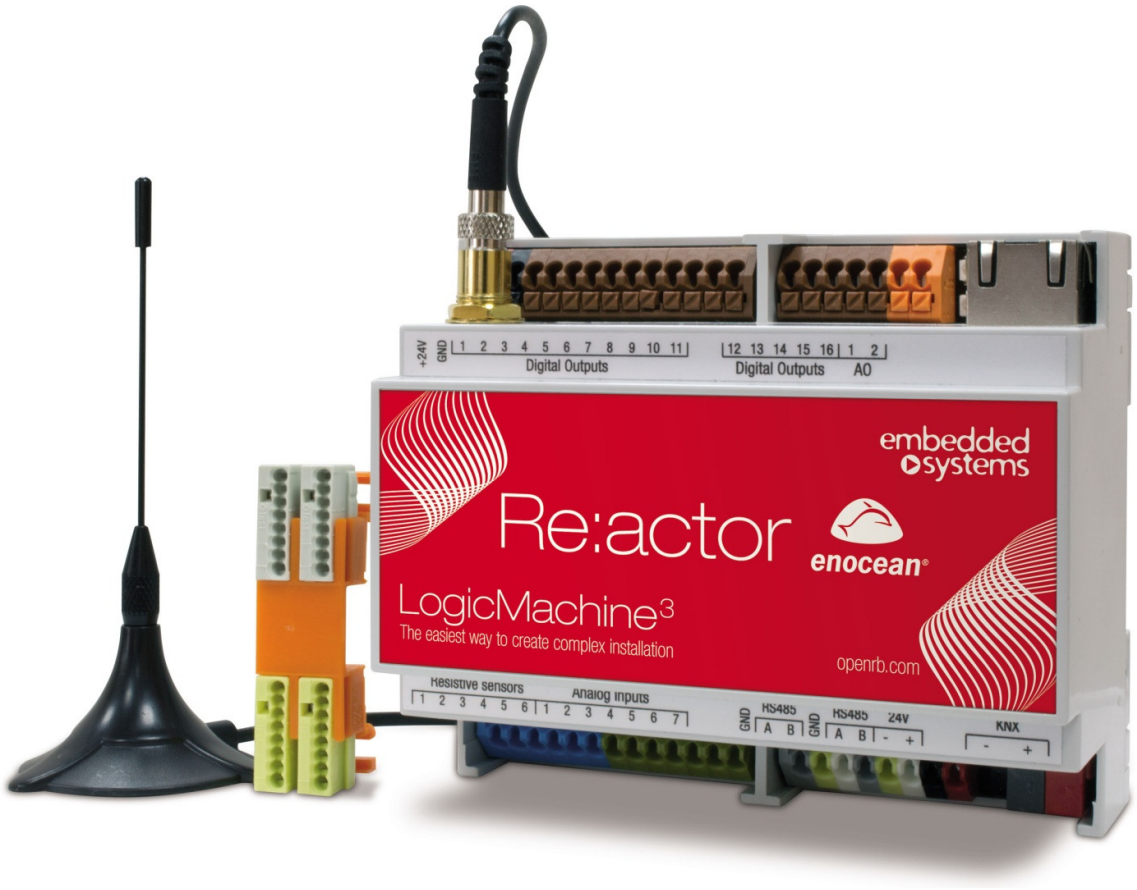
KNXlogic.eu

LogicMachine3 Re:actor

Product Manual



Document Issue 1.0
June, 2013

Technical Support:
support@openrb.com

Copyright

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 Logic Machine 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

Logic Machine is your easiest way to program complex logic in KNX/EIB, Modbus, BACnet, EnOcean networks. The Logic Machine will enable you to efficiently customize building automation processes, easily delivering unlimited flexibility benefit to end users in a cost-effective way.

Logic Machine is an embedded platform with integrated TPUART, Ethernet, USB interfaces. Logic Machine allows to use it as IP Router, cross-standard gateway, logic engine, visualization WEB SCADAserver. Scripting templates provides user-friendly, flexible configuration interface. Via applying custom scripts the Logic Machine 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.

Firmware updates are available at www.openrb.com

Caution
Security advice

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

Mounting advice

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

Electrical connection

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

Contents

Device specification

Application

Logical functions; WEB SCADA visualization for PC and touch-devices; cross-standard gateway between KNXnet/IP, Modbus TCP/IP, Modbus RTU, BACnet MS/TP, BACnet IP, EnOcean, HDL, GSM and other protocols; integration with third party devices over RS232 or RS485 serial ports – DALI, AV, IR; Data logger with trends; HVAC

Types of product

Re:actor	LM3-REACTOR

Standards and norms compliance

CE conformity:	EMBS-CE-111231/01
EMC:	EN61000-6-1 EN61000-6-3
PCT	Certificate

Technical data:

Power supply:	7-36V DC	
Power consummation:	1.5W	
Interface:	EnOcean 868MHz	1
	10BaseT/100BaseTX	1
	RS485	2
	USB2.0	1
	TPUART2	KNX/EIB compatible
	Digital output	16 (380 mA continues current on output). Optoislated from KNX/EIB bus. Additional 24V power source is supported.
	Resistive sensor inputs	6 (PT100, PT1000, 0 Ω...20M Ω)
	Analog/binary inputs	7 (0-30V with configurable threshold voltage, 12bit resolution)
	Analog outputs	2 (0-10V, 12bit resolution, 20mA max current)
Connections:	KNX bus:	Bus Connection Terminal 0.8mm2
	Power supply:	Clamp, 1.5mm2

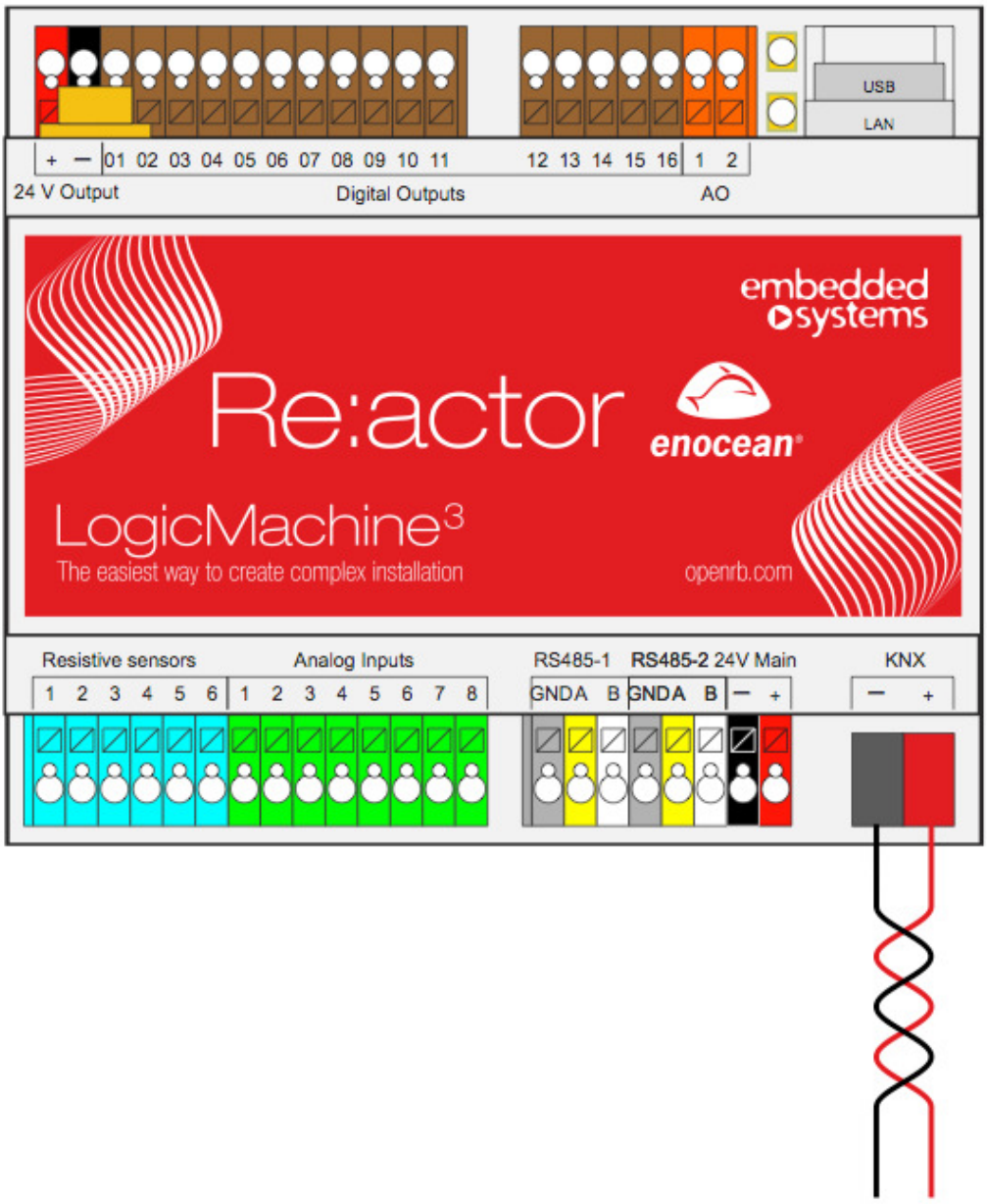
	Serial:	Clam, 1.5mm2
	IO:	Clam, 1.5mm2
Operating elements	LED	1 – CPU load 1 - Activity
Enclosure:	Material:	Polyamide
	Color:	Gray
	Dimensions:	104(W)x90(H)x51(L) mm
Usage temperature:	-5C ... +45C	
Storage temperature:	-25C ... +55C	
Weight:	150g	
Warranty:	2 years	
Warranty:	2 years	

LogicMachine3 Re:actor kit contains:

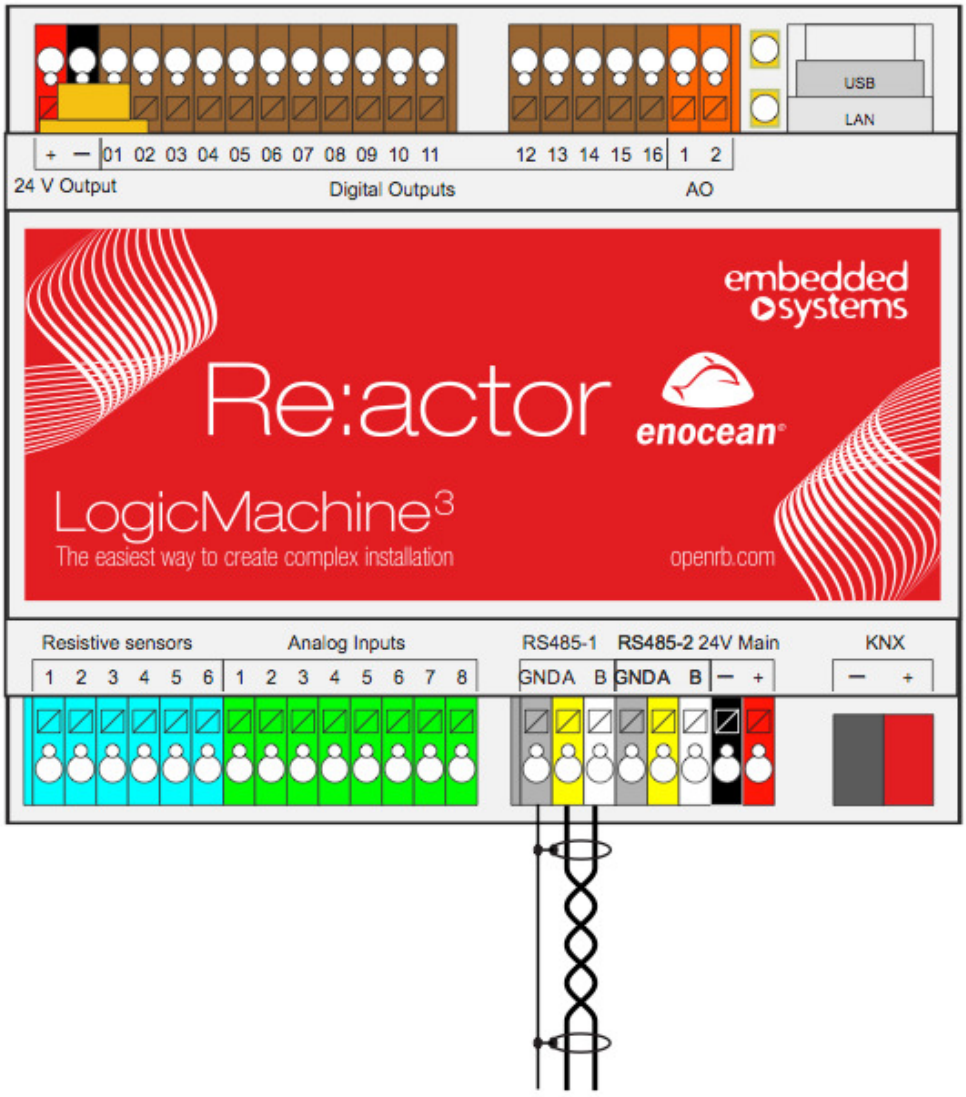
- Embedded board with preinstalled software
- Plastic DIN-rail case
- 868MHz antenna
- 4 x 8pin extension connectors

