

Embedded Systems SIA, VAT No LV40003411103  
47. Katolu str., Riga, LV 1003, LATVIA  
Phone: +371 67648888, fax: +371 67205036, e-mail: [sales@openrb.com](mailto:sales@openrb.com)

# LogicMachine4

## Product Manual



Document Issue 1.0  
December, 2014

Technical Support:  
[support@openrb.com](mailto:support@openrb.com)

## Copyright

Copyright © 2011 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** is your easiest way to program complex logic in KNX/EIB, Modbus, BACnet, EnOcean, DALI, 1-Wire networks. The LogicMachine will enable you to efficiently customize building automation processes, easily delivering unlimited flexibility benefit to end users in a cost-effective way.

**LogicMachine** is an embedded platform with integrated TPUART. LogicMachine allows to use it as IP Router, cross-standard gateway, logic engine, and visualization WEB SCADA server. Scripting templates provides user-friendly, flexible configuration interface. Via applying custom scripts the LogicMachine can simultaneously act as thermostat, security panel, lighting controller, etc

## 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](mailto:support@openrb.com).

Firmware updates are available at [www.openrb.com](http://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

<b>DEVICE SPECIFICATION.....</b>	<b>7</b>
<b>TERMINAL CONNECTION SCHEMES .....</b>	<b>10</b>
<b>FACTORY DEFAULT, DISCOVER IP .....</b>	<b>12</b>
<b>STANDARDS SUPPORTED .....</b>	<b>12</b>
<b>QUICK STARTUP GUIDE.....</b>	<b>14</b>
DEFAULT IP CONFIGURATION.....	14
DISCOVER LOGICMACHINE IP ADDRESS .....	16
FIRMWARE UPGRADE.....	17
LOGICMACHINE FOR KNX/EIB NETWORK CONFIGURATION MANAGEMENT WITH ETS .....	18
KNX AND IP ROUTER SETTINGS.....	20
QUICK GUIDE - CREATE VISUALIZATION FOR IPAD/PC.....	24
<b>GRAPHICAL USER INTERFACE LOGIN .....</b>	<b>31</b>
<b>1. LOGICMACHINE CONFIGURATION .....</b>	<b>32</b>
1.2. SCRIPTING .....	33
1.1.1. Block programming .....	33
1.1.2. Block functions .....	35
1.1.3. Adding a new script .....	37
1.1.4. Event-based scripting .....	39
1.2.3. Resident scripting .....	40
1.2.4. Scheduled scripting.....	40
1.2.5. Script editor .....	40
1.2.6. Object functions.....	42
1.2.7. Returned object functions, group communication functions.....	43
1.2.8. Group communication functions.....	44
1.2.9. Object function examples.....	44
1.2.10. Data type functions, data types.....	45
1.2.11. Data types .....	45
1.2.12. Data storage function .....	46
1.2.13. Alert function .....	47
1.2.14. Log function .....	47
1.2.15. Scheduled scripting date/time format .....	48
1.2.16. Time function .....	48
1.2.17. Data Serialization.....	48
1.2.18. String functions .....	49
1.2.19. Input and output functions .....	53
1.2.20. Script control functions.....	54
1.2.21. JSON library .....	54
1.2.22. Conversion.....	54
1.2.23. Bit operators .....	55
1.2.24. Input and Output Facilities.....	56
1.2.25. Mathematical functions.....	57
1.2.26. Table manipulations .....	59
1.2.27. Operating system facilities.....	59
1.2.28. Extended function library .....	61
1.2.29. User libraries.....	62
1.2.30. Common functions .....	63
1.2.31. Start-up (init) script .....	63
1.2.32. Tools .....	64
1.3. OBJECTS.....	65
1.3.1. Object parameters .....	66
1.3.2. Object visualization parameters .....	67
1.3.3. Change the object state.....	68
1.3.5. Object control bar .....	69
1.3.6. Filter objects.....	70
1.4. OBJECT LOGS .....	71
1.4.1. Export logs.....	72



1.5.	SCHEDULERS .....	75
1.5.1.	Add new scheduler.....	75
1.5.2.	Scheduler events .....	76
1.5.3.	Scheduler holidays.....	76
1.6.	TREND LOGS.....	77
1.6.1.	Add new trend log.....	77
1.6.2.	Trend logs functions .....	78
1.7.	VISUALIZATION STRUCTURE .....	80
1.7.1.	Levels / Plans.....	80
1.7.2.	Layouts / Widgets.....	83
1.8.	VISUALIZATION.....	86
1.8.1.	Plan editor .....	86
1.8.2.	Object .....	87
1.8.3.	Link .....	89
1.8.4.	Text Label .....	90
1.8.5.	Image .....	90
1.8.6.	Frame .....	91
1.8.7.	Gauge .....	92
1.8.8.	Camera .....	93
1.8.9.	Graph.....	94
1.9.	VIS.GRAPHICS .....	96
1.10.	UTILITIES .....	98
1.11.	ALERTS .....	102
1.12.	ERROR LOG .....	103
1.13.	LOGS .....	103
1.14.	HELP .....	104
<b>2.</b>	<b>USER MODE VISUALIZATION .....</b>	<b>105</b>
2.1.	CUSTOM DESIGN USERMODE VISUALIZATION .....	106
<b>3.</b>	<b>TOUCH VISUALIZATION .....</b>	<b>106</b>
<b>4.</b>	<b>SYSTEM CONFIGURATION .....</b>	<b>108</b>
4.1.	CHANGING PASSWORD .....	108
4.2.	PACKAGES.....	109
4.3.	UPGRADE FIRMWARE .....	110
4.4.	REBOOT LOGIC MACHINE .....	110
4.5.	SHUTDOWN LOGIC MACHINE.....	110
4.6.	INTERFACE CONFIGURATION .....	111
4.6.1.	Ethernet interface data throughput graph.....	112
4.7.	ROUTING TABLE .....	113
4.7.1.	Dynamic routes.....	113
4.7.2.	Static routes .....	113
4.8.	ARP TABLE.....	114
4.9.	FTP SERVER .....	114
4.10.	SYSTEM MONITORING .....	115
4.11.	NTP CLIENT .....	115
4.12.	SYSTEM STATUS .....	116
4.13.	NETWORK STATUS .....	116
4.14.	NETWORK UTILITIES .....	117
4.15.	SYSTEM LOG .....	117
4.16.	RUNNING PROCESSES .....	118
<b>5.</b>	<b>USER MODE SCHEDULERS .....</b>	<b>119</b>
5.1.	EVENTS .....	119
5.2.	HOLIDAYS .....	120
<b>6.</b>	<b>TREND LOGS .....</b>	<b>121</b>
<b>7.</b>	<b>MODBUS RTU/TCP INTERCONNECTION WITH LM.....</b>	<b>123</b>
7.1.	Modbus device profile .....	123
7.2.	Modbus RTU settings.....	124
7.3.	Adding Modbus device.....	125

7.4.	<i>Program address for UIO20 Modbus device</i> .....	126
7.5.	MODBUS SLAVE EXAMPLES .....	127
<b>6.</b>	<b>BACNET IP INTERCONNECTION WITH LM4</b> .....	<b>131</b>
6.1.	<i>BACnet server mode: transparent data transfer to BACnet network</i> .....	131
6.2.	<i>BACnet client mode</i> .....	132
<b>7.</b>	<b>1-WIRE CONFIGURATION</b> .....	<b>134</b>
<b>8.</b>	<b>DALI CONFIGURATION</b> .....	<b>136</b>
8.1.	DALI OBJECT MAPPING.....	137
8.2.	ACCESS DALI BUS FROM SCRIPTS.....	137
<b>9.</b>	<b>ENOCAN INTERCONNECTION WITH LOGICMACHINE</b> .....	<b>140</b>
9.1.	ENOCAN INTERFACES .....	140
9.2.	ENOCAN TO KNX MAPPING .....	140
9.3.	KNX TO ENOCAN MAPPING .....	142
<b>10.</b>	<b>CEC/HDMI INTEGRATION WITH LM4</b> .....	<b>145</b>
10.1.	CEC FUNCTION .....	145
10.2.	CEC COMMAND GENERATOR .....	147
10.3.	COMMON COMMANDS .....	147
<b>11.</b>	<b>DMX INTERCONNECTION WITH LM4</b> .....	<b>148</b>
11.1.	EXAMPLES .....	150
<b>12.</b>	<b>3G MODEM CONNECTION WITH LM4</b> .....	<b>152</b>
12.1.	EXAMPLES .....	154
12.2.	SEND SMS MESSAGES TO SPECIFIC SIM NUMBERS AFTER GROUP-READ OR GROUP-WRITE IS TRIGGERED 155	
12.3.	SEND SMS MESSAGES WITHOUT 3G MODEM.....	155
<b>13.</b>	<b>HDL PROTOCOL INTEGRATION IN LOGICMACHINE4</b> .....	<b>157</b>
13.1.	HDL FUNCTION.....	157
13.2.	USAGE EXAMPLE – HDL DIMMER CONTROL .....	159
13.3.	USAGE EXAMPLE – HDL RELAY CONTROL.....	159
<b>14.</b>	<b>COMMUNICATION WITH RS232/RS485 SERIAL PORTS</b> .....	<b>161</b>
<b>15.</b>	<b>BLUETOOTH 4.0 INTEGRATION</b> .....	<b>163</b>
<b>16.</b>	<b>SIP SERVER ON LOGICMACHINE</b> .....	<b>165</b>
<b>17.</b>	<b>OBJECT VALUE EXPORT VIA XML</b> .....	<b>167</b>
17.1.	ALERTS, ERRORS VALUES .....	169
<b>18.</b>	<b>READ ALERTS RSS FEEDS FROM LOGICMACHINE</b> .....	<b>170</b>

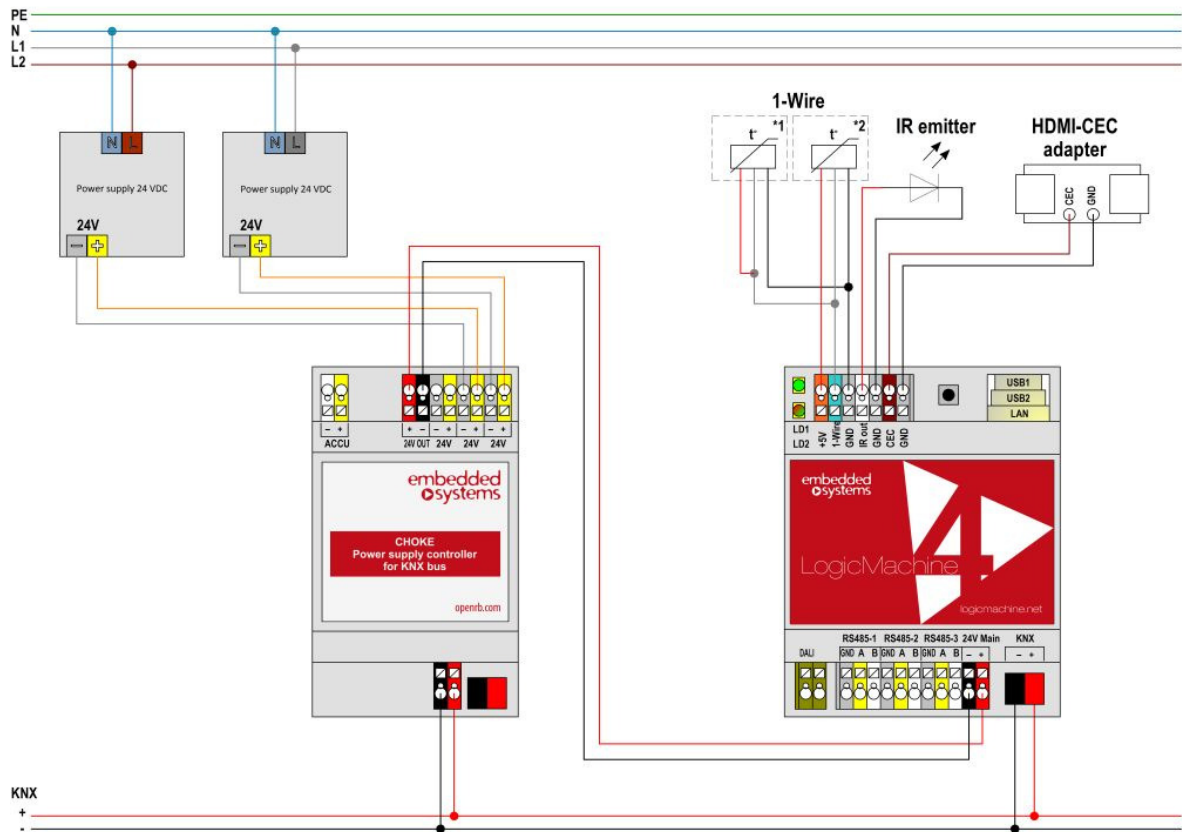


Enclosure:	Material:	Polyamide
	Color:	Gray
	Dimensions:	70(W)x90(H)x51(L) mm
Usage temperature:	0C ... +45C	
Storage temperature:	-15C ... +55C	
Weight:	150g	
Warranty:	2 years	
Relative Humidity:	10...95 % without condensation	

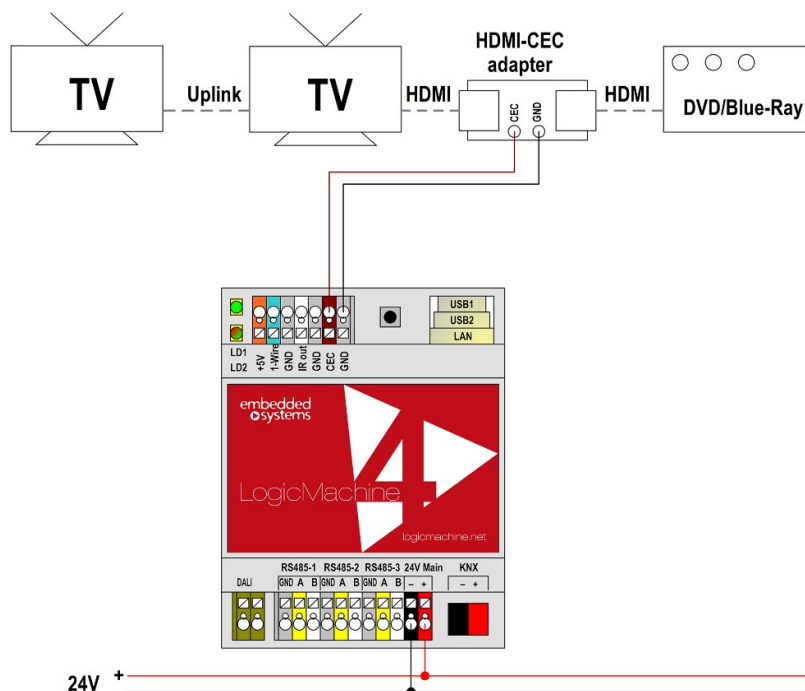
**LogicMachine4 kit contains:**

- Embedded board with preinstalled software
- Plastic DIN-rail case
- CEC-HDMI adapter (by request when making order)

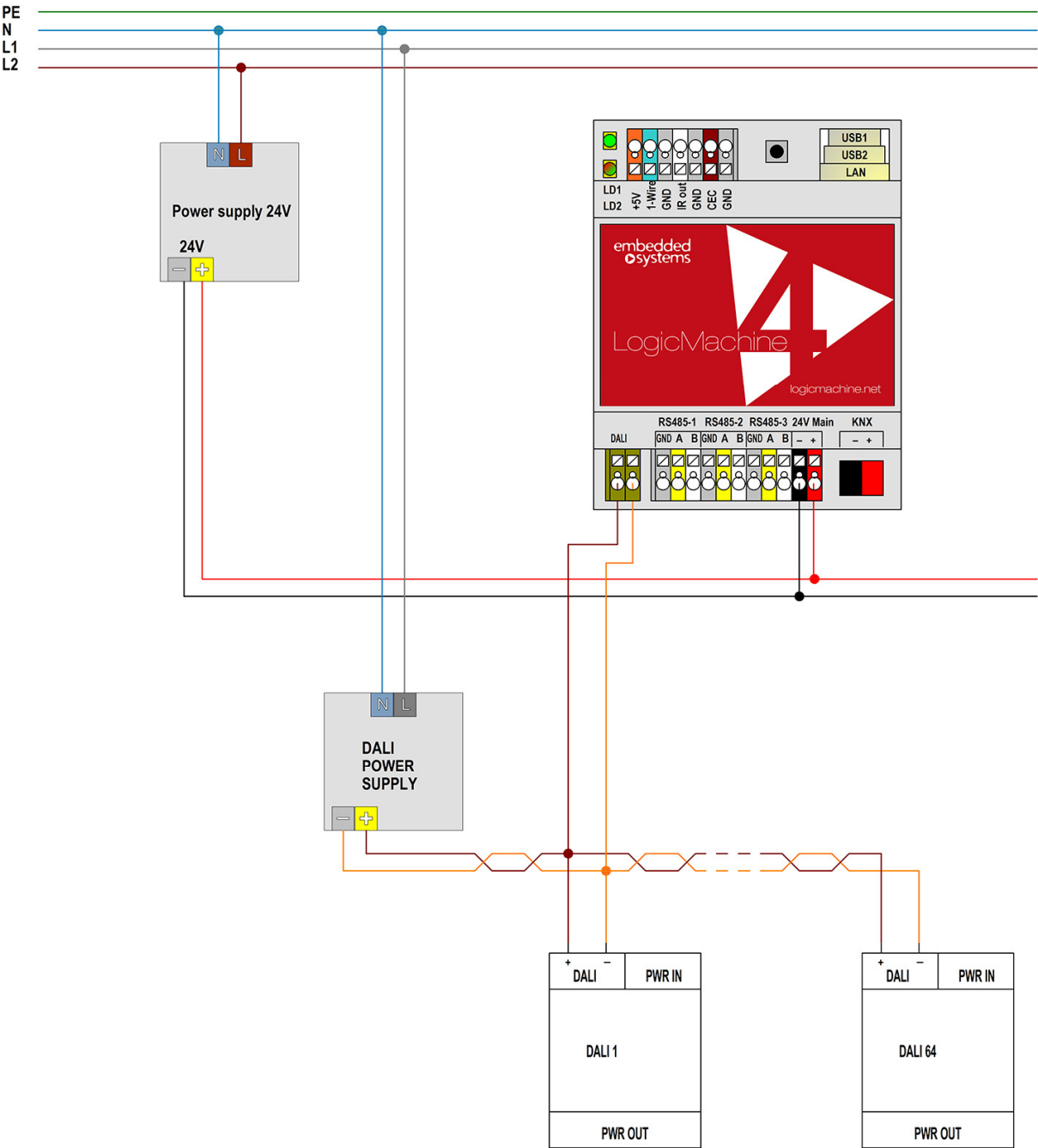
## Terminal connection schemes



## CEC connection



# DALI connection



## Factory default, discover IP

There is a reset button on the side of LogicMachine4. You can either reboot the device by pressing this 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

There is also another possibility to discover IP address – LM3 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*

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

## 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 (Huawei E173 and similar modem support through USB) for sending SMS notifications and controlling the installation by receiving SMS commands.
- DMX512 (in the box, through RS485)
- DALI
- 1-Wire
- CEC/HDMI
- Ekey biometrical access systems (RS485)
- HVAC systems can be controller through RS232 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
- ..

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.

## Quick startup guide

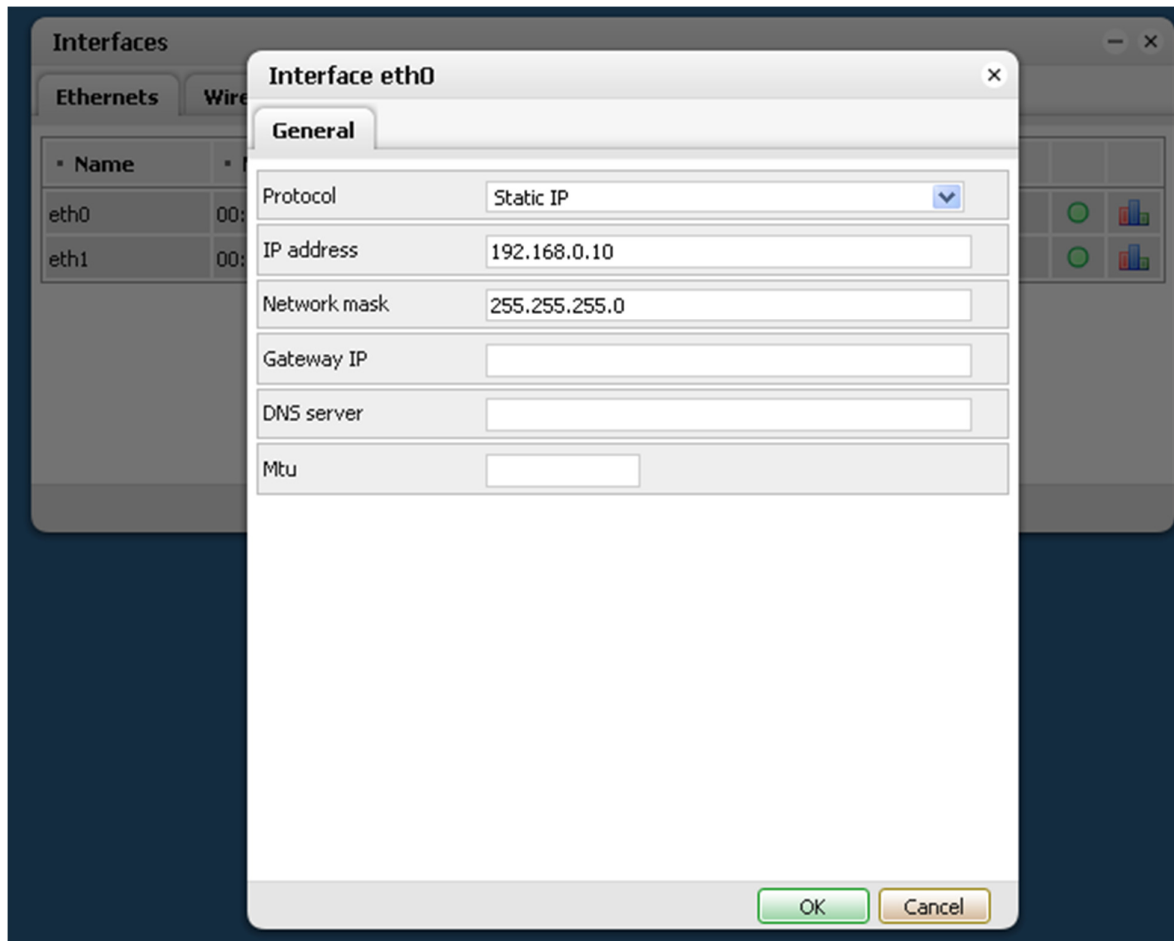
- 1) Mount the device on DIN rail
- 2) Connect the KNX bus cable
- 3) Connect 24V power supply to the device (red pole to 24V+, grey pole to GND)
- 4) Connect Ethernet cable coming from the PC

### Default IP configuration

<i>Logic Machine/System Configuration</i> Login name	<b>admin</b>
<i>Logic Machine/System Configuration</i> Password	<b>admin</b>
<i>User mode visualization/Touch visualization</i> Login name	Read-only: <b>visview</b>  Write: <b>viscontrol</b>  Write + admin level: <b>visadmin</b>
<i>User mode visualization/Touch visualization</i> Password	Read-only: <b>visview</b>  Write: <b>viscontrol</b>  Write + admin level: <b>visadmin</b>
IP address on LAN	<b>192.168.0.10</b>
Networks mask on LAN	255.255.255.0

### Change IP settings

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



- **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

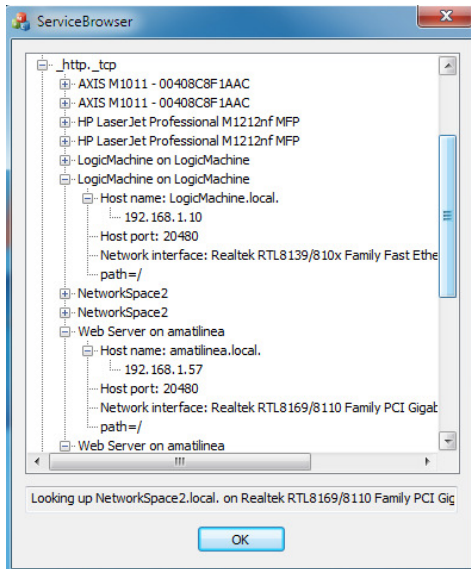
When changes are done, the following icon appears in the top-right corner. This should be applied changes to take effect.

Apply changes

## Discover LogicMachine IP address

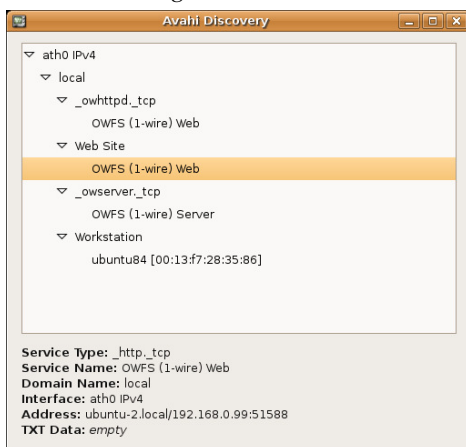
### Windows PC

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](http://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.grokkt.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.



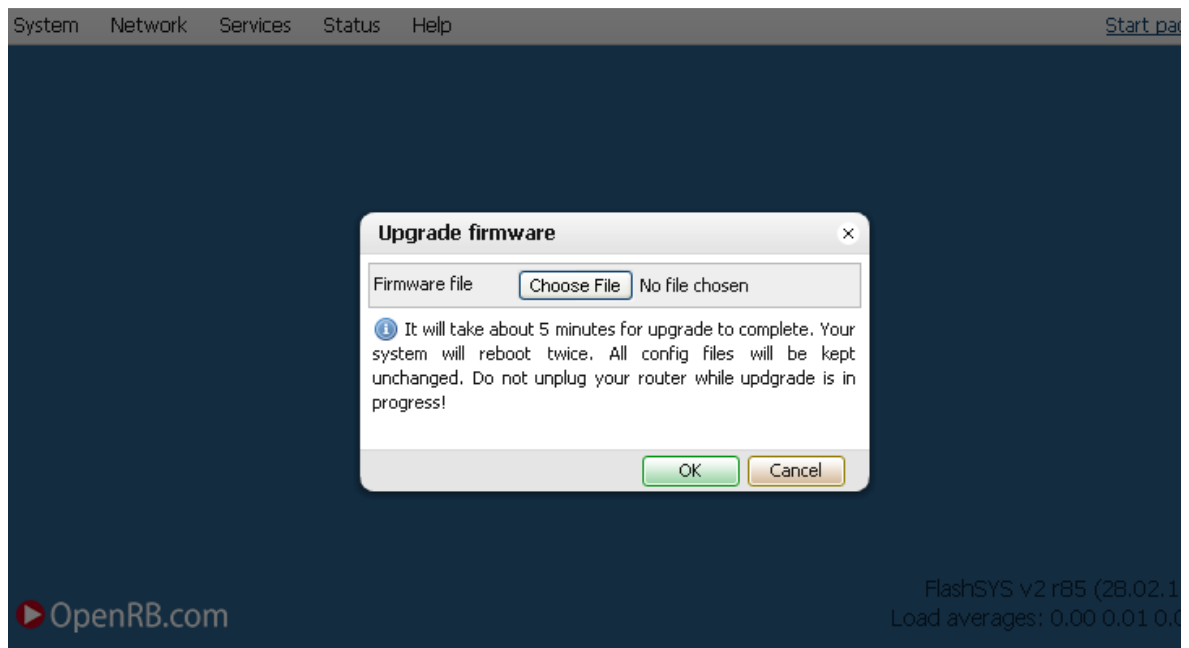
## **Firmware upgrade**

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

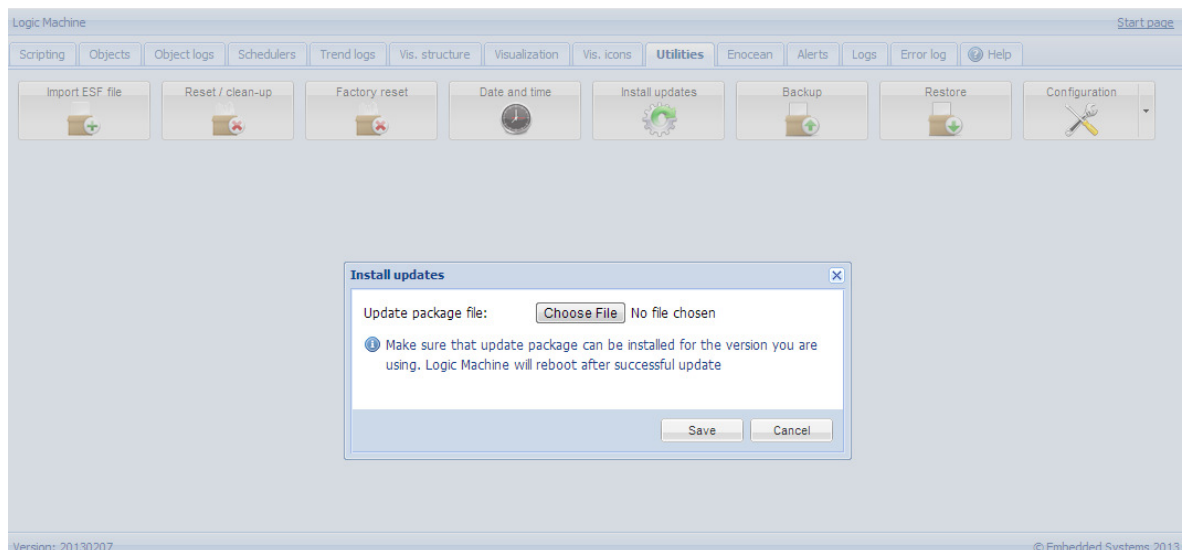
**Note!** After each upgrade, we strongly recommend to clean your browser cache.

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](http://www.openrb.com).

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



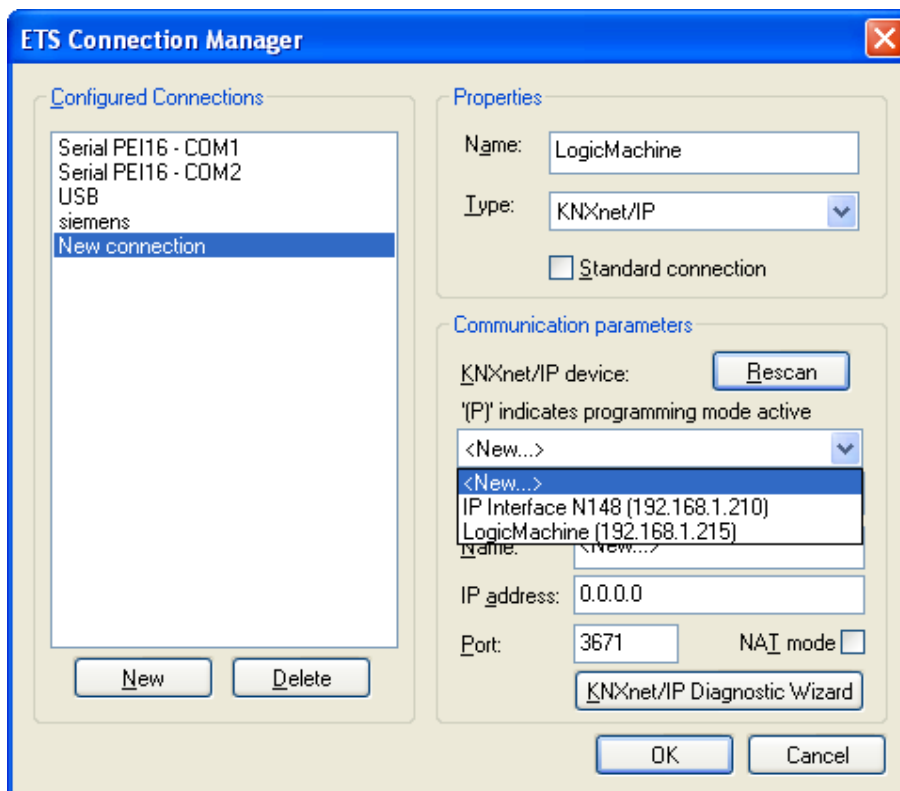
LogicMachine visualization upgrade or patch installation can be done in *Utilities* tab and press on *Install updates* icon. After \*.LMU file is chosen from the corresponding location press *Save* button. The device will be rebooted after 5 seconds and new firmware will be installed.



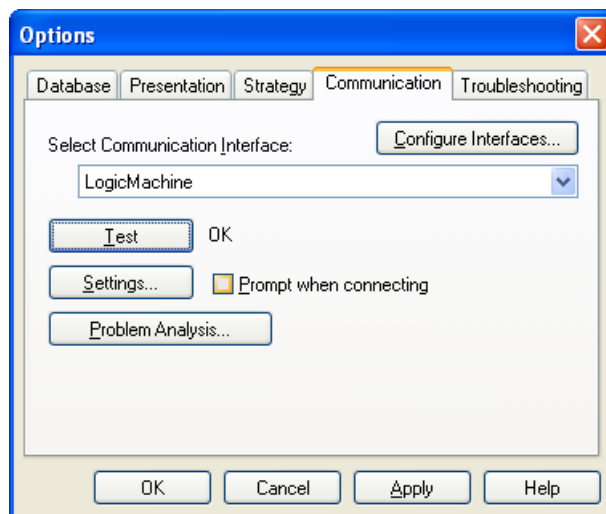
## LogicMachine for KNX/EIB network configuration management with ETS

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.

## 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 ☐

KNX address 15.15.255

KNX IP features ☒

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. LogicMachine3 Reactor 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**

General IP > TP filter TP > IP filter

Apply filter to tunneling ☐

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 1/1/1-1/1/2

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** [X]

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

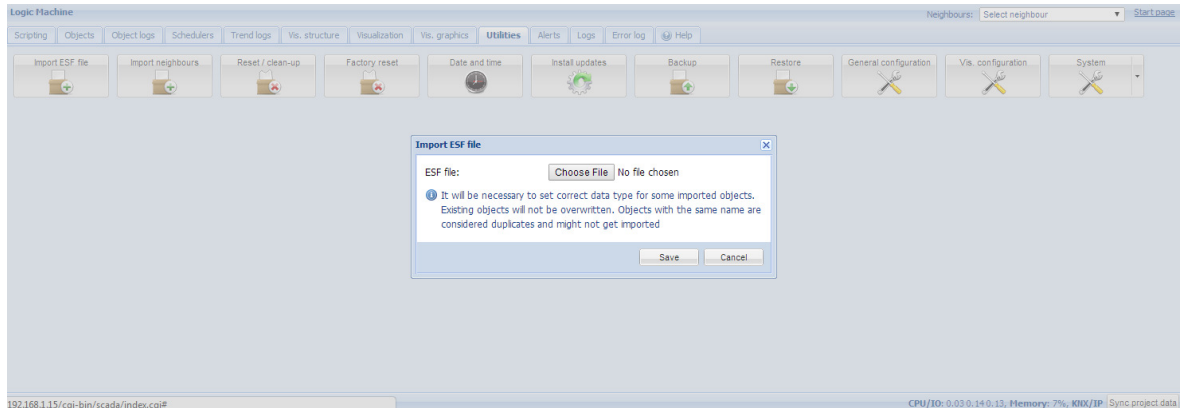
- ***Apply filter to virtual objects*** – either to apply filter policy to objects added in Objects tab as virtual objects without attraction to bus
- ***SRC policy*** [*No filter / Accept selected individual addresses / Drop selected individual addresses*]- policy to apply to the list of source individual 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.

