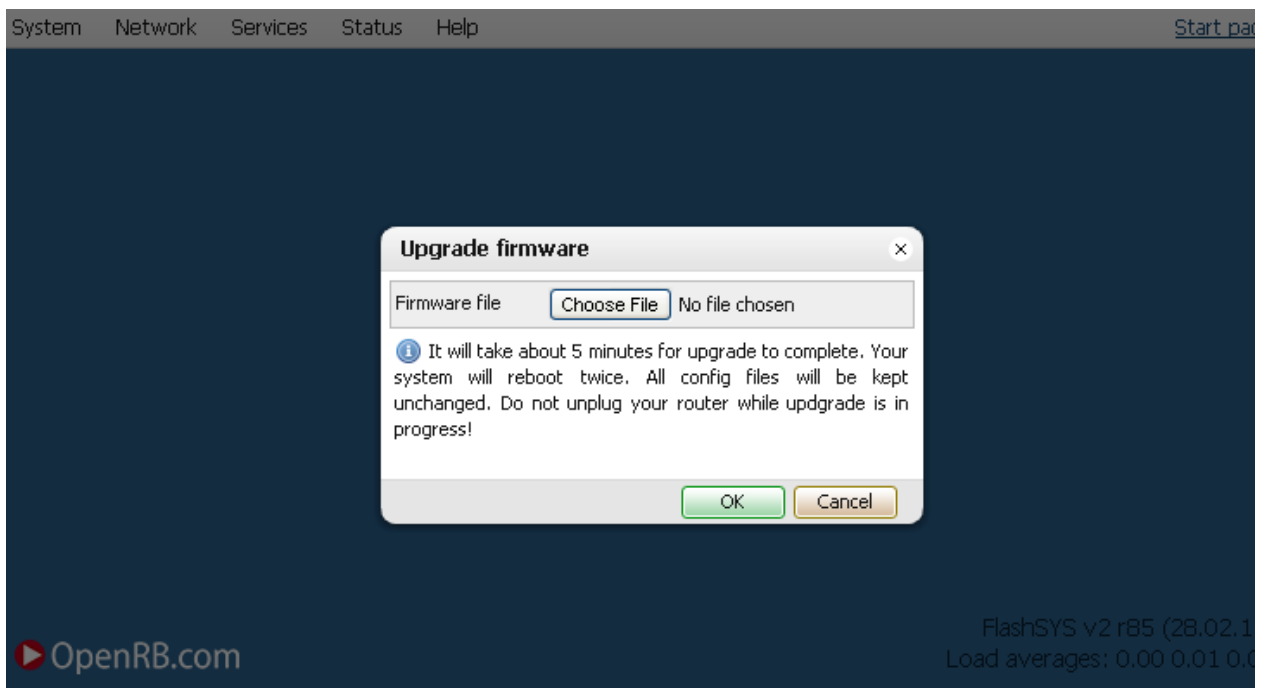
3.6. Firmware upgrade

Note! Before each upgrade please backup your visualization, scripts and object in *Logic Machine* → *Tools* → *Backup*.

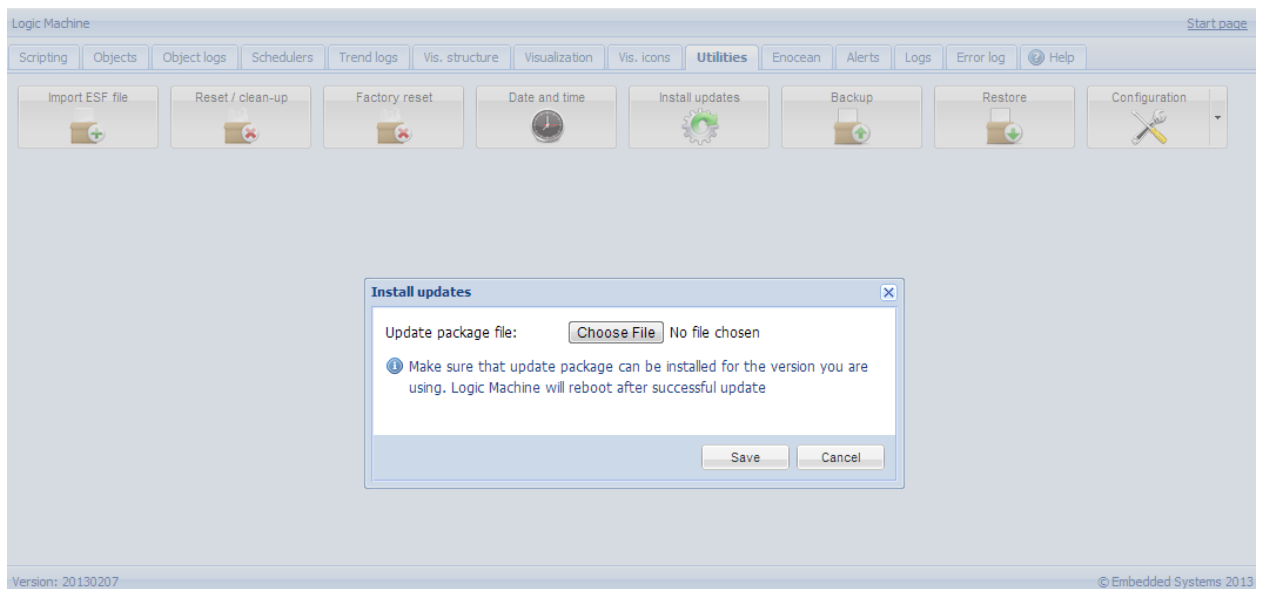
Note! After each upgrade, we strongly recommend to clean your browser cache.

Use web browser to perform upgrade of the software of Logic Machine. Firmwares are available in a form of images and could be downloaded from support page of www.openrb.com.

Complete system upgrade can be done in *System Configuration* → *System* → *Upgrade firmware*

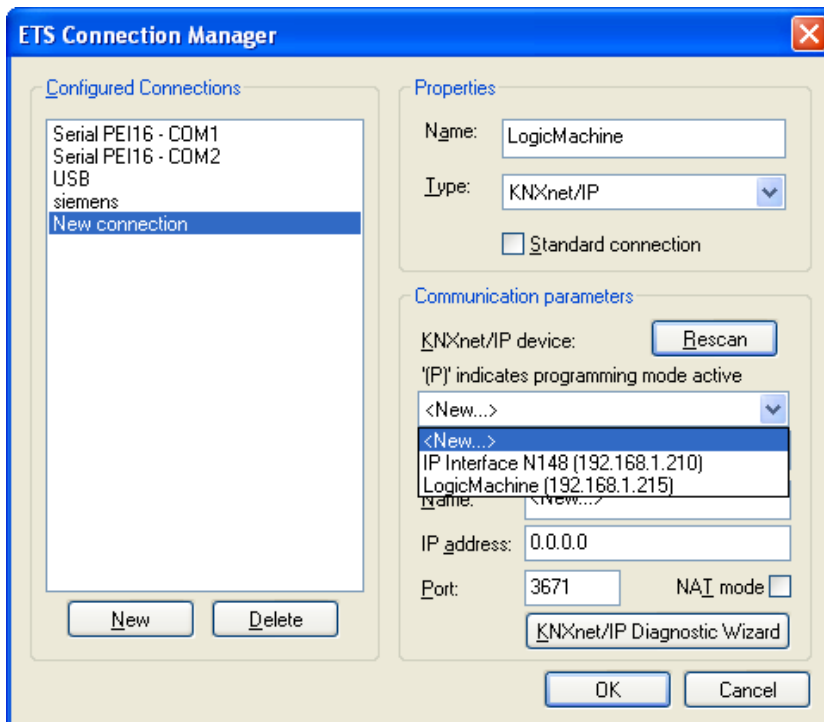


LogicMachine visualization upgrade or patch installation can be done in *Utilities* tab and press on *Install updates* icon. After *.LMU file is chosen from the corresponding location press *Save* button. The device will be rebooted after 5 seconds and new firmware will be installed.



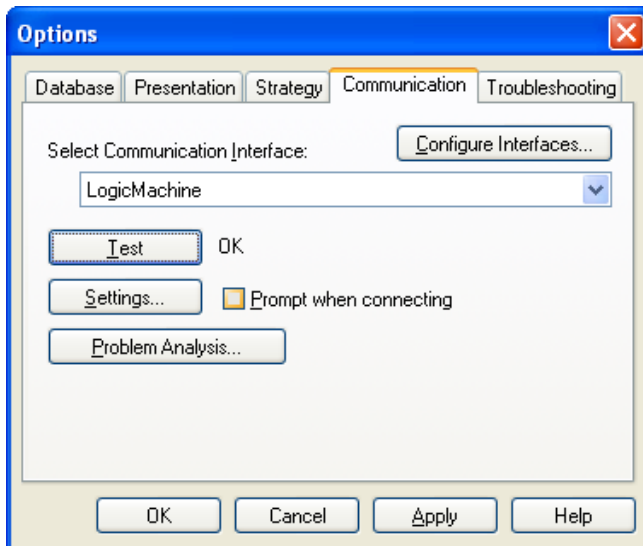
3.7. LogicMachine for KNX/EIB network configuration management with ETS


To use LogicMachine with KNXnet/IP functionality and program other KNX bus devices, the device should be added into *ETS Connection Manager*.



- Go to *Extras* → *Options* → *Communication* → *Configure interfaces*
 - Put some freely chosen *Name* for the connection

- Chose **Type = KNXnet/IP**
- Press **Rescan** button and then choose from the drop down menu found LogicMachine
- Press **OK**
- Back in **Options → Communication** window select newly created interface as **Communication Interface** from the drop-down menu.
- To test the communication with ETS, press **Test** button.



- Make sure that bus status is Online – press  button in ETS.

3.8. KNX and IP Router settings

KNX specific configuration is located in *System configuration → Network → KNX connection* window.

KNX connection	
General IP > TP filter TP > IP filter	
Mode	TP-UART
ACK all group telegrams	<input type="checkbox"/>
KNX address	15.15.255
KNX IP features	<input checked="" type="checkbox"/>
Multicast IP	224.0.23.12
Multicast TTL	1
Maximum telegrams in queue	100

OK Cancel

General tab

- **Mode** [TP-UART / EIBnet IP Tunneling / EIBnet IP Tunneling(NAT mode) / EIBnet IP Routing] – KNX connection mode. LogicMachine5 has TPUART interface by default built-in. **Note!** If there is no KNX TP connected to the device, it will automatically offer to switch to KNXnet/IP mode.
- **ACK all group telegrams** – acknowledge receipt of telegram to all group communication
- **Parameter**–KNX corresponding interface in OS of the system
- **KNX address** – KNX physical address of the device
- **KNX IP features** – Use this device with KNX IP features e.g. for KNXnet/IP network configuration
- **Multicast IP** – multicast IP address
- **Multicast TTL** – Time to live for multicast telegram in seconds
- **Maximum telegrams in queue** – count of maximum telegrams in the queue

IP > TP filter

Filtering table for telegrams going from IP network to KNX TP1 is located in this submenu.

KNX connection [Close]

General | **IP > TP filter** | TP > IP filter

Apply filter to tunneling

SRC policy: No filter

Ind. address list: [Empty text area]

i One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.

DST group policy: No filter

Group address list: 1/1/1-1/1/2

i One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.
Note: KNX IP features are required for filter to work.
 Filtering lists are updated at once, changing policies requires restart.

OK Cancel

- **Apply filter to tunneling** – either to apply filter policy to telegrams in tunneling mode. If ETS is used it is recommended to turn this feature off.
- **SRC policy** [No filter / Accept selected individual addresses / Drop selected individual addresses]– policy to apply to the list of source addresses
- **Ind. address list** – list of individual addresses. One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.
- **DST group policy**[No filter / Accept selected group addresses / Drop selected group addresses]– policy to apply to the list of destination group addresses
- **Group address list** – list of group addresses. One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.

Note! KNX IP features should be on for filter to work. Filtering lists are updated at once, changing policies requires restart.

Note that group address list can be filled automatically by checking necessary group addresses in *LogicMachine* → *Objects* list

Logic Machine

Reactor Scripting **Objects** Object logs Schedulers Trend logs Vis. structure Visualization Vis. graphics Utilities Modbus EnOcean Alerts Logs Error log Help

Object filter	Group address	Object name	IP > TP filter	TP > IP filter	Event script	Data type	Current value	Log	Export
Name or group address: <input type="text"/>	1/1/1	Digital output 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		01.001 switch	on	<input type="checkbox"/>	<input type="checkbox"/>
Data type: <input type="text"/>	1/1/2	Digital output 16	<input checked="" type="checkbox"/>	<input type="checkbox"/>		01.1 bit (boolean)	1	<input type="checkbox"/>	<input type="checkbox"/>
	1/1/3	Digital output 2	<input type="checkbox"/>	<input type="checkbox"/>		01.1 bit (boolean)	1	<input type="checkbox"/>	<input type="checkbox"/>
	1/1/4	Digital output 3	<input type="checkbox"/>	<input type="checkbox"/>		01.001 switch	on	<input type="checkbox"/>	<input type="checkbox"/>

TP > IP filter

Filtering table for telegrams going from KNX TP1 to IP network is located in this submenu.

KNX connection

General **IP > TP filter** TP > IP filter

Apply filter to virtual objects

SRC policy No filter

Ind. address list

One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.

DST group policy No filter

Group address list

1/1/1

One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.

Note: KNX IP features are required for filter to work.
Filtering lists are updated at once, changing policies requires restart.

OK Cancel

- **Apply filter to virtual objects** – either to apply filter policy to objects added in Objects tab as virtual objects without attraction to bus
- **SRC policy** [No filter / Accept selected individual addresses / Drop selected individual addresses] – policy to apply to the list of source individual addresses
- **Ind. address list** – list of individual addresses. One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.



- **DST group policy** [No filter / Accept selected group addresses / Drop selected group addresses]– policy to apply to the list of destination group addresses
- **Group address list** – list of group addresses. One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.

Note! *KNX IP features* should be on for filter to work. Filtering lists are updated at once, changing policies requires restart.

3.9. Quick guide - MOSAIC application for easy visualization

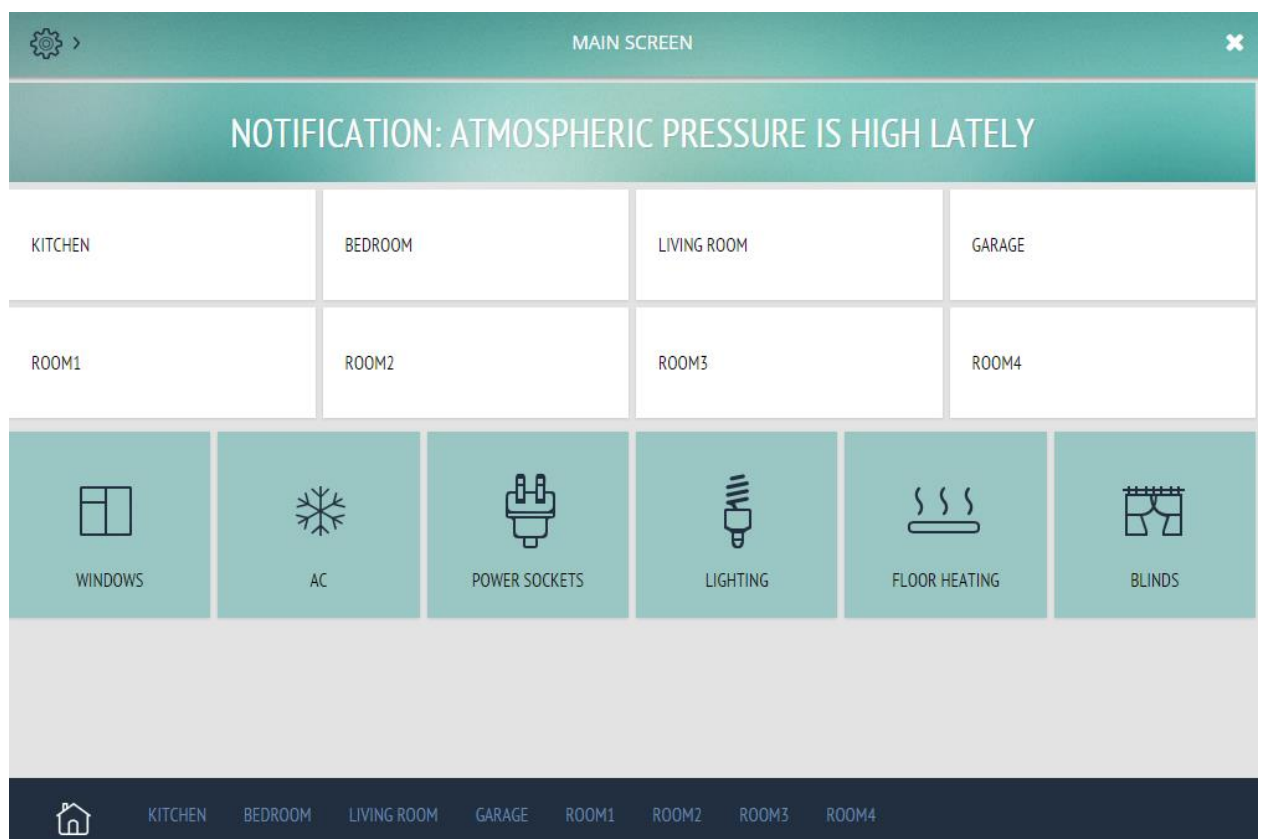
Mosaic app is the fastest way to create a nice visualization for your installation.

Getting started.

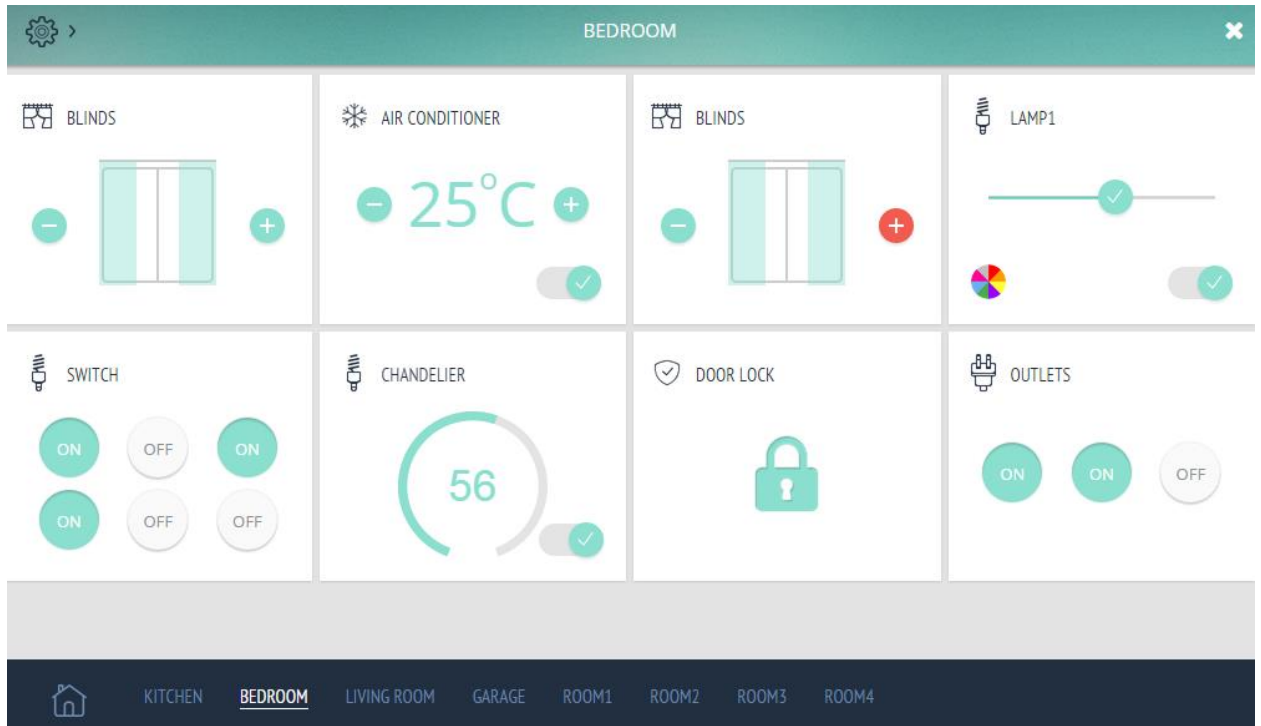
1. Open the controller's web interface by typing it's IP address in your web browser.
2. Click on Mosaic Editor Mode icon. You will see the constructor interface with clear template.
3. Once you've entered the Editor Mode you are ready to construct a visualization.
4. When you have made visualization, enter Client  mode 

Client mode Home screen

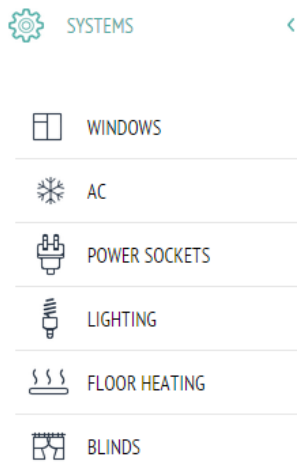
This is a first screen that you will see after opening Mosaic application. First page consists of Notification, Room and Control Type areas. You can browse the objects either by rooms or by functions.



For example, entering one of room, you see the following view



There are also Control Type shortcuts on the left side by clicking on Settings button



THEME: DEFAULT, RED, BLACK

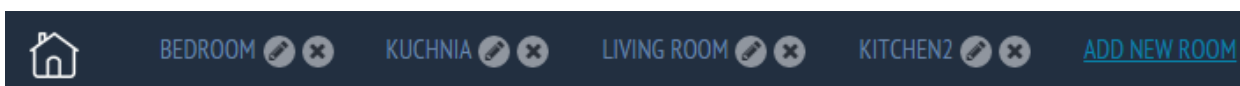
In Settings you can also change skin of the visualization

Advanced visualization with more widgets is planned to be available for fixed monthly developers fee.



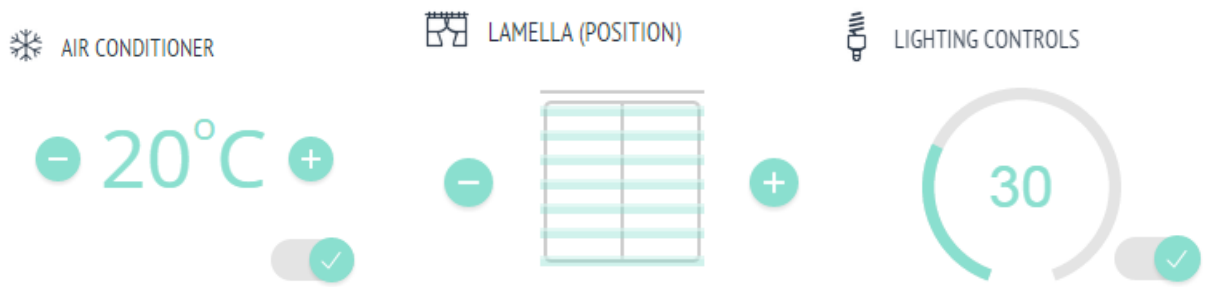
Building structure / Editor Mode

On the bottom panel you can set up your building structure by adding new rooms onto it. This panel is also designed to navigation by rooms for end-users. Just click on *Add new room*, then type a title for it and press "enter". This will take you to the screen of the new room. Now you can start to fill it with widgets. If you want to rename or delete a room just right-click on it's title and select the option you want.



Widgets

To add new widget on the home or room screen just click to *Add new widget* at the top right corner to open the widgets panel. Once you opened the widget panel choose a widget you want to add and click on it - it will appear on the screen. A popup window with widget's properties opens automatically at the same moment.



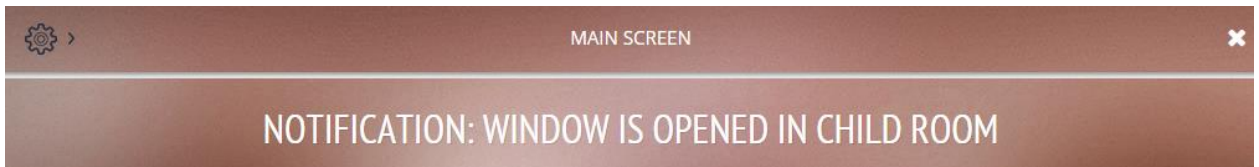
Now you can set a title for this widget and link KNX objects to widget's controls. After you've set up a widget click on the "Add this widget" button. Well done, the widget on the screen. Right-click on it to see/edit its properties or delete the widget.

Lighting Controls ✕

Title:	<input type="text" value="central lamp"/>
On/Off Object:	<input type="text" value="0/0/1"/>
On/Off Status Object:	<input type="text" value="0/1/1"/>
Dimmer Object:	<input type="text" value="1/1/11 (light1-1)"/>
Dimmer Status Object:	<input type="text" value="0/1/2"/>
Color Object:	<input type="text" value="1/1/12 (light1-2)"/>
Color Status Object:	<input type="text" value="1/1/12 (light1-2)"/>

Notifications

There is a special Messages field in Client mode where you can send specific notifications or alerts. Use storage name *mosaic-message* to write notifications.



Custom widgets

Here you can download Mosaic custom widget creation manual: http://openrb.com/wp-content/uploads/2016/11/Mosaic_widgets_eng.pdf

Here you can download couple of custom widgets examples together with the instruction on how to install them:

<http://forum.logicmachine.net/showthread.php?tid=122&pid=2651#pid2651>

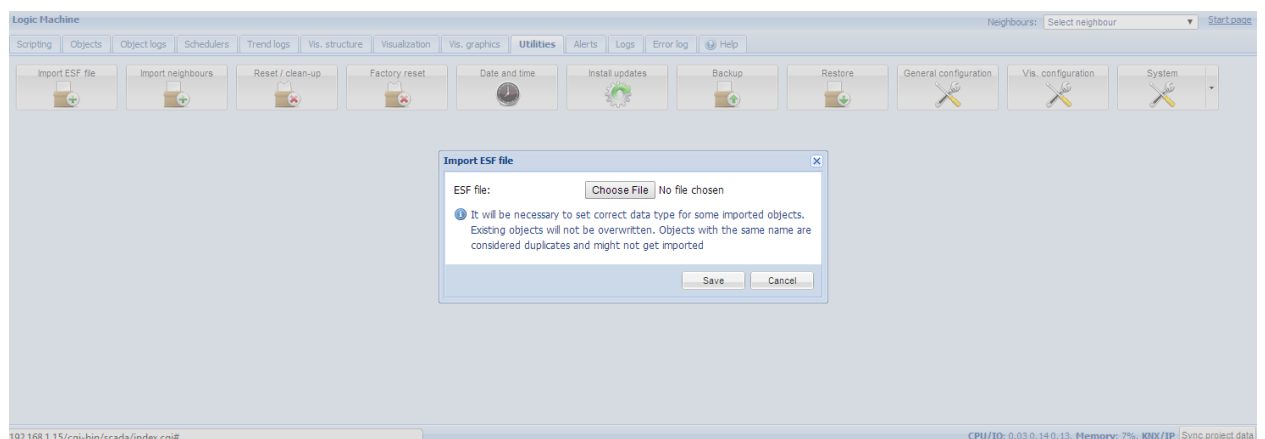
3.10. Quick guide - create visualization for iPad/PC

You can download ready LM backup files here:

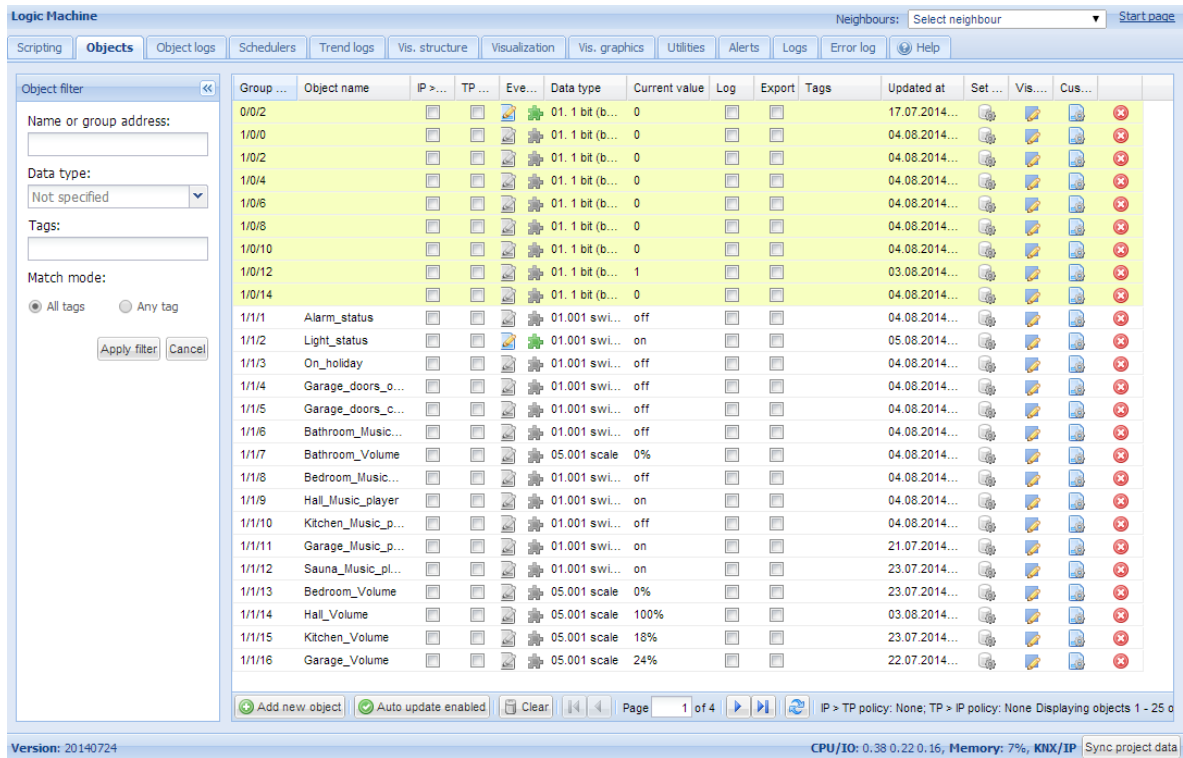
<http://forum.logicmachine.net/showthread.php?tid=196>

Import objects

Fastest way is to import *.ESF file from ETS in *Logic Machine* → *Utilities* → *Import ESF file*.

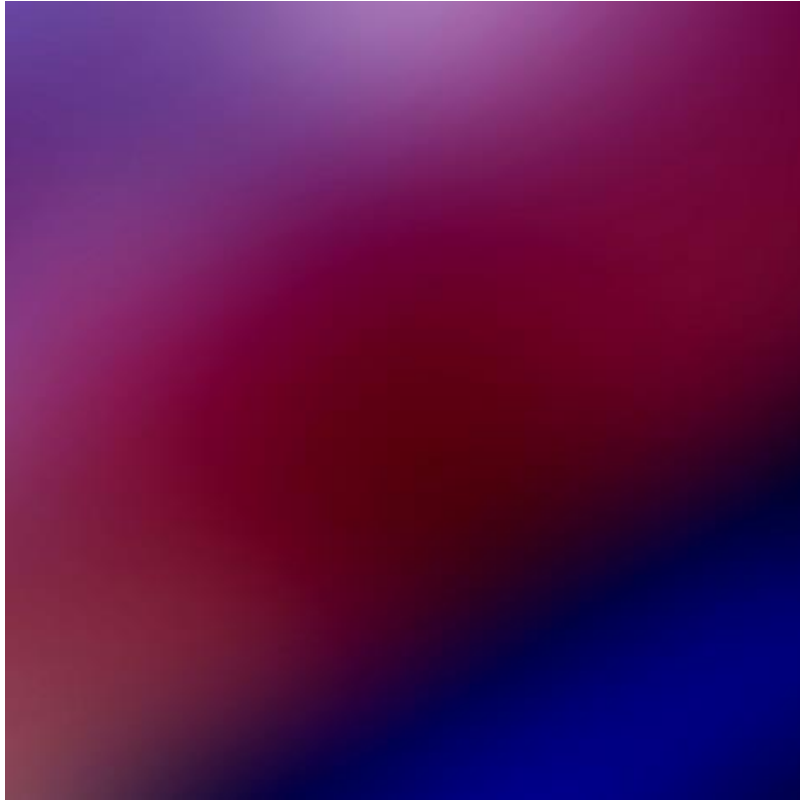


Or connect LM to the bus and it will detect objects automatically (in yellow) in *Objects* tab once they are activated. Objects can be added manually as well.



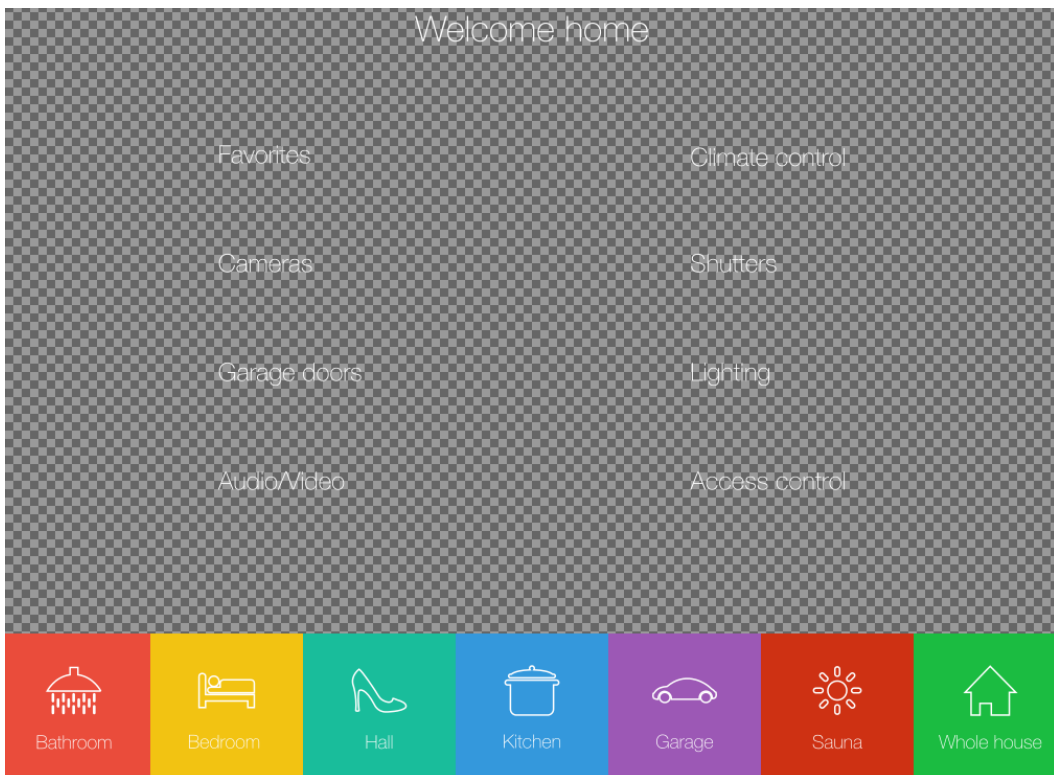
Prepare graphics

Either in Adobe Illustrator or any ready images can be used. In this example we use professionally created designs in Illustrator in SVG form (so we can do scaling depending of the screen size and not losing the quality)

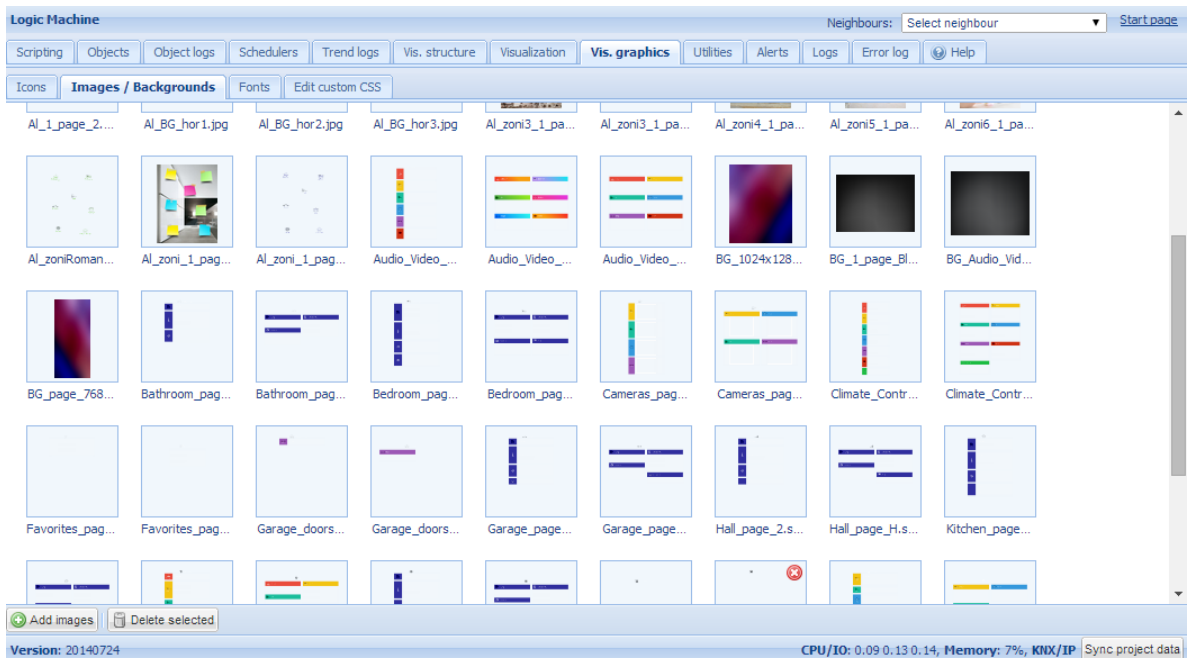


a) basic background which can be changed by necessity

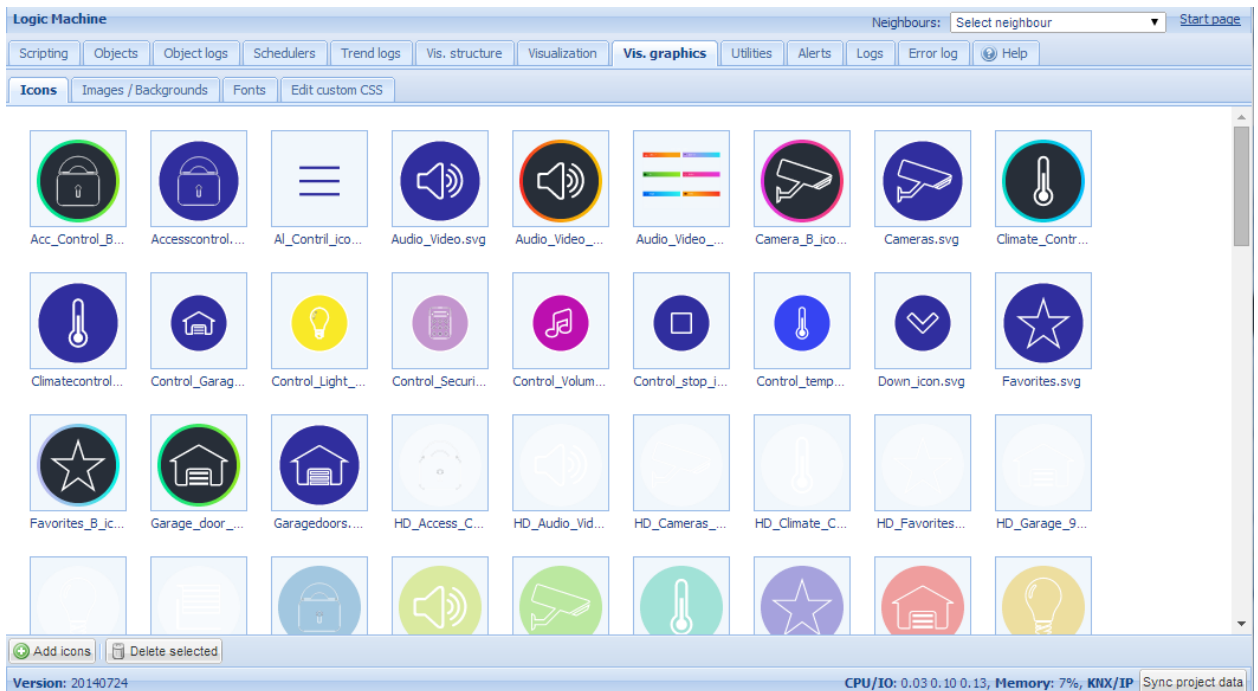
b) foreground which will stay unchanged




Add both files in *Logic Machine* → *Vis. Graphics* → *Images/Backgrounds*

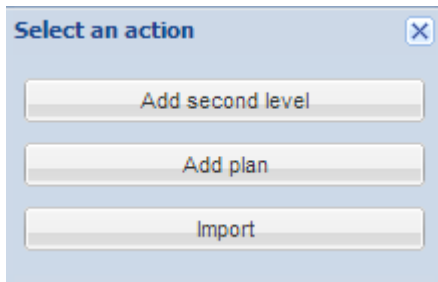


Prepare set of icons (preferably in SVG form) and add them in *Logic Machine* → *Vis. Graphics* → *Icons*. Or you can use icons predefined in LogicMachine by default.

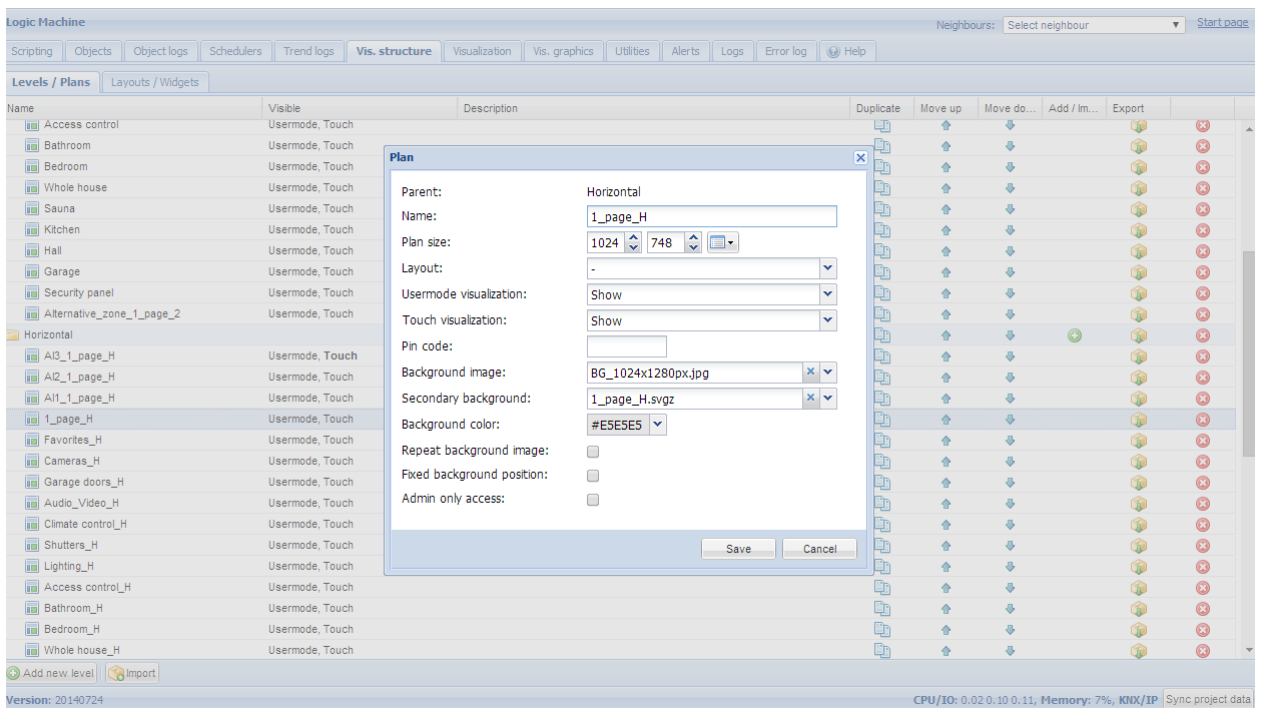


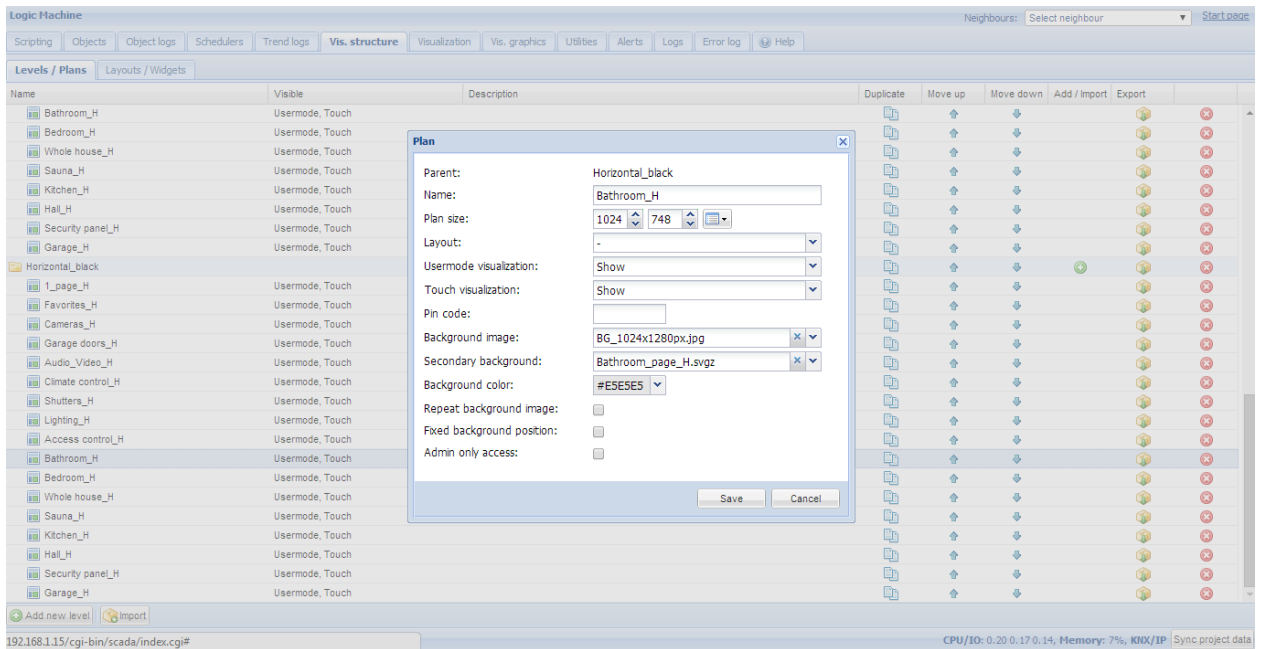
Create “floor” structure and add objects to the map

In *Logic Machine* → *Vis.structure* menu the structure of the visualization is defined and visualization backgrounds are uploaded. Use icon to add  floor.



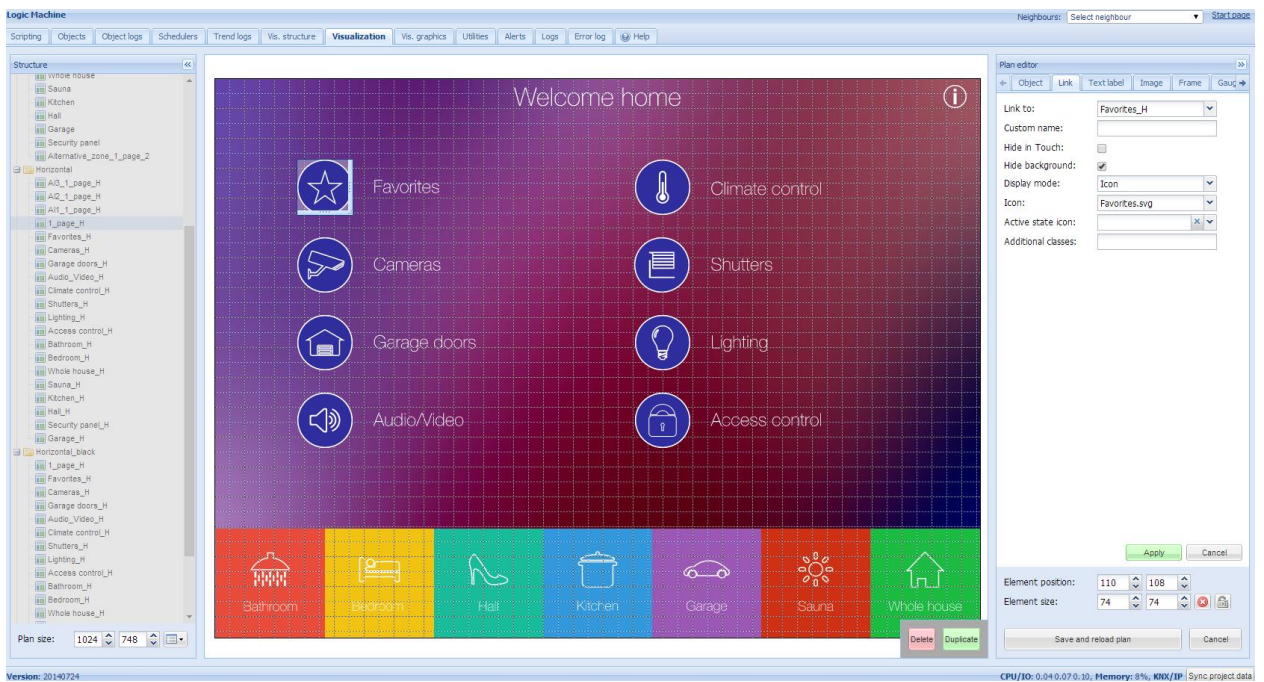
In this example we will create a new floor named “1_page_H” and “Bathroom_H”. First Floor will be a dashboard with link to other rooms and functions. Choose screen resolution for which you are creating this visualization, choose first and second background images from the ones added before.





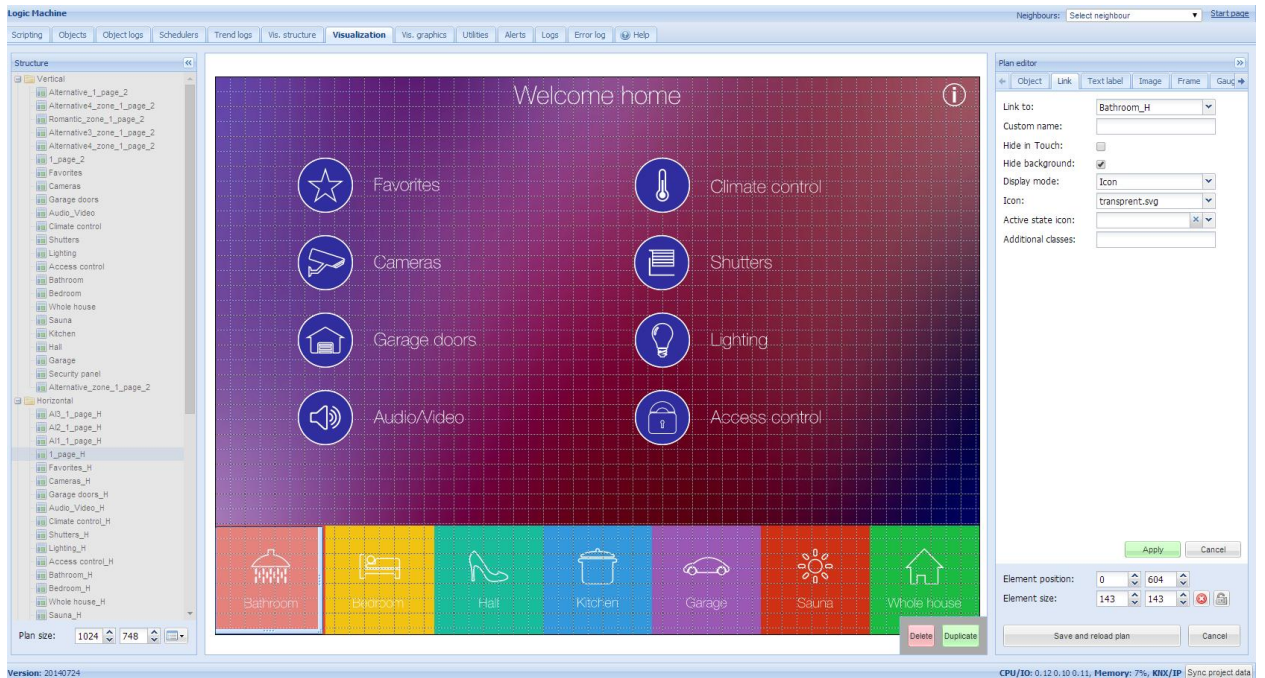
Add objects to newly created visualization map

After the building and floor structure is defined, it is visualized in *Visualization* tab. Controlled and monitored objects can be added and managed in this section. Both side bars can be minimized by pressing on left/right arrow icon making the map more visible especially on small displays.