Terminal connection schemes

KNX TP



RS485

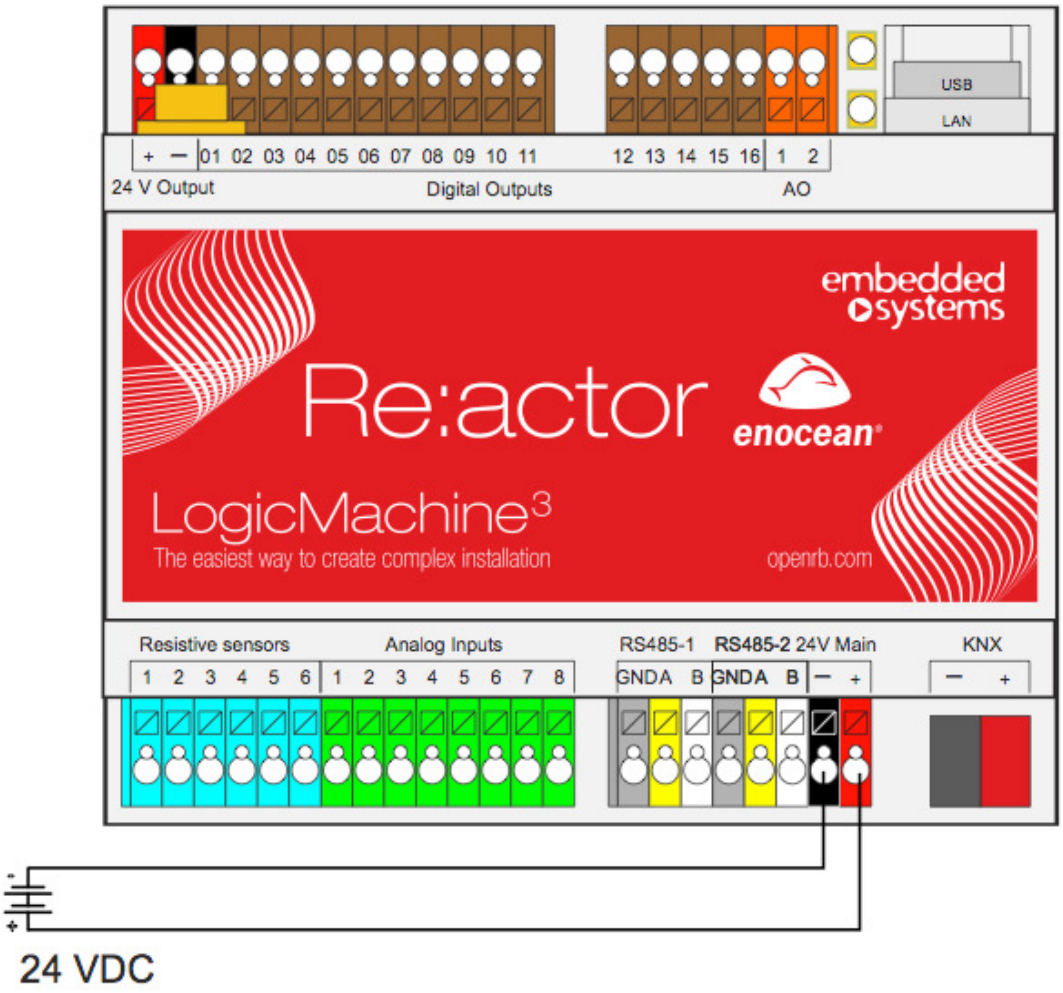


The EIA-485 differential line consists of two pins:

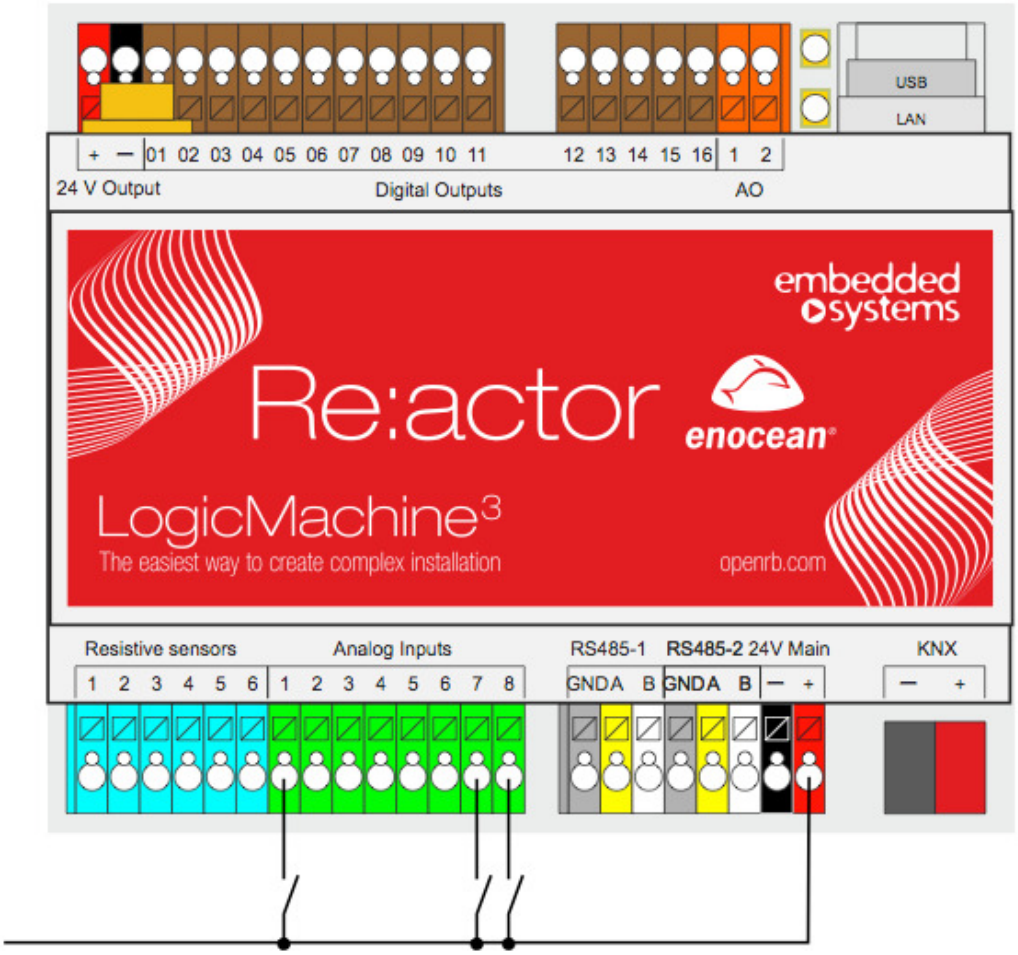
A aka '-' aka TxD-/RxD- aka inverting pin

B aka '+' aka TxD+/RxD+ aka non-inverting pin

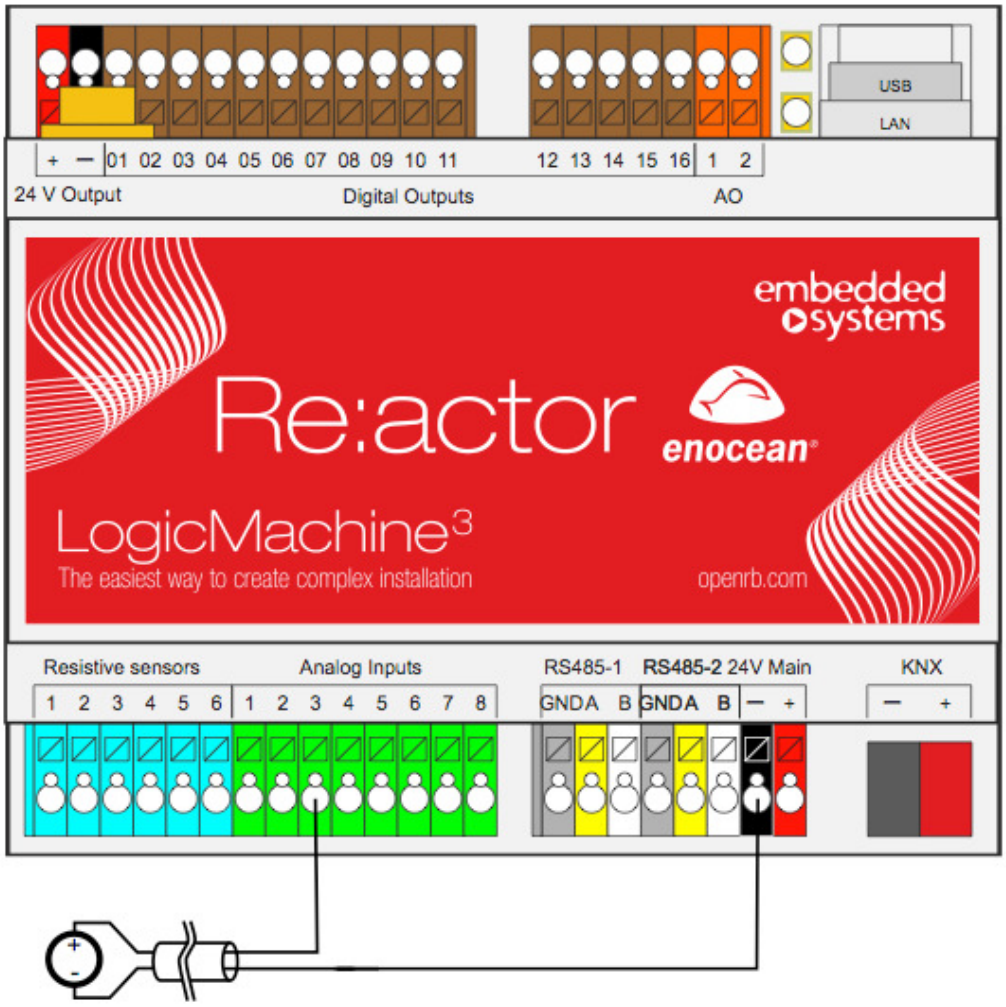
24V power supply



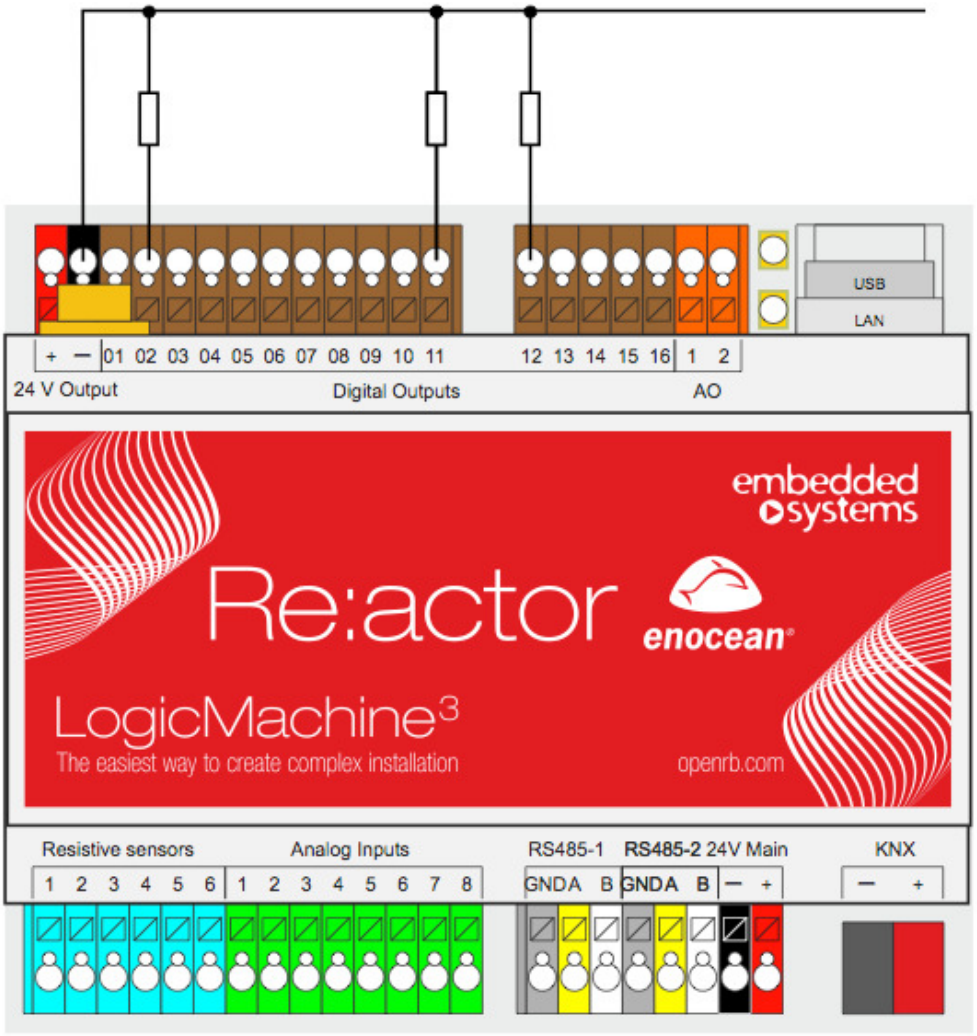
Analog inputs (e.g. reed contact)