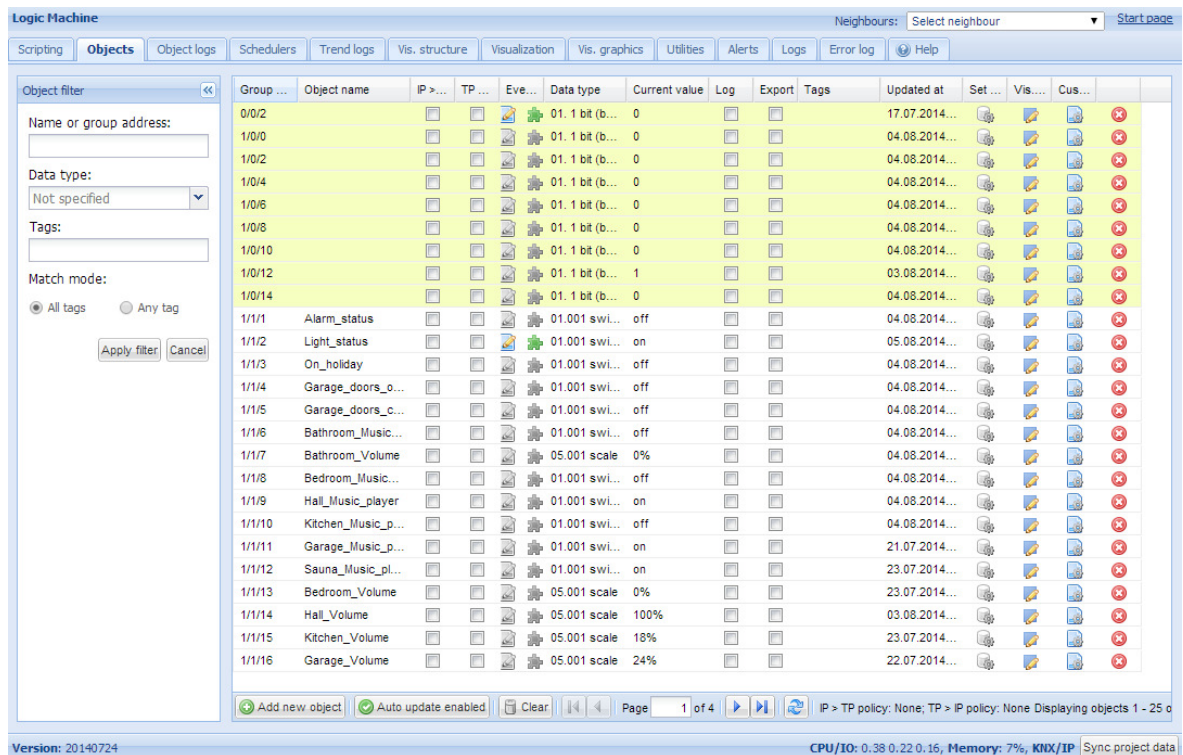
# Quick guide - create visualization for iPad/PC

## 1. Import objects

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



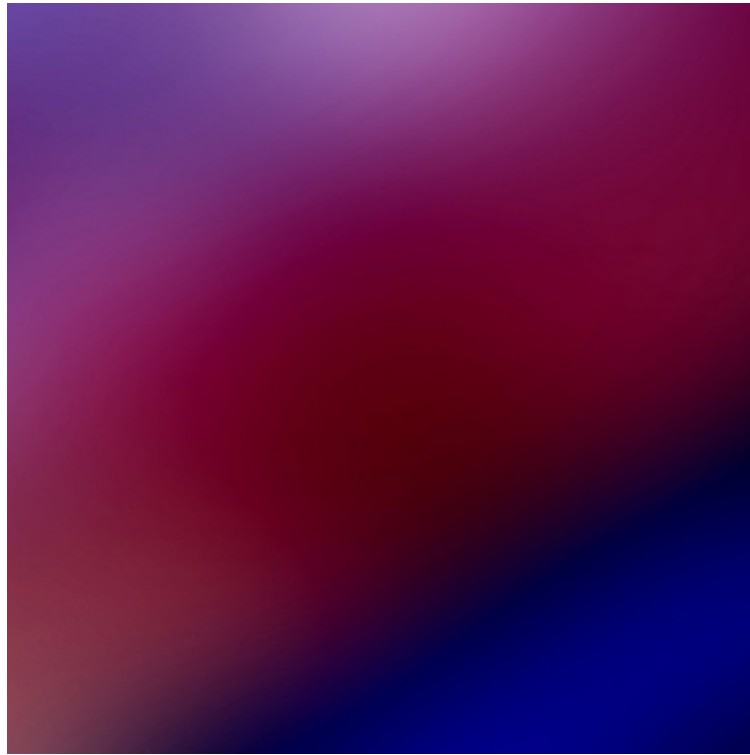
Or connect LM to the bus and it will detect objects automatically (in yellow) in *Objects* tab once they are activated. Objects can be added manually as well.



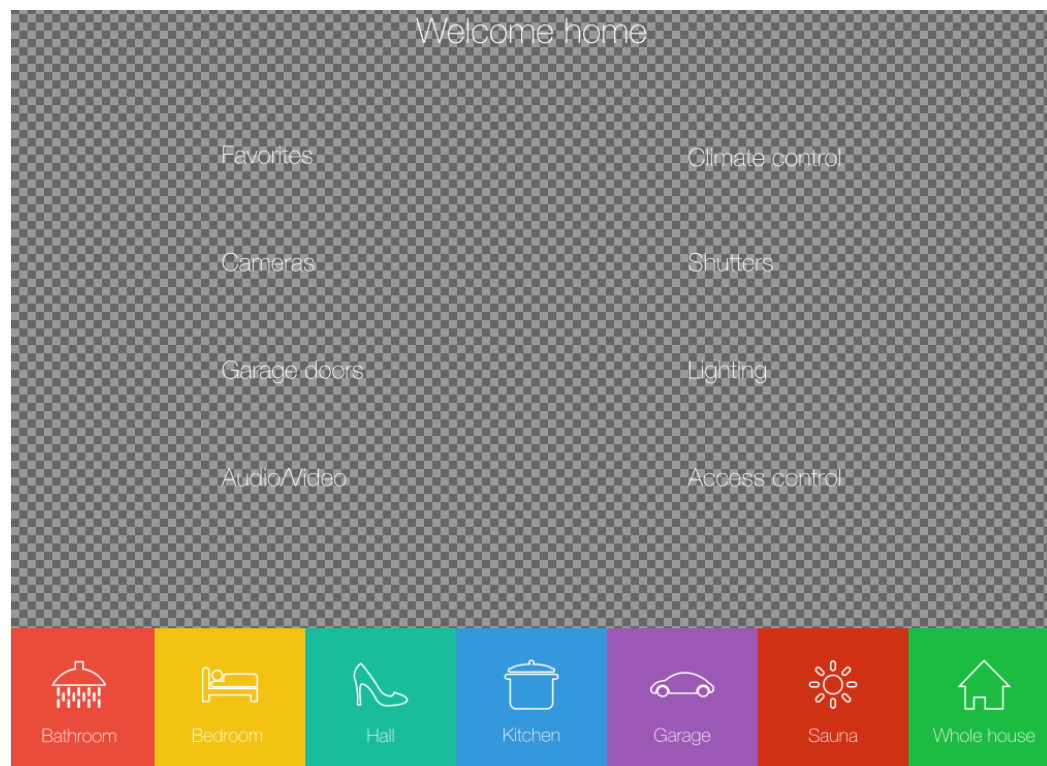
## 2. Prepare graphics

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)

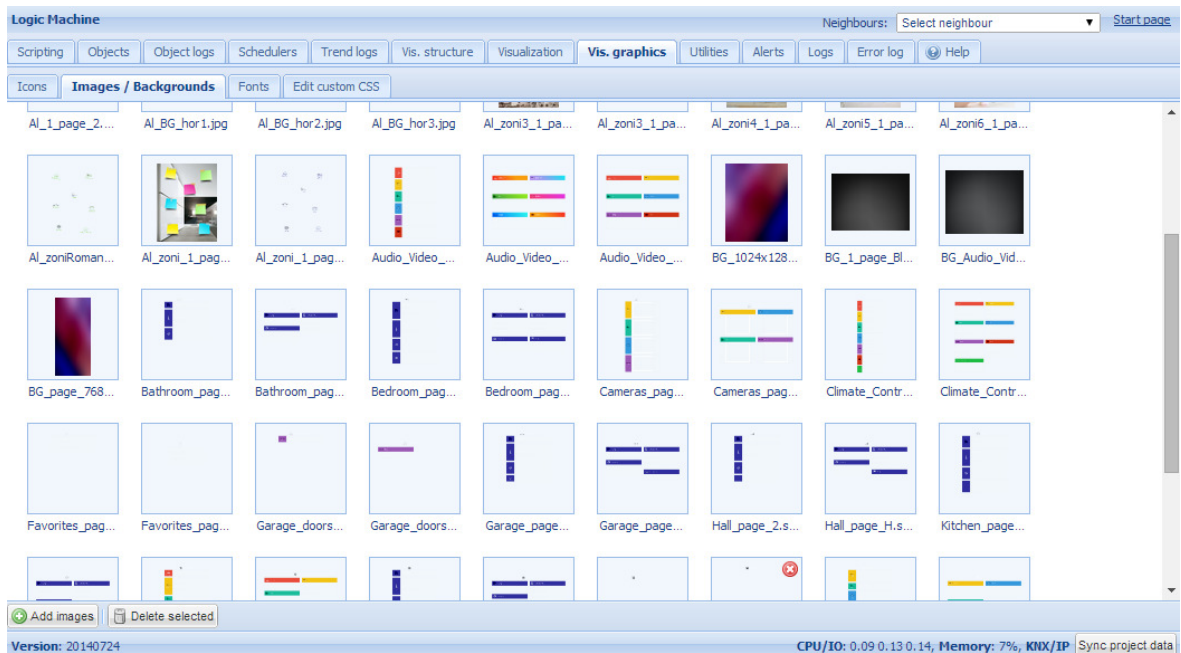
- a) basic background which can be changed by necessity



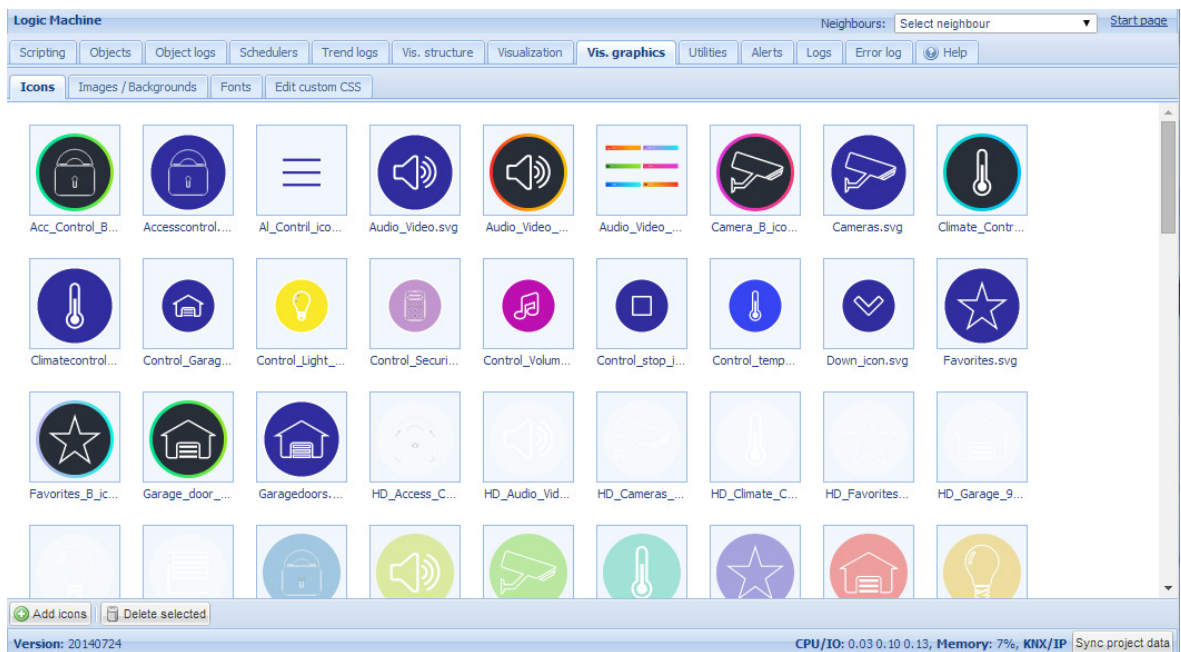
- b) foreground which will stay unchanged




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

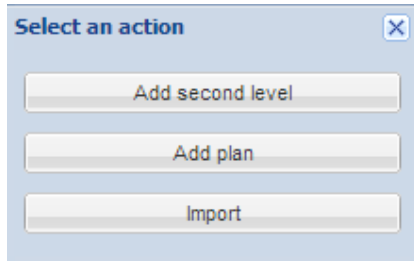


Prepare set of icons (preferably in SVG form) and add them in *Logic Machine* → *Vis. Graphics* → *Icons*. Or you can use icons predefined in LogicMachine by default.

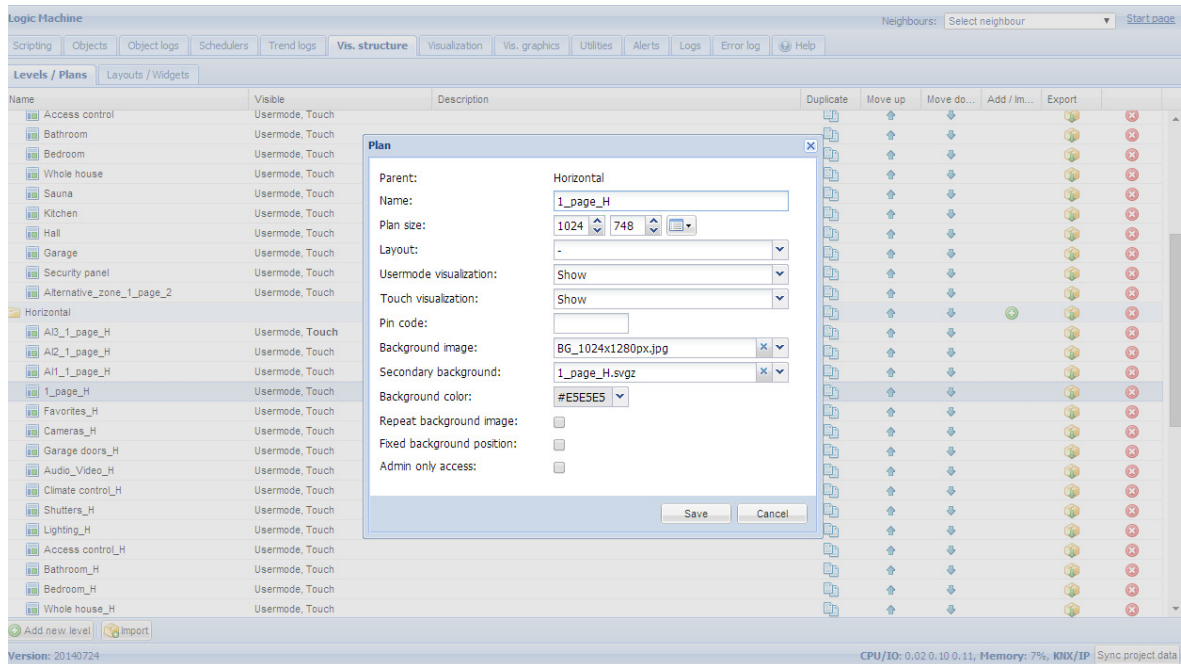


## Create “floor” structure and add objects to the map

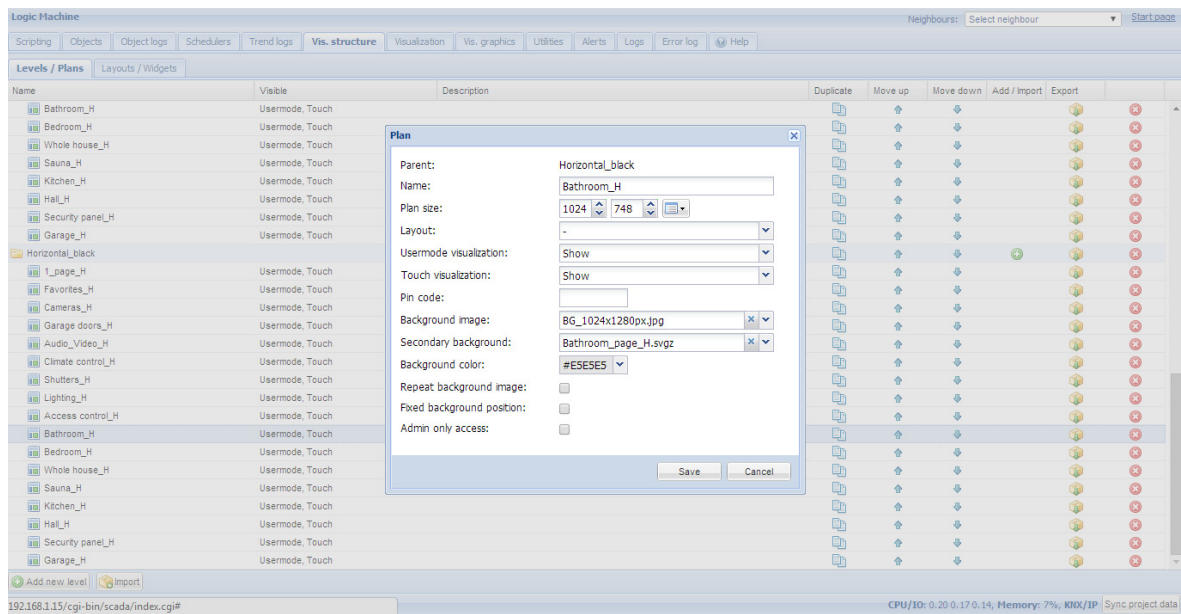
In *Logic Machine* → *Vis.structure* menu the structure of the visualization is defined and visualization backgrounds are uploaded. Use icon to add  floor.



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.

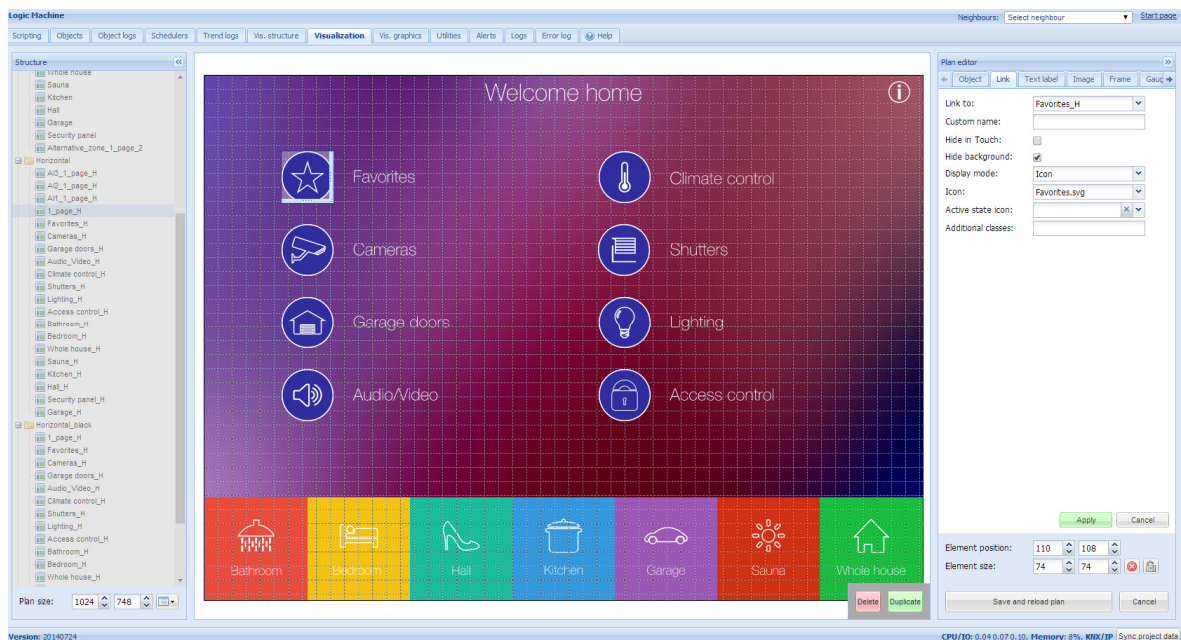






### Add objects to newly created visualization map

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.

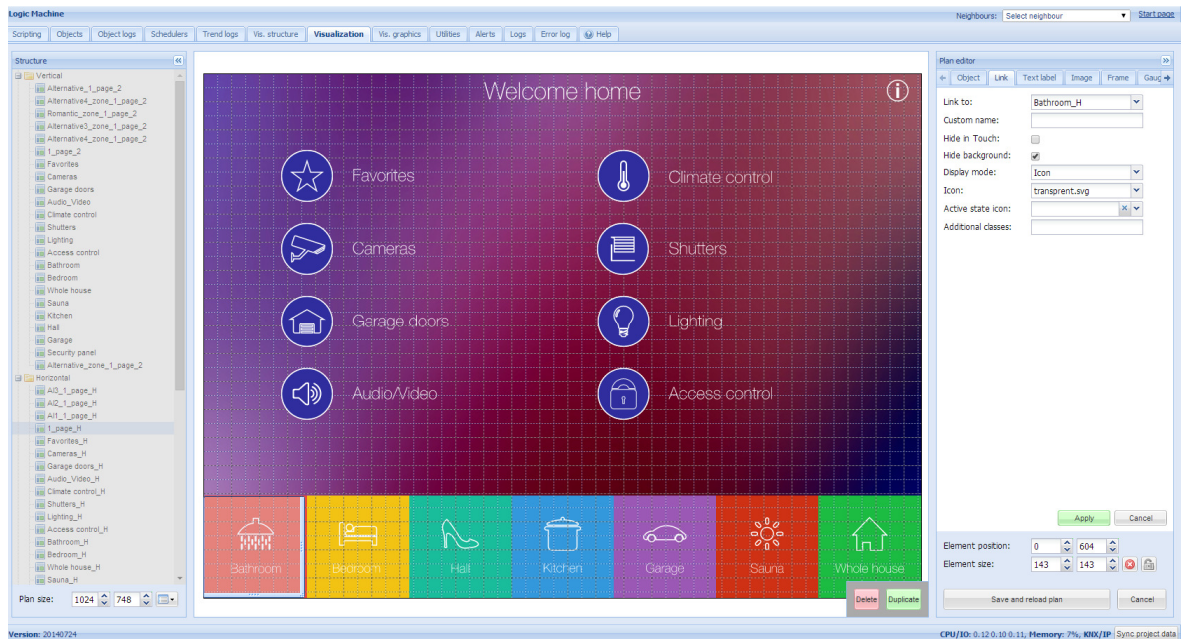


Objects can be added to the map by clicking on *Unlock current floor plan for editing* button. In this example we are creating first page of visualization which will link to other Floors with



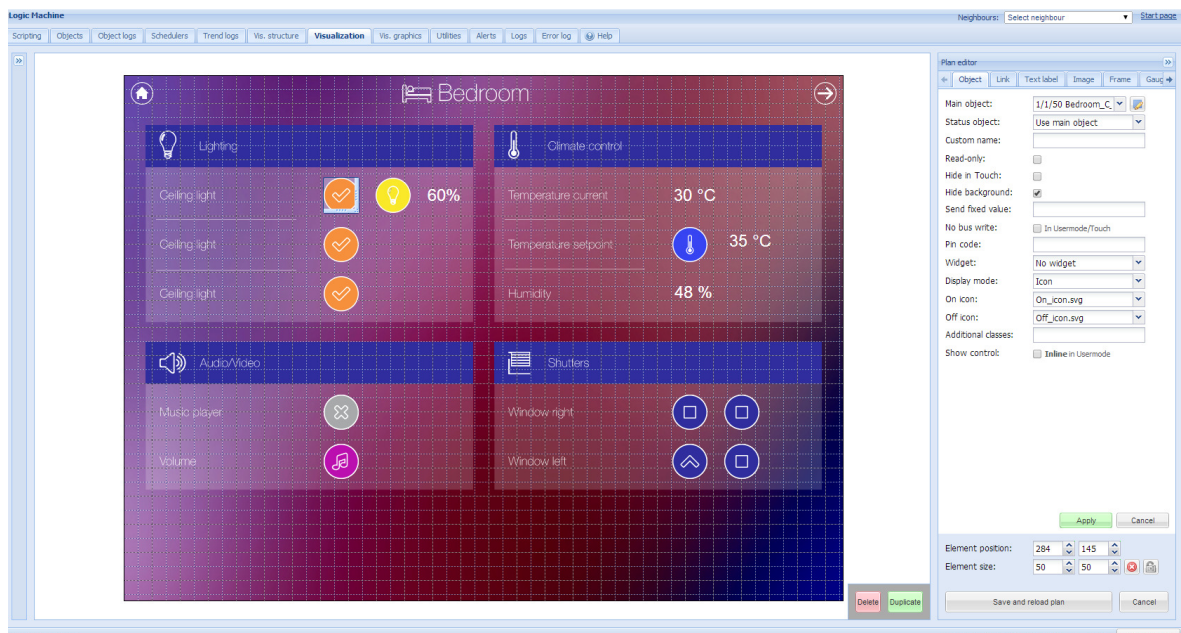
specific object control. Add link by clicking on Link tab, choosing specific icon, scale it and place in desired location.

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.

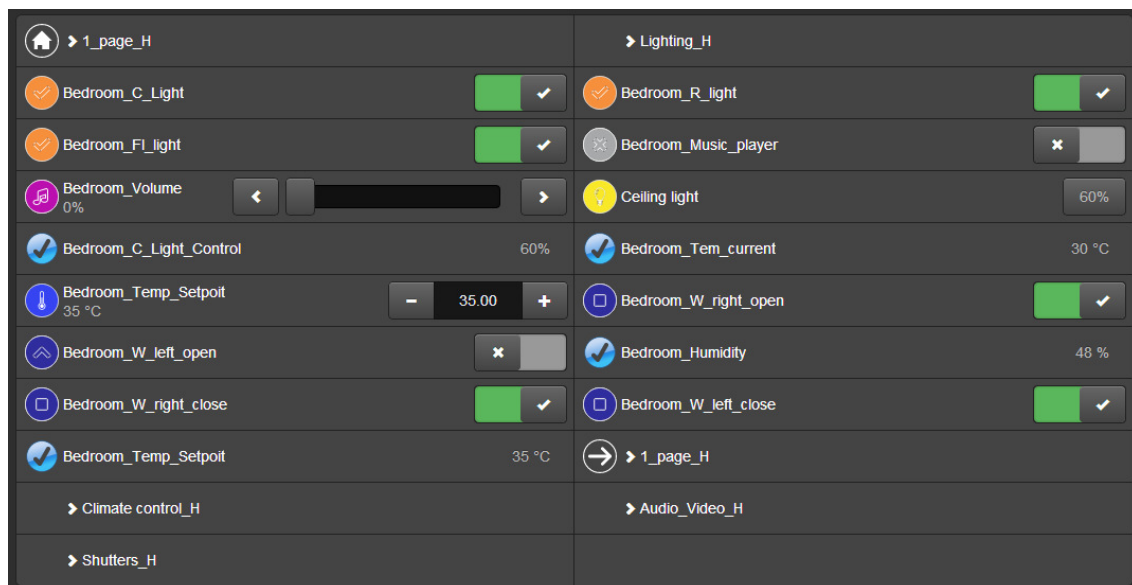


### Launching visualization on touch device (iPad in this case)

- 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.

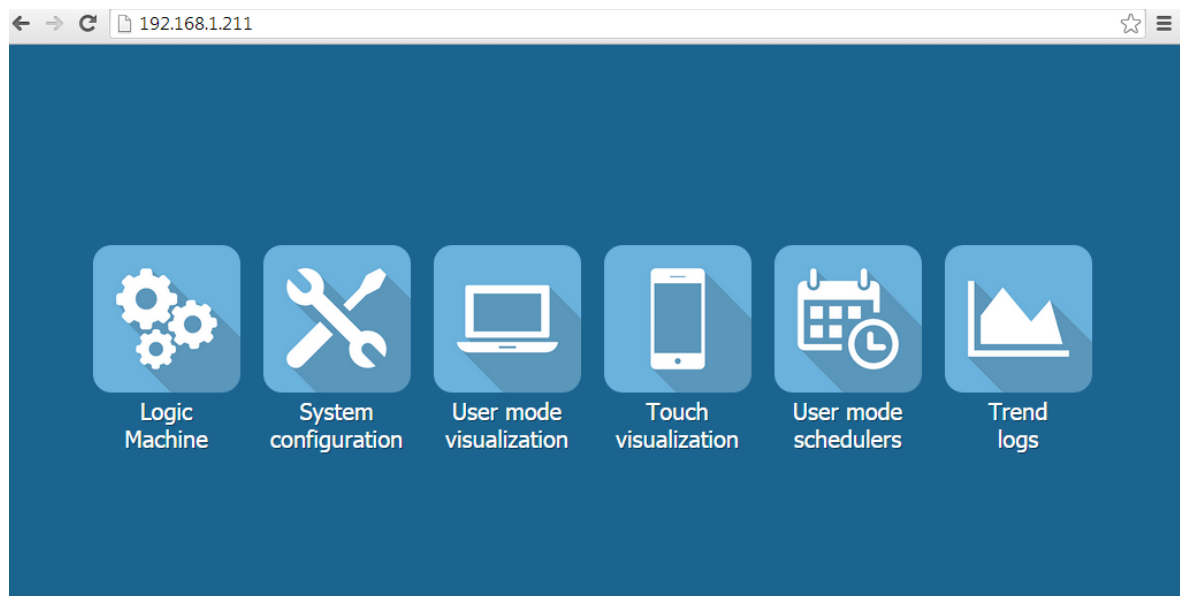


## Graphical User Interface Login

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
- **Network configuration** – IP and KNXnet/IP specific configuration
- **User mode visualization** – defined visualization maps with objects
- **Touch visualization** – Visualization system for iPhone/iPod/iPad/Android touch screen devices
- **User mode schedulers** – User defined schedulers
- **Trend logs** – Trends for data logs

## 1. 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
- **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
- **BACnet** – BACnet client with scanner
- **Modbus** – Modbus mapper
- **Enocean** – EnOcean mapper
- **1-wire** – 1-wire object mapping to KNX
- **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

## 1.2. 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!** Data format — in most cases data is stored and transferred between LogicMachine parts using hex-encoded strings (2 bytes per 1 byte of data).*

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

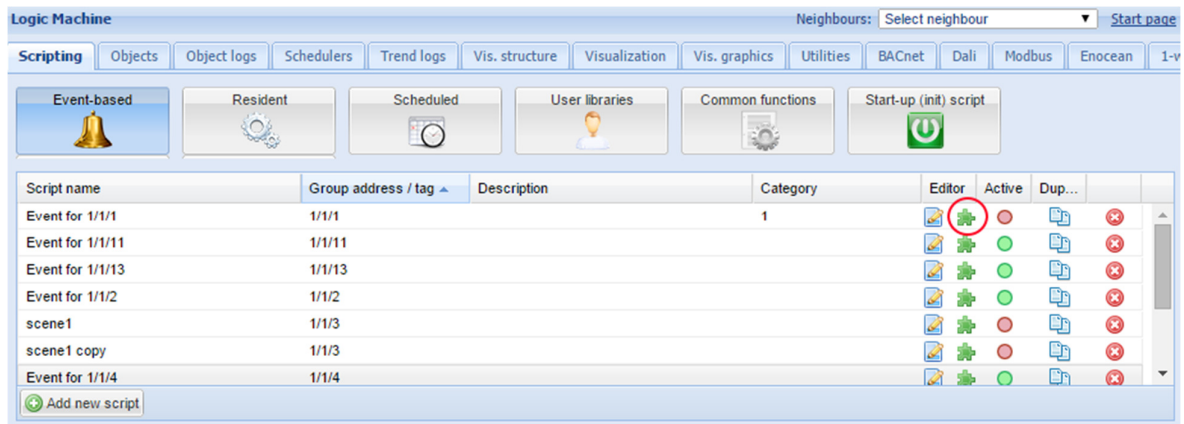
**Common functions** – common functions to call from other scripts

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

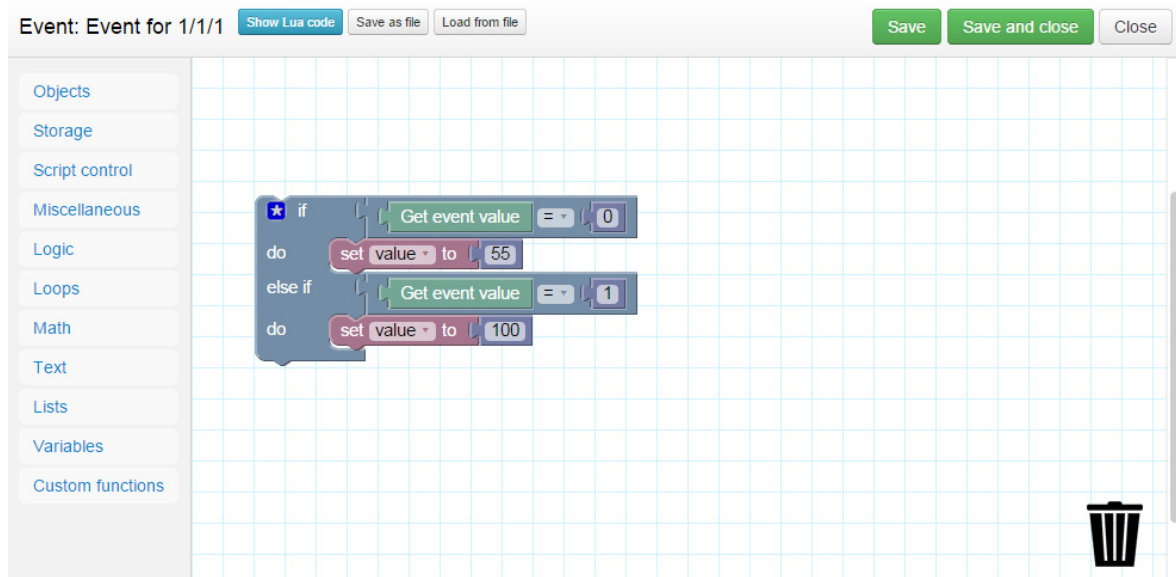
### 1.1.1. Block programming

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

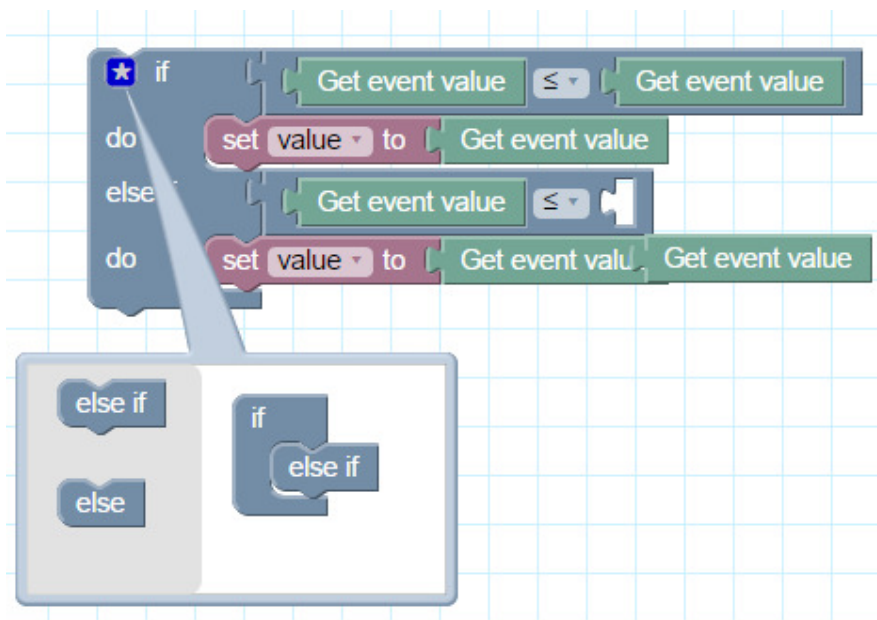
Once the script is added, you can see puzzle icon to access Block editor.



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)

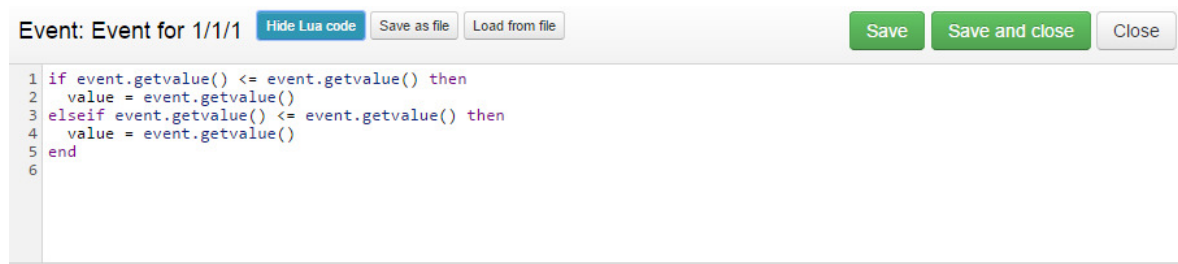


Press Delete button or drag the block to the garbage if you want to delete it



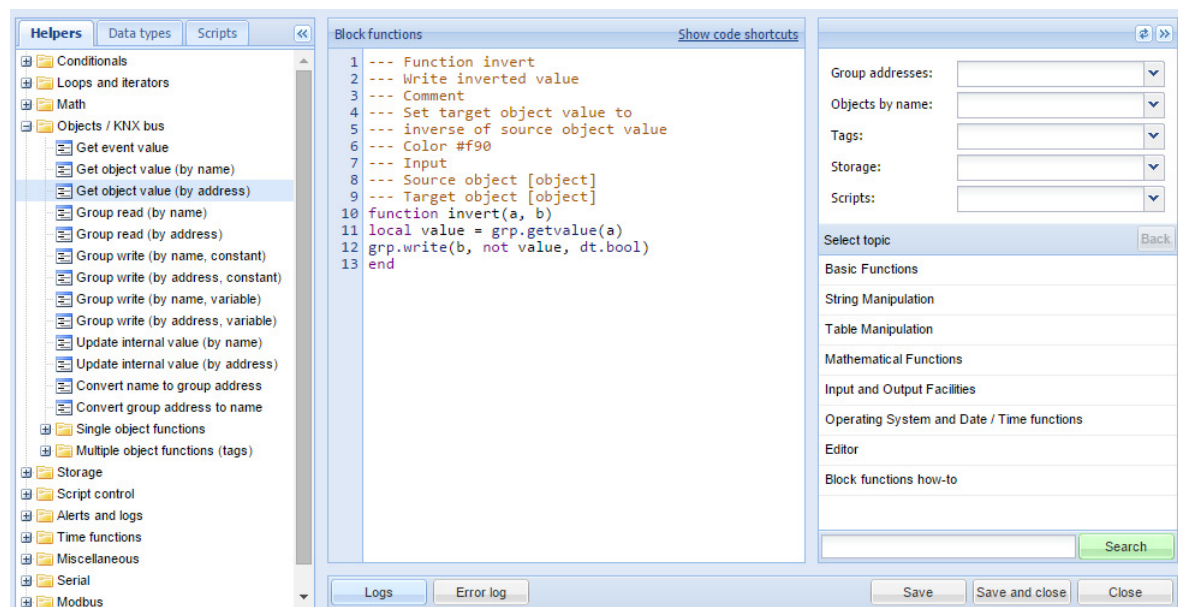


You can always look at the LUA code by clicking on *Show/Hide Lua code* button. This will allow to learn the scripting language.