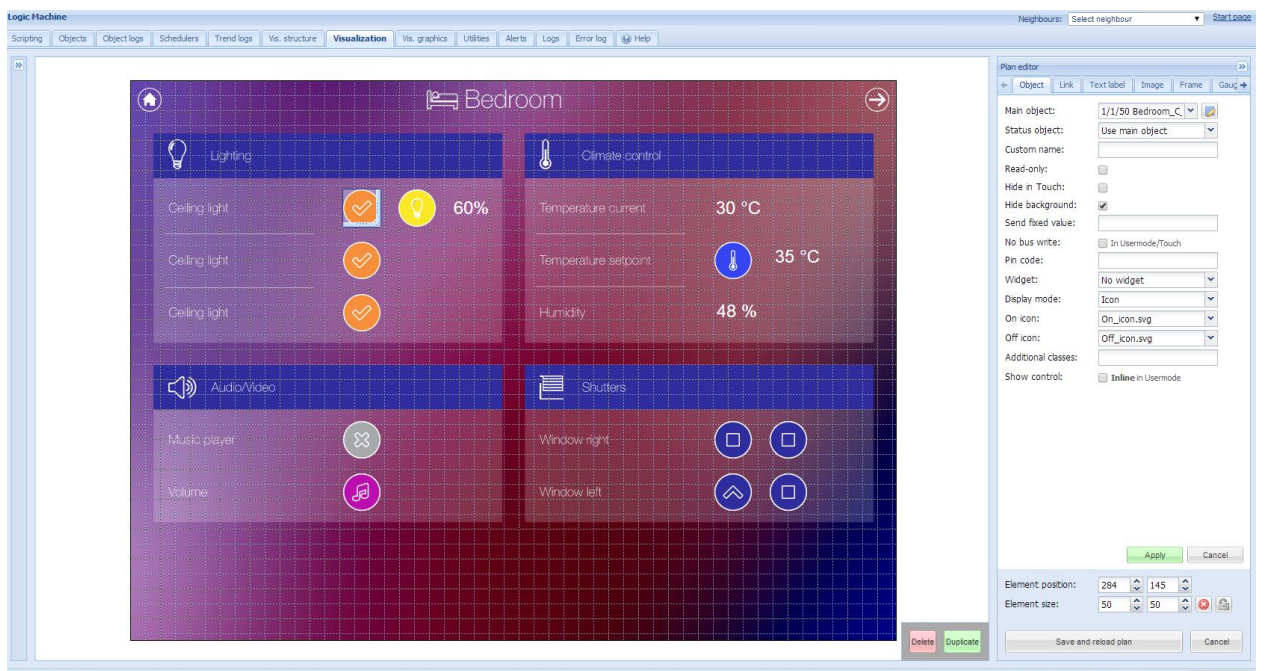
Objects can be added to the map by clicking on *Unlock current floor plan for editing* button. In this example we are creating first page of visualization which will link to other Floors with specific object control. Add link by clicking on Link tab, choosing specific icon, scale it and place in desired location.

This example's secondary background already contains icons on it, so what is needed, is to add transparent image in *Vis.graphics* and add this image on top of every icon.



When all links are defined, press *Save and reload floor plan* button.

In same way fill the Bedroom plan with object parameters in Object tab.



Launching visualization on touch device (iPad in this case)

- Make sure your iPad is connected wirelessly to the Logic Machine
- In the browser enter Logic Machine's IP (default 192.168.0.10).
- Click on the User *mode visualization*
- Save the application as permanent/shortcut in your iPad



Touch visualization is also automatically created with list of Floor objects.

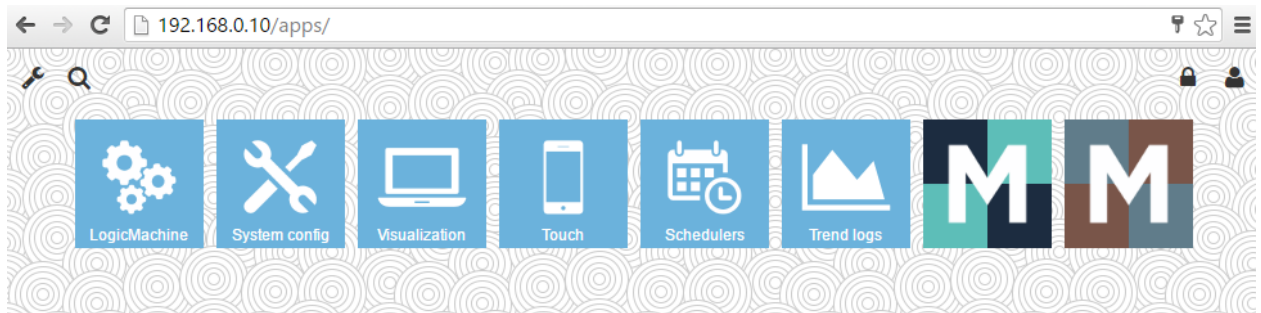
> 1_page_H	> Lighting_H
Bedroom_C_Light <input checked="" type="checkbox"/> ✓	Bedroom_R_light <input checked="" type="checkbox"/> ✓
Bedroom_FL_light <input checked="" type="checkbox"/> ✓	Bedroom_Music_player <input checked="" type="checkbox"/> ✕
Bedroom_Volume 0% <input type="range"/>	Ceiling light 60%
Bedroom_C_Light_Control 60%	Bedroom_Tem_current 30 °C
Bedroom_Temp_Setpoint 35 °C <input type="text" value="35.00"/>	Bedroom_W_right_open <input checked="" type="checkbox"/> ✓
Bedroom_W_left_open <input checked="" type="checkbox"/> ✕	Bedroom_Humidity 48 %
Bedroom_W_right_close <input checked="" type="checkbox"/> ✓	Bedroom_W_left_close <input checked="" type="checkbox"/> ✓
Bedroom_Temp_Setpoint 35 °C	> 1_page_H
> Climate control_H	> Audio_Video_H
> Shutters_H	

4. Graphical User Interface Login

KNX/EIB LogicMachine has IP address 192.168.0.10 set by default to LAN interface. Use this address as www address in the browser's address field.

Note! Make sure that the PC connecting to the LogicMachine has IP set from the same subnet.


After successful login a default page appears.

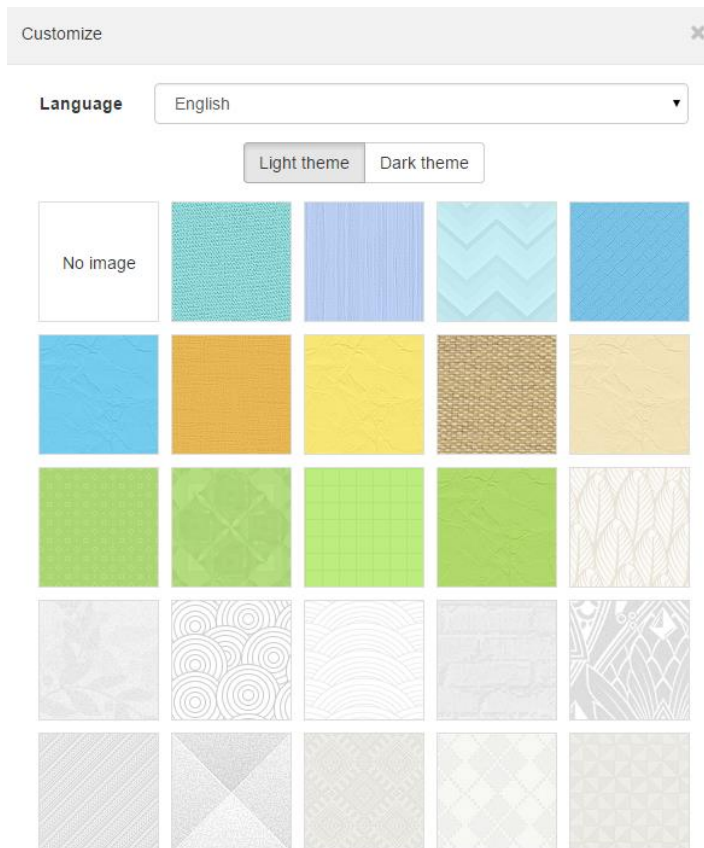


- **Logic Machine** – visualization creator, scripts, object relations, alerts, KNX objects and KNX objects, designing building view and visualization maps
- **System config** – IP and KNX specific configuration
- **Visualization** – defined visualization maps with objects
- **Touch** – Visualization system for iPhone/iPod/iPad/Android touch screen devices
- **Schedulers** – User defined schedulers
- **Trend logs** – Trends for data logs
- **Mosaic app** – Mosaic easy visualization creation and presentation apps

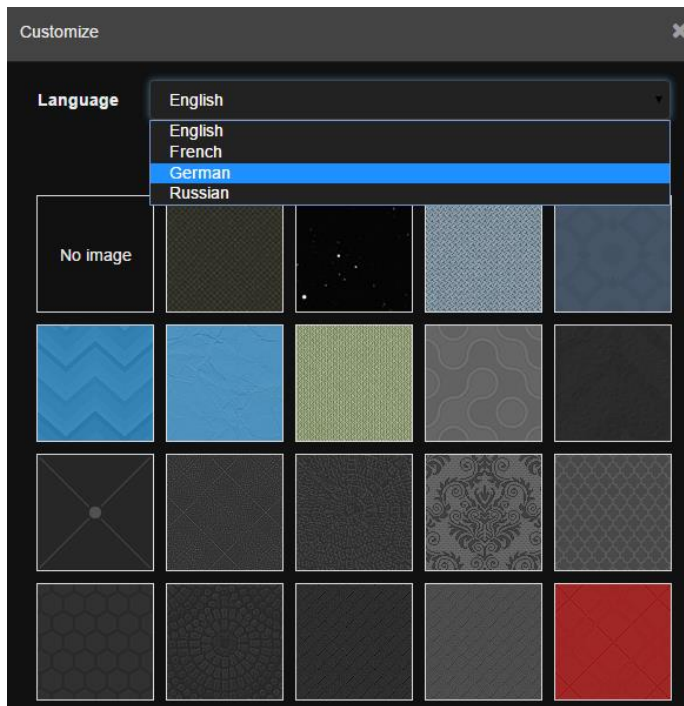
First screen of the interface is a constructor of applications – you can change applications which you see on specific device, change background color, install or remove apps, sort order etc. Note that the mentioned settings are individual for each device you are connecting from.

4.1. *Customize background / Language*

By clicking on Customize icon , you access the window where you can choose background image of your first screen for this particular device




You can choose the interface language by clicking on *Language* drop-down menu.

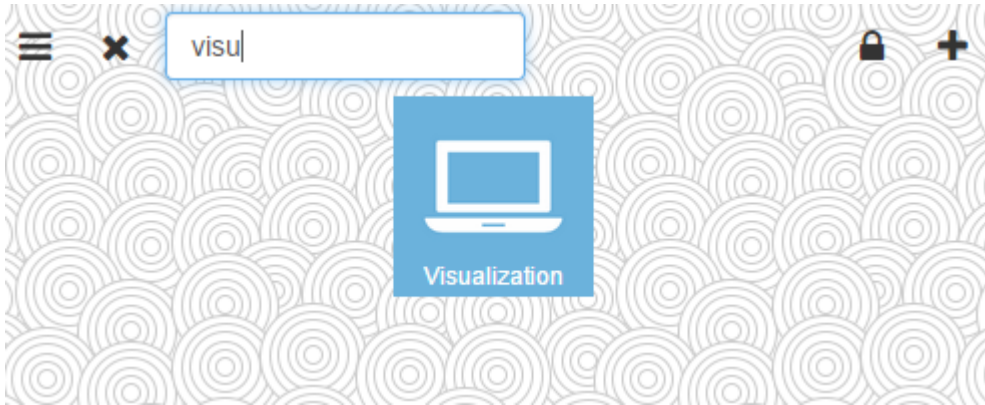


After you choose the style, the interface is automatically set to chosen background




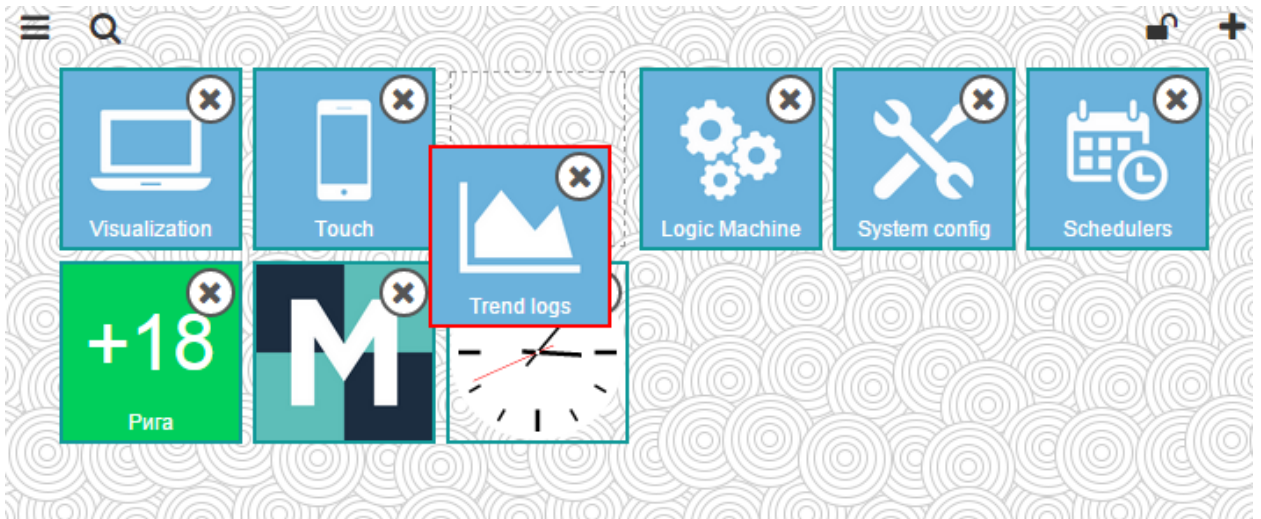
4.2. Find applications


By clicking on the zoom icon  on the left top corner, you can quickly find applications containing search phrase.




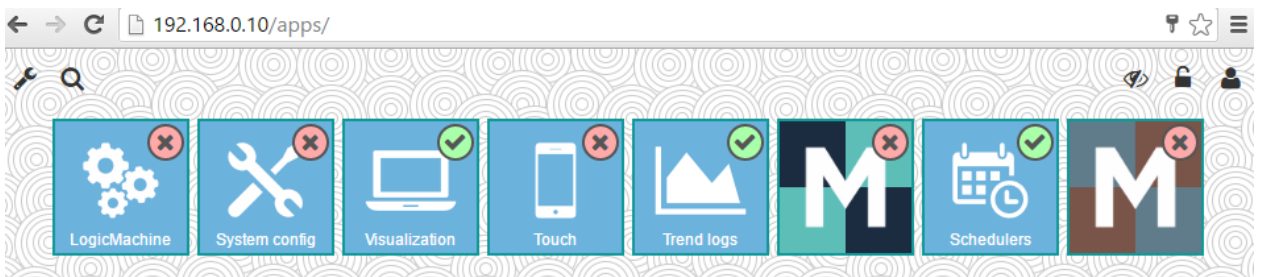
4.3. Unlock the screen for sorting order and hiding apps

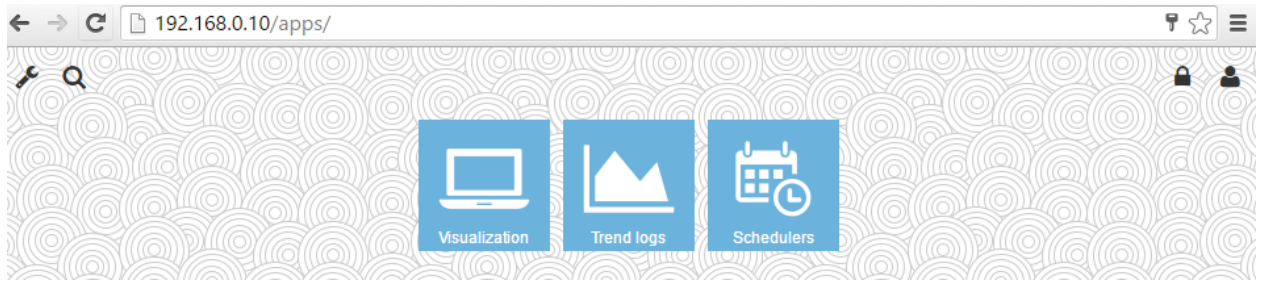
Sorting the order of applications is available when the screen is unlocked. Press Unlock icon for this purpose 




If you press Visibility icon  you will be able to hide/unhide apps from main screen. This setting can be disabled by admin.

For example if you disable specific apps and finish sorting, click Lock icon  to see the new screen.





4.4. Admin mode: adding/removing/administering apps

Enter admin mode by clicking on Admin icon  and entering the password.

Password
admin



Enter admin password ✕


Change admin password

Once in admin mode, click on Settings icon  and press *Change admin password* button.


Another setting here is *Allow users to show/hide apps* which will enable/disable the possibility to show or hide apps for end-users as shown in 4.3.

Change default page view for users

While in admin mode, do the necessary page adjustments – you can change background texture, hide/unhide/sort apps by using same icons as in user mode (see 4.3)  

Once all necessary changes are done, click on Save icon  to save the default page.

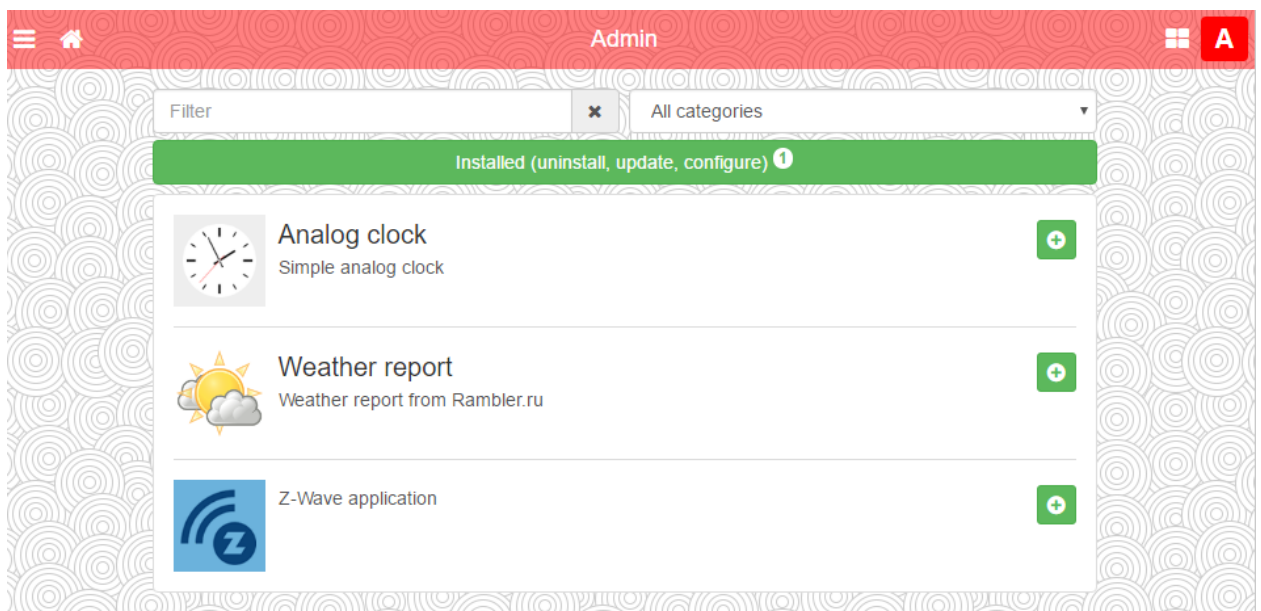
Add / remove apps


Click on plus icon  to enter the administration window of apps. If you see the following window, make sure you have set correct IP/gateway/DNS settings for your LogicMachine (*System config* → *Network* → *Interfaces*)

Load failed, check that device has Internet connection and correct DNS configuration

OK

On the default App management page you see available applications.



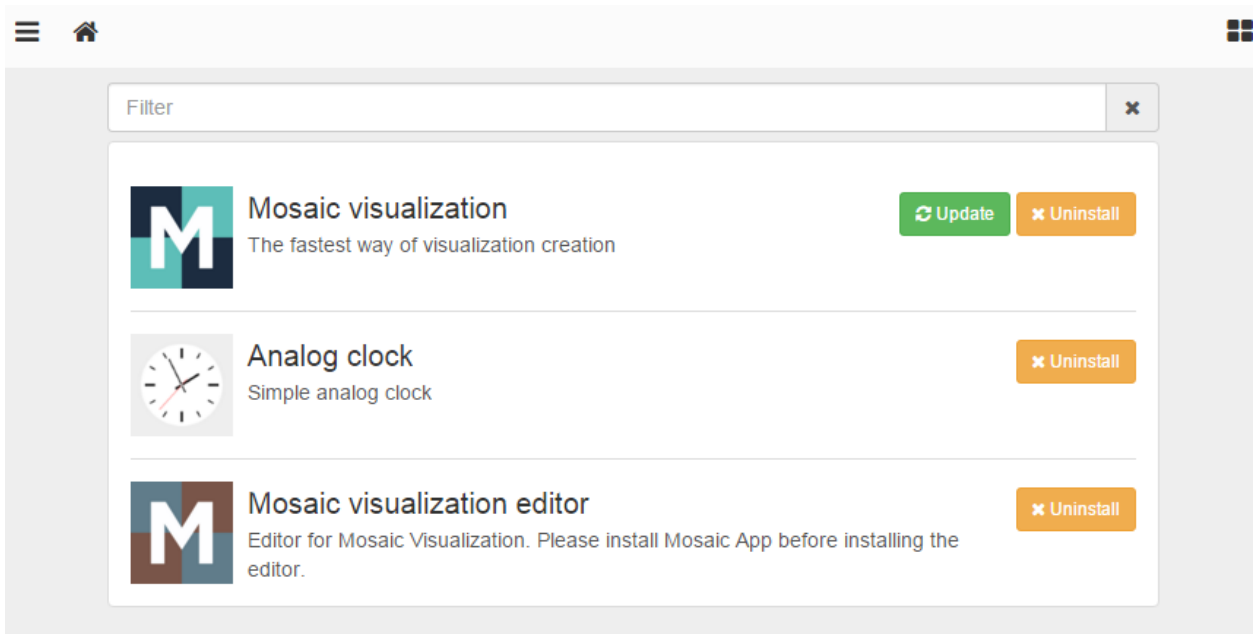
You can add the App to your first screen by clicking on Install button  and approve the choice

Install Analog clock?

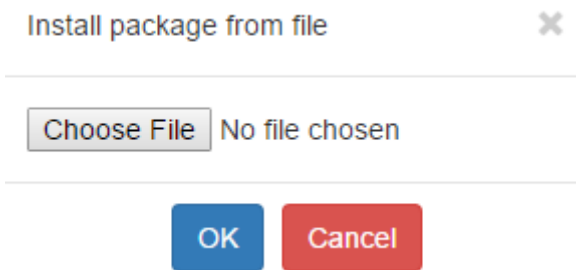
Yes


No

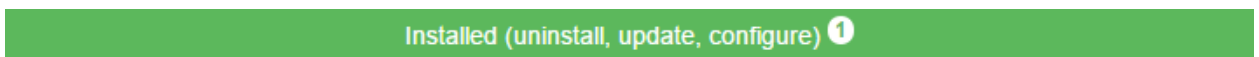
The installed Apps will appear then in **Installed** section where you can uninstall them by necessity.




You can install the app also from the file, by clicking on **Install from file** entry




To update app click on respective Update icon  or from main administration screen – on the following icon:



By pressing on this icon  you will be redirected to initial front page.

Exit admin mode

Click  icon to exit admin mode.

5. Application development

Available libraries/frameworks

- jQuery v2 (<http://jquery.com/>)
- Bootstrap v3 (<http://getbootstrap.com>)
- Font Awesome v4 (<http://fontawesome.io>)

Bootstrap comes without *Glyphicons*, use *Font Awesome* instead.

Base directory structure

- `/data` – apps and widgets are stored here, accessible at `http://IP/apps/data/`
- `/libs` – Lua library storage, loaded via `require('custom.lib')` where *lib* is library name.
- `/user` – allows storing user files and LP scripts, accessible at `http://IP/user/`

App / Widget structure

Application name (ID) must be unique and can only contain alphanumeric characters, hyphens and underscores. Maximum name length is 64 characters.

Directory structure

- `index.lp` or `index.html` – required for apps, unless `url` is specified, clicking app icon will open app directory in the same window. Applications must provide a *Back* button so user can return to starting page
- `icon.svg` or `icon.png` – required for apps, contains application icon, *SVG* is recommended
- `widget.lp` or `widget.js` – required for widgets, can contain *JavaScript* + *Lua* code or pure *JavaScript* source which displays widget contents
- `title` – optional for apps, text file with title that is shown beneath the icon
- `url` – optional for apps, text file with *URL* that should be open when icon is clicked
- `style.css` – optional for widget, contains custom *CSS* stylesheet for given widget
- `config.lp` or `config.html` – optional configuration file, see description below

In widget mode icon element ID is the same as widget name, all other HTML element IDs must be prefixed with unique application name to minimize collisions between different applications. The same rule applies to CSS selectors.

Default widget size is 100×100px. Width/height can be increased by calling `setWidgetSize(cols,`

rows) on widget element. Width formula: $cols * 110 - 10$, height formula: $rows * 110 - 10$

Example

Clock widget which takes double width/height and places SVG image which fills all available space inside of widget container:

```
(function() {
  // get widget element and set double width/height
  var el = $('#clock').setWidgetSize(2, 2);
  $('<object type="image/svg+xml"></object>') // object allows SVG+JavaScript
  .css('width', '100%') // full width
  .css('height', '100%') // full height
  .attr('data', '/apps/data/clock/clock.svg') // SVG image source
  .appendTo(el); // add to container
})();
```

Configuration

- Application directory must contain either *config.lp* or *config.html* file
- This file must contain *form* element, id must be set in *myapp-config* format, where *myapp* is unique application name
- Data exchange is done via events triggered on *form* element:
 - *config-load* – (to app) provides an object with all configuration key/value pairs
 - *config-check* – (to app) triggered when *Save* button is clicked, app configuration must either show an error message if configuration is invalid or trigger *config-save*
 - *config-save* – (from app) saves configuration on server side and closed modal window, application must pass configuration parameters as an object
- Configuration can be accessed from Lua using these functions:
 - *config.get(app, key, default)* – returns single value for given application name, default value or *nil* if key is not found
 - *config.getall(app)* – return *table* with all configuration values for given application name or *nil* if configuration is empty
 - *config.set(app, key, value)* – adds a new key/value pair or overwrites an existing one
 - *config.setall(app, cfg)* – overwrites existing config with given *cfg table* with keys/values
 - *config.delete(app, key)* – deletes existing key/value pair
- Unpublished apps that have configuration file present will appear under *Dev apps* in admin page

Example (config.html)

Create a simple form element with single numeric input which accepts values in 0..100 range

```
<form id="myapp-config">
  <div class="form-group">
    <label for="myapp-input">Numeric input</label>
    <input type="number" name="input" id="myapp-input" class="form-control" min="0"
max="100">
  </div>
</form>

<script>
(function() {
  var el = $('#myapp-config') // form element
    , input = $('#myapp-input'); // input element

  // set element values when config is loaded
  el.on('config-load', function(event, data) {
    $.each(data, function(key, value) {
      $('#myapp-' + key).val(value);
    });
  });

  // runs when Save button is clicked
  el.on('config-check', function() {
    var val = parseInt(input.val(), 10) // input value
      , min = parseInt(input.attr('min'), 10) // minimum value
      , max = parseInt(input.attr('max'), 10); // maximum value

    // invalid value
    if (isNaN(val) || val < min || max < val) {
      alert('Please enter a value between ' + min + ' and ' + max);
    }
    // all good, save configuration
    else {
      el.triggerHandler('config-save', { input: val });
    }
  });
})();
</script>
```

localStorage wrapper functions

localStorage allows saving client-side configuration. Several functions are provided to safely execute *localStorage* functions, as they might fail in some cases like private mode on iOS. It also allows storing any values that can be serialized using *JSON.stringify*.

- *storeSet(key, value)* – sets key/value pair
- *storeGet(key)* – retrieves key value, returns *null* when key is not found
- *storeRemove(key)* – removes key from storage

Storage keys must be prefixed with unique application name to minimize collisions between different applications

Examples

Get currently selected theme (light/dark)

```
var theme = storeGet('theme') || 'light';
```

Store JavaScript objects

```
var user = { name: 'John', surname: 'Doe', age: 42 };  
storeSet('myapp_user', user);
```

Translation

- `$.i18n.lang` – current language or *undefined* if default language is used
- `$.i18n.add(ns, dictionary)` – adds translations to current dictionary, *ns* must be a unique application name
- `$.i18n.translate(key, default, vars)` or `$.tr(key, default, vars)` – translates a given *key* or uses *default* value if translation is not found for current language. Additional *vars* object can be passed to replace variables inside of translation text

Example 1

```
// register translation for application "myapp"
$.i18n.add('myapp', {
  // translation for mylang
  mylang: {
    hello: 'Hello %{username}, current temperature is %{temperature}',
    goodbye: 'Goodbye %{username}'
  }
});

var text = $.tr('myapp.hello', 'No translation', { username: 'John', temperature: 21 });

// alerts "Hello John, current temperature is 21" if current language is "mylang"
// otherwise alerts "No translation"
alert(text);
```

Example 2

You can apply translation to *jQuery* selectors by using *tr* function: all HTML elements that have *tr* class and *data-tr-key* attribute will have contents replaced with translated version

HTML:

```
<div id="myapp-container">
  <span class="tr" data-tr-key="myapp.hello">Hello!</span>
</div>
```

JavaScript:

```
// register french translation
$.i18n.add('myapp', {
  fr: {
    hello: 'Bonjour!'
  }
});

// apply translation to all elements inside of myapp-container
$('#myapp-container').tr();
```

LP scripts

Allows mixing HTML and Lua inside a single file, Lua chunks must be enclosed in `<? ?>` tags, closing tag at the end of the document is not required.

Example

Print current date

```
<!DOCTYPE html>
<html>
<body>Current date is <? write(os.date()) ?></body>
</html>
```

Available functions:

- *header(hdr)* – adds a custom header to the output
- *getvar(name)* – returns named *GET/POST* variable or *nil* when variable is not set
- *getvars()* - returns all *GET/POST* variables as *Lua table*
- *getcookie(name)* – returns named *cookie* contents or *nil* when cookie is not set
- *print(...)* – outputs any number of variables, ending output with *CRLF*
- *write(...)* – similar to *print* but does not output *CRLF* at the end
- *escape(val)* – escape single/double quotes, less than/greater than characters to HTML entities

Library package is loaded via *require('apps')* and provides access to these functions:

- all built-in LM functions: *alert, log, grp, storage* etc
- *config* library
- *vprint(...)* and *vprinthex(...)* functions to view variable contents in human-readable form
- *json* library

Example

Output multiplication table. Size can be a *GET/POST* variable in 1..20 range (defaults to 10).

```
<!DOCTYPE html>
<html>
<body>
<?
size = getvar('size') -- GET/POST variable
size = tonumber(size) or 0 -- convert to number
if size < 1 or 20 < size then
    size = 10 -- set to default value if empty or invalid
end
?>
<table border="1" cellpadding="3">
<? for i = 1, size do ?>
    <tr>
```

```
<? for j = 1, size do ?>
  <td><? write(i * j) ?></td>
<? end ?>
</tr>
<? end ?>
</table>
</body>
</html>
```

Full Lua function reference manual is available at:

<http://openrb.com/docs/lua.htm>

Object functions

Most functions use *alias* parameter — either object group address or object name. (e.g. '1/1/1' or 'My object')

Finding single/multiple objects

grp.find(alias)

Returns single object for given alias. Object value will be decoded if data type is set.

Returns *nil* when object cannot be found, otherwise it returns *table* with the following items:

- address — object group address
- updatetime — latest update time in *UNIX timestamp* format. Use `os.date()` to convert to readable date formats
- name — unique object name
- datatype — object data type
- decoded — set to *true* when decoded value is available
- value — decoded object value

grp.tag(tags [, mode])

Returns a *table* containing objects with given tag. Tags parameter can be either table or a string. Mode parameter can be either 'or' (default — returns objects that have any of given tags) or 'and' (return objects that have all of given tags). You can use object functions on the returned table.

grp.dpt(dpt, [strict])

Find all objects with matching data type. *Dpt* can be either a *string* ("bool", "scale", "uint32" etc) or a field from *dt* table (`dt.bool`, `dt.scale`, `dt.uint32`). For example, if *dpt* is set to `dt.uint8`, in normal mode all sub-datatypes like `dt.scale` and `dt.angle` will be

included. If exact data type match is required, set *strict* to *true*.

grp.all()

Returns a table with all known objects.

Helpers

grp.alias(alias)

Converts group address to object name or name to address. Returns *nil* when object cannot be found.

grp.getvalue(alias)

Returns value for given alias or *nil* when object cannot be found.

Bus requests

grp.write(alias, value [, datatype])

Sends group write request to given alias. Data type is taken from the database if not specified as third parameter. Returns boolean as the result.

grp.response(alias, value [, datatype])

Similar to `grp.write`. Sends group response request to given alias.

grp.read(alias)

Sends group read request to given alias. Note: this function returns immediately and cannot be used to return the result of read request. Use event-based script instead.

grp.update (alias, value [, datatype])

Similar to `grp.write`, but does not send new value to the bus. Useful for objects that are used only in visualization.

Tag manipulation

grp.gettags(alias)

Returns a *table* with all tags that are set for given alias.

grp.addtags(alias, tags)

Adds single or multiple tags to given alias. *Tags* parameter can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags).

grp.removetags(alias, tags)

Removes single or multiple tags from given alias. *Tags* parameter can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags).

grp.removealltags(alias)

Removes all tags for given alias.

grp.settags(alias, tags)

Overwrites all tags for given alias. *Tags* parameter can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags)

Object creation and modification

grp.setcomment(alias, comment)

Sets *comment* field for given alias

grp.create(config)

Creates a new or overwrites an existing object based on provided *config*, which must be a *Lua table*. Returns object ID on success, nil plus error message otherwise.

Config fields:

- *datatype* – *required*, object data type. Can be either a *string* (“bool”, “scale”, “uint32” etc) or a field from *dt* table (dt.bool, dt.scale, dt.uint32)
- *name* – *optional*, unique object name. If an object with the same name already exists, numeric prefix will be added
- *comment* – *optional*, object comment (*string*)
- *units* – *optional*, object units/suffix (*string*)
- *address* – *optional*, object group address (*string*). If not set the first free address from configured range will be used
- *tags* – *optional*, object tags, can be either a *string* (single tags) or *Lua table* consisting of strings (multiple tags)

If an object with the same group address already exists, only *units*, *datatype* and *comment fields* will be changed. All other properties will be kept unchanged.

Examples

Create new object with known address

```
address = grp.create({
  datatype = dt.float16,
  address = '1/1/1',
  name = 'My first object',
  comment = 'This is my new object',
  units = 'W',
  tags = { 'My tag A', 'My tag B' },
})
```

Create new object with automatic address assignment


```
address = grp.create({
  datatype = dt.bool,
  name = 'My second object',
})
```

Database functions

SQLite v3 is used as the database engine.

Note: Database tables must be prefixed with unique application name to minimize collisions between different applications.

Core functions

- *db:execute(query)* – executes given query, return value can be either a database cursor or query result
- *db:escape(value)* – escapes given *string* value so it can be safely used in a query
- *db:query(query, ...)* – executes given query, question marks in the query are replaced by additional parameters (see examples below)

INSERT/UPDATE/DELETE helpers

Note: *Lua tables* passed as *values* and *where* parameters must not have fields that are not present in given database table. Otherwise query will fail

- *db:insert(tablename, values)* – performs *INSERT* query based on given *values*
- *db:update(tablename, values, where)* – performs *UPDATE* query based on given *values* and *where* parameters
- *db:delete(tablename, where)* – performs *DELETE* query based on *where* parameter

SELECT helpers

Note: parameters must be passed in the same way as for *db:query()* function

- *db:getone(query, ...)* – returns first field value from the first matching row from given query
- *db:getrow(query, ...)* – returns first matching row from given query
- *db:getlist(query, ...)* – returns complete query result as *Lua table*, where each table item is first field from each row
- *db:getall(query, ...)* – returns complete query result as *Lua table*, where each table item is *Lua table* with field→value mapping

Examples

```
-- Query parameter replacement
db:query('UPDATE table SET field=? WHERE id=?', 'test', 42)
-- Same as INSERT INTO table (id, value) VALUES (42, 'test')
db:insert('table', {
  id = 42,
  value = 'test',
})
-- Same as UPDATE table SET value='test' WHERE id=42
db:update('table', { value = 'test' }, { id = 42 })
-- Same as DELETE FROM table WHERE id=42
db:delete('table', { id = 42 })
```

6. LogicMachine configuration

Login	Password
admin	admin

This is a home directory for LogicMachine configuration management. The main menu consists of the following menus:

- **Scripting** – scripting repository management
- **Objects**– KNX bus object management
- **Object logs**– KNX bus object historical logs
- **Schedulers**– administrator interface for user mode schedulers
- **Trend logs** – administrator interface for trend logs
- **Scenes** – scene visual editor
- **Vis.structure** – visualization structure definition
- **Visualization**– Visualization creation, control and monitoring
- **Vis.graphics**– icon, background, font management
- **Utilities** – utilities including import from ETS, reset object DB, backup, update system installation
- **User access** – user access level definition
- **BACnet** – BACnet client with scanner
- **DALI** – DALI master
- **1-Wire** – 1-wire configuration
- **Modbus** – Modbus mapper
- **BLE** – Bluetooth device mapping (over USB gateway)
- **Alerts** – alert messages defined with *alert* function
- **Logs** – log messages defined with *log* function
- **Error log** – error messages in KNX bus
- **Help** – documentation for scripting syntaxes

6.1. Scripting

Scripting menu allows adding and managing various scripts, depending on the type of the script. There are two ways to program logics – blocks and via Lua programming language. Most of the Lua language aspects are covered in the first edition of "Programming in Lua" which is freely available at <http://lua.org/pil/>

Note! Here is available LUA Reference Manual for LogicMachine: <http://openrb.com/docs/lua.htm>

There are six main types of scripts:

Event-based – scripts that are executed when a group event occurs on the bus. Usually used when nearly real-time response is required.

Resident– scripts that use polling to check for object state changes. Usually used for heating and ventilation when data is gathered from more than one group address.

Scheduled– scripts that run at the required time and day. Can be used for various security systems and presence simulations.

User libraries – user defined scripts to call from other scripts

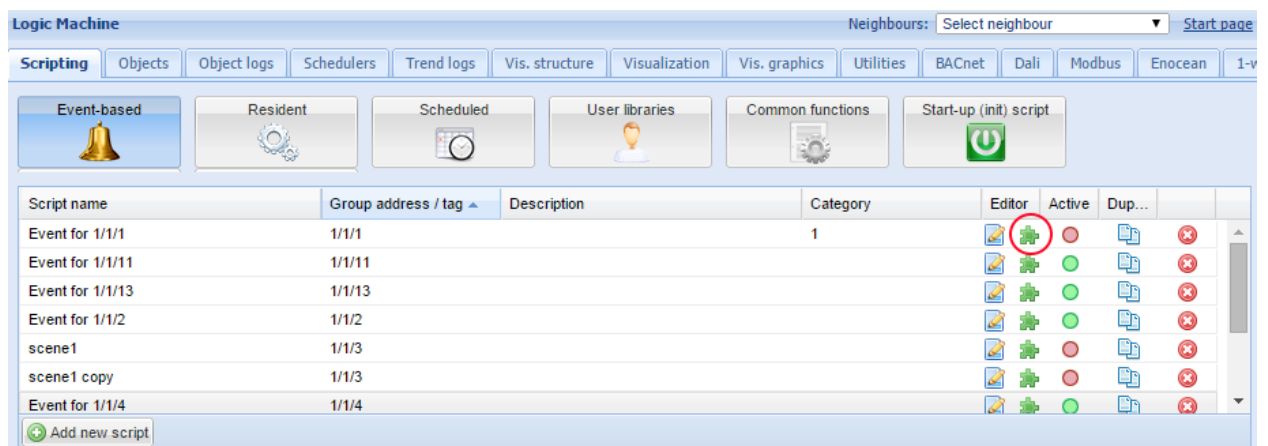
Common functions – common functions to call from other scripts

Start-up (init) script – initialization script that is run upon system starting.

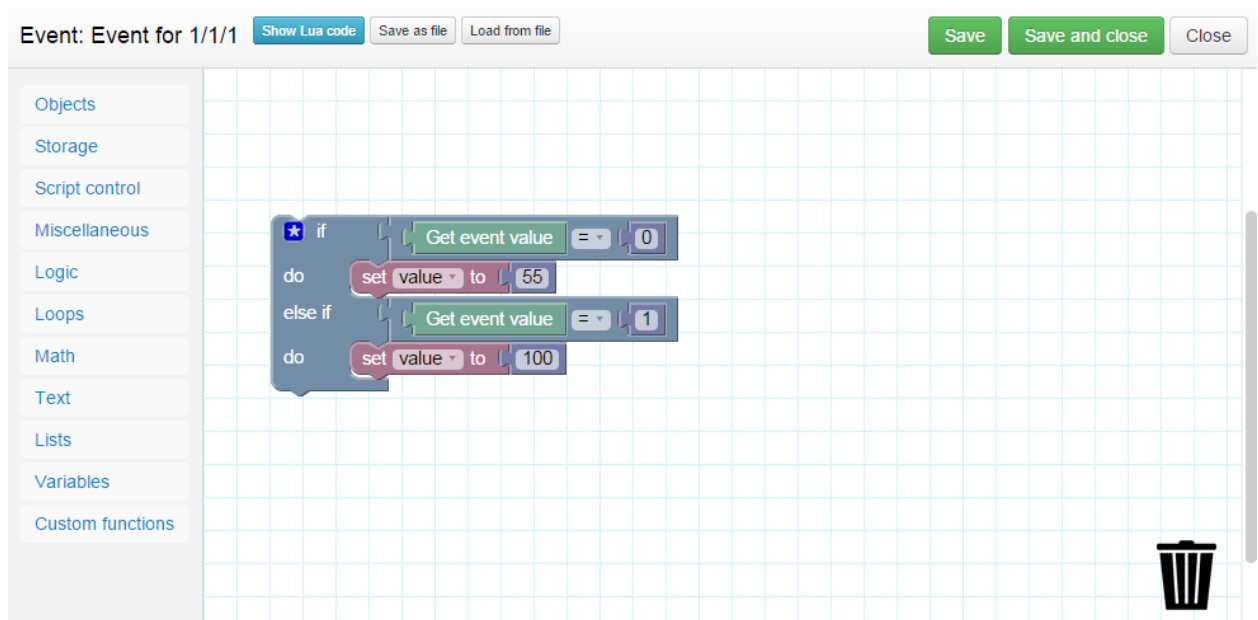
6.1.1. Block programming

In order to create blocks, enable this functionality in *Utilities* → *General configuration* → *Enable Block Editor*.

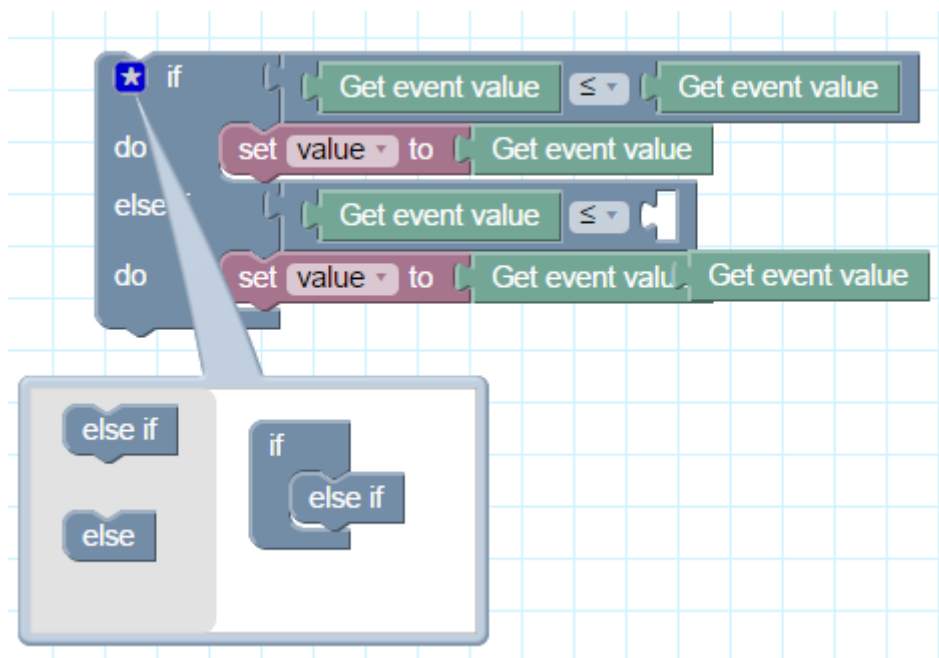
Once the script is added, you can see puzzle icon to access Block editor.



Blocks are sorted by categories on the left side. Each block is puzzle based and can be put only in appropriate location / other block.



If the block is indicated with the blue label on the top left corner, you can define the structure of the block (e.g. If Else)



Press Delete button or drag the block to the garbage if you want to delete it



You can always look at the LUA code by clicking on *Show/Hide Lua code* button. This will allow to learn the scripting language.

```
Event: Event for 1/1/1 Hide Lua code Save as file Load from file Save Save and close Close  
1 if event.getvalue() <= event.getvalue() then  
2   value = event.getvalue()  
3 elseif event.getvalue() <= event.getvalue() then  
4   value = event.getvalue()  
5 end  
6
```

6.1.2. Block functions

In Scripting menu there is *Block functions* button. Here you can create custom block functions which can be later used as ready block in Block editor.

Each function must have a special comment in order to be converted to a block.

- First line must have **Function** keyword followed by the function name
- Second line contains short function description which is shown as block title

- If third line contains **Comment** keyword, all following lines until Input or Output will be added to block comment tooltip
- Optionally, block color may be specified in hexadecimal format (#f00 or #ff9900) or numeric format as hue value between 0 and 359
- Following lines contain input and output lists. Each block can have any number of inputs and outputs:
Inputs are a function parameter, other blocks can be connected to inputs by default. If input definition has **[object]**, **[storage]** or **[tag]** in its name then the input is replaced with object, storage or tag selection input.

Each output variable is assigned to the corresponding function return value.

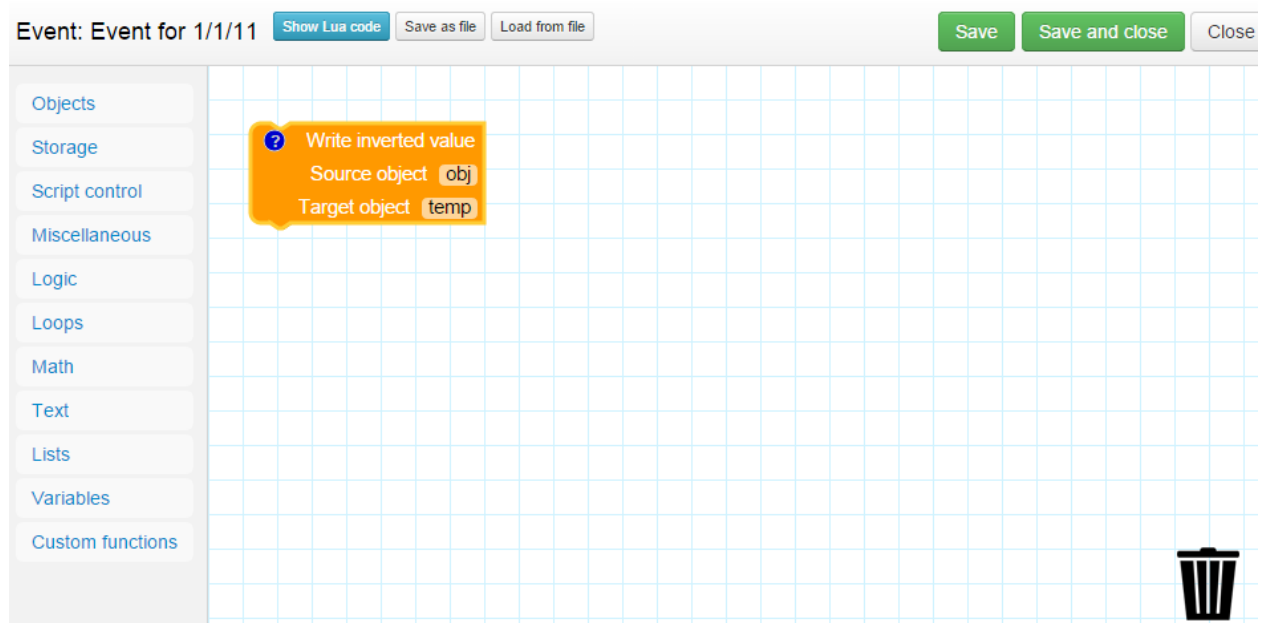
Example:

```

--- Function invert
--- Write inverted value
--- Comment
--- Set target object value to
--- inverse of source object value
--- Color #f90
--- Input
--- Source object [object]
--- Target object [object]
function invert(a, b)
local value = grp.getvalue(a)
grp.write(b, not value, dt.bool)
end

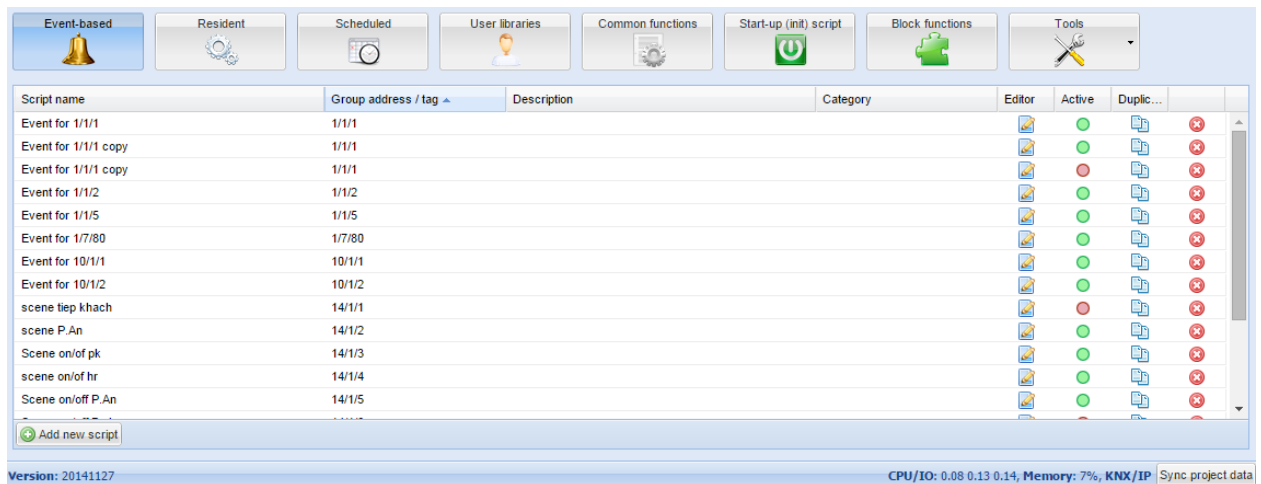
```

Once block function is added, it is available as a block in Block editor.



6.1.3. Adding a new script

Click on *Add new script* button on the bottom part of the *Event-based*, *Resident* or *Scheduled* submenus



The following fields should be filled when adding a new script:

Event-based

Event-based script

Script name: Scene_away

Group address / tag: 2/2/2

Active:

Execute on group read:

Category: [Dropdown]

Description: [Text Area]

Save Cancel

- **Script name** – the name of the script
- **Group address / Tag** – specific group address or tag name on which the script will be triggered
- **Active**– specifies whether the script is active (green circle) or disabled (red circle)
- **Execute on group read**– specifies whether the script is executed on KNX group read telegram

- **Category** – a new or existing name of the category the script will be included. This will not affect on script action, helps only by grouping the scripts and watching by categories in *Tools* ☞ *Print* script listings page
- **Description**– description of the script

Resident

Resident script

Script name:

Sleep interval (seconds):

Active:

Category:

Description:

- **Script name** – the name of the script
- **Sleep interval (seconds)** – interval after which the script will be executed.
- **Active**– specifies whether the script is active (green circle) or disabled (red circle)
- **Category** – a new or existing name of the category the script will be included. This will not affect on script action, helps only by grouping the scripts and watching by categories in *Tools* ☞ *Print* script listings page
- **Description**– description of the script

Scheduled

Scheduled script

Script name:

Minute:

Hour:

Day of the month:

Month of the year:

Day of the week:

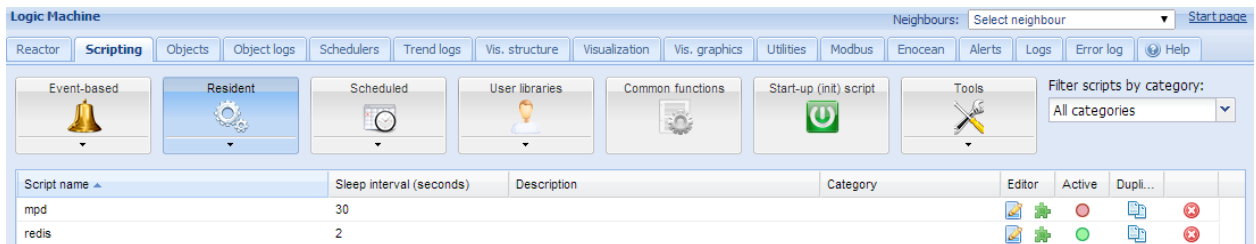
Active:

Category:

Description:

- **Script name** – the name of the script
- **Minute** – Minute
- **Hour** – Hour
- **Day of the month** – Day of the month
- **Month of the year** – Month of the year
- **Day of the week** – Day of the week
- **Active**– specifies whether the script is active (green circle) or disabled (red circle)
- **Category** – a new or existing name of the category the script will be included. This will not affect on script action, helps only by grouping the scripts and watching by categories in *Tools* [Print script listings page](#)
- **Description**– description of the script

List of scripts



There are five actions you can do with each of the script:

Duplicate – Duplicate the script with its source code

Editor – Enter scripting editor to write specific code for the particular program. It can be source code editor or block programming

Active – Make script active (green) or deactivate it (red)

Delete – Delete the script. When pressing this icon the confirmation is asked to accept the delete.