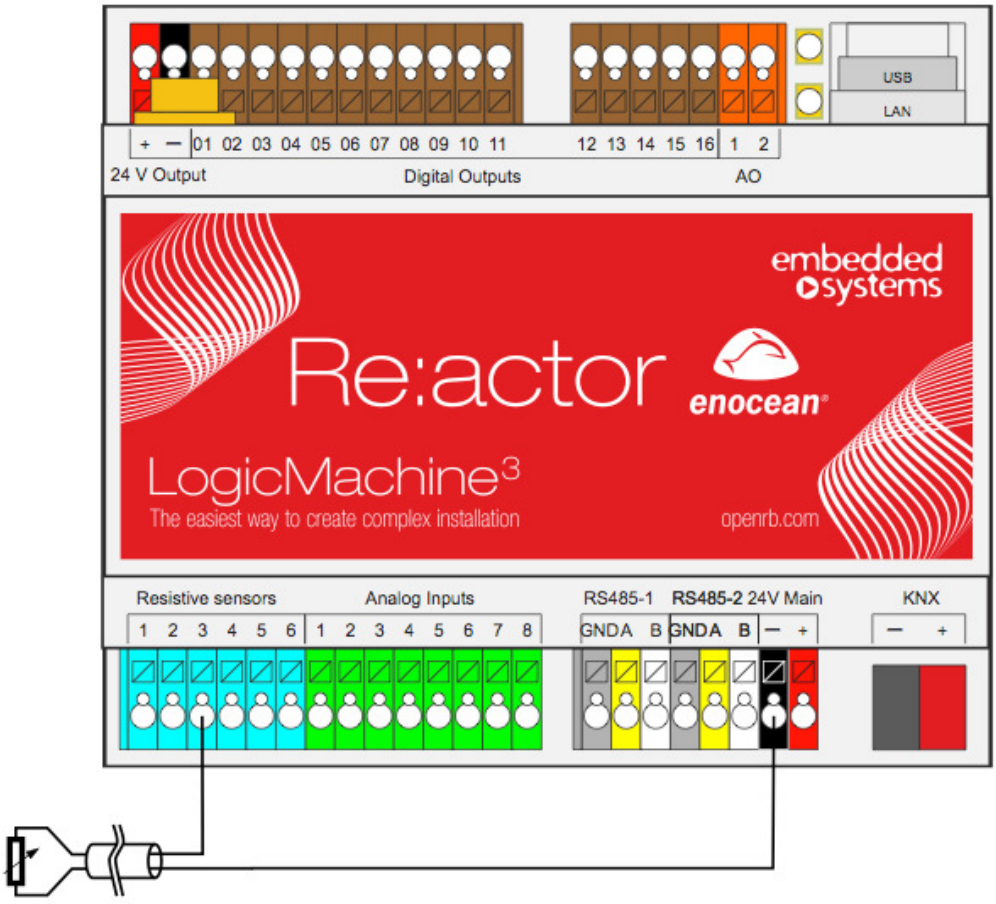
Analog inputs, 0-10V



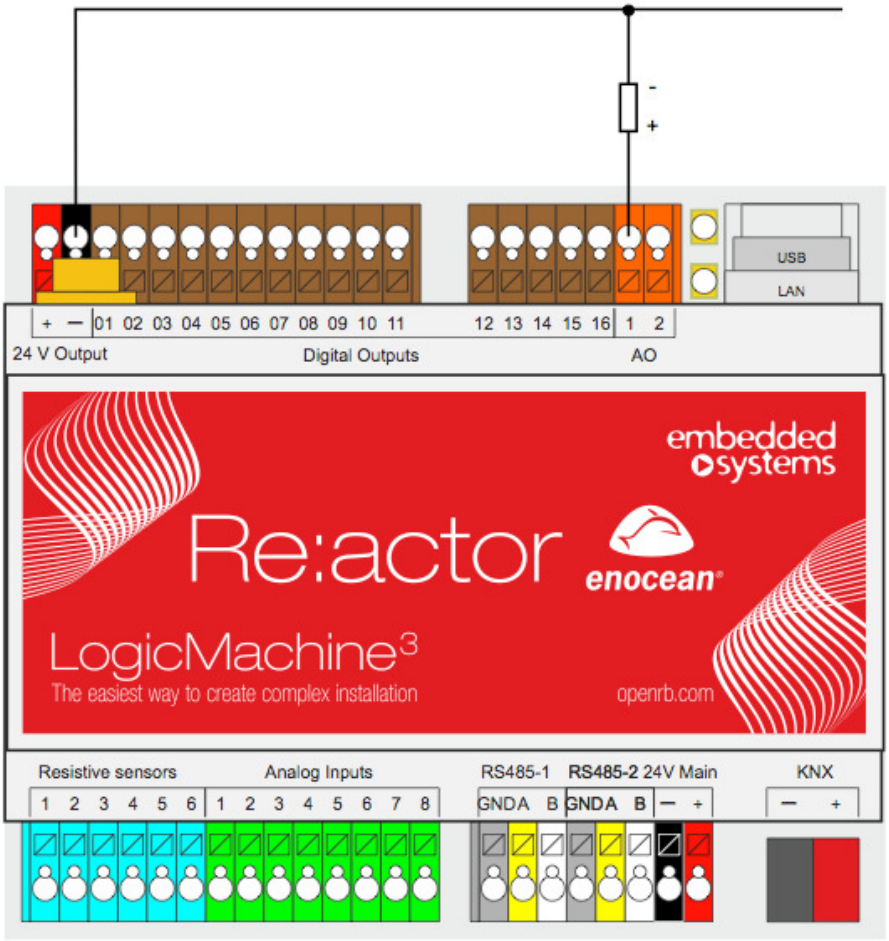
Digital output



Resistive sensor input



Analog output



Factory default, discover IP

There is a reset button on the side of Logic Machine 2. 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

Logic Machine is compatible with the following standards:

- KNX/EIB TP, KNXnet/IP
- Modbus TCP, Modbus RTU
- BACnet IP, BACnet MS/TP (in development)
- GSM (Huawei E173 and similar modem support through USB) for sending SMS notifications and controlling the installation by receiving SMS commands.
- EnOcean (built-in bi-directional transceiver)
- DMX (in the box, through RS485)
- DALI (support is done over RS485 by using external RS485-DALI interface)
- Ekey biometrical access systems (RS485)
- HVAC systems can be controller through RS232 interface by using scripting
- SMTP/Email, SSL
- SIP (works as PBX for controlling calls, in development)
- XML (export object values, alerts or errors)
- 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 Logic Machine can act as BACnet to Enocean gateway or Modbus to GSM etc.

Quick startup guide

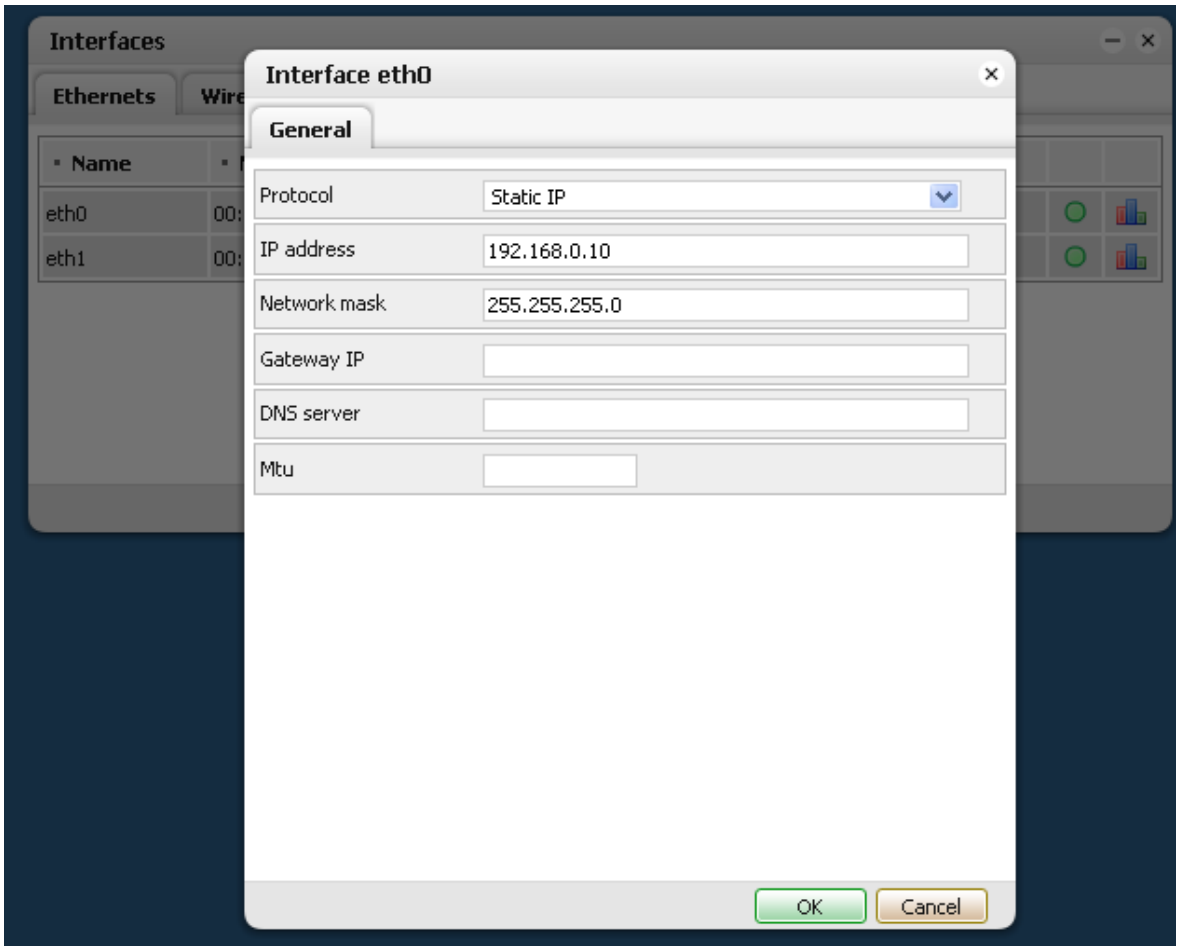
1) Mount the device on DIN rail
2) Connect the KNX bus cable
3) Connect 24V power supply to the device (red pole to *24V+*, grey pole to *GND*)
4) Connect Ethernet cable coming from the PC

Default IP configuration

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

Change IP settings

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



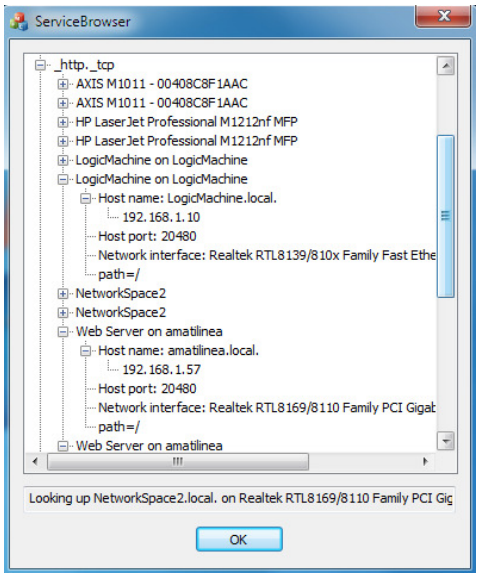
- ***Protocol***– specific protocol used for addressing
  - ***Static IP*** – static IP address. By default 192.168.0.10
  - ***DHCP*** – use DHCP protocol to get IP configuration.
    - ***Current IP***– the IP address got from DHCP server. This field appears only if the IP address is given otherwise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

Discover Logic Machine 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

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.