### 1.1.2. 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:

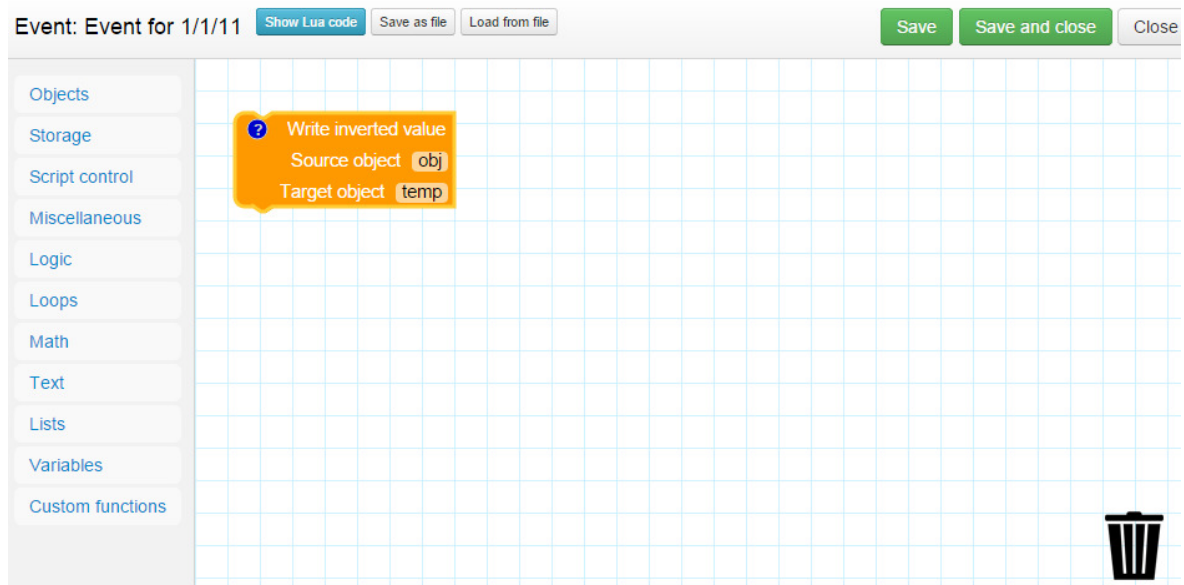
Inputs are a function parameter, other blocks can be connected to inputs by default. If input definition has **[object]**, **[storage]** or **[tag]** in its name then the input is replaced with object, storage or tag selection input.

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
```

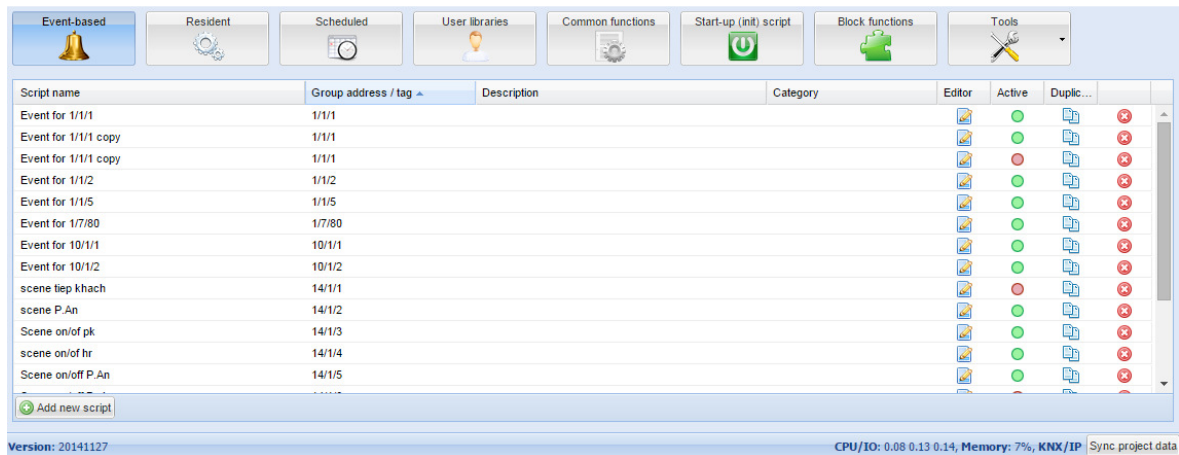
Once block function is added, it is available as a block in Block editor.





### 1.1.3. 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

- **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:

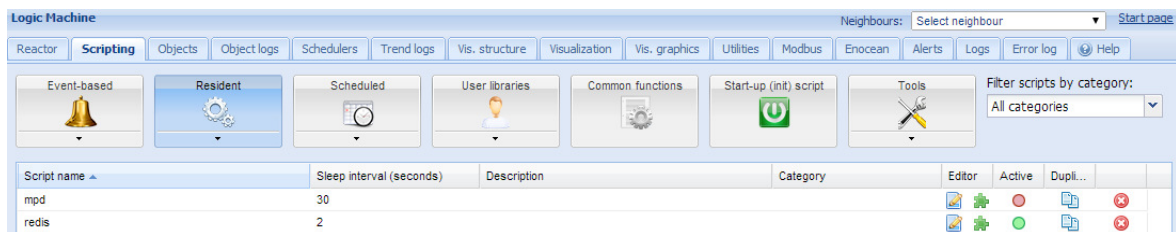
Active: ☐

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.

#### **1.1.4. 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)**

### 1.2.3. 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.

### 1.2.4. 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.

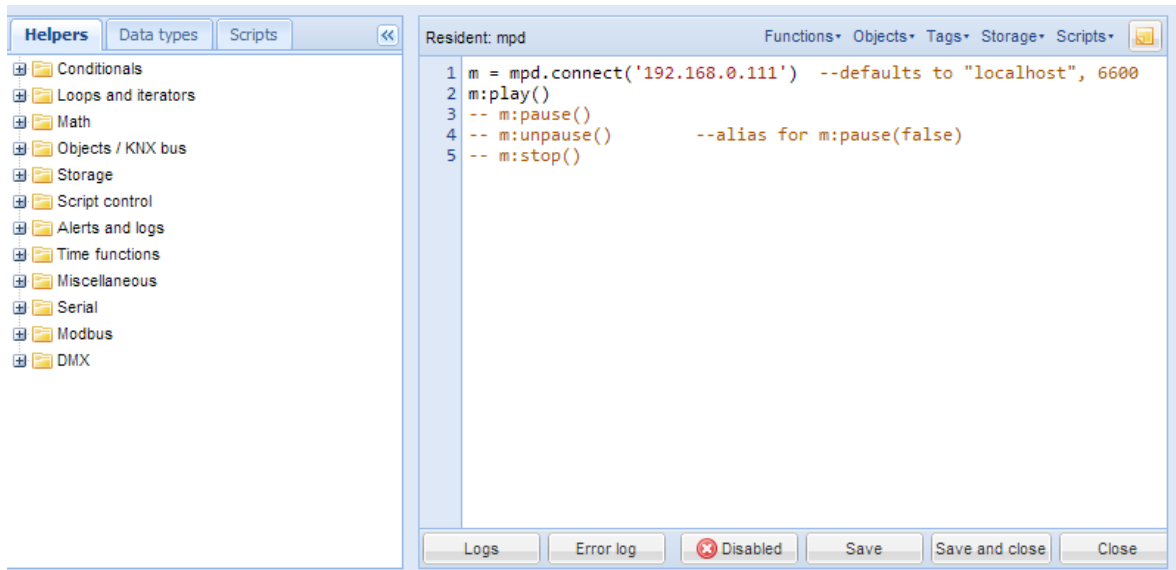
**N** — execute script exactly at N minute, hour or day.

**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

### 1.2.5. 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..Until etc.

*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.

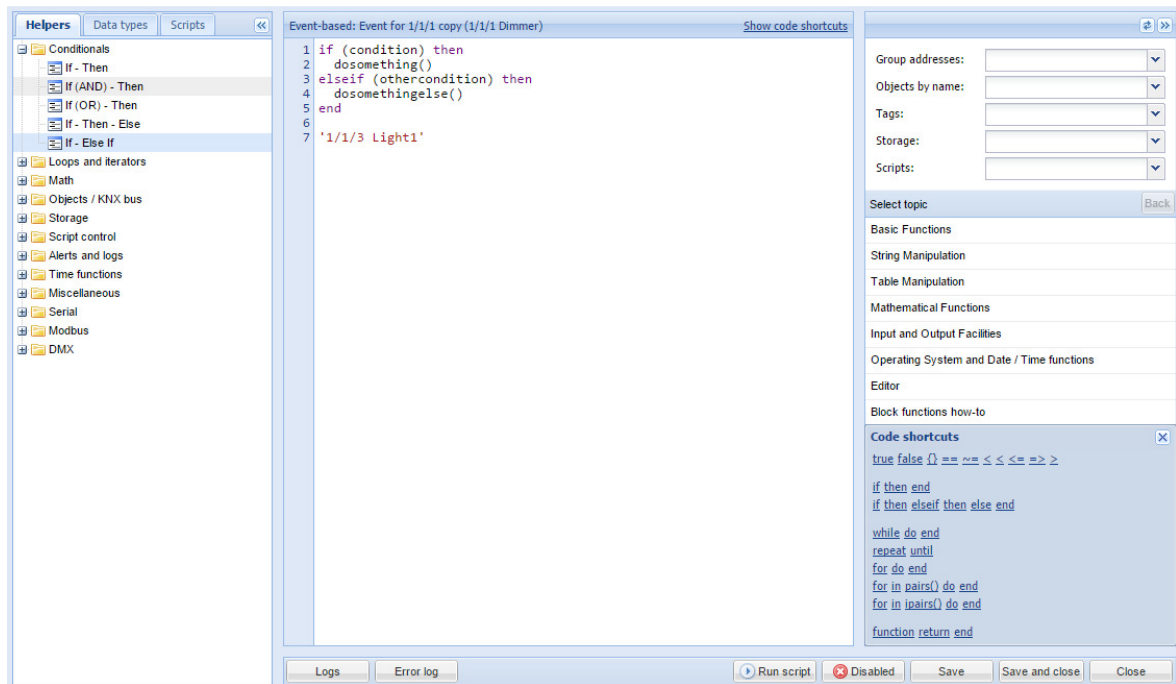
*DMX* – Communication with DMX devices

**Data types** – choose object by data type

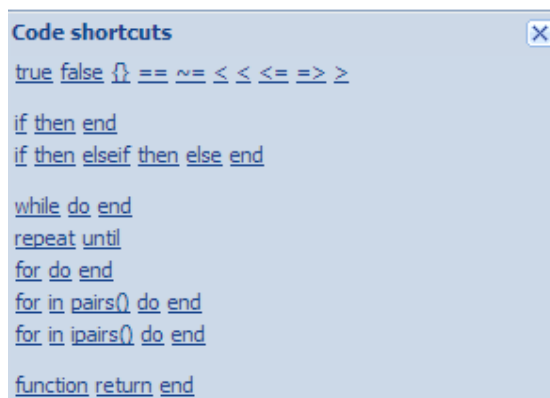
**Scripts** – list of all scripts added in the LogicMachine

## Code helpers on the right side of the editor

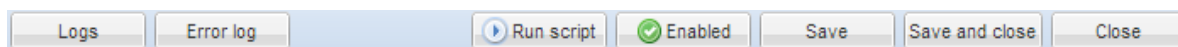
There is a special section in scripting editor which allows quickly find functions, objects or tags by name and storage variables.



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.



### **1.2.6. 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.

### **1.2.7. 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.

## **1.2.8. 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.

## **1.2.9. 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.ifmyobjectthen
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.ifmyobjectthen
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
```

### 1.2.10. 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

### 1.2.11. 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

### 1.2.12. 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.ifttype(objectdata)=='table'then
3.ifobjectdata.temperature> 24 then
4.-- do something if temperature level is too high
5.end
6.end
```

### 1.2.13. 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
```

### 1.2.14. 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)
```

### **1.2.15.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

### **1.2.16.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)

### **1.2.17.Data Serialization**

#### *serialize.encode (value)*

Generates a storable representation of a value.

#### *serialize.decode (value)*

Creates a Lua value from a stored representation.

### 1.2.18.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\  
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 substring (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.

## ***1.2.19. 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 in io.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
```

### 1.2.20. 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

### 1.2.21. 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.

### 1.2.22. Conversion

Compatibility layer: *lmcore* 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.

### **1.2.23.Bit operators**

*bit.bnot (value)*

Binary not

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

Binary and between any number of variables

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

Binary or between any number of variables

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

Binary xor between any number of variables

*bit.lshift (value, shift)*

Left binary shift

*bit.rshift (value, shift)*

Right binary shift

### 1.2.24. 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.

## **1.2.25. 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.

## **1.2.26. Table manipulations**

This library provides generic functions for table manipulation. It provides all its functions inside the table. 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.

## **1.2.27. 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.



## 1.2.28. 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 in pairs(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)*

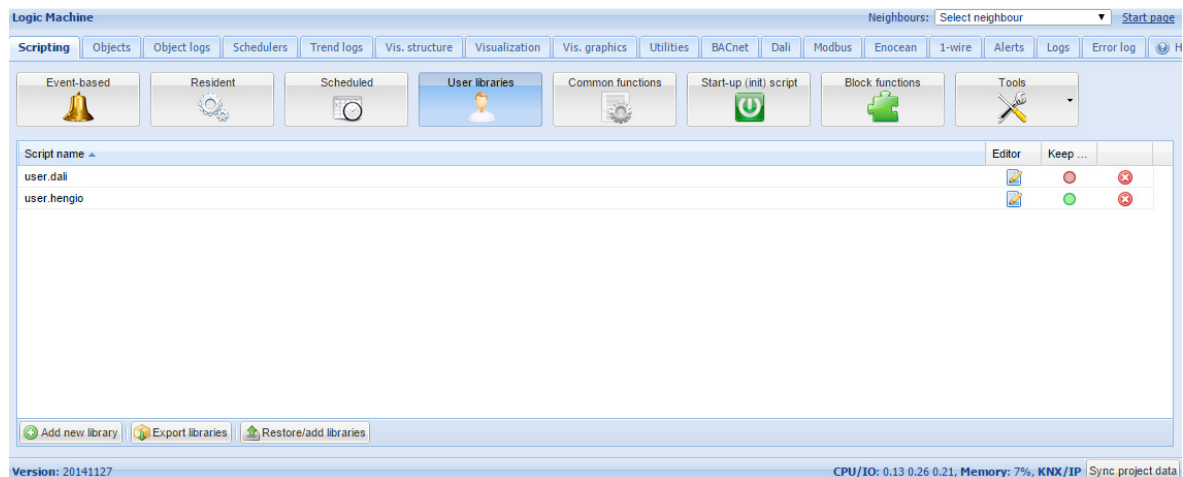
Receives an argument of any type and converts it to a string in a reasonable format. For complete control of how numbers are converted, use *string.format*.

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".

## **1.2.29. User libraries**

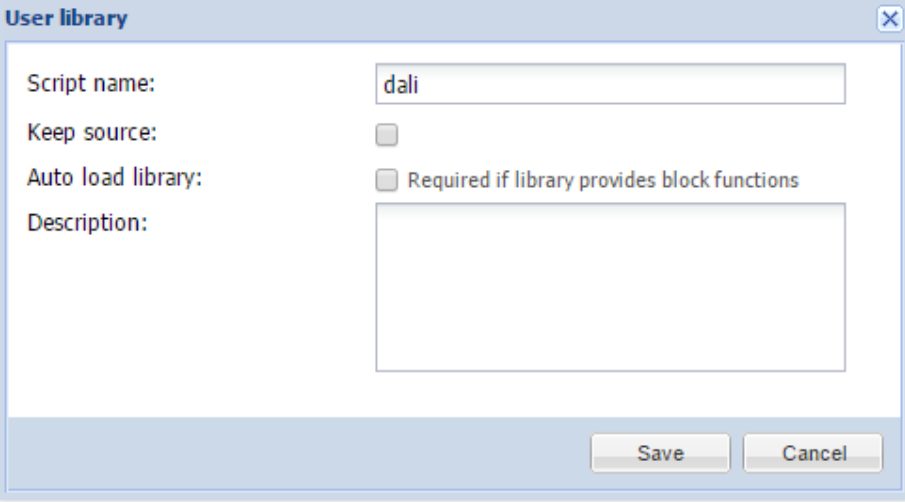


User libraries usually contain user defined functions which are later called from other scripts.

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.



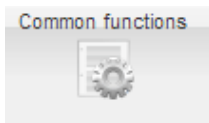
The 'User library' dialog box contains the following fields and controls:

- Script name:** A text input field containing the value 'dali'.
- Keep source:** An unchecked checkbox.
- Auto load library:** An unchecked checkbox with the label 'Required if library provides block functions' to its right.
- Description:** A large, empty text area.
- Buttons:** 'Save' and 'Cancel' buttons at the bottom right.

*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.

### 1.2.30. 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*.

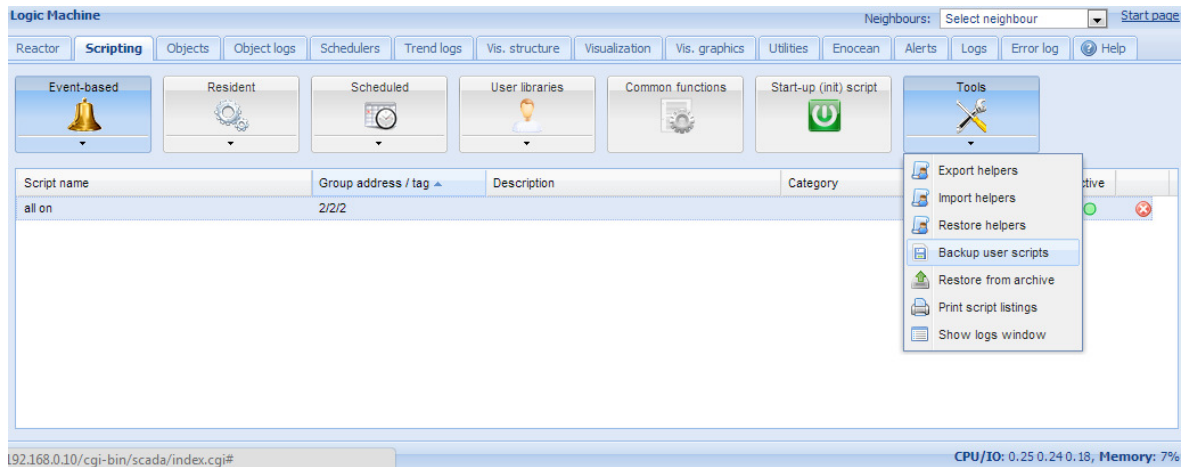


### 1.2.31. Start-up (init) script

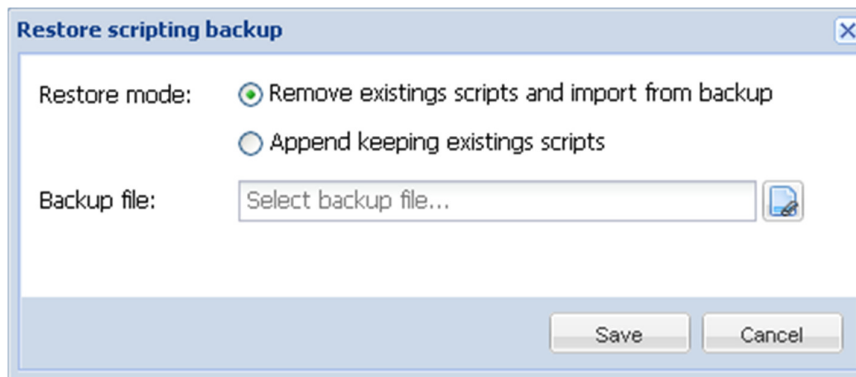
Init script is used for initialization on specific system or bus values on system start. Init script is run each time after system is restarted for some reason.



## 1.2.32.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))

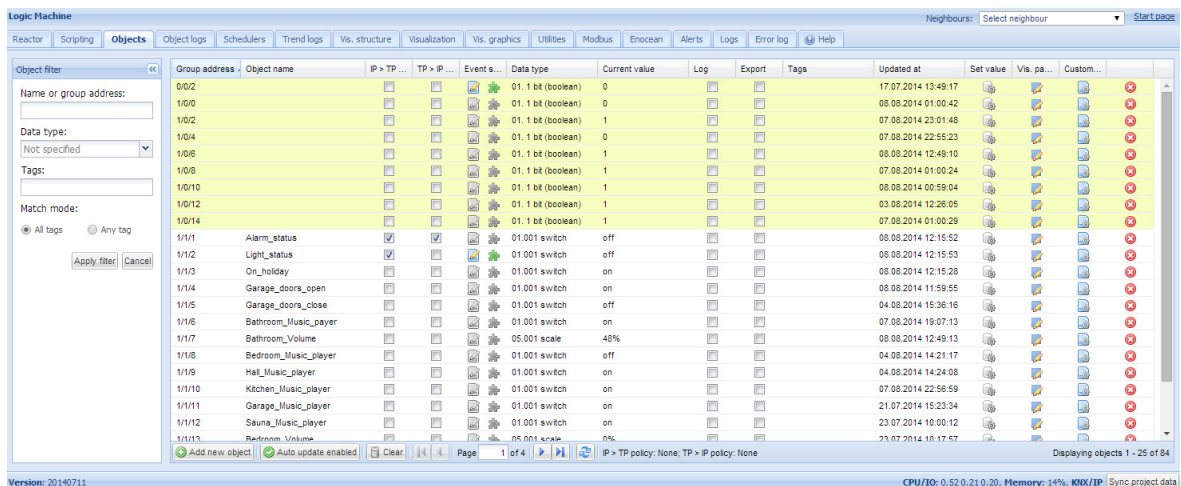
  -- update 0/0/2 with the same data
  knxobject.write('0/0/2', presence.data, dt.bool)
else
  alert('read error')
end
```

➤ *Show logs window* – show logs in separate window

## 1.3. 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*)



The screenshot shows the 'Objects' window in the Logic Machine software. It displays a table of KNX network objects with columns for Group address, Object name, IP > TP, TP > IP, Event s..., Data type, Current value, Log, Export, Tags, Updated at, Set value, Vis. pa..., and Custom... The table lists various objects such as Alarm\_status, Light\_status, On\_holiday, Garage\_doors\_open, Garage\_doors\_close, Bathroom\_Music\_player, Bathroom\_Volume, Bedroom\_Music\_player, Hall\_Music\_player, Kitchen\_Music\_player, Garage\_Music\_player, Sauna\_Music\_player, and Bedroom\_Volume. The status bar at the bottom indicates 'Displaying objects 1 - 25 of 84'.

Group address	Object name	IP > TP	TP > IP	Event s...	Data type	Current value	Log	Export	Tags	Updated at	Set value	Vis. pa...	Custom...
0/0/2					01.1 bit (boolean)	0				17.07.2014 13:49:17			
1/0/0					01.1 bit (boolean)	0				08.08.2014 01:00:42			
1/0/2					01.1 bit (boolean)	1				07.08.2014 23:01:48			
1/0/4					01.1 bit (boolean)	0				07.08.2014 22:55:23			
1/0/6					01.1 bit (boolean)	1				08.08.2014 12:49:10			
1/0/8					01.1 bit (boolean)	1				07.08.2014 01:00:24			
1/0/10					01.1 bit (boolean)	1				08.08.2014 00:59:04			
1/0/12					01.1 bit (boolean)	1				03.08.2014 12:26:05			
1/0/14					01.1 bit (boolean)	1				07.08.2014 01:00:29			
1/1/1	Alarm_status				01.001 switch	off				08.08.2014 12:15:52			
1/1/2	Light_status				01.001 switch	off				08.08.2014 12:15:53			
1/1/3	On_holiday				01.001 switch	on				08.08.2014 12:15:28			
1/1/4	Garage_doors_open				01.001 switch	on				08.08.2014 11:59:55			
1/1/5	Garage_doors_close				01.001 switch	off				04.08.2014 15:36:16			
1/1/6	Bathroom_Music_player				01.001 switch	on				07.08.2014 19:07:13			
1/1/7	Bathroom_Volume				05.001 scale	48%				08.08.2014 12:49:13			
1/1/8	Bedroom_Music_player				01.001 switch	off				04.08.2014 14:21:17			
1/1/9	Hall_Music_player				01.001 switch	on				04.08.2014 14:24:08			
1/1/10	Kitchen_Music_player				01.001 switch	on				07.08.2014 22:56:59			
1/1/11	Garage_Music_player				01.001 switch	on				21.07.2014 15:23:34			
1/1/12	Sauna_Music_player				01.001 switch	on				23.07.2014 10:00:12			
1/1/13	Bedroom_Volume				05.001 scale	0%				21.07.2014 10:17:57			

### 1.3.1. Object parameters

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


The screenshot shows a software window titled "Edit object". It contains the following fields and controls:

- Object name:** A text box containing "Temp room".
- Group address:** A text box containing "1/3/1".
- Data type:** A dropdown menu with "09.001 Temperature" selected.
- Current value:** A text box containing "0".
- Tags:** An empty text box.
- Units / suffix:** A text box containing "C".
- Log:** A checked checkbox.
- High priority log:** An unchecked checkbox.
- Export:** An unchecked checkbox.
- Poll interval (seconds):** A numeric input field with up and down arrow buttons.
- Object comments:** A large, empty text area.
- Buttons:** "Save" and "Cancel" buttons at the bottom right.

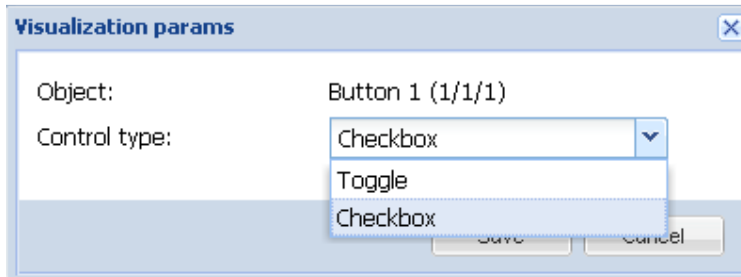
- **Object name** – Name for the object
- **Group address** – Group address of this object
- **Data type** – KNX data type for the object. This has to be set once the LM sniffs the new object for proper work.
- **Units / suffix** – units for the object which will appear on the visualization along with 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

### 1.3.2. 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

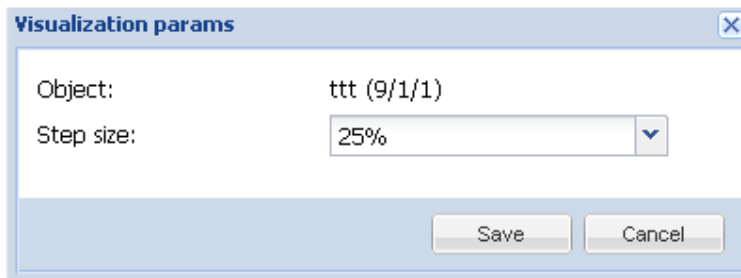


- **Control type** – type of the visual control element

- Toggle
- Checkbox



#### 4 bit (3 bit controlled)



- **Step size** – step size for example for blinds control

2 bit (1 bit controlled), 1 byte unsigned integer (scale), 1 byte signed integer, 2 byte unsigned integer, 2 byte signed integer, 2 byte floating point (temperature), 4 byte unsigned integer, 4 byte signed integer, 4 byte floating point

#### 1byte

**Object visualization parameters** [X]

Object: rrr (2/1/3)

Control type: Slider [v]

Minimum value: 0 [up/down]

Maximum value: 100 [up/down]

Step: 5 [up/down]

[Save] [Cancel]

- Control type – type of the visual control element
  - Slider


ttt 17 [Slider bar with green knob]

○ Direct input / Step +/-

ttt 12 [- 12 +]

- Minimum value
- Maximum value

### 1.3.3. *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.



Set object value

Object name: Weather T High

Group address: 5/1/5

Data type: 09. 2 byte floating point

New value (21):

Save Cancel

Set object value

Object name: Output 1


Group address: 1/2/1

Data type: 01.001 switch

New value:


Save Cancel


### 1.3.4. Custom values

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

Custom values

Default text:

Object value:  Display text:  

Object value:  Display text:  

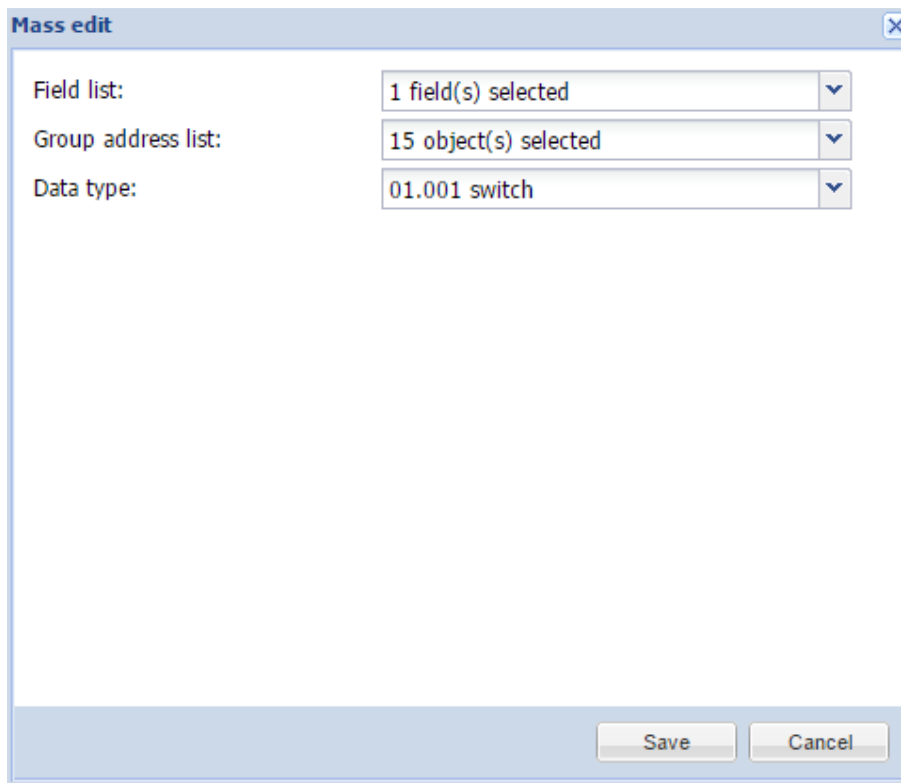
Save Cancel

### 1.3.5. Object control bar

 Add new object
  Auto update enabled
  Clear
  Mass edit
 Page  of 1
 



- **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



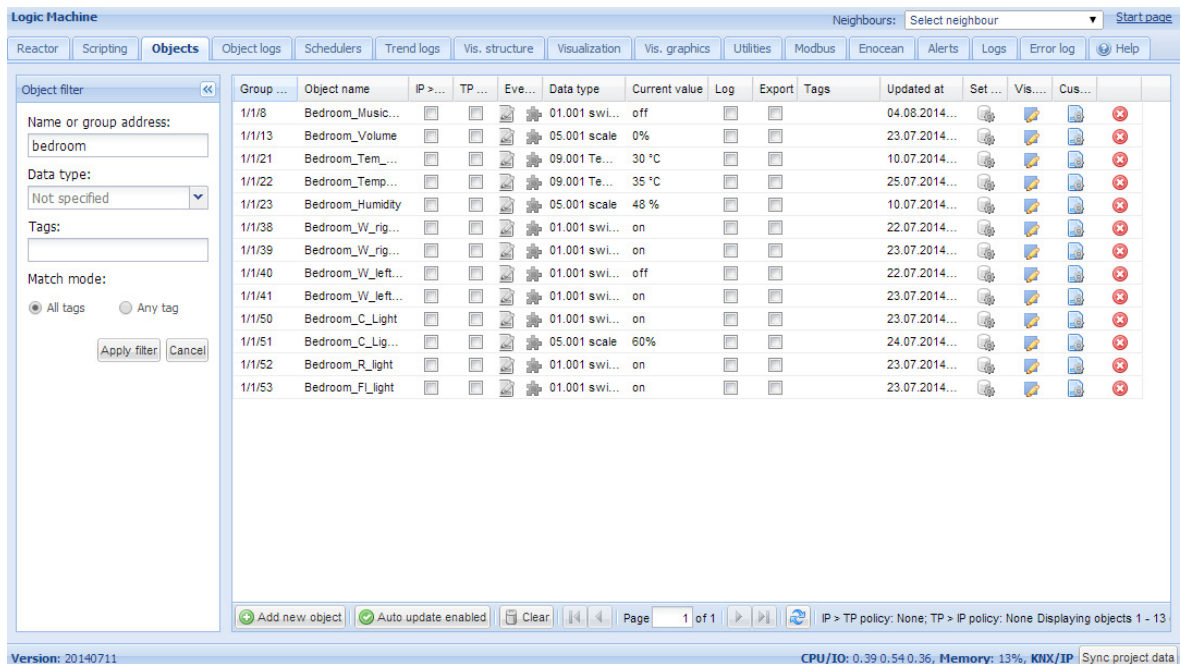
There is also the following bar on the bottom of the configuration screen:

CPU/IO: 0.23 0.33 0.23, Memory: 13%, KNX/IP Sync project data

- **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. 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

### 1.3.6. 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.



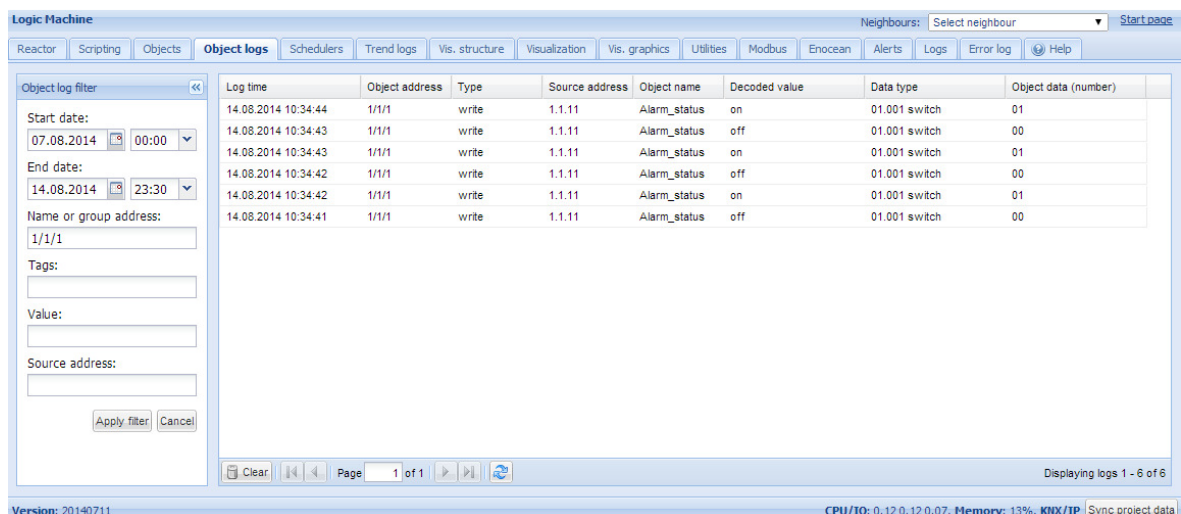
### Match mode:

*All tags* – represents AND function when all tags should match

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

## 1.4. 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*

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

- Interface language: English
- List items per page: 25
- 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
- Enable Blockly editor: ☒
- Code editor tab size: 2

Below the settings, there is an information icon and a warning message:

- 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

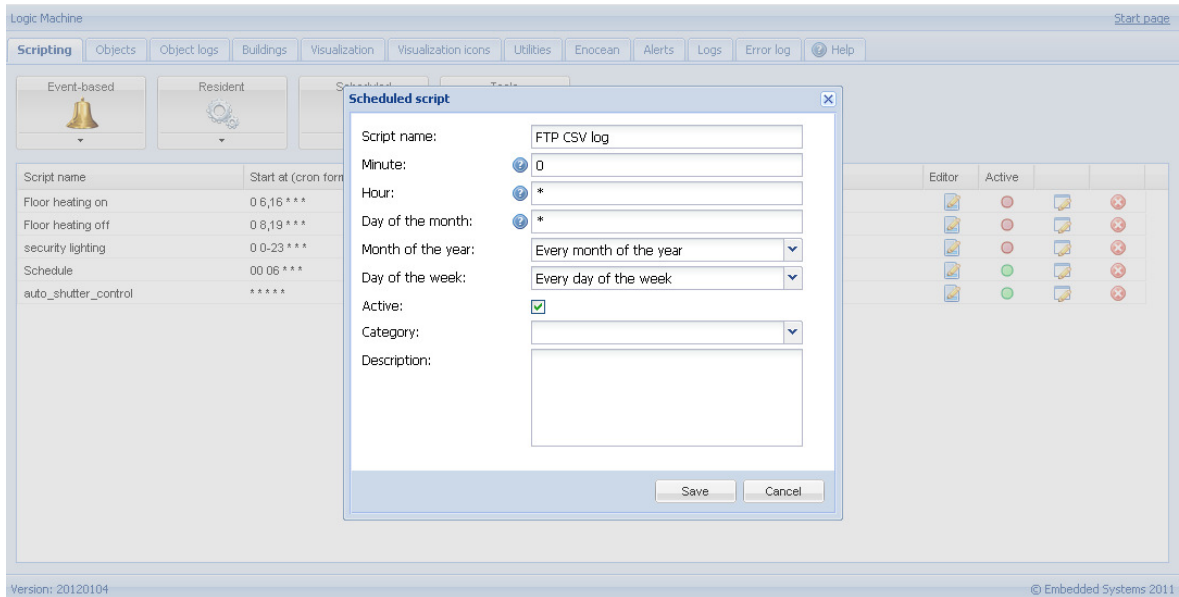
At the bottom right, there are 'Save' and 'Cancel' buttons.

### 1.4.1. 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 'ftlogin', 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://ftplogin: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 inipairs(db:getall(query))do
14.  objects[tounumber(object.address)]= {
15.    datatype=tounumber(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 inipairs(db:getall(query, logtime))do
26.  object = objects[tounumber(row.address)]
27.