6.1.4. Event-based scripting

Event-based scripting can be used to implement custom logic for group address or tag events. User-defined function is executed when a "group write" or "group read" (if checked while adding the script) event occurs for given group address. Event information is stored in global **event** variable. Variable contents:

- *dstraw (integer)* — raw destination group address
- *srcraw (integer)* — raw source individual address
- *dst (string)* — decoded destination group address (for example: 1/1/4)
- *src (string)* — decoded source individual address (for example: 1.1.2)
- *type (string)* — type of event, either "groupwrite", "groupread", "groupresponse". Currently user-defined scripts are bound to "group write" events only.

- *dataraw (integer/string)* — raw binary data
- *datahex (string)* — data as a hex-encoded string which can be used to convert value to Lua variable

Note! *event* variable is available only in Event-based functions, not in Resident and Scheduled.

Note! All event-based scripts are executed in a single queue-like manner. Make sure event scripts do not contain infinite loops, sleep calls or other blocking parts.

Note! To get event value in scripts, use the following command: **a = event.getvalue()**

Note! To get event group address object name, use the following command:
a = grp.alias(event.dst)

6.1.5. Resident scripting

Resident scripts are executed infinite amount of times. Scripts are put into inactive state after each call and are resumed after delay timer expires.

Note! even though resident scripts are executed in parallel they should not have infinite loops or it will not be possible to reload scripts after editing.

6.1.6. Scheduled scripting

Scheduled scripts are executed when the system time matches the specified script start time. Scheduled script is run only once after each timer call.

Scheduled scripting date/time format

Scheduled scripting uses standard [cron](#) format for date/time parameters. Valid values are:

***** — execute script every minute, hour or day.


***/N** — execute script every N minutes, hours or days. N is an integer, script is executed when current value divided by N gives 0 in modulo. For example, script with hour parameter set to */8 will be executed when hour is 0, 8 and 16.

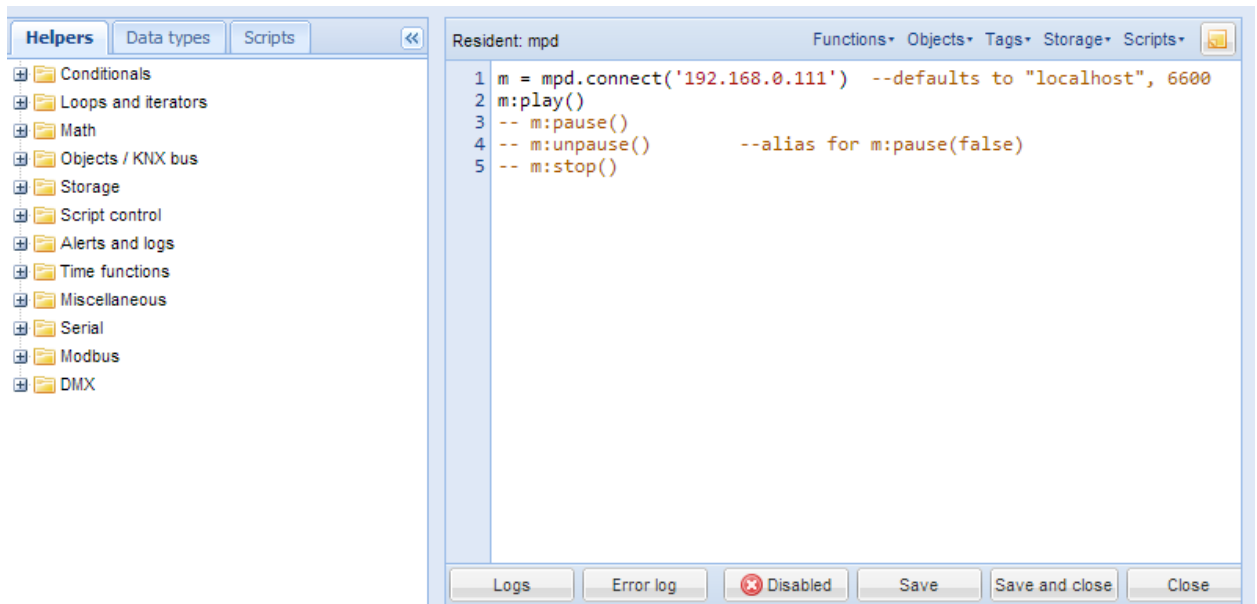
N — execute script exactly at N minute, hour or day.

N-K — execute script when minute, hour or day is between N-K range (inclusive).

N,K — it is possible to specify several N and N-K type parameters separated by comma. For example, script with minute parameter set to 15,50-52 will get executed when minute is 15, 50, 51 and 52

6.1.7. Script editor

When a script is added  icon appears in *Editor* column that allows opening a script in scripting editor and re-working it with built-in code snippets.



The idea is that not knowing the syntaxes you get a helper for writing your own scripts. Code snippets save also a time and make the coding much more convenient. After clicking on appropriate snippet, it automatically adds code to the editor field.

There are three main groups of Script editor:

Helpers – predefined code snippets, like if-then statement. Helpers consist of three main sub-groups:

Conditionals – If Else If, If Then etc.

Loops and iterators – Array, Repeat..Untiletc

Math – Random value, Ceiling, Absolute value, Round etc.

Objects/KNX bus – Get object value, Group read, Group write, Update interval etc.

Storage – Get data from storage, Save data to storage

Script control – Get other script status, enable or disable other scripts

Alerts and logs – Alert, Log variables, Formatted alert

Time functions – Delay script execution

Miscellaneous – Sunrise/sunset etc.

Serial – Communication through internal LogicMachine IO ports

Modbus – Create RTU/TCP connection, Write register, Read register etc.

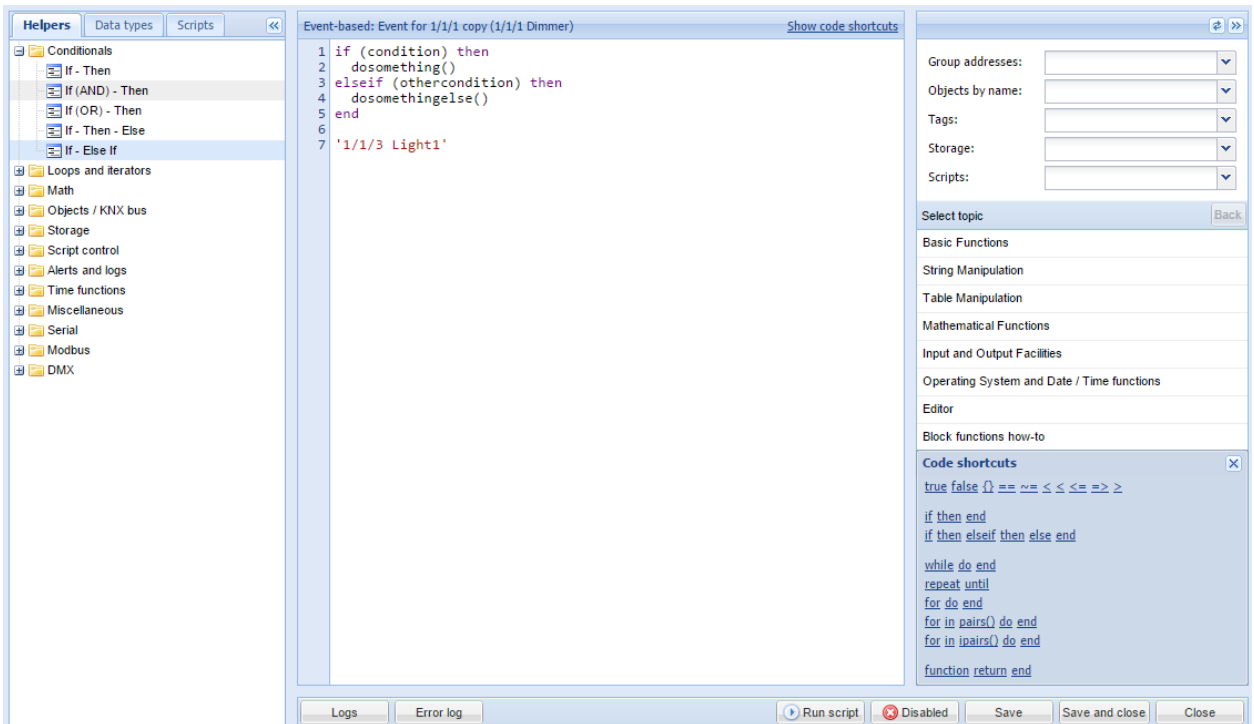
DMX – Communication with DMX devices

Data types – choose object by data type

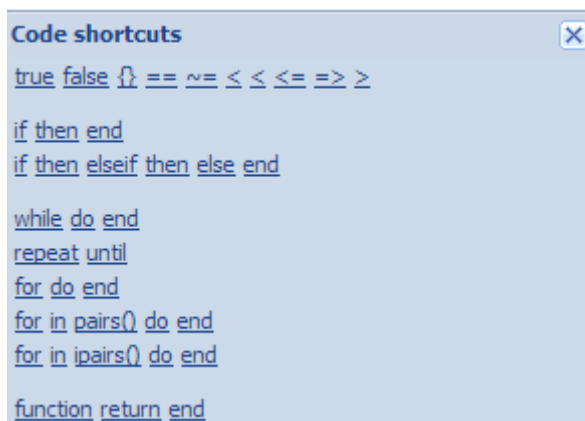
Scripts – list of all scripts added in the LogicMachine

Code helpers on the right side of the editor

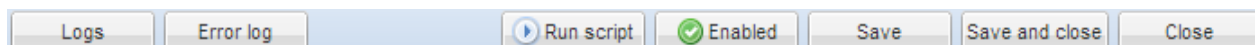
There is a special section in scripting editor which allows quickly find functions, objects or tags by name and storage variables.



There is also a code shortcut button, which helps with most common function structure.



There are also following helpful button in the script editor, which allows quickly access Error Logs, Test the script, Enable or disable it.



6.1.8. Object functions

grp provides simplified access to the objects stored in the database and group address request helpers.

Most functions use `alias` parameter — object group address or unique object name. (e.g. '1/1/1' or 'My object')

grp.getvalue(alias)

Returns value for the given alias or Lua `nil` when object cannot be found.

grp.find(alias)

Returns single object for the given alias. Object value will be decoded automatically only if the data type has been specified in the 'Objects' module. Returns Lua `nil` when object cannot be found, otherwise it returns Lua `table` with the following items:

- `address` — object group address
- `updatetime` — latest update time in UNIX timestamp format. Use Lua `os.date()` to convert to readable date formats

When object data type has been specified in the 'Objects' module the following fields are available:

- `name` — unique object name
- `datatype` — object data type as specified by user
- `decoded` — set to `true` when decoded value is available
- `value` — decoded object value

grp.tag(tags, mode)

Returns Lua `table` containing objects with the given tag. Tags parameter can be either Lua `table` or a string. Mode parameter can be either 'all' (return objects that have all of the given tags) or 'any' (`default` — returns objects that have any of the given tags). You can use *Returned object functions* on the returned table.

grp.alias(alias)

Converts group address to object name or name to address. Returns Lua `nil` when object cannot be found.

6.1.9. Returned object functions, group communication functions

Objects received by using `grp.find(alias)` or `grp.tag(tags, mode)` have the following functions attached to them:

Always check that the returned object was found otherwise calling these functions will result in an error. See the example below.

object:write(value, datatype)

Sends group write request to object's group address. Data type is taken from the database if not specified as second parameter. Returns Lua `boolean` as the result.

object:response(value, datatype)

Similar to `object:write`. Sends group response request to object's group address.

object:read()

Sends group read request to object's group address. Note: this function returns immediately and cannot be used to return the result of read request. Use event-based script instead.

object:update(value, datatype)

Similar to `object:write`, but **does not send** new value to the bus. Useful for objects that are used only in visualization.

6.1.10. Group communication functions

These functions should only be used if it is required to access objects by group address directly, it is recommended to use single or multiple object functions.

grp.write(alias, value, datatype)

Sends group write request to the given alias. Data type is taken from the database if not specified as third parameter. Returns Lua `boolean` as the result.

grp.response(alias, value, datatype)

Similar to `grp.write`. Sends group response request to the given alias.

grp.read(alias)

Sends group read request to the given alias. Note: this function returns immediately and cannot be used to return the result of read request. Use event-based script instead.

grp.update(alias, value, datatype)

Similar to `grp.write`, but **does not send** new value to the bus. Useful for objects that are used only in visualization.

6.1.11. Object function examples

Find object by name and write new value.

```
1. myobject=grp.find('My object')
2. -- grp.find will return nil if object was not found
3. if myobject then
4. myobject:write(1)-- update object value with 1
5. end
```

Find object by address and write new value.

```
1. myobject=grp.find('1/1/15')
```

2. -- verify that the requested object was found
3. **if** myobject **then**
4. myobject:write(52.12, dt.float16)-- explicitly set data type to dt.float16 (2-byte floating point)
5. **end**

Switch all binary objects tagged 'lights' off.

1. lights =grp.tag('lights')
2. lights:write(false)

Group write to the specified group address and data type.

1. grp.write('1/1/1', true, dt.bool)-- write 1-bit 'on' to 1/1/1
2. grp.write('1/1/2', 50, dt.scale)-- write 1-byte 50% to 1/1/2

6.1.12. Data type functions, data types

knxdatatype object provides data encoding and decoding between Lua and KNX data formats.

knxdatatype.decode(value, datatype)

Converts hex-encoded data to Lua variable based on given data type. Data type is specified either as KNX primary data type (integer between 1 and 16) or a secondary data type (integer between 1000 and 16000). Return values:

- success — decoded data as Lua variable (type depends on data type), value length in bytes
- error — nil, error string

6.1.13. Data types

The following data types can be used for encoding and decoding of KNX data. Data representation on Lua level and predefined constants (in bold) is given below:

- 1 bit (boolean) - **dt.bool** — boolean
- 2 bit (1 bit controlled) - **dt.bit2** — number
- 4 bit (3 bit controlled) - **dt.bit4** — number
- 1 byte ASCII character - **dt.char** — string
- 1 byte unsigned integer - **dt.uint8** — number
- 1 byte signed integer - **dt.int8** — number
- 2 byte unsigned integer - **dt.uint16** — number
- 2 byte signed integer - **dt.int16** — number
- 2 byte floating point - **dt.float16** — number
- 3 byte time / day - **dt.time** — table with the following items:
 - day — number (0-7)

- hour — number (0-23)
- minute — number (0-59)
- second — number (0-59)
- **3 byte date - `dt.date`** — table with the following items:
 - day — number (1-31)
 - month — number (1-12)
 - year — number (1990-2089)
- **4 byte unsigned integer - `dt.uint32`** — number
- **4 byte signed integer - `dt.int32`** — number
- **4 byte floating point - `dt.float32`** — number
- **4 byte access control - `dt.access`** — number, currently not fully supported
- **14 byte ASCII string - `dt.string`** — string, null characters ('\0') are discarded during decoding

6.1.14. Data storage function

storage object provides persistent key-value data storage for user scripts. Only the following Lua data types are supported:

- boolean
- number
- string
- table

`storage.set(key, value)`

Sets new value for the given key. Old value is overwritten. Returns boolean as the result and an optional error string.

`storage.get(key, default)`

Gets value for the given key or returns default value (`nil` if not specified) if key is not found in the data storage.

Note: all user scripts share the same data storage. Make sure that same keys are not used to store different types of data.

Examples

- The following examples shows the basic syntax of `storage.set`. Result will return boolean `true` since the passed parameters are correct

```
result=storage.set('my_stored_value_1', 12.21)
```

- This example will return `false` as the result because we are trying to store a function which is not possible.

```

1. testfn=function(t)
2. return t * t
3. end
4. result =storage.set('my_stored_value_2', testfn)-- this will result in an error

```

- The following examples shows the basic syntax of `storage.get`. Assuming that key value was not found, first call will return `nil` while second call will return number `0` which was specified as a default value.

```

1. result =storage.get('my_stored_value_3')-- returns nil if value is not found
2. result =storage.get('my_stored_value_3', 0)-- returns 0 if value is not found

```

- When storing tables make sure to check the returned result type. Assume we have created a storage item with key `test_object_data`.

```

1. objectdata={}
2. objectdata.temperature=23.1
3. objectdata.scene='default'
4. result =storage.set('test_object_data', objectdata)-- store objectdata variable as
   'test_object_data'

```

- Now we are retrieving data from storage. Data type is checked for correctness.

```

1. objectdata=storage.get('test_object_data')
2. if type(objectdata)=='table' then
3. if objectdata.temperature> 24 then
4. -- do something if temperature level is too high
5. end
6. end

```

6.1.15. Alert function

`alert(message, [var1, [var2, [var3]]])`

Stores alert message and current system time in the main database. All alerts are accessible in the "Alerts" module. This function behaves exactly as Lua `string.format`.

Example

```

1. temperature = 25.3
2. if temperature > 24 then
3. -- resulting message: 'Temperature levels are too high: 25.3'
4. alert('Temperature level is too high: %.1f', temperature)
5. end

```

6.1.16. Log function

`log(var1, [var2, [var3, ...]])`

Converts variables to human-readable form and stores them in the main database. All items are accessible in the "Logs" module.

Example

```
1. -- Log function accepts Lua nil, boolean, number and table (up to 5 nested levels) type
   variables
2. a = { key1 = 'value1', key2 = 2}
3. b = 'test'
4. c = 123.45
5. -- Logs all passed variables
6. log(a, b, c)
```

6.1.17. Scheduled scripting date/time format

Scheduled scripting uses standard *cron* format for date/time parameters. Valid values are:

- * — execute script every minute, hour or day.
- */N — execute script every N minutes, hours or days. N is an integer, script is executed when current value divided by N gives 0 in modulo. For example, script with hour parameter set to */8 will be executed when hour is 0, 8 and 16.
- N — execute script exactly at N minute, hour or day.
- N-K — execute script when minute, hour or day is between N-K range (inclusive).
- N,K — it is possible to specify several N and N-K type parameters separated by comma. For example, script with minute parameter set to 15,50-52 will get executed when minute is 15, 50, 51 and 52

6.1.18. Time function

`os.sleep(delay)`

Delay the next command execution for the `delay` seconds.

`os.microtime ()`

Returns two values: current timestamp in seconds and timestamp fraction in nanoseconds

`os.udifftime (sec, usec)`

Returns time difference as floating point value between now and timestamp components passed to this function (seconds, nanoseconds)

6.1.19. *Data Serialization*

serialize.encode (value)

Generates a storable representation of a value.

serialize.decode (value)

Creates a Lua value from a stored representation.

6.1.20. *String functions*

This library provides generic functions for string manipulation, such as finding and extracting substrings, and pattern matching. When indexing a string in Lua, the first character is at position 1 (not at 0, as in C).

Indices are allowed to be negative and are interpreted as indexing backwards, from the end of the string. Thus, the last character is at position -1, and so on.

The string library provides all its functions inside the table `string`. It also sets a meta table for strings where the `__index` field points to the string table. Therefore, you can use the string functions in object-oriented style. For instance, `string.byte(s, i)` can be written as `s:byte(i)`. The string library assumes one-byte character encodings.

string.trim (str)

Trims the leading and trailing spaces off a given string.

string.split (str, sep)

Splits string by given separator string. Returns Lua table.

string.byte (s [, i [, j]])

Returns the internal numerical codes of the characters `s[i]`, `s[i+1]`, ..., `s[j]`. The default value for `i` is 1; the default value for `j` is `i`. Note that numerical codes are not necessarily portable across platforms.

string.char (...)

Receives zero or more integers. Returns a string with length equal to the number of arguments, in which each character has the internal numerical code equal to its corresponding argument. Note that numerical codes are not necessarily portable across platforms.

string.find (s, pattern [, init [, plain]])

Looks for the first match of `pattern` in the string `s`. If it finds a match, then `find` returns the indices of `s` where this occurrence starts and ends; otherwise, it returns `nil`. A third, optional numerical argument `init` specifies where to start the search; its default value is 1 and can be negative. A value of `true` as a fourth, optional argument `plain` turns off the pattern matching facilities, so the function does a plain "find substring" operation, with no characters in `pattern` being considered "magic". Note that if `plain` is given, then `init` must be given as well. If the `pattern` has captures, then in a successful match the captured values are also returned, after the two indices.

string.format (formatstring, ...)

Returns a formatted version of its variable number of arguments following the description given in its first argument (which must be a string). The format string follows the same rules as the printf family of standard C functions. The only differences are that the options/modifiers *, l, L, n, p, and h are not supported and that there is an extra option, q. The q option formats a string in a form suitable to be safely read back by the Lua interpreter: the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written. For instance, the call

```
string.format('%q', 'a string with "quotes" and \n new line')
```

will produce the string:

```
"a string with \"quotes\" and \n new line"
```

The options *c*, *d*, *E*, *e*, *f*, *g*, *G*, *i*, *o*, *u*, *X*, and *x* all expect a number as argument, whereas *q* and *s* expect a string. This function does not accept string values containing embedded zeros, except as arguments to the *q* option.

string.gmatch (s, pattern)

Returns an iterator function that, each time it is called, returns the next captures from pattern over strings. If pattern specifies no captures, then the whole match is produced in each call. As an example, the following loop

```
1. s = "hello world from Lua"
2. for w in string.gmatch(s, "%a+") do
3.   print(w)
4. end
```

will iterate over all the words from string *s*, printing one per line. The next example collects all pairs *key=value* from the given string into a table:

```
1. t = {}
2. s = "from=world, to=Lua"
3. for k, v in string.gmatch(s, "(%w+)=(%w+)") do
4.   t[k] = v
5. end
```

For this function, a '^' at the start of a pattern does not work as an anchor, as this would prevent the iteration.

string.gsub (s, pattern, repl [, n])

Returns a copy of *s* in which all (or the first *n*, if given) occurrences of the pattern have been replaced by a replacement string specified by *repl*, which can be a string, a table, or a function. *gsub* also returns, as its second value, the total number of matches that occurred.

If *repl* is a string, then its value is used for replacement. The character % works as an escape character: any sequence in *repl* of the form %*n*, with *n* between 1 and 9, stands for the value of the *n*-th capture string (see below). The sequence %0 stands for the whole match. The sequence %% stands for a single %.

If *repl* is a table, then the table is queried for every match, using the first capture as the key; if the pattern specifies no captures, then the whole match is used as the key.

If *repl* is a function, then this function is called every time a match occurs, with all captured substrings passed as arguments, in order; if the pattern specifies no captures, then the whole match is passed as a sole argument.

If the value returned by the table query or by the function call is a string or a number, then it is used as the replacement string; otherwise, if it is *false* or *nil*, then there is no replacement (that is, the original match is kept in the string).

Examples:

```
x =string.gsub("hello world", "(%w+)", "%1 %1")
--> x="hello hello world world"

x =string.gsub("hello world", "%w+", "%0 %0", 1)
--> x="hello hello world"

x =string.gsub("hello world from Lua", "(%w+)%s*(%w+)", "%2 %1")
--> x="world hello Lua from"

x =string.gsub("home = $HOME, user = $USER", "%$(%w+)", os.getenv)
--> x="home = /home/roberto, user = roberto"

x =string.gsub("4+5 = $return 4+5$", "%$(.-)%$", function(s)
returnloadstring(s)()
end)
--> x="4+5 = 9"

local t ={name="lua", version="5.1"}
x =string.gsub("$name-$version.tar.gz", "%$(%w+)", t)
--> x="lua-5.1.tar.gz"
```

string.len (s)

Receives a string and returns its length. The empty string "" has length 0. Embedded zeros are counted, so "a\000bc\000" has length 5.

string.lower (s)

Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged. The definition of what an uppercase letter is depends on the current locale.

string.match (s, pattern [, init])

Looks for the first match of pattern in the string s. If it finds one, then match returns the captures from the pattern; otherwise it returns *nil*. If pattern specifies no captures, then the whole match is returned. A third, optional numerical argument *init* specifies where to start the search; its default value is 1 and can be negative.

string.rep (s, n)

Returns a string that is the concatenation of n copies of the string s.

string.reverse (s)

Returns a string that is the string s reversed.

string.sub (s, i [, j])

Returns the substring of s that starts at i and continues until j; i and j can be negative. If j is absent, then it is assumed to be equal to -1 (which is the same as the string length). In

particular, the call `string.sub(s,1,j)` returns a prefix of `s` with length `j`, and `string.sub(s, -i)` returns a suffix of `s` with length `i`.

string.upper (s)

Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged. The definition of what a lowercase letter is depends on the current locale.

Patterns

Character Class:

A character class is used to represent a set of characters. The following combinations are allowed in describing a character class:

- **x**: (where `x` is not one of the magic characters `^$()%.[]*+-?`) represents the character `x` itself.
- **.**: (a dot) represents all characters.
- **%a**: represents all letters.
- **%c**: represents all control characters.
- **%d**: represents all digits.
- **%l**: represents all lowercase letters.
- **%p**: represents all punctuation characters.
- **%s**: represents all space characters.
- **%u**: represents all uppercase letters.
- **%w**: represents all alphanumeric characters.
- **%x**: represents all hexadecimal digits.
- **%z**: represents the character with representation 0.
- **%x**: (where `x` is any non-alphanumeric character) represents the character `x`. This is the standard way to escape the magic characters. Any punctuation character (even the non magic) can be preceded by a `'%'` when used to represent itself in a pattern.
- **[set]**: represents the class which is the union of all characters in `set`. A range of characters can be specified by separating the end characters of the range with a `'-'`. All classes `%x` described above can also be used as components in `set`. All other characters in `set` represent themselves. For example, `[%w_]` (or `[_%w]`) represents all alphanumeric characters plus the underscore, `[0-7]` represents the octal digits, and `[0-7%l%-]` represents the octal digits plus the lowercase letters plus the `'-'` character.
- The interaction between ranges and classes is not defined. Therefore, patterns like `[%a-z]` or `[a-%%]` have no meaning.
- **^[set]**: represents the complement of `set`, where `set` is interpreted as above.

For all classes represented by single letters (`%a`, `%c`, etc.), the corresponding uppercase letter represents the complement of the class. For instance, `%S` represents all non-space characters. The definitions of letter, space, and other character groups depend on the current locale. In particular, the class `[a-z]` may not be equivalent to `%l`.

Pattern Item:

A pattern item can be:

- a single character class, which matches any single character in the class;
- a single character class followed by `'*'`, which matches 0 or more repetitions of characters in the class. These repetition items will always match the longest possible sequence;

- a single character class followed by '+', which matches 1 or more repetitions of characters in the class. These repetition items will always match the longest possible sequence;
- a single character class followed by '-', which also matches 0 or more repetitions of characters in the class. Unlike '*', these repetition items will always match the shortest possible sequence;
- a single character class followed by '?', which matches 0 or 1 occurrence of a character in the class;
- %n, for n between 1 and 9; such item matches a substring equal to the n-th captured string (see below);
- %bxy, where x and y are two distinct characters; such item matches strings that start with x, end with y, and where the x and y are balanced. This means that, if one reads the string from left to right, counting +1 for an x and -1 for a y, the ending y is the first y where the count reaches 0. For instance, the item %b() matches expressions with balanced parentheses.

Pattern:

A pattern is a sequence of pattern items. A '^' at the beginning of a pattern anchors the match at the beginning of the subject string. A '\$' at the end of a pattern anchors the match at the end of the subject string. At other positions, '^' and '\$' have no special meaning and represent themselves.

Captures:

A pattern can contain sub-patterns enclosed in parentheses; they describe captures. When a match succeeds, the substrings of the subject string that match captures are stored (captured) for future use. Captures are numbered according to their left parentheses. For instance, in the pattern "(a*(.)%w(%s*))", the part of the string matching "a*(.)%w(%s*)" is stored as the first capture (and therefore has number 1); the character matching "." is captured with number 2, and the part matching "%s*" has number 3.

As a special case, the empty capture () captures the current string position (a number). For instance, if we apply the pattern "()aa()" on the string "flaaap", there will be two captures: 3 and 5. A pattern cannot contain embedded zeros. Use %z instead.

6.1.21. Input and output functions

io.exists (path)

Checks if given path (file or directory) exists. Return boolean.

io.readfile (file)

Reads whole file at once. Return file contents as a string on success or nil on error.

io.writefile (file, data)

Writes given data to a file. Data can be either a value convertible to string or a table of such values. When data is a table then each table item is terminated by a new line character. Return boolean as write result when file can be open for writing or nil when file cannot be accessed.

Example: Write event status to log file located on plugged USB flash drive:


```

1. value = knxdatatype.decode(event.datahex, dt.bool)
2. data =string.format('%s value is %s', os.date('%c'), tostring(value))
3. -- write to the end of log file preserving all previous data
4. file =io.open('/mnt/usb/log.txt', 'a+')
5. file:write(data .. '\r\n')
6. file:close()

```

Output:

```

Mon Jan 3 05:25:13 2011 value is false
Mon Jan 3 05:25:14 2011 value is true
Mon Jan 3 05:25:32 2011 value is false
Mon Jan 3 05:25:33 2011 value is true

```

Example: Read data from file (config in format key=value)

```

1. for line inio.lines('/mnt/usb/config.txt')do
2. -- split line by '=' sing
3. items = line:split('=')
4. -- two items, line seems to be valid
5. if #items == 2 then
6. key = items[1]:trim()
7. value = items[2]:trim()
8. alert('[config] %s = %s', key, value)
9. end
10. end

```

6.1.22. Script control functions

script.enable('scriptname')

Enable the script with the name scriptname.

script.disable('scriptname')

Disable the script with the name scriptname.

status = script.status('scriptname')

Returns true/false if script is found, nil otherwise

6.1.23. JSON library

Note: json is not loaded by default, use *require('json')* before calling any functions from this library.

json.encode (value)

Converts Lua variable to JSON string. Script execution is stopped in case of an error.

json.pencode (value)

Converts Lua variable to JSON string in protected mode, returns nil on error.

json.decode (value)

Converts JSON string to Lua variable. Script execution is stopped in case of an error.

json.pdecode (value)

Converts JSON string to Lua variable in protected mode, returns nil on error.

6.1.24. Conversion

Compatibility layer: *lmc core* is an alias of *cnv*.

cnv.strtohex (str)

Converts given binary string to a hex-encoded string.

cnv.hextostr (hex [, keepnulls])

Converts given hex-encoded string to a binary string. NULL characters are ignored by default, but can be included by setting second parameter to true.

cnv.tonumber (value)

Converts the given value to number using following rules: numbers and valid numeric strings are treated as is, boolean *true* is 1, boolean *false* is 0, everything else is *nil*.

cnv.hextoint(hexvalue, bytes)

Converts the given hex string to an integer of a given length in bytes.

cnv.inttohex(intvalue, bytes)

Converts the given integer to a hex string of given bytes.

cnv.strtohex(str)

Converts the given binary string to a hex-encoded string.

cnv.hextostr(hexstr)

Converts the given hex-encoded string to a binary string.

6.1.25. Bit operators

bit.bnot (value)

Binary not

bit.band (x1 [, x2...])

Binary and between any number of variables

bit.bor (x1 [, x2...])

Binary and between any number of variables

bit.bxor (x1 [, x2...])

Binary and between any number of variables

bit.lshift (value, shift)

Left binary shift

bit.rshift (value, shift)

Right binary shift

6.1.26. Input and Output Facilities

The I/O library provides two different styles for file manipulation. The first one uses implicit file descriptors; that is, there are operations to set a default input file and a default output file, and all input/output operations are over these default files. The second style uses explicit file descriptors.

When using implicit file descriptors, all operations are supplied by table *io*. When using explicit file descriptors, the operation *io.open* returns a file descriptor and then all operations are supplied as methods of the file descriptor.

The table *io* also provides three predefined file descriptors with their usual meanings from C: *io.stdin*, *io.stdout*, and *io.stderr*. The I/O library never closes these files.

Unless otherwise stated, all I/O functions return *nil* on failure (plus an error message as a second result and a system-dependent error code as a third result) and some value different from *nil* on success.

io.close ([file])

Equivalent to *file:close()*. Without a file, closes the default output file.

io.flush ()

Equivalent to *file:flush* over the default output file.

io.input ([file])

When called with a file name, it opens the named file (in text mode), and sets its handle as the default input file. When called with a file handle, it simply sets this file handle as the default input file. When called without parameters, it returns the current default input file. In case of errors this function raises the error, instead of returning an error code.

io.lines ([filename])

Opens the given file name in read mode and returns an iterator function that, each time it is called, returns a new line from the file. Therefore, the construction

```
for line in io.lines(filename) do body end
```

will iterate over all lines of the file. When the iterator function detects the end of file, it returns nil (to finish the loop) and automatically closes the file.

The call `io.lines()` (with no file name) is equivalent to `io.input():lines()`; that is, it iterates over the lines of the default input file. In this case it does not close the file when the loop ends.

io.open (filename [, mode])

This function opens a file, in the mode specified in the string mode. It returns a new file handle, or, in case of errors, nil plus an error message. The mode string can be any of the following:

- "r": read mode (the default);
- "w": write mode;
- "a": append mode;
- "r+": update mode, all previous data is preserved;
- "w+": update mode, all previous data is erased;
- "a+": append update mode, previous data is preserved, writing is only allowed at the end of file.

The mode string can also have a 'b' at the end, which is needed in some systems to open the file in binary mode. This string is exactly what is used in the standard C function *fopen*.

io.output ([file])

Similar to `io.input`, but operates over the default output file.

6.1.27. Mathematical functions

This library is an interface to the standard C math library. It provides all its functions inside the table `math`.

math.abs (x)

Returns the absolute value of x.

math.acos (x)

Returns the arc cosine of x (in radians).

math.asin (x)

Returns the arc sine of x (in radians).

math.atan (x)

Returns the arc tangent of x (in radians).

math.atan2 (y, x)

Returns the arc tangent of y/x (in radians), but uses the signs of both parameters to find the quadrant of the result. (It also handles correctly the case of x being zero.)

math.ceil (x)

Returns the smallest integer larger than or equal to x.

math.cos (x)

Returns the cosine of x (assumed to be in radians).

math.cosh (x)

Returns the hyperbolic cosine of x .

math.deg (x)

Returns the angle x (given in radians) in degrees.

math.exp (x)

Returns the value e^x .

math.floor (x)

Returns the largest integer smaller than or equal to x .

math.fmod (x, y)

Returns the remainder of the division of x by y that rounds the quotient towards zero.

math.frexp (x)

Returns m and e such that $x = m2^e$, e is an integer and the absolute value of m is in the range $[0.5, 1)$ (or zero when x is zero).

math.huge

The value HUGE_VAL, a value larger than or equal to any other numerical value.

math.ldexp (m, e)

Returns $m2^e$, (e should be an integer).

math.log (x)

Returns the natural logarithm of x .

math.log10 (x)

Returns the base-10 logarithm of x .

math.max (x, ...)

Returns the maximum value among its arguments.

math.min (x, ...)

Returns the minimum value among its arguments.

math.modf (x)

Returns two numbers, the integral part of x and the fractional part of x .

math.pi

The value of pi.

math.pow (x, y)

Returns x^y . (You can also use the expression x^y to compute this value.)

math.rad (x)

Returns the angle x (given in degrees) in radians.

math.random ([m [, n]])

This function is an interface to the simple pseudo-random generator function `rand` provided by ANSI C. (No guarantees can be given for its statistical properties.)

When called without arguments, returns a uniform pseudo-random real number in the range [0,1). When called with an integer number `m`, `math.random` returns a uniform pseudo-random integer in the range [1,m]. When called with two integer numbers `m` and `n`, `math.random` returns a uniform pseudo-random integer in the range [m, n].

math.randomseed (x)

Sets `x` as the "seed" for the pseudo-random generator: equal seeds produce equal sequences of numbers.

math.sin (x)

Returns the sine of `x` (assumed to be in radians).

math.sinh (x)

Returns the hyperbolic sine of `x`.

math.sqrt (x)

Returns the square root of `x`. (You can also use the expression `x^0.5` to compute this value.)

math.tan (x)

Returns the tangent of `x` (assumed to be in radians).

math.tanh (x)

Returns the hyperbolic tangent of `x`.

6.1.28. Table manipulations

This library provides generic functions for table manipulation. It provides all its functions inside the `table` library. Most functions in the `table` library assume that the table represents an array or a list. For these functions, when we talk about the "length" of a table we mean the result of the `length` operator.

table.concat (table [, sep [, i [, j]])

Given an array where all elements are strings or numbers, returns `table[i]..sep..table[i+1] ... sep..table[j]`. The default value for `sep` is the empty string, the default for `i` is 1, and the default for `j` is the length of the table. If `i` is greater than `j`, returns the empty string.

table.insert (table, [pos,] value)

Inserts element `value` at position `pos` in `table`, shifting up other elements to open space, if necessary. The default value for `pos` is `n+1`, where `n` is the length of the table, so that a call `table.insert(t,x)` inserts `x` at the end of table `t`.

table.maxn (table)

Returns the largest positive numerical index of the given table, or zero if the table has no positive numerical indices. (To do its job this function does a linear traversal of the whole table.)

table.remove (table [, pos])

Removes from table the element at position pos, shifting down other elements to close the space, if necessary. Returns the value of the removed element. The default value for pos is n, where n is the length of the table, so that a call *table.remove(t)* removes the last element of table t.

table.sort (table [, comp])

Sorts table elements in a given order, in-place, from *table[1]* to *table[n]*, where n is the length of the table. If comp is given, then it must be a function that receives two table elements, and returns true when the first is less than the second (so that not *comp(a[i+1],a[i])* will be true after the sort). If comp is not given, then the standard Lua operator < is used instead.

The sort algorithm is not stable; that is, elements considered equal by the given order may have their relative positions changed by the sort.

6.1.29. *Operating system facilities*

os.date ([format [, time]])

Returns a string or a table containing date and time, formatted according to the given string format. If the time argument is present, this is the time to be formatted (see the *os.time* function for a description of this value). Otherwise, date formats the current time.

If format starts with '!', then the date is formatted in Coordinated Universal Time. After this optional character, if format is the string *"*t"*, then date returns a table with the following fields: year (four digits), month (1--12), day (1--31), hour (0--23), min (0--59), sec (0--61), wday (weekday, Sunday is 1), yday (day of the year), and isdst (daylight saving flag, a boolean).

If format is not *"*t"*, then date returns the date as a string, formatted according to the same rules as the C function *strftime*.

When called without arguments, date returns a reasonable date and time representation that depends on the host system and on the current locale (that is, *os.date()* is equivalent to *os.date("%c")*).

os.difftime (t2, t1)

Returns the number of seconds from time t1 to time t2. In POSIX, Windows, and some other systems, this value is exactly t2-t1.

os.execute ([command])

This function is equivalent to the C function *system*. It passes command to be executed by an operating system shell. It returns a status code, which is system-dependent. If command is absent, then it returns nonzero if a shell is available and zero otherwise.

os.exit ([code])

Calls the C function *exit*, with an optional code, to terminate the host program. The default value for code is the success code.

os.getenv (varname)

Returns the value of the process environment variable *varname*, or *nil* if the variable is not defined.

os.remove (filename)

Deletes the file or directory with the given name. Directories must be empty to be removed. If this function fails, it returns *nil*, plus a string describing the error.

os.rename (oldname, newname)

Renames file or directory named *oldname* to *newname*. If this function fails, it returns *nil*, plus a string describing the error.

os.time ([table])

Returns the current time when called without arguments, or a time representing the date and time specified by the given table. This table must have fields *year*, *month*, and *day*, and may have fields *hour*, *min*, *sec*, and *isdst* (for a description of these fields, see the *os.date* function). The returned value is a number, whose meaning depends on your system. In POSIX, Windows, and some other systems, this number counts the number of seconds since some given start time (the "epoch"). In other systems, the meaning is not specified, and the number returned by time can be used only as an argument to *date* and *difftime*.

os.tmpname ()

Returns a string with a file name that can be used for a temporary file. The file must be explicitly opened before its use and explicitly removed when no longer needed. On some systems (POSIX), this function also creates a file with that name, to avoid security risks. (Someone else might create the file with wrong permissions in the time between getting the name and creating the file.) You still have to open the file to use it and to remove it (even if you do not use it).

When possible, you may prefer to use *io.tmpfile*, which automatically removes the file when the program ends.

6.1.30. Extended function library

toboolean(value)

Converts the given value to boolean using following rules: *nil*, boolean *false*, 0, empty string, '0' string are treated as *false*, everything else as *true*

string.split(str, sep)

Splits the given string into chunks by the given separator. Returns Lua table.

knxlib.decodeia(indaddressa, indaddressb)

Converts binary-encoded individual address to Lua string. This function accepts either one or two arguments (interpreted as two single bytes).

knxlib.decodega(groupaddressa, groupaddressb)

Converts binary-encoded group address to Lua string. This function accepts either one or two arguments (interpreted as two single bytes).

knxlib.encodega(groupaddress, separate)

Converts Lua string to binary-encoded group address. Returns group address a single Lua number when second argument is *nil* or *false* and two separate bytes otherwise.

ipairs(t)

Returns three values: an iterator function, the table *t*, and 0, so that the construction

```
for i,v in ipairs(t)do body end
```

will iterate over the pairs (1,t[1]), (2,t[2]), ..., up to the first integer key absent from the table.

next (table [, index])

Allows a program to traverse all fields of a table. Its first argument is a table and its second argument is an index in this table. *next* returns the next index of the table and its associated value. When called with *nil* as its second argument, *next* returns an initial index and its associated value. When called with the last index, or with *nil* in an empty table, *next* returns *nil*. If the second argument is absent, then it is interpreted as *nil*. In particular, you can use *next(t)* to check whether a table is empty. The order in which the indices are enumerated is not specified, even for numeric indices. (To traverse a table in numeric order, use a numerical *for* or the *ipairs* function.)The behavior of *next* is undefined if, during the traversal, you assign any value to a non-existent field in the table. You may however modify existing fields. In particular, you may clear existing fields.

pairs(t)

Returns three values: the *next* function, the table *t*, and *nil*, so that the construction

```
for k,v inpairs(t)do body end
```

will iterate over all key–value pairs of table *t*.

tonumber (e [, base])

Tries to convert its argument to a number. If the argument is already a number or a string convertible to a number, then `tonumber` returns this number; otherwise, it returns `nil`.

An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter 'A' (in either upper or lower case) represents 10, 'B' represents 11, and so forth, with 'Z' representing 35. In base 10 (the default), the number can have a decimal part, as well as an optional exponent part. In other bases, only unsigned integers are accepted.

tostring (e)

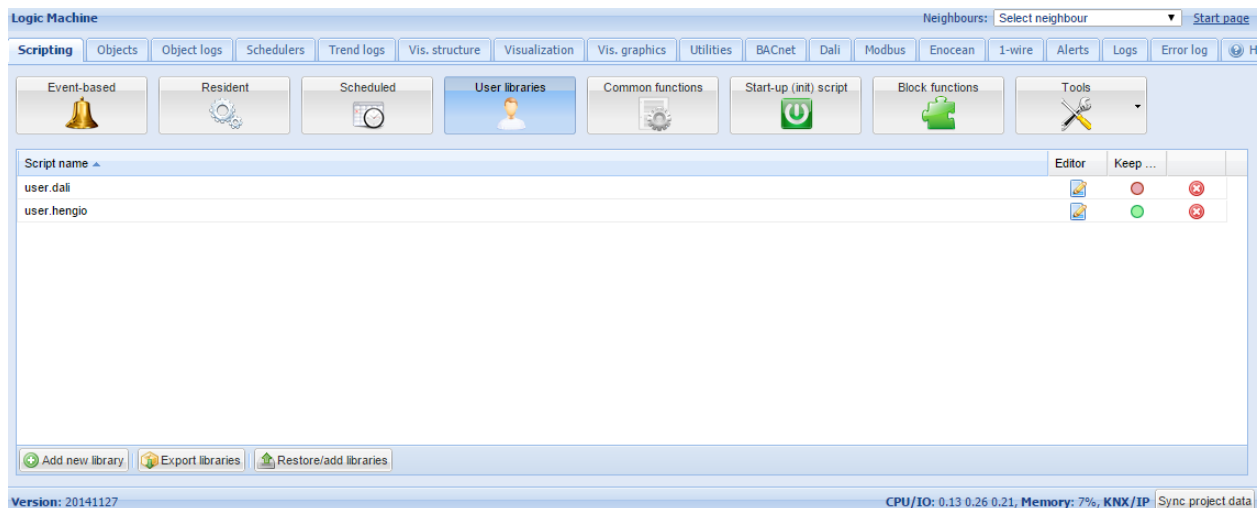
Receives an argument of any type and converts it to a string in a reasonable format. For complete control of how numbers are converted, use `string.format`.

If the meta table of `e` has a `__tostring` field, then `tostring` calls the corresponding value with `e` as argument, and uses the result of the call as its result.

type (v)

Returns the type of its only argument, coded as a string. The possible results of this function are "nil" (a string, not the value `nil`), "number", "string", "boolean", "table", "function", "thread", and "userdata".

6.1.31. User libraries

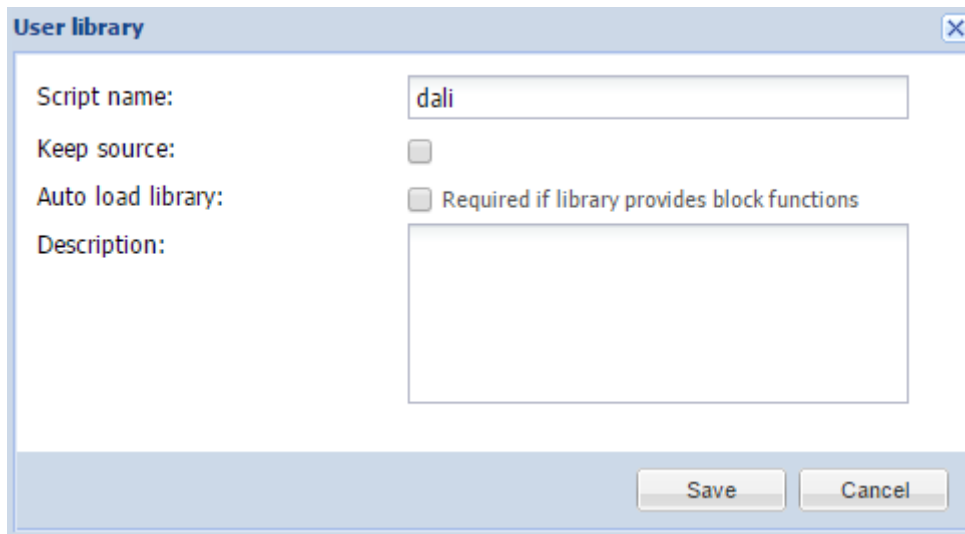


User libraries usually contain user defined functions which are later called from other scripts.

You have to include your library in the script with the following command:
require('user.test') unless you have enabled *Auto load library*.

Secure the code

There is an option *keep source* available for user libraries. Once disabled, the code is compiled in the binary form and can't be seen for further editing. If this option is enabled, the source code is seen in the editor.



Auto load library means that the library will be automatically loaded so you don't have to use **require** when writing scripts. Also this have to be checked if Block programming is used.

6.1.32. Common functions

Common functions contains library of globally used functions. They can be called from any script, any time, without special including like with *user libraries*. Functions like *sunrise/sunset*, *Email* are included by default in *Common functions*.

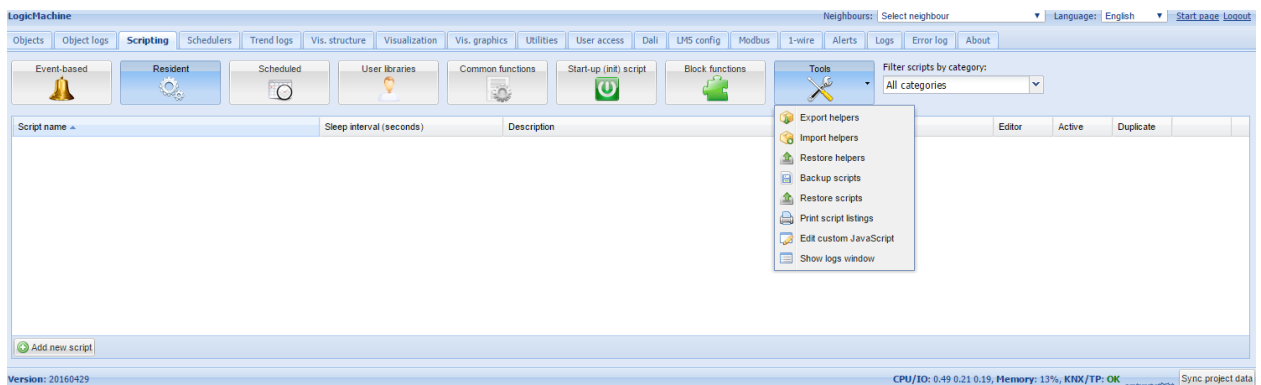


6.1.33. Start-up (init) script

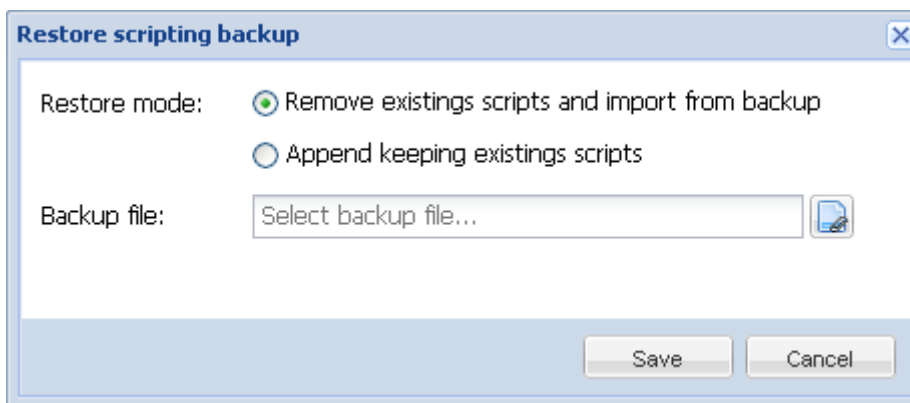
Init script is used for initialization on specific system or bus values on system start. Init script is run each time after system is restarted for some reason.



6.1.34. Tools



- **Export helpers** – export scripting helpers
- **Import helpers** – import scripting helpers
- **Restore helpers** – restore default scripting helpers
- **Backup user scripts** – backup all scripts in *.gz file
- **Restore from archive** – restore script from archive (*.gz) file with two possibilities:
 - Remove existing scripts and import from backup
 - Append keeping existing (s) scripts



- **Print script listings** – shows all scripts with codes in list format sorted by Categories.