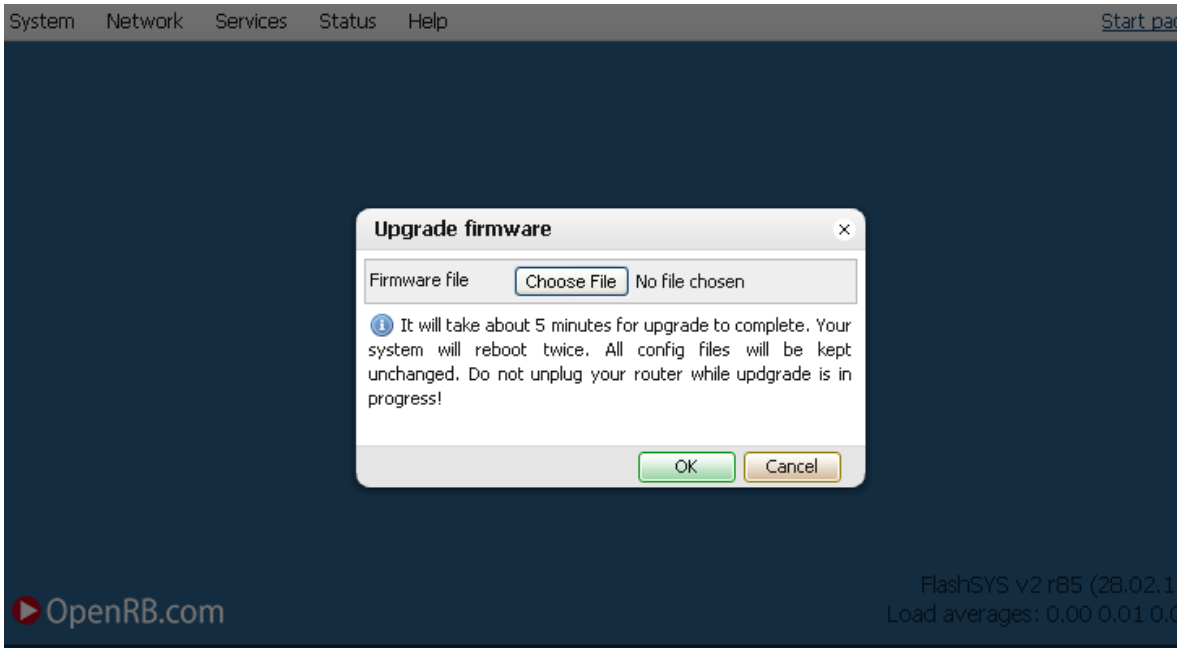
Firmware upgrade

Note! Before each upgrade please backup your visualization, scripts and object in *Logic Machine → Tools → Backup*, as the database is cleaned during the upgrade.

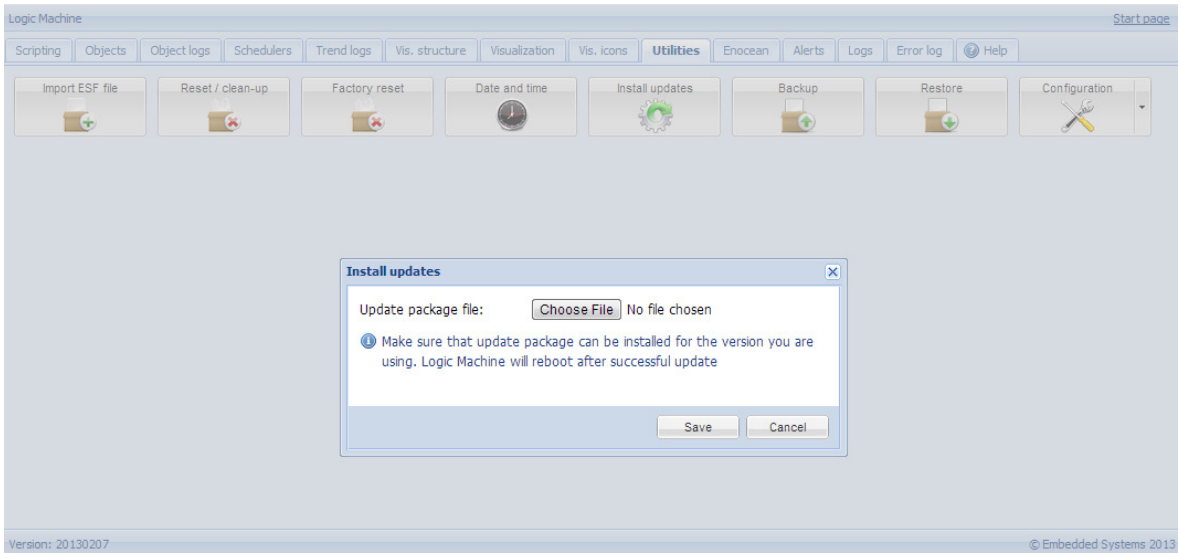
Note! After each upgrade, we strongly recommend to clean your browser cache.

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 ofwww.openrb.com.

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



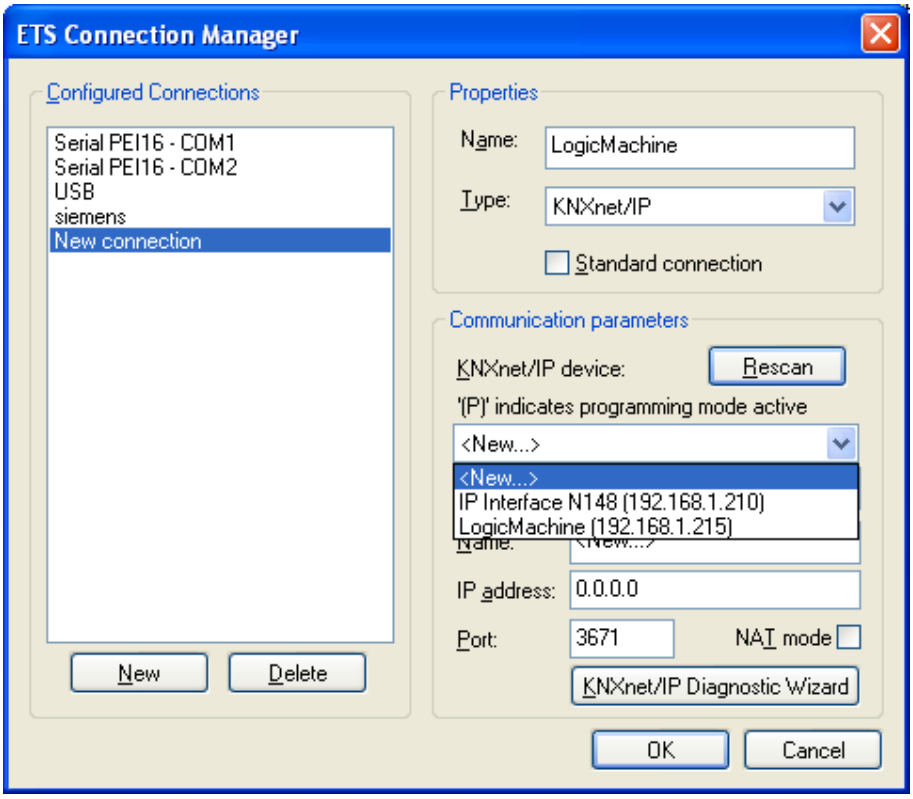
Logic Machine visualization upgrade can be done in *Utilities* tab and press on *Install updates* icon. After *.LMU file is chosen from the corresponding location press *Save* button. The device will be rebooted after 5 seconds and new firmware will be installed.



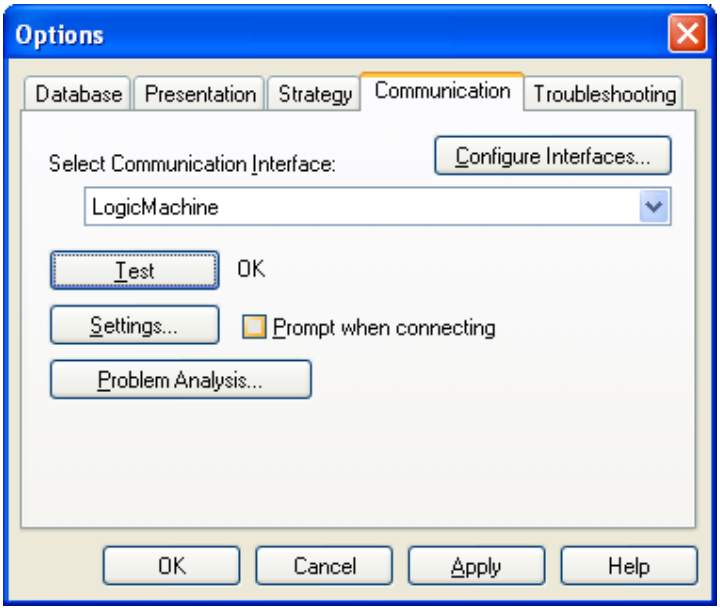
Logic Machine for KNX/EIB network configuration management with ETS

To use Logic Machine 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*



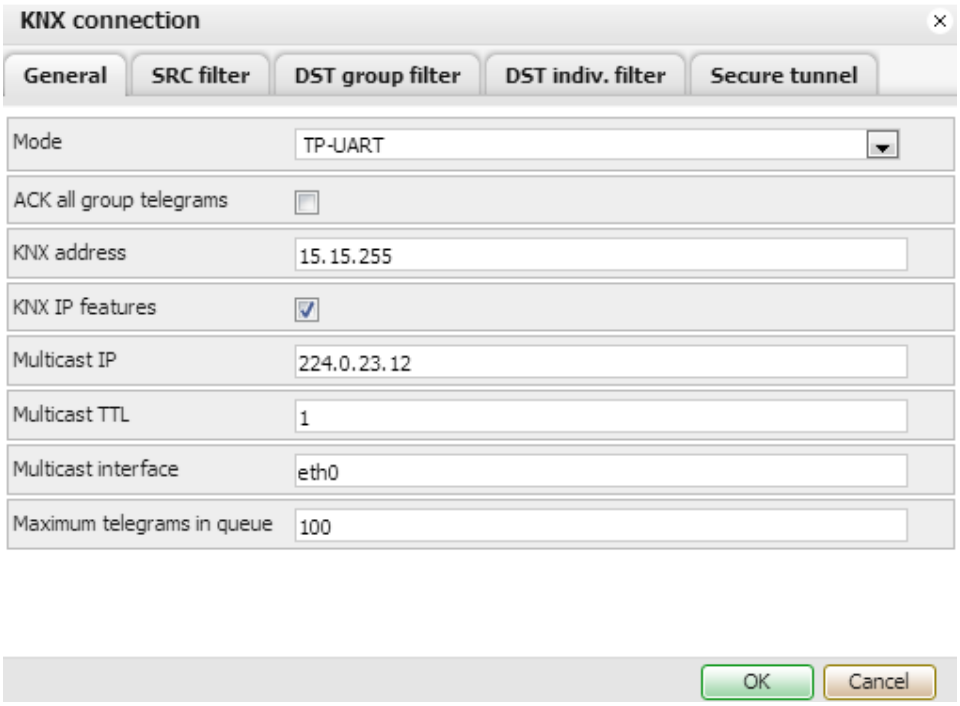
- Put some freely chosen *Name* for the connection
- Chose ***Type = KNXnet/IP***
- Press ***Rescan*** button and then choose from the drop down menu found Logic Machine
- Press ***OK***
- Back in ***Options*** → ***Communication*** window select newly created interface as ***Communication Interface*** 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