```

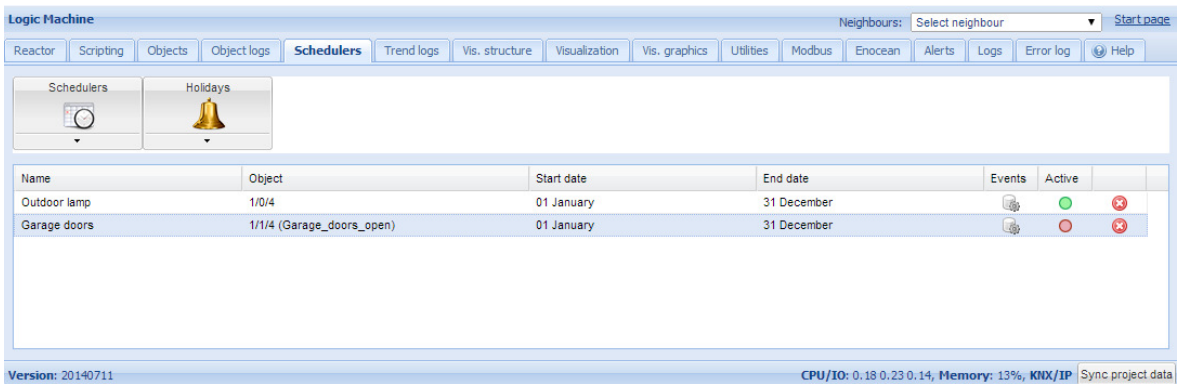
```

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. elseif datatype == 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

```

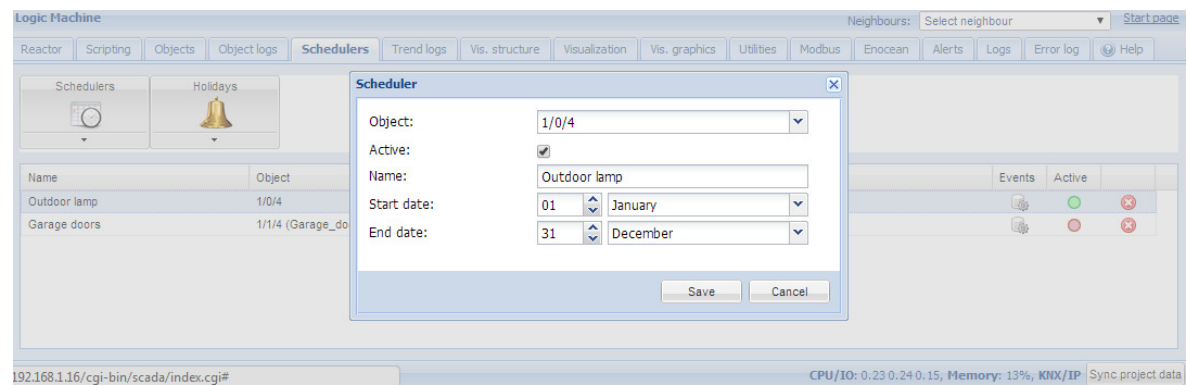
## 1.5. Schedulers

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.



### 1.5.1. Add new scheduler

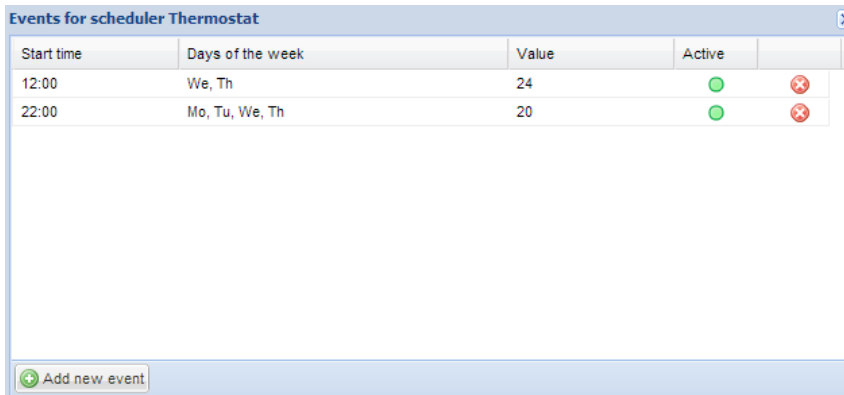
By clicking on the Schedulers → Add new scheduler you will see such parameter window:



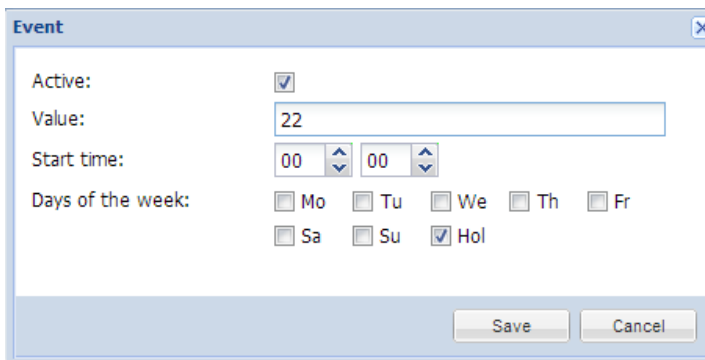
- **Object** – the object group address which will be controlled by scheduler
- **Active** – define this scheduler as active or not
- **Name** – name of the scheduler
- **Start date** – start date of the scheduler
- **End date** – end date of the scheduler

### 1.5.2. 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="button" value="X"/>
22:00	Mo, Tu, We, Th	20	<input checked="" type="checkbox"/>	<input type="button" value="X"/>



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

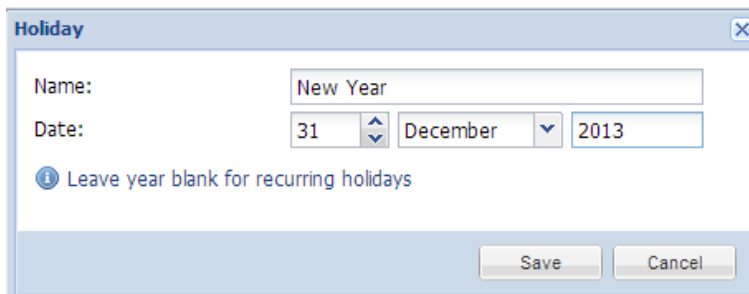
**Start time** – start time for the event

**Days of the week** – days of the week when the event will be triggered.

**Hol**– holidays which are defined in *Holidays* tab

### 1.5.3. 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

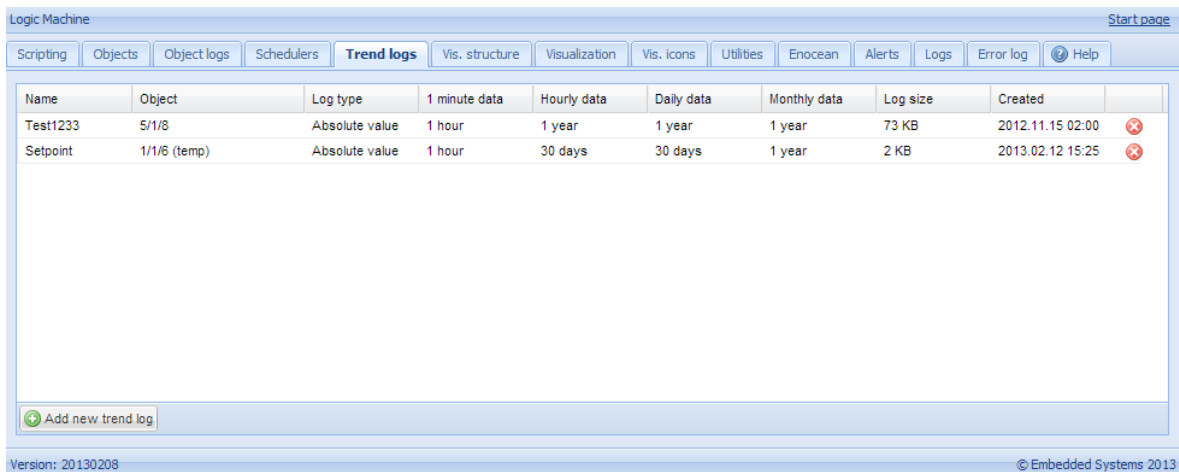


**Name** – the name of the holiday entry

**Date** – date of the holiday

## 1.6. Trend logs

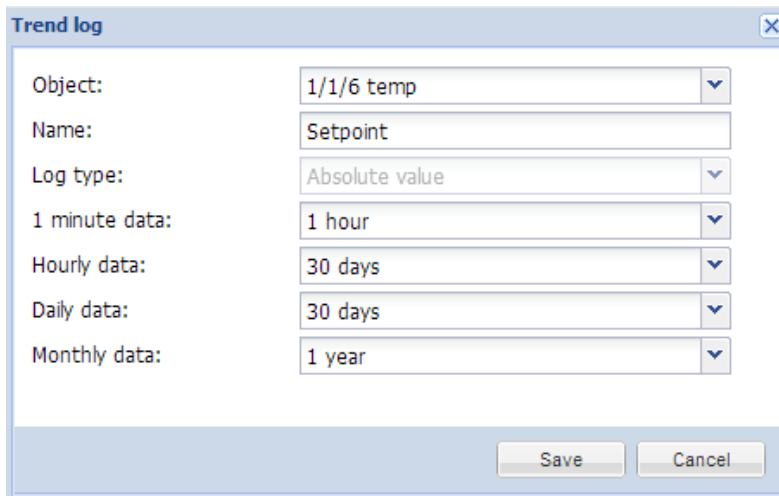
Trends logs are administration of user mode trends, used to see historical object graphical values, compare with other period values.



Name	Object	Log type	1 minute data	Hourly data	Daily data	Monthly data	Log size	Created
Test1233	5/1/8	Absolute value	1 hour	1 year	1 year	1 year	73 KB	2012.11.15 02:00
Setpoint	1/1/6 (temp)	Absolute value	1 hour	30 days	30 days	1 year	2 KB	2013.02.12 15:25

Version: 20130208 © Embedded Systems 2013

### 1.6.1. Add new trend log



**Trend log**

Object: 1/1/6 temp

Name: Setpoint

Log type: Absolute value

1 minute data: 1 hour

Hourly data: 30 days

Daily data: 30 days

Monthly data: 1 year

Save Cancel

**Object** – choose from list of object the one to make trends for

**Name** – name of the trend

**Log type** [*Counter*, *Absolute value*] – type of the log. *Counter* type is used to count the date, *Absolute value* – saves the actual readings

**1 minute data** – 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

**Hourly data** – average value of hourly data for specific time interval

**Daily data** –average value of daily data for specific time interval

**Monthly data** – average value of monthly 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.

### 1.6.2. 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('genohm-scada.trends')
```

Fetch one or many values for the given period:

```
trends.fetch(name, mode, period)  
trends.fetchone(name, mode, period)
```

#### Parameters:

- **name** – trend log name, required
- **mode** – either 'day', 'month' or 'year', required
- **period** – optional, will use current date if not specified  
If specified, must be a Lua table with the following fields:
  - day* – required for day mode only
  - month* – required for day and month modes
  - year* – required for all modes

#### Return values:

*fetch* returns Lua table with values for the given period or nil on error. Number of values in the table depends on period and log retention settings. For example, in month mode this function can return values for each day or only a single value for the whole month

*fetchone* returns single value for the given period or nil on error

#### Example:

```
require('genohm-scada.trends')  
  
-- fetch current value  
today = trends.fetchone('Gas', 'day')  
  
-- get current date as table and set day to yesterday
```

```

date = os.date('*t')
date.day = date.day - 1

-- fetch previous value
yesterday = trends.fetchone('Gas', 'day', date)

```

*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('genohm-scada.trends')

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

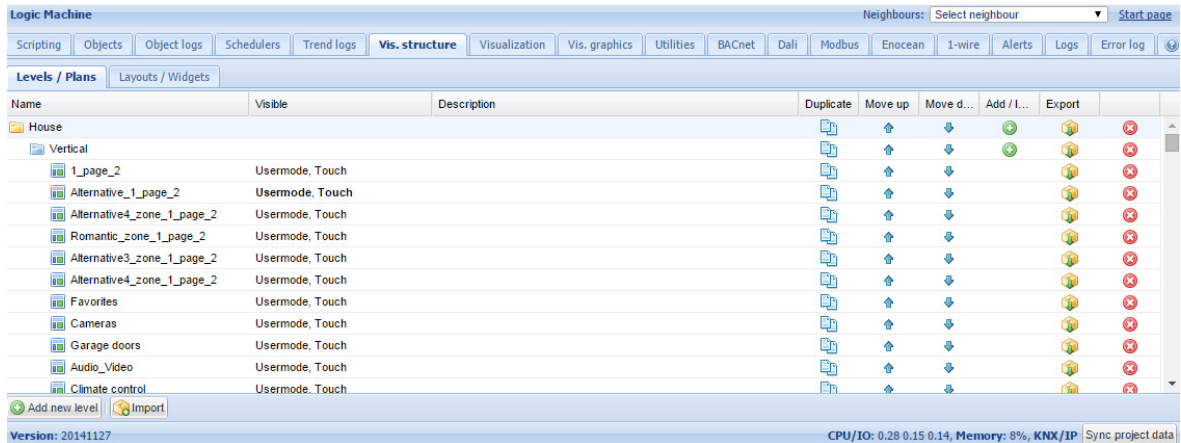
-- get total hot water usage for year 2011
value = trends.fetchone('Hot Water', 'year', { year = 2011 })

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

```

## 1.7. Visualization structure

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

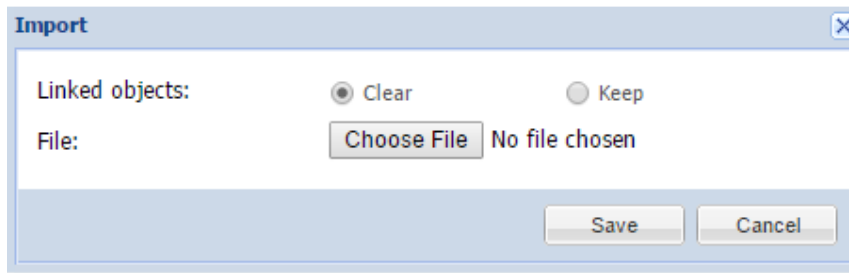


### 1.7.1. Levels / Plans



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.

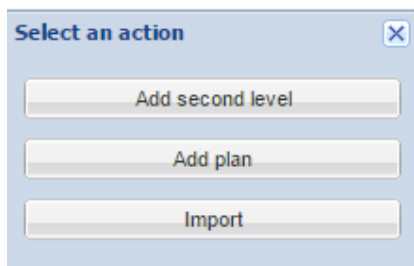
The screenshot shows a dialog box titled 'Level'. It has three input fields: 'Level name:' with the value 'Villa', 'Pin code:' with the value '203', and 'Description:' which is empty. At the bottom right, there are two buttons: 'Save' and 'Cancel'.

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.



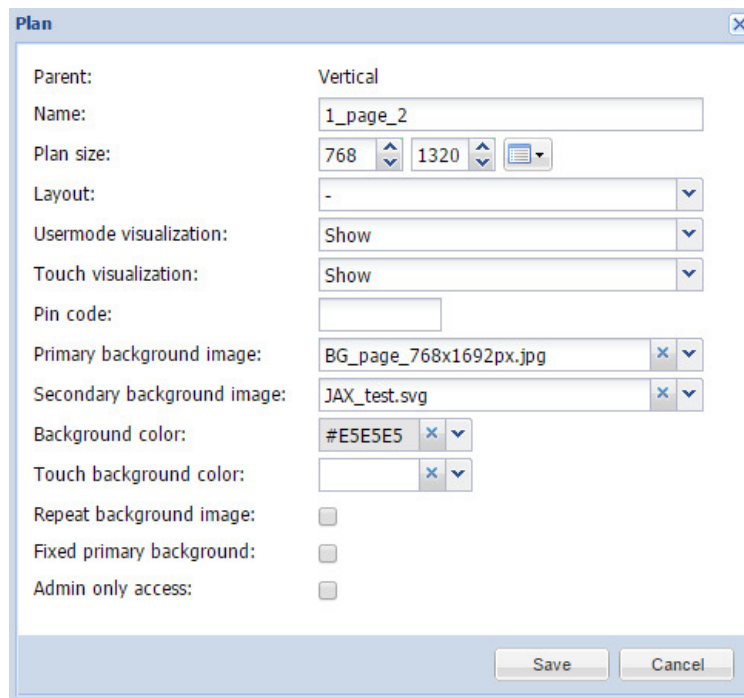
The 'Import' dialog box has a title bar with a close button. It contains two radio buttons for 'Linked objects': 'Clear' (selected) and 'Keep'. Below this is a 'File:' label followed by a 'Choose File' button and the text 'No file chosen'. At the bottom right are 'Save' and 'Cancel' buttons.

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 has a title bar with a close button. It contains three buttons: 'Add second level', 'Add plan', and 'Import'.

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



The 'Plan' dialog box has a title bar with a close button. It contains the following fields and controls:

- Parent: Vertical
- Name: 1\_page\_2
- Plan size: 768, 1320, and a dropdown menu
- Layout: -
- Usermode visualization: Show
- Touch visualization: Show
- Pin code:
- Primary background image: BG\_page\_768x1692px.jpg
- Secondary background image: JAX\_test.svg
- Background color: #E5E5E5
- Touch background color:
- Repeat background image: ☐
- Fixed primary background: ☐
- Admin only access: ☐

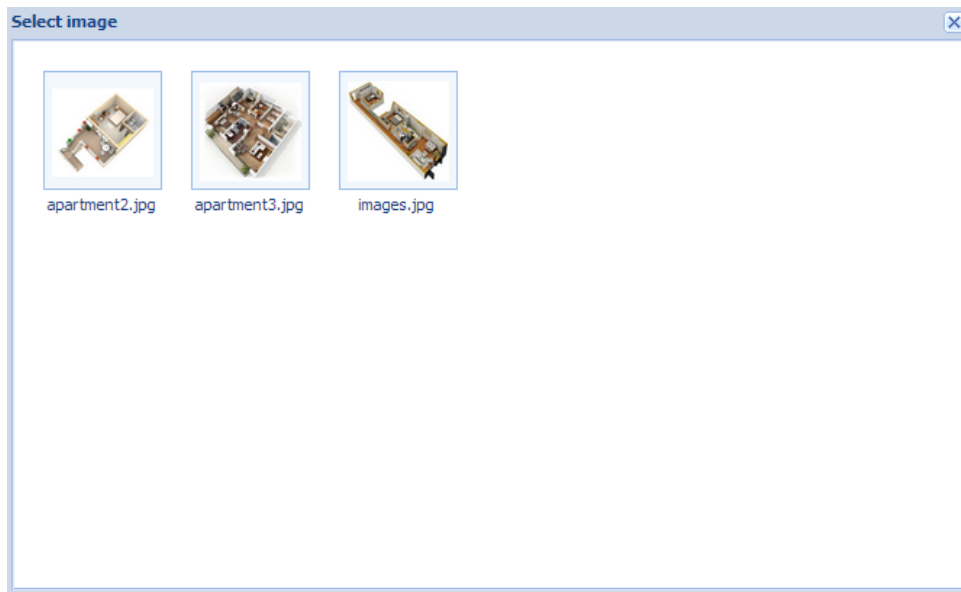
At the bottom right are 'Save' and 'Cancel' buttons.





- **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 size of this parameter:

iPad landscape, fullscreen (XGA) 1024 x 748
iPad landscape, browser (XGA) 1024 x 672
iPad portrait, fullscreen (XGA) 768 x 1004
iPad portrait, browser (XGA) 768 x 928
Tablet landscape (WSVGA) 1024 x 600
Tablet portrait (WSVGA) 600 x 1024
Laptop / Tablet landscape (WXGA) 1280 x 800
Laptop / Tablet portrait (WXGA) 800 x 1280
Laptop / Tablet landscape (HD) 1360 x 768
Laptop / Tablet portrait (HD) 768 x 1360
Big screen (Full HD) 1920 x 1080

- **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

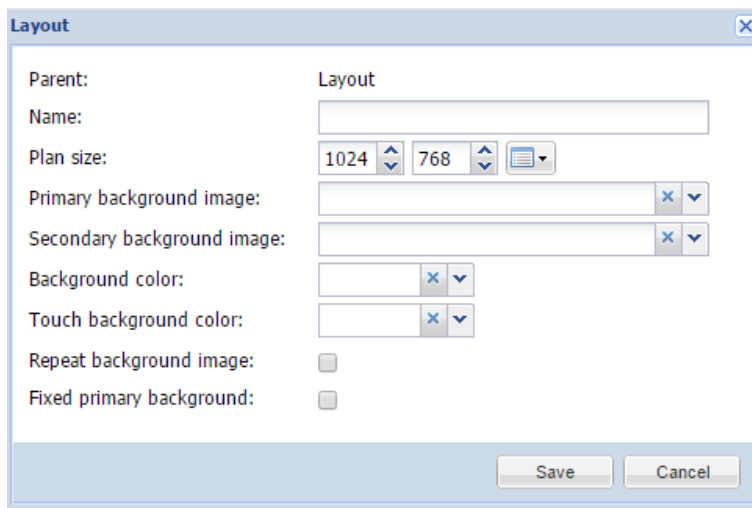
When clicking on Background image, the following window appears with background images which has to be added in *Vis.graphics* → *Images/Backgrounds* in advance:



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 .

### 1.7.2. 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

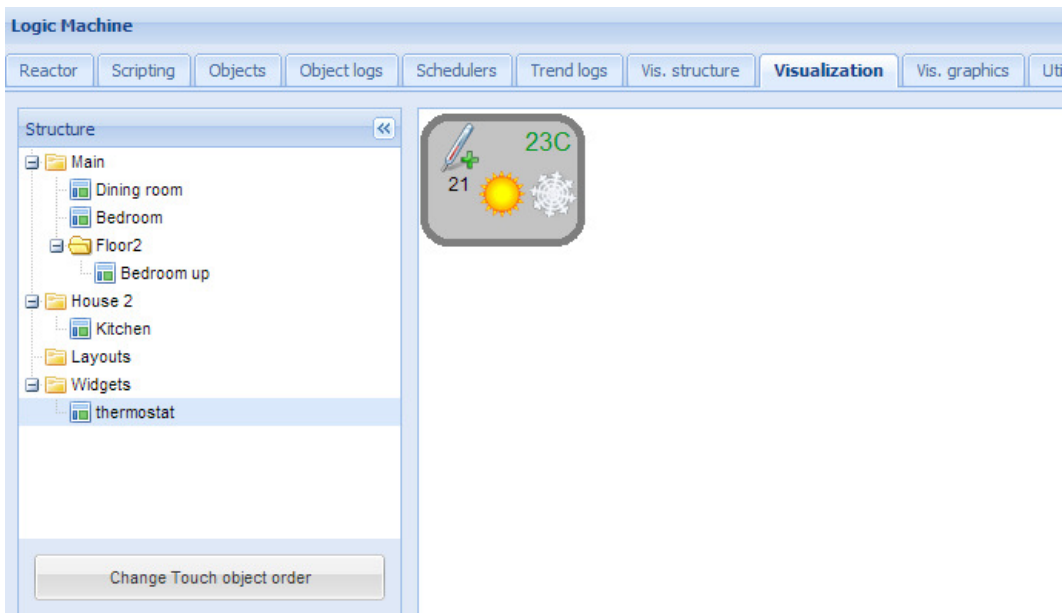
Widgets are used to combine several objects under one object in visualization.

Background image for the widget should be added in *Vis.graphics* → *Images/Backgrounds* in advance.

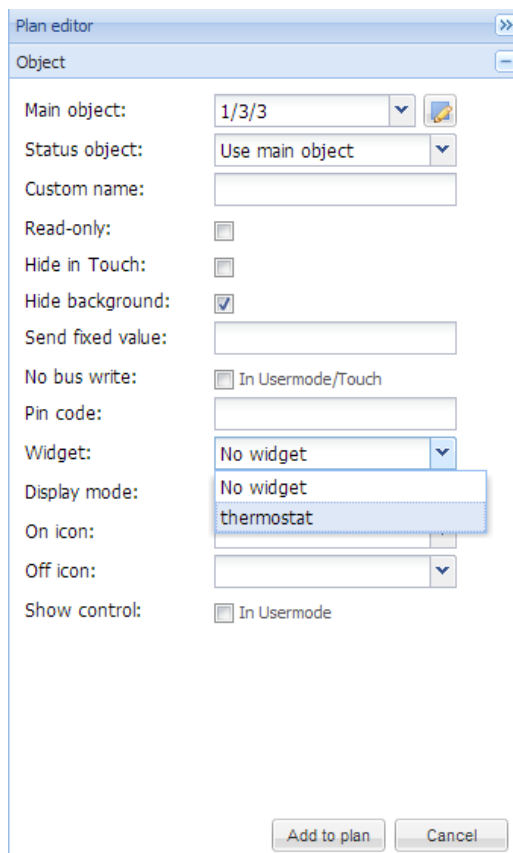
- **Parent** – name of parent widget
- **Name** – name for the widget
- **Plan size** – plan size in pixels. There are predefined resolutions available when clicking on the icon on the right side of this parameter
- **Widget position** – default position of the widget on the screen
- **Primary background 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



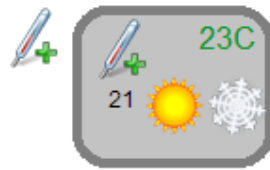
When you have defined the widget in *Layouts/Widgets* tab, you can add objects to it in *Visualization* tab.



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.



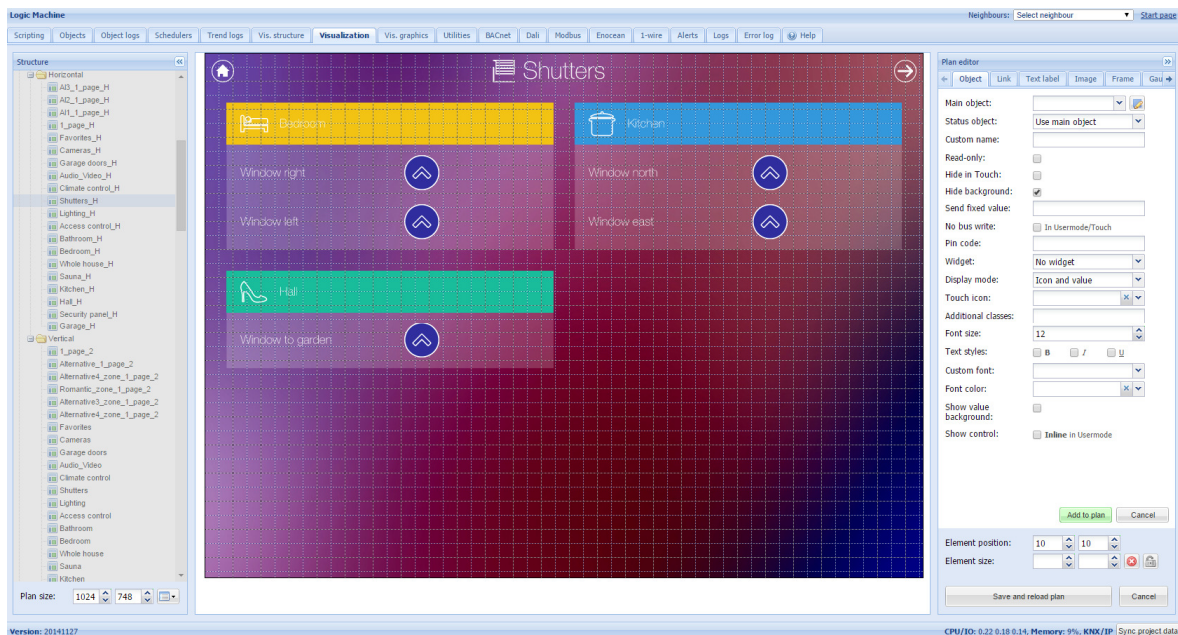
Once added, you can try out the widget in *Usermode visualization* by clicking on added object (temperature sensor icon on the left), the widget appears on click.



## 1.8. 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.



### 1.8.1. 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



### 1.8.2. Object

Plan editor

Object Link Text label Image Frame Gau

Main object: [dropdown] [icon]

Status object: Use main object [dropdown]

Custom name: [text input]

Read-only: ☐

Hide in Touch: ☐

Hide background: ☒

Send fixed value: [text input]

No bus write: ☐ In Usermode/Touch

Pin code: [text input]

Widget: No widget [dropdown]

Display mode: Icon and value [dropdown]

Touch icon: [text input] [x] [dropdown]

Additional classes: [text input]

Font size: 12 [up/down]

Text styles: ☐ B ☐ I ☐ U

Custom font: [dropdown]

Font color: [text input] [x] [dropdown]

Show value background: ☐

Show control: ☐ Inline in Usermode

Add to plan Cancel

Element position: 10 [up/down] 10 [up/down]

Element size: [up/down] [up/down] [x] [lock icon]

Save and reload plan Cancel

- **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 specific setting defining either to show the control in Usermode visualization without icon

-
27.00
+

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.

Min value	Max value	Icon
-10	0	sun-moon-off
0	10	sun-moon-on
10	20	sun-rain-on
20	30	sun-rain-off

Save Cancel

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.

Element position: 460 65

Element size: 50 50

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.

### 1.8.3. 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.

Plan editor

Object Link Text label Image Frame Gauge

Link to: Garage doors

Custom name: Link to garage

Hide in Touch: ☐

Hide background: ☒

Display mode: Icon

Icon: Garagedoors.svg

Active state icon:

Additional classes:

Add to plan Cancel

Element position: 460 65

Element size: 50 50

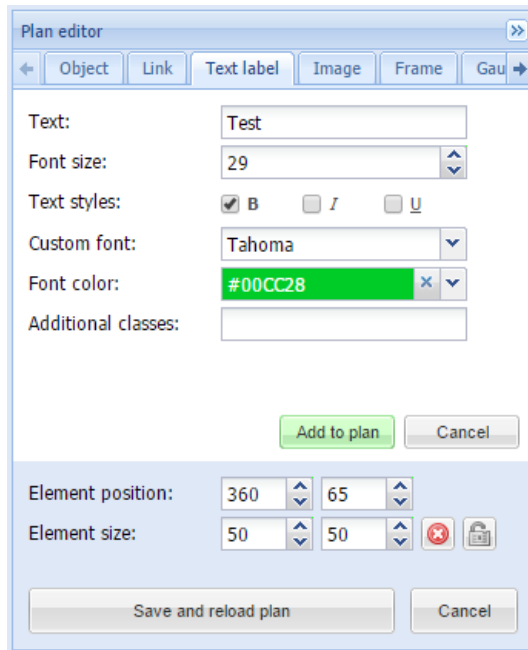
Save and reload plan Cancel

- **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.

### 1.8.4. Text Label

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



The screenshot shows the 'Plan editor' dialog box with the 'Text label' tab selected. The dialog contains the following fields and controls:

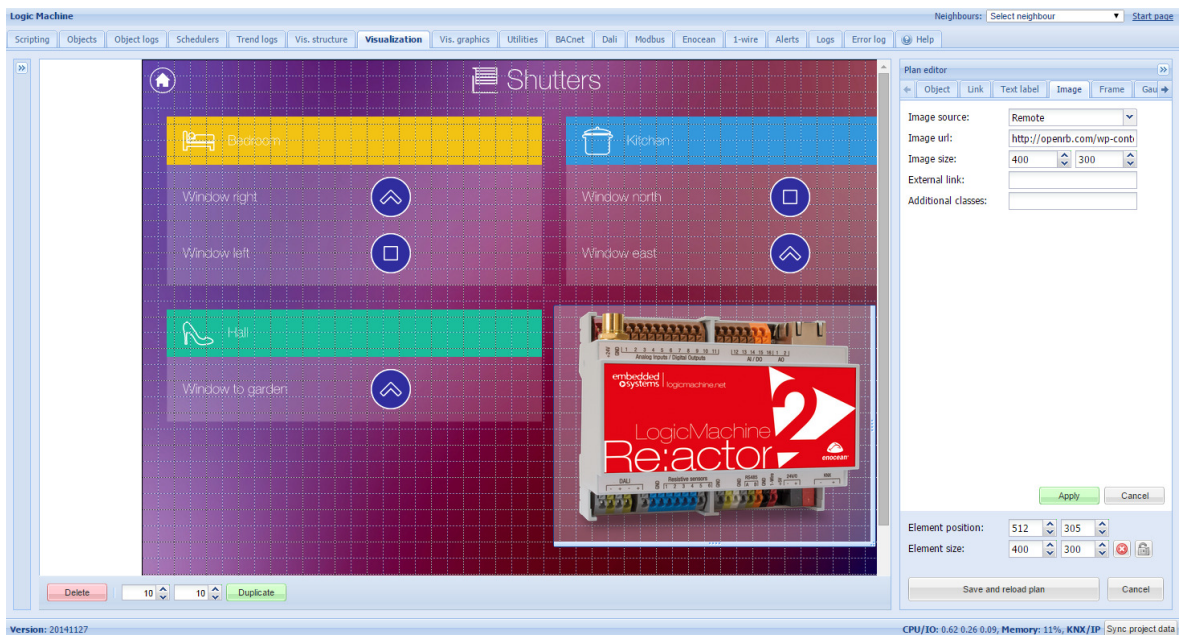
- Text:** A text input field containing 'Test'.
- Font size:** A spin box set to 29.
- Text styles:** Checkboxes for Bold (checked), Italic, and Underline.
- Custom font:** A dropdown menu showing 'Tahoma'.
- Font color:** A color picker showing '#00CC28'.
- Additional classes:** An empty text input field.
- Buttons:** 'Add to plan' (green), 'Cancel', 'Save and reload plan', and another 'Cancel'.
- Element position:** Two spin boxes for X (360) and Y (65) coordinates.
- Element size:** Two spin boxes for width (50) and height (50), with a red 'X' icon and a lock icon.

- **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.