Category: Presence

Presence simulator (id: 1)

Type: Resident

Active: Yes

Script sleep interval: 20

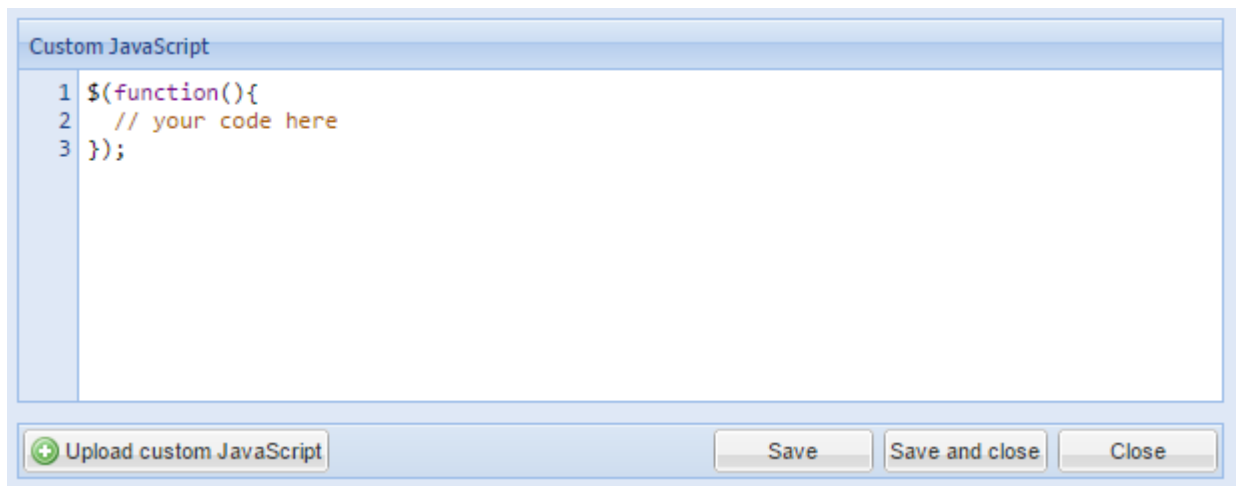
Synchronizes 0/0/2 value with 0/0/1

```
-- if object exists "presence" variable will be a table, nil otherwise
presence = knxobject.get('address', '0/0/1')

-- check that object exists and data has been decoded
if presence and presence.decoded then
  -- result will be either "value = true" or value = "false"
  alert('value = %s', tostring(presence.data))

  -- update 0/0/2 with the same data
  knxobject.write('0/0/2', presence.data, dt.bool)
else
  alert('read error')
end
```

- **Show logs window** – show logs in separate window
- **Edit custom JavaScript**



With custom JavaScripts it is possible to create different dynamic tasks, like detect short/long press from visualization using one icon or we can open specific Floor/Plan when some grp address is triggered - useful e.g. when you want IP Camera page automatically to be opened when Intercom button is pressed:

```
$(function(){
  if (typeof objectStore !== 'undefined') {
    var id = Scada.encodeGroupAddress('1/1/2');

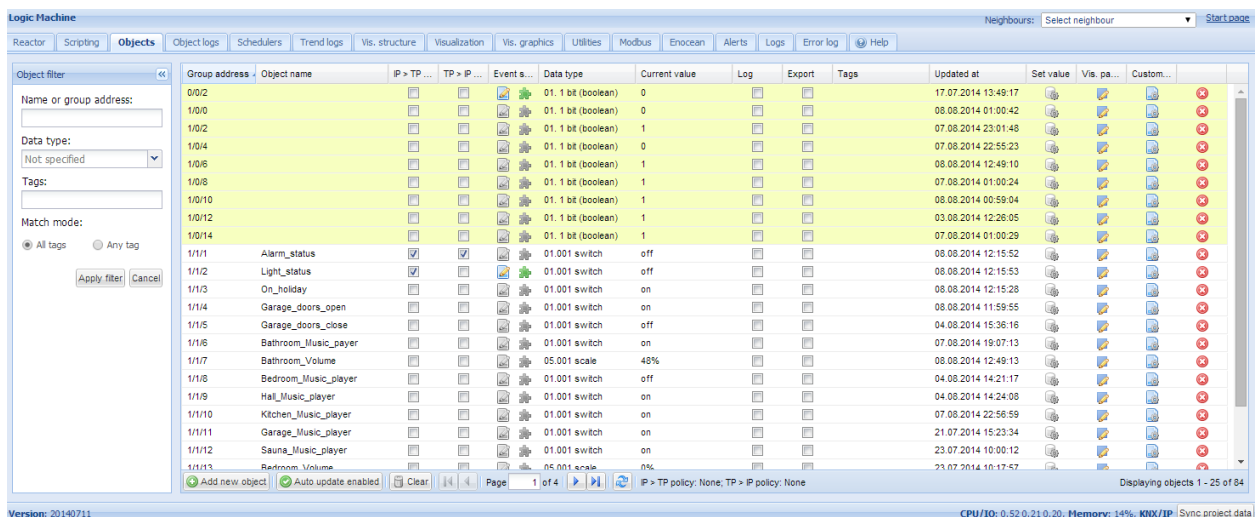
    objectStore.addListener(id, function(object, type) {
      if (type == 'value') {
        showPlan(69);
      }
    });
  }
});
```

See more examples here: <http://forum.logicmachine.net/showthread.php?tid=275>

6.2. Objects

List of KNX network objects appears in *Objects* menu. The object appears in the list by way of:

- sniffing the bus for telegrams from unknown group addresses (if enabled in *Utilities*)
- adding manually
- importing ESF file (in *Utilities*)



6.2.1. Object parameters

To change the settings for existing or new objects, press on the specific list entry.

- **Object name** – Name for the object
- **Group address** – Group address of this object
- **Data type** – KNX data type for the object. This has to be set once the LM sniffs the new object for proper work.
- **Units / suffix** – units for the object which will appear on the visualization along with the value
- **Log** – enable logging for this object. Logs will appear in *Objects logs* menu.
- **High priority log** – mark the object for high priority logging; when the log database is cleared, first standard logs are cleared, only then high priority
- **Export** – Make object visible by remote XML requests and in BACnet network (if KNX – BACnet gateway functionality is used)
- **Poll interval (seconds)** – perform automatic object read after some time interval
- **Tags** – assign this object to some tag which can be later used in writing scripts, for example, *All_lights_first_floor*.
- **Current value** – Current value of the object
- **Object comments** – Comment for the object

There is a possibility to sort the objects by one of the following – Name, Group address, Data type, Current value, Tags, Comments

6.2.2. RGB group object

A special *RGB color* data type is added in *Data type* list.

Edit object

Object name: light1-2

Group address: 1/1/12

Data type: 232.600 RGB color

Current value: 4133887

Tags:

Units / suffix:

Log:


High priority log:

Export:

Poll interval (seconds):

Object comments:

Save Cancel

In *Visualization Parameters*  you can do the following settings for the object:

Object visualization parameters

Object: light1-2 (1/1/12)

Send after each color pick:

Number of presets to show: 3

Preset 1: #00B2

Preset 2: #7AFF

Preset 3: #CC00

Save Cancel

Send after each color pick – specifies either to send the telegram automatically into KNX bus once the color is selected in color picker.


Number of presets to show – count of predefined presets in color picker in Visualization

Preset 1..6 – preset color

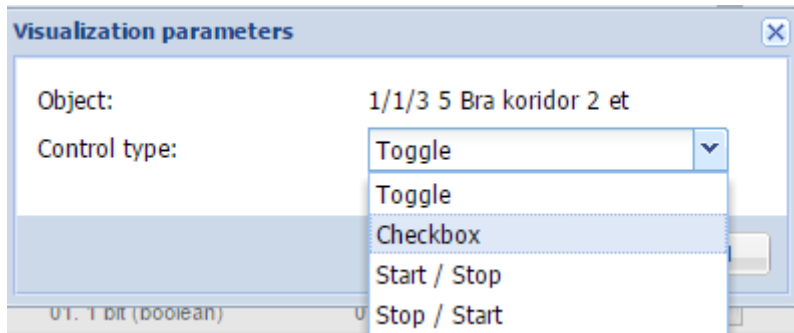
When you add the object with RGB color data type in the Visualization, the color picker with predefined colors appears.



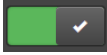

6.2.3. Object visualization parameters

By pressing on the  button of the corresponding object you can set specific visualization parameters for this type of object.

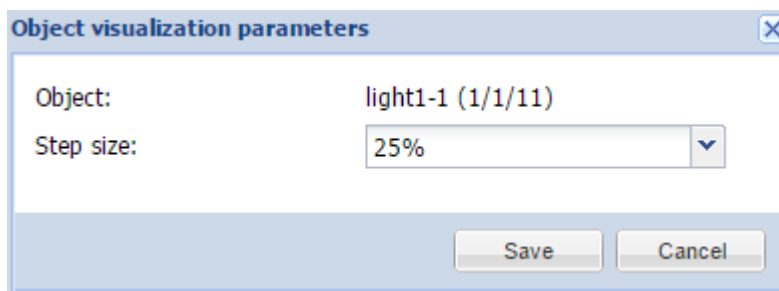
1 bit



- **Control type** – type of the visual control element which will appear in Touch Visualization

- Toggle 
- Checkbox 
- Start/Stop – while icon is pressed it send one value, when it is released, it send opposite value

4 bit (3 bit controlled)

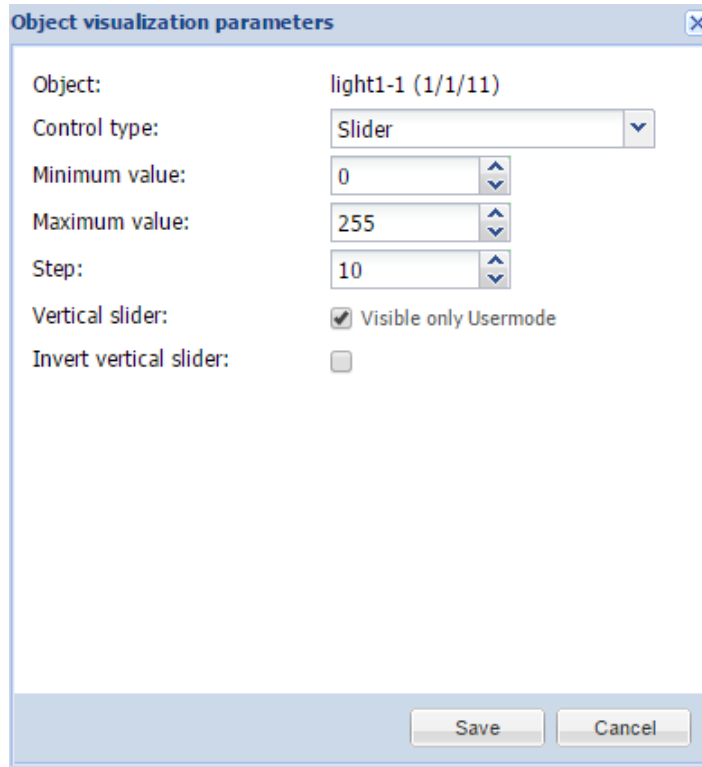


- **Step size** – step size for object change, example for blinds control

1byte and 4byte float

- Control type – type of the visual control element

○ Slider

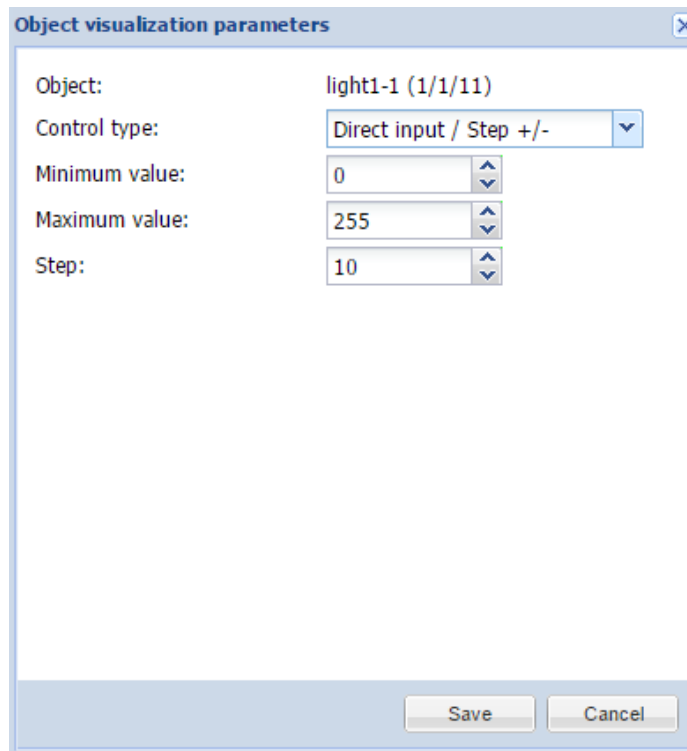


inimum value – minimum value on the slider

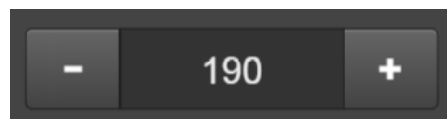
aximum value – maximum value on the slider

- *Step* – step for one slider movement
- *Vertical slider* – special option for Usermode visualization
- *Invert vertical slider* – invert vertical slider so the maximum is on top

○ Direct input / Step +/-

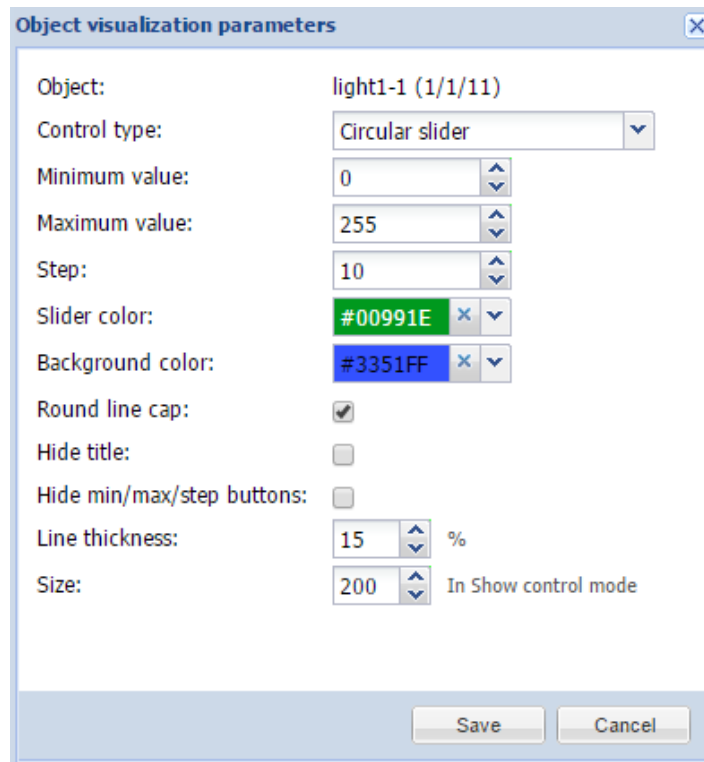



- *Minimum value* – minimum value on the control bar
- *Maximum value* – maximum value on the control bar
- *Step* – step for one position change

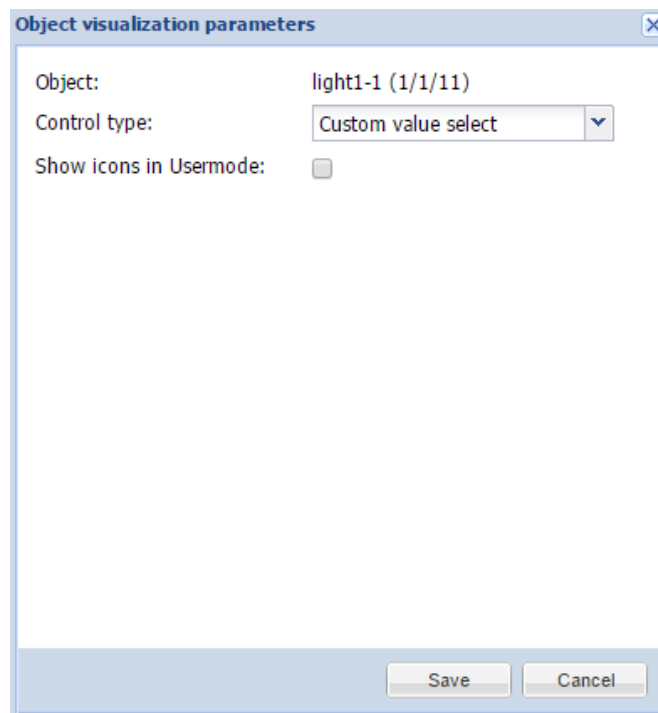


○ Circular slider

- *Minimum value* – minimum value on the control bar
- *Maximum value* – maximum value on the control bar
- *Step* – step for one position change
- *Slider color* – color of slider
- *Background color* – background color of the slider
- *Round line cap* – make round ends of slider
- *Hide title* – hide title
- *Hide min/max/step buttons* – hide min, max and step buttons
- *Line thickness* – specify the thickness of slider line
- *Size* – Size in px of the control




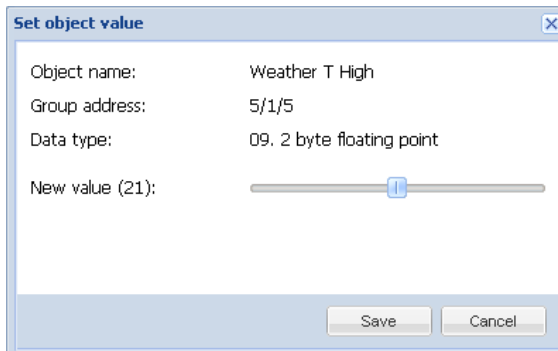
- Custom value select – select from list of custom values. Custom values should be defined  in
 - *Show icons in Usermode* – show icons instead of values for the object in visualization. You will be able to choose from defined icons/custom values. Icons should be defined in visualization constructor as *Additional*



Icons

6.2.4. Change the object state

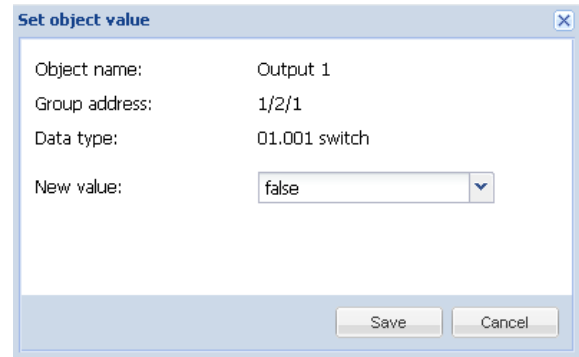
In the object list, by pressing on the  button, you can change the state of the object. The appearance of the *New value* depends on what visualization parameters are set for specific object.



The dialog box titled "Set object value" shows the following information:

- Object name: Weather T High
- Group address: 5/1/5
- Data type: 09. 2 byte floating point
- New value (21): A slider control with a blue knob.

Buttons: Save, Cancel




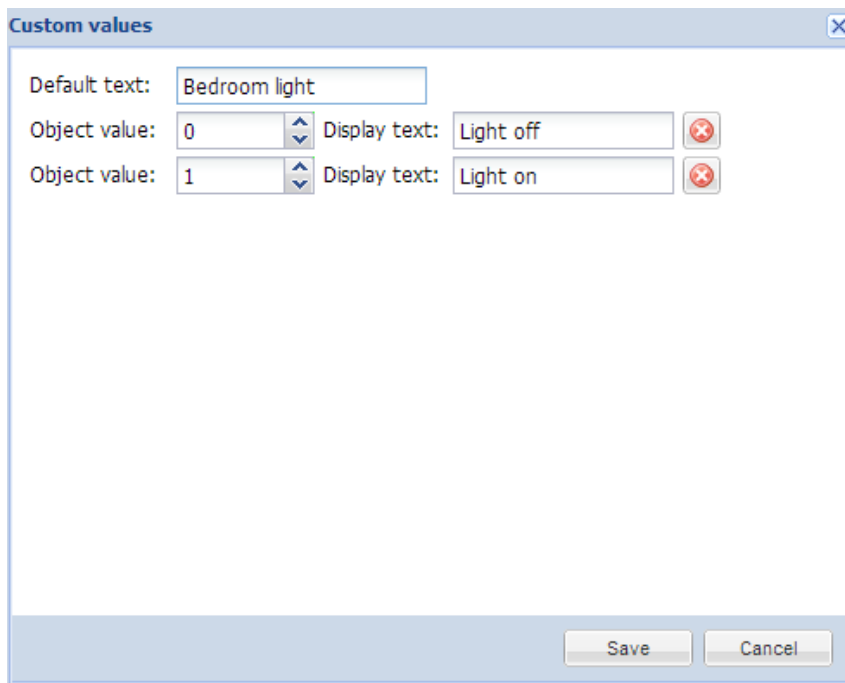
The dialog box titled "Set object value" shows the following information:

- Object name: Output 1
- Group address: 1/2/1
- Data type: 01.001 switch
- New value: A dropdown menu showing "false".

Buttons: Save, Cancel

6.2.5. Custom values

If special value naming is necessary, use this icon  to set it up (only for Boolean and Integer data types)



The dialog box titled "Custom values" shows the following information:

- Default text: Bedroom light
- Object value: 0, Display text: Light off
- Object value: 1, Display text: Light on

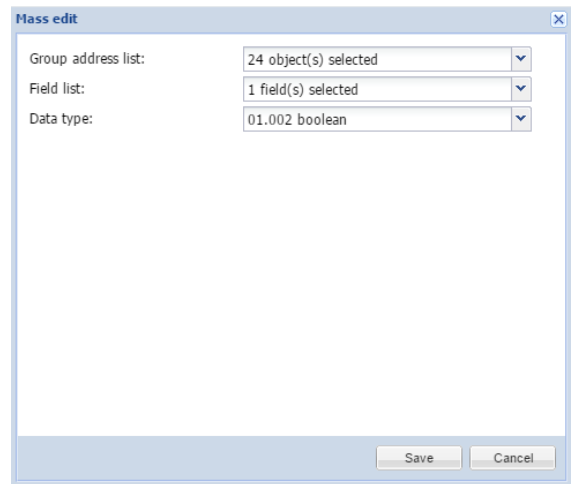
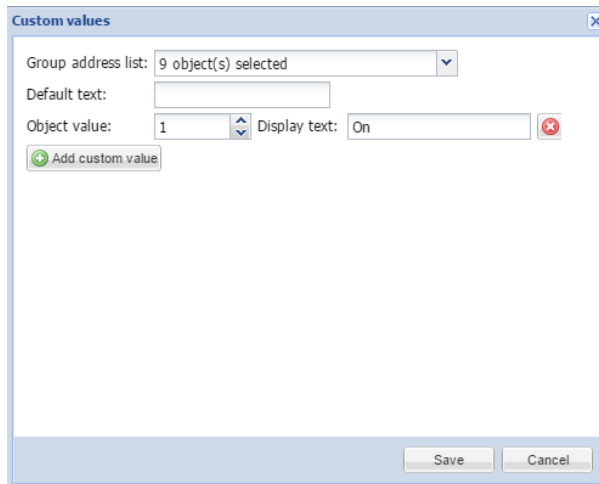
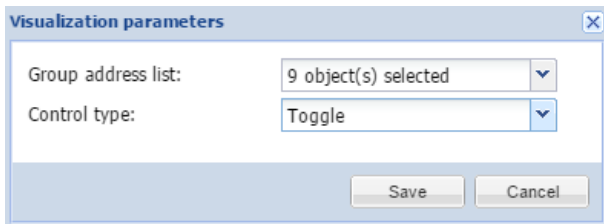
Buttons: Save, Cancel

6.2.6. Object control bar



- **Add new object** – Manually add new object to the list
- **Auto update enabled** – Specifies either the object list is updated automatically or not

- **Clear** – Clear the list of group addresses
- **Next/Previous page** – move to next or previous page
- **Refresh** – refresh the object list
- **Mass edit** – mass edit objects by a specific criteria – object properties, visualization parameters or custom values.



- **Mass delete** – delete mass object either by current Mass Edit filters or all unnamed objects

There is also the following bar on the bottom of the configuration screen:



- **CPU/IO** –Load average. The load average represents the average system load over a period of time. It conventionally appears in the form of three numbers which represent the system load during the last one-, five-, and fifteen-minute periods. The lower number the better.

Note! Inspect your running tasks if the load exceeds the level 0.70!

More on UNIX style load calculation can be found here:

[http://en.wikipedia.org/wiki/Load_\(computing\)#Unix-style_load_calculation](http://en.wikipedia.org/wiki/Load_(computing)#Unix-style_load_calculation)

- **Memory** – memory usage in %
- **KNX/IP / KNX/TP** – type of connection to KNX bus. If KNX/TP is set and it is not available, there will be error notification
- **Sync project data** – save all project data to internal flash by pressing this button. Otherwise the data is saved once in 30 minutes from RAM to Flash, or when Reboot or Shutdown commands are sent
- **KNX statistics graphs** – shows average KNX bus load

6.2.7. Filter objects

On the left side of the object list there is filtering possible. To perform the filtering type the name, group address, tag or specify the data type of the object and press on *Filter* button.

The screenshot shows the Logic Machine software interface. On the left, the 'Object filter' panel is active, showing a search for 'bedroom' in the 'Name or group address' field. The 'Data type' is set to 'Not specified'. The 'Match mode' is set to 'All tags'. The main area displays a table of objects with columns for Group, Object name, IP, TP, Data type, Current value, Log, Export, Tags, and Updated at. The table lists 13 objects related to a bedroom, such as 'Bedroom_Music...', 'Bedroom_Volume', 'Bedroom_Tem...', 'Bedroom_Humidity', and 'Bedroom_FL_light'. The status bar at the bottom indicates 'Version: 20140711' and 'CPU/IO: 0.39 0.54 0.36, Memory: 13%, KNX/IP: Sync project data'.

Group ...	Object name	IP >...	TP ...	Eve...	Data type	Current value	Log	Export	Tags	Updated at	Set ...	Vis...	Cus...
1/1/8	Bedroom_Music...				01.001 swi...	off				04.08.2014...			
1/1/13	Bedroom_Volume				05.001 scale	0%				23.07.2014...			
1/1/21	Bedroom_Tem...				09.001 Te...	30 °C				10.07.2014...			
1/1/22	Bedroom_Temp...				09.001 Te...	35 °C				25.07.2014...			
1/1/23	Bedroom_Humidity				05.001 scale	48 %				10.07.2014...			
1/1/38	Bedroom_W_rig...				01.001 swi...	on				22.07.2014...			
1/1/39	Bedroom_W_rig...				01.001 swi...	on				23.07.2014...			
1/1/40	Bedroom_W_left...				01.001 swi...	off				22.07.2014...			
1/1/41	Bedroom_W_left...				01.001 swi...	on				23.07.2014...			
1/1/50	Bedroom_C_Light				01.001 swi...	on				23.07.2014...			
1/1/51	Bedroom_C_Lig...				05.001 scale	60%				24.07.2014...			
1/1/52	Bedroom_R_light				01.001 swi...	on				23.07.2014...			
1/1/53	Bedroom_FL_light				01.001 swi...	on				23.07.2014...			

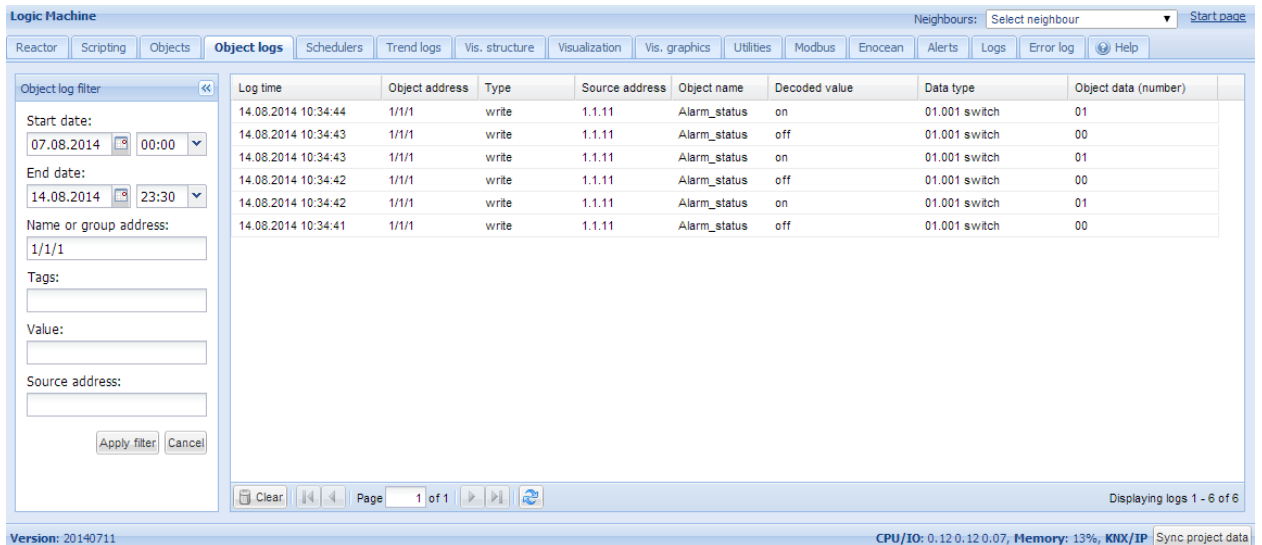
Match mode:

All tags – represents AND function when all tags should match

Any tag – represents OR function when any one of listed should match

6.3. Object logs

Object historical telegrams are available in *Object logs*. Once logging is enabled for object, all it's further history will be logged.

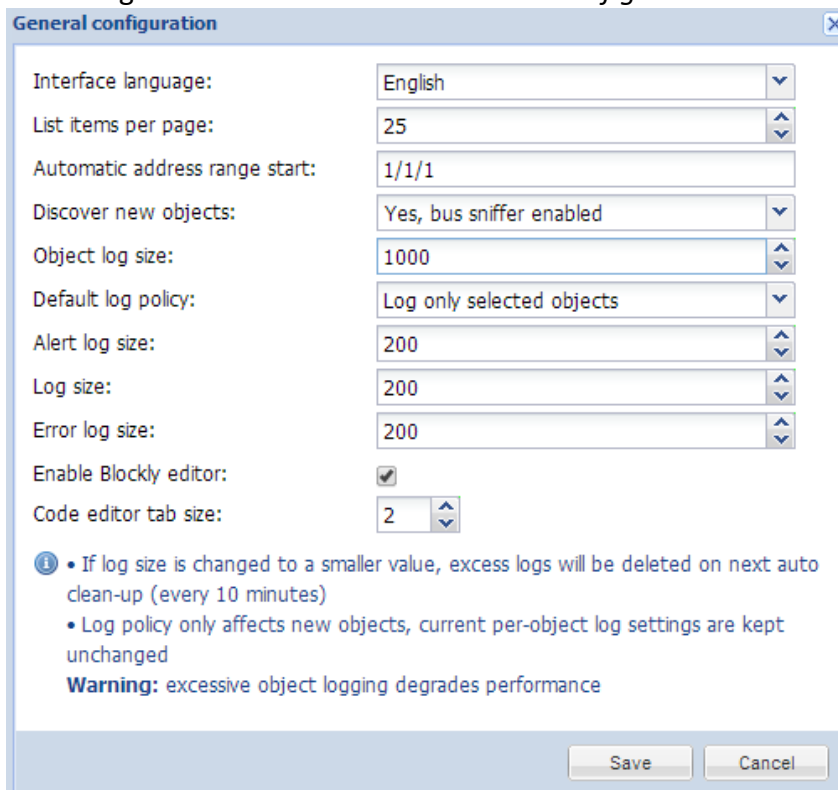


Filtering is available when there is a need to find specific period information

- **Start date** – start date and time for log filtering
- **End date** – start date and time for log filtering
- **Name or group address** – specific name or group address of object
- **Tags** – tag names
- **Value** – specific object value
- **Source address** – specific source address

You can clear all logs by pressing on *Clear* button.

Size of log is defined in *Utilities* → *General Configuration*

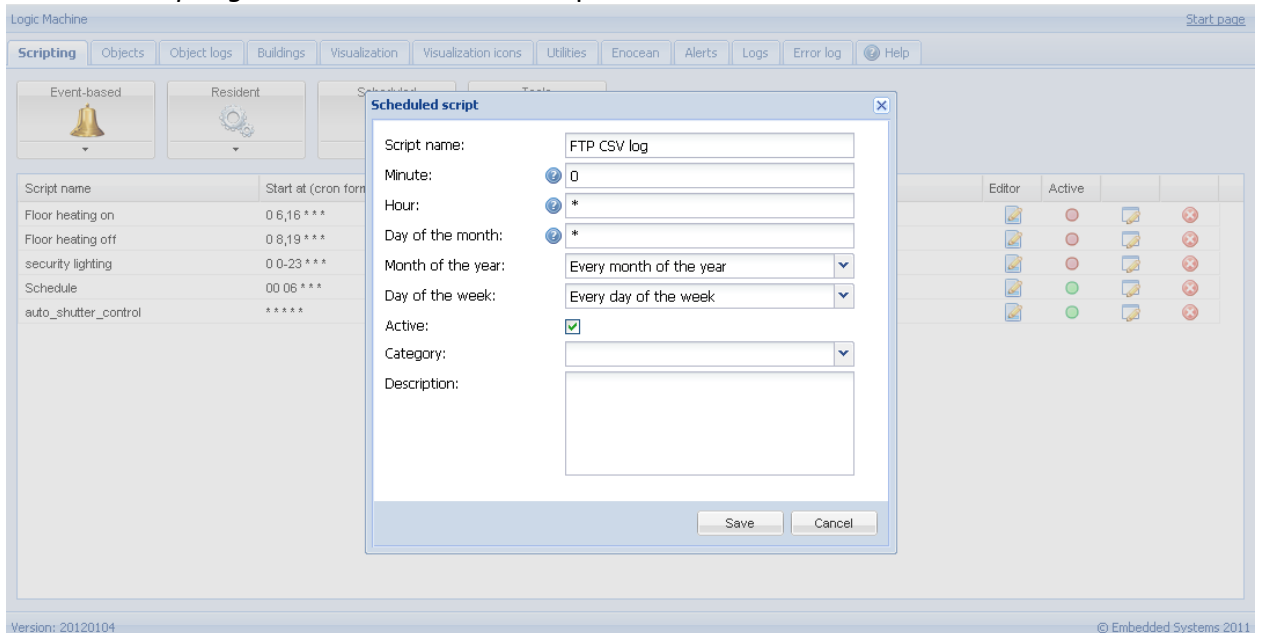


6.3.1. Export logs

Example

Once an hour, make CSV file with all objects logs and send to external FTP server with IP 192.168.1.11, login 'ftplgin', password 'ftppassword'.

- In *Scripting* -> *Scheduled* add the script which will run once an hour



- Add the following code in Script editor for this particular script.

```
1. require('socket.ftp')
2.
3. -- ftp file
4. ftpfile=string.format('ftp://ftplgin:ftppassword@192.168.1.11/%s.csv', os.date('%Y-%m-%d_%H-%M'))
5. -- get past hour data (3600 seconds)
6. logtime=os.time() - 60*60
7.
8. -- list of objects by id
9. objects ={}
10.
11. -- objects with logging enabled
12. query = 'SELECT address, datatype, name FROM objects WHERE disablelog=0'
13. for _, object in ipairs(db:getall(query)) do
14.   objects[tonumber(object.address)]={}
15.   datatype=tonumber(object.datatype),
16.   name =tostring(object.name or ''),
17. }
18. end
```

```

19.
20. -- csv buffer
21. buffer ={'date',"address","name","value"}
22.
23. -- get object logs
24. query='SELECT src, address, datahex, logtime, eventtype FROM objectlog WHERE logtime>=
? ORDER BY id DESC'
25. for _, row in ipairs(db:getall(query, logtime))do
26.   object = objects[tonumber(row.address)]
27.
28. -- found matching object and event type is group write
29. if object and row.eventtype=='write' then
30.   datatype=object.datatype
31.
32. -- check that object datatype is set
33. if datatype then
34. -- decode data
35.   data =knxdatatype.decode(row.datahex, datatype)
36.
37. -- remove null chars from char/string datatype
38. if datatype==dt.char or datatype==dt.string then
39.   data =data:gsub('%z+', '')
40. -- date to DD.MM.YYYY
41. elseifdatatype==dt.date then
42.   data =string.format('%02d.%02d.%02d', data.day, data.month, data.year)
43. -- time to HH:MM:SS
44. elseif datatype==dt.time then
45.   data =string.format('%02d:%02d:%02d', data.hour, data.minute, data.second)
46. end
47. else
48.   data =''
49. end
50.
51. -- format csv row
52. logdate=os.date('%Y.%m.%d %H:%M:%S', row.logtime)
53. csv=string.format('%q,%q,%q,%q', logdate, knxlib.decodega(row.address), object.name,
tostring(data))
54.
55. -- add to buffer
56. table.insert(buffer, csv)
57. end
58. end
59.
60. -- upload to ftp only when there's data in buffer
61. if #buffer > 1 then
62.   result, err =socket.ftp.put(ftpfile, table.concat(buffer, '\r\n'))
63. end

```

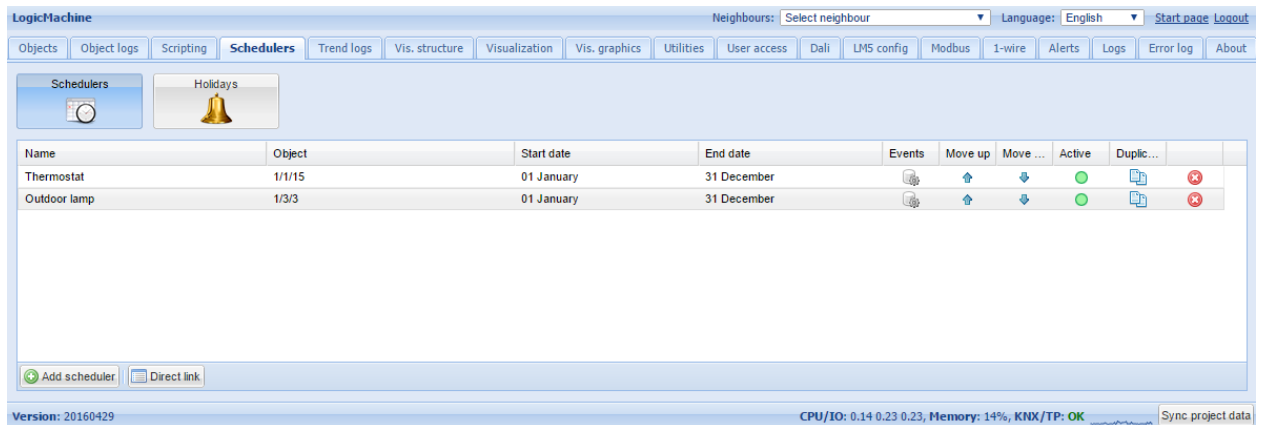
```

64.
65. -- error while uploading
66. if err then
67.   alert('FTP upload failed: %s', err)
68. end

```

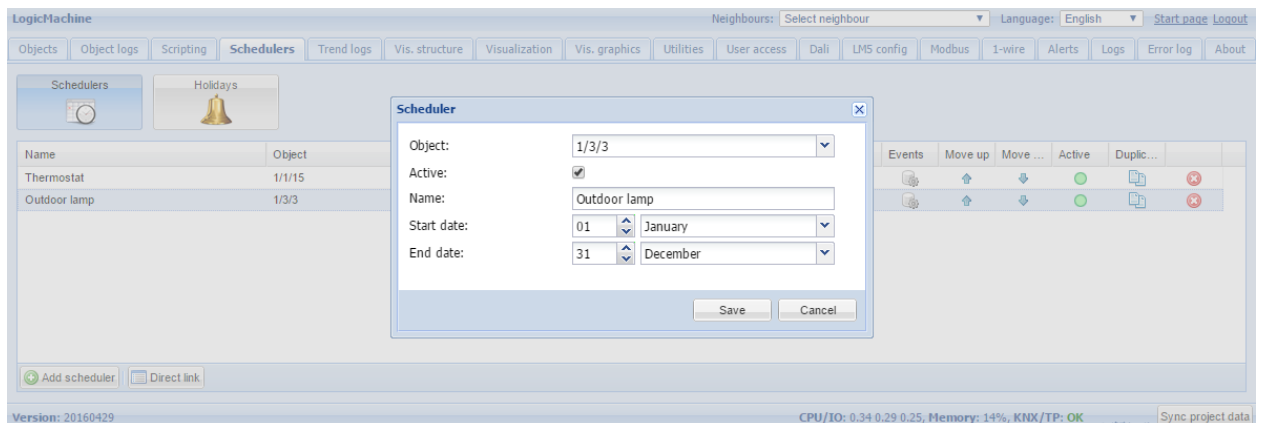
6.4. Schedulers

Schedulers contain administration of user mode schedulers. Schedulers allow for end user to control KNX group address values based on the date or day of the week.



6.4.1. Add new scheduler

By clicking on the *Schedulers* → *Add new scheduler* you will see such parameter window:



- **Object** – the object group address which will be controlled by scheduler
- **Active** – define this scheduler as active or not
- **Name** – name of the scheduler
- **Start date** – start date of the scheduler
- **End date** – end date of the scheduler

6.4.2. Scheduler events

Event can be added both in administrator interface as well as by end user in the special *User mode schedulers* interface.

Start time	Days of the week	Value	Active	
12:00	We, Th	24	<input checked="" type="checkbox"/>	<input type="checkbox"/>
22:00	Mo, Tu, We, Th	20	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Active:

Value:

Start time:

Days of the week: Mo Tu We Th Fr
 Sa Su Hol

- **Active** – define the event active or not
- **Value** – value to send to the group address when the event will be triggered
- **Start time** – start time for the event
- **Days of the week** – days of the week when the event will be triggered.
Hol – holidays which are defined in *Holidays* tab

6.4.3. Scheduler holidays

Once the event will be marked to run in *Hol*, Holiday entries will be activated.

Name:

Date:

Leave year blank for recurring holidays

- **Name** – the name of the holiday entry
- **Date** – date of the holiday

6.4.4. Direct link

To get direct link to a specific scheduler click on *Direct link* button on bottom left part.

Direct link

Scheduler: Outdoor lamp

Link: /scada-vis/schedulers?id=2

Include IP / host:

Show holidays:

6.5. Trend logs

Trends logs are administration of user mode trends, used to see historical object graphical values, compare with other period values.

Name	Object	Log type	Decimal places	Trend resolution	Resolution data	Daily data	Log size	Created	Move up	Move d...	
Thermostat bedroom	1/1/15 (Thermostat)	Absolute value	2	1 hour	180 days	2 years	40 KB	2016.06.28 13...	⬆	⬇	⊗
Humidity	1/1/22 (Humidity sensor)	Absolute value	2	5 minutes	30 days	2 years	74 KB	2016.06.28 13...	⬆	⬇	⊗

LogicMachine Neighbours: Select neighbour Language: English Start page Logout

Objects Object logs Scripting Schedulers **Trend logs** Vis. structure Visualization Vis. graphics Utilities User access Dali LMS config Modbus 1-wire Alerts Logs Error log About

Version: 20160429 CPU/IO: 0.18 0.26 0.24, Memory: 15%, KNX/TP: OK Sync project data

6.5.1. Add new trend log

Trend log

Object: 1/1/15 Thermostat

Name: Thermostat

Log type: Counter

Trend resolution: 1 hour

Decimal places: 2

Resolution data: 180 days

Daily data: 2 years

Always show zero: On graph Y axis

Save Cancel

- **Object** – choose from list of object the one to make trends for
- **Name** – name of the trend
- **Log type** [**Counter, Counter with negative delta, Absolute value**] – type of the log. *Counter* type is used to count the date, *Absolute value* – saves the actual readings
- **Trend resolutions** [**5 min .. 1 hour**] – average value of 1 minute for specific time interval data will be shown on the trend. E.g. if 1 hour – trend step will be 1 hour with average 60 readings data
- **Decimal places** – decimal places for the presentation
- **Daily data** – average value of daily data for specific time interval

Note! One trend data point reading takes 8bytes of flash memory. E.g. reading some value once in every 10 minutes, will consume ~0.4MB of flash each year.

6.5.2. Direct link

To get direct link to a specific trend log click on *Direct link* button on bottom left part.

Direct link

Trend log: Thermostat bedroom

Link: <http://192.168.1.72/scada-vis/trends?id=1>

Include IP / host:

6.5.3. Trend logs functions

To process logged information in trends, you can use built in trend log functions from scripting.

Include library before calling trend log functions:

```
require('trends')
```

Include library before calling trend log functions

```
trends.fetch(name, dates, resolution)  
trends.fetchone(name, dates, resolution)
```

Fetch one or many values for the given period

Parameters:

- **name** trend log name, required
- **dates** Lua table with two items - 'start' and 'end', each item must contain 'year', 'month', 'day' keys, required
- **resolution** optional, will use trend resolution if not specified, set to 86400 for retrieve daily data

Return values:

- **fetch** returns Lua table with values for the given period or nil on error. Number of values depends on period, resolution and data retention settings
- **fetchone** returns single value for the given period or nil on error

Example:

```
require('trends')  
  
-- will fetch data between 2016.04.15 00:00 and 2016.04.16 00:00  
dates = {  
  ['start'] = { year = 2016, month = 4, day = 15 },  
  ['end'] = { year = 2016, month = 4, day = 16 },  
}  
  
-- fetch current value  
day = trends.fetchone('Gas', dates)  
  
-- get data for the past year  
dates = {}  
dates['start'] = os.date('*t')  
dates['start'].year = dates['start'].year - 1  
dates['end'] = os.date('*t')  
  
-- fetch previous value  
yearly = trends.fetch('Gas', dates, 86400)
```

➤ **trends.NaN** value is used for points which contain invalid values or cannot be found. The default value is 0, but it can also be set to 0 / 0 (NaN - not a number).

Example:

```
require('trends')

-- use "not a number" for invalid values
trends.NaN = 0 / 0

-- get data for the past year
dates = {}
dates['start'] = os.date('*t')
dates['start'].year = dates['start'].year - 1
dates['end'] = os.date('*t')

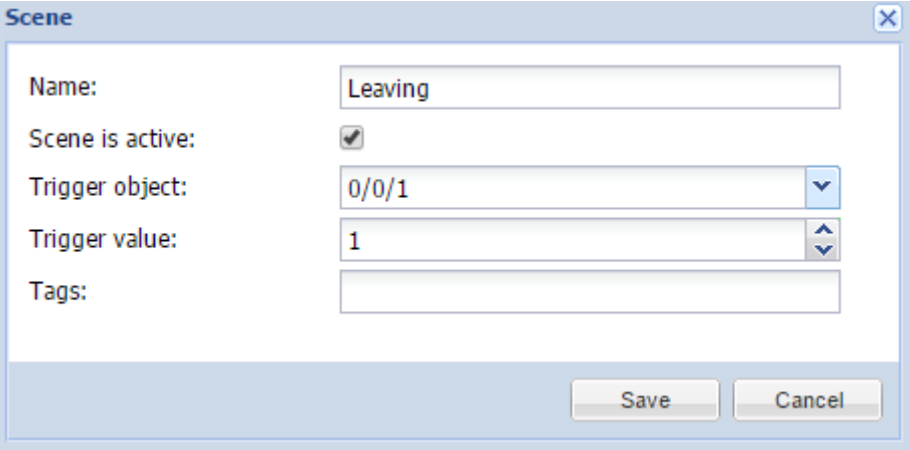
value = trends.fetchone('Hot Water', dates)

-- NaN ~= NaN, means value was not found
if value ~= value then
    return
end
```

6.6. Scenes

Scenes section allows creating scenes visually (without using event-based scripting).

Add scene



The screenshot shows a dialog box titled "Scene" with a close button in the top right corner. The dialog contains the following fields and controls:

- Name:** A text input field containing the text "Leaving".
- Scene is active:** A checkbox that is checked.
- Trigger object:** A dropdown menu showing the value "0/0/1".
- Trigger value:** A spin box showing the value "1".
- Tags:** An empty text input field.

At the bottom right of the dialog, there are two buttons: "Save" and "Cancel".

- **Name** – scene name
- **Scene is active** – define either this scene is active or not
- **Trigger object** – group address for scene trigger object
- **Trigger value** – scene object trigger value

Scene sequence

List of object and the sequence is defined here.

Object	Value	Move up	Move down	Set value	
0/0/2	0	↑	↓		
0/2/4	0	↑	↓		
1/1/4 1 bra otdelno koridor 2 et	0	↑	↓		
1/5/15	0	↑	↓		
2/2/1	0	↑	↓		

Add object

Object:

Write to bus:

Save Cancel

Buttons: Add object, Run scene, Save live values

- **Object** – group address
- **Write to bus** – define either to send KNX bus telegram

6.7. Visualization structure

In *Vis.structure* menu the structure of the visualization is defined and visualization backgrounds are uploaded.

Logic Machine

Neighbours: Select neighbour Start page

Scripting Objects Object logs Schedulers Trend logs **Vis. structure** Visualization Vis. graphics Utilities BACnet Dali Modbus Enocan 1-wire Alerts Logs Error log

Levels / Plans Layouts / Widgets

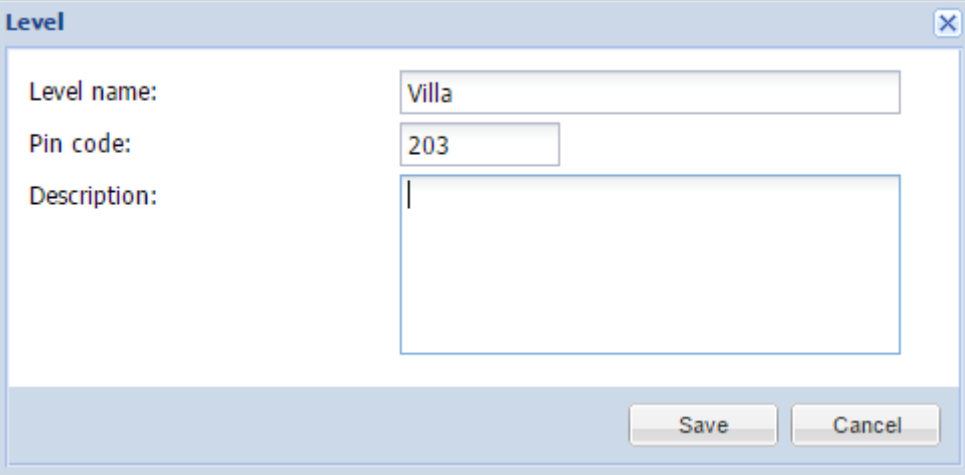
Name	Visible	Description	Duplicate	Move up	Move d...	Add / L...	Export	
House				↑	↓			
Vertical				↑	↓			
1_page_2	Usermode, Touch			↑	↓			
Alternative_1_page_2	Usermode, Touch			↑	↓			
Alternative4_zone_1_page_2	Usermode, Touch			↑	↓			
Romantic_zone_1_page_2	Usermode, Touch			↑	↓			
Alternative3_zone_1_page_2	Usermode, Touch			↑	↓			
Alternative4_zone_1_page_2	Usermode, Touch			↑	↓			
Favorites	Usermode, Touch			↑	↓			
Cameras	Usermode, Touch			↑	↓			
Garage doors	Usermode, Touch			↑	↓			
Audio_Video	Usermode, Touch			↑	↓			
Climate control	Usermode, Touch			↑	↓			

Buttons: Add new level, Import

Version: 20141127 CPU/IO: 0.28 0.15 0.14, Memory: 8%, KNX/IP Sync project data

6.7.1. Levels / Plans

By default there is *Main* level added. To add a new level/building, press “*Add new level*” button. Please note that you can limit access to this specific level by adding PIN code.

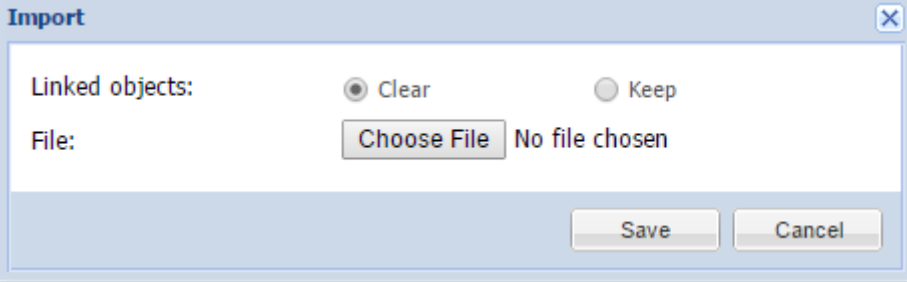


The 'Level' dialog box is shown with the following fields and values:

- Level name: Villa
- Pin code: 203
- Description: (empty)

Buttons: Save, Cancel



You can also add a new level by importing it from the file (which is exported on other LM for example). Press *Import* button for this purpose. Object linkage can be either cleared or imported as-is.

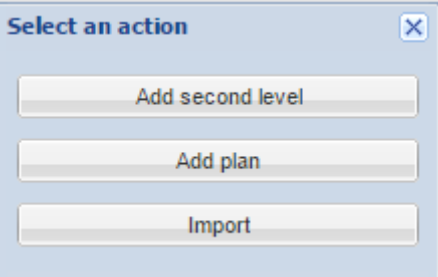


The 'Import' dialog box is shown with the following options and fields:

- Linked objects: Clear Keep
- File: Choose File No file chosen

Buttons: Save, Cancel

Once a new level is added, you can add second level or upload floor pictures related to this particular building. To add a new entry, click on the green icon , to delete a specific entry press on the red icon .



The 'Select an action' dialog box is shown with the following buttons:

- Add second level
- Add plan
- Import

When adding new plan, the following parameters should be defined:

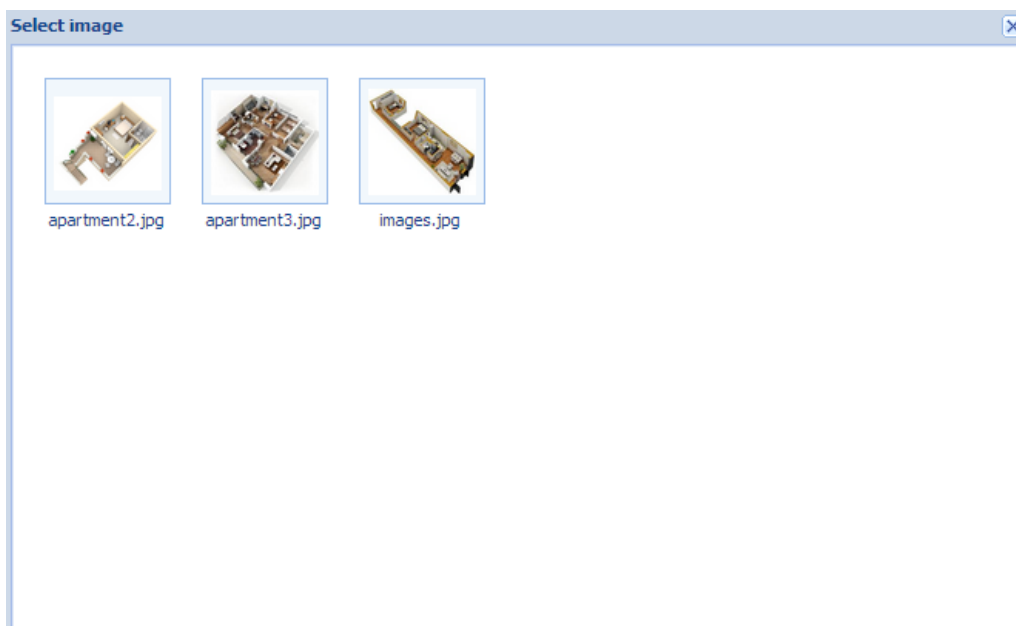
- **Parent** – name of parent level
- **Name** – name for the plan
- **Plan size** – plan size in pixels. There are predefined resolutions available when clicking on the icon on the right side of this parameter:







- **Layout** – layout for this specific plan. All object from Layout will be duplicated on this particular plan including background color and plan image if they are not defined separately for this specific plan
- **Usermode visualization [Show, Show and make default, Hide]** – visibility for this particular plan in Usermode visualization
- **Touch visualization [Show, Show and make default, Hide]** – visibility for this particular plan in Touch visualization

- **PIN code** – specify PIN code to access the plan
- **Primary background image** – choose primary background image from the list added in *Vis.graphics* → *Images/Backgrounds*
- **Secondary background image** – choose secondary background image from the list added in *Vis.graphics* → *Images/Backgrounds*
- **Background color** – choose background color of the plan
- **Touch background color** – define a color for touch visualization
- **Repeat background image** – either to show the image once or repeat it and fill the whole plan
- **Fixed primary background** – specify if first background image should be fixed. By enabling this, you can enable Parallax effect for your visualization
- **Admin only access** – enable admin only access for this floor

When clicking on Background image, the following window appears with background images which has to be added in *Vis.graphics* → *Images/Backgrounds* in advance:

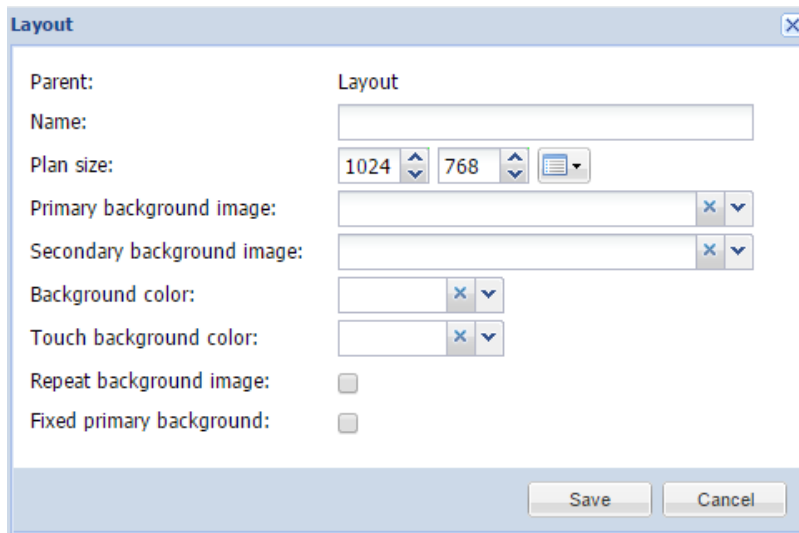


You can duplicate the plan with all its objects and settings by pressing on  icon. Levels can be sorted by pressing  and  icons. You can export the plan structure by clicking in this icon .