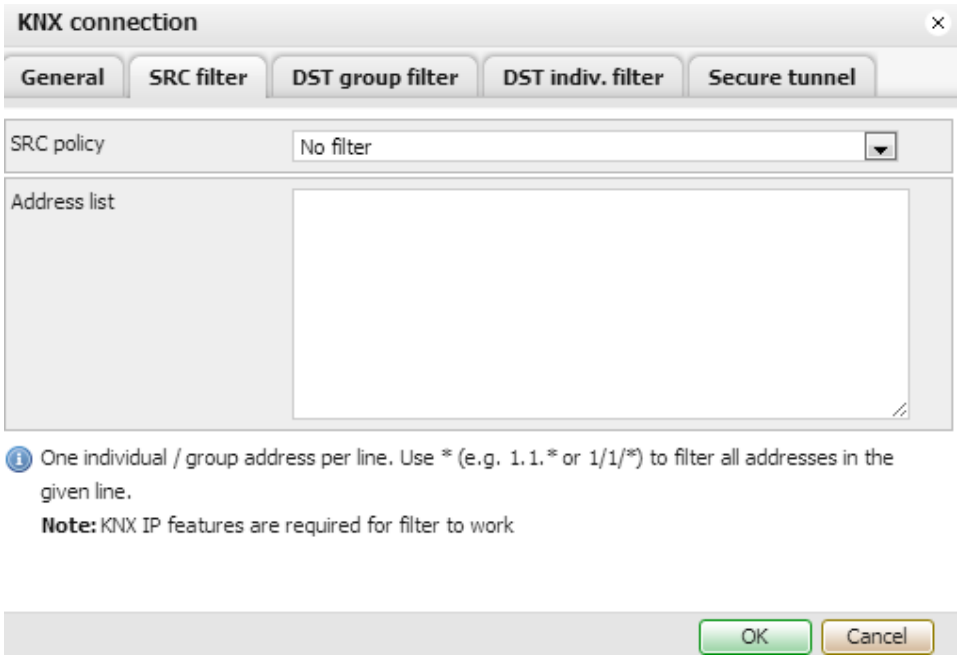
KNX specific configuration is located in *Network configuration→Network → KNX connection* window.



General tab

- ***Mode*** *[TP-UART / EIBnet IP Tunneling / EIBnet IP Tunneling(NAT mode) / EIBnet IP Routing]* – KNX connection mode. LogicMachine3 Re:actor 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 qmode.
- ***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 interface*** – multicast interface to use when sending KNX telegrams to other KNX networks over TCP/IP
- ***Multicast IP*** – multicast IP address
- ***Maximum telegrams in queue*** – count of maximum telegrams in the queue

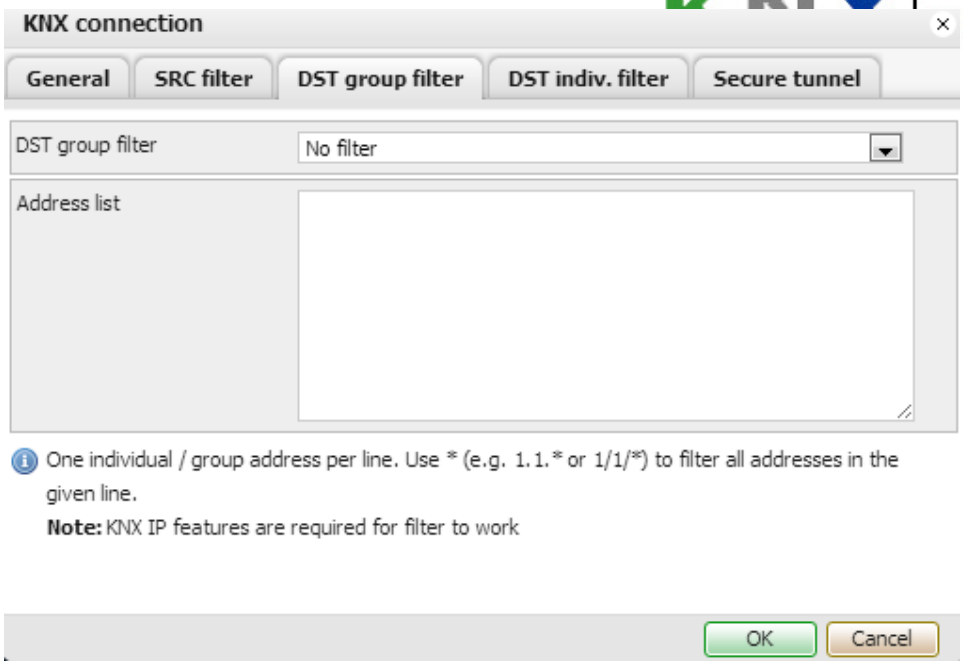
Source filter tab



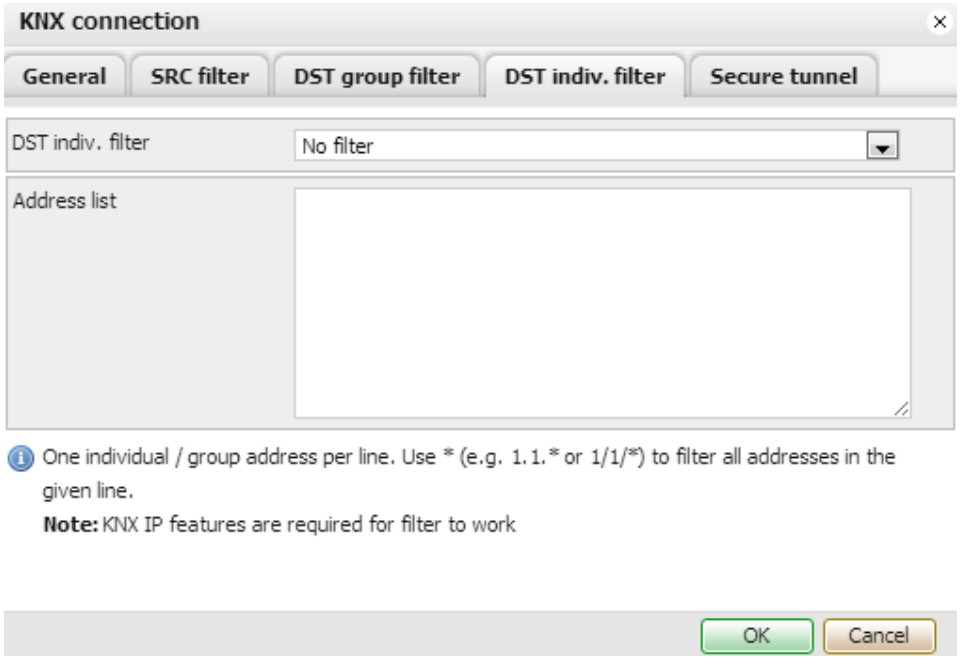
- ***SRC policy*** *[No filter / Accept selected individual addresses / Drop selected individual addresses]*– policy to apply to the list of source addresses
- ***Address list*** – list of individual or group addresses. One address per line. Use * (e.g. 1.1.* or 1/1/*) to filter all addresses in the given line. Note! *KNX IP features* should be on for filter to work

Destination group filter tab

- ***DST group filter*** *[No filter / Accept selected individual addresses / Drop selected individual addresses]*– policy to apply to the list of destination group addresses
- ***Address list*** – list of group addresses. One address 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



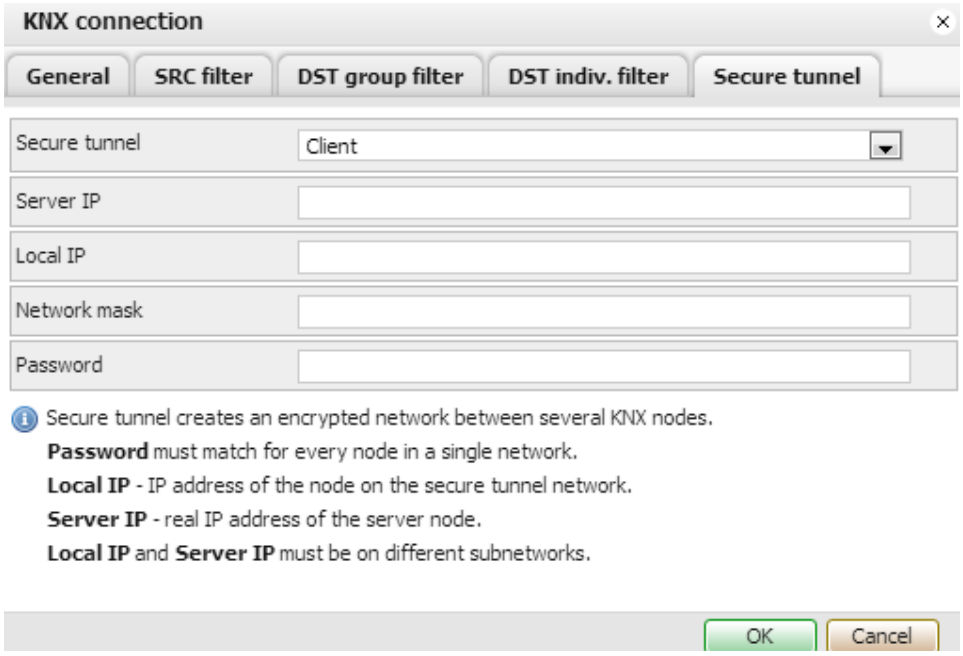
Destination individual filter tab



- ***DST indiv. filter*** *[No filter / Accept selected individual addresses / Drop selected individual addresses]*– policy to apply to the list of destination addresses
- ***Address list*** – list of individual addresses. One address 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

Secure tunnel tab

You can make a secure tunnel between two KNX networks. In comparison with standard tunnels, which use UDP protocol, this tunneling uses TCP what makes it very reliable thanks to package delivery acknowledgement. This ensures that sender always knows if the package is delivered to the recipient.

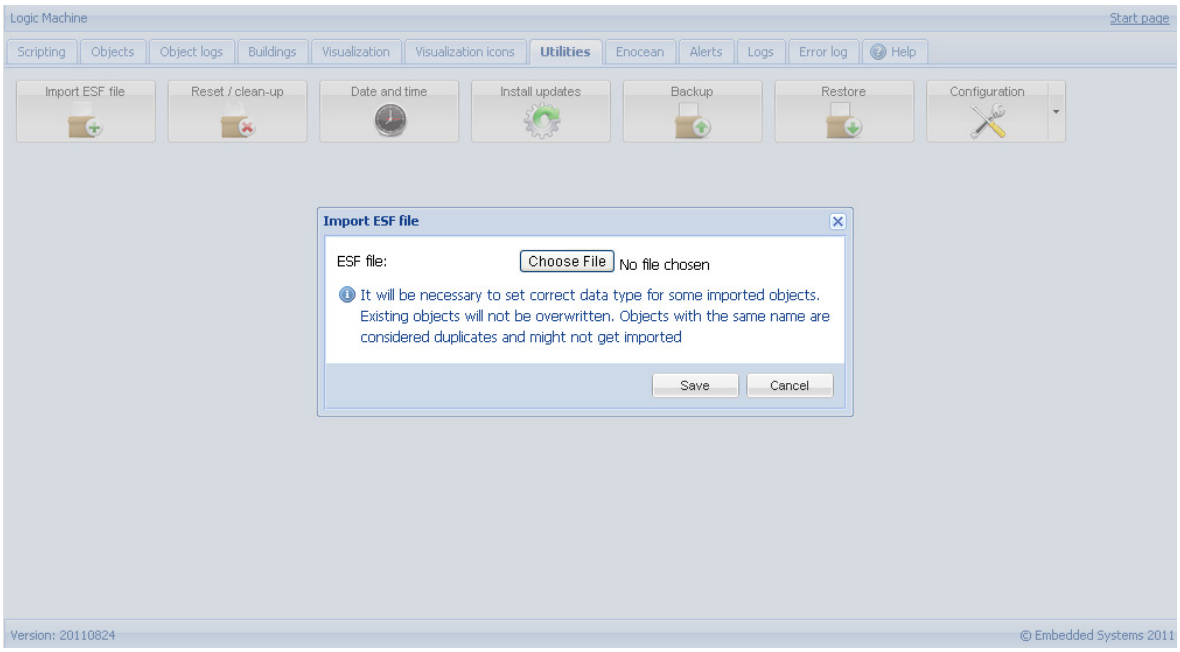


- ***Secure tunnel*** *[Disabled / Client / Server]* – secure tunnel mode
- ***Server IP*** – in case of secure client, server IP should be specified here
- ***Local IP***– local IP address
- ***Network mask*** – network mask
- ***Password***– password