### 1.8.5. Image

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

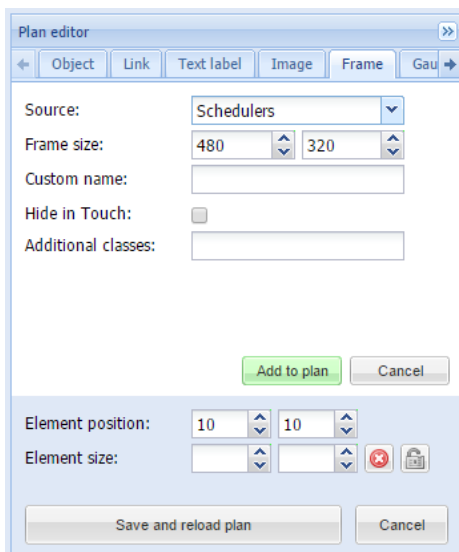


- **Image source [Local; Remote]** – image source location
- **Source url / Select image** – Source URL of the image or image from local database
- **Image size** – width and height of the image
- **External link** – external link URL when pressing on the image
- **Additional classes** – additional CSS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

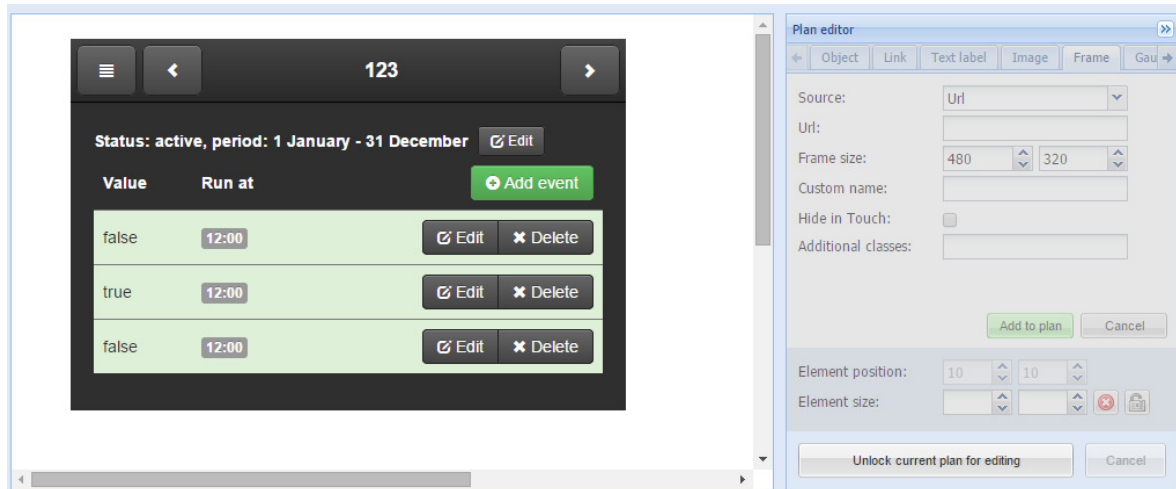
### 1.8.6. Frame

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



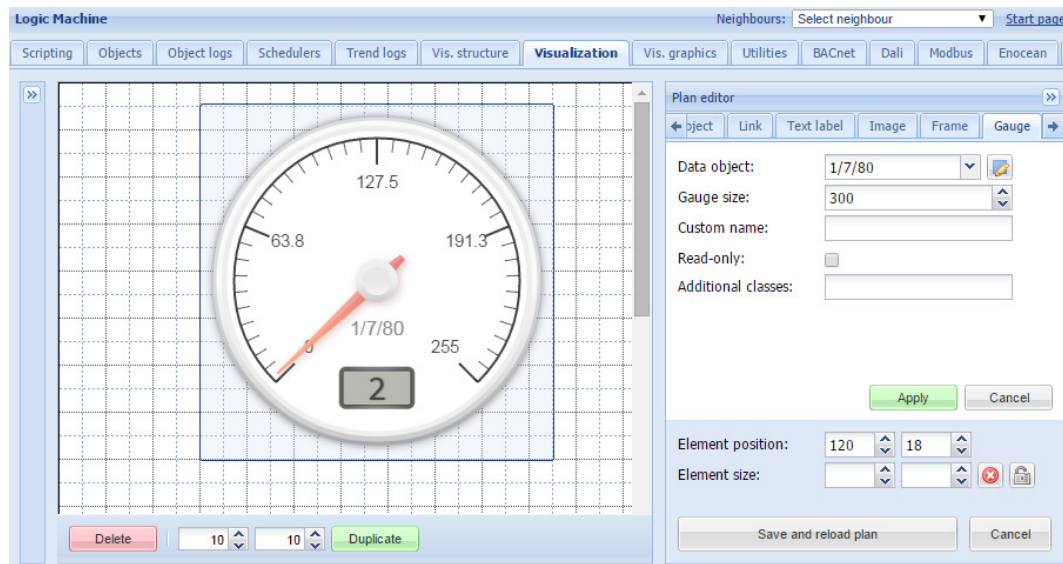


- **Source [Url, Schedulers; Trend logs]** – frame source
- **Url** – Source URL of the page to integrate
- **Frame size** – width and height of the frame
- **Custom name** – custom name of the frame object
- **External link** – external link URL when pressing on the image
- **Hide in Touch** – defines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



### 1.8.7. 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.

### 1.8.8. Camera

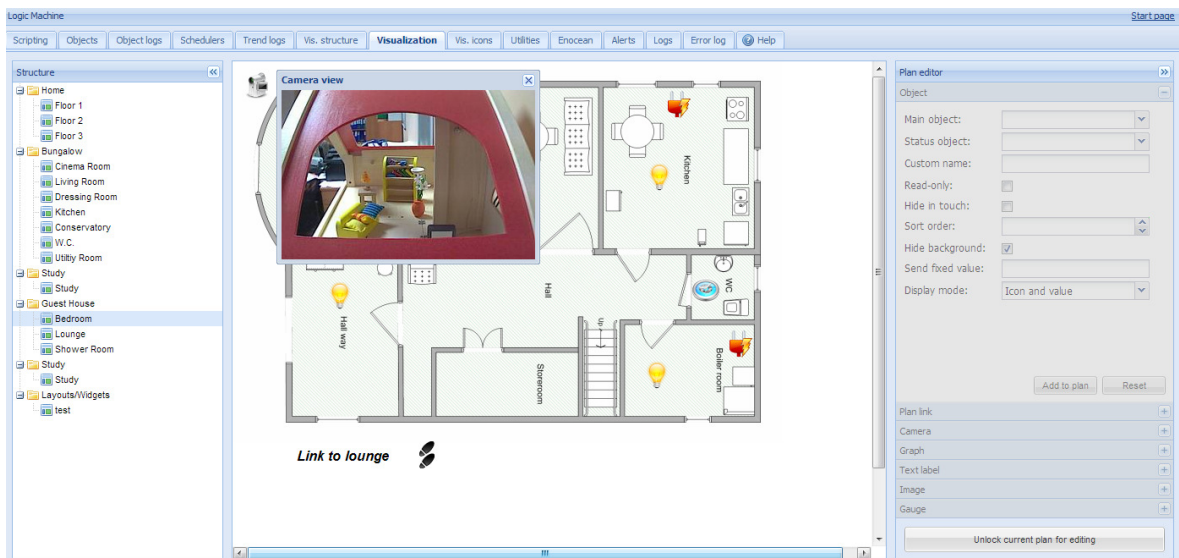
LogicMachine supports third party IP web camera integration into its visualization.

The screenshot shows the 'Plan editor' window with the 'Camera' tab selected. The 'Source url' field contains 'http://192.168.1.212/mjpg'. The 'Window size' is set to 640 by 480. The 'Icon' is 'camera.png'. The 'Auto open window' checkbox is unchecked, and the 'Hide background' checkbox is checked. The 'Additional classes' field is empty. The 'Add to plan' button is highlighted in green. At the bottom, the 'Element position' is set to 10x10, and the 'Element size' field is empty with a red 'x' icon. The 'Save and reload plan' button is also visible.

- **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

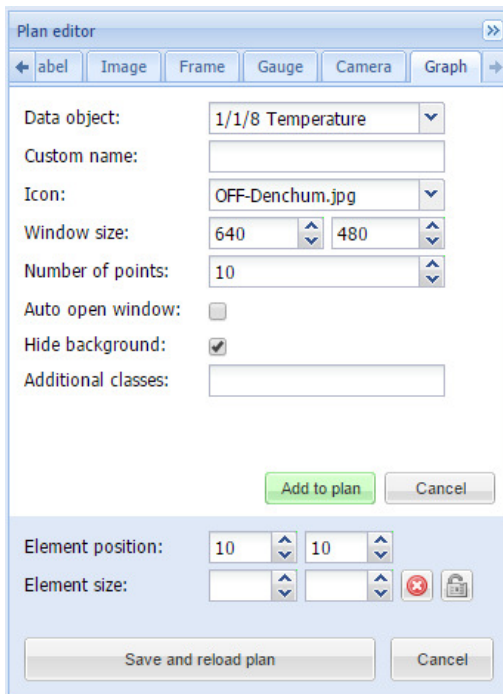
**Note!** If IP camera requires user name and password, enter the url in form ***http://USER:PASSWORD@IP***

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.



### 1.8.9. Graph

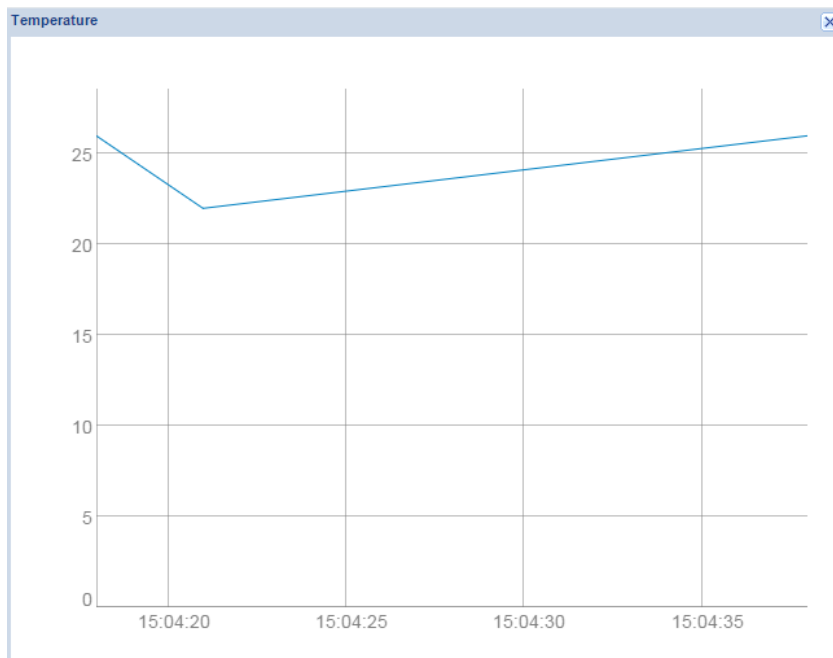
Real-time graphs can be integrated into visualization system to monitor the current and old value of scale-type objects. Make sure logging is enabled for the object in *Object* tab which values is planned to be shown in the graph.



- **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

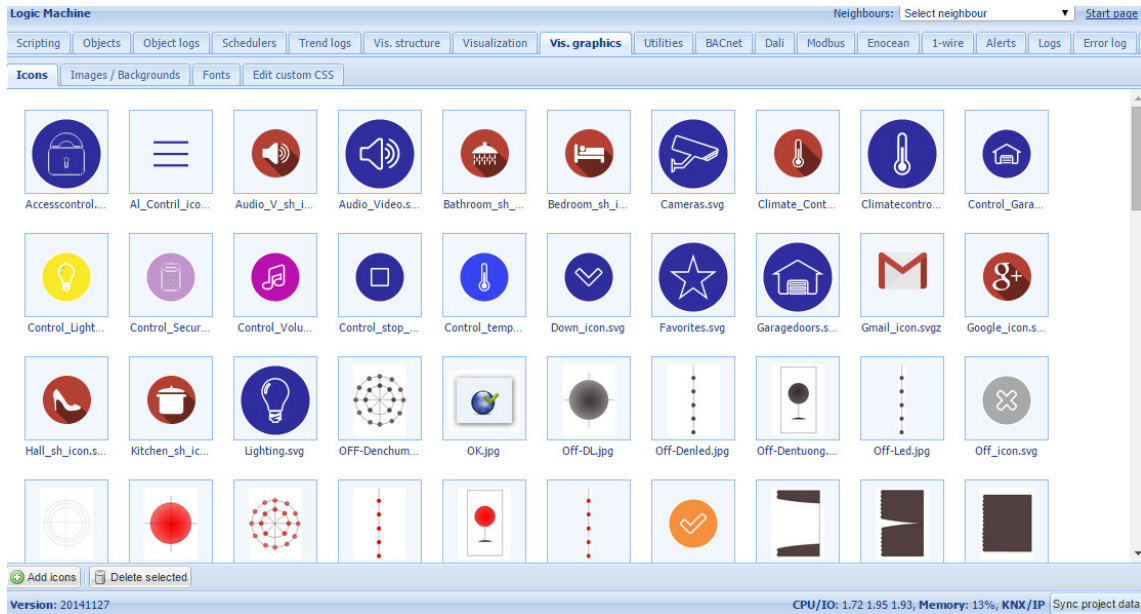
- ***Auto open window*** – graph window is automatically opened
- ***Hide background*** – hide icon background
- ***Additional classes*** – additional CSS classes for the element

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.

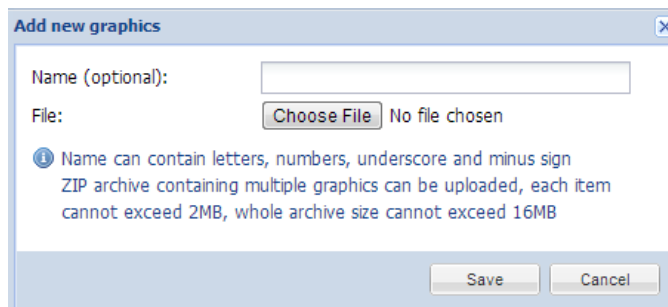


## 1.9. Vis.graphics

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



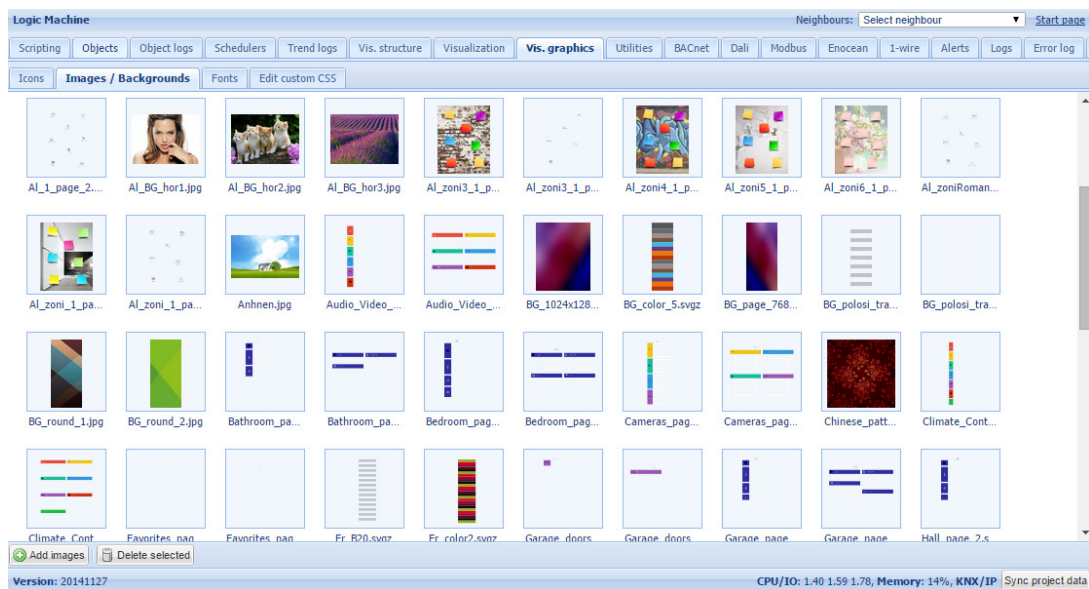
Press on *Add icons* button to add a new entry. The system accepts any size icons. GIF is also supported.



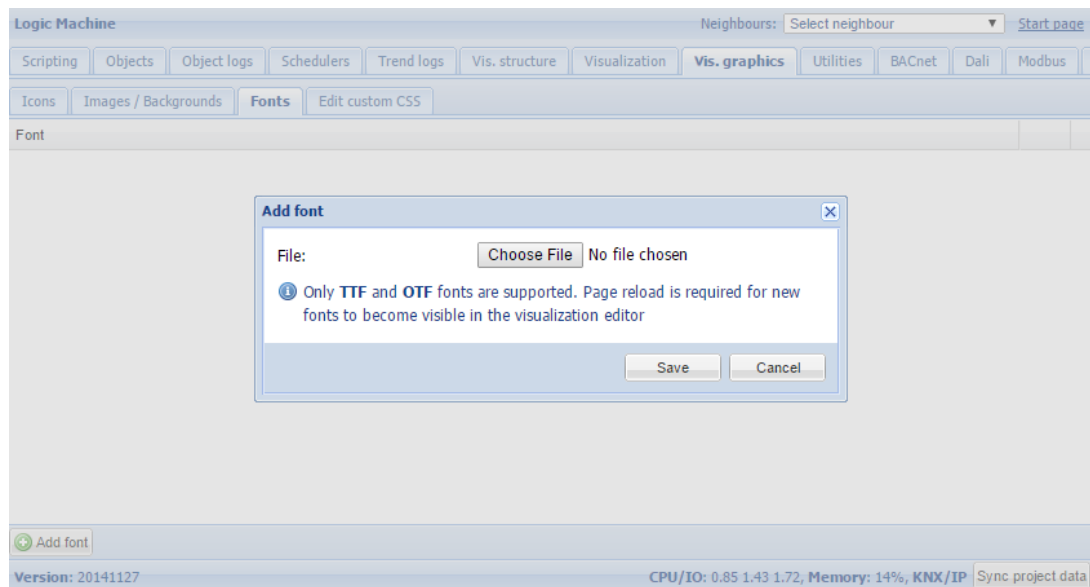
**Name (optional)** – the name of the icon

**File** – Icon file location

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



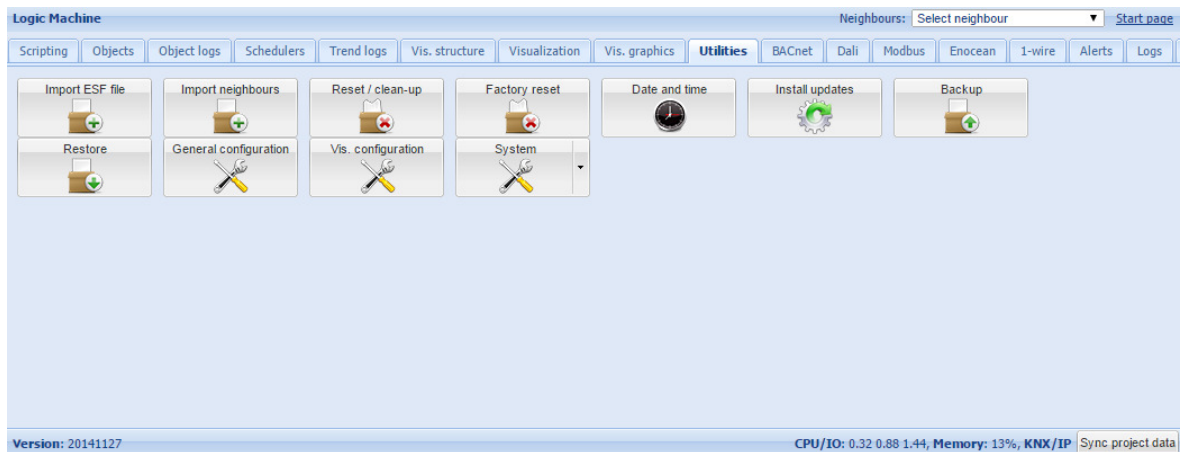
In *Fonts* tab you can add custom fonts



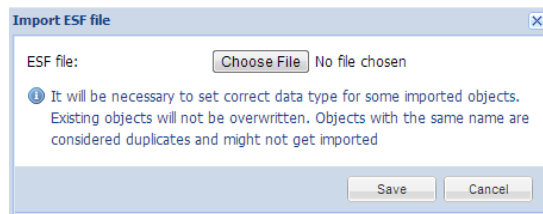
In *Custom CSS* tab you can add your CSS style for the visualization which you can use when adding elements into visualization, so any elements of Look and Feel is customizable with this solution.

## 1.10. Utilities

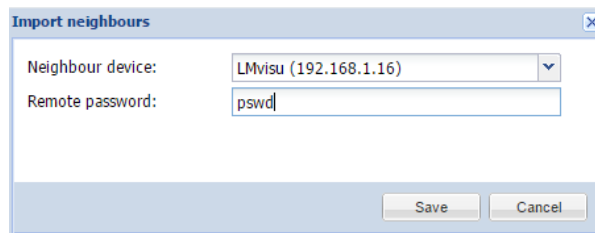
There are following utilities in the tab available:



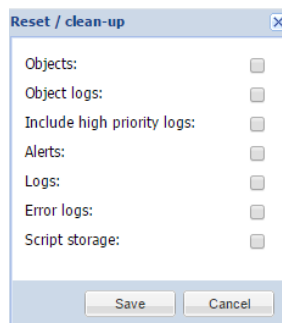
**Import ESF file**– imports ETS object file. It will be necessary to set correct data types for some imported objects. Existing objects will not be overwritten. Objects with the same name are considered duplicates and might not be imported



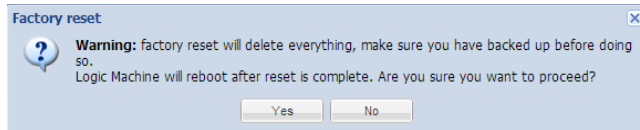
**Import neighbours** – import list of objects from network LM devices



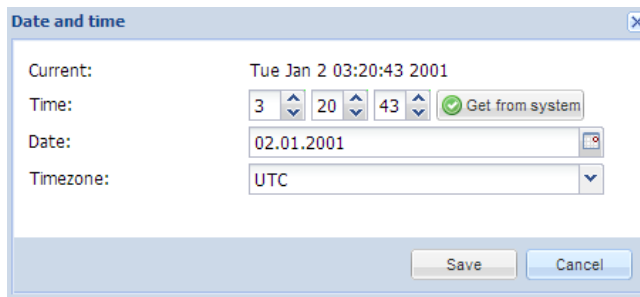
**Reset / clean-up** – delete all objects from the Logic Machine, they disappear from visualization aswell



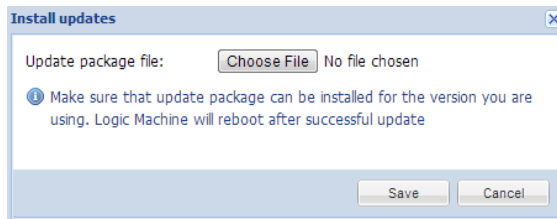
**Factory reset**– delete all configuration and return to factory defaults



**Date and time** – data and time settings

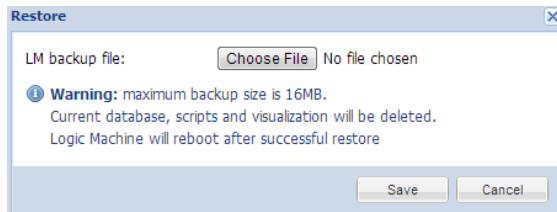


**Install updates** – install LogicMachine update file \*.lmu. LogicMachine will reboot after successful update



**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
- List items per page: 25
- 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
- Enable Block editor: ☒
- Code editor tab size: 2

Informational notes:

- 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

**List items per page** –count of lines per page e.g. *Objects*, *Object logs*, *Alerts* etc.

**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

**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



## Vis. Configuration – visualization specific settings

Vis. configuration

Usermode sidebar: Show as overlay (auto-hide) ▼

Usermode view: Center plans, enable auto-sizing ▼

Usermode page transition: No transition ▼

Usermode auto-size upscaling: ☐

Usermode background color:

Usermode background image:

Custom font:

Use dark theme: ☐

Visualization pin code:

Enable swipe gesture: ☒

Dim inactive visualization after:  minutes

Dimming level:  %

Show alerts in Usermode: ☐

Save Cancel

**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

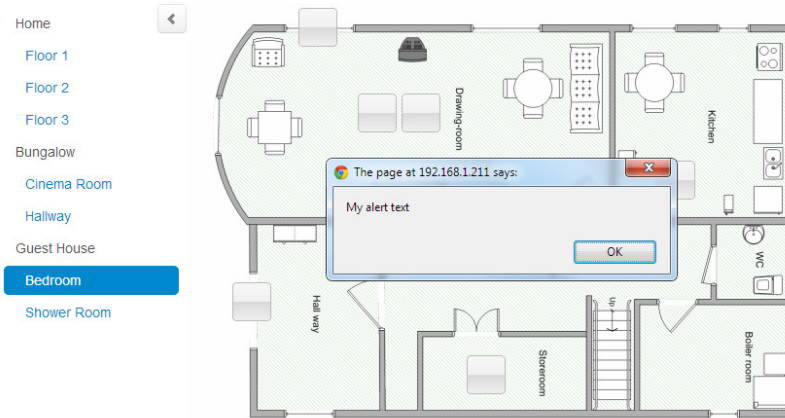
**Visualization pin code** – PIN code to access visualization

**Enable swipe gesture** – check to enable swipe gesture to move across plans from your touch device

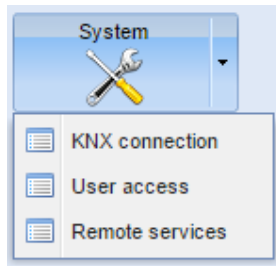
**Dim inactive visualization after** – define time in minutes after which the screen will be dimmed where visualization is opened

**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.



## 1.11. 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
<div> <span>⏪</span> <span>⏴</span> Page <input type="text" value="1"/> of 93 <span>⏵</span> <span>⏩</span> <span>🔄</span> </div>	
Displaying alerts 1 - 25 of 2317	

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

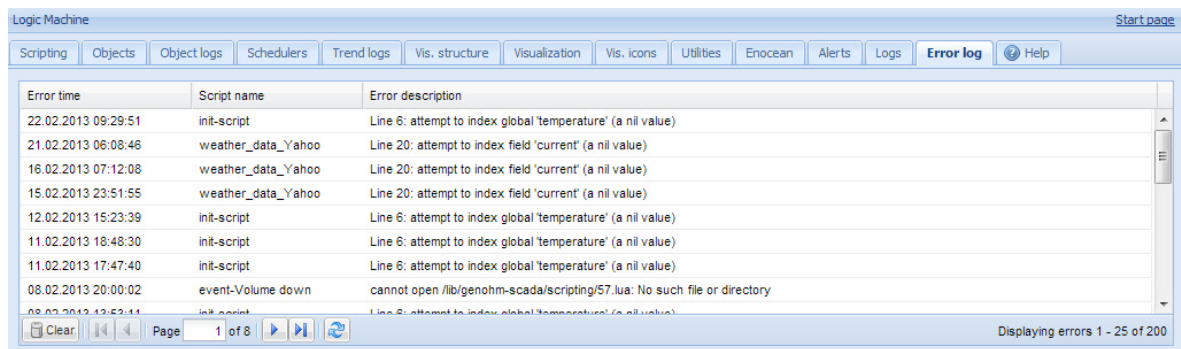


## Example

```
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
```

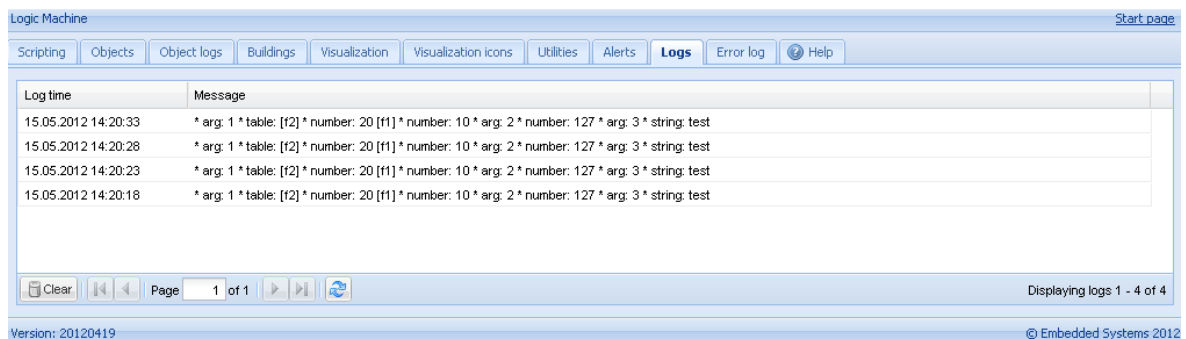
## 1.12. Error log

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



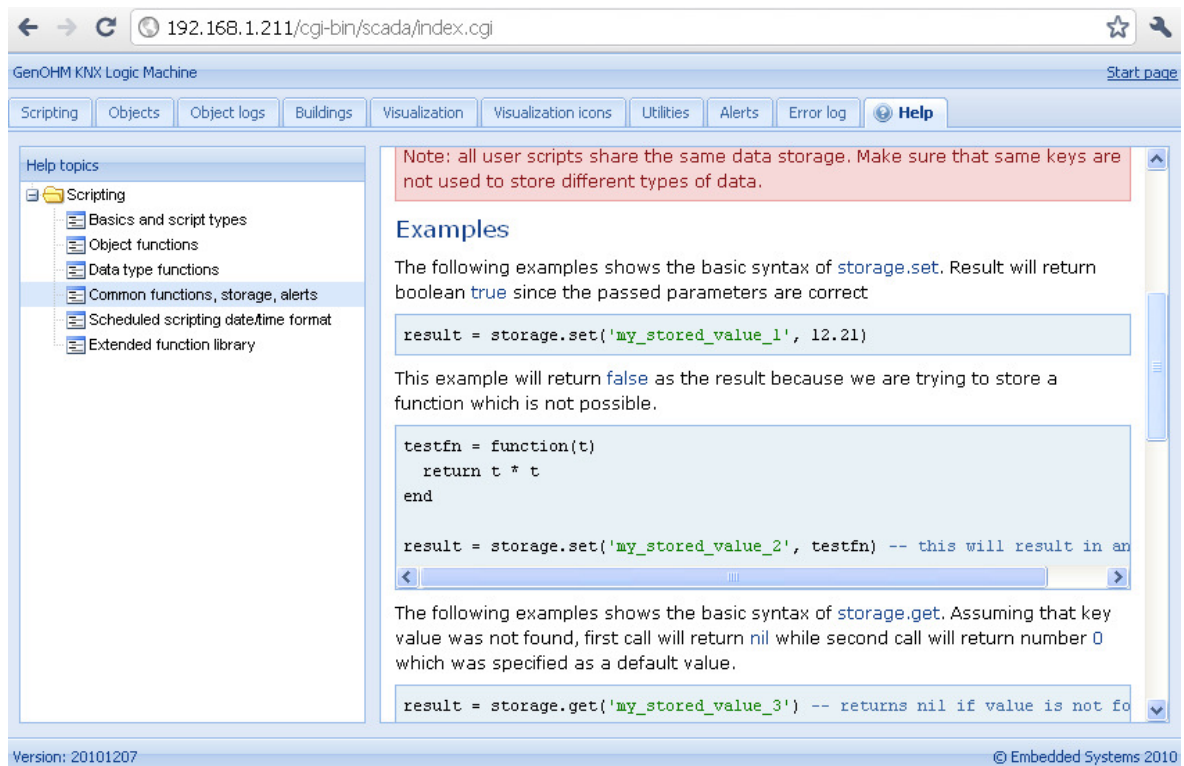
## 1.13. Logs

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



## 1.14. Help

Documentation for scripting syntaxes is displayed in *Help* tab.

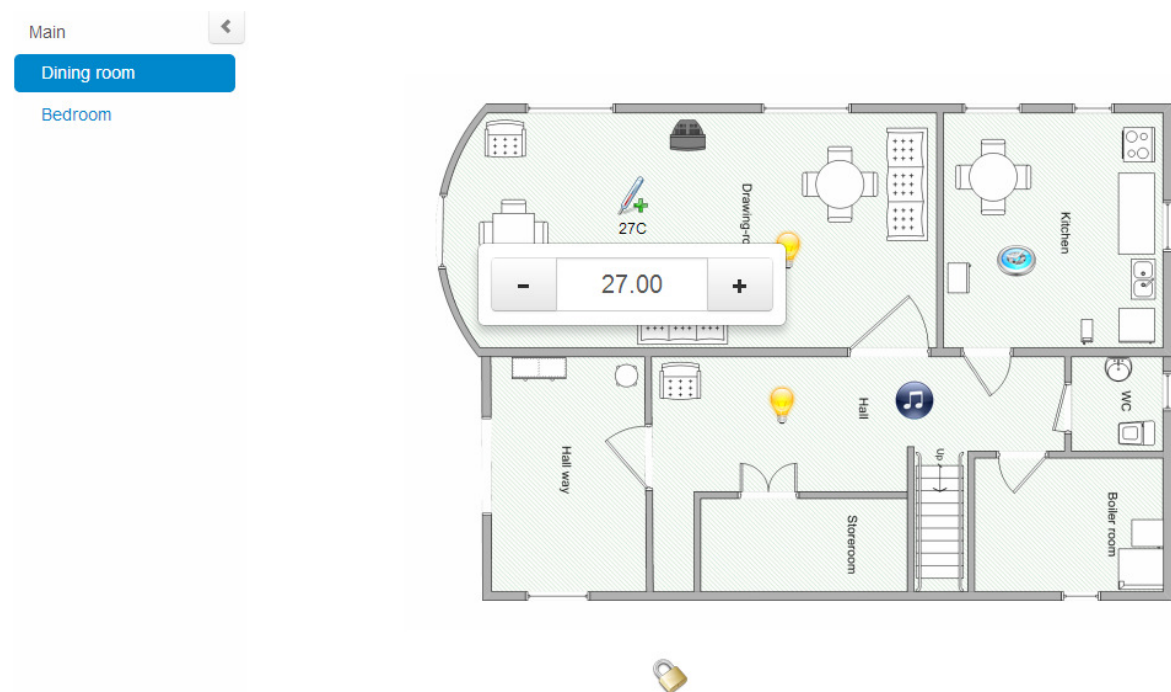


## 2. User mode visualization

*User mode visualization* contains created visualization maps.

There are three access levels: read, write, admin (password access can be also disabled)

Access level	Login	Password
Read-only	Visview	visview
Write	viscontrol	viscontrol
Write+admin level	visadmin	visadmin



## 2.1. Custom design Usermode visualization

Through Custom CSS styles it is possible to create different type of visualization maps.



## 3. Touch visualization

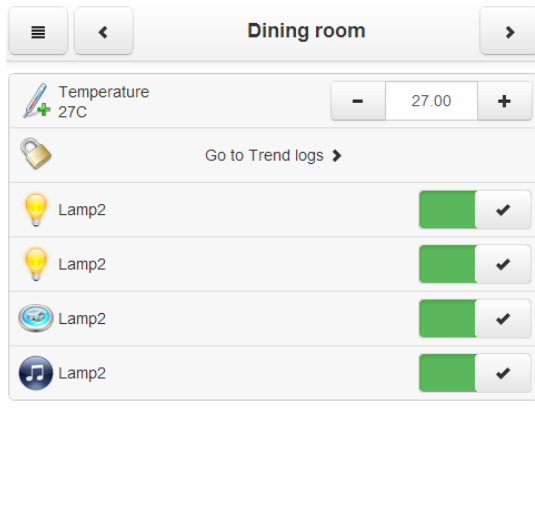
Touch visualization is designed for iPhone/iPod/iPad/Android touch screen devices. All objects which are added in *Logic Machine* configuration by default are visible in touch visualization (if there is no *Hide in touch* option enabled).

There are three access levels: read, write, admin

Access level	Login	Password
Read-only	visview	visview
Write	viscontrol	viscontrol
Write+admin level	visadmin	visadmin



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.

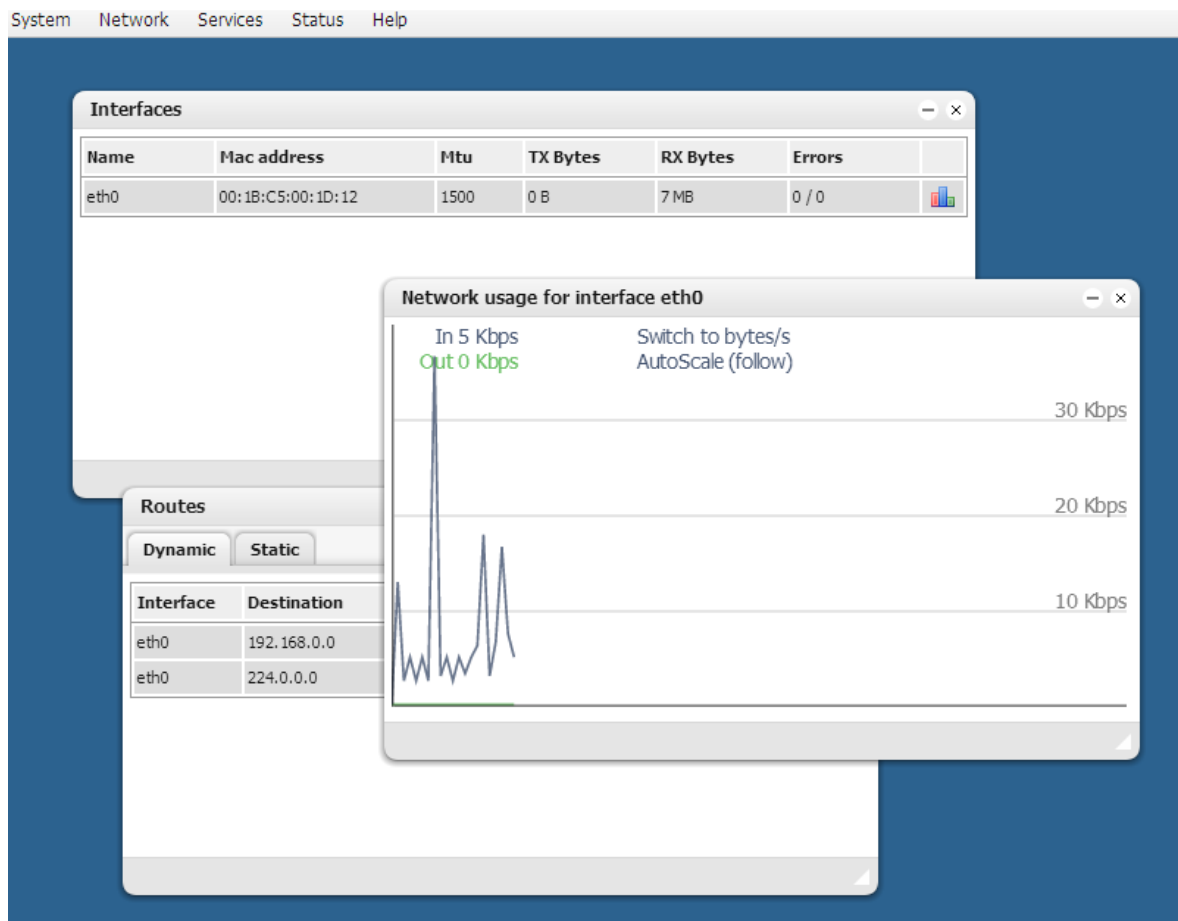


#### Launching visualization on touch device (iPad in this case)

- 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

## 4. 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.

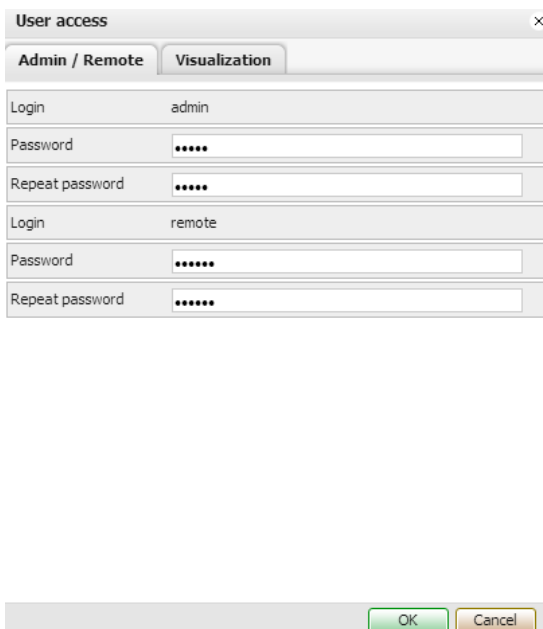


Login	Password
admin	admin

### 4.1. Changing password

The login and password configuration window is located in *System* → *User access*.