6.7.2. Layouts / Widgets

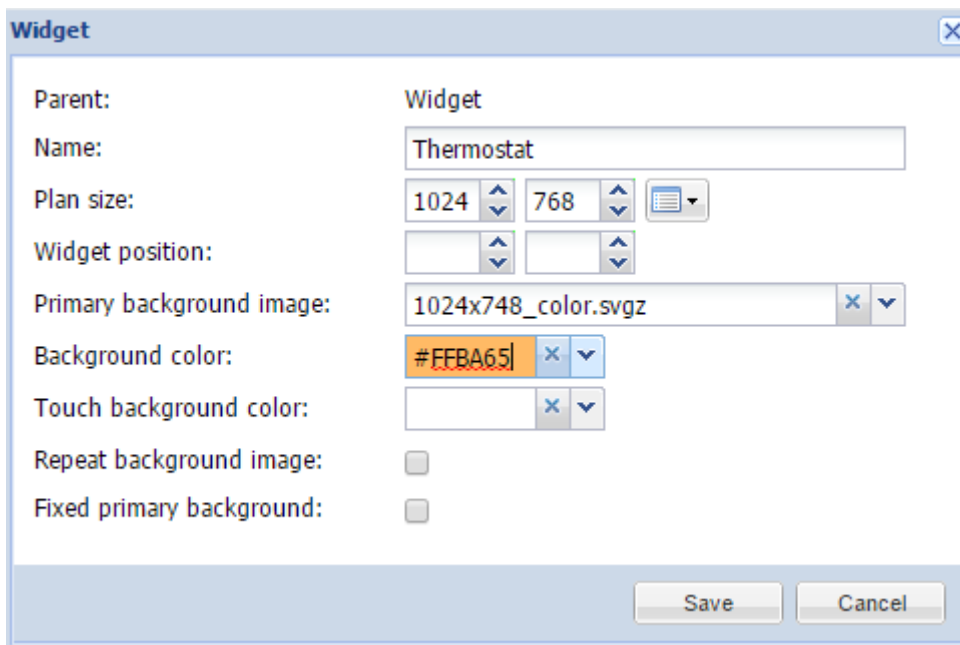
Layouts are used as templates for further use when adding *Levels* in *Levels/Plans* tab.

Layouts will not be visible from the Usermode/Touch visualizations. When you add any background, objects to layouts level in *Visualization*, they will automatically appear on all linked Levels.



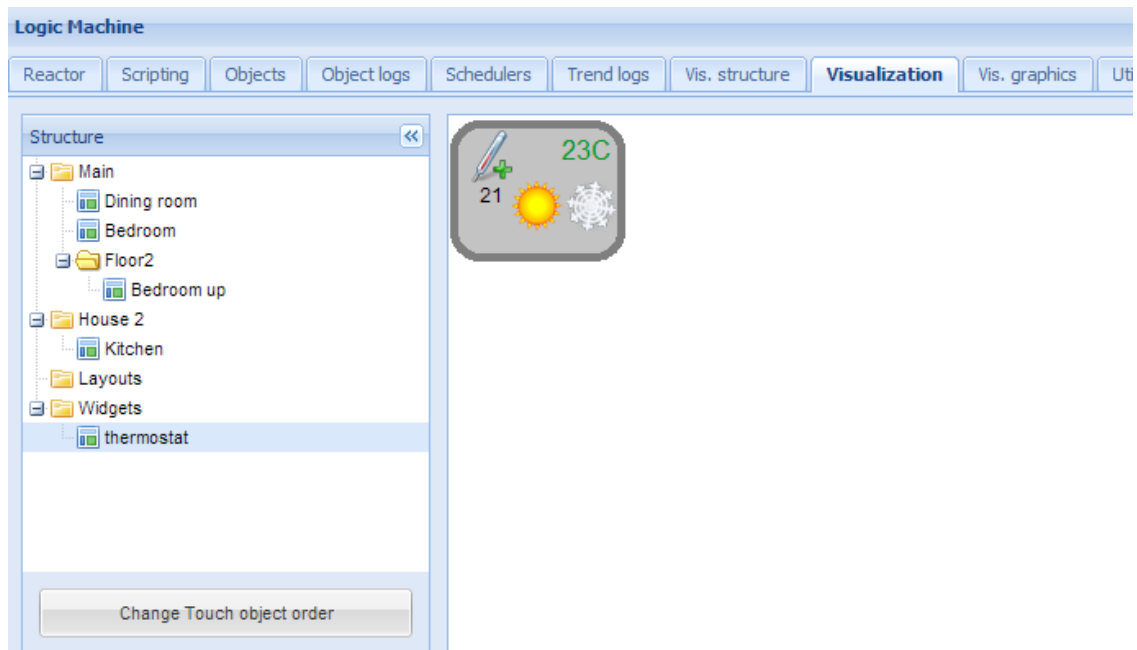
- **Parent** – name of parent layout
- **Name** – name for the layout
- **Plan size** – plan size in pixels. There are predefined resolutions available when clicking on the icon on the right side of this parameter
- **Primary background image** – choose primary background image from the list added in *Vis.graphics* → *Images/Backgrounds*
- **Secondary background image** – choose secondary background image from the list added in *Vis.graphics* → *Images/Backgrounds*
- **Background color** – choose background color of the plan
- **Touch background color** – define a color for touch visualization
- **Repeat background image** – either to show the image once or repeat it and fill the whole plan
- **Fixed primary background** – specify if first background image should be fixed. By enabling this, you can enable Parallax effect for your visualization

Widgets are used to combine several objects under one object in visualization. Background image for the widget should be added in *Vis.graphics* → *Images/Backgrounds* in advance.

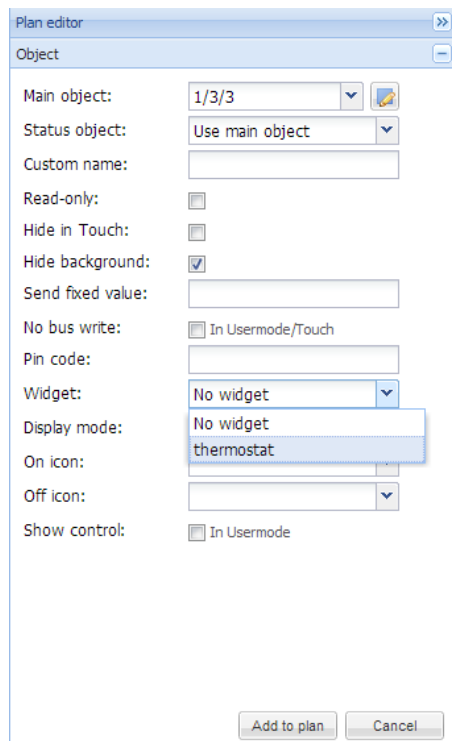


- **Parent** – name of parent widget
- **Name** – name for the widget
- **Plan size** – plan size in pixels. There are predefined resolutions available when clicking on the icon on the right side of this parameter
- **Widget position** – default position of the widget on the screen
- **Primary background image** – choose primary background image from the list added in *Vis.graphics* → *Images/Backgrounds*
- **Background color** – choose background color of the widget
- **Touch background color** – define a color for touch visualization
- **Repeat background image** – either to show the image once or repeat it and fill the whole plan
- **Fixed primary background** – specify if first background image should be fixed. By enabling this, you can enable Parallax effect for your visualization

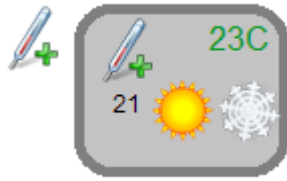
When you have defined the widget in *Layouts/Widgets* tab, you can add objects to it in *Visualization* tab.



When you have added necessary objects to the widget, you can choose it when adding objects for main Levels e.g. Bedroom in Main level.




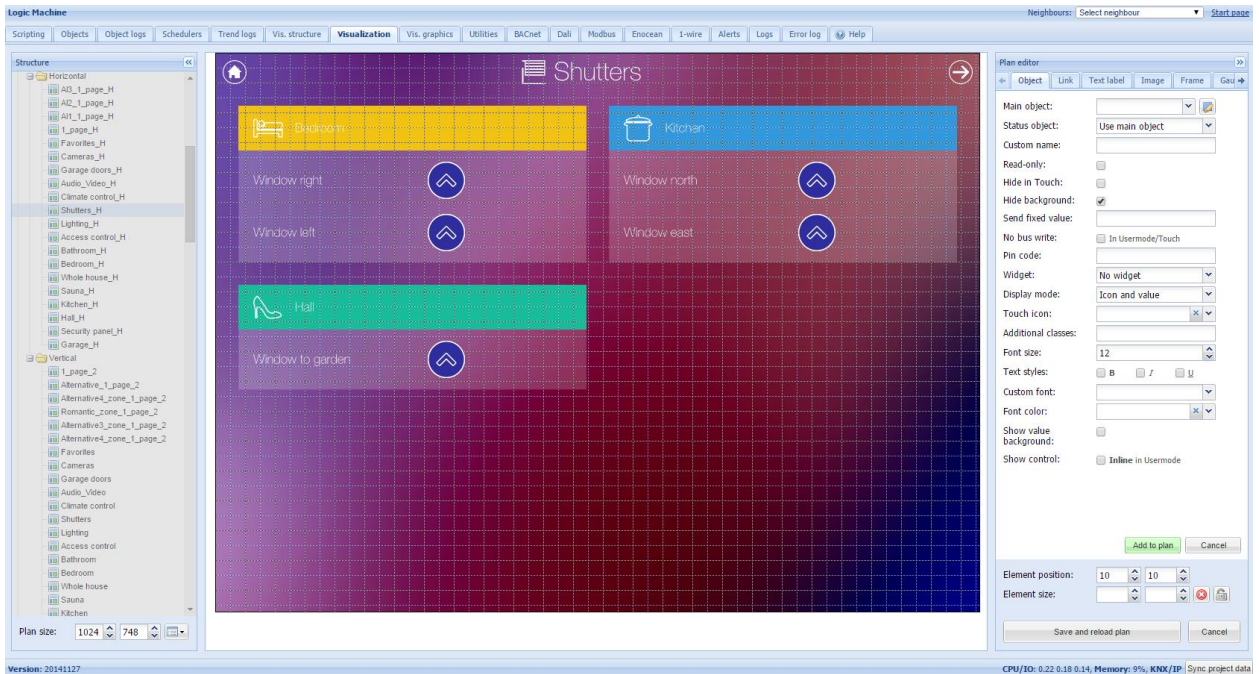
Once added, you can try out the widget in *Usermode visualization* by clicking on added object (temperature sensor icon on the left), the widget appears on click.



6.8. Visualization

After the building and floor structure is defined in *Vis.structure* tab, it is visualized in *Visualization* tab. Controlled and monitored objects can be added and managed in this section.

Both side bars can be minimized by pressing on  icon making the map more visible especially on small displays.



6.8.1. Plan editor

Plan editor is located on the right side of the visualization map. By clicking on *Unlock current plan for editing* button, the following main menus appear for configuration:

- **Object** – new object to be added to the map
- **Link** – linking several floors with special icons
- **Text Label** – text label to put on visualization

- **Image** – Add specific image on the visualization
- **Frame** – add frame object to the visualization
- **Gauge** – Metering gauge
- **Camera** – IP web camera integration into visualization
- **Graph** – Real-time graph to monitor value of scale-type objects

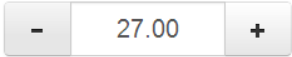
While in editing mode, on the left side you can change plan resolution on the fly



When some object is selected and in the editing mode, there appears Delete / Duplicate buttons so you can either delete or copy the object



6.8.2. Object

- **Main object** – list of existing group addresses on KNX/EIB bus, the ones available for configuration in *Objects* tab
- **Status object** – list of status objects on KNX/EIB bus
- **Custom name** – Name for the object
- **Read-only** – the object is read-only, no write permission
- **Hide in touch** – do not show this object in *Touch Visualization*
- **Hide background** – Hide icon background
- **Send fixed value** – Allows to send specific value to the bus each time the object is pressed
- **No bus write** – do not send telegram into the bus once clicked on this object in Usermode/Touch visualizations
- **PIN code** – PIN code which will be asked to provide when click on this object to perform group write
- **Widget** – specify widget which will be launched when click on this object
- **Display mode [icon and value; icon; value]** – how to display the object
- **Touch icon** – icon for Touch visualization
- **On icon** – On state icon for binary-type objects. Icons library is located in *Vis.graphics* → *Icons tab*
- **Off icon** – Off state icon for binary-type objects. Icons library is located in *Vis.graphics* → *Icons tab*
- **Additional classes** – additional CSS classes for the element
- **Show control** – scale-type object either to show the control in  without icon specific setting defining Usermode visualization
- **Show value background** – show value background color

Plan editor >>

← Object Link Text label Image Frame Gau →

Main object:

Status object: Use main object

Custom name:

Read-only:

Hide in Touch:

Hide background:

Send fixed value:

No bus write: In Usermode/Touch

Pin code:

Widget: No widget

Display mode: Icon and value

Touch icon:

Additional classes:

Font size: 12

Text styles: B I U

Custom font:

Font color:

Show value background:

Show control: Inline in Usermode

Element position:

Element size:

➤ **Visualization parameters**

Defines global or local visualization parameter for specific group address.

Select which parameters to edit

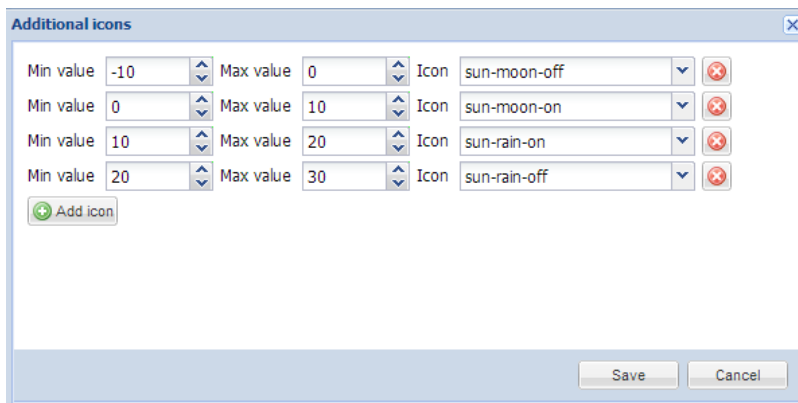
Global (per-object) parameters – specify type of visualization parameter for this specific group address as seen in point 6.2.3. Once specified as global, the same visualization parameter will be used with this group address.



Local (per-element) parameters – specify type of visualization parameter for this specific group address for this specific element as seen in point 6.2.3. Once specified as local, it will only be used in this specific place.

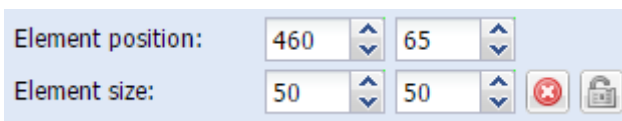
Override global parameters with local – override global parameter for this specific element to local

Clear local parameters – clear all local parameters

For scale-type objects additional button appears while specifying parameters – *Additional icons*. It's possible to define different icons for different object values in the window.



On the bottom of setting you can see element position and size parameters, which you can freely change. By pressing  you will reset size. By pressing  you can lock aspect ratio.



Once the object parameters are defined, press *Add to plan* button and newly created object will appear. You can move the object to the location it will be located. Note that while being in editing mode, the object will not work. When all necessary objects are added, press *Save and reload plan* button so the objects starts functioning.

You can edit each added object when clicking on it while in Editing mode.

6.8.3. Link

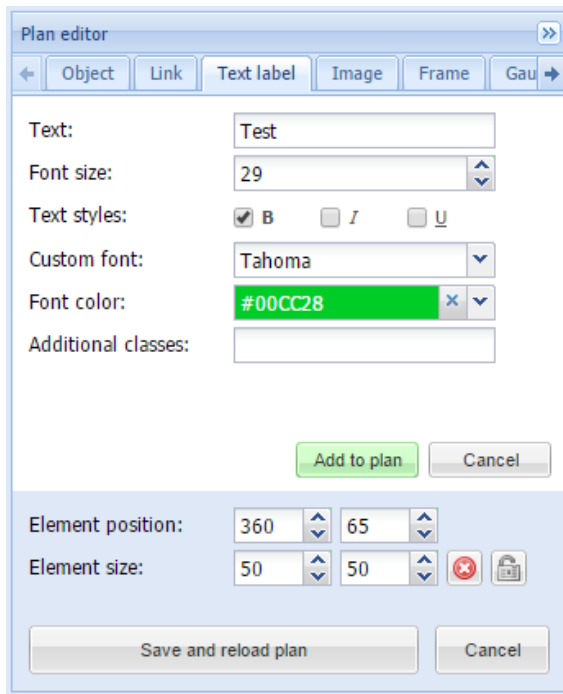
In order to make visualization more convenient, there are floor links integrated. You can add icons or text on the map, which links to other floors.

- **Link to** – Linked plan name or link to Schedulers / Trends or External Link (use the link in form <http://www.openrb.com>)
- **Custom name** – name for the link
- **Hide in touch** – do not show this object in *Touch Visualization*
- **Hide background**– Hide icon background
- **Display mode [Icon; Value]** – either to show icon or its value
- **Icon** – Icon which will be showed in visualization (if chosen, no further parameters are available)
- **Active state icon** – active state icon if the link is to current plan (in case you have several smaller plans on one visualization and want to display the current one)
- **Additional classes** – additional CSS classes for the element

Once the floor link parameters are defined, press *Add to plan* button and newly created object will appear. You can move the object to the location it will be located. Note that while being in editing mode, the object will not work. Press on *Save and reload plan* button so the objects starts functioning.

6.8.4. Text Label

Text labels can be added and moved across the visualization map.

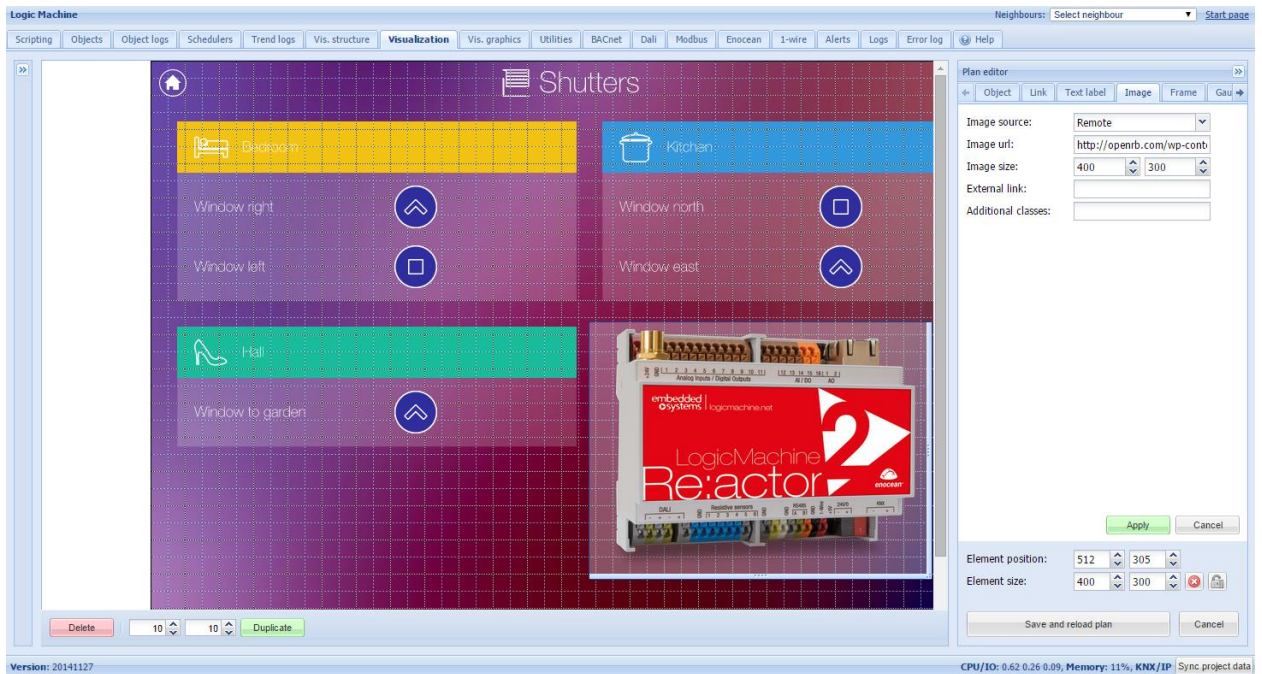


- **Text** – label text
- **Font size** – label font size
- **Text style** – style of the text – bold, italic, underscored
- **Custom font** – font name
- **Font color**– label font color
- **Additional classes** – additional CSS classes for the element

Once the label parameters are defined, press *Add to plan* button and newly created object will appear on the map. You can move the object to the location it will be located. Press on *Save and reload plan* button so the objects starts functioning.

6.8.5. Image

Image section allows adding images from the internet into the visualization map. Useful for example, to grab dynamic weather cast images.



- **Image source [Local; Remote]** – image source location
- **Source url / Select image** – Source URL of the image or image from local database
- **Image size** – width and height of the image
- **External link** – external link URL when pressing on the image
- **Additional classes** – additional CSS classes for the element

Once the image parameters are defined, press *Add to plan* button and newly created object will appear on the map. You can move the object to the location it will be located. Press on *Save and reload plan* button so the objects starts functioning.

6.8.6. Frame

With Frame functionality you can integrate 3rd party applications, we resources or local Trends/Schedulers into one common visualization.

- **Source [Url, Schedulers; Trend logs]** – frame source
- **Url** – Source URL of the page to integrate
- **Frame size** – width and height of the frame
- **Custom name** – custom name of the frame object
- **External link** – external link URL when pressing on the image
- **Hide in Touch** – defines either to hide frame in Touch visualization

➤ **Additional classes** – additional CSS classes for the element

Plan editor

Object Link Text label Image Frame Gau

Source: Schedulers

Frame size: 480 320

Custom name:

Hide in Touch:

Additional classes:

Add to plan Cancel

Element position: 10 10

Element size:

Save and reload plan Cancel

The screenshot shows a mobile application interface on the left and the Plan editor dialog box on the right. The application interface displays a list of events with columns for 'Value' and 'Run at', and buttons for 'Add event', 'Edit', and 'Delete'. The Plan editor dialog box is configured for a 'Url' source and includes fields for 'Url', 'Frame size', 'Custom name', 'Hide in Touch', 'Additional classes', 'Element position', and 'Element size'. The 'Add to plan' button is highlighted in green.

Plan editor

Object Link Text label Image Frame Gau

Source: Url

Url:

Frame size: 480 320

Custom name:

Hide in Touch:

Additional classes:

Add to plan Cancel

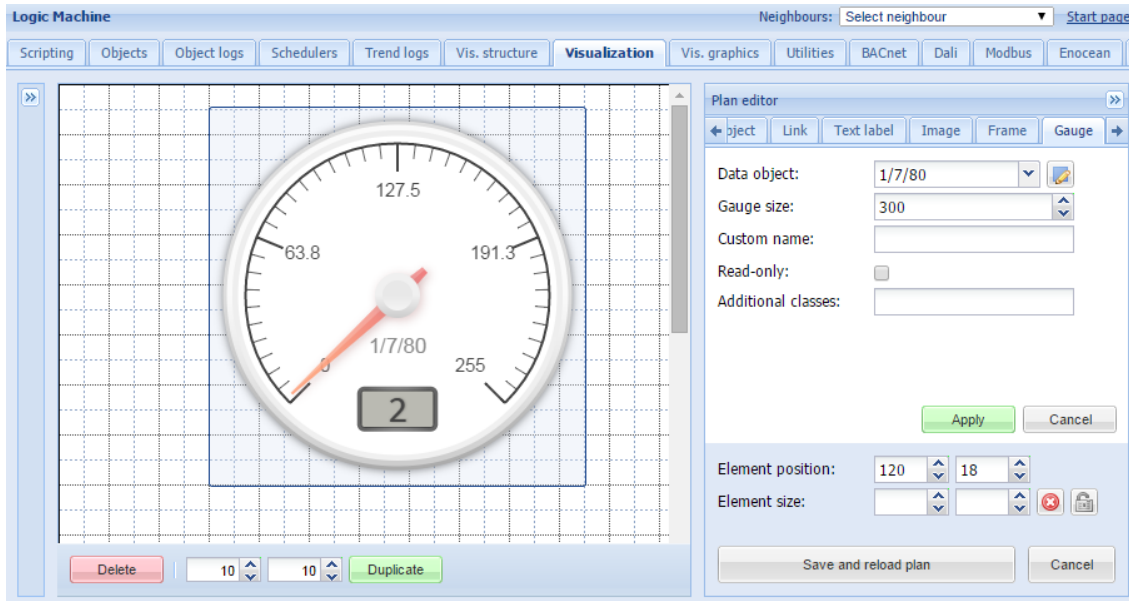
Element position: 10 10

Element size:

Unlock current plan for editing Cancel

6.8.7. Gauge

Gauge allows visualizing and changing object value in the gauge.

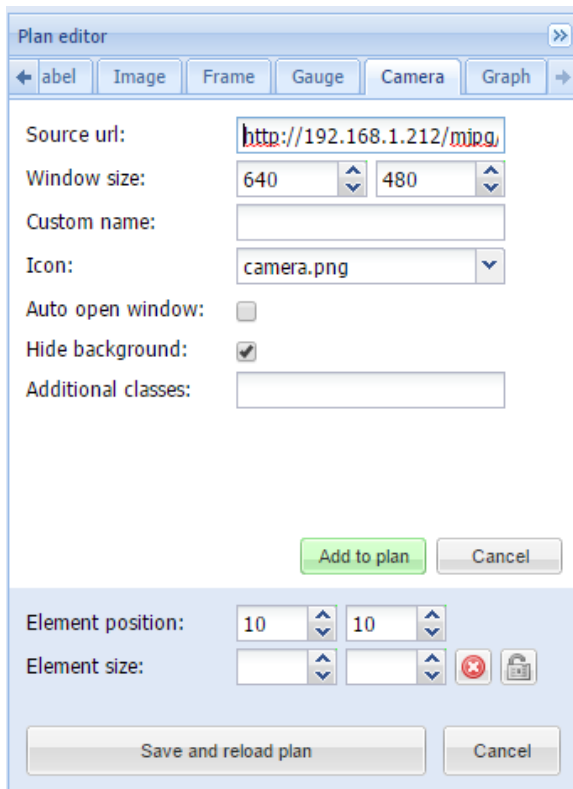


- **Data object** – KNX group address
- **Gauge size** – size of the gauge
- **Custom name** – custom name for the object
- **Read only** – make the gauge read only
- **Additional classes** – additional CSS classes for the element

Once the gauge parameters are defined, press *Add to plan* button and newly created object will appear on the map. You can move the object to the location it will be located. Press on *Save and reload plan* button so the objects starts functioning.

6.8.8. Camera

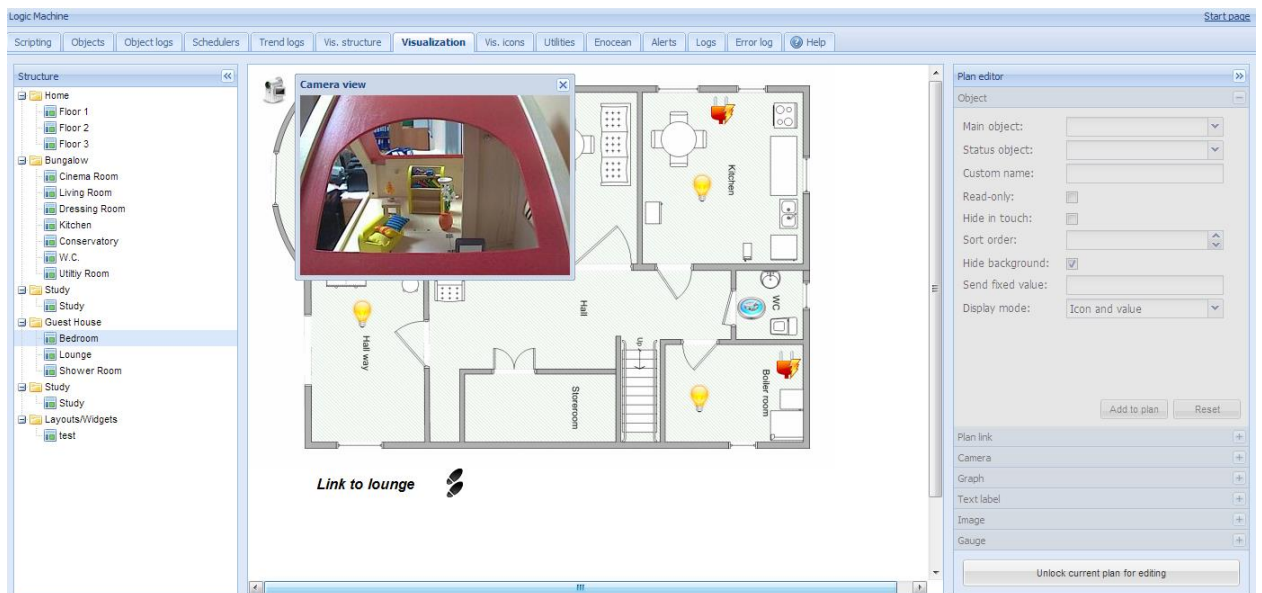
LogicMachine supports third party IP web camera integration into its visualization.



- **Source url** – source address of the video stream
- **Window size** – size of the window of camera picture
- **Custom name** – name for the object
- **Icon** – icon for the object
- **Auto open window** – automatically open video window, otherwise it is launched by click on the icon
- **Hide background**– hide icon background
- **Additional classes** – additional CSS classes for the element

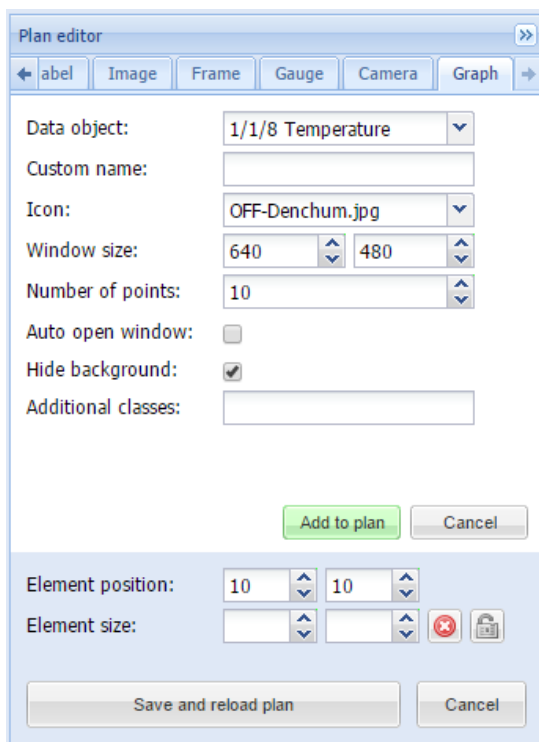
Note! If IP camera requires user name and password, enter the url in form ***http://USER:PASSWORD@IP***

Once the camera parameters are defined, press *Add to plan* button and newly created object will appear in look of video camera. You can move the object to the location it will be located. Note that while being in editing mode, the object will not work. Press on *Save and reload plan* button so the objects starts functioning. By pressing on video camera, a new sub-window appears with a picture from your IP web camera. The window can be freely moved to other location so not to cover other visualization objects.



6.8.9. Graph

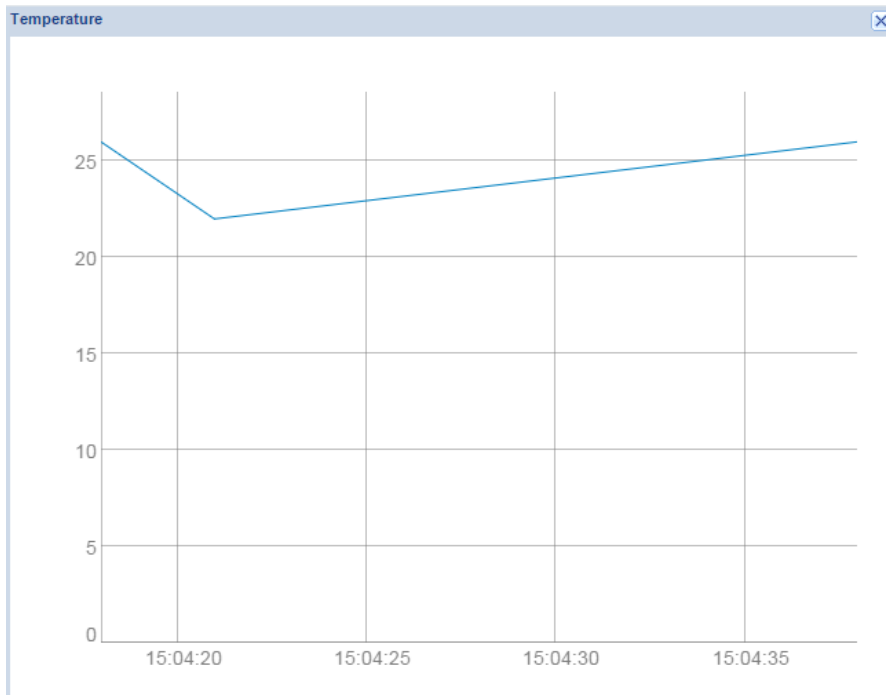
Real-time graphs can be integrated into visualization system to monitor the current and old value of scale-type objects. Make sure logging is enabled for the object in *Object* tab which values is planned to be shown in the graph.



- **Data object** – group address of the object
- **Custom name** – name of the object
- **Icon**– icon to launch the graph
- **Windows size** – size of the graph window

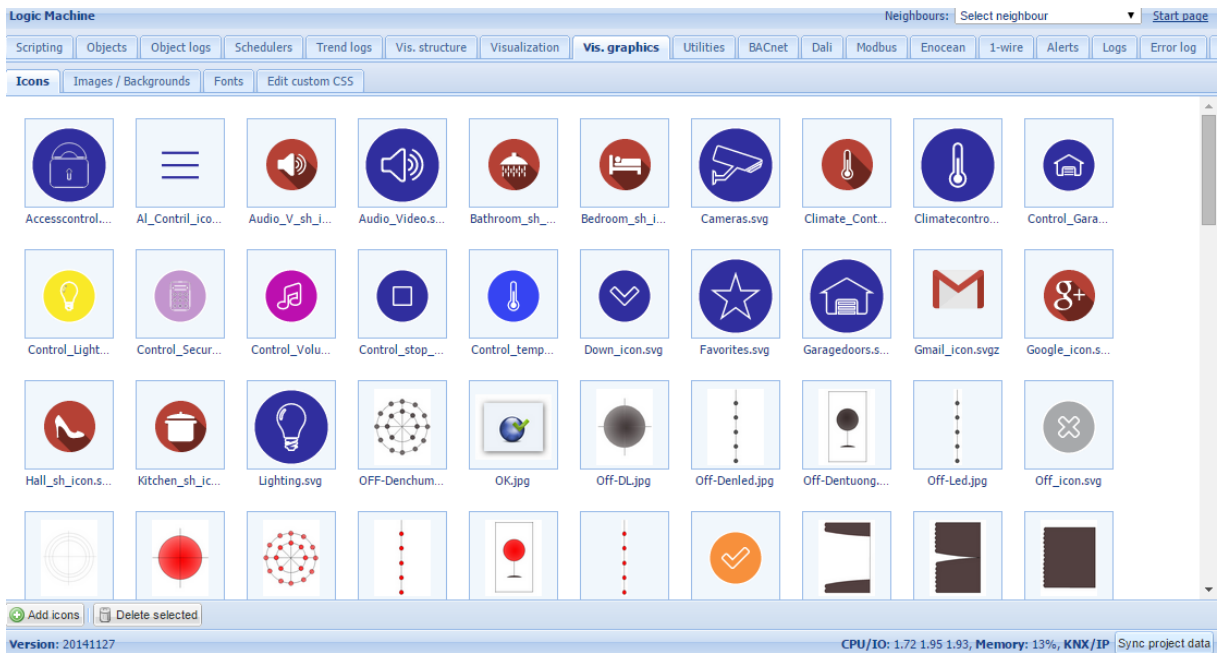
- **Number of points** – number of data points to show in the graph
- **Auto open window** – graph window is automatically opened
- **Hide background** – hide icon background
- **Additional classes** – additional CSS classes for the element

Once the graph parameters are defined, press *Add to plan* button and newly created object will appear. You can move the object to the location it will be located. Note that while being in editing mode, the object will not work. Press on *Save and reload plan* button so the objects starts functioning.

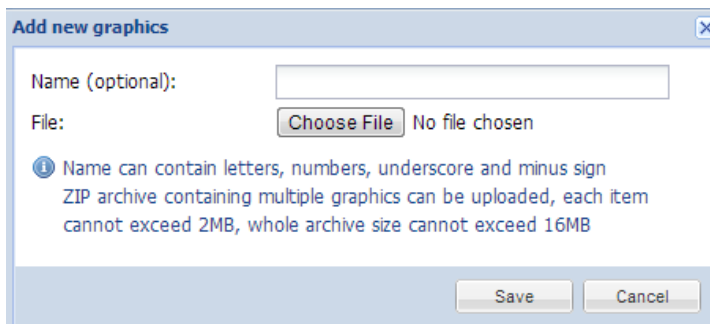


6.9. Vis.graphics

The list of predefined icons, list of images and backgrounds is available in *Vis.graphics* tab.

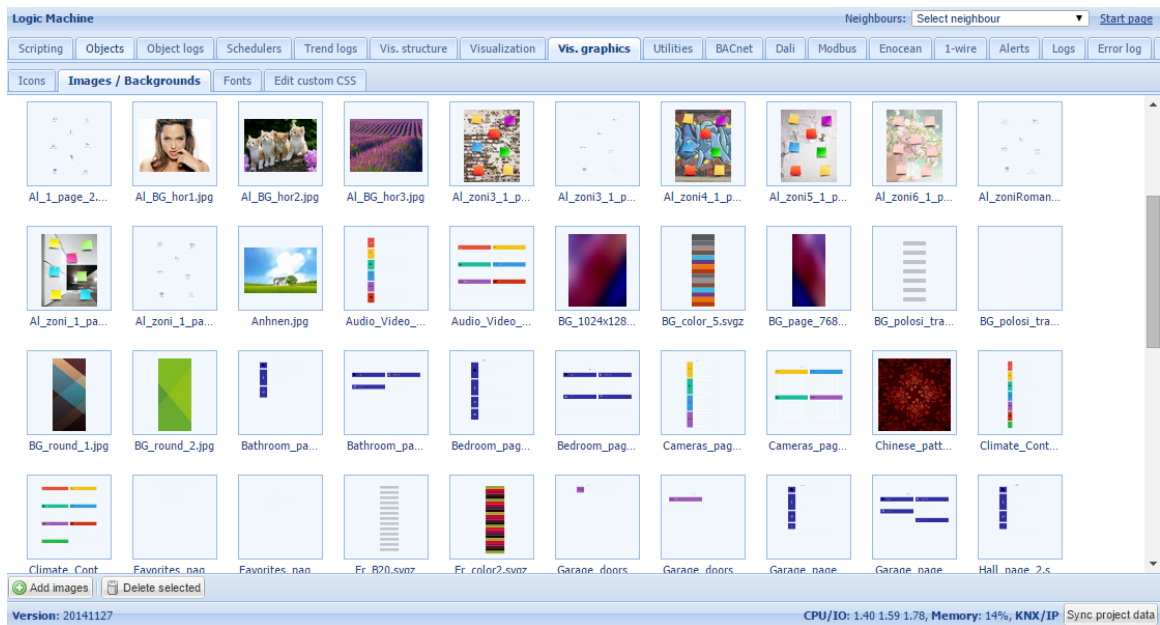


Press on *Add icons* button to add a new entry. The system accepts any size icons. GIF is also supported.

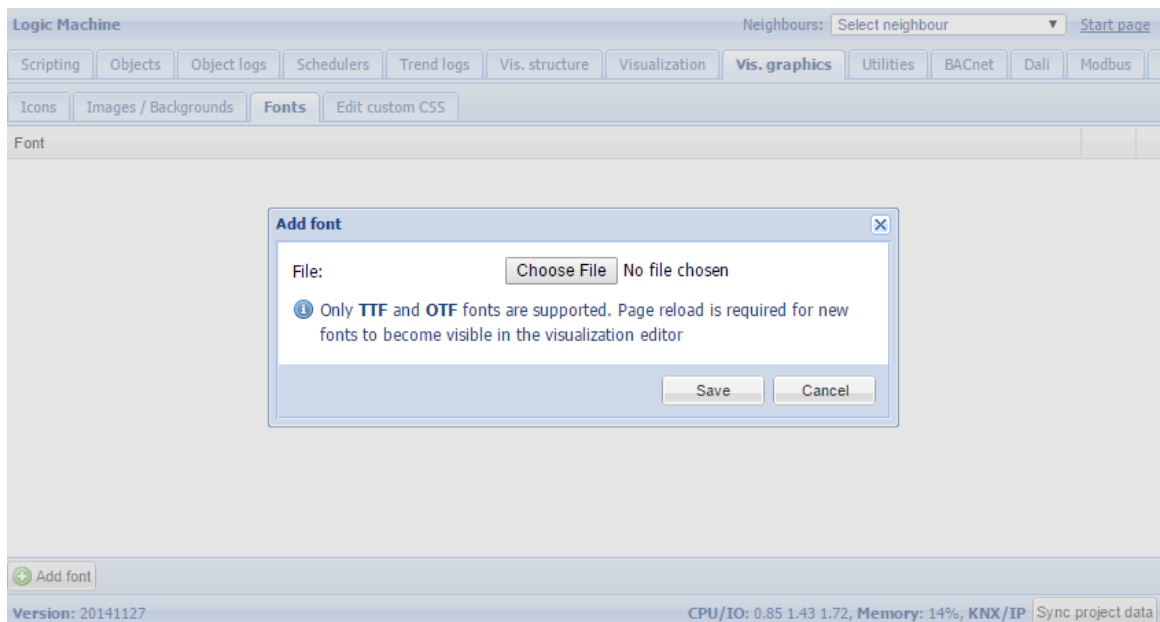


- **Name (optional)** – the name of the icon
- **File** – Icon file location

Images/Backgrounds tab is used to upload image files for visualization purposes



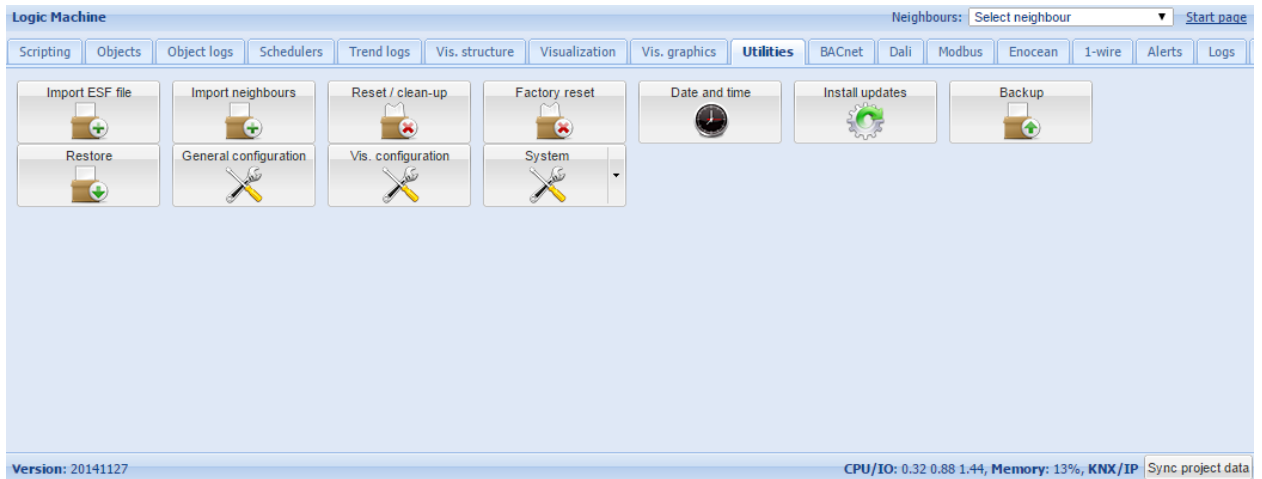
In *Fonts* tab you can add custom fonts



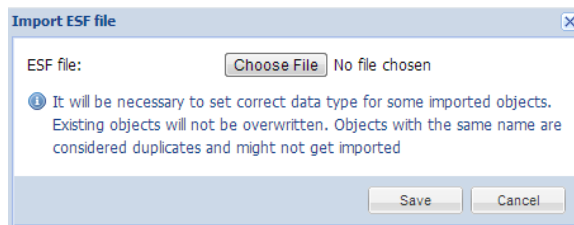
In *Custom CSS* tab you can add your CSS style for the visualization which you can use when adding elements into visualization, so any elements of Look and Feel is customizable with this solution.

6.10. Utilities

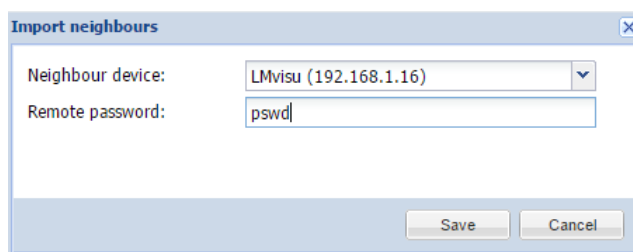
There are following utilities in the tab available:



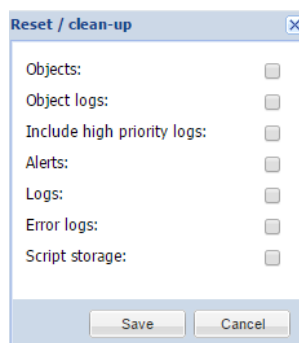
Import ESF file– imports ETS object file. It will be necessary to set correct data types for some imported objects. Existing objects will not be overwritten. Objects with the same name are considered duplicates and might not be imported



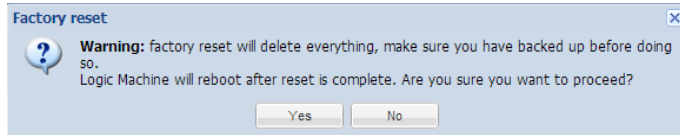
Import neighbours – import list of objects from network LM devices



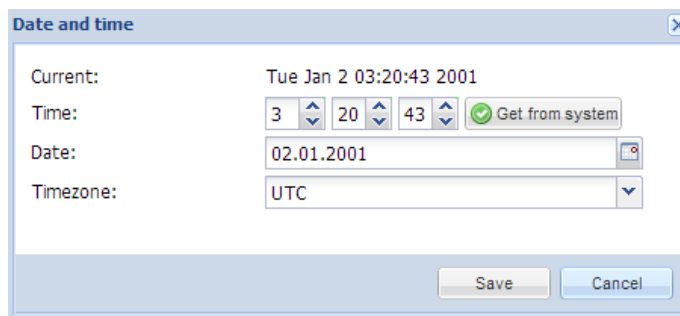
Reset / clean-up – delete all objects from the Logic Machine, they disappear from visualization as well



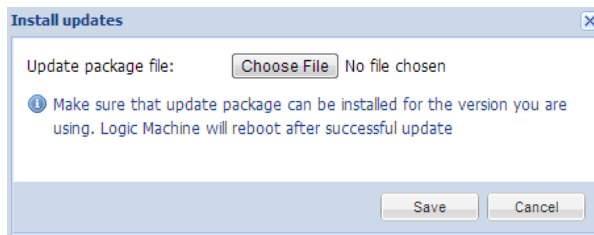
Factory reset– delete all configuration and return to factory defaults



Date and time – data and time settings

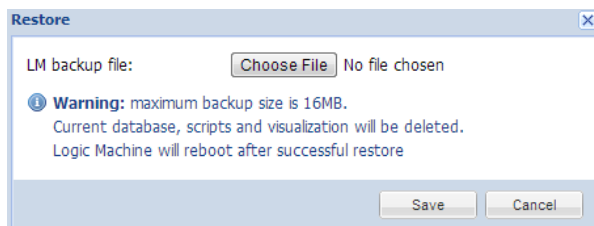


Install updates – install LogicMachine update file *.lmu. LogicMachine will reboot after successful update

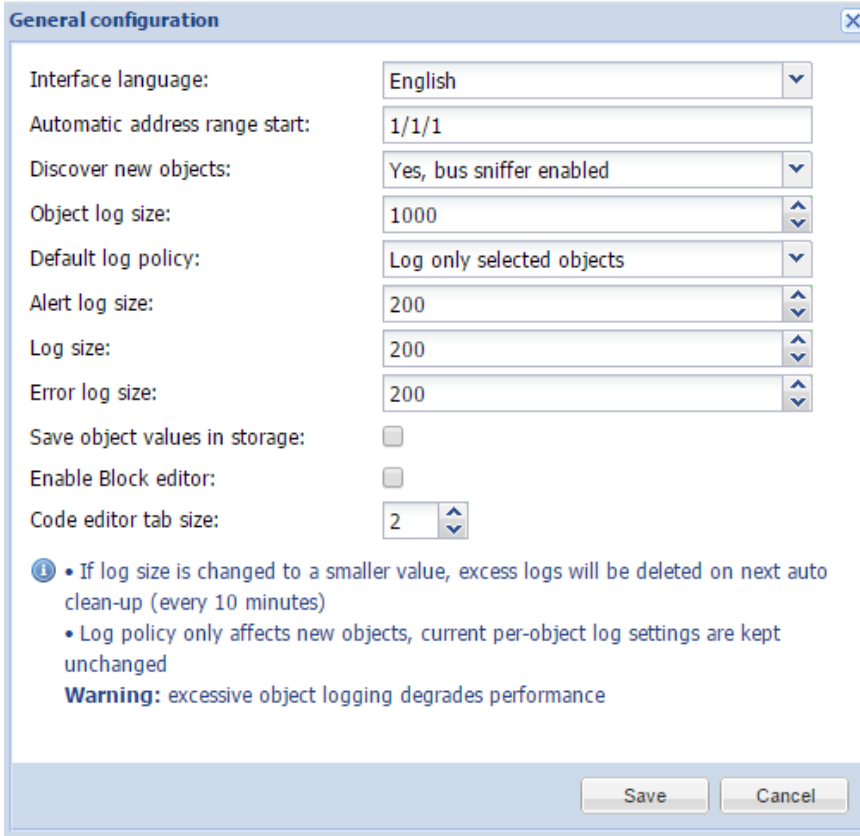


Backup – backup all objects, logs, scripts, visualization.