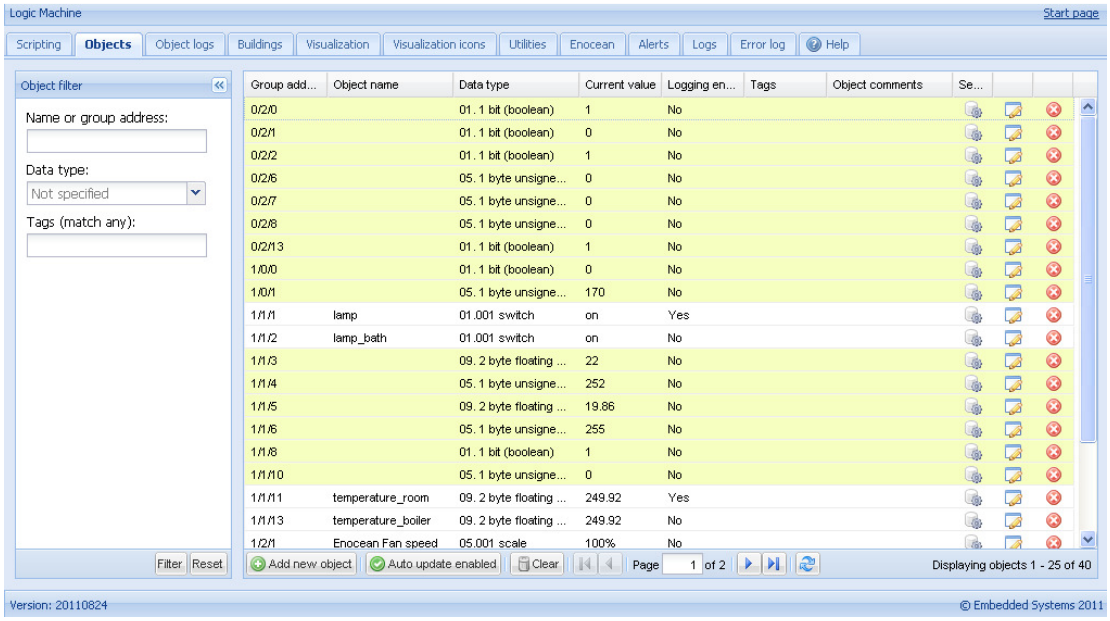
Quick guide - create visualization for iPad/PC

Import objects

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

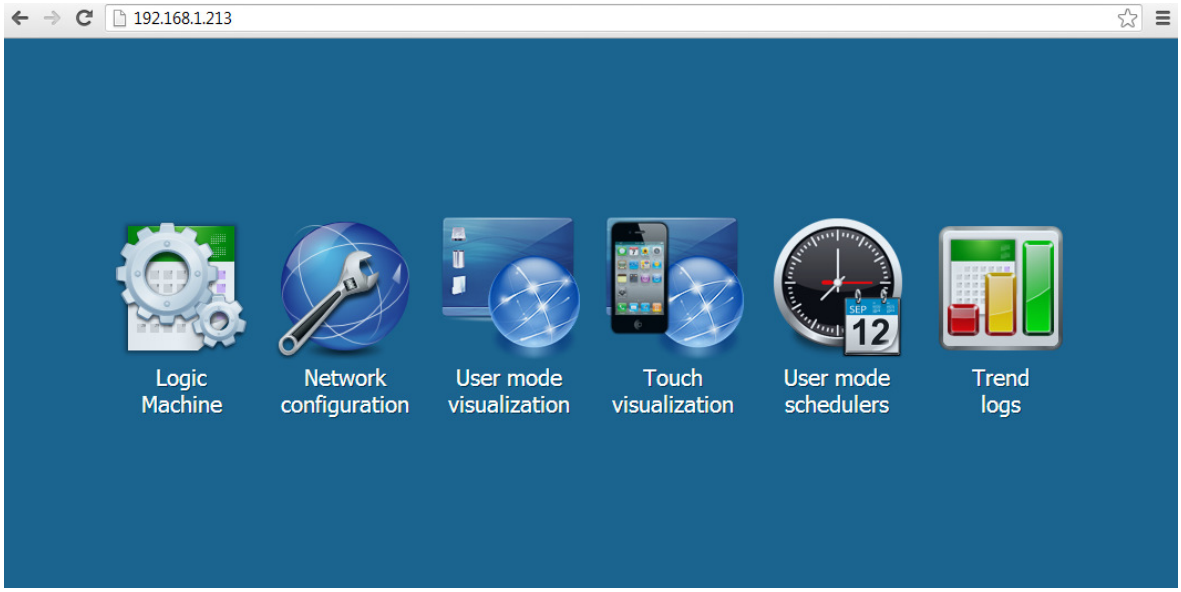


Or connect LM to the bus and it will detect objects automatically in *Objects* tab once they are activated. Objects can be added manually as well.



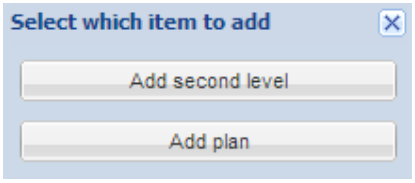
Create “floor” structure and add objects to the map

Connect to Logic Machine (*Logic Machine*) with default access parameters (**IP:** 192.168.0.10; **login/password**: admin/admin)

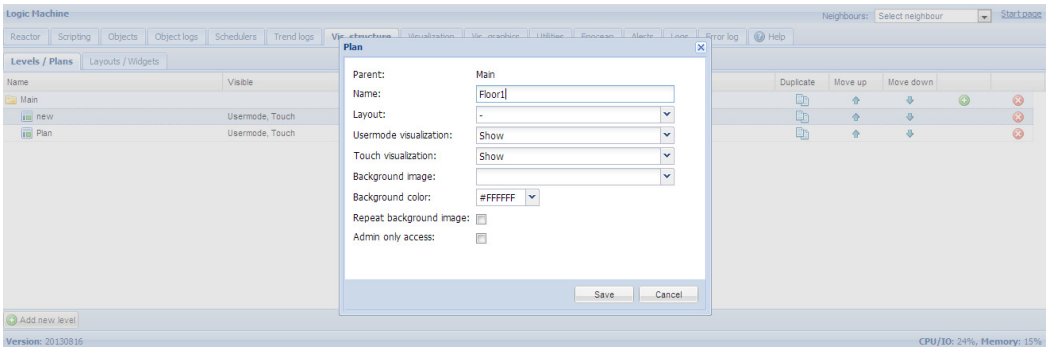


Create “building/floor” structure and add objects to the map

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

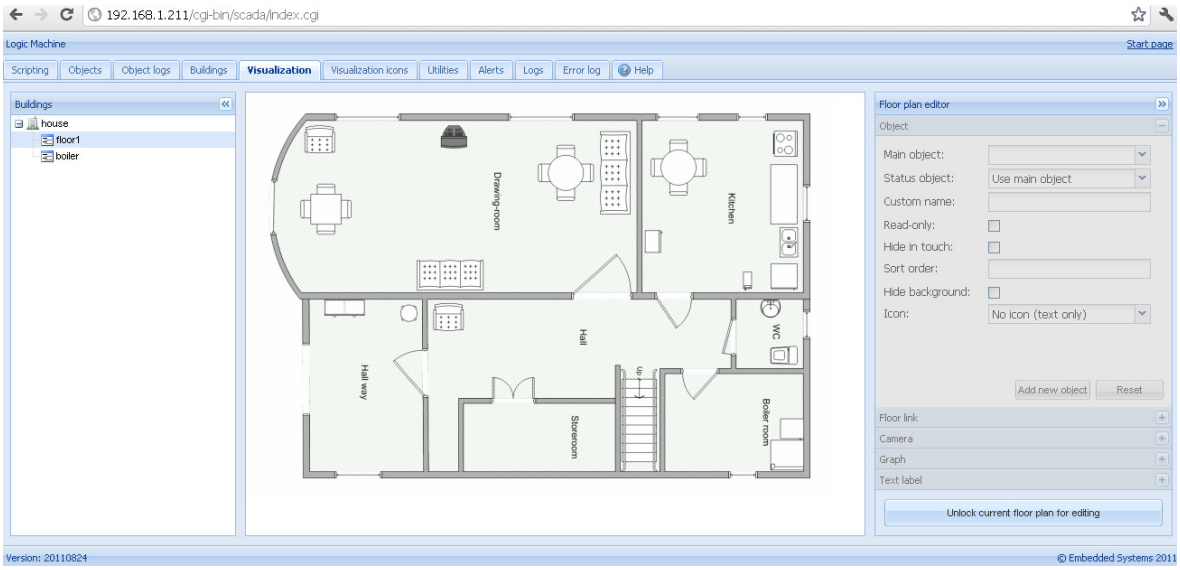


Choose either to add as second floor level or add plan for this particular floor level.

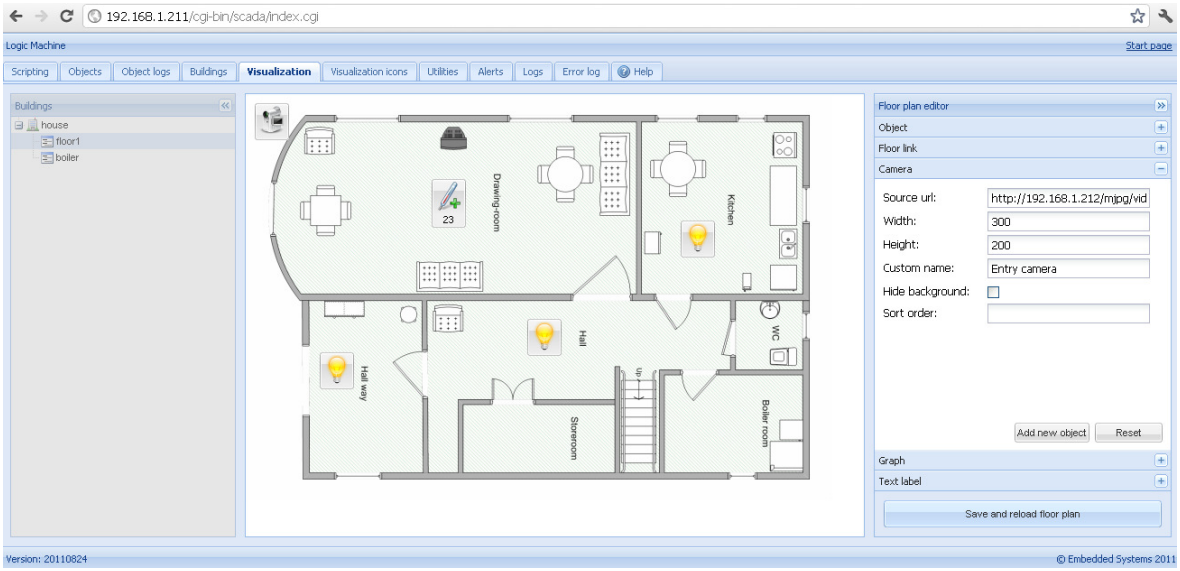


Add objects to newly created visualization map

After the building and floor structure is defined in *Buildings* 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 left/right arrow icon making the map more visible especially on small displays.



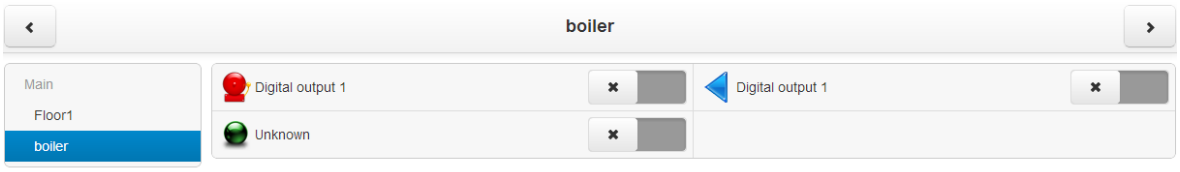
Existing objects can be added to the map by clicking on *Unlock current floor plan for editing* button. Once the object parameters are defined, press *Add new object* button and newly created object will appear. You can move the object to the location it will be located. Note that while being in editing mode, the object will not work.



When all necessary objects and cameras are added, press *Save and reload floor plan* button so everything starts functioning.