**User access**

Admin / Remote Visualization

Login admin

Password .....

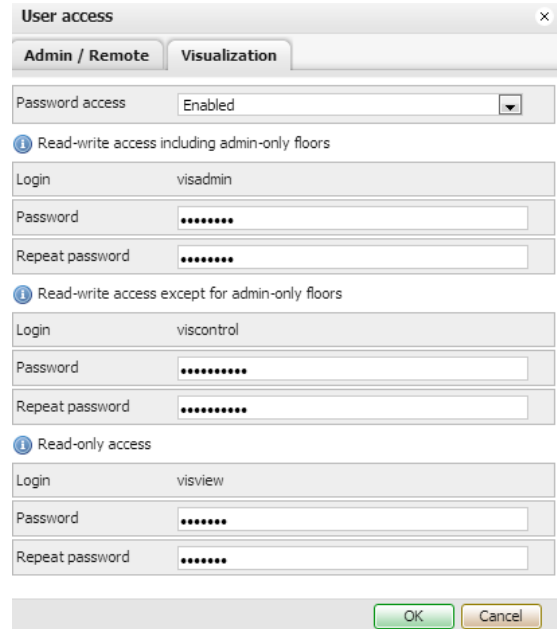
Repeat password .....

Login remote

Password .....

Repeat password .....

OK Cancel



**User access**

Admin / Remote Visualization

Password access Enabled

Read-write access including admin-only floors

Login visadmin

Password .....

Repeat password .....

Read-write access except for admin-only floors

Login viscontrol

Password .....

Repeat password .....

Read-only access

Login visview

Password .....

Repeat password .....

OK Cancel

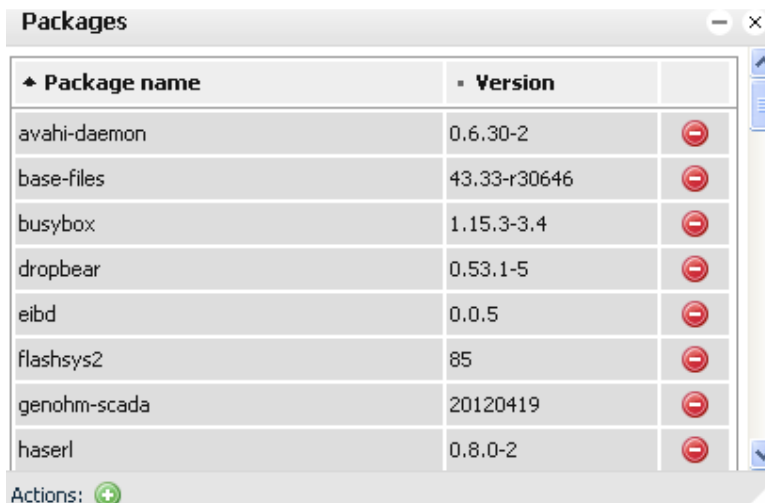
Access control is separated in 3 tabs:

**Admin/Remote** – access parameters for *Logic Machine*, *Network Configuration*, *RSS* and *XML*

**Visualization** – access parameters for *Touch* and *User mode visualization*

## 4.2. Packages

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

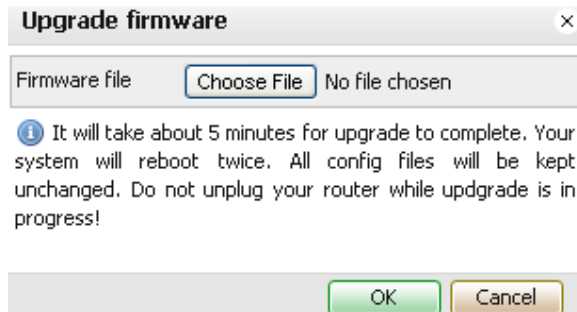


Package name	Version	
avahi-daemon	0.6.30-2	⊖
base-files	43.33-r30646	⊖
busybox	1.15.3-3.4	⊖
dropbear	0.53.1-5	⊖
eibd	0.0.5	⊖
flashsys2	85	⊖
genohm-scada	20120419	⊖
haserl	0.8.0-2	⊖

Actions: ⊕

### 4.3. Upgrade firmware

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



### 4.4. Reboot Logic Machine


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

### 4.5. 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.

## 4.6. 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.

Interfaces						
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

General

Protocol

Static IP

IP address

192.168.10.96

Network mask

255.255.255.0

Gateway IP

192.168.10.2

DNS server

8.8.8.8

Mtu

OK


Cancel

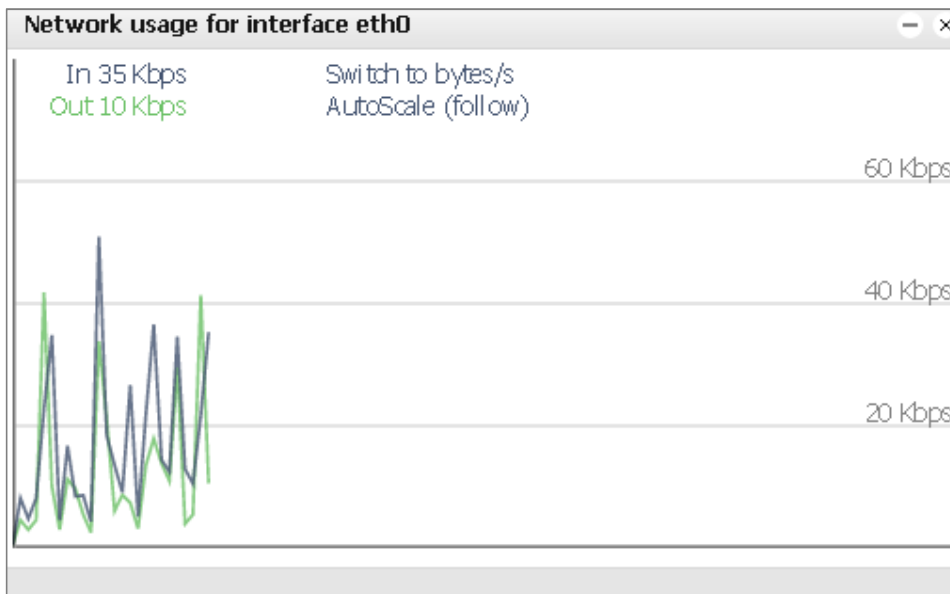
- **Protocol**– specific protocol used for addressing
  - Static IP** – static IP address. By default 192.168.0.10
  - DHCP** – use DHCP protocol to get IP configuration.

**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

#### 4.6.1. Ethernet interface data throughput graph

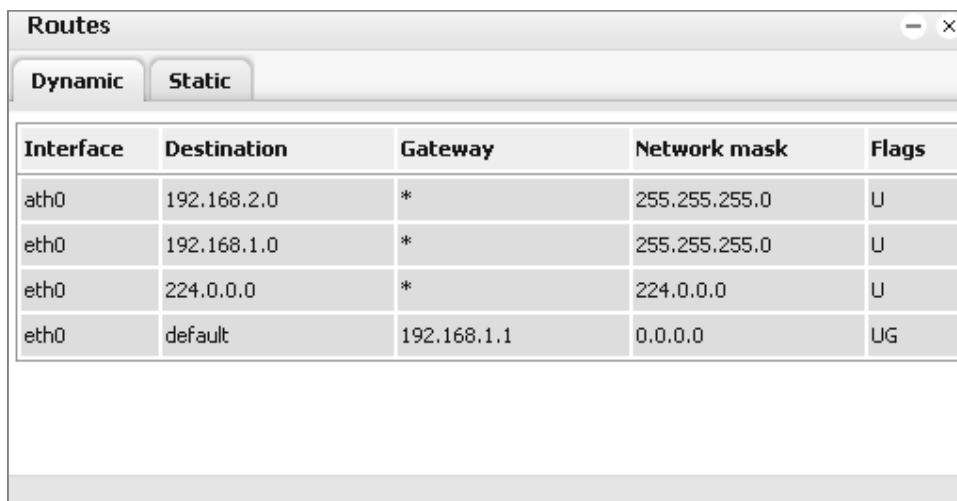
On the main window of the Ethernets tab, if you click on the  button, a new window is opened. It draws a real-time graph of the traffic flow passing the interface (both In and Out). There is a possibility to switch the units of measurement – bytes/s or bytes/s.



## 4.7. Routing Table

System routing table is located in *Network* → *Routes* menu. The window is divided in two parts – Static routes and Dynamic routes.

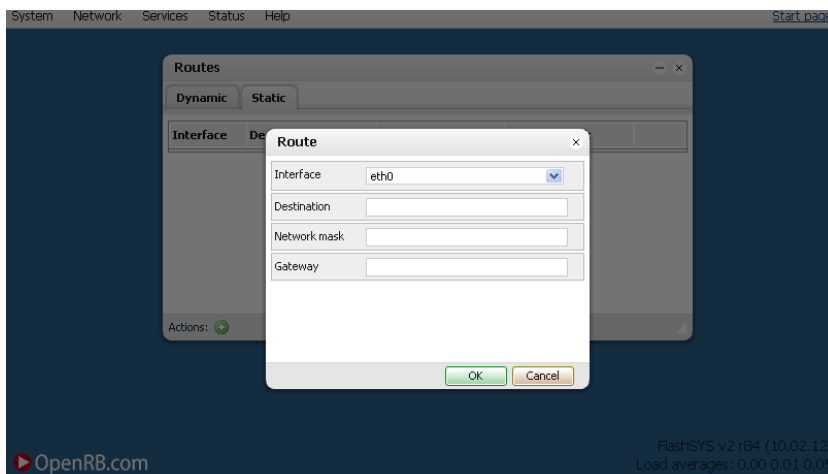
### 4.7.1. Dynamic routes



Dynamic				
Interface	Destination	Gateway	Network mask	Flags
ath0	192.168.2.0	*	255.255.255.0	U
eth0	192.168.1.0	*	255.255.255.0	U
eth0	224.0.0.0	*	224.0.0.0	U
eth0	default	192.168.1.1	0.0.0.0	UG

- **Interface** – interface name
- **Destination** – destination IP address
- **Network mask** – network mask
- **Gateway** – gateway IP address

### 4.7.2. Static routes



- **Interface** – interface name
- **Destination**– destination IP address
- **Network mask** – network mask
- **Gateway** – gateway IP address

## 4.8. ARP table

Address Resolution Protocol table is listed in *Network* → *ARP table*.

ARP table				
Interface	IP address	Mask	MAC address	Flags
eth0	192.168.1.208	*	00:0e:2e:cd:35:e9	0x2
eth0	192.168.1.100	*	00:1c:c0:54:88:cb	0x2

## 4.9. FTP server

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

FTP server

Server status

Enabled

Port

21

Username

ftp

Password

Leave password to blank to keep it unchanged

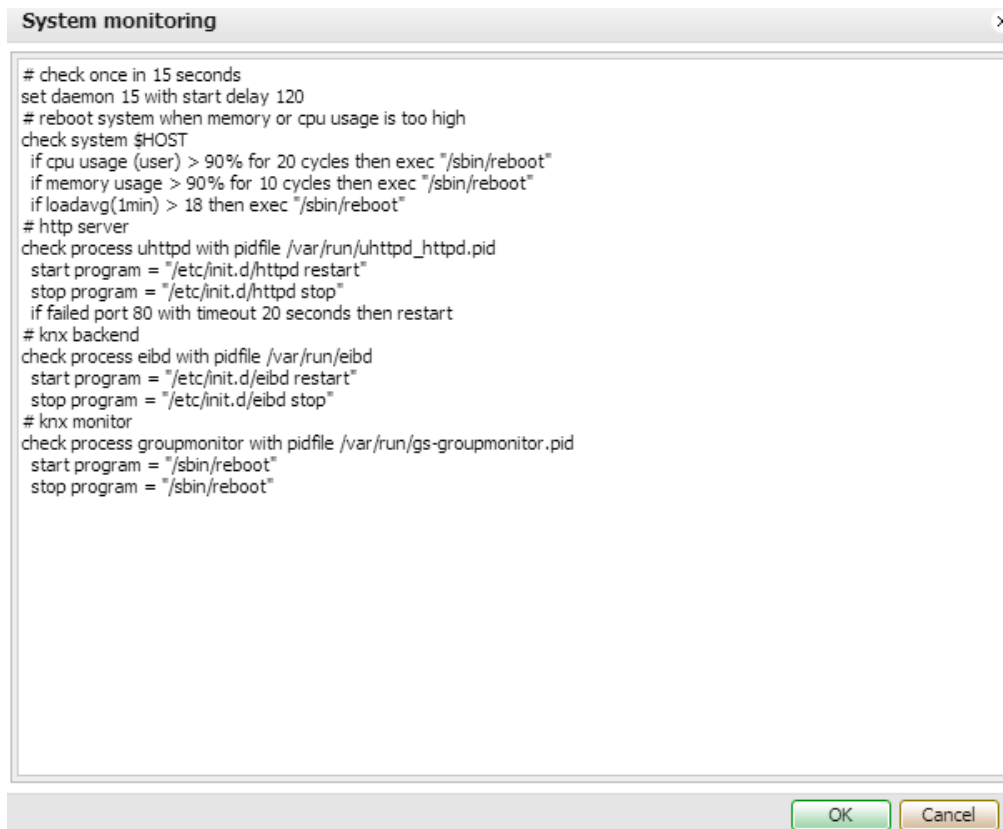
OK

Cancel

- **Server status** – secure tunnel mode
- **Port** – port of the service
- **Username** – login name, *ftp*
- **Password** – password, length 4-20 symbols

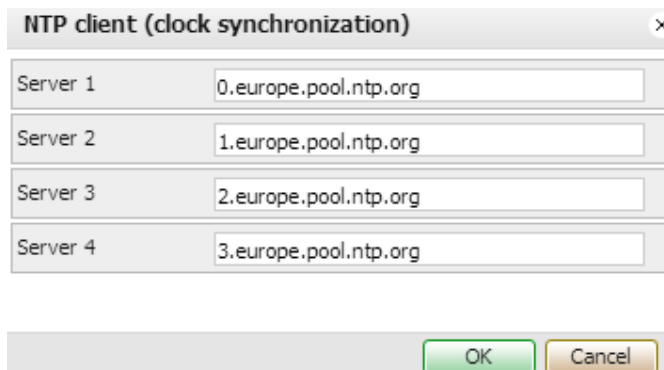
## 4.10. System monitoring

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



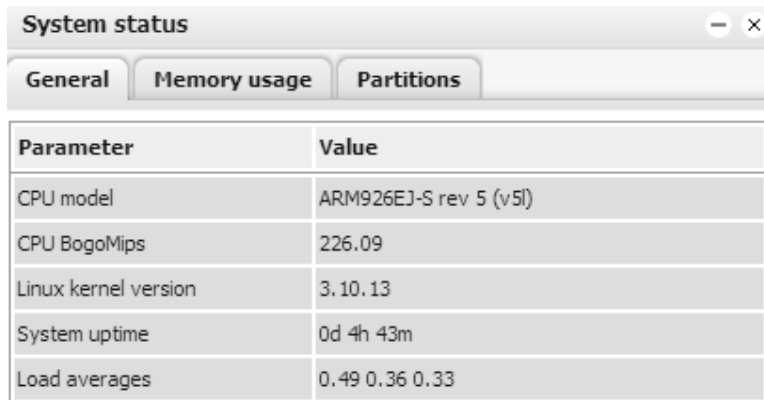
## 4.11. NTP client

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



## 4.12. System status

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

A screenshot of a 'System status' window. It has three tabs: 'General', 'Memory usage', and 'Partitions'. The 'General' tab is selected. Below the tabs is a table with two columns: 'Parameter' and 'Value'.

Parameter	Value
CPU model	ARM926EJ-S rev 5 (v5l)
CPU BogoMips	226.09
Linux kernel version	3.10.13
System uptime	0d 4h 43m
Load averages	0.49 0.36 0.33

## 4.13. Network status

Network overview of IP setting and transferred data can be seen in *Status* → *Network status* window.

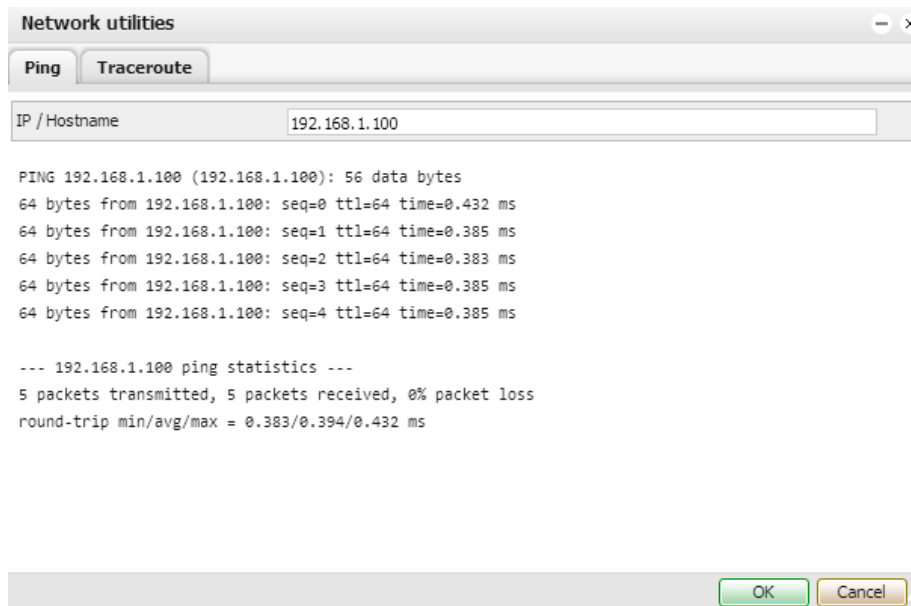
A screenshot of a 'Network status' window. It contains a table with network interface details.

Name	Mac address	IP address	Mtu	TX Bytes	RX Bytes	Errors	
eth0	00:1B:C5:00:13:4D	192.168.1.211	1500	6 MB	6 MB	0 / 0	



## 4.14. Network utilities

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



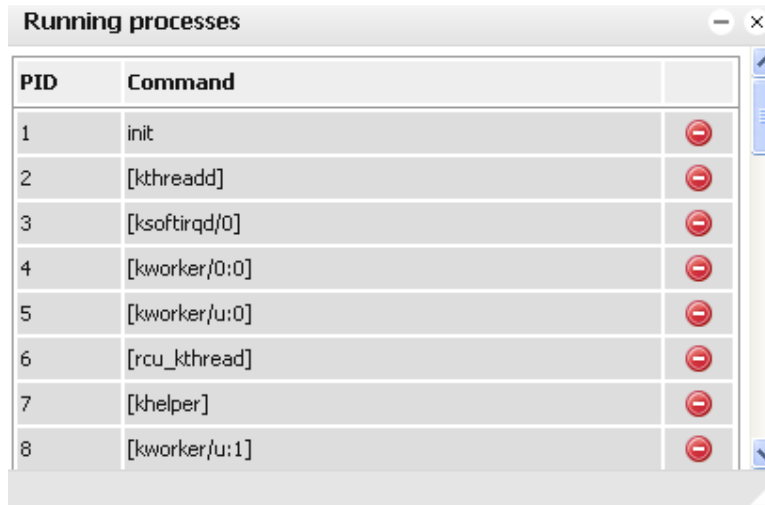
## 4.15. System log

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



## 4.16. 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 third column containing red circular icons with minus signs. 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]	⊖

## 5. 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.

### 5.1. Events

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

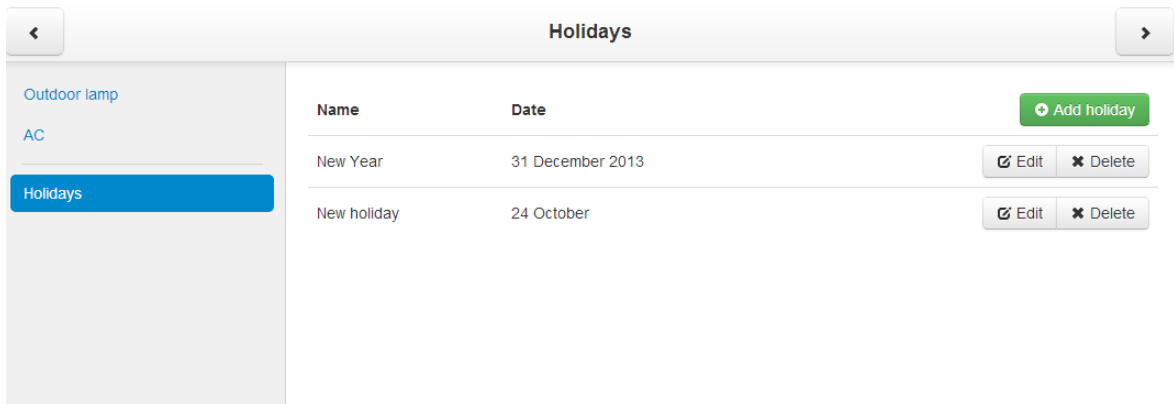
The screenshot shows a web interface for configuring an 'Outdoor lamp' scheduler. The interface is divided into three main sections:

- Left Sidebar:** Contains a blue button labeled 'Outdoor lamp' and two links, 'AC' and 'Holidays', in blue text.
- Central Panel:**
  - At the top, it says 'Status: active, period: 1 January - 31 December' with an 'Edit' button.
  - Below this is a table with two columns: 'Value' and 'Run at'.
  - The table contains two rows:
    - Row 1: 'Light off' | '12:00 Tu-Fr'
    - Row 2: 'Light off' | '13:00 Sa-Su Holiday'
- Right Panel (Add event):**
  - Has a checkbox 'Event is active' which is checked.
  - Below it is a 'Run at' section with two columns of time pickers. The first column shows '12' and the second shows '00'. Above each is an up arrow and below is a down arrow.
  - Below the time pickers are buttons for days of the week: 'Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa', 'Su', and a 'Holiday' button.
  - Below these is a 'Value' section with a dropdown menu currently showing 'Light on'.
  - At the bottom are 'Save' and 'Cancel' buttons.

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

## 5.2. Holidays

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

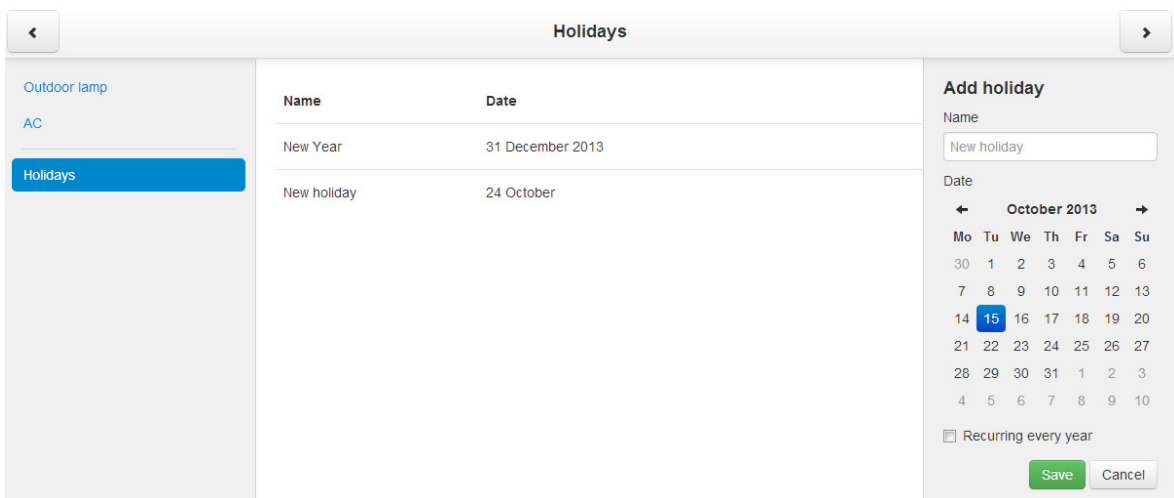


The screenshot shows a web interface titled "Holidays". On the left is a sidebar with a menu containing "Outdoor lamp", "AC", and "Holidays" (which is highlighted in blue). The main area contains a table with two columns: "Name" and "Date".

Name	Date	
New Year	31 December 2013	<a href="#">✎ Edit</a> <a href="#">✕ Delete</a>
New holiday	24 October	<a href="#">✎ Edit</a> <a href="#">✕ Delete</a>

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

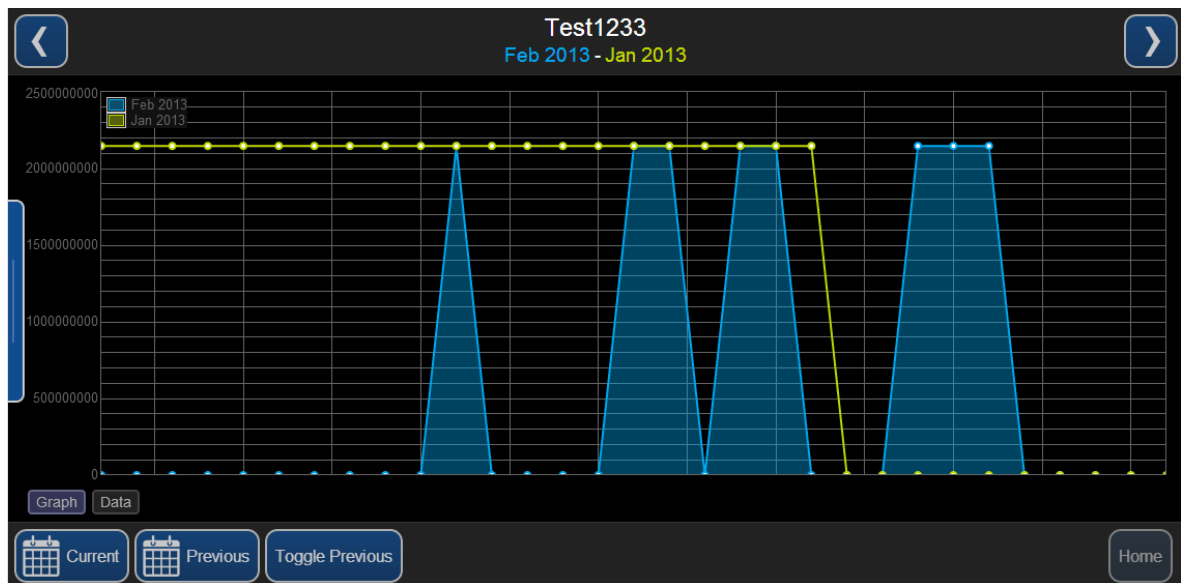
Click on *Add new holiday* button to specify a holiday.



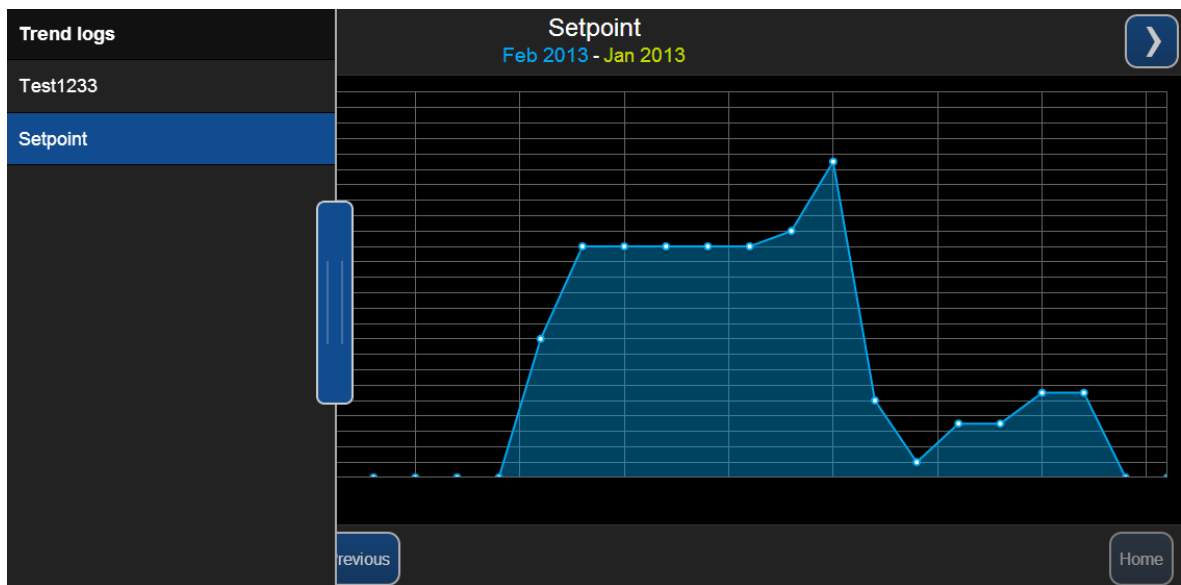
This screenshot shows the same "Holidays" interface, but with the "Add holiday" modal open on the right side. The modal has a "Name" input field containing "New holiday" and a "Date" section. The date section includes a calendar for "October 2013" with days of the week (Mo, Tu, We, Th, Fr, Sa, Su) and dates (1-31). The date "15" is selected. Below the calendar is a checkbox labeled "Recurring every year" which is currently unchecked. At the bottom of the modal are "Save" and "Cancel" buttons.

## 6. Trend logs

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



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

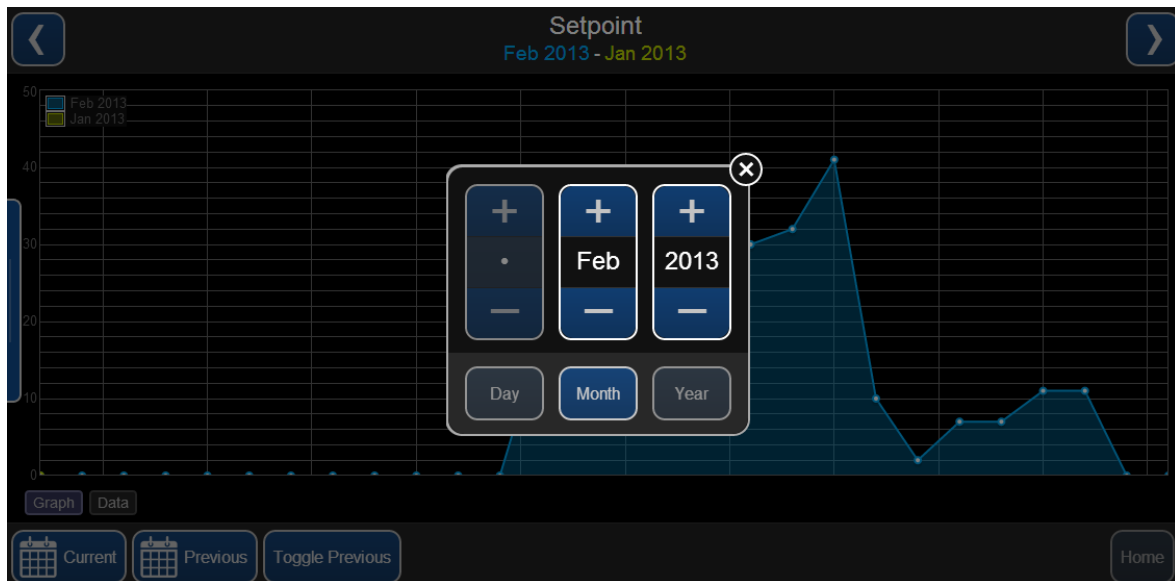


**Current** – Current trend is drawn in blue, you can choose either to show Day, Month or Year view

**Previous** – previous time period, you can choose either to show Day, Month or Year view

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

**Home** – LogicMachine home screen.



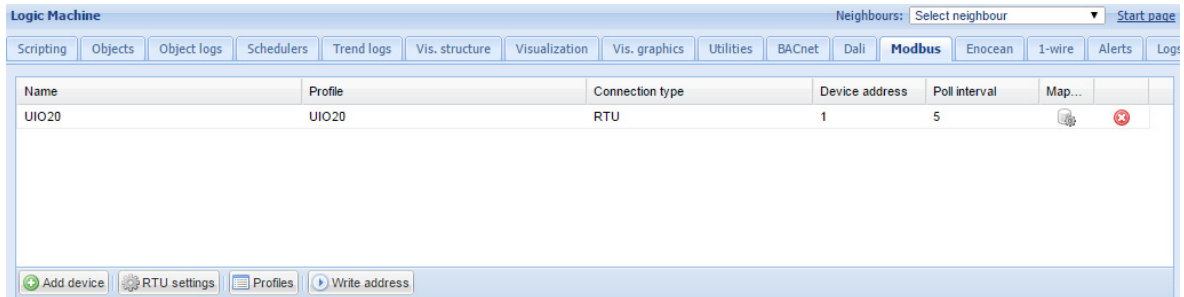
Data points can be shown also in a way of table which can be later exported as CSV file.

The figure shows a table titled "Setpoint" for the period "Feb 2013 - Jan 2013". The table has columns for "Day", "Feb 2013", and "Jan 2013". The data shows a peak in Feb 2013.

Day	Feb 2013	Jan 2013
2	0	0
3	0	
4	0	
5	0	
6	0	
7	0	
8	0	
9	0	
10	0	
11	0	
12	0	
13	0	
14	18	
15	30	
16	30	
17	30	
18	30	

## 7. Modbus RTU/TCP interconnection with LM

Modbus RTU is supported over RS485 interface. 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

### 7.1. 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 **discreteinput** (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,  $\text{value} = \text{value\_base} + \text{value} * \text{value\_multiplier}$  (*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  $\text{key} \rightarrow \text{value}$  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*)

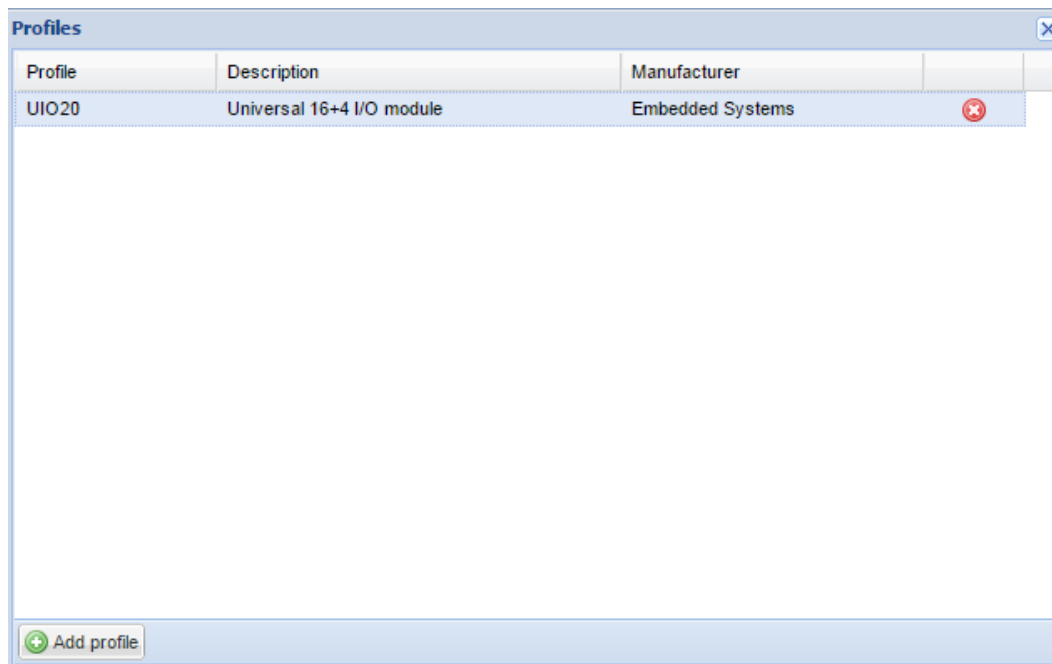
**Address\_scale** – Address of register containing value scale,  $\text{value} = \text{value} * 10^{\text{scale}}$  (*Number*)

**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*)

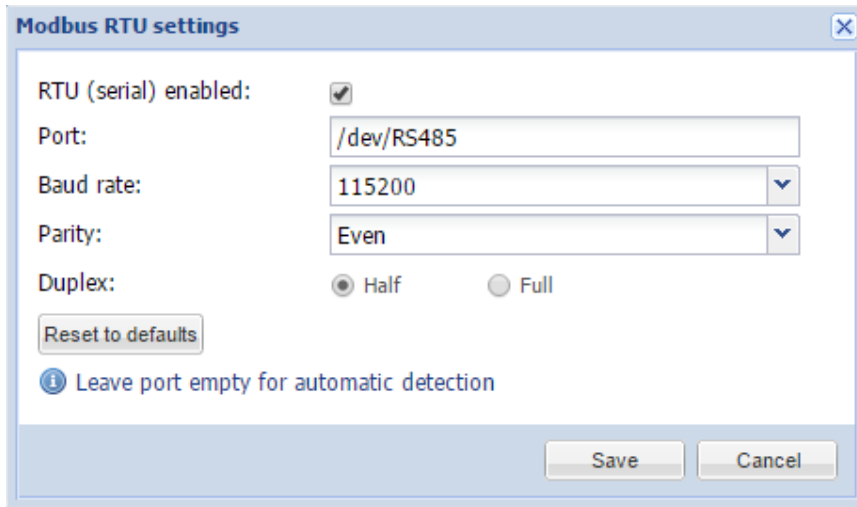
When the Modbus device profile file is created, upload it by clicking on *Profiles* button.