Restore– restore configuration from backup



General Configuration – system general settings



The screenshot shows a 'General configuration' dialog box with the following settings:

- Interface language: English
- Automatic address range start: 1/1/1
- Discover new objects: Yes, bus sniffer enabled
- Object log size: 1000
- Default log policy: Log only selected objects
- Alert log size: 200
- Log size: 200
- Error log size: 200
- Save object values in storage:
- Enable Block editor:
- Code editor tab size: 2

Informational text:

- If log size is changed to a smaller value, excess logs will be deleted on next auto clean-up (every 10 minutes)
- Log policy only affects new objects, current per-object log settings are kept unchanged

Warning: excessive object logging degrades performance

Buttons: Save, Cancel

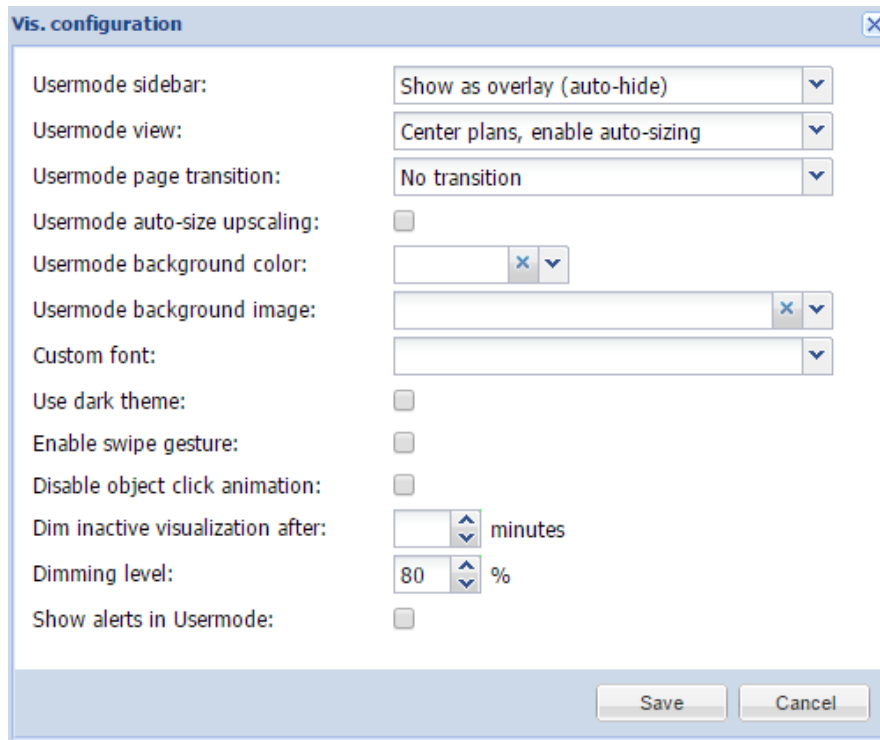
- **Interface language** – interface language
- **Automatic address range start** – start group address when using automatic addressing in scripts, IO settings and other
- **Discover new objects**– either KNX object sniffer is enabled. If yes, once triggered all new objects will appear automatically in the Objects list
- **Object log size** – max count of object logs
- **Default log policy**– either to log status change for all objects or only for checked objects
- **Alert log size** – max count of alerts logged
- **Log size** – max count of logs
- **Error log size** – max count of errors logged
- **Enable block editor** – either to enable scripting block editor
- **Save object values in storage** – save object values in REDIS database to access from apps
- **Code editor tab size** – specify tab size to be used in the scripting editor

Note! If log size is changed to a smaller value, excess logs will be deleted on next auto clean-up (every 10 minutes)

Note! Log policy only affects new objects, current per-object log settings are kept unchanged

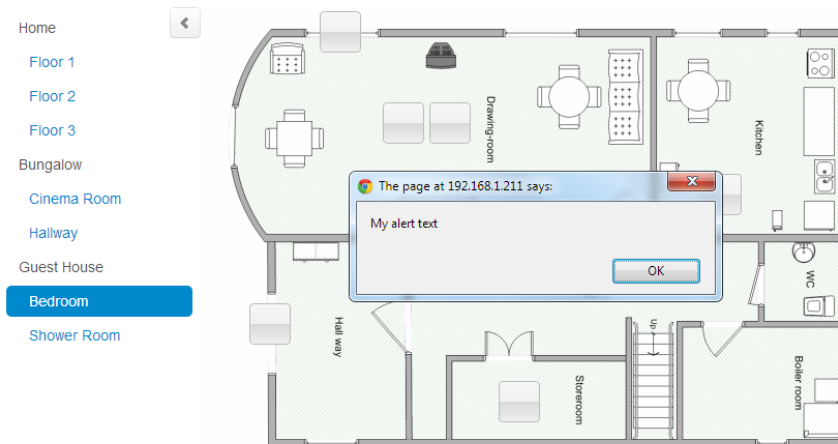
Warning! Excessive object logging degrades LogicMachine performance. Please follow this example to store logs on local FTP or automatically export to external FTP server:

Vis. Configuration – visualization specific settings

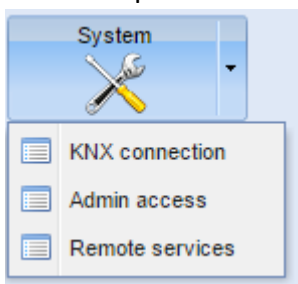


- **Usermode sidebar** [*Show docked, Show as overlay (auto-hide), Hide (fullscreen mode)*] – visibility of sidebar when in Usermode Visualization
- **Usermode view** [*Align plans to top left, no size limit; Center plans, limit size; Center plans, enable auto-sizing; Center horizontally, auto-size width*] – defines the look of Usermode visualization
- **Usermode page transition** [*Flip X; Flip Y; Shrink; Expand; Slide up; Slide down, Slide left; Slide right; Slide up big; Slide down big; Slide left big; Slide right big*] – transition when changing plans in visualization
- **Usermode auto-size upscaling** – enable this to scale the visualization automatically on each display device. Please note to use SVG format images and icons so the quality is not affected by upscaling
- **Usermode background color** – background color in usermode visualization
- **Usermode background image** – specific image for usermode visualization
- **Custom font** – select custom font to use in visualization
- **Use dark theme** – check to enable dark theme in both usermode and touch visualizations
- **Enable swipe gesture** – check to enable swipe gesture to move across plans from your touch device
- **Disable object click animation** – disable object click animation
- **Dim inactive visualization after** – define time in minutes after which the screen will be dimmed where visualization is opened

- **Dim level** – dim level for the display
- **Show alerts in Usermode** – once new Alerts is triggered it will pop-up in User mode visualization



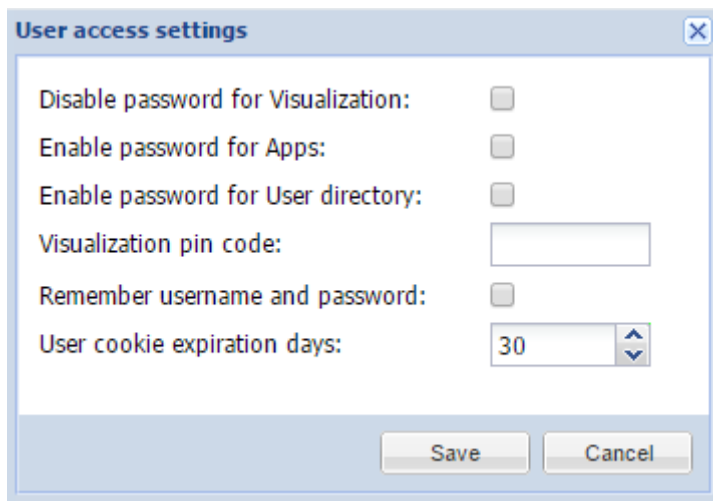
System – by clicking on the arrow near System button, *KNX Connection*, *User Access*, *Remote Services* settings can be access. By clicking on the *System* button, network configuration window opens in new browser’s tab.



6.11. User access

User access management is located in *User access* tab.

[User access settings](#)



- **Disable password for Visualization** – disable password access for visualization
- **Enable password for Apps** – enable password to enter the initial Apps screen of LogicMachine (when entering <http://IP> in the web-browser)
- **Enable password for User directory** – enable password access for User directory
- **Visualization PIN code** – global PIN code visualization
- **Remember username and password** – remember user login credentials after entered once for specified time interval
- **User cookie expiration days** – user cookie expiration days

User directory

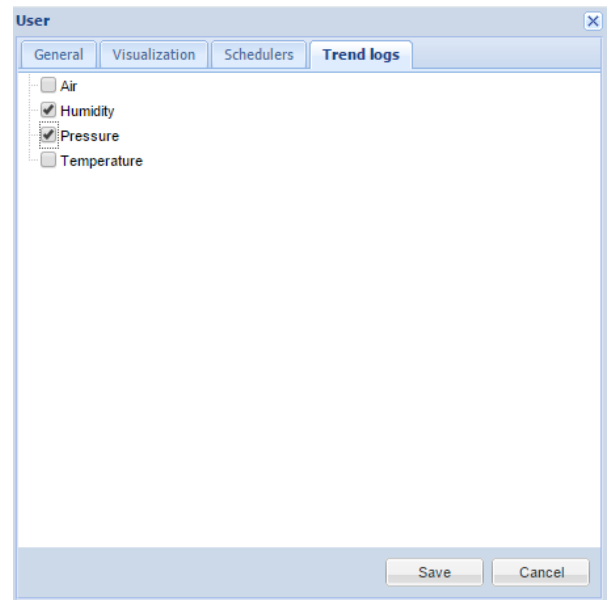
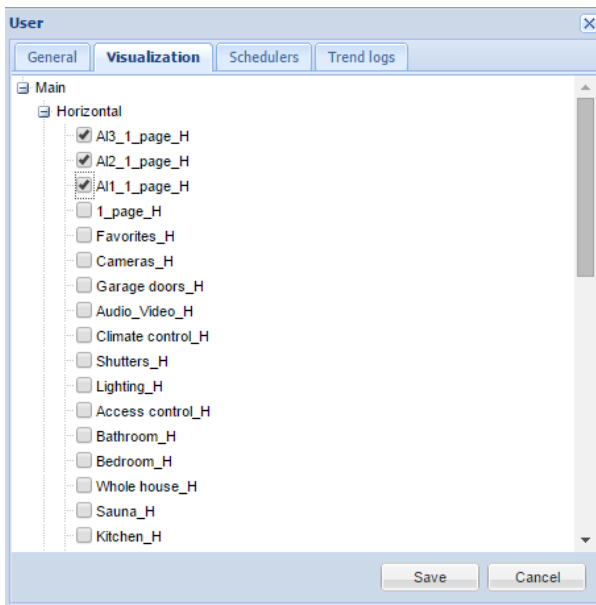
You can upload files which are accessible through the main web server via FTP. In *System config* --> *Services* --> *FTP server* you have to enable the FTP server and set password for apps user. Then you can upload files into user directory which can then be accessed at <http://IP/user>. Password authentication for this directory can be enabled/disabled in *Logic Machine* --> *User access* --> *User access settings*.

Adding users

The screenshot shows a 'User' configuration window with the following fields and values:

Field	Value
Name:	Edgars
Login:	edgars
Password:	•••••
Repeat password:	•••••
Visualization access:	Partial
Schedulers access:	Partial
Trends access:	Partial

- **Name** – name of the user
- **Login** – login name
- **Password** – password
- **Repeat password** – repeat password
- **Visualization access [None, Partial, Full]** – type of Visualization access
- **Schedulers access [None, Partial, Full]** – type of Schedulers access
- **Trends access [None, Partial, Full]** – type of Trends access
 - None* – access is limited
 - Partial* – access is granted for specific visualization floors, schedulers and trends
 - Full* – full access



Access logs

Shows a list of access logs

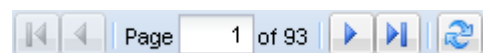
6.12. Alerts

In *Alert* tab a list of alert messages defined with **alert** function in scripts is located. The messages are stored on the compact flash. Information on system start and KNX connection status messages are also automatically displayed in this window.

Alert time	Message
01.01.1970 10:20:42	read error
01.01.1970 10:20:22	read error
01.01.1970 10:20:02	read error
01.01.1970 10:12:58	read error

Page 1 of 93 Displaying alerts 1 - 25 of 2317

On the communication panel you can jump by pages and reload the page.



Example

1. temperature = 25.3
- 2.

```

3. if temperature > 24 then
4. -- resulting message: 'Temperature Levels are too high: 25.3'
5. alert('Temperature level is too high: %.1f', temperature)
6. end

```

6.13. Error log

Error messages from scripts are displayed in *Error log* tab.

The screenshot shows the Logic Machine interface with the 'Error log' tab selected. The interface includes a navigation bar with tabs for Scripting, Objects, Object logs, Schedulers, Trend logs, Vis. structure, Visualization, Vis. icons, Utilities, Enocan, Alerts, Logs, Error log, and Help. The main area displays a table of error messages:

Error time	Script name	Error description
22.02.2013 09:29:51	init-script	Line 6: attempt to index global 'temperature' (a nil value)
21.02.2013 08:08:46	weather_data_Yahoo	Line 20: attempt to index field 'current' (a nil value)
16.02.2013 07:12:08	weather_data_Yahoo	Line 20: attempt to index field 'current' (a nil value)
15.02.2013 23:51:55	weather_data_Yahoo	Line 20: attempt to index field 'current' (a nil value)
12.02.2013 15:23:39	init-script	Line 6: attempt to index global 'temperature' (a nil value)
11.02.2013 18:48:30	init-script	Line 6: attempt to index global 'temperature' (a nil value)
11.02.2013 17:47:40	init-script	Line 6: attempt to index global 'temperature' (a nil value)
08.02.2013 20:00:02	event-Volume down	cannot open /lib/genohm-scada/scripting/57.lua: No such file or directory
08.02.2013 13:52:11	init-script	Line 6: attempt to index global 'temperature' (a nil value)

At the bottom of the window, there is a 'Clear' button, navigation arrows, and a page indicator 'Page 1 of 8'. The status bar at the bottom right indicates 'Displaying errors 1 - 25 of 200'.

6.14. Logs

Logs can be used for scripting code debugging. The log messages appear defined by *log* function.

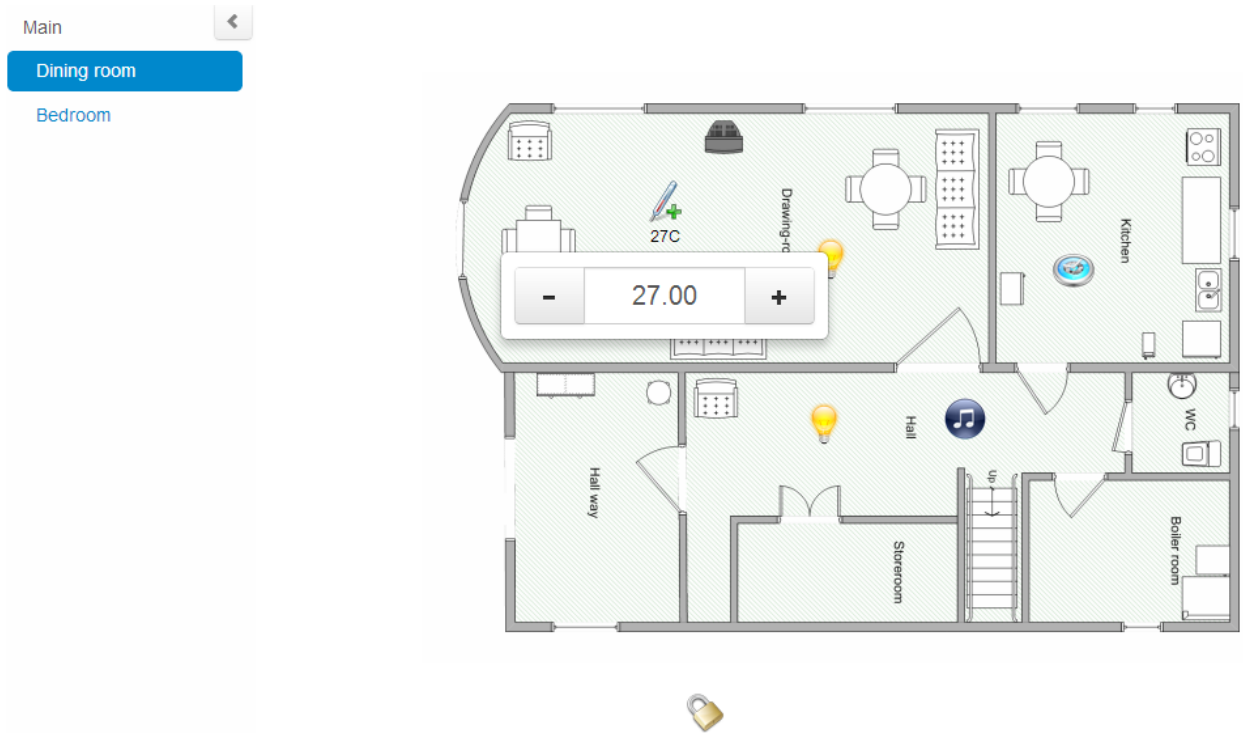
The screenshot shows the Logic Machine interface with the 'Logs' tab selected. The interface includes a navigation bar with tabs for Scripting, Objects, Object logs, Buildings, Visualization, Visualization icons, Utilities, Alerts, Logs, Error log, and Help. The main area displays a table of log messages:

Log time	Message
15.05.2012 14:20:33	* arg: 1 * table: {f2} * number: 20 {f1} * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
15.05.2012 14:20:28	* arg: 1 * table: {f2} * number: 20 {f1} * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
15.05.2012 14:20:23	* arg: 1 * table: {f2} * number: 20 {f1} * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
15.05.2012 14:20:18	* arg: 1 * table: {f2} * number: 20 {f1} * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test

At the bottom of the window, there is a 'Clear' button, navigation arrows, and a page indicator 'Page 1 of 1'. The status bar at the bottom right indicates 'Displaying logs 1 - 4 of 4'. The footer shows 'Version: 20120419' and '© Embedded Systems 2012'.

7. User mode visualization

User mode visualization contains created visualization maps. A password and users to access specific visualization maps can be created in *Logic Machine* --> *User access*



7.1. Custom design Usermode visualization

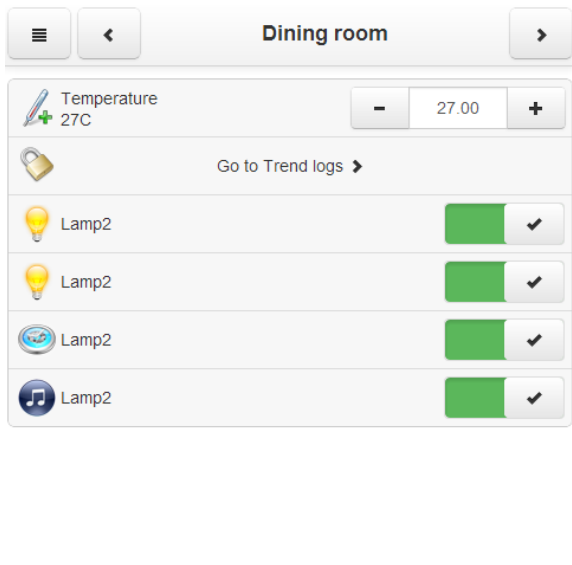
Through Custom CSS styles it is possible to create different type of visualization maps. Custom CSS can be done in *Vis. Graphics* → *Edit custom CSS* tab. For more information of CSS examples please see our user forum: <http://forum.logicmachine.net/>



8. Touch visualization

Touch visualization is designed for iPhone/iPod/iPad/Android touch screen devices. All objects which are added in *Logic Machine* configuration by default are visible in touch visualization (if there is no *Hide in touch* option enabled).

The main window is Building view where you can choose which Floor from which Building to control. Once you choose the floor, all objects which are assigned to it, are listed and can be controlled.



Launching visualization on touch device (iPad in this case)

- Make sure your iPad is connected wirelessly to the LogicMachine (either through separate access point or directly to Logic Machine's USB WiFi adapter).
- In the browser enter Logic Machine's IP (default 192.168.0.10).
- Click on the Touch Visualization icon.
- Save the application as permanent/shortcut in your iPad

9. System configuration

System configuration allows managing router functionality on KNX/EIB LogicMachine as well as do access control management, upgrade firmware, see network and system status and others.

The screenshot displays the system configuration interface with the following components:

- System Configuration Menu:** System, Network, Services, Status, Help
- Interfaces Window:**

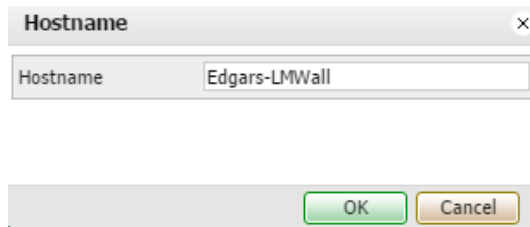
Name	Mac address	Mtu	TX Bytes	RX Bytes	Errors
eth0	00:1B:C5:00:1D:12	1500	0 B	7 MB	0 / 0
- Network usage for interface eth0 Window:**
 - In 5 Kbps
 - Out 0 Kbps
 - Switch to bytes/s
 - AutoScale (follow)
 - Graph showing network usage over time with a peak near 30 Kbps.
- Routes Window:**
 - Dynamic | Static
 - Table:

Interface	Destination
eth0	192.168.0.0
eth0	224.0.0.0

Login	Password
admin	admin

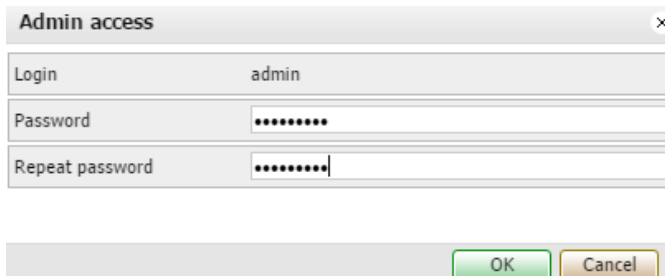
9.1. Hostname

Hostname can be change in *System* → *Hostname*. This name will appear when searching for the device through Zeroconf or Discovery applications.



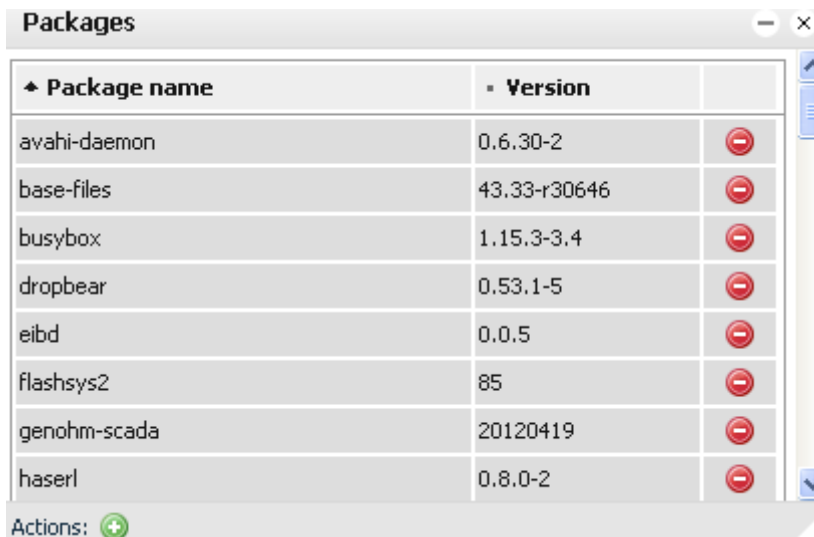
9.2. Changing Admin password

The admin password configuration window is located in *System* → *Admin access*.



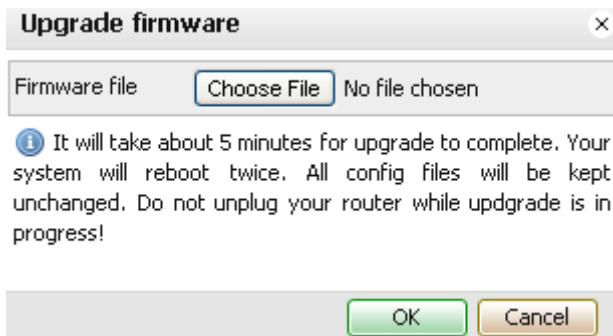
9.3. Packages

System → *Packages* shows the packages installed in the system. You can add new packaged by pressing on +



9.4. Upgrade firmware

System → *Upgrade firmware* is used to do a full upgrade of the system (both OS part as well as LogicMachine part).



9.5. Reboot Logic Machine

You can restart the LogicMachine by executing *System* → *Reboot* command.

9.6. Shutdown Logic Machine

You can shutdown the LogicMachine by executing *System* → *Shutdown* command. It is advisable to shutdown the system before plug out the power, because the database is saved safely.

9.7. Interface configuration

Ethernet interface is listed in the first tab. There are possibilities to disable/enable or to take a look at the traffic flow graph using special icons on the right side.

Name	Mac address	Mtu	TX Bytes	RX Bytes	Errors	
eth0	00:1B:C5:00:1D:12	1500	0 B	7 MB	0 / 0	


By clicking on the interface you get to the configuration.

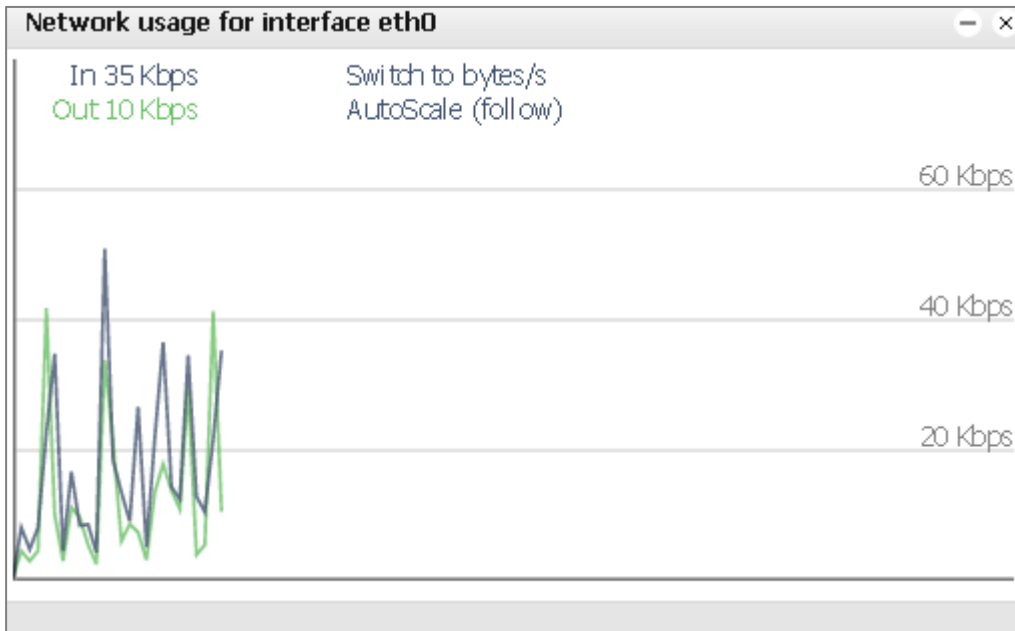
Interface eth0	
Protocol	Static IP
IP address	192.168.1.13
Network mask	255.255.255.0
Gateway IP	192.168.1.100
DNS server 1	8.8.8.8
DNS server 2	
MTU	

- **Protocol**– specific protocol used for addressing
 - Static IP** – static IP address. By default 192.168.0.10
 - DHCP** – use DHCP protocol to get IP configuration.
 - Current IP**– the IP address got from DHCP server. This field appears only if the IP address is given otherwise it's hidden.

- **Network mask** – network mask. By default 255.255.255.0 (/24)
- **Gateway IP** – gateway IP address
- **DNS server** – DNS server IP address
- **MTU**– maximum transmission unit, the largest size of the packet which could be passed in the communication protocol. By default 1500

Ethernet interface data throughput graph

On the main window of the Ethernets tab, if you click on the  button, a new window is opened. It draws a real-time graph of the traffic flow passing the interface (both In and Out). There is a possibility to switch the units of measurement – bytes/s or bytes/s.



9.8. KNX connection

KNX specific configuration is located in *Network* → *KNX connection* window.

General tab

The screenshot shows the "KNX connection" window with the "General" tab selected. The configuration fields are as follows:

- Mode: TP-UART
- ACK all group telegrams:
- KNX address: 15.15.255
- KNX IP features:
- Multicast IP: 224.0.23.12
- Multicast TTL: 1
- Maximum telegrams in queue: 100
- TOS priority level (0 = no priority): 0
- Encryption key: [Empty text field]
- Enable only secure communication:

Below the fields is an information icon (i) followed by the text: "Setting Encryption key will enable encryption of routing telegrams. Reception of normal telegrams will still work. Tunneling and non-secure routing is disabled if only secure communication is enabled. All devices must have the same date/time set otherwise encrypted telegrams will be rejected." At the bottom right, there are "OK" and "Cancel" buttons.

- **Mode** [*TP-UART / EIBnet IP Tunneling / EIBnet IP Tunneling(NAT mode) / EIBnet IP Routing*] – KNX connection mode. LogicMachine5 has TPUART interface by default built-in. **Note!** If there is no KNX TP connected to the device, it will automatically offer to switch to KNXnet/IP mode.
- **ACK all group telegrams** – acknowledge receipt of telegram to all group communication
- **KNX address** – KNX physical address of the device
- **KNX IP features** – Use this device with KNX IP features e.g. for KNX device configuration from ETS using LogicMachine
- **Multicast IP** – multicast IP address
- **Multicast TTL** – Time to live for multicast telegram in seconds
- **Maximum telegrams in queue** – count of maximum telegrams in the queue
- **TOS priority level (0=no priority)** – type of service in the telegram. See more here: https://en.wikipedia.org/wiki/Type_of_service
- **Encryption key** – key for secure IP communication along LogicMachines. Setting Encryption key will enable encryption of routing telegrams. Reception of normal telegrams will still work. Tunneling and non-secure routing is disabled if only secure communication is enabled. All devices must have the same date/time set otherwise encrypted telegrams will be rejected.
- **Enable only secure communication** – define either only encrypted communication is accepted

IP > TP filter

Filtering table for telegrams going from IP network to KNX TP1 is located in this submenu.

KNX connection [Close]

General | **IP > TP filter** | TP > IP filter

Apply filter to tunneling

SRC policy: No filter

Ind. address list: [Empty list box]

i One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.

DST group policy: No filter

Group address list: 1/1/1-1/1/2

i One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.
Note: KNX IP features are required for filter to work.
 Filtering lists are updated at once, changing policies requires restart.

OK Cancel

- **Apply filter to tunneling** – either to apply filter policy to telegrams in tunneling mode. If ETS is used it is recommended to turn this feature off.
- **SRC policy** [No filter / Accept selected individual addresses / Drop selected individual addresses]– policy to apply to the list of source addresses
- **Ind. address list** – list of individual addresses. One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.
- **DST group policy**[No filter / Accept selected group addresses / Drop selected group addresses]– policy to apply to the list of destination group addresses
- **Group address list** – list of group addresses. One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.

Note! KNX IP features should be on for filter to work. Filtering lists are updated at once, changing policies requires restart.

Note that group address list can be filled automatically by checking necessary group addresses in *LogicMachine* → *Objects* list

Logic Machine

Reactor Scripting **Objects** Object logs Schedulers Trend logs Vis. structure Visualization Vis. graphics Utilities Modbus EnOcean Alerts Logs Error log Help

Object filter	Group address	Object name	IP > TP filter	TP > IP filter	Event script	Data type	Current value	Log	Export
Name or group address: <input type="text"/>	1/1/1	Digital output 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		01.001 switch	on	<input type="checkbox"/>	<input type="checkbox"/>
Data type: <input type="text"/>	1/1/2	Digital output 16	<input checked="" type="checkbox"/>	<input type="checkbox"/>		01.1 bit (boolean)	1	<input type="checkbox"/>	<input type="checkbox"/>
	1/1/3	Digital output 2	<input type="checkbox"/>	<input type="checkbox"/>		01.1 bit (boolean)	1	<input type="checkbox"/>	<input type="checkbox"/>
	1/1/4	Digital output 3	<input type="checkbox"/>	<input type="checkbox"/>		01.001 switch	on	<input type="checkbox"/>	<input type="checkbox"/>

TP > IP filter

Filtering table for telegrams going from KNX TP1 to IP network is located in this submenu.

KNX connection ×

General **IP > TP filter** TP > IP filter

Apply filter to virtual objects

SRC policy No filter ▼

Ind. address list

i One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.

DST group policy No filter ▼

Group address list

1/1/1

i One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.

Note: KNX IP features are required for filter to work.
Filtering lists are updated at once, changing policies requires restart.

OK
Cancel

- **Apply filter to virtual objects** – either to apply filter policy to objects added in Objects tab as virtual objects without attraction to bus

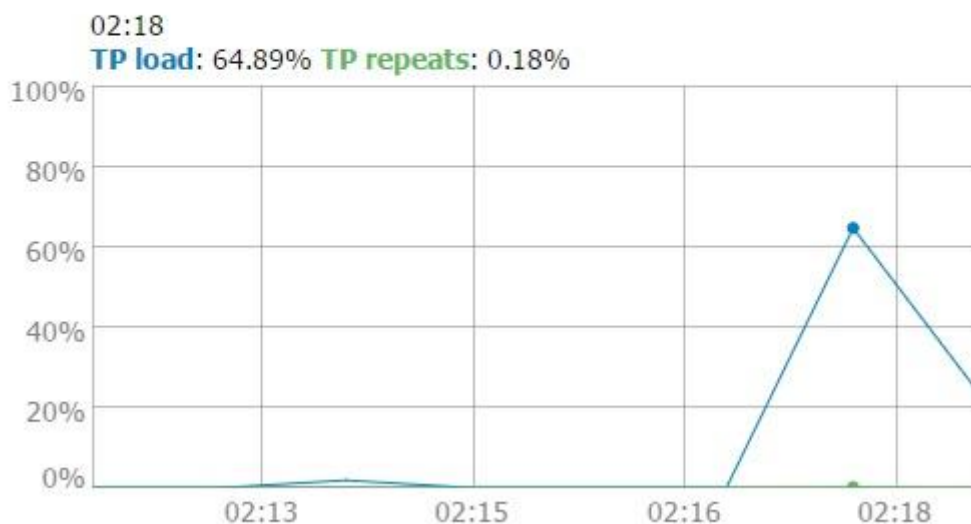
- **SRC policy** [No filter / Accept selected individual addresses / Drop selected individual addresses]– policy to apply to the list of source individual addresses
- **Ind. address list** – list of individual addresses. One address/range per line. Use * (e.g. 1.1.*) to filter all addresses in the given line.
- **DST group policy** [No filter / Accept selected group addresses / Drop selected group addresses]– policy to apply to the list of destination group addresses
- **Group address list** – list of group addresses. One address/range per line. Use * (e.g. 1/1/*) to filter all addresses in the given line.

Note! *KNX IP features* should be on for filter to work. Filtering lists are updated at once, changing policies requires restart.

9.9. KNX statistics

KNX related statistics can be found in *Network* → *KNX statistics* menu.

Period	TP load	TP repeats	TP RX/TX	IP RX/TX
Last minute	23.34%	0	86 / 124	124 / 74
Last hour	11.27%	1	325 / 486	486 / 280
Total	11.27%	1	325 / 486	486 / 280



9.10. BACnet settings

BACnet server specific configuration can be done in *Network* → *BACnet Settings*

Server enabled	<input type="checkbox"/>
Device ID	127001
Password	mybacpwd
Object priority	16
Add group address to object name	<input type="checkbox"/>
Port	47808
BBMD IP	
BBMD port	
BBMD lease time (seconds)	

Server enabled – specify if BACnet server is enabled or not

Device ID – device ID in BACnet network

Password – device password

Object priority – object priority

Add group address to object name – add automatically the address to object name

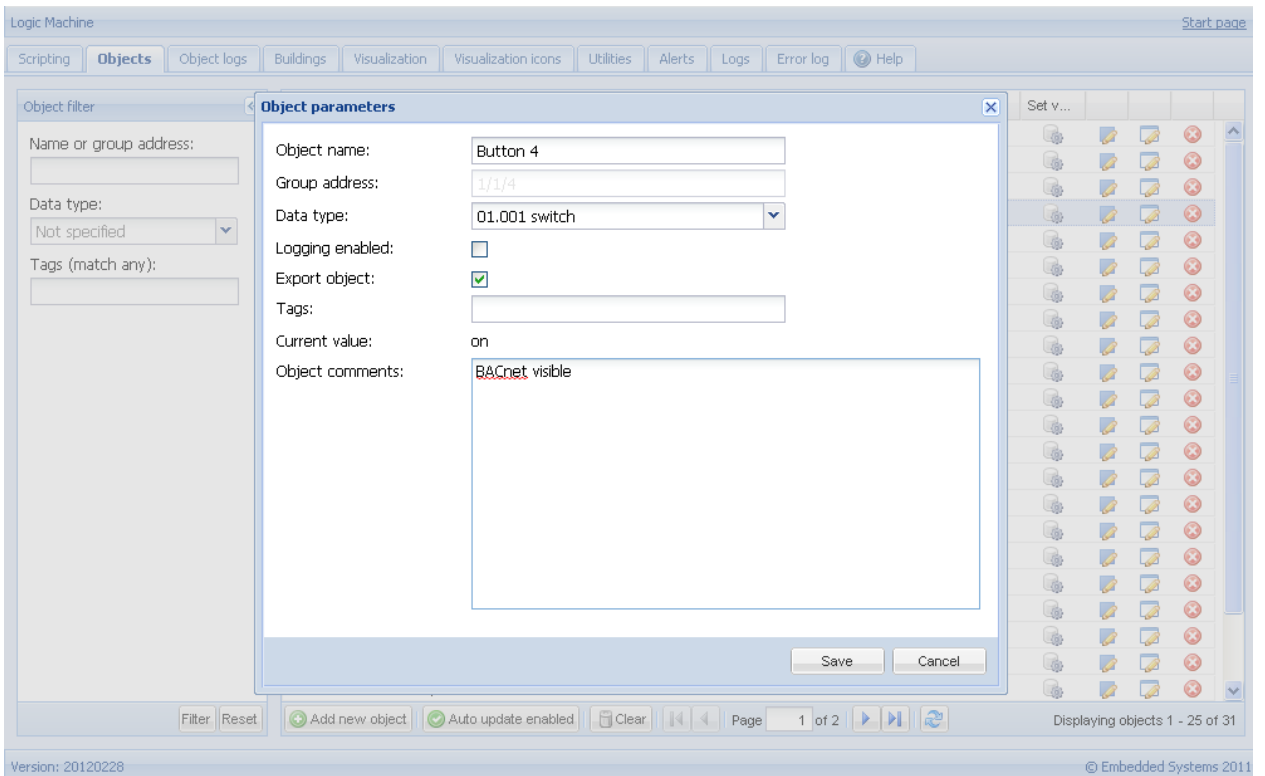
Port – port number

BBMD IP – BACnet router IP. When router IP and port are set, LM will act as a foreign device and will attempt to register with BACnet router.

BBMD port – BACnet router port. When router IP and port are set, LM will act as a foreign device and will attempt to register with BACnet router

BBMD lease time (seconds) – registration resend interval

To make KNX/EIB objects BACnet readable/writable, mark necessary objects in LogicMachine as “Export object”. Binary objects will appear as Binary Values, other numeric values will appear as Analog Values. Other types are not currently supported. KNX bus write changes priority array value at configured object priority index



9.11. BACnet objects

In *Network* → *BACnet objects* you can see marked objects on LogicMachine which are sent to BACnet network.

BACnet objects - x

Device name: LogicMachine_222 Download CSV

Device ID: 222

Object priority: 16

Port: 47808

Type	Instance	Device name	Current value
2 (AV)	6500	PassivPlus 1 (3.1.100)	29
2 (AV)	6501	PassivPlus 2 (3.1.101)	29

9.12. HTTP server

In case additional www ports are needed to run the web-server on, use *Network* → *HTTP server* menu. Default HTTP port is 80, default HTTPS port is 443.

The screenshot shows a dialog box titled "HTTP server" with a close button (X) in the top right corner. It contains two input fields: "Additional HTTP port" with the value "8000" and "Additional HTTPS port" which is empty. Below the fields is an information icon (i) followed by the text "Default HTTP port: 80, default HTTPS port: 443". At the bottom of the dialog are two buttons: "OK" and "Cancel".

9.13. FTP server

You can enable access to FTP server of LogicMachine by enabling this service in *Service* → *FTP Server*.

The screenshot shows a dialog box titled "FTP server" with a close button (X) in the top right corner. It contains several fields: "Server status" is a dropdown menu set to "Enabled"; "Port" is a text box with "21"; "Username" is a text box with "ftp"; "Password" is an empty text box; "Username" is a text box with "apps"; "Password" is an empty text box; "External IP" is an empty text box; "Passive mode min port" is an empty text box; and "Passive mode max port" is an empty text box. Below the fields is an information icon (i) followed by the text: "Leave password blank to keep it unchanged. External IP and passive mode ports must be set when you want to access FTP behind NAT. Make sure both FTP port and passive mode port range are forwarded on your router." At the bottom of the dialog are two buttons: "OK" and "Cancel".

- **Server status** – define either FTP server is enabled or disabled
- **Port** – port of the service
- **Username** – login name, *ftp*
- **Password** – password for user *ftp*, length 4-20 symbols

- **Username** – login name to user directory <http://IP/user>, *apps* user. You can enable or disable password authorization for this directory in *Logic Machine* → *User access* → *User access settings*
- **Password** – password for user *apps*, length 4-20 symbols
- **Passive mode min port** – FTP passive mode minimum port
- **Passive mode max port** – FTP passive mode maximum port

9.14. Remote services

- **Service status** – define either remote services are enabled or disabled
- **Username** – user name
- **Password** – password

URL

Change the IP and password according to your LM settings

<http://remote:remote@192.168.0.10/scada-remote?m=rss&r=alerts>

Request parameters

m set the return value format

- **json**
- **xml**
- **rss** only for alerts and errors

r requested function name

- **alerts** newest 50 alerts

Return values:

- **alert** alert text
- **time** alert time (UNIX timestamp)

- `date` alert time (RFC date)

- `errors` newest 50 errors

Return values:

 - `error` error text
 - `script` error script name
 - `time` error time (UNIX timestamp)
 - `date` error time (RFC date)

- `objects` list of objects marked for export, ordered by update time

Return values:

 - `address` object address (e.g. "1/1/1")
 - `name` object name (e.g. "My object")
 - `data` decoded object value (e.g. 42 or "01.01.2012")
 - `datatype` object datatype (e.g. 1 or 5.001)
 - `time` object update time (UNIX timestamp)
 - `date` object update time (RFC date)
 - `comment` object comment (e.g. "Second floor entry lights")
 - `tags` optional array of object tags (e.g. "Light", "Second floor")

- `grp` execute one of grp functions

Parameters:

 - `fn` function name, *required*
 - `getvalue` returns current object value if found
 - `find` return object info
 - `write` send KNX bus group write telegram
 - `response` send KNX bus group response telegram
 - `read` send KNX bus group read telegram
 - `update` update local LM object value without KNX bus group write
 - `alias` group address or name, *required*

- `value` new value to write, *required* for write / response / update, except for time and date datatypes

Parameters for `time` datatype:

- `day` number (0-7), day of the week, *optional*
- `hour` number (0-23)
- `minute` number (0-59)
- `second` number (0-59)

Parameters for `date` datatype:

- `day` number (1-31)
- `month` number (1-12)
- `year` number (1990-2089)

- `datatype` *optional* for write / response / update, data type is taken from the **database** if not specified

Possible values:

bool	bit2	bit4	char	uint8	int8	uint16	int16	float16
time	date	uint32	int32	float32	access	string		

Examples

Write value of `50` to `1/1/1`

<http://remote:remote@192.168.0.10/scada-remote?m=json&r=grp&fn=write&alias=1/1/1&value=50>

Write boolean value to `1/1/2`, you can use `true` or `false`, as well as `1` or `0`

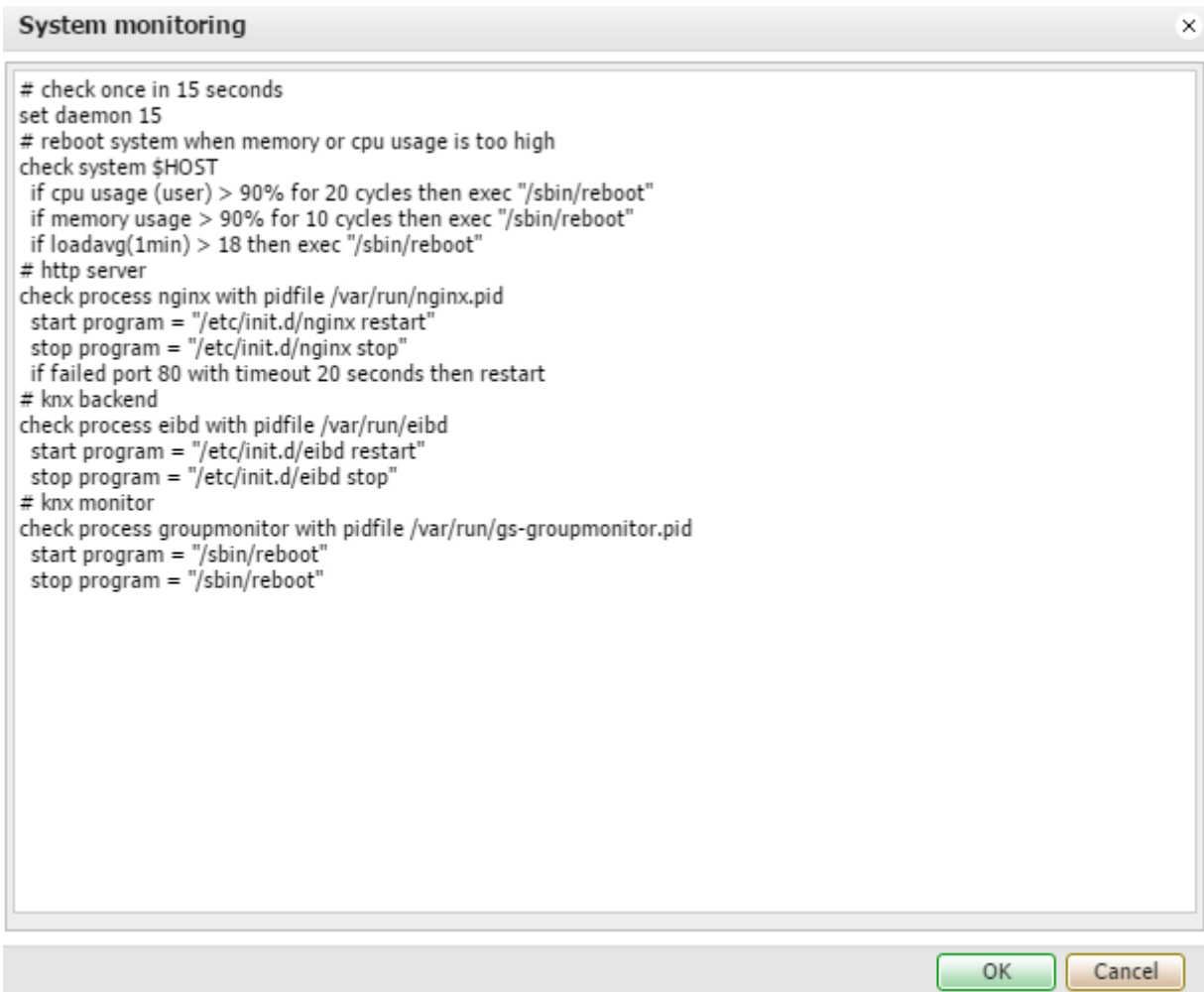
<http://remote:remote@192.168.0.10/scada-remote?m=json&r=grp&fn=write&alias=1/1/2&value=true>

Explicit datatype setting to `scale`, send `50` to `1/1/1`

<http://remote:remote@192.168.0.10/scada-remote?m=json&r=grp&fn=write&alias=1/1/1&value=50&datatype=scale>

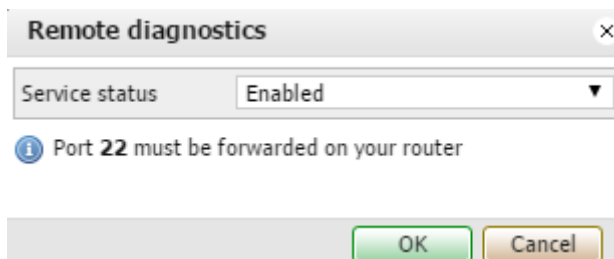
9.15. System monitoring

System monitoring is used to monitor system processes, hardware. In case of failure, the system will be rebooted or specific task restarted.



9.16. Remote diagnostics

Remote diagnostics should be enabled only when there is remote Embedded Systems support necessary for the device. It enables SSH access to the device.



- **Service status** – define either remote SSH access is enabled or disabled.

9.17. NTP client

NTP servers can be specified in *Service* → *NTP client* window.

NTP client (clock synchronization) ✕

Server 1	<input type="text" value="0.europe.pool.ntp.org"/>
Server 2	<input type="text" value="1.europe.pool.ntp.org"/>
Server 3	<input type="text" value="2.europe.pool.ntp.org"/>
Server 4	<input type="text" value="3.europe.pool.ntp.org"/>

9.18. System status

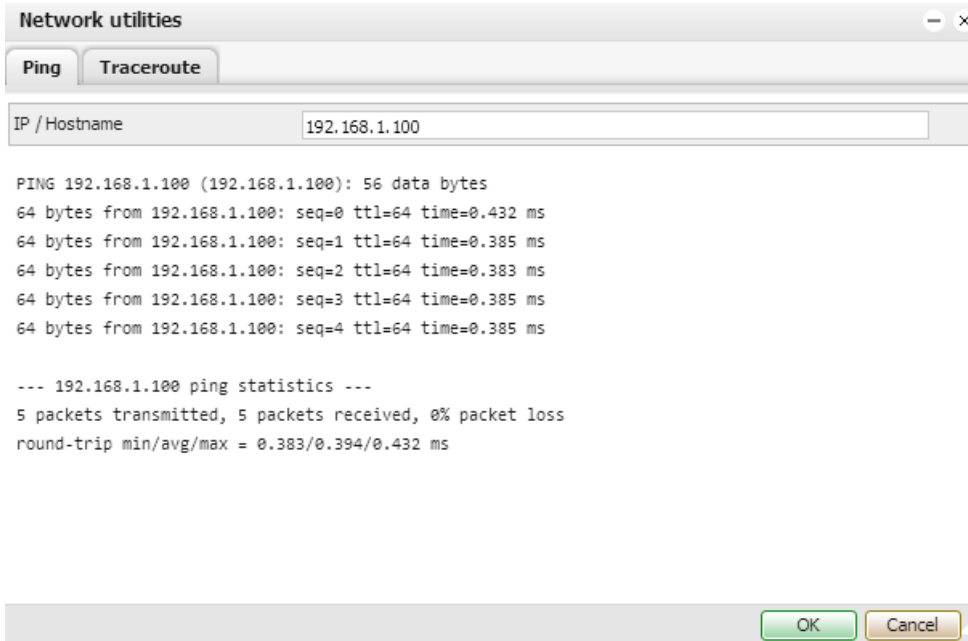
General system status with CPU usage, Memory usage, Partition, Serial ports information can be seen in *Status* → *System status* window.

System status — ✕

Parameter	Value
CPU model	ARM926EJ-S rev 5 (v5l)
CPU BogoMips	227.12
Linux kernel version	3.18.24
System uptime	2d 2h 0m
Load averages	0.01 0.03 0.05

9.19. Network utilities

Ping and *Traceroute* utilities are located in *Status* → *Network utilities* window. Both IP address and DNS names are accepted.



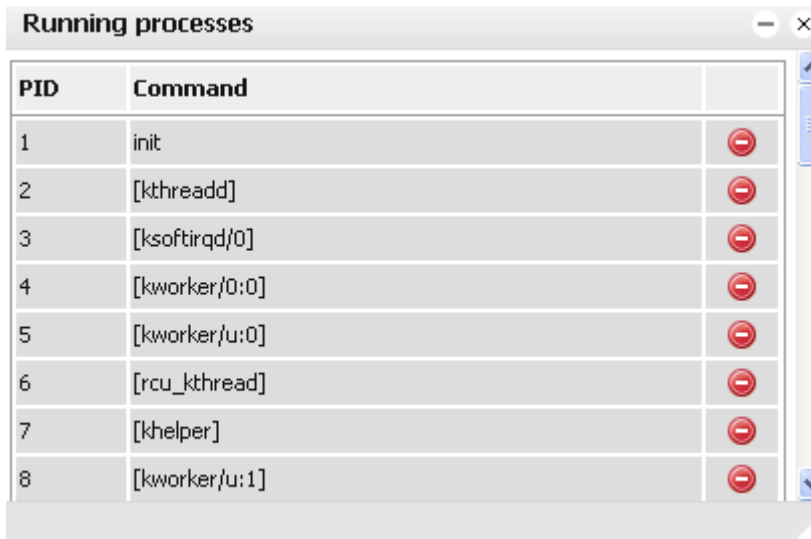
9.20. System log

Operating system log is available in *Status* → *System log*.



9.21. Running processes

System running processes can be seen in *Status* → *Running processes* window.



The screenshot shows a window titled "Running processes" with a table of system processes. The table has three columns: "PID", "Command", and a red minus sign icon. The processes listed are:

PID	Command	
1	init	⊖
2	[kthreadd]	⊖
3	[ksoftirqd/0]	⊖
4	[kworker/0:0]	⊖
5	[kworker/u:0]	⊖
6	[rcu_kthread]	⊖
7	[khelper]	⊖
8	[kworker/u:1]	⊖

10. User mode schedulers

User mode schedulers contains user-friendly interface for end-user to manage scheduler tasks, for example, specify thermostat values depending of the day of the week, time and holidays.

10.1. Events

Each scheduler is mapped to specific group address in administration panel (*see section 1.4 of this manual*).