Launching visualization on touch device (iPad in this case)

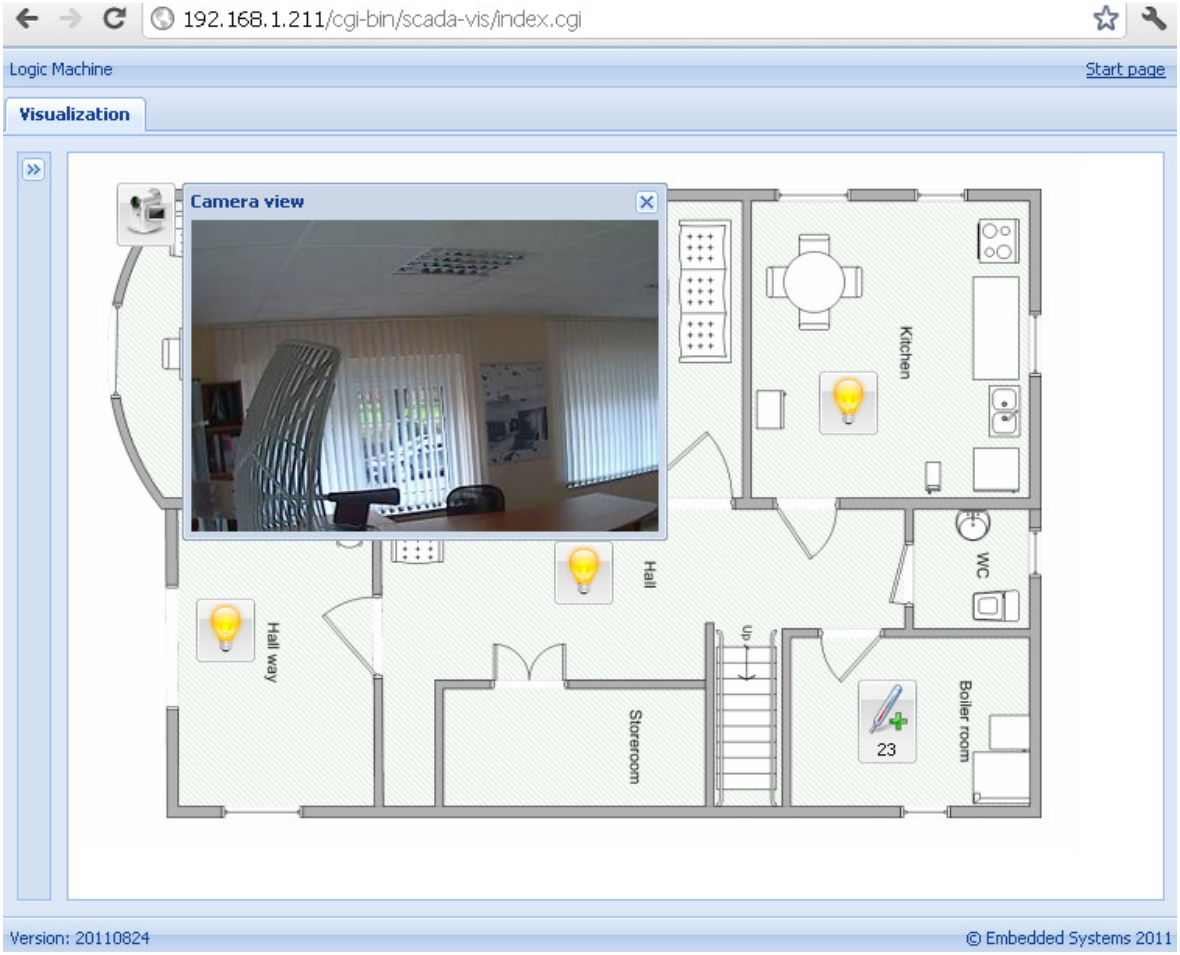
- Make sure your iPad is connected wirelessly to the Logic Machine (either through separate access point or directly to Logic Machine's USB WiFi adapter).
- In the browser enter Logic Machine's IP (default 192.168.0.10).
- Click on the User *mode visualization* or *Touch visualization* icon.
- Save the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



Launching visualization on PC, iPad or any other touch device with large enough screen

- Make sure your PC/touch device is able to access Logic Machine and enter it's IP in the browser (default 192.168.0.10).
- Click on the *User Mode Visualization* and enter the "floor" you want to see.

- Then minimize side bar by pressing on left-arrow icon to make the map more visible.

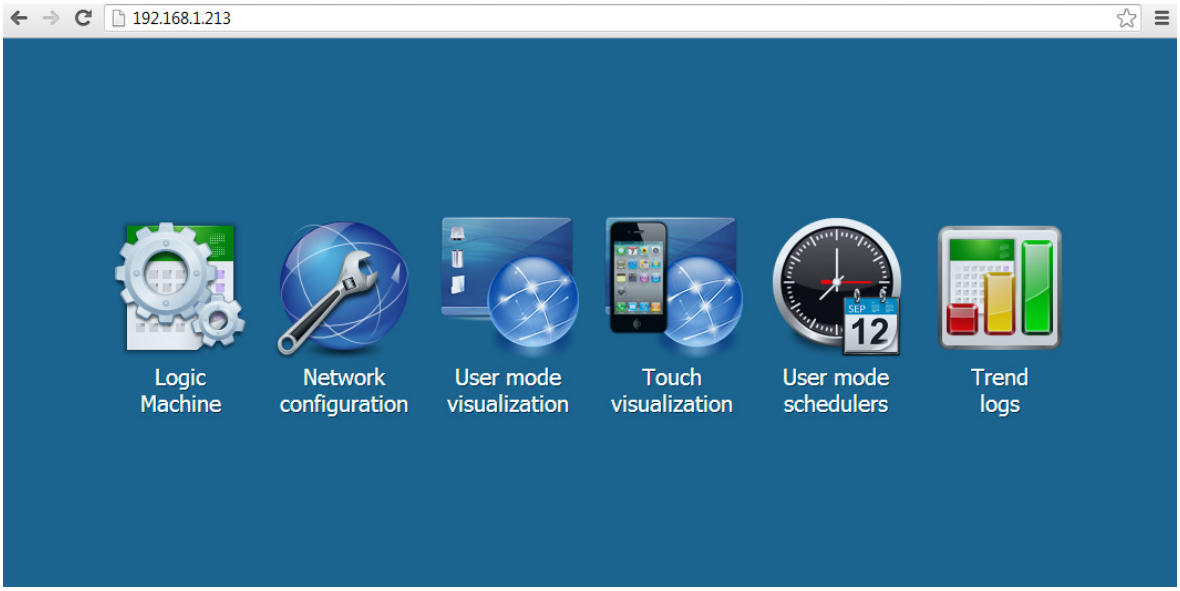


Graphical User Interface Login

KNX/EIB Logic Machine 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 Logic Machine 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. Logic Machine configuration

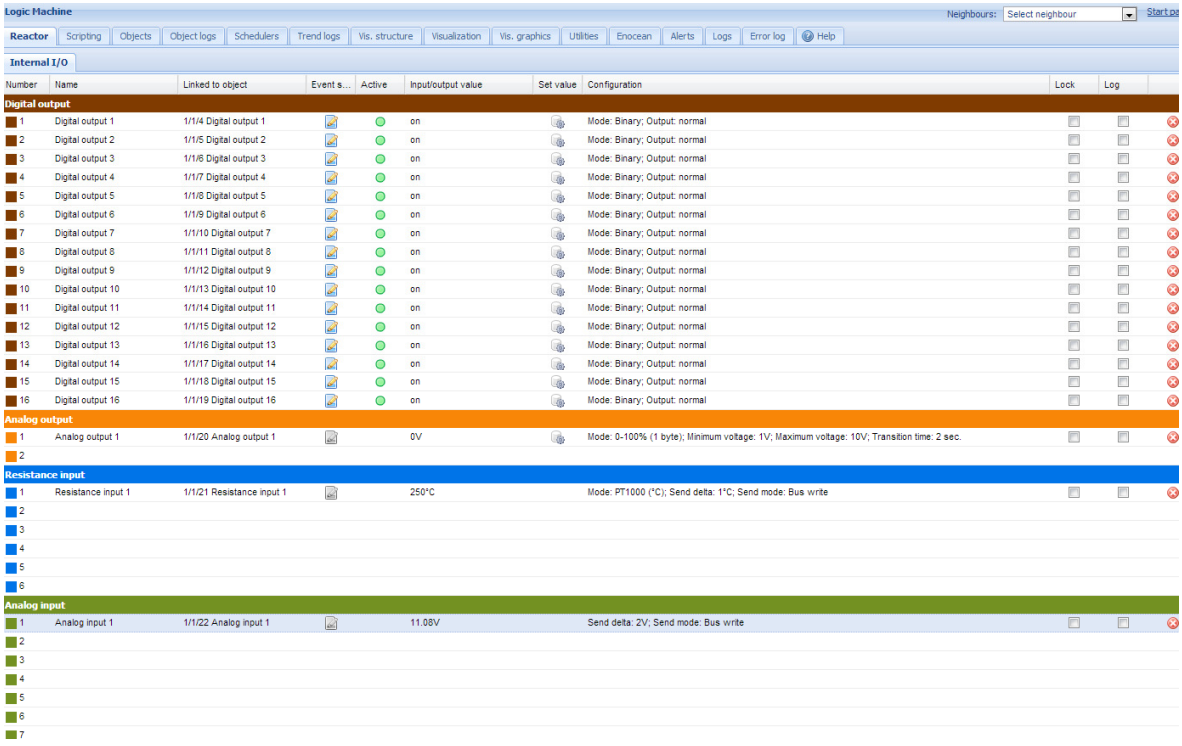
Login	Password
admin	admin

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

Reactor – LM3 Re:actor specific IO settings
Scripting – scripting repository management
Objects– list of KNX network objects
Object logs– KNX bus object historical logs
Schedulers– administrator interface for user mode schedulers
Trend logs – administrator interface for trend logs
Vis.structure – building definition and image file upload
Visualization– Visualization management, control and monitoring
Vis.icons– icon management
Utilities – utilities including import from ETS, reset object DB, backup, update system installation
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

1.1. Reactor

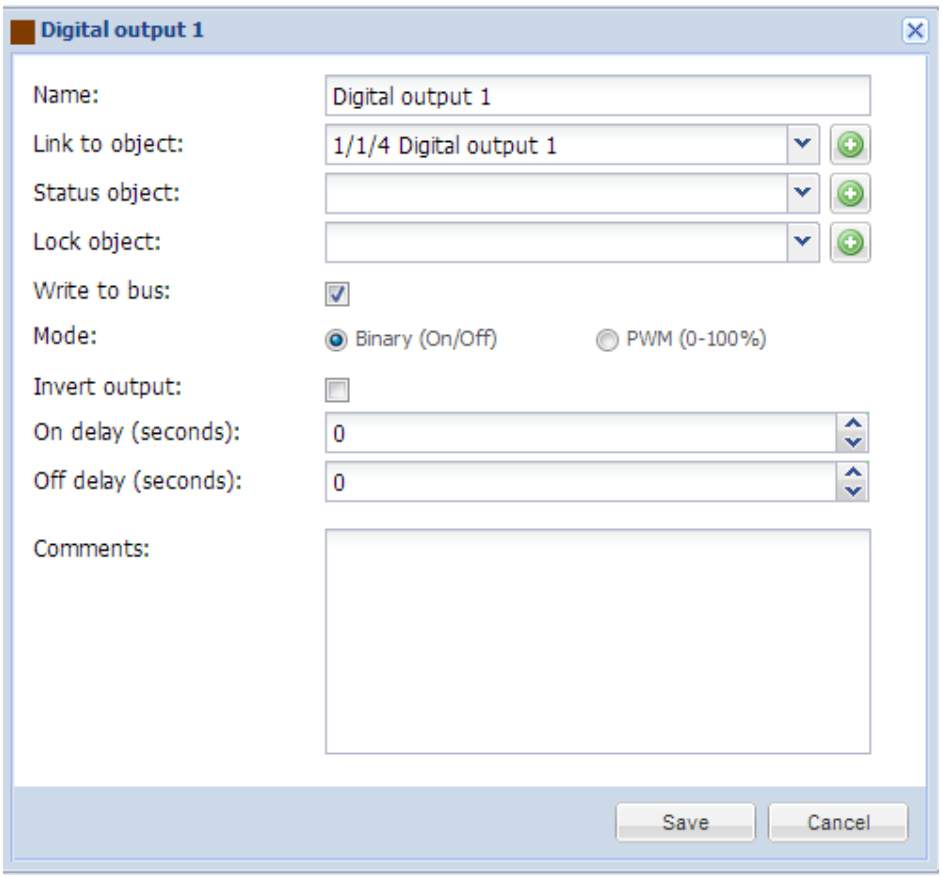
Reactor IO configuration and mapping to KNX is done under *Reactor* tab. No additional software is needed to configure KNX mapping of IO ports.