## 7.2. Modbus RTU settings

If the communication is over Modbus RTU protocol (over RS-485 serial port), you should do base serial port settings by clicking on *RTU settings* button.





The 'Modbus RTU settings' dialog box contains the following fields and controls:

- RTU (serial) enabled:** A checkbox that is checked.
- Port:** A text field containing '/dev/RS485'.
- Baud rate:** A dropdown menu showing '115200'.
- Parity:** A dropdown menu showing 'Even'.
- Duplex:** Two radio buttons, 'Half' (selected) and 'Full'.
- Reset to defaults:** A button.
- Information:** A blue icon followed by the text 'Leave port empty for automatic detection'.
- Buttons:** 'Save' and 'Cancel' buttons at the bottom right.

**RTU (serial) enabled** – define either RTU is enabled or not

**Port** – port name. In case of several RS-485 ports on the device, the name of the port is incremented by one, e.g. RS485-1, RS485-2, RS485-3 etc.

**Baud rate** – baud rate for the connection

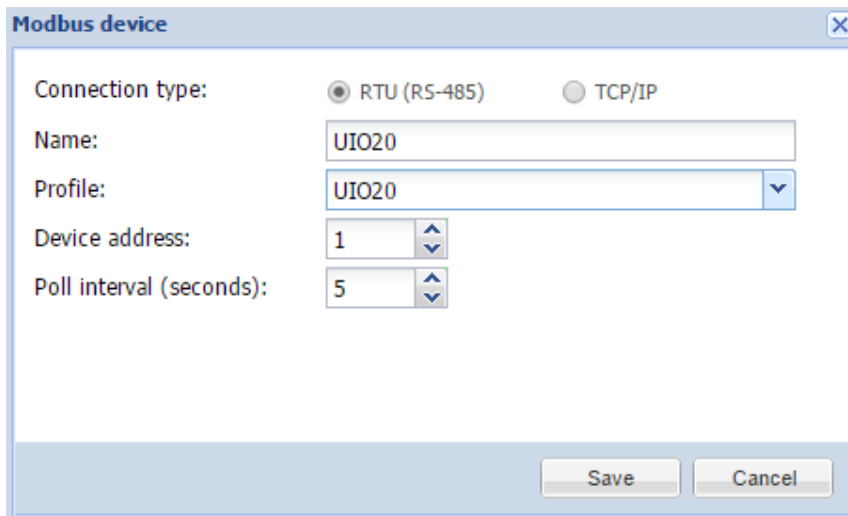
**Parity** – parity for the connection

**Duplex** – specify either it is *half* or *full* duplex

**Reset to defaults** – reset RTU settings to defaults

### 7.3. Adding Modbus device

Once profiles are defined and RTU settings set, add Modbus device by clicking *Add device* button.



The 'Modbus device' dialog box contains the following fields and controls:

- Connection type:** Two radio buttons, 'RTU (RS-485)' (selected) and 'TCP/IP'.
- Name:** A text field containing 'UIO20'.
- Profile:** A dropdown menu showing 'UIO20'.
- Device address:** A spinner box showing '1'.
- Poll interval (seconds):** A spinner box showing '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

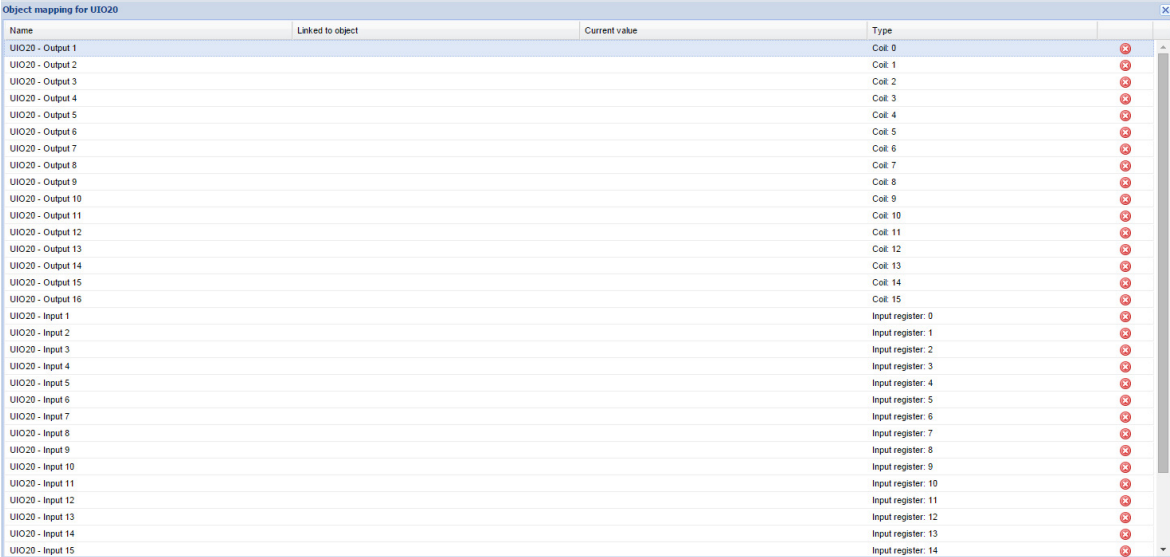
**Device address** – device address

**Poll interval (seconds)** – interval to poll the device

**IP** – IP address of the device in case Modbus TCP is used

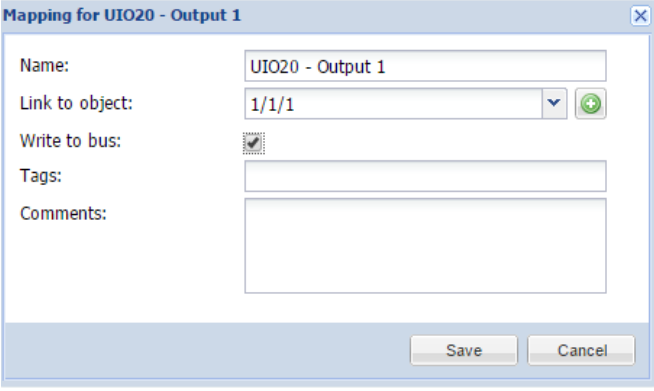
**Port** – Communication port of the device in case Modbus TCP is used

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.



**Mapping for UIO20 - Output 1**

Name: UIO20 - Output 1

Link to object: 1/1/1

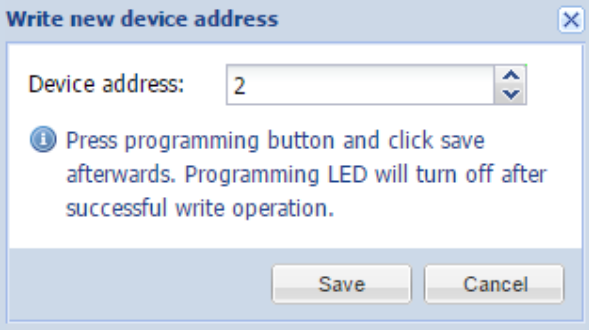
Write to bus: ☒

Tags:

Comments:


Save Cancel

#### 7.4. Program address for UIO20 Modbus device



**Write new device address**

Device address: 2

 Press programming button and click save afterwards. Programming LED will turn off after successful write operation.

Save Cancel

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.

## 7.5. 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 specific function
51. function mbproxy:request(fn, ...)
52.     local res, err = self.rpc:request({
53.         fn = fn,
54.         params = { ... },
55.         slaveid = self.slaveid or 0,
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('/dev/RS485', 38400, 'E', 8, 1, 'H')*

set baudrate and other serial port parameters

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.      ifnot mb then
3.      require('luamodbus')
4.      mb = luamodbus.rtu()
5.      mb:open('/dev/ttyS2', 38400, 'E', 8, 1, 'H')
6.      mb:connect()
7.
8.      -- slave id
9.      mb:setslave(10)
10.
11.     -- init slave storage for coils, discrete inputs, holding registers and input registers
12.     mb:setmapping(10, 10, 10, 10)
13.
14.     -- coil write callback
15.     mb:setwritecoilcb(function(coil, value)
16.     if coil == 0 then
17.         grp.write('1/1/1', value, dt.bool)
18.     else
19.         alert('coil: %d = %s', coil, tostring(value))
20.     end
21.     end)
22.
23.     -- register write callback
24.     mb:setwriteregistercb(function(register, value)
25.     if register == 0 then
26.         -- send value limited to 0..100
27.         grp.write('4/1/5', math.min(100, value), dt.scale)
28.     else
29.         alert('register: %d = %d', register, value)
30.     end
31.     end)
32.     end
33.
34.     -- server part init
35.     ifnot server then
36.     require('rpc')
37.
38.     -- incoming data handler
39.     local handler =function(request)
40.     local fn, res
41.
42.     fn =tostring(request.fn)
```

```

43.
44.     ifnot mb[ fn ]then
45.         return{nil, 'unknown function ' .. fn }
46.     end
47.
48.     iftype(request.params)=='table'then
49.         table.insert(request.params, 1, mb)
50.         res ={ mb[ fn ](unpack(request.params))}
51.     else
52.         res ={ mb[ fn ](mb)}
53.     end
54.
55.     return res
56. end
57.
58.     server = rpc.server('127.0.0.1', 28002, 'mbproxy', handler, 0.01)
59. end
60.
61. mb:handleslave()
62. 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

```

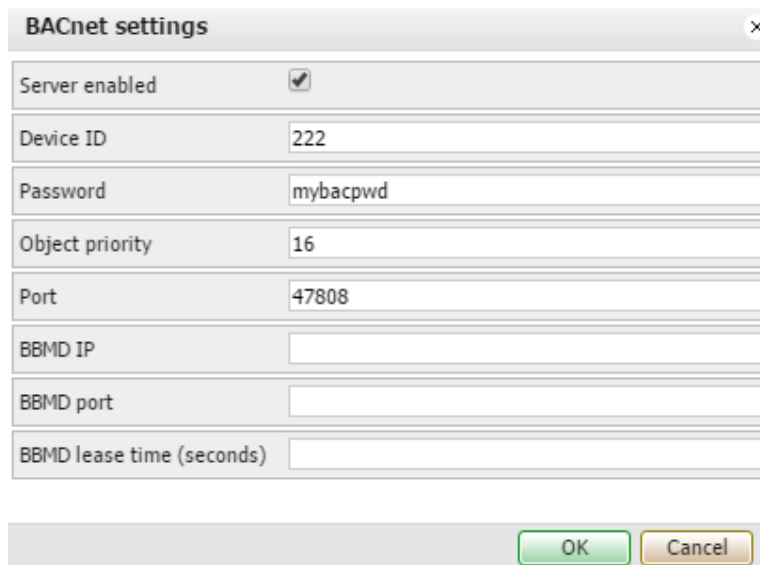
1. value = event.getvalue()
2. mb = mbproxy.new()
3. mb:setregisters(5, value)

```

## 6. BACnet IP interconnection with LM4

### 6.1. BACnet server mode: transparent data transfer to BACnet network

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



BACnet settings	
Server enabled	<input checked="" type="checkbox"/>
Device ID	222
Password	mybacpwd
Object priority	16
Port	47808
BBMD IP	
BBMD port	
BBMD lease time (seconds)	

OK Cancel

**Server enabled** – specify if BACnet server is enabled or not

**Device ID** – device ID in BACnet network

**Password** – device password

**Object priority** – object priority

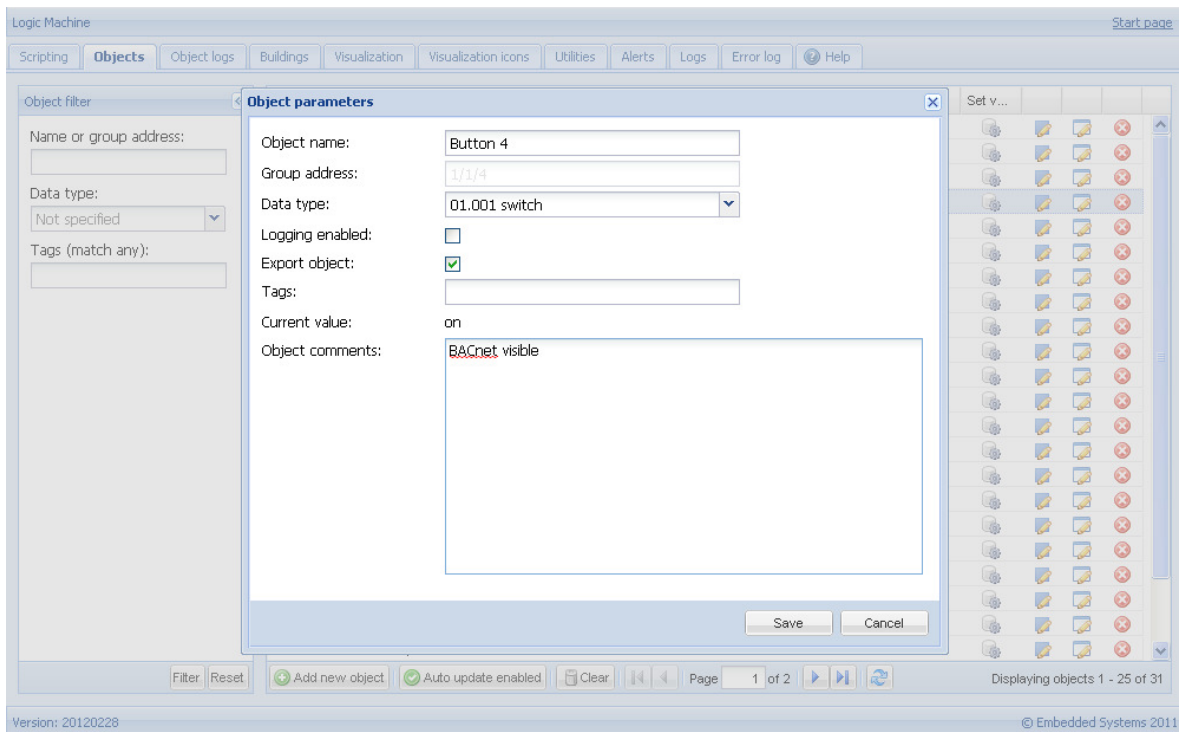
**Port** – port number

**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



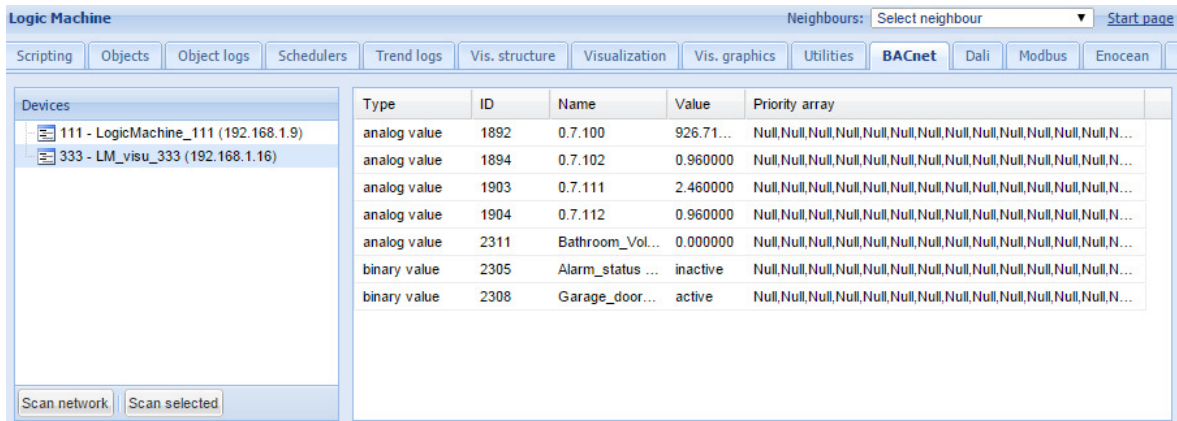
In *System Configuration* → *Network* → *BACnet objects* you can see marked objects on LogicMachine which are sent to BACnet network.

BACnet objects			
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

## 6.2. 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)*

## 7. 1-wire configuration

1-wire is a bus technology which is built based on client-server topology and allowing to connect up to 300 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.

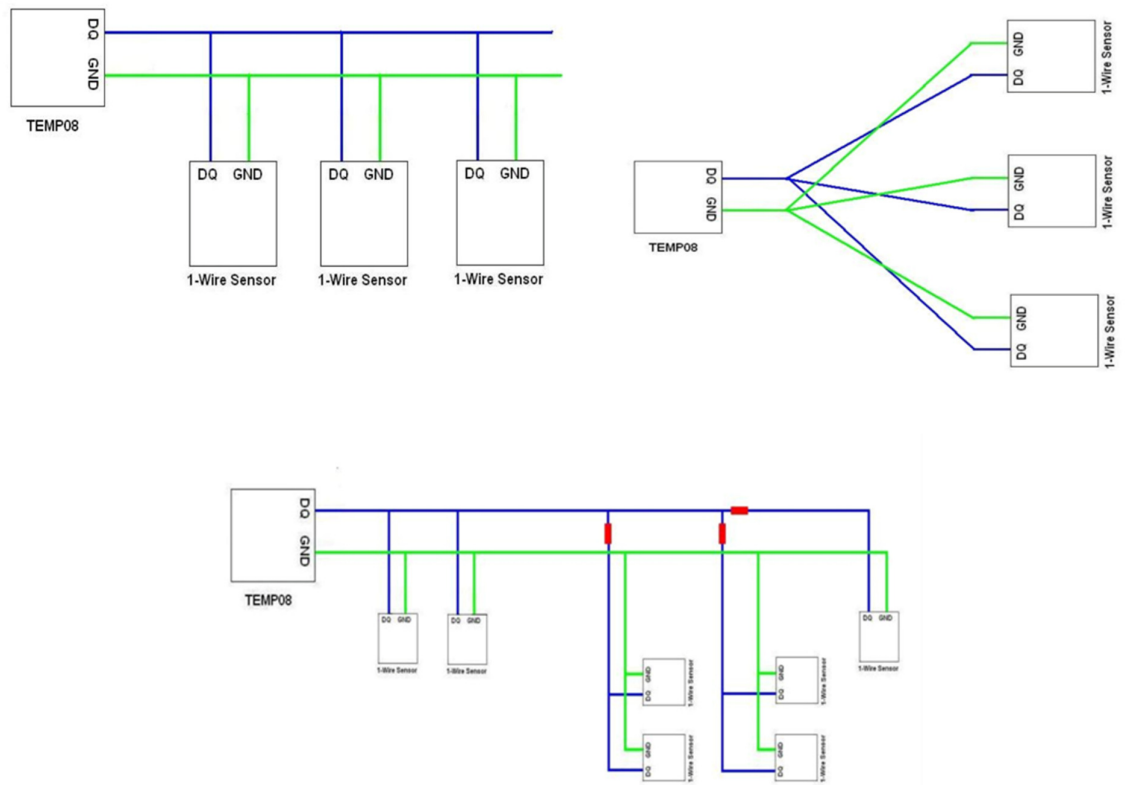
### Advantages of 1-wire over KNX:

- No need in ETS
- Very cost-effective
- You can use the same wiring as KNX does and connect all standard sensors

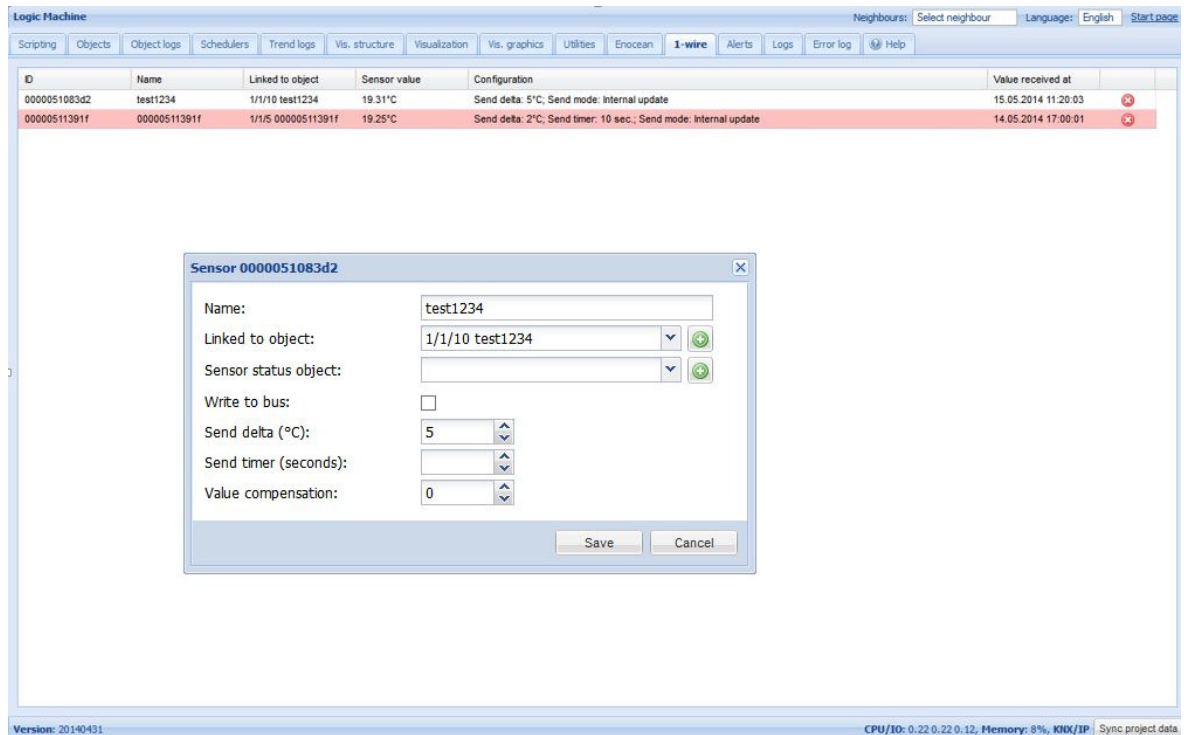
### Advantages of 1-wire over resistive sensors:

- 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 LogicMachine4



**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

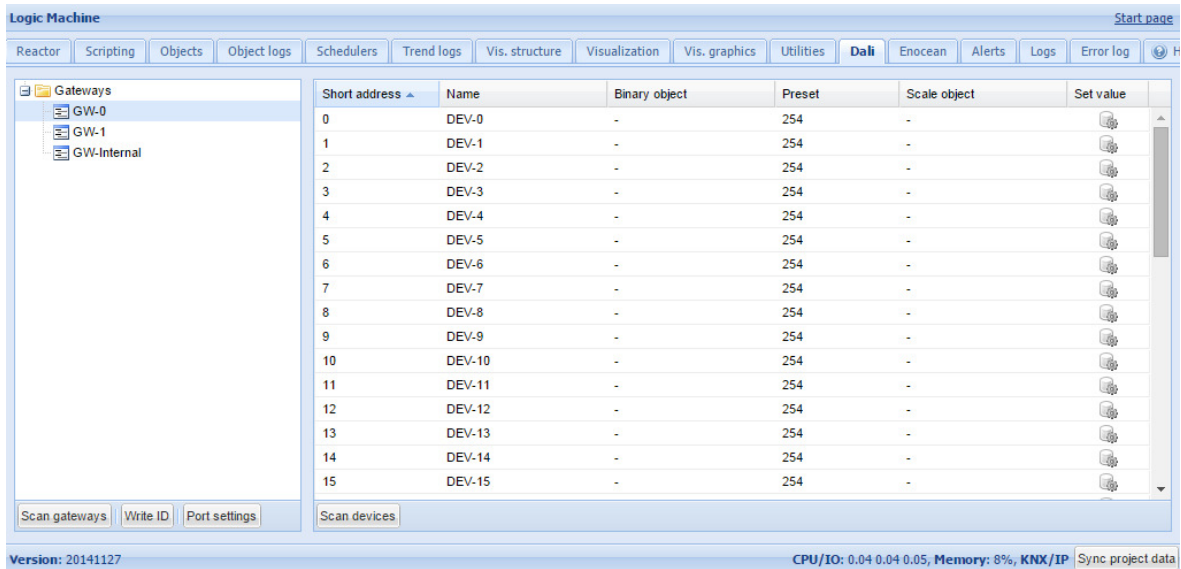
**Send delta** – define either to send delta of temperature sensor

**Send timer (seconds)** – define interval in which send the measurement

**Value compensation** – compensate value of the reading of temperature

## 8. DALI configuration

LogicMachine4 and Reactor V2 have DALI Master built-in. We recommend to connect no more than 32 ballasts to one DALI line. If more ballasts are necessary to connect, you can use external DALI-RS-485 interfaces and connect to RS-485 port.

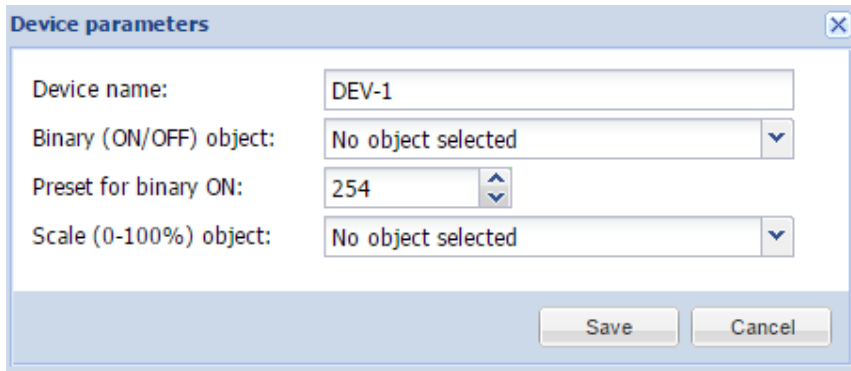


- **Scan gateways** - scans for currently connected gateways, address mapping for missing devices is deleted automatically
- **Write ID** - allows setting a unique address for each gateway
- **Scan devices** - scans for currently connected DALI devices to the selected gateway, assigns short address automatically. You can also set not to overwrite existing addresses during scan
- **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.

## 8.1. 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 

## 8.2. Access DALI bus from scripts

If you want to access DALI devices from other scripts, you can use **dalicmd** function.

*dalicmd(gateway, command, parameters)*

### Parameters:

*gateway* - gateway id (0..63)

*command* - DALI command to execute

*parameters* - Lua table:

*addrtype* - address type, only required for addressable commands, possible values:

*short group broadcast*

*address* - short or group address

*value* - additional value to send

### 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 })
```

## DALI commands

In the list below please see description of parameters of function **dalicmd**.

**Command** – dalicmd() parameter command

**Description** – description of command

**Addressable** – + means that this is addressable command, dalicmd() requires existence of parameter cmddats and addrtype\_V 'broadcast'. Empty field means that the command is non-addressable and parameter cmddats may be unused

**Value** – interval of values of parameter value\_V.

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	+	+	

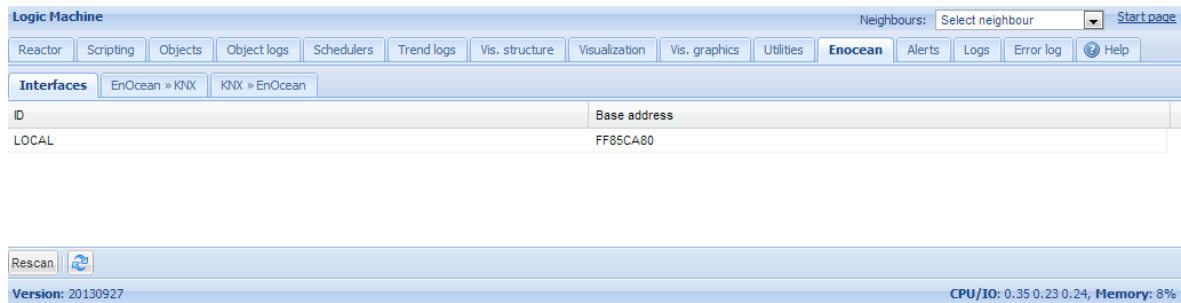
queryversion	query version number	+	+	
querydtr	query content dtr	+	+	
querydevicetype	query device type	+	+	
queryphysicalmin	query physical minimum level	+	+	
querypowerfailure	query power failure	+	+	
queryactual	query actual level	+	+	
querymax	query max level	+	+	
querymin	query min level	+	+	
querypoweron	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)	+	+	
queryrandomaddrh	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
searchaddrh	set search address (m)			0..255
searchaddrh	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

## 9. EnOcean interconnection with LogicMachine

Logic Machine3 Reactor and Reactor V2 have EnOcean transceiver built-in with no limitation on supported count of devices. You need to use external USB gateway for LogicMachine4 to activate this functionality.

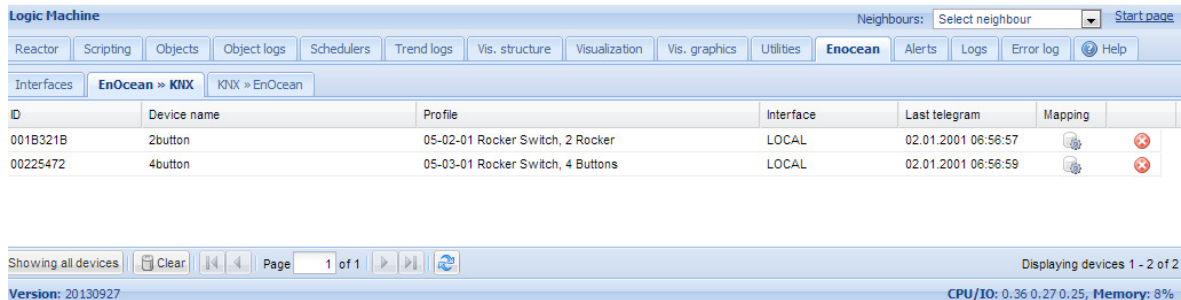
### 9.1. EnOcean interfaces

EnOcean interface Base address can be found in *EnOcean* → *Interfaces* tab.

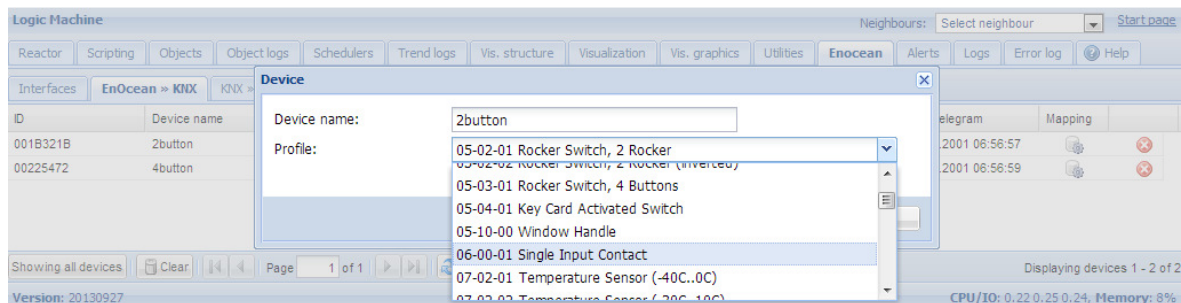


### 9.2. EnOcean to KNX mapping

All telegrams received from EnOcean devices appears in *EnOcean* → *KNX* section.

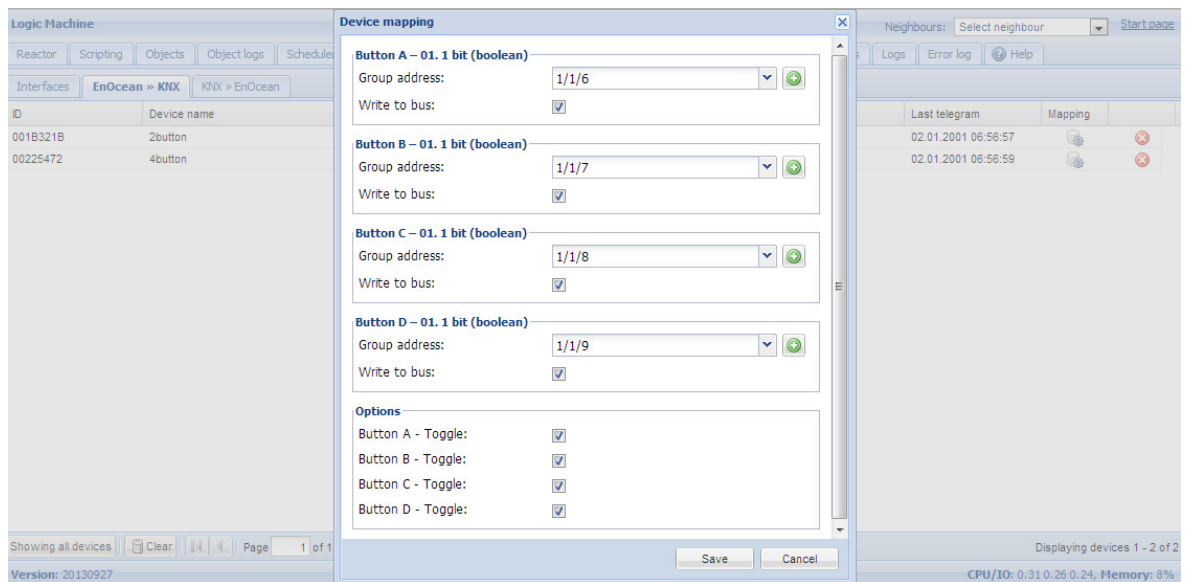


Once some specific device has to be mapped to KNX, the corresponding row has to be clicked and profile has to be chosen. There are all main profiles predefined in the list.

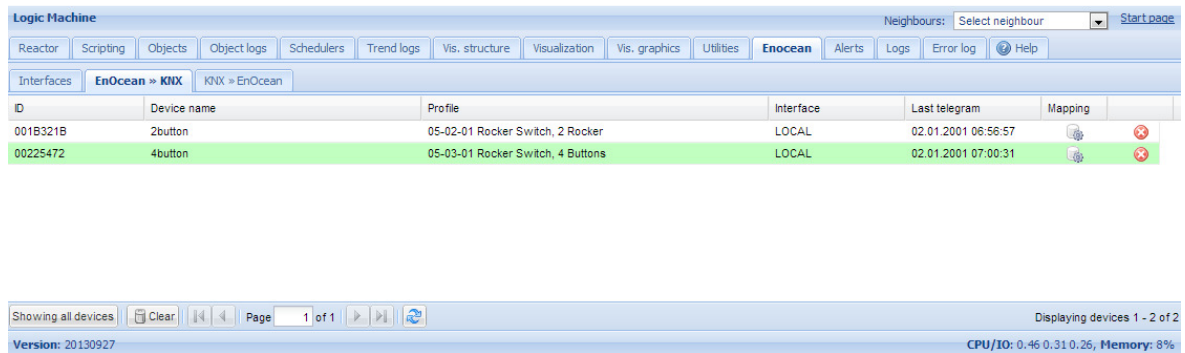


Once the device profile is set, you can map functionality of the specific device to KNX group addresses by clicking on *Mapping* icon.

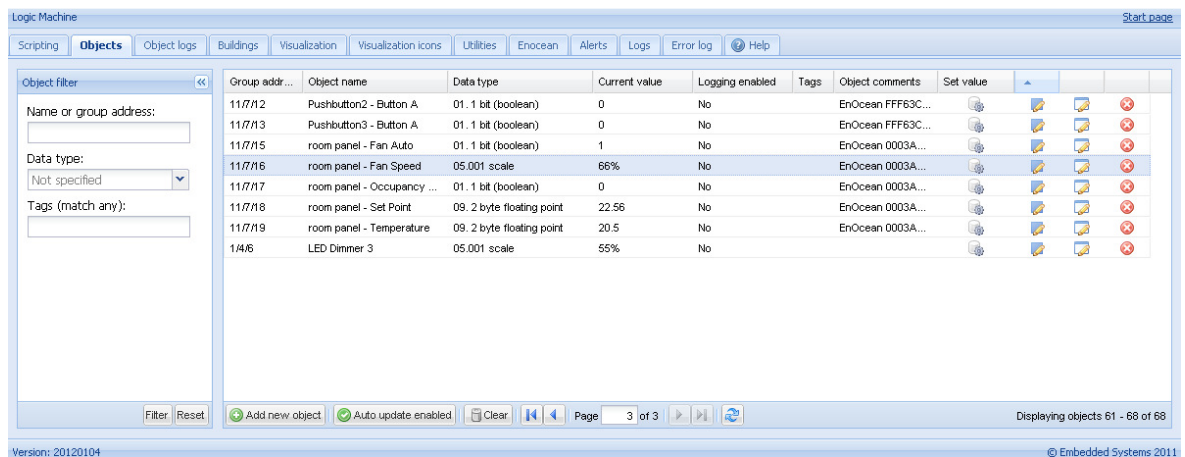




When EnOcean gateway received telegram from specific device, the respective row gets light green.

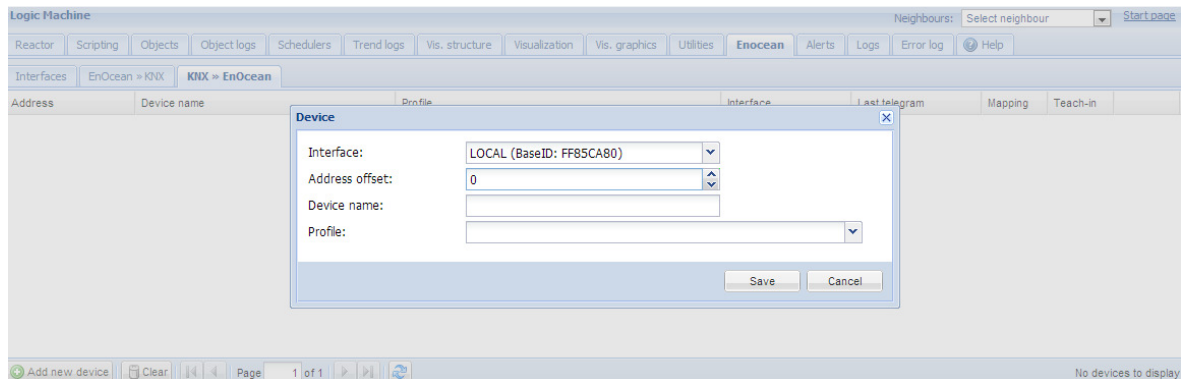


Respective KNX group addresses get updated with the new values.



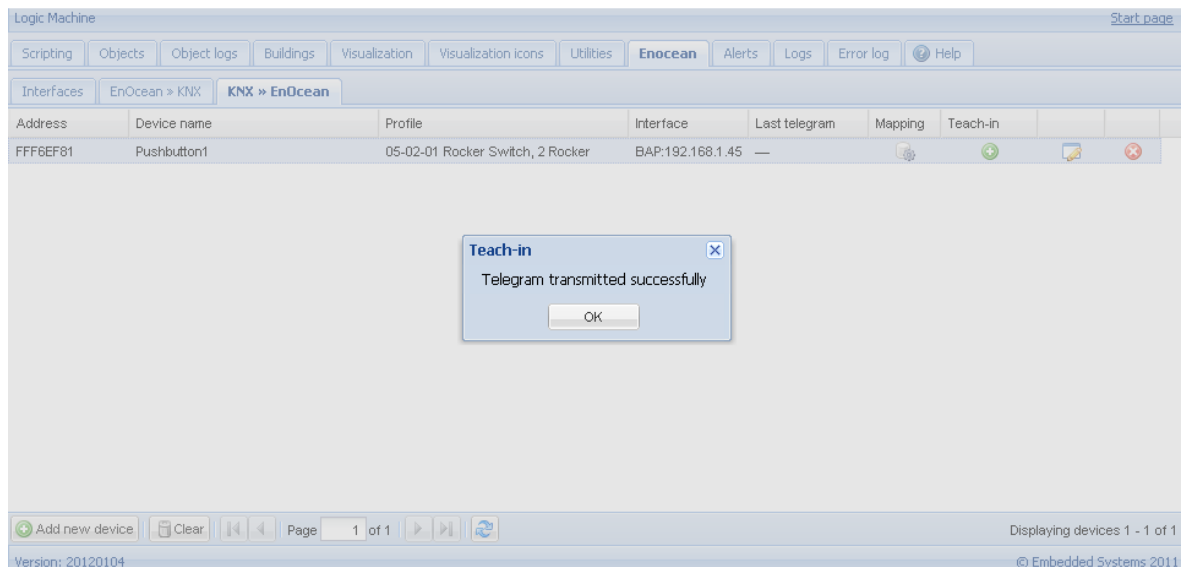
### 9.3. KNX to EnOcean mapping

You should click on Add new device button to add EnOcean device which will be communicated from specific KNX object.

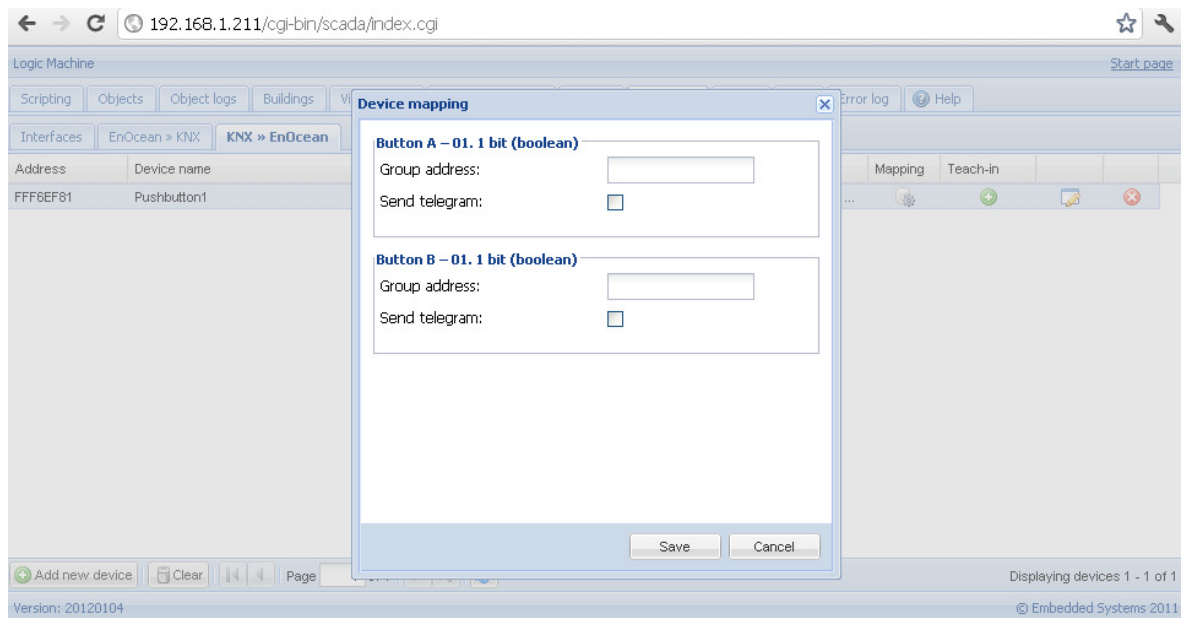


Once the device is added, you should pair it with specific device in EnOcean network, press Tech-in button.

**Note!** EnOcean device should be set in learning mode in order to pair it successfully.



Further this device can be mapped with specific KNX addresses. When KNX object value will be updated, the telegram will be sent to respective EnOcean device.





## 10. CEC/HDMI integration with LM4

Consumer Electronics Control (CEC) is an HDMI feature designed to allow the user to command and control up-to 15 CEC-enabled devices, that are connected through HDMI, by using only one of their remote controls (for example by controlling a television set, set-top box, and DVD player using only the remote control of the TV. In comparison with InfraRed, it is possible, for example, to switch off/on the power of TV.

As HDMI has limited cable length, we have developed additional CEC/HDMI adapter which is located near the CEC-compatible control device like TV and wire is put to LogicMachine4 (not limited in length). CEC adapter is available by request when purchasing LM4.

The most easiest and quickest way to check if your device is CEC compatible is to use Google Chromecast device.

### 10.1. CEC function

Add the following function in *Scripting* -> *Common Functions*