The screenshot displays the configuration interface for an 'Outdoor lamp' scheduler. On the left, a sidebar lists 'Outdoor lamp', 'AC', and 'Holidays'. The main area shows the scheduler's status as 'active' with a period from '1 January - 31 December'. Below this is a table of events:

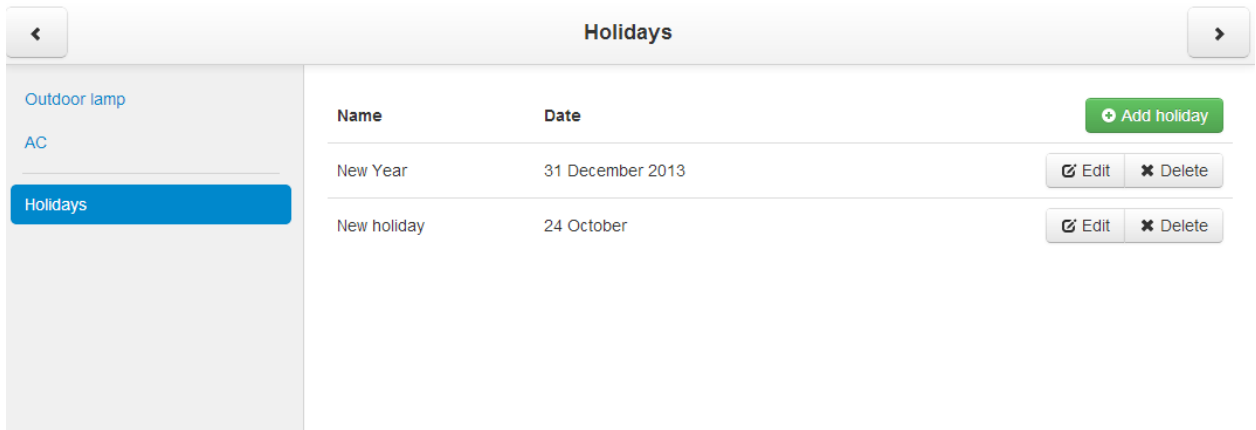
Value	Run at
Light off	12:00 Tu-Fr
Light off	13:00 Sa-Su Holiday

On the right, the 'Add event' panel is visible. It includes a checked 'Event is active' checkbox, a 'Run at' section with time selection (12:00) and day selection (All), and a 'Value' dropdown menu currently set to 'Light on'. 'Save' and 'Cancel' buttons are at the bottom.

When adding the new task for specific scheduler you can specify day of the week, start time, value to send to the object.

10.2. Holidays

In *Holidays* special days are specified which are then used adding new events.

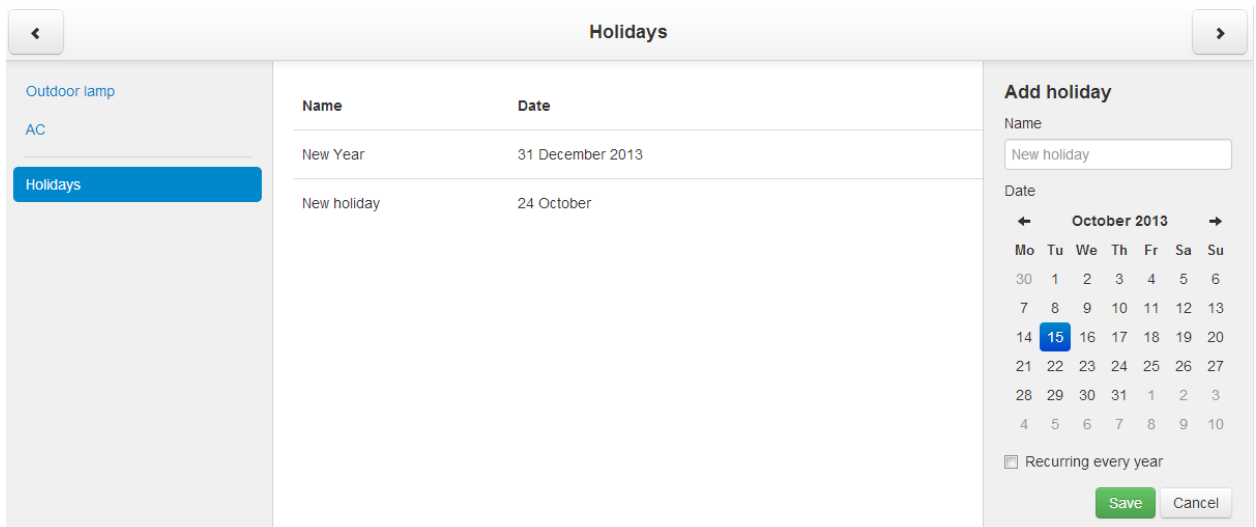


The screenshot shows a web interface titled "Holidays". On the left is a sidebar with a menu containing "Outdoor lamp", "AC", and "Holidays" (which is highlighted in blue). The main area contains a table with two columns: "Name" and "Date".

Name	Date	
New Year	31 December 2013	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
New holiday	24 October	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

At the top right of the table area is a green button labeled "Add holiday".

Click on *Add new holiday* button to specify a holiday.



This screenshot shows the same "Holidays" interface as above, but with the "Add holiday" modal open on the right side. The modal contains a "Name" field with the text "New holiday" and a "Date" field. The date field is a calendar for October 2013, with the 15th selected. Below the calendar is a checkbox labeled "Recurring every year" and two buttons: "Save" and "Cancel".

Name	Date
New Year	31 December 2013
New holiday	24 October

Add holiday

Name:

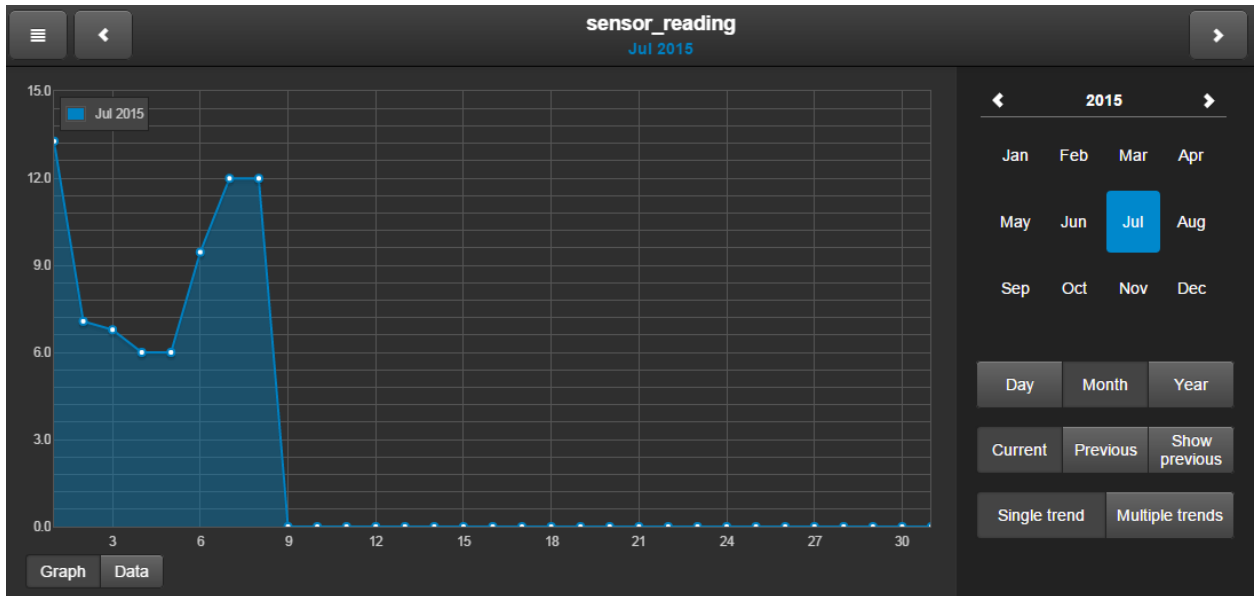
Date:


Mo	Tu	We	Th	Fr	Sa	Su
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

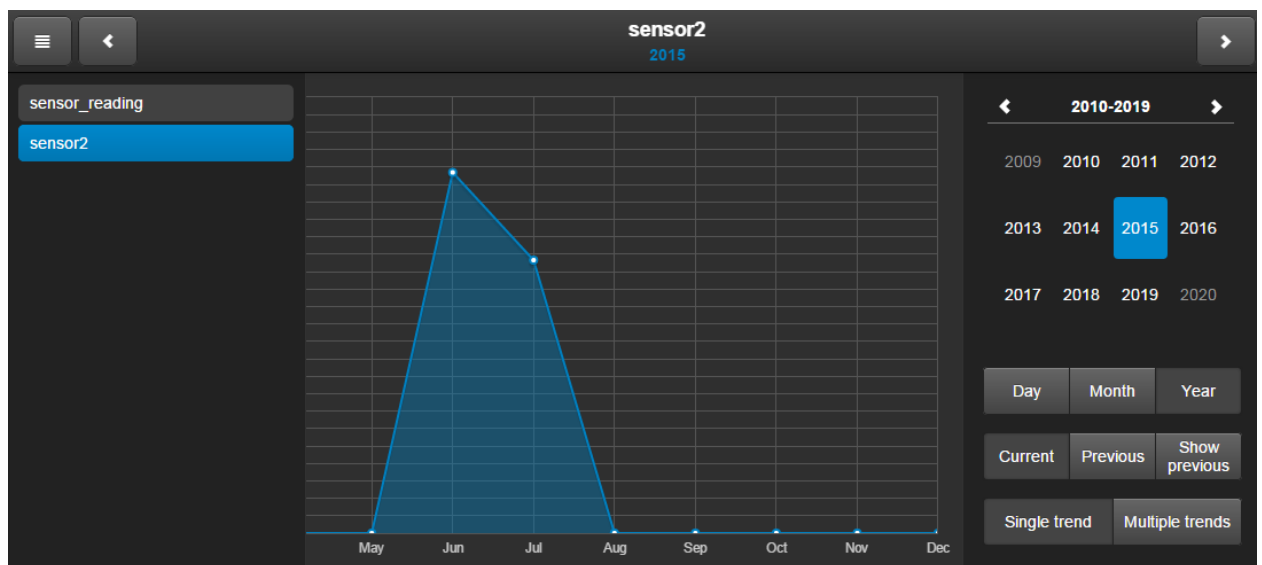
Recurring every year

11. Trend logs

Trend logs are end user interface for trends (*defined in administrator interface in section 1.5*).

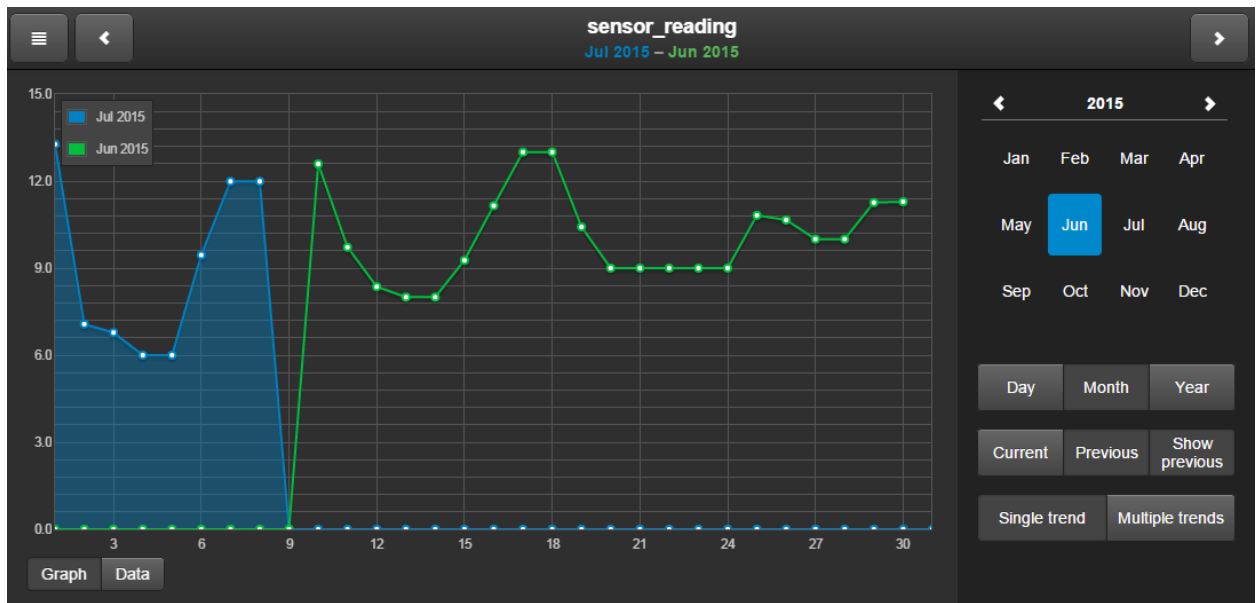


By clicking on the menu button  you can change to different trends where each is mapped to a specific KNX group address.

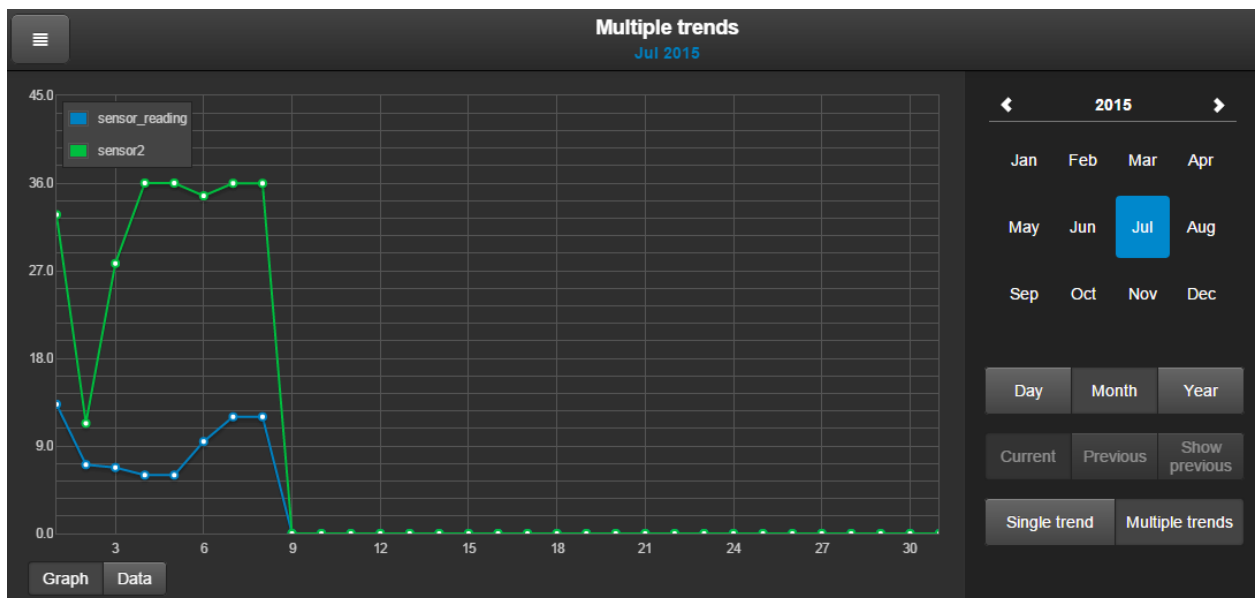


- **Day** – Trend with Day view
- **Month** – Trend with Month view
- **Year** – Trend with year view
- **Current** – Current trend is drawn in blue, you can choose either to show Day, Month or Year view
- **Previous** – previous time period, you can choose either to show Day, Month or Year view

- **Show previous** – when enabled a yellow trend line appears showing *Previous* trend above *Current* trend



- **Single trend** – view single trend
- **Multiple trends** – view multiple trends. When this mode is chosen, you can select several object on the left side to be shown



By clicking on *Data* button, data points will be shown in a way of table which can be later exported as CSV file.

☰
←
sensor_reading
Jul 2015 – Jun 2015
→

Download CSV

	Jul 2015	Jun 2015
1	13.27	0
2	7.07	0
3	6.78	0
4	6	0
5	6	0
6	9.45	0
7	11.99	0
8	11.99	0
9	0	0
10	0	12.59
11	0	9.72
12	0	8.36

Graph
Data

← **2015** →

Jan

Feb

Mar

Apr

May

Jun

Jul

Aug

Sep

Oct

Nov

Dec

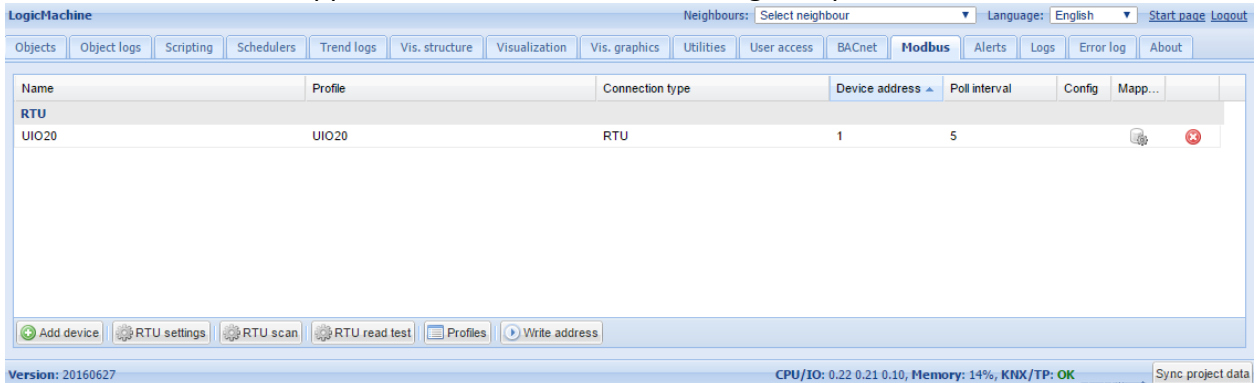
Day
Month
Year

Current
Previous
Show previous

Single trend
Multiple trends

12. Modbus RTU/TCP interconnection with LM

Modbus TCP support is added by installing a special package through *Sys config* → *System* → *Packages*. Modbus TCP is supported over Ethernet port. Modbus communication is done either from visual Modbus mapper for Modbus Master or through scripts for Modbus Slave.



Modbus Master – user graphical mapper interface in Modbus tab

Modbus Slave – to use LM as Modbus Slave, disable Modbus RTU in Modbus → RTU settings, and use scripts for the communication

12.1. Modbus device profile

First thing you should do is to define Modbus device profile – it is a *.json file with the following structure e.g. a fragment from UIO20 device by Embedded Systems:

```
{
  "manufacturer": "Embedded Systems",
  "description": "Universal 16+4 I/O module",
  "mapping": [
    { "name": "Output 1", "bus_datatype": "bool", "type": "coil", "address": 0, "writable": 1 },
    { "name": "Input 1", "bus_datatype": "float16", "type": "inputregister", "address": 0, "value_multiplier": 0.001, "units": "V" }
  ]
}
```

Name – Object name, e.g. Output 2 (String, Required)

Bus_datatype - KNX object data type, key from *dt* table, e.g. float32 (String/Number, Required)

Type – Modbus register type, possible values: coil discreteinput register inputregister (String, Required)

Address – Register address (0-based) (Number, Required)

Writable - Set to true to enable writing to register if type is either coil or register (Boolean)

Datatype – Modbus value data type. If set, conversion will be done automatically.

Possible values: uint16 int16 float16 uint32 int32 float32 uint64 int64 quad10k s10k (String)

Value_delta – New value is sent when the difference between previously sent value and current value is larger than delta. Defaults to 0 (send after each read) (Number)

Value_multiplier – Multiply resulting value by the specified number, $value = value_base + value * value_multiplier$ (Number)

Value_bitmask – Bit mask to apply, shifting is done automatically based on least significant 1 found in the mask (Number)

Value_nan – Array of 16-bit integers. If specified and read operation returns the same array no further processing of value is done (Array)

Value_conv – Apply one of built-in conversion functions (String, Internal)

Value_custom – Name of a built-in enumeration or a list of key -> value mapping, resulting value will be 0 if key is not found (String/Object)

Internal – Not visible to user when set to true, should be used for scale registers (Boolean)

Units – KNX object units/suffix (String)

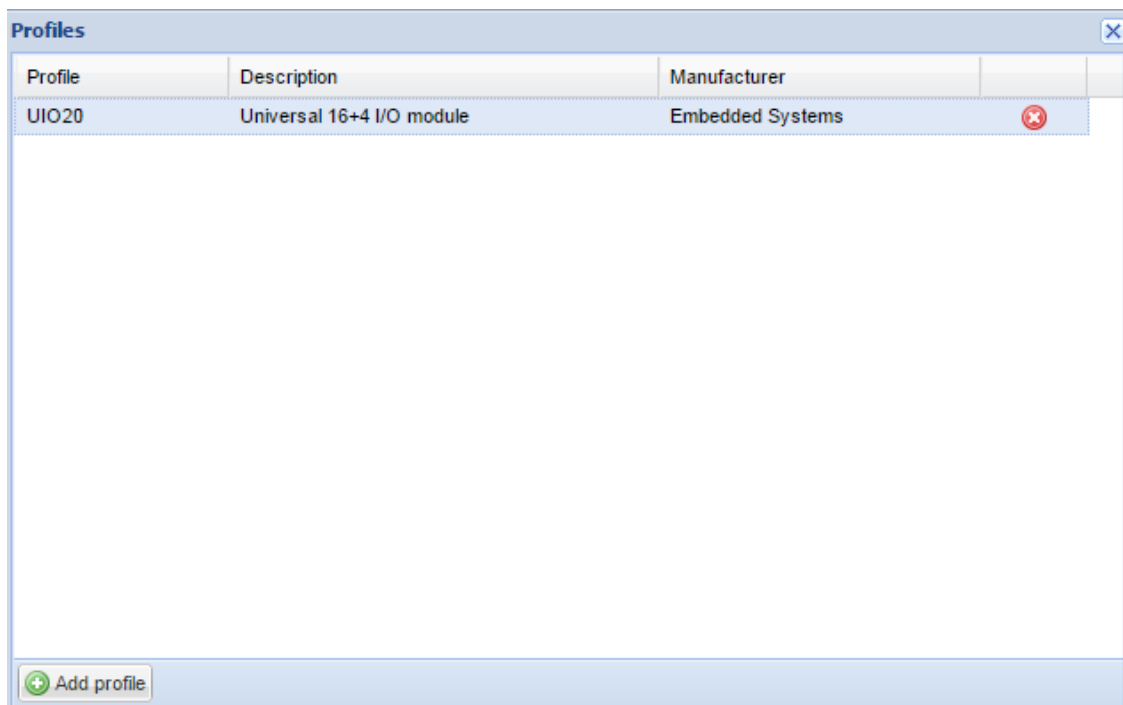
Address_scale – Address of register containing value scale, $value = value * 10 ^ scale$ (Number)

Read_count – Number of register to read at once (for devices that only support reading of a specific block of registers) (Number)

Read_swap – Swap register order during conversion (endianness) (Boolean)

Read_offset – Position of first register of data from the block of registers (0-based) (Number)

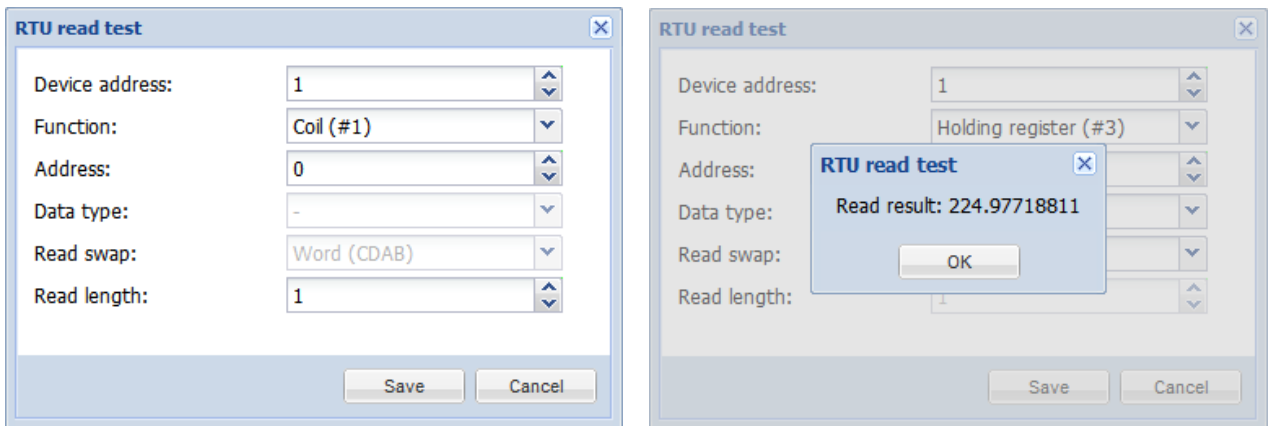
When the Modbus device profile file is created, upload it by clicking on *Profiles* button.



12.2. Reading ModBus RTU coil / register from the interface

As creating new Modbus profiles is not the most user-friendly task, we have added a new feature that allows reading any Modbus coil or register straight from the user interface. This should help users to find correct settings and addresses before creating new profiles. For now it only works with RTU connection, TCP is planned to be implemented later.

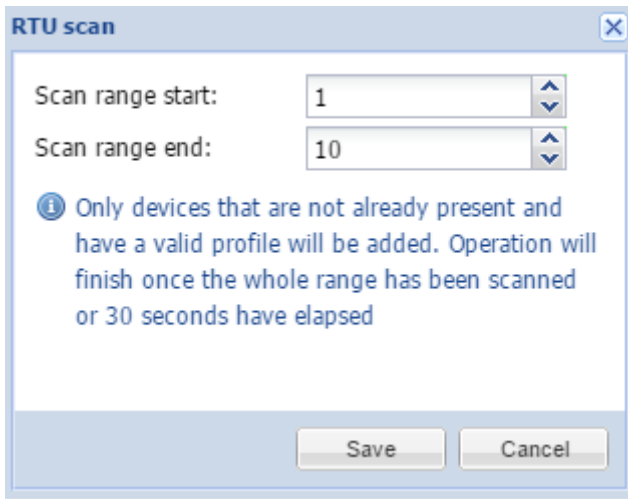
By pressing RTU read test button you get the following options:



- **Device address** – ModBus device address
- **Function (Coil, Discrete input, Holding register, Input register)** – ModBus function
- **Address** – address where data is located
- **Data type** – data type, can be chosen only for registers
- **Read swap (None (ABCD); Word (CDAB); Byte (BADDC); Byte and word (DCBA))** – read data swapped in chosen way
- **Read length** – read length of registers/coils

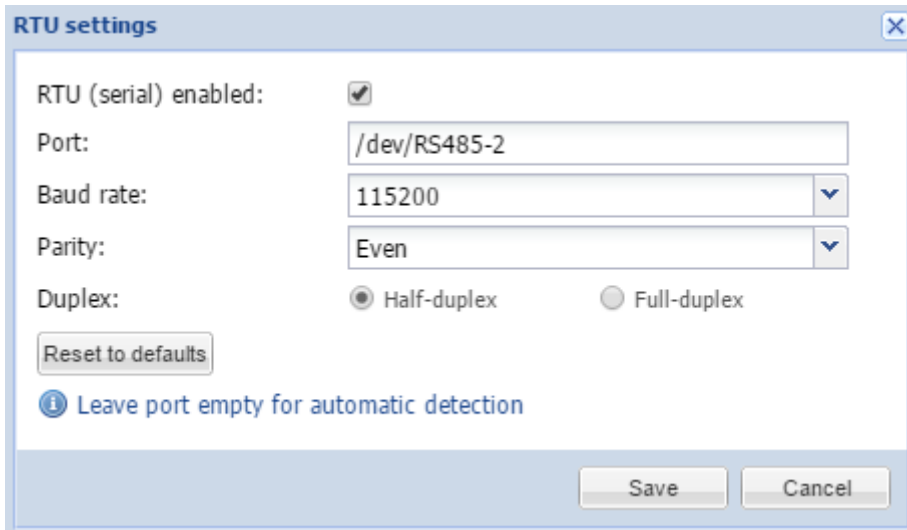
12.3. RTU Scan

Scan RS-485 ports for connected ModBus RTU devices. Only devices that are not already present and have a valid profile will be added. Operation will finish once the whole range has been scanned or 30 seconds have elapsed



- **Scan range start** – start ModBus device address
- **Scan range end** – end ModBus device address

12.4. RTU settings



- **RTU (serial) enabled** – defines either Modbus RTU is enabled
- **Port (/dev/RS485-1; /dev/RS485-2)** – specify the port to communicate or leave blank for automatic detection.
- **Baud rate (1200 .. 500000)** – baud rate
- **Parity (None 1 stop bit; Odd, Even, None 2 stop bits)** – parity
- **Duplex** – define either half or full duplex communication


12.5. Adding Modbus device

Once profiles are added, add Modbus device by clicking *Add device* button.

The 'Modbus device' dialog box contains the following fields and options:

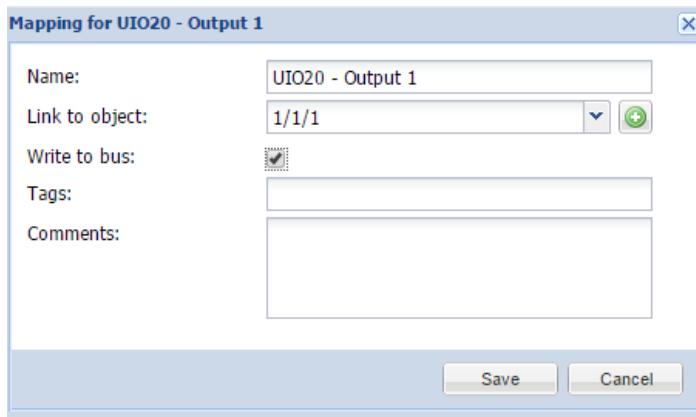
- Connection type:** Radio buttons for RTU (RS-485) and TCP/IP (selected).
- Name:** Text input field containing 'Modbus device name'.
- Profile:** Dropdown menu showing 'UIO20'.
- IP:** Text input field containing '192.168.1.34'.
- Port:** Spin box set to '502'.
- Device address:** Spin box set to '1'.
- Poll interval:** Spin box set to '5'.
- Buttons:** 'Save' and 'Cancel' buttons at the bottom right.

- **Connection type** – define either it is Modbus RTU or Modbus TCP connection
- **Name** – name of the device
- **Profile** – profile of the device
- **Device address** – device address
- **Poll interval (seconds)** – interval to poll the device
- **IP** – IP address of the device in case Modbus TCP is used
- **Port** – Communication port of the device in case Modbus TCP is used

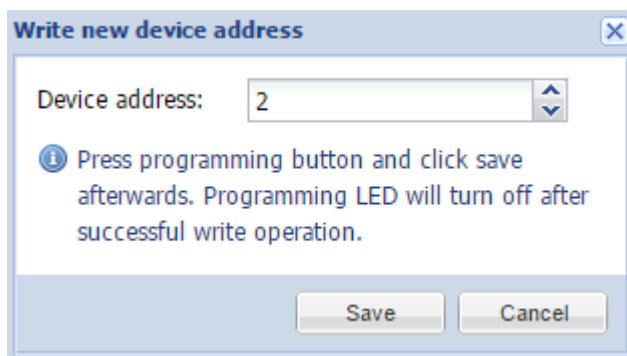
Once the device is added, you can do mapping to KNX addresses by clicking on  icon. First, you see a list of all objects on the Modbus device.

Name	Linked to object	Current value	Type
UIO20 - Output 1			Coil: 0
UIO20 - Output 2			Coil: 1
UIO20 - Output 3			Coil: 2
UIO20 - Output 4			Coil: 3
UIO20 - Output 5			Coil: 4
UIO20 - Output 6			Coil: 5
UIO20 - Output 7			Coil: 6
UIO20 - Output 8			Coil: 7
UIO20 - Output 9			Coil: 8
UIO20 - Output 10			Coil: 9
UIO20 - Output 11			Coil: 10
UIO20 - Output 12			Coil: 11
UIO20 - Output 13			Coil: 12
UIO20 - Output 14			Coil: 13
UIO20 - Output 15			Coil: 14
UIO20 - Output 16			Coil: 15
UIO20 - Input 1			Input register: 0
UIO20 - Input 2			Input register: 1
UIO20 - Input 3			Input register: 2
UIO20 - Input 4			Input register: 3
UIO20 - Input 5			Input register: 4
UIO20 - Input 6			Input register: 5
UIO20 - Input 7			Input register: 6
UIO20 - Input 8			Input register: 7
UIO20 - Input 9			Input register: 8
UIO20 - Input 10			Input register: 9
UIO20 - Input 11			Input register: 10
UIO20 - Input 12			Input register: 11
UIO20 - Input 13			Input register: 12
UIO20 - Input 14			Input register: 13
UIO20 - Input 15			Input register: 14

Click on specific object to do mapping.



12.6. Program address for UIO20 Modbus device



There is a separate Write address button to program address for UIO20 device. Press programming button and click save afterwards. Programming LED will turn off after successful write operation.

Once script is added, you can add the code in the Script Editor. There are lots of predefined code blocks in the Helpers.

12.7. Modbus Slave examples

Add the following code to *Common functions*

```

1. -- modbus proxy
2. mbproxy = {
3. -- supported function list
4.   functions = {
5.     'readdo',
6.     'readcoils',
7.     'readdi',
8.     'readdiscreteinputs',
9.     'readao',
10.    'readregisters',
11.    'readai',
12.    'readinputregisters',

```



```

13. 'writebits',
14. 'writemultiplebits',
15. 'writeregisters',
16. 'writemultipleregisters',
17. 'reportslaveid',
18. 'getcoils',
19. 'getdiscreteinputs',
20. 'getinputregisters',
21. 'getregisters',
22. 'setcoils',
23. 'setdiscreteinputs',
24. 'setinputregisters',
25. 'setregisters',
26. },
27. -- new connecton init
28.   new =function()
29.   require('rpc')
30.   local mb =setmetatable({}, { __index = mbproxy })
31.
32.     mb.slaveid =0
33.     mb.rpc = rpc.client('127.0.0.1', 28002, 'mbproxy')
34.
35.   for _, fn in ipairs(mbproxy.functions)do
36.     mb[ fn ]=function(self, ...)
37.   return mb:request(fn, ...)
38.   end
39.   end
40.
41.   return mb
42.   end
43. }
44.
45. -- set local slave id
46. function mbproxy:setslave(slaveid)
47.   self.slaveid = slaveid
48. end
49.
50. -- send rpc request for a spefic function
51. function mbproxy:request(fn, ...)
52. local res, err = self.rpc:request({
53.   fn = fn,
54.   params ={ ... },
55.   slaveid = self.slaveid or0,
56. })
57.
58. -- request error
59. if err then
60. return nil, err

```

```

61. -- request ok
62. else
63. -- reply with an error
64. if res[ 1 ]==nil then
65. return nil, res[2]
66. -- normal reply
67. else
68. return unpack(res)
69. end
70. end
71. end

```

Handler (resident script with 0 delay) configuration

1. *mb:open()*

Open Modbus TCP connection

2. *mb:setslave(10)*

set slave device id

3. *mb:setmapping(10, 10, 10, 10)*

set number coils, discrete inputs, holding registers and input registers

4. *mb:setwritecoilcb(function(coil, value)...*

callback function which is executed for each coil write

5. *mb:setwriteregistercb(function(coil, value)...*

callback function which is executed for each register write

Handler script example

```

1.  -- modbus init
2.  if not mb then
3.  require('luamodbus')
4.  mb = luamodbus.tcp()
5.  mb:open()
6.
7.  -- init slave storage for coils, discrete inputs, holding registers and input registers
8.  mb:setmapping(10, 10, 10, 10)
9.
10. -- coil write callback
11. mb:setwritecoilcb(function(coil, value)
12. if coil == 0 then
13.     grp.write('1/1/1', value, dt.bool)
14. else
15.     alert('coil: %d = %s', coil, tostring(value))

```

```

16. end
17. end)
18.
19. -- register write callback
20.   mb:setwriteregistercb(function(register, value)
21. if register == 0 then
22. -- send value limited to 0..100
23.     grp.write('4/1/5', math.min(100, value), dt.scale)
24. else
25.     alert('register: %d = %d', register, value)
26. end
27. end
28. end
29.
30. -- server part init
31. if not server then
32. require('rpc')
33.
34. -- incoming data handler
35. local handler = function(request)
36. local fn, res
37.
38.     fn =tostring(request.fn)
39.
40. if not mb[ fn ]then
41. return{nil, 'unknown function ' .. fn }
42. end
43.
44. if type(request.params)=='table' then
45. table.insert(request.params, 1, mb)
46.     res ={ mb[ fn ](unpack(request.params))}
47. else
48.     res ={ mb[ fn ](mb)}
49. end
50.
51. return res
52. end
53.
54. server = rpc.server('127.0.0.1', 28002, 'mbproxy', handler, 0.01)
55. end
56.
57. mb:handleslave()
58. server:step()

```

Example: event script which changes modbus slave coil (address 0)

Must be mapped to a group address with binary value.

```

1. value = event.getvalue()
2. mb = mbproxy.new()
3. mb:setcoils(0, value)

```

Example: event script which changes modbus slave register (address 5)

Must be mapped to a group address with scaling (0..100) value

```

1. value = event.getvalue()
2. mb = mbproxy.new()
3. mb:setregisters(5, value)

```

LM interconnection with PLC over Modbus TCP

sleep time = 0. It only supports binary objects as coils and 1-byte / 2-byte integer objects as registers. Number of coils and registers is not limited, object mapping can be set by filling coils, registers and regdt tables.

```

1. if not mb then
2.   require('genohm-scada.eibdgm')
3.   require('luamodbus')
4.
5.   -- list of coil mapping, starting from 0
6.   coils = { '1/1/1', '1/1/2' }
7.
8.   -- list of register mapping, starting from 0
9.   registers = { '2/2/2', '3/3/3' }
10.
11.  -- list of register data types, element count must match registers table
12.  regdt = { dt.int8, dt.uint16 }
13.
14.  -- knx group write callback
15.  function knxgroupwrite(event)
16.    local value
17.
18.    -- try to find matching coil
19.    for id, addr in ipairs(coils) do
20.      if event.dst == addr then
21.        value = knxdatatype.decode(event.datahex, dt.bool)
22.        mb:setcoils(id - 1, value)
23.      end
24.    end
25.
26.    -- try to find matching register
27.    for id, addr in ipairs(registers) do
28.      if event.dst == addr then
29.        value = knxdatatype.decode(event.datahex, regdt[ id ])

```

```

30.     mb:setregisters(id - 1, value)
31.     end
32. end
33. end
34.
35. -- coil write callback
36. function mbwritecoils(coil, value)
37.     local addr = coils[ coil + 1 ]
38.     if addr then
39.         grp.write(addr, value, dt.bool)
40.     end
41. end
42.
43. -- register write callback
44. function mbwriteregisters(register, value)
45.     local addr = registers[ register + 1 ]
46.     if addr then
47.         grp.write(addr, value, regdt[ register + 1])
48.     end
49. end
50.
51. -- knx group monitor, handles group writes
52. knxclient = eibdgm:new({ timeout = 0.1 })
53. knxclient:sethandler('groupwrite', knxgroupwrite)
54.
55. -- modbus slave, listen on all interfaces and default port 502
56. mb = luamodbus.tcp()
57. mb:open('0.0.0.0', 502)
58.
59. -- setting slave id is optional
60. -- mb:setslave(1)
61.
62. mb:setreceivetimeout(0.1)
63. mb:setmapping(#coils, 0, #registers, 0)
64.
65. -- init coils
66. for id, addr in ipairs(coils) do
67.     value = grp.getvalue(addr)
68.     mb:setcoils(id - 1, value)
69. end
70.
71. -- init registers
72. for id, addr in ipairs(registers) do
73.     value = grp.getvalue(addr)
74.     mb:setregisters(id - 1, value)
75. end
76.
77. -- set callbacks for coil and register write
78. mb:setwritecoilcb(mbwritecoils)
79. mb:setwriteregistercb(mbwriteregisters)

```

```
80. end
81.
82. -- handle modbus and knx
83. mb:handleslave()
84. knxclient:step()
```

13. BACnet IP interconnection with LM

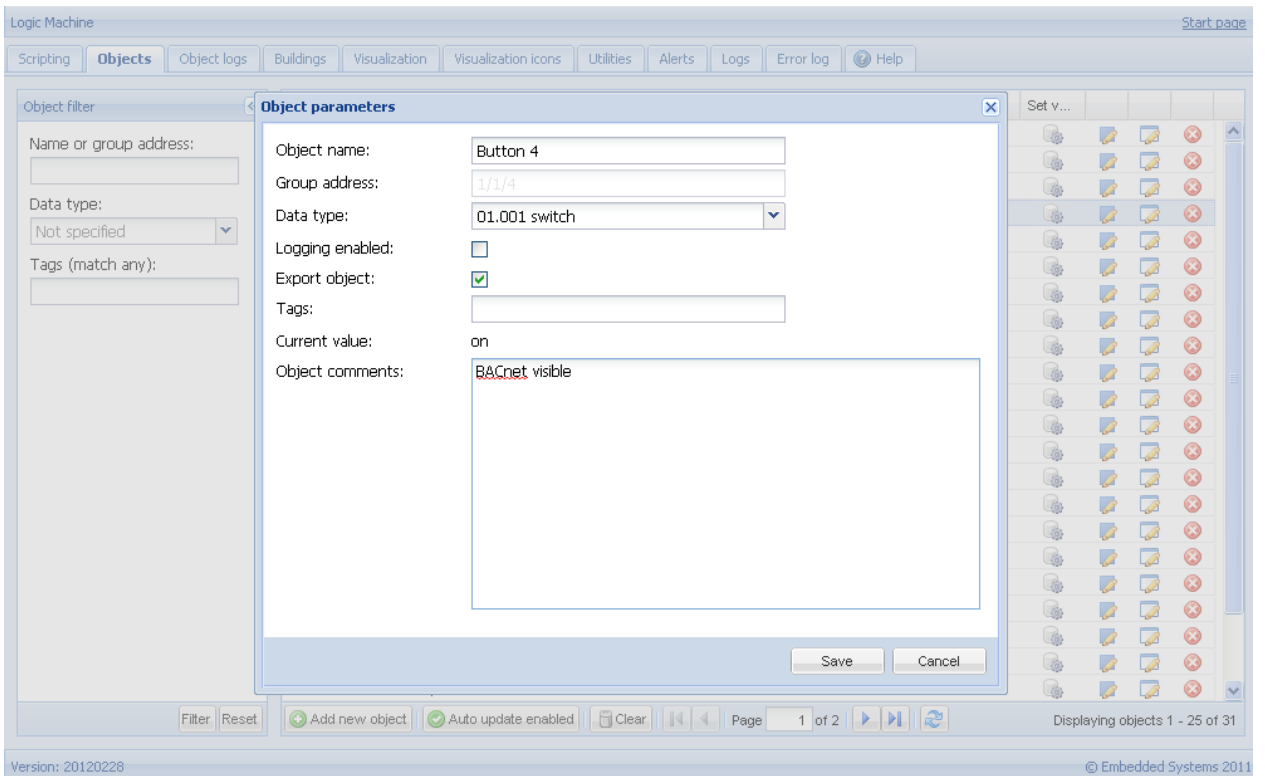
13.1. BACnet server mode: transparent data transfer to BACnet network

BACnet server specific configuration can be done in *Sys Config* → *Network* → *BACnet Settings*.

Field	Value
Server enabled	<input checked="" type="checkbox"/>
Device ID	222
Password	mybacpwd
Object priority	16
Port	47808
BBMD IP	
BBMD port	
BBMD lease time (seconds)	

- **Server enabled** – specify if BACnet server is enabled or not
- **Device ID** – device ID in BACnet network
- **Password** – device password
- **Object priority** – object priority
- **Port** – port number
- **BBMD IP** – BACnet router IP. When router IP and port are set, LM will act as a foreign device and will attempt to register with BACnet router.
- **BBMD port** – BACnet router port. When router IP and port are set, LM will act as a foreign device and will attempt to register with BACnet router
- **BBMD lease time (seconds)** – registration resend interval

To make KNX/EIB objects BACnet readable/writable, mark necessary objects in LogicMachine as “Export object”. Binary objects will appear as Binary Values, other numeric values will appear as Analog Values. Other types are not currently supported. KNX bus write changes priority array value at configured object priority index



In *System Configuration* → *Network* → *BACnet objects* you can see marked objects on LogicMachine which are sent to BACnet network.

BACnet objects - x

Device name: LogicMachine_222 Download CSV

Device ID: 222

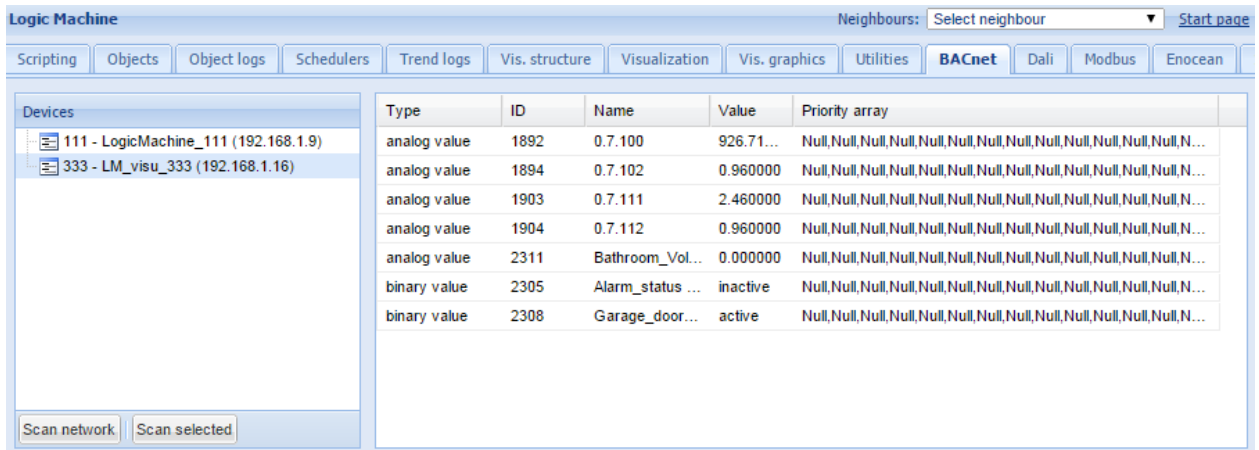
Object priority: 16

Port: 47808

Type	Instance	Device name	Current value
2 (AV)	6500	PassivPlus 1 (3.1.100)	29
2 (AV)	6501	PassivPlus 2 (3.1.101)	29

13.2. BACnet client mode

Normally this mode is used to interconnect LogicMachine, for example, with VRV systems over BACnet IP protocol. The settings are available in BACnet tab.



By clicking on *Scan Network* button you can see a list of BACnet server devices on the network. With *Scan Selected* you can rescan specific BACnet server for respective objects.

Mapping to KNX objects currently is done over scripting.

Before using any BACnet function, you must include the library:

```
require('bacnet')
```

Read current value of binary or analog object:

```
bacnet.readvalue(device_id, object_type, object_id)
```

Read binary object:

```
value = bacnet.readvalue(127001, 'binary value', 2305)
```

Read analog object:

```
value = bacnet.readvalue(127001, 'analog value', 2306)
```

Write new value to binary or analog object priority array:

```
bacnet.write = function(device_id, object_type, object_id, value, priority)
```

Value can be nil, boolean, number or a numeric string

Priority parameter is optional, lowest priority is used by default

Set binary object value:

```
bacnet.write(127001, 'binary value', 2305, true)
```

Set analog object value:

```
bacnet.write(127001, 'analog value', 2306, 22.5)
```

Set binary object value at priority 12:

```
bacnet.write(127001, 'binary value', 2305, true, 12)
```

Set analog object value at priority 10:

```
bacnet.write(127001, 'analog value', 2306, 22.5, 10)
```

Clear binary object value at priority 12:

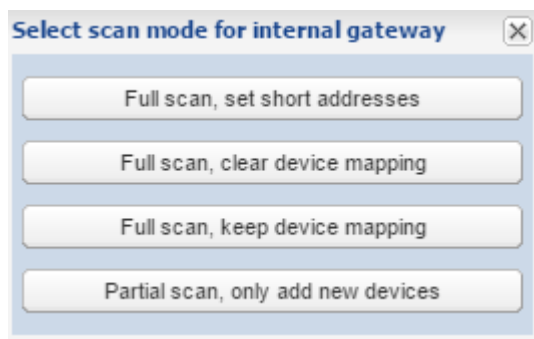
```
bacnet.write(127001, 'binary value', 2305, nil, 12)
```

14. DALI configuration

LogicMachine5 Power (LM5p-DW1) has two DALI masters built in, each supports up to 64 DALI ballasts. We recommend to connect no more than 32 ballasts to one DALI line. If more ballasts are necessary to connect, you can use more DALI-RS-485 interfaces and connect to RS-485 port.

Short address	Name	Binary object	Preset	Scale object	Set value
0	DEV-0	-	254	-	
1	DEV-1	-	254	-	
2	DEV-2	-	254	-	
3	DEV-3	-	254	-	
4	DEV-4	-	254	-	
5	DEV-5	-	254	-	
6	DEV-6	-	254	-	
7	DEV-7	-	254	-	
8	DEV-8	-	254	-	
9	DEV-9	-	254	-	
10	DEV-10	-	254	-	
11	DEV-11	-	254	-	
12	DEV-12	-	254	-	
13	DEV-13	-	254	-	
14	DEV-14	-	254	-	
15	DEV-15	-	254	-	

- **Scan gateways** - scans for currently connected gateways, address mapping for missing devices is deleted automatically
- **Write ID** - allows setting a unique address for each gateway
- **Scan devices** - scans for currently connected DALI devices to the selected gateway. There is one of 4 options to choose from.



Full scan, set short addresses - scans for currently connected DALI devices to the selected gateway, assigns short address automatically starting from 0,1,2,...

Full scan, clear device mapping - scans for currently connected DALI devices to the selected gateway without assigning short addresses, clear KNX grp address mapping to devices

Full scan, keep device mapping - scans for currently connected DALI devices to the selected gateway without assigning short addresses, keeps device mapping to KNX grp

addresses

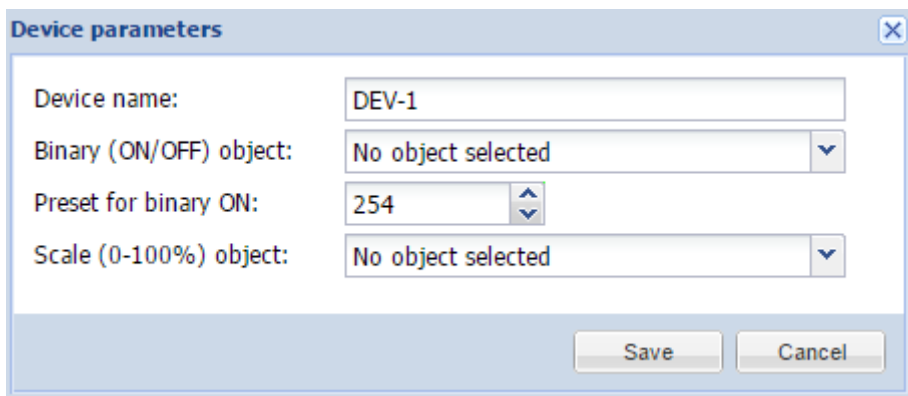
Partial scan, only add new devices - scans for newly added DALI devices to the selected gateway without assigning short addresses. Missing devices are not removed from the list

- **Port settings** – serial port name if there are external DALI-RS-485 interfaces connected

For each DALI device, you can set a custom name and map to binary on/off and scale object. This allows communication with DALI devices from KNX bus and visualization without any additional scripts.

14.1. DALI object mapping

Once DALI objects are scanned, you can click on corresponding object and perform the configuration.



- **Device name** – name of the DALI device
- **Binary (ON/OFF) object** – map to KNX binary object
- **Preset for binary ON** – preset on binary ON
- **Scale (0-100%) object** – map to KNX scale object

You can set up specific value by clicking on this icon 

14.2. Access DALI bus from scripts

If you want to access DALI devices from other scripts, you can use **dalicmd** function.

```
res, err = dalicmd(gwid, cmd, params)
```

Parameters

- **gwid** (*number/string*) gateway id: gateway number or **internal** when internal DALI exists
- **cmd** (*string*) command to send, refer to command table for possible values
- **params** (*table*) command parameters

Params (Lua table):

- `addrtype` (*string*) address type, only required for addressable commands, possible values: `short` `group` `broadcast`
- `address` (*number*) short or group address
- `value` (*number*) additional value to send

3 addressing modes are supported

- `broadcast` all slaves should react: `{ addrtype = 'broadcast' }`
- `short` only one slave having a unique short address should react: `{ addrtype = 'short', address = SLAVE_ID }`
- `group` several slaves belonging to a group should react: `{ addrtype = 'group', address = GROUP_ID }`

Command types

If command is addressable, it's possible to provide address type and address in `params` table.

If command expects a reply it must be addressed so only one slave can reply, otherwise a collision will happen. In case of success, reply is a binary string, usually consisting of a single byte. You can convert it to number like this:

```
-- query status of slave with short address 5 on the internal DALI bus
res, err = dalicmd('internal', 'querystatus', { addrtype = 'short', address = 5 })
-- read ok

if res then
    status = res:byte()
end
```

If command has a value range, `params` table must have a value field which is an integer in the specified range. For example, `arc` command accepts a value from 0 to 254:

```
-- set level to 42 for all slave on the internal DALI bus
dalicmd('internal', 'arc', { addrtype = 'broadcast', value = 42 })
```

Setting DTR

For commands where DTR is needed prior to executing command, use `setdtr` command to set the value:

```
-- set dtr for ballast 5 to 200
dalicmd('internal', 'setdtr', { addrtype = 'short', address = 5, value = 200 })
```

Example (use gateway with id 1, switch all ballasts off, set ballast with short address 5 to full on)

```
require('user.dali')

dalicmd(1, 'arc', { addrtype = 'broadcast', value = 0 })
dalicmd(1, 'arc', { addrtype = 'short', address = 5, value = 254 })
```

Example (set maximum value for ballast 5 to value 200; the ballast is connected on internal DALI gateway on LogicMachine)

```
require('user.dali')

dalicmd('internal', 'setdtr', { addrtype = 'short', address = 5, value = 200 })
dalicmd('internal', 'storemax', { addrtype = 'short', address = 5 })
```

Example (log all ballast short addresses which are connected to internal DALI gateway)

```
require('user.dali')
res, err = dalicmd('internal', 'queryshortaddr', { addrtype = 'broadcast' })
if res then
    log(res:byte())
else
    log(err)
end
```

Example (add 4 DALI short addressed to one group with nr. 7)

```
require('user.dali')
dalicmd('internal', 'addtogroup', { addrtype = 'short', address = 1, value = 7 })
dalicmd('internal', 'addtogroup', { addrtype = 'short', address = 2, value = 7 })
dalicmd('internal', 'addtogroup', { addrtype = 'short', address = 3, value = 7 })
dalicmd('internal', 'addtogroup', { addrtype = 'short', address = 4, value = 7 })
```

Setting group 7 to a certain value:

```
require('user.dali')
value = event.getvalue()
value = math.floor(value * 2.54)
dalicmd('internal', 'arc', { addrtype = 'group', address = 7, value = value })
```

Example (functions for calling and saving scenes, can be used not only for DALI)

First, you have to define each scene via a Lua table where each item is a table with two items: group address and value. Each scene has a unique id which can be a number or a string.

callscene(id) sets all objects in given scene to their specified value. First it looks for a saved scene in storage and uses default values if storage is empty.

savescene(id) gets current values of all objects from given scene and saves the whole scene in storage.

```
scenes = {}
scenes[1] = {
  { '15/1/1', 50 },
  { '15/1/2', 75 },
  { '15/1/3', 90 },
}
function callscene(id)
  local key, scene
  key = 'scene_' .. id
  scene = storage.get(key, scenes[ id ])
  if type(scene) ~= 'table' then
    alert('Scene ' .. id .. ' not found')
    return
  end
  for _, item in ipairs(scene) do
    grp.write(item[ 1 ], item[ 2 ])
  end
end
```

```

function savescene(id)
    local key, scene
    scene = scenes[ id ]
    if type(scene) ~= 'table' then
        alert('Scene ' .. id .. ' not found')
        return
    end
    for i, item in ipairs(scene) do
        scene[ i ][ 2 ] = grp.getvalue(item[ 1 ])
    end
    key = 'scene_' .. id
    storage.set(key, scene)
end

```

Example (Binary dimmer for DALI lamps to be able dim DALI lamp from physical pushbutton)

1) Add *bindimmer* function to Common functions

```

function bindimmer(up, down, out, event)
    local main, rev, step, val, new, delay
    step = 10 -- in %
    delay = 0.5 -- in seconds
    -- ignore "stop" command
    val = tonumber(event.datahex, 16)
    if val == 0 then
        return
    end
    -- up, normal mode
    if event.dst == up then
        main, rev = up, down
    -- down, reverse step
    elseif event.dst == down then
        main, rev = down, up
        step = -step
    end
end

```



```

-- invalid object
else
    return
end
-- current output object value
val = grp.getvalue(out) or 0
while true do
    -- main object in "stop" state
    if not grp.getvalue(main) then
        return
    end
    -- reverse object in "start" state
    if grp.getvalue(rev) then
        return
    end
    -- get new value
    new = math.min(100, val + step)
    new = math.max(0, new)
    -- no change, stop
    if new == val then
        return
    end
    -- write new value
    val = new
    grp.write(out, new, dt.scale)
    -- wait for next run
    os.sleep(delay)
end
end
end

```

- 2) Create 3 objects:
 - 1/1/1 - binary (dim up)
 - 1/1/2 - binary (dim down)

1/1/3 - 1-byte scale (output)

3) Create an event script for each binary object:

```
bindimmer('1/1/1', '1/1/2', '1/1/3', event)
```

4) You can tune *step* and *delay* variables in *bindimmer* function to adjust dimming speed.

DALI commands

Command	Description	Addressable	Reply	Value
arc	direct arc power control	+		0..254
off	turn off	+		
up	turn on	+		
down	down	+		
stepup	step up	+		
stepdown	step down	+		
recallmin	recall max level	+		
recallmax	recall min level	+		
stepdownoff	step down and off	+		
stepupon	on and step up	+		
gotoscene	go to scene			0..15
reset	reset	+		
storeactual	store actual level in the dtr	+		
storemax	store the dtr as max level	+		
storemin	store the dtr as min level	+		
storesystemfailure	store the dtr as system failure level	+		
storepoweron	store the dtr as power on level	+		
storefadetime	store the dtr as fade time	+		
storefaderate	store the dtr as fade rate	+		
storescene	store the dtr as scene	+		0..15
removescene	remove from scene	+		0..15
addtogroup	add to group	+		0..15
removefromgroup	remove from group	+		0..15
storeshortaddress	store dtr as short address	+		
querystatus	query status	+	+	
queryballast	query ballast	+	+	
querylampfailure	query lamp failure	+	+	
querylamppoweron	query lamp power on	+	+	
querylimiterror	query limit error	+	+	
queryresetstate	query reset state	+	+	
querymissingshort	query missing short address	+	+	

queryversion	query version number	+	+	
querydtr	query content dtr	+	+	
querydevicetype	query device type	+	+	
queryphysicalmin	query physical minimum level	+	+	
querypowerfailure	query power failure	+	+	
queryactual	query actual level	+	+	
querymax	query max level	+	+	
querymin	query min level	+	+	
querypoweron	query power on level	+	+	
querysystemfailure	query system failure level	+	+	
queryfadetimerate	query fade time / fade rate	+	+	
queryscene	query scene level (scenes 0-15)	+	+	0..15
querygroupslow	query groups 0-7	+	+	
querygroupshigh	query groups 8-15	+	+	
queryrandomaddrh	query random address (h)	+	+	
queryrandomaddrm	query random address (m)	+	+	
queryrandomaddrl	query random address (l)	+	+	
terminate	terminate			
setdtr	set data transfer register (dtr)			0..255
initialise	initialise			
randomise	randomise			
compare	compare		+	
withdraw	withdraw			
searchaddrh	set search address (h)			0..255
searchaddrm	set search address (m)			0..255
searchaddrl	set search address (l)			0..255
programshortaddr	program short address			0..63
verifyshortaddr	verify short address		+	0..63
queryshortaddr	query short address		+	
physicalselection	physical selection			
enabledevicetype	enable device type x			0..255

15. DMX interconnection with LM

DMX protocol support is realized upon RS485 serial port.

DMX function

Add the following user library in *Scripting* → *User libraries*.

```
1. local luadm = require('luadm')
2. module('DMX', package.seeall)
3.
4. local DMX = {}
5.
6. -- default params
7. local defaults = {
8.   -- storage key
9.   skey = 'dmx_line_1',
10.  -- RS-485 port
11.  port = '/dev/RS485',
12.  -- number of calls per second
13.  resolution = 20,
14.  -- total number of channels to use
15.  channels = 3,
16.  -- transition time in seconds, does not include DMX transfer time
17.  transition = 2,
18. }
19.
20. -- value setter
21. function set(chan, val, key)
22.   key = key or defaults.skey
23.   chan = tonumber(chan) or 0
24.   val = tonumber(val) or -1
25.
26.   -- validate channel number and value
27.   if chan >= 1 and chan <= 512 and val >= 0 and val <= 255 then
28.     storage.exec('lset', key, chan - 1, val)
29.   end
30. end
31.
32. -- value getter
33. function get(chan, key)
34.   local res, val
35.   key = key or defaults.skey
36.   chan = tonumber(chan) or 0
37.
38.   -- validate channel number and value
39.   if chan >= 1 and chan <= 512 then
40.     res = storage.exec('lrange', key, chan - 1, chan - 1)
41.     if type(res) == 'table' then
```

```

42.         val = tonumber(res[ 1 ])
43.     end
44. end
45.
46.     return val
47. end
48.
49. -- DMX init, returns new DMX object
50. function init(params)
51.     local n, k, v, _
52.
53.     -- create metatable and set user parameters
54.     n = setmetatable({}, { __index = DMX })
55.     n.params = params or {}
56.
57.     _, n.conn = pcall(require('redis').connect)
58.
59.     -- merge parameters that are set by user
60.     for k, v in pairs(defaults) do
61.         if n.params[ k ] == nil then
62.             n.params[ k ] = v
63.         end
64.     end
65.
66.     n:reset()
67.
68.     return n
69. end
70.
71. function DMX:reset()
72.     local err, chan, params
73.
74.     params = self.params
75.     self.dm, err = luadm.open(params.port)
76.
77.     -- error while opening
78.     if err then
79.         os.sleep(1)
80.         error(err)
81.     end
82.
83.     -- set channel count
84.     self.dm:setcount(params.channels)
85.
86.     -- number of transaction ticks
87.     self.ticks = math.max(1, params.transition * params.resolution)
88.
89.     -- calculate sleep time
90.     self.sleep = 1 / params.resolution
91.

```

```

92.     -- reset channel map
93.     self.channels = {}
94.
95.     -- empty channel value map
96.     self.conn:ltrim(params.skey, 1, 0)
97.
98.     -- fill channel map
99.     for chan = 1, params.channels do
100.         self.channels[ chan ] = { current = 0, target = 0, ticks = 0 }
101.
102.         -- turn off by default
103.         self.conn:lpush(params.skey, 0)
104.         self.dm:setchannel(chan, 0)
105.     end
106. end
107.
108. -- get new values
109. function DMX:getvalues()
110.     local max, channels, ticks, values, val
111.
112.     max = self.params.channels
113.     channels = self.channels
114.     ticks = self.ticks
115.     values = self.conn:lrange(self.params.skey, 0, max - 1) or {}
116.
117.     -- check for new values for each channel
118.     for chan = 1, max do
119.         val = tonumber(values[ chan ]) or 0
120.
121.         -- target value differs, set transaction
122.         if val ~= channels[ chan ].target then
123.             channels[ chan ].target = val
124.             channels[ chan ].delta = (channels[ chan ].target - channels[
chan ].current) / ticks
125.             channels[ chan ].ticks = ticks
126.         end
127.     end
128. end
129.
130. -- main loop handler
131. function DMX:run()
132.     self:getvalues()
133.
134.     -- transition loop
135.     for i = 1, self.params.resolution do
136.         self:step()
137.         self.dm:send()
138.         os.sleep(self.sleep)
139.     end
140. end

```

```

141.
142. -- single transition step
143. function DMX:step()
144.     local chan, channels, t
145.
146.     channels = self.channels
147.
148.     -- transition for each channel
149.     for chan = 1, self.params.channels do
150.         t = channels[ chan ].ticks
151.
152.         -- transition is active
153.         if t > 0 then
154.             t = t - 1
155.
156.             channels[ chan ].current = channels[ chan ].target - channels[
chan ].delta * t
157.             channels[ chan ].ticks = t
158.
159.             self.dm:setchannel(chan, channels[ chan ].current)
160.         end
161.     end
162. end

```

DMX handler script

Add the following resident script with sleep interval = 0, adjust port and channel as needed

```

1. if not dmxhandler then
2.     require('user.dmx')
3.     dmxhandler = DMX.init({
4.         port = '/dev/RS485', -- RS-485 port name
5.         channels = 8, -- number of DMX channels to use
6.         transition = 2, -- soft transition time in seconds
7.     })
8. end
9.
10. dmxhandler:run()

```

Setter (used in other scripts)

```
DMX.set(channel, value)
```

Mark DMX objects

Create objects with DMX tag, where last part of group address is DMX address (starting from 1).
Create event script mapped to DMX tag.

```

1. require('user.dmx')
2. -- get ID as group address last part (x/y/ID)
3. id = tonumber(event.dst:split('/')[3])
4. -- get event value (1 byte scaling)
5. value = event.getvalue()
6. -- convert from [0..100] to [0..255]
7. value = math.floor(value * 2.55)
8. -- set channel ID value
9. DMX.set(id, value)

```

Predefined scene example

The following example should be placed inside a resident script. Sleep time defines scene keep time (at least 1 second).

```

1. if not scenes then
2.   -- 3 channel scene
3.   scenes = {
4.     { 255, 0, 0 },
5.     { 0, 255, 0 },
6.     { 0, 0, 255 },
7.     { 255, 255, 0 },
8.     { 0, 255, 255 },
9.     { 255, 0, 255 },
10.    { 255, 255, 255 },
11.   }
12.
13.   current = 1
14. end
15.
16. -- set current scene values
17. scene = scenes[ current ]
18. for i, v in ipairs(scene) do
19.   DMX.set(i, v)
20. end
21.
22. -- switch to next scene
23. current = current + 1
24. if current > #scenes then
25.   current = 1
26. end

```

Random scene example

The following example should be placed inside a resident script. Sleep time defines scene keep time (at least 1 second).

```

1. -- number of steps to use, e.g. 3 steps = { 0, 127, 255 }
2. steps = 5
3. -- number of channels to set

```



```
4. channels = 3
5. -- first channel number
6. offset = 1
7.
8. for i = offset, channels do
9.   v = math.random(0, (steps - 1)) * 255 / (steps - 1)
10.   DMX.set(i, math.floor(v))
11. end
```

16. 1-wire configuration

LM5 Power (LM5p-DW1) has two 1-wire interfaces built-in. 1-wire is a bus technology which is built based on client-server topology and allowing to connect up to 100 devices to one controller. It is either 2-wire or 3-wire bus installation. In case of 2-wire system, a parasitic powering is used directly from the bus, normally up to 20 devices can work in this way. In case of bigger amount of 1-wire sensors, you can use LogicMachine 5V DC output to power 1-wire devices.

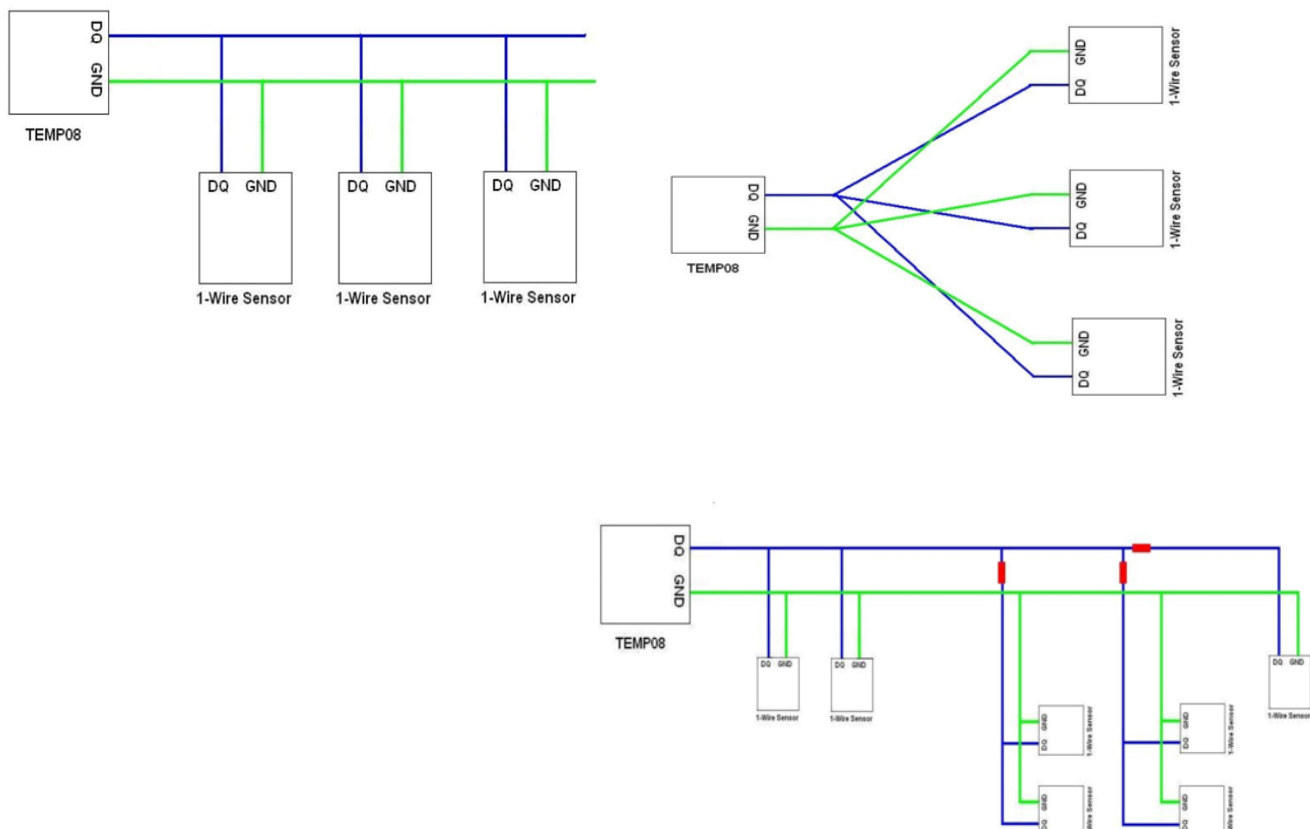
Advantages of 1-wire over KNX:

- No need in ETS
- Very cost-effective
- You can use the same wiring as KNX does and connect all standard sensors

Advantages of 1-wire over resistive sensors:

- Substantial savings on equipment
- Easier connection diagram allows to reduce the complexity of laying wiring
- Extension possibility: connection of additional sensors without changing basic wiring
- Ability of remote monitoring of sensors (open circuit, short circuit etc.)
- No need to take into account the resistance of conductors like in the circuit with resistive sensors

1-wire connection diagrams:



Once 1-wire sensors are connected to the 1-wire interface of LogicMachine5

The screenshot shows the Logic Machine software interface. At the top, there are tabs for Scripting, Objects, Object logs, Schedulers, Trend logs, Vis. structure, Visualization, Vis. graphics, Utilities, Enocean, 1-wire, Alerts, Logs, Error log, and Help. Below the tabs is a table with columns: ID, Name, Linked to object, Sensor value, Configuration, and Value received at. The table contains two rows of sensor data. A dialog box titled "Sensor 000051083d2" is open, showing configuration options for a specific sensor. The dialog box has the following fields:

ID	Name	Linked to object	Sensor value	Configuration	Value received at
000051083d2	test1234	1/1/10 test1234	19.31°C	Send delta: 5°C; Send mode: Internal update	15.05.2014 11:20:03
0000511391f	0000511391f	1/1/5 0000511391f	19.25°C	Send delta: 2°C; Send timer: 10 sec; Send mode: Internal update	14.05.2014 17:00:01

The dialog box fields are:

- Name: test1234
- Linked to object: 1/1/10 test1234
- Sensor status object: (empty)
- Write to bus:
- Send delta (°C): 5
- Send timer (seconds): (empty)
- Value compensation: 0

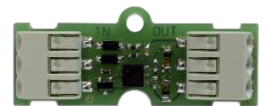
Buttons: Save, Cancel

Version: 20140431 CPU/IO: 0.22 0.22 0.12, Memory: 8%, KNX/IP Sync project data

- **Name** – name of the 1wire device
- **Linked to object** – mapped KNX object
- **Sensor status object** – mapped KNX status object
- **Write to bus** – define either to write telegram in KNX bus on read value
- **Send delta** – define either to send delta of temperature sensor
- **Send timer (seconds)** – define interval in which send the measurement
- **Value compensation** – compensate value of the reading of temperature

1-wire PUZZLE - 2 Universal Digital Inputs / Outputs

With PUZZLE you can interconnect regular binary sensors with 1-wire fieldbus or use the device as digital output to control relays/contactors.



By pairing the PUZZLE with LogicMachine Ambient, you can finish the installation in very cost-effective way. Up to 64 PUZZLE can be connected to one 1-wire port of LogicMachine and all objects can be then managed from any of KNX, Modbus, BACnet etc. devices.



Note! Max 1-wire network length is 100 m, you need 3-wires: +5V, 1-Wire, GND. It can take up to 0.5 seconds for each sensor reading. We recommend to connect up to 30 sensors.

Further, in the latest firmware of LogicMachine there are features like increased polling time of 1-wire objects, implemented keepalive object which indicates of error if there was lost connection with at least one of 1-wire device. Also in the configuration menu on LogicMachine you can see pre-configured 1-wire devices and newly found ones in separate lists.

17. 3G modem connection with LM

LogicMachine has a regular 3G modem driver built-in (Huawei and other vendor supported). Currently this can be used for SMS notifications only – receiving and sending commands. The modem has to be plugged into USB port. We suggest to use external 5V powering for the modem because by USB2.0 standard the output current on USB is 0.75A, but some modems requires up to 2A which is out of standard so the modem can lack the power and get disconnected.

List of supported 3G modems can be found here:

http://openrb.com/wp-content/uploads/2015/12/3G_USB_device_reference.txt

First thing is to lower the modem speed by adding the following code in *Start-up / Init* script:

1. `os.execute('echo 1 > /sys/bus/platform/devices/ci_hdrc.0/force_full_speed')`
2. `os.execute('echo 1 > /sys/bus/platform/devices/ci_hdrc.1/force_full_speed')`
3. `os.execute('usbreset /dev/bus/usb/001/001')`

After you need to add SMS handler program – a resident script with sleep interval 0.

Note! Change white list telephone numbers and SIM card's PIN code in the below script.

```
1.-- init
2.ifnot modem then
3.-- allowed numbers, SMS message from other number will be ignored
4.  numbers ={'1234567890', '0123456789'}
5.-- replace 0000 with SIM pin number, or remove the line below if PIN check is disabled
6.pincode='0000'
7.-- modem communication port, ttyUSB2 for Huawei E173
8.  comport ='ttyUSB2'
9.-- open serial port
10.  modem =AT:init('/dev/' .. comport)
11.-- command parser
12.  parser =function(cmd, sender)
13.local find, pos, name, mode, offset, value, jvalue, obj
14.cmd=cmd:trim()
15.  mode =cmd:sub(1, 1):upper()
16.if mode =='W'or mode =='R'then
17.cmd=cmd:sub(3):trim()
18.-- parse object name/address
19.  find =cmd:sub(1, 1)=='and'or' '
20.  offset = find ==''and 1 or0
21.-- pad with space when in read mode
22.if mode =='R'and find ==' 'then
23.cmd=cmd .. ' '
24.end
25.-- find name
26.pos=cmd:find(find, 1 + offset, true)
27.-- name end not found, stop
```

```

28. ifnot post then
29. return false
30. end
31. -- get name part
32.     name =cmd:sub(1 + offset, pos - offset):trim()
33. if mode == 'W' then
34.     value =cmd:sub(pos + offset):trim()
35. ifnot value then
36. return false
37. end
38. -- try decoding value
39. jvalue=json.pdecode(value)
40.     value =jvalue ~=nil and jvalue or value
41. -- send to bus
42. grp.write(name, value)
43. -- read request
44. else
45. obj=grp.find(name)
46. -- send read request and wait for update
47. if obj then
48. obj:read()
49. os.sleep(1)
50. -- read new value
51.     value =grp.getvalue(name)
52. -- got value, send response
53. if value ~=nil then
54. jvalue=json.pencode(value)
55. if obj.name then
56.     name =string.format('%s (%s)', obj.name, obj.address)
57. end
58. cmd=string.format('Value of %s is %s', name, jvalue)
59. modem:sendsms(sender, cmd)
60. end
61. end
62. end
63. end
64. end
65. -- incoming sms handler
66. handler =function(sms)
67.     alert('incoming sms from %s (%s)', sms.sender, sms.data)
68. -- sms from known number, call parser
69. if table.contains(numbers, sms.sender) then
70.     parser(sms.data, sms.sender)
71. end
72. end
73. -- set sms handler
74. modem:setsmshandler(handler)
75. -- send pin if set
76. if pincode then
77. modem:send('AT+CPIN=' .. pincode)

```

```
78. end
79. -- set to pdu mode
80. modem:send('AT+CMGF=0')
81. -- enable sms notifications
82. modem:send('AT+CNMI=1,1,0,0,0')
83. alert('SMS handler started')
84. end
85. modem:run()
```

Command syntax:

- a. Write to bus:
W ALIAS VALUE
- b. Read from bus:
R ALIAS

On read request, script will reply with SMS message containing current value of selected object.

ALIAS can be:

- a. Group address (e.g. 1/1/1)
- b. Name (e.g. Obj1). If name contains spaces then it must be escaped using double quotes (e.g. "Room Temperature")

NOTE:

- a. Object data type and name must be set in Objects tab. Otherwise script won't be able to read and write to object.
- b. Only ASCII symbols are accepted in the message.

17.1. Examples

Binary write (send the following SMS to switch kitchen lights on):

```
W 1/1/1 true
```

Scaling write (send the following SMS to set value 67% for red LED):

```
W LED1Red 67
```

Temperature (floating point) write (send the following SMS to make setpoint in the living room to 22.5 degrees):

```
W "Room Setpoint" 22.5
```

Read (send the following SMS to read the security panel value:

```
R 2/1/1
```

17.2. Send SMS messages to specific SIM numbers after group-read or group-write is triggered

Task: Assume we have an Event-based script which triggers a program once group-read or group-write is triggered for address 1/1/1. We want to send SMS to numbers 23335555 and 23335556 with 1/1/1 actual status.

```
1. require('socket')
2.
3. client =socket.udp()
4.
5. -- in the message field the number where SMS has to be send should be specified at the
   beginning
6. local msg='23335555 1/1/1 changes its value to: ' .. tonumber(event.datahex)
7. client:sendto(msg, '127.0.0.1', 12535)
8.
9. msg='23335556 1/1/1 changes its value to: ' .. tonumber(event.datahex)
10. client:sendto(msg, '127.0.0.1', 12535)
```

17.3. Send SMS messages without 3G modem

How to send event SMS to mobile phone from LogicMachine through Twilio service, without external 3G adapter?

You can use Twilio service which offers free of charge SMS in the test period and messaging at \$0.01 for regular usage. The only disadvantage is it will use your standard Internet connection to send messages to Twilio servers (not via GSM as with 3G adapters).

Twilio account

You can get ID and Token needed for the below example by registering on Twilio. Make sure you enter a verified SIM number list / recipients in your account. Or please contact us for ready example with our account data.

Function

Add the following function in *Scripting* → *Common functions*

```
1. function sms(id, token, from, to, body)
2.     local escape = require('socket.url').escape
3.     local request = require('ssl.https').request
4.     local url = string.format('https://%s:%s@api.twilio.com/2010-04-
   01/Accounts/%s/Messages.json', id, token, id)
5.     local body = string.format('From=%s&To=%s&Body=%s', escape(from),
   escape(to), escape(body))
6.
7.     return request(url, body)
```


8. `end`

Event-based script

Add event-based program for specific object, like 1/1/2 in this example

```
1. value = event.getvalue()
2.
3. from_nr = '+37112345679' -- put sender SIM nr here
4. to_nr = '+37112345678' -- put receipient SIM nr here
5. id_nr = 'ACe56f5' -- put your ID here
6. token_nr = '598c6ff' -- put your token here
7.
8. sms(id_nr, token_nr, from_nr, to_nr, 'The value for 1/1/2 has changed
   to'..toString(value))
```

18. Communication with RS232/RS485 serial ports

The following are the naming of Serial ports for different versions of Logic Machine.

LM4		Reactor, LM5		Reactor V2	
GND		GND		GND	
RS485 A	RS485-1	RS485 A	RS485-1	RS485 A	RS485
RS485 B		RS485 B		RS485 B	
GND		GND			
RS485 A	RS485-2	RS485 A	RS485-2		
RS485 B		RS485 B			
GND					
RS485 A	RS485-3				
RS485 B					

Reactor V3		LM5L, LM5-RIO, LM5-RIOE	
GND		RS485 A	RS485-1
RS485 A	RS485	RS485 B	
RS485 B			GND
		RS485 A	
		RS485 B	
		GND	
		TX	RS232
		RX	
		GND	

Note! LM5 series devices have 1 definitive serial port RS-485 and other one can work either as RS-485 or RS-232. The one will work which is most recently opened.

If the following command is used, you activate RS-485 second port:

```
port = serial.open('/dev/RS485-2', { baudrate = 115200, parity = 'even', duplex = 'half' })
```

If the following command is used, you activate RS-232 port:

```
port = serial.open('/dev/RS232', { baudrate = 9600, parity = 'even', duplex = 'full' })
```

Functions

Include library before calling serial functions:

```
require('serial')
```

Opens given port, returns: port handle, or, in case of error, nil plus error message

```
port, err = serial.open(device, params)
```

Parameters:

- **device** port device name, required
- **params** parameters table, optional, (defaults are in bold):
 - **baudrate** 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400
 - **parity** "none", "even", "odd"
 - **databits** 5, 6, 7, 8
 - **stopbits** 1, 2
 - **duplex** "full", "half" (Note: "half" is required for RS-485)

Reads the specified number of bytes, execution is blocked until read is complete

```
res, err = port.read(bytes)
```

Parameters:

- **bytes** number of bytes to read

Reads until timeout occurs or the specified number of bytes is received, whichever happens first.

Returns data plus number of bytes read, or, in case of error, nil plus error message.

```
res, err = port.read(bytes, timeout)
```

Parameters:

- **bytes** number of bytes to read
- **timeout** maximum time to wait for read to complete, minimum value and timer resolution is 0.1 seconds

Flushes any read/unsent bytes

```
port.flush()
```

Closes serial port, no other port functions may be called afterwards

```
port.close()
```

Examples

Write to port

```
port:write('test data')
```

Blocking read (script will block until 10 characters are read)

```
data=port:read(10)
```

Timeout read (script will wait for 10 characters for 20 seconds)

```
data=port:read(10, 20)
```

Close serial port

```
port:close()
```

Resident script, RS-485 echo test

```
-- open port on first call
if not port then
  require('serial')
  port = serial.open('/dev/RS485-1', { baudrate = 9600, parity = 'even', duplex = 'half' })
  port:flush()
end

-- port ready
if port then
  -- read one byte
  char = port:read(1, 1)
  -- send back if read succeeded
  if char then
    port:write(char)
  end
end
```

HEX to RS-485 example

```
require('serial')
port = serial.open('/dev/RS485-1', {
  baudrate = 4800,
  parity = 'none',
  duplex = 'half'
})
```

```
cmd = string.char(0xAB, 0xF1, 0xFF, 0xFF, 0xFF, 0xFF, 0xBE, 0xD1, 0x01, 0xFE, 0xFF, 0xFF,
0x0A, 0x24)
cmd = 'ABF1FFFFFFFFBED101FEFFFF0A24'
```

```
port:write(cmd)
```

Check which **cmd** works for you, as it can be either hex-encoded readable data or hex representation of binary data. You might also need to change the parity config

<http://openrb.com/docs/serial.htm>

19. Bluetooth 4.0 integration

Bluetooth can be integrated over USB-Bluetooth adapter.

Some of supported Bluetooth 4.0 USB adapters:

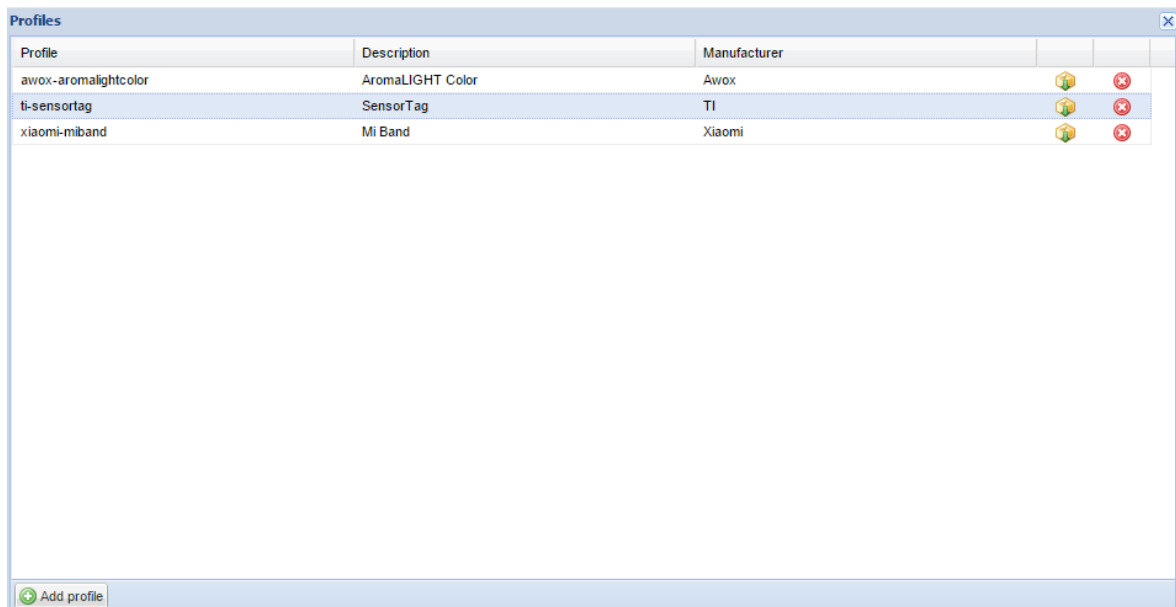
- Broadcom BCM20702A0
- Trust 18187
- Belkin F8T065bf
- Plugable USB Bluetooth 4.0
- Laird BT820

Configuration of Bluetooth is located in *LogicMachine* → *BLE* tab.

A support for any BLE device can be added if a communication protocol will be provided and will not change in the future software release of BLE device.

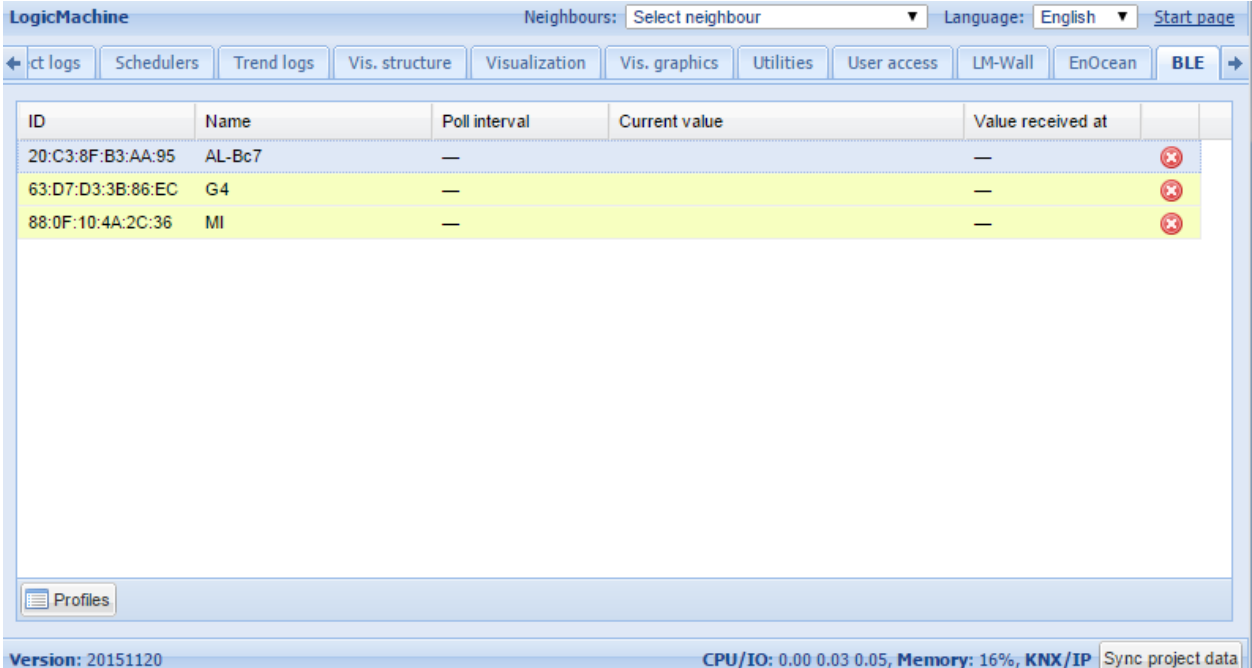
Profiles

List of supported BLE devices can be found by clicking *Profiles* button. To add a new profile, you have to upload *.lua profile file by clicking on *Add profile* button.



Mapping functionality to KNX group addresses

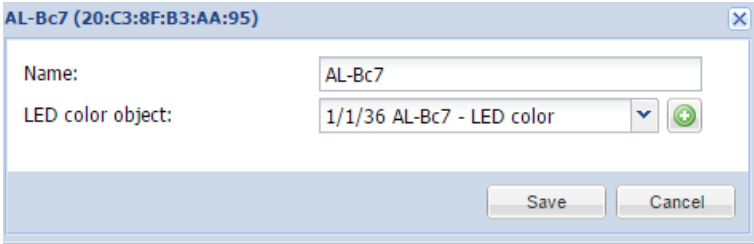
Once BLE device is seen by LogicMachine Ambient, it will automatically appear in the list.



The screenshot shows the LogicMachine interface with a table of detected BLE devices. The table has columns for ID, Name, Poll interval, Current value, and Value received at. Three devices are listed: AL-Bc7, G4, and MI. The MI device is highlighted in yellow.

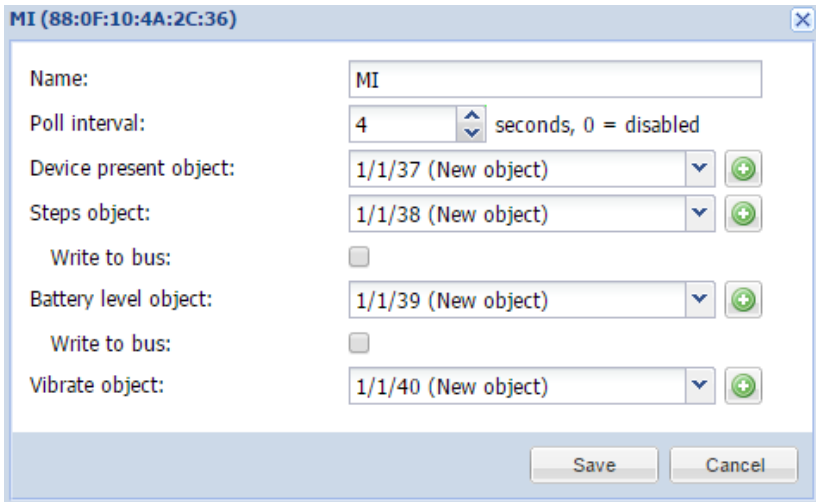
ID	Name	Poll interval	Current value	Value received at
20:C3:8F:B3:AA:95	AL-Bc7	—	—	—
63:D7:D3:3B:86:EC	G4	—	—	—
88:0F:10:4A:2C:36	MI	—	—	—

By clicking on specific device, you can map its functionality to KNX group addresses. For AWOX AromLight Color BLE lamp, you can map LED color object to KNX.



The screenshot shows the configuration dialog for device AL-Bc7 (20:C3:8F:B3:AA:95). The Name field is set to AL-Bc7. The LED color object is mapped to 1/1/36 AL-Bc7 - LED color. There are Save and Cancel buttons at the bottom.

There are following mapping objects for Xiaomi Mi Band wearable – device present object, steps counter, battery level, vibrate the band.



The screenshot shows the configuration dialog for device MI (88:0F:10:4A:2C:36). The Name field is set to MI. The Poll interval is set to 4 seconds. The Device present object is mapped to 1/1/37 (New object). The Steps object is mapped to 1/1/38 (New object). The Battery level object is mapped to 1/1/39 (New object). The Vibrate object is mapped to 1/1/40 (New object). There are Save and Cancel buttons at the bottom.

Example

Alpha MIO BLE watch has heart-rate as one of objects. This event-based script will switch on ventilation if the heart-rate is >80 and switch off if its lower

```
20. value = event.getvalue()
21. if value > 80 then
22.   grp.write('2/2/2', true)
23. else
24.   grp.write('2/2/2', false)
25. end
```

21. SIP server on LogicMachine

Task: How to pair SIP door entry systems with building automation project? In LogicMachine we have built SIP registrar which can send SIP requests to final SIP clients. For example, one can install Linphone SIP client app on touch devices which are used for visualization control. Upon SIP request from door entry system, LogicMachine will forward the request to the respective SIP client / recipient. On this client's device a new window will appear with options to answer or reject the call. When the call is answered, you will see video and audio from the door entry system. When the call is finished, Linphone app will go to the background.

SIP package installation on LM:

Add the following Resident script, 60 sec sleep time, run once:

```
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/terminfo_5.7-5_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/libncurses_5.7-5_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/libreadline_5.2-2_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3_3.3.7-1_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3-mod-maxfwd_3.3.7-1_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3-mod-registrar_3.3.7-1_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3-mod-rr_3.3.7-1_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3-mod-sl_3.3.7-1_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3-mod-tm_3.3.7-1_mxs.ipk')
os.execute('opkg --force-depends install http://dl.openrb.com/pkg/kamailio/kamailio3-mod-usrloc_3.3.7-1_mxs.ipk')

os.execute('/etc/init.d/kamailio enable')
os.execute('/etc/init.d/kamailio start')
```

Check if LM has Internet access

Check that IP, gateway, subnet, DNS are set correctly set.

Interface eth0	
Protocol	Static IP
IP address	192.168.1.16
Network mask	255.255.255.0
Gateway IP	192.168.1.100
DNS server 1	8.8.8.8
DNS server 2	
Mtu	
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

SIP client application

You can use for example Linphone as your SIP client. You have to enter IP of LogicMachine in its settings.

The screenshot shows the 'SIP ACCOUNT' settings in the Linphone application on an iPad. The settings are as follows:

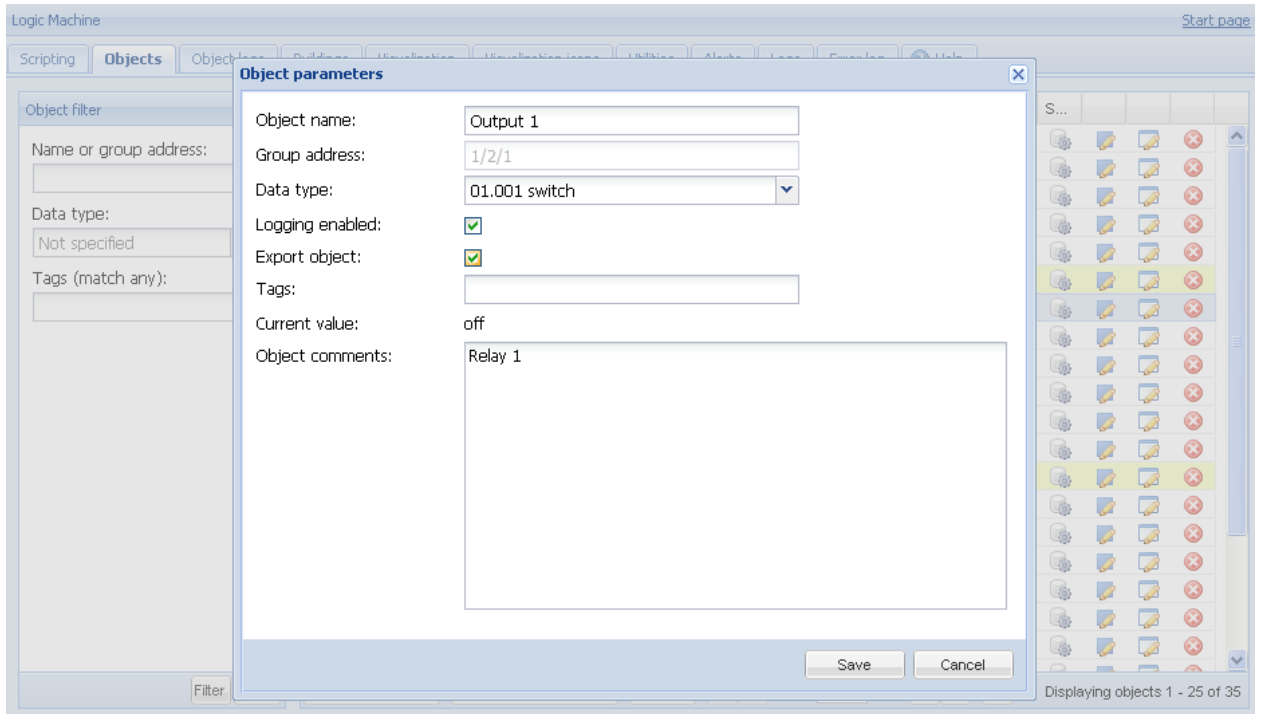
- SIP ACCOUNT**
 - Run assistant
 - User name: 2
 - Password: [Redacted]
 - Domain: 192.168.1.16
 - Proxy: [Redacted]
 - Transport: UDP
 - Outbound proxy: OFF
 - AVPF: OFF
- SETTINGS**
 - Enable video: ON
 - Audio: >
 - Video: >
 - Call: >
 - Network: >
 - Advanced: >

The bottom navigation bar includes History, Contacts, a central grid icon, Chat, and Settings.

22. Object value export via XML

Make KNX objects XML readable

In the *Objects* tab click on the objects which you want to receive the current value by XML request. Check the Export object



XML request from external PC

The XML request looks like this:

<http://remote:remote@192.168.1.211/cgi-bin/scada-remote/request.cgi?m=xml&r=objects>

Parameters:

- **address** – object address (e.g. “1/1/1”)
- **name** – object name (e.g. “My object”)
- **data** – decoded object value (e.g. 42 or “01.01.2012”)
- **datatype** – object datatype (e.g. 1 or 5.001) – standard KNX data types
- **time** – object update time (UNIX timestamp)
- **date** – object update time (RFC date)
- **comment** – object comment (e.g. “Second floor entry lights”)
- **tags** – optional array of object tags (e.g. “Light”, “Second floor”)

Note! To get list of objects that have been updated after specific time you can pass an optional “updatetime” parameter (UNIX timestamp format)

← → ↻ 192.168.1.211/cgi-bin/scada-remote/request.cgi?m=xml&r=objects ☆ 🔧

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<objects>
  <object>
    <comment/>
    <name>Weather Temperature</name>
    <address>5/1/2</address>
    <date>Tue, 14 Feb 2012 23:41:45 -1000</date>
    <time>1329298905</time>
    <data>-4</data>
    <datatype>9</datatype>
  </object>
  <object>
    <comment/>
    <name>Weather T Low</name>
    <address>5/1/4</address>
    <date>Tue, 14 Feb 2012 23:41:45 -1000</date>
    <time>1329298905</time>
    <data>-13</data>
    <datatype>9</datatype>
  </object>
  <object>
    <comment/>
    <name>Weather T High</name>
    <address>5/1/5</address>
    <date>Tue, 14 Feb 2012 23:41:45 -1000</date>
    <time>1329298905</time>
    <data>-8</data>
    <datatype>9</datatype>
  </object>
</objects>
```

Login, Password for remote XML request

Login and password can be changed in *Network Configuration* → *System* → *GUI Login* → *Admin/Remote* tab.

GUI login

Admin / Remote Visualization

Login admin

Password

Repeat password

Admin user has access to Logic Machine and Network Configuration interfaces

Login remote

Password

Repeat password

OK Cancel

23. Alerts, Errors values

In similar way also Alerts and Errors can be read by XML requests.

Alerts XML request:

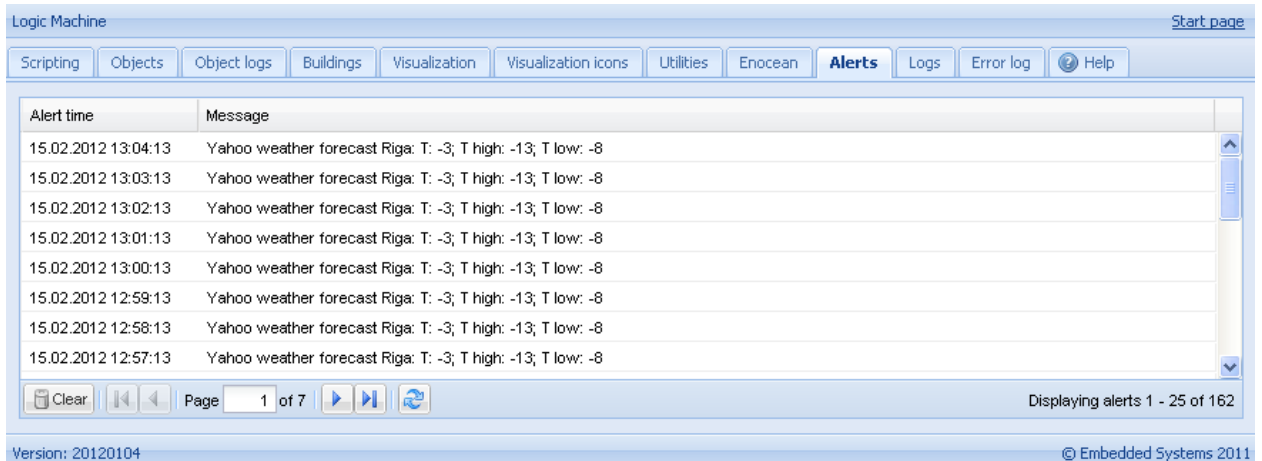
<http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=xml&r=alerts>

Errors XML request:

<http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=xml&r=errors>

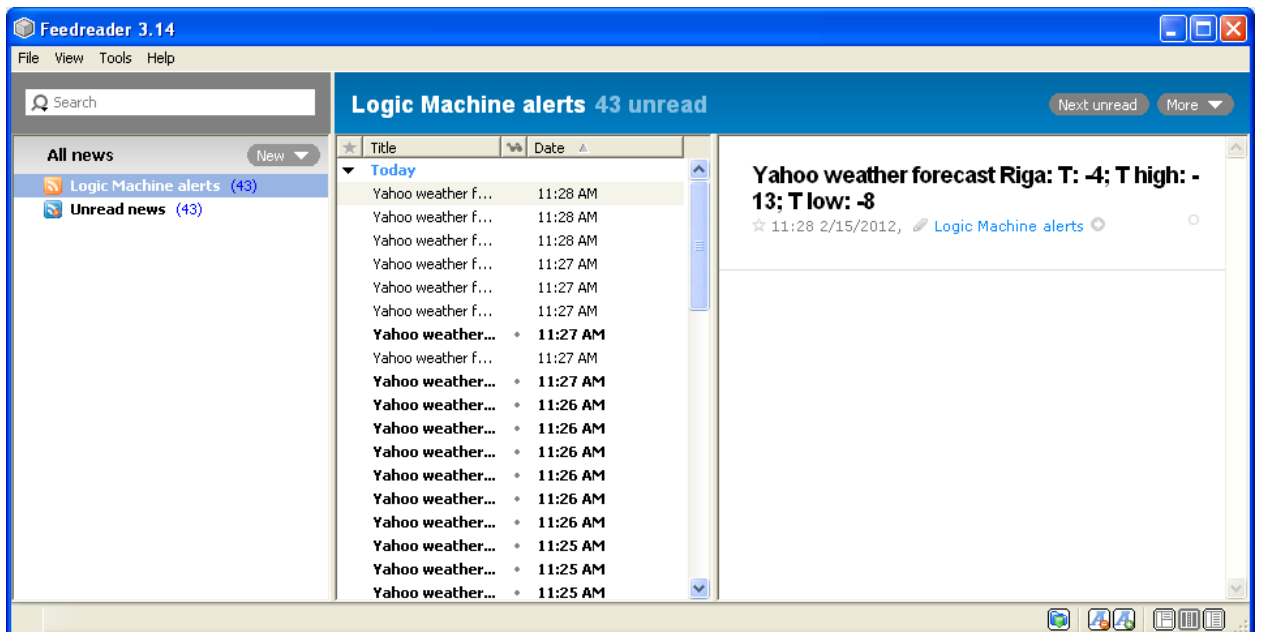
24. Read Alerts RSS feeds from LogicMachine

It is possible to read Alerts and Errors messages by remote RSS readers.



Add new RSS feed in the RSS reader

- Use the following URL:
- <http://remote:remote@192.168.1.211/cgi-bin/scada-remote/request.cgi?m=rss&r=alerts>
- 50 latest alerts will be shown
- *alert time* will be shown in *UNIX timestamp*, *alert date* will be shown as *RFC date*



Error tab content by RSS

RSS can be used to read Error tab content as well. In this case the URL would look like:

<http://remote:remote@192.168.1.211/cgi-bin/scada-remote/request.cgi?m=rss&r=errors>

Login, Password for remote RSS requests

Login and password can be changed in *System Configuration* → *System* → *User access* → *Admin/Remote* tab.

The screenshot shows a dialog box titled "User access" with a close button (X) in the top right corner. It has two tabs: "Admin / Remote" (selected) and "Visualization". The dialog is divided into two sections. The first section is for the "admin" user, with "Login" set to "admin". It has a "Password" field with six dots and a "Repeat password" field with six dots. The second section is for the "remote" user, with "Login" set to "remote". It also has a "Password" field with six dots and a "Repeat password" field with six dots. At the bottom of the dialog, there are "OK" and "Cancel" buttons.

25. Other examples

Different examples, 3rd party protocol integration and other useful applications can be found here:

<http://openrb.com/all-examples/>

<http://forum.logicmachine.net/>