```
1. cec = {}
2.
3. cec.init = function()
4.   if not cec.port then
5.     require('serial')
6.     cec.port = serial.open('/dev/ttyAPP0')
7.     cec.port:flush()
8.   end
9. end
10.
11. cec.send = function(data)
12.   local cmd, res
13.
14.   if type(data) ~= 'string' then
15.     return nil, 'invalid data'
16.   elseif #data < 2 then
17.     return nil, 'data too short'
18.   elseif #data > 30 then
19.     return nil, 'data too long'
20.   end
21.
22.   cec.init()
23.
24.   cmd = string.char(0x00, 0xA0, #data) .. data
25.   cec.port:write(cmd)
26.
27.   res = cec.port:read(2, 0.2)
28.   if type(res) == 'string' and #res == 2 and res:byte(1) == 0x5A then
29.     if res:byte(2) == 0x10 then
30.       return true
```

```

31.         elseif res:byte(2) == 0x40 then
32.             return nil, 'transmitter busy'
33.         end
34.     end
35.
36.     return nil, 'reply error'
37. end
38.
39. cec.poll = function()
40.     local cmd, len, frame, count
41.
42.     cec.init()
43.
44.     cmd = string.char(0x00, 0xA0, 0x01, 0xAA)
45.     cec.port:write(cmd)
46.
47.     res = cec.port:read(2, 0.2)
48.     if type(res) == 'string' and #res == 2 and res:byte(1) == 0xA5 then
49.         len = res:byte(2)
50.
51.         if len == 0 then
52.             return false, 0
53.         end
54.
55.         res = cec.port:read(len, 0.2)
56.
57.         if type(res) == 'string' and #res == len then
58.             count = res:byte(1) - 1
59.             return res:sub(2), count
60.         end
61.     end
62.
63.     return nil, 'reply error'
64. end

```

## 10.2. CEC command generator

CEC command generator is available here: <http://www.cec-o-matic.com/>

As Source choose TV. The ID which you get, use in as shown in the example below in form 0xID\_nr.

my crystal clear HDMI-CEC message

source				destination			
TV	Recording 1	Recording 2	Tuner 1	TV	Recording 1	Recording 2	Tuner 1
Playback 1	Audio system	Tuner 2	Tuner 3	Playback 1	Audio system	Tuner 2	Tuner 3
Playback 2	Playback 3	Tuner 4	Playback 3	Playback 2	Playback 3	Tuner 4	Playback 3
Reserved (C)	Reserved (D)	Reserved (E)	Unregistered	Reserved (C)	Reserved (D)	Reserved (E)	Broadcast

End-user features   Supporting features   General protocol   All messagesclassified according to the HDMI specification

▶ One Touch Play

▶ One Touch Record

▼ Deck Control

Enables a device to control (e.g. play, fast forward etc.) and interrogate a Playback Device (a deck).

Deck StatusGive Deck StatusDeck ControlPlay

message ID : 1A - Used to request the status of a device, regardless of whether or not it is the current active source (Directly addressed)

parameters

Status Request

On ▼

## 10.3. Common commands

Send stand-by to all devices

```
cmd = string.char(0xBF, 0x36)
cec.send(cmd)
```

Returns single telegram from buffer and count of telegrams stored (up to 16). Returns false, 0 if buffer is empty

```
cec.poll()
```

## 11. DMX interconnection with LM4

DMX protocol support is realized upon RS485 serial port.

### Usage

```
d =DMX:init(parameters)
d:run()
```

### Parameters

- **channels** – (optional, defaults to 3) number of DMX channels to use
- **resolution** – (optional, defaults to 20) number of DMX updates per second. Larger value gives smoother transitions, but increases CPU usage
- **transition** – (optional, defaults to 2) soft transition time in seconds
- **port**– (optional) RS-485 port name, usually you don't have to change this value

### Common function

The following program has to be added in Common functions library.

```
DMX = {
  -- default params
  defaults = {
    -- storage key
    skey = 'dmx_chan_',
    -- RS-485 port
    port = '/dev/ttyS2',
    -- number of calls per second
    resolution = 20,
    -- total number of channels to use
    channels = 3,
    -- transition time in seconds, does not include DMX transfer time
    transition = 2,
  },
  -- value setter
  set = function(i, v)
    -- validate channel number
    if type(i) ~= 'number' and i >= 1 and i <= 512 then
      -- validate channel value
      if type(v) ~= 'number' and v >= 0 and v <= 255 then
        storage.set(DMX.defaults.skey .. i, v)
      end
    end
  end
}

-- DMX init, returns new DMX object
function DMX:init(params)
  require('luadm')

  local n = setmetatable({}, { __index = DMX })
  local k, v

  -- set user parameters
  n.params = params

  -- copy parameters that are set by user
  for k, v in pairs(DMX.defaults) do
    if n.params[k] == nil then
      n.params[k] = v
    end
  end

  n:reset()

  return n
end

function DMX:reset()
  local err, chan

  self.dm, err = luadm.open(self.params.port)

  -- error while opening
  if err then
    os.sleep(1)
    error(err)
  end
end
```



```

-- set channel count
self.dm:setcount(self.params.channels)

-- number of transaction ticks
self.ticks = math.max(1, self.params.transition * self.params.resolution)

-- calculate sleep time
self.sleep = 1 / self.params.resolution

-- reset channel map
self.channels = {}

-- fill channel map
for chan = 1, self.params.channels do
self.channels[ chan ] = { current = 0, target = 0, ticks = 0 }

    -- turn off by default
    storage.set(self.params.skey .. chan, 0)
    self.dm:setchannel(chan, 0)
end
end

-- get new values
function DMX:getvalues()
local chan, val

    -- check for new values for each channel
    for chan = 1, self.params.channels do
        val = storage.get(self.params.skey .. chan)

        -- target value differs, set transcation
        if val ~= self.channels[ chan ].target then
            self.channels[ chan ].target = val
            self.channels[ chan ].delta = (self.channels[ chan ].target - self.channels[ chan ].current) / self.ticks
            self.channels[ chan ].ticks = self.ticks
        end
    end

-- main loop handler
function DMX:run()
local i, bs, bm, as, am, delta
local res = self.params.resolution

if not self.calibrated then
bs, bm = os.microtime()
end

self:getvalues()

-- transition loop
for i = 1, res do
self:step()
    self.dm:send()

    -- wait until next step
    os.sleep(self.sleep)
end

-- calibrate delay loop to match 1 second
if not self.calibrated then
as, am = os.microtime()
delta = (as - bs) + (am - bm) / 1000000

if delta > 1.05 then
    self.sleep = self.sleep - math.max(10, self.sleep / res)
else
    self.calibrated = true
end
end

-- single transition step
function DMX:step()
local chan, t

    -- transition for each channel
    for chan = 1, self.params.channels do
        t = self.channels[ chan ].ticks

        -- transition is active
        if t > 0 then
            t = t - 1

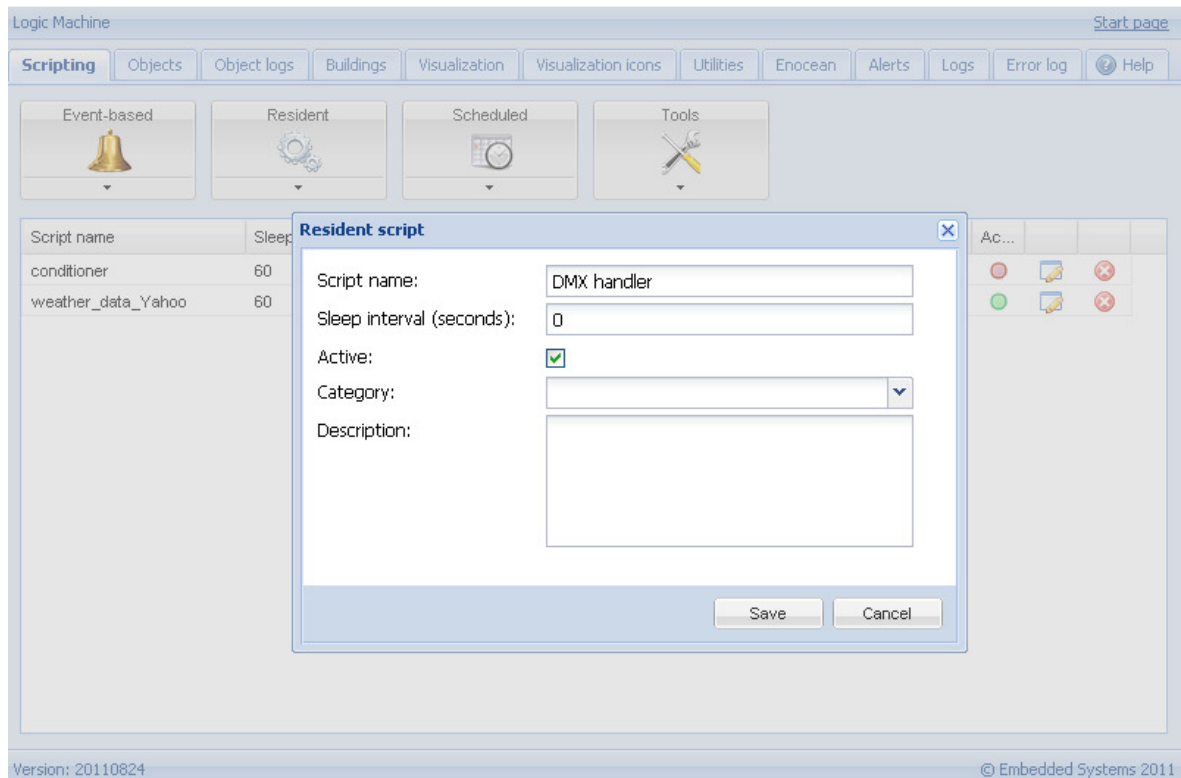
            self.channels[ chan ].current = self.channels[ chan ].target - self.channels[ chan ].delta * t
            self.channels[ chan ].ticks = t

            self.dm:setchannel(chan, self.channels[ chan ].current)
        end
    end
end
end

```

## DMX handler programs

DMX handler should be placed inside a resident script. Sleep time interval must be set to 0.



Once the resident script is added we can add the program source in Script Editor

```

1. ifnot d then
2.   d =DMX:init({
3.     channels = 3,
4.     transition = 2,
5.   })
6. end
7.
8. d:run()

```

### Setter (used in other scripts)

```
DMX.set(channel, value)
```

- **channel**– DMX channel number [1..512]
- **value** – DMX channel value [0..255]

## 11.1. Examples

Predefined scene example: The following example should be placed inside a resident script. Sleep time defines scene keep time (at least 1 second).

```
1. ifnot 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.fori, 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.fori= offset, channels do
9.  v =math.random(0, (steps - 1))* 255 /(steps - 1)
10.DMX.set(i, math.floor(v))
11.end

```

## 12. 3G modem connection with LM4

LogicMachine4 has standard 3G modem driver built-in (Huawei and other vendor support). Currently this can be used for SMS notifications only – receiving and sending commands. The modem has to be plugged into any of USB ports of LM4 and it starts operating immediately. 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.

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' and 1 or 0
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.ifnotposthen
29.returnfalse
```

```

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. returnfalse
37. end
38. -- try decoding value
39. jvalue=json.pdecode(value)
40.     value =jvalue ~=nilandjvalueor 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. ifobjthen
48. obj:read()
49. os.sleep(1)
50. -- read new value
51.     value =grp.getvalue(name)
52. -- got value, send response
53. if value ~=nilthen
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. iftable.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. ifpincode 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.

## **12.1. 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

## 12.2. 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.localmsg='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)
```

## 12.3. 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))
```



12.4.

## 13. HDL protocol integration in LogicMachine4

**Note!** Please contact Embedded Systems team to receive a special package to integrate HDL support into your LM4. Once you have the file, add it in *Network configuration -> System -> Packages*.

### 13.1. HDL function

Add HDL script in *Scripting -> Tools -> User function library*

```
1.HDL ={
2.-- destination ip
3.dstip='192.168.1.7',
4.-- packet constant data
5.  magic ='HDLMIRACLE',
6.lcode=string.char(0xAA, 0xAA),
7.-- source device settings
8.srcsubnet=1,
9.srcdevice=254,
10.devicetype= 0xFFFE,
11.-- command types
12.cmd={
13.chanreg= 0x0031, -- single channel regulate
14.chanregreply= 0x0032, -- single channel regulate answerback
15.chanstat= 0x0033, -- read status of single channel targets
16.chanstatreply= 0x0034, -- single channel targets status answerback
17.}
18.}
19.
20.HDL.init=function()
21.require('json')
22.require('crc16')
23.require('socket')
24.
25.localip, chunk, chunks, data
26.-- read interface data
27.  data =json.pdecode(io.readproc('if-json'))
28.
29.ifnot data ornot data.eth0 then
30.error('cannot get interface data')
31.end
32.
33.-- ip header
34.HDL.iphdr=''
```

```

35. -- broadcast address
36. HDL.bcast= data.eth0.bcast
37.
38. -- split ip address into chunks
39. chunks= data.eth0.inetaddr:split('.')
40.
41. -- add ip address chunks
42. for i= 1, 4 do
43.     chunk =tonumber(chunks[i])
44. HDL.iphdr=HDL.iphdr ..string.char(chunk)
45. end
46. end
47.
48. HDL.decode=function(packet)
49. local len, data, src, crc
50.
51. -- primary header
52. if packet:sub(5, 14) ~=HDL.magic then
53. return nil, 'magic'
54. end
55.
56. -- Leading code
57. if packet:sub(15, 16) ~=HDL.lcode then
58. return nil, 'lcode'
59. end
60.
61. -- get data Length and check against
62. len=packet:byte(17)
63. if len and len + 16 ~=packet:len() then
64. return nil, 'len'
65. end
66.
67. -- get packet data and check crc
68. data =packet:sub(17, len + 14)
69. crc=packet:byte(len + 15)* 0x100 + packet:byte(len + 16)
70. if crc16(data) ~=crc then
71. return nil, 'crc'
72. end
73.
74. -- return parsed packet

```

Change HDL parameters in the function to correct ones

```

HDL = {
  -- destination ip
  dstip = '192.168.1.7',
  -- packet constant data
  magic = 'HDLMIRACLE',
  lcode = string.char(0xAA, 0xAA),
  -- source device settings
  srcsubnet = 1,
  srcdevice = 254,
  devicetype = 0xFFFE,
  -- command types
  cmd = {
    chanreg = 0x0031, -- single channel regulate
    chanregreply = 0x0032, -- single channel regulate answerback
    chanstat = 0x0033, -- read status of single channel targets
    chanstatreply = 0x0034, -- single channel targets status answerback
  }
}

```

## 13.2. Usage example – HDL dimmer control

**Task** of this example is to change HDL dimmer value on specific KNX group address change.

- Add new object in Objects tab
- Add Event-based script which will monitor newly created object
- In Scripting Editor specify the following code for this script

```

1. local value = dpt.decode(event.datahex, dt.scale)
2. HDL.chanreg(1, 12, 1, value, 1)

```

### HDL.chanreg function description

```
HDL.chanreg(dstsubnet, dstdevice, chan, value, delay)
```

#### **Parameters:**

- *dstsubnet* – device subnet
- *dstdevice* – device address
- *chan* – channel number (1..n)
- *value* – value (0..100, or true / false)
- *delay* – transition time or delay in seconds (0..65535), by default is 0

### Test the program

If you change the value for object 4/1/1 in Objects menu with Set Value, it will automatically change dimmer state in HDL network.

## 13.3. Usage example – HDL relay control

**Task** of this example is to change HDL dimmer value on specific KNX group address change.

- Add new object in Objects tab
- Add Event-based script which will monitor newly created object
- In Scripting Editor specify the following code for this script

```
1.local value =dpt.decode(event.datahex, dt.bool)
2.HDL.chanreg(1, 11, 1, value))
```

### Test the program

If you change the value for object 4/1/2 in Objects menu with Set Value, it will automatically change the relay state in HDL network.

## 14. Communication with RS232/RS485 serial ports

The following are the naming of Serial ports for different versions of Logic Machine.

LM4		Reactor		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					

### 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/ttyS2', { 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
```

## 15. Bluetooth 4.0 integration

### Task:

Interconnect LogicMachine with Mio Alpha watch and map heart-rate measurement to KNX group address. Upon exceeding specific heart-rate measurement, switch on ventilation on group address 2/2/2. In same way any other Bluetooth LE 4.0 sensor with open API or iBeacon can be integrated with any supported standard by LogicMachine.

### Some of supported Bluetooth 4.0 USB adapters:

- Broadcom BCM20702A0
- Trust 18187
- Belkin F8T065bf
- Plugable USB Bluetooth 4.0
- Laird BT820

### Steps:

- Add 1byte object 1/1/1 in Objects menu
- Add the following code to Resident script with interval = 0 seconds

```
1. if proc then
2.   line = proc:read()
3.   parseline(line)
4. else
5.   mac = 'D7:2D:DA:DF:E4:34' -- MAC of AlphaMio watch
6.
7.   -- bring bt interface up
8.   os.execute('hciconfig hci0 up')
9.   os.sleep(2)
10.
11.   -- read heart rate data
12.   proc = io.popen('gatttool -b ' .. mac .. ' -t random --char-write-req
    -a 0x0025 -n 0100 --listen')
13.   count = 0
14.
15.   function parseline(line)
16.     local pos, rate
17.
18.     -- invalid data
19.     if not line then
20.       return
21.     end
22.
23.     -- find value marker
24.     pos = line:find('value: ', 1, true)
25.     if not pos then
26.       return
```

```

27.     end
28.
29.     -- get current heart rate
30.     rate = tonumber(line:sub(pos + 10, pos + 11), 16)
31.
32.     -- send each 5 reads
33.     count = count + 1
34.     if count == 5 then
35.         grp.update('1/1/1', rate)
36.         count = 0
37.     end
38. end
39. end

```

- Add event-based script heart-rate object 1/1/1. This script will switch on ventilation if the heart-rate is >80 and switch off if its lower

```

1. value = event.getvalue()
2. if value > 80 then
3.     grp.write('2/2/2', true)
4. else
5.     grp.write('2/2/2', false)
6. end

```



## 16. 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

OK

Cancel

## SIP client application

You can use for example Linphone as your SIP client. You have to enter IP of LogicMachine in its settings.

Settings

Run assistant

SIP ACCOUNT

User name

2

Password

Domain

192.168.1.16

Proxy

Transport

UDP

Outbound proxy

OFF

AVPF

OFF

SETTINGS

Enable video

ON

Audio

Video

Call

Network

Advanced

History

Contacts

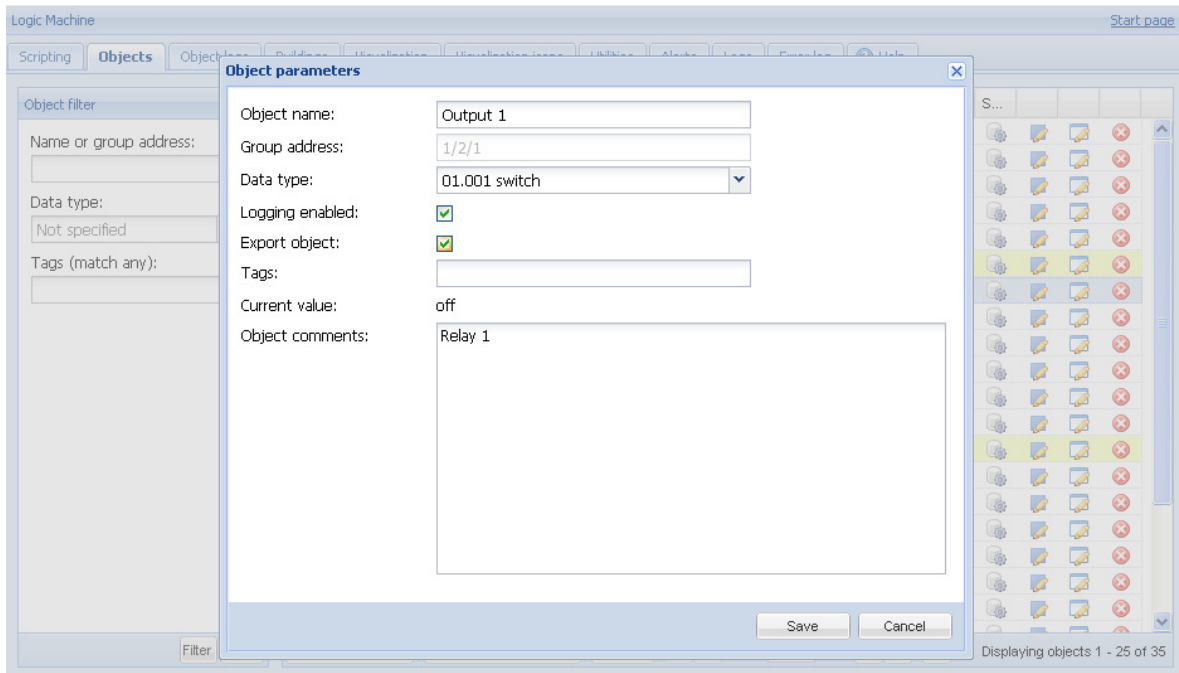
Chat

Settings

## 17. 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

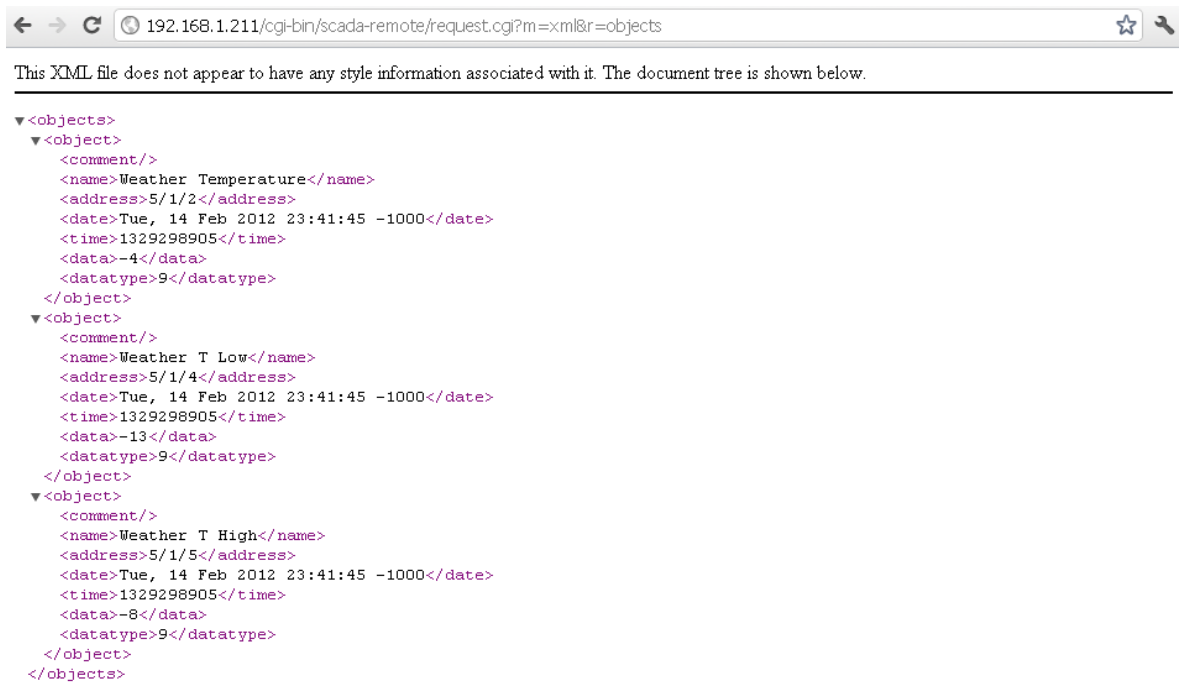
The XML request looks like this:

<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)



### Login, Password for remote XML request

Login and password can be changed in *Network Configuration* → *System* → *GUI Login* → *Admin/Remote* tab.

## 17.1. Alerts, Errors values

In similar way also Alerts and Errors can be read by XML requests.

Alerts XML request:

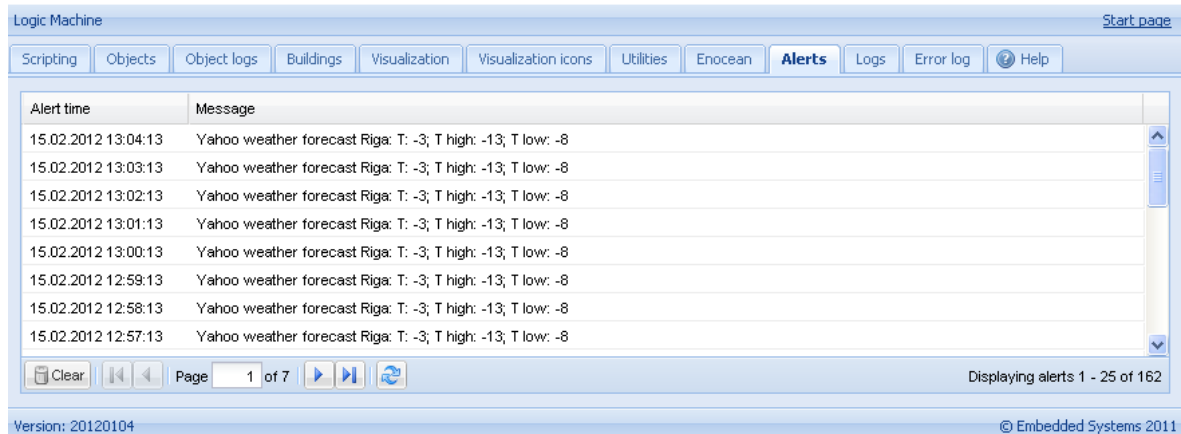
<http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=xml&r=alerts>

Errors XML request:

<http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=xml&r=errors>

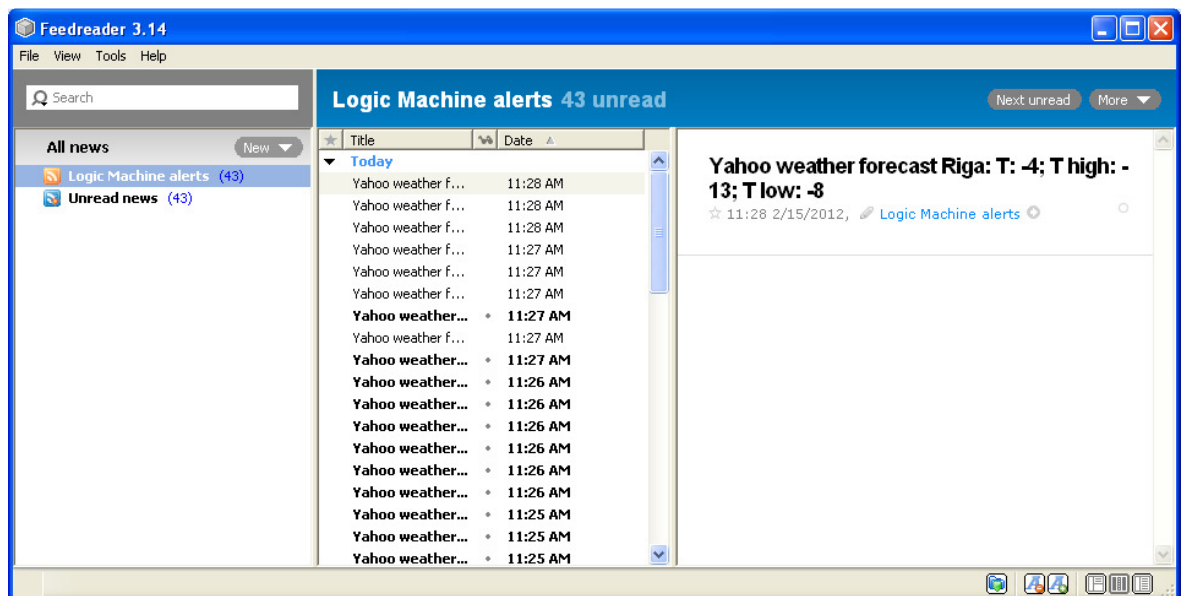
## 18. Read Alerts RSS feeds from LogicMachine

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 image shows a 'User access' dialog box with a close button (X) in the top right corner. It has two tabs: 'Admin / Remote' (selected) and 'Visualization'. The dialog contains two sections for user configuration. The first section is for the 'admin' user, with fields for 'Login' (pre-filled with 'admin'), 'Password' (masked with '\*\*\*\*\*'), and 'Repeat password' (masked with '\*\*\*\*\*'). The second section is for the 'remote' user, with fields for 'Login' (pre-filled with 'remote'), 'Password' (masked with '\*\*\*\*\*'), and 'Repeat password' (masked with '\*\*\*\*\*'). At the bottom of the dialog are 'OK' and 'Cancel' buttons.

User access	
Admin / Remote Visualization	
Login	admin
Password	*****
Repeat password	*****
Login	remote
Password	*****
Repeat password	*****

OK Cancel