1.1.1. Digital output

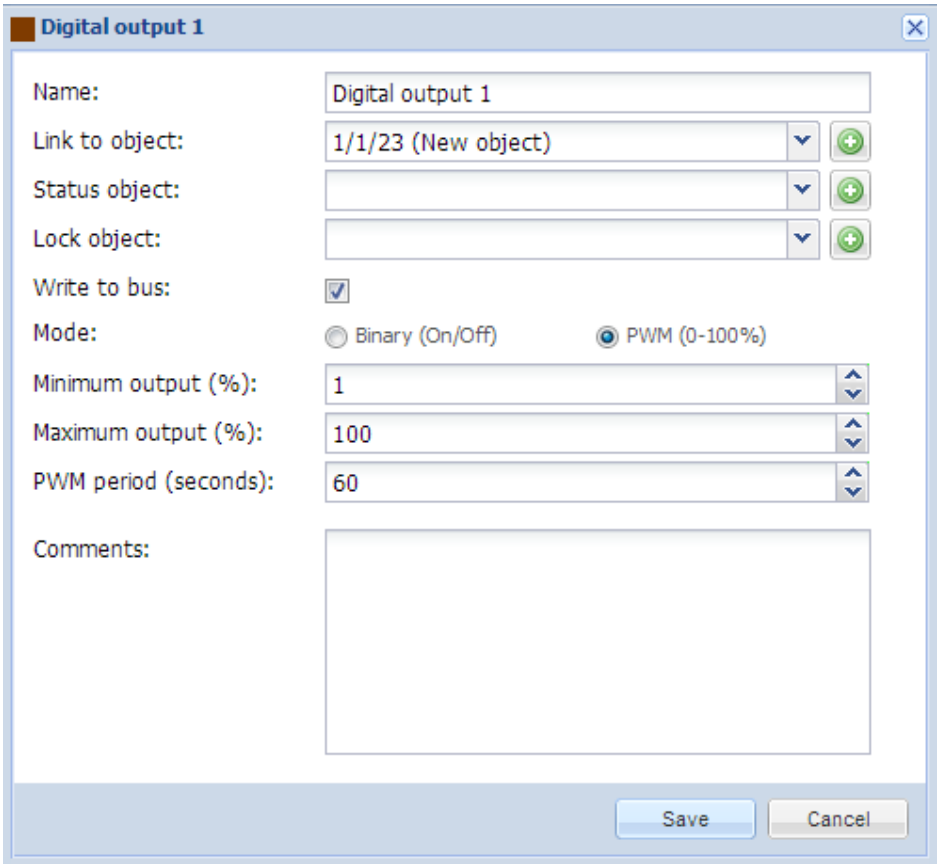
You can set up digital output either as binary output or PWM. Binary output is usually used to control external relays/contactors. PWM output is used to control external devices like thermoelectric valves.

Binary (On/Off) output



- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [Binary(on/Off); PWM(0-100%]*** – output mode
- ***Invert output*** – defines either to invert the output value
- ***On delay (seconds)*** – delay in seconds when getting in On state
- ***Off delay (seconds)*** – delay in seconds when getting in Off state
- ***Comments*** – comment of the object

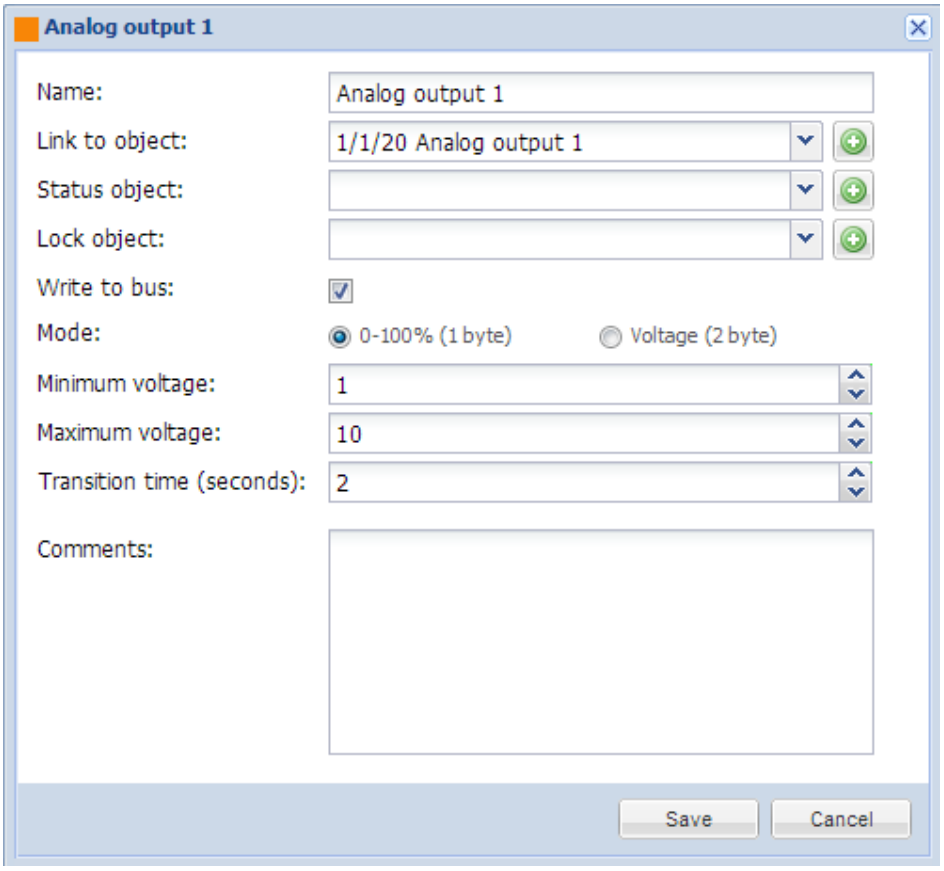
PWM (0-100%) output



- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [Binary(on/Off); PWM(0-100%]*** – output mode
- ***Minimum output (%)*** – minimum output value
- ***Maximum output (%)*** – maximum output value
- ***PWM period (seconds)*** – pulse-width-modulation period
- ***Comments*** – comment of the object

1.1.2. Analog output

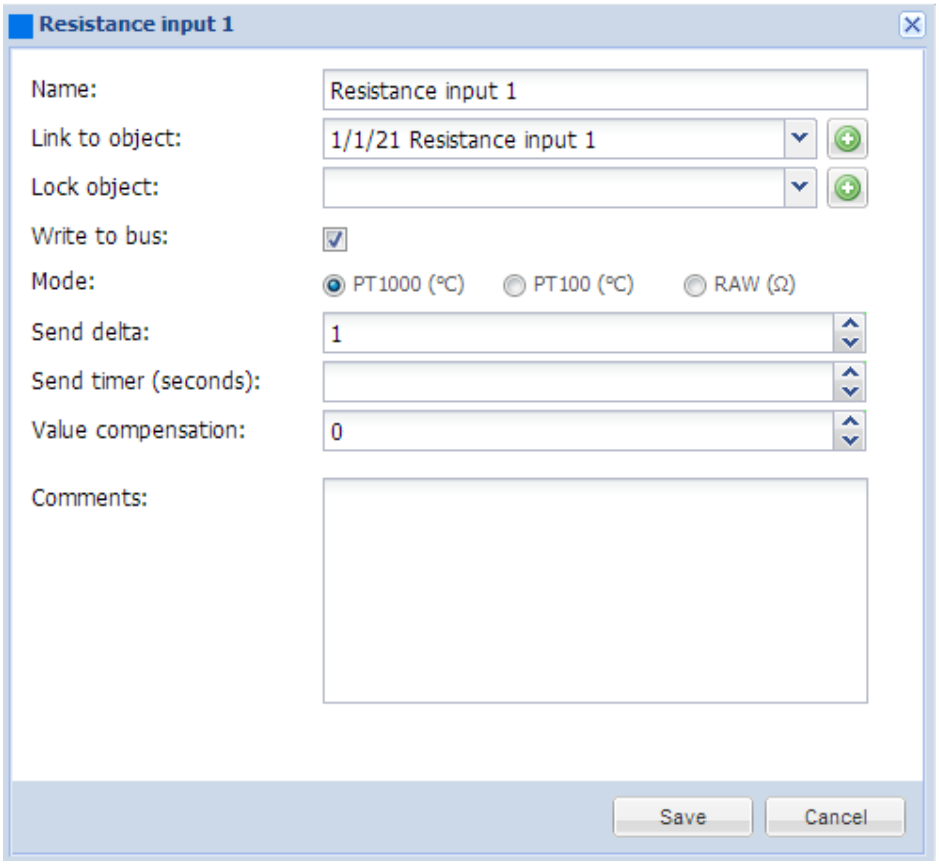
Analog output can be used either as 1byte (0-100%) or 2byte (voltage) output.



- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [0-100%(1byte); Voltage(2byte)]*** – output mode
- ***Minimum voltage*** – minimum voltage value
- ***Maximum voltage*** – maximum voltage value
- ***Transition time (seconds)*** – transition time between two values
- ***Comments*** – comment of the object

1.1.3. Resistive input

Resistive input can be either PT1000 temperature sensor, PT100 temperature sensor or show the data in raw form.

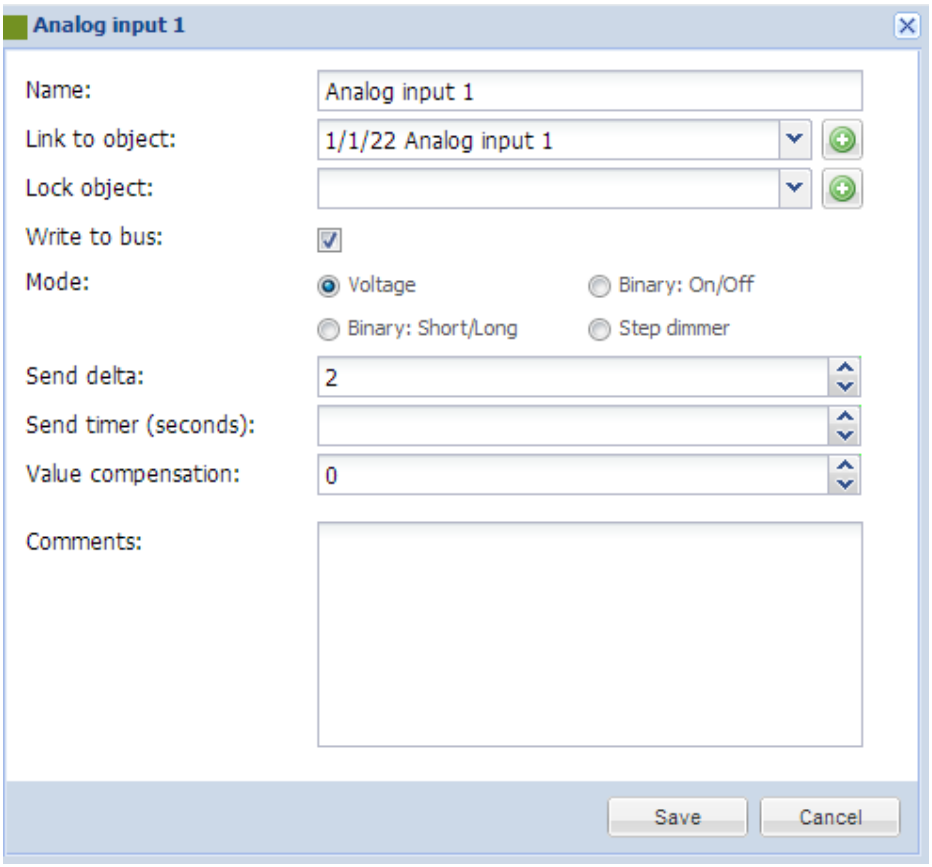


- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [PT1000; PT100, RAW]*** – type of input sensor
- ***Send delta*** – send the data upon specific delta value has changed
- ***Send timer (seconds)*** – time interval to send the reading
- ***Value compensation*** – compensation of the value
- ***Comments*** – comment of the object

1.1.4. Analog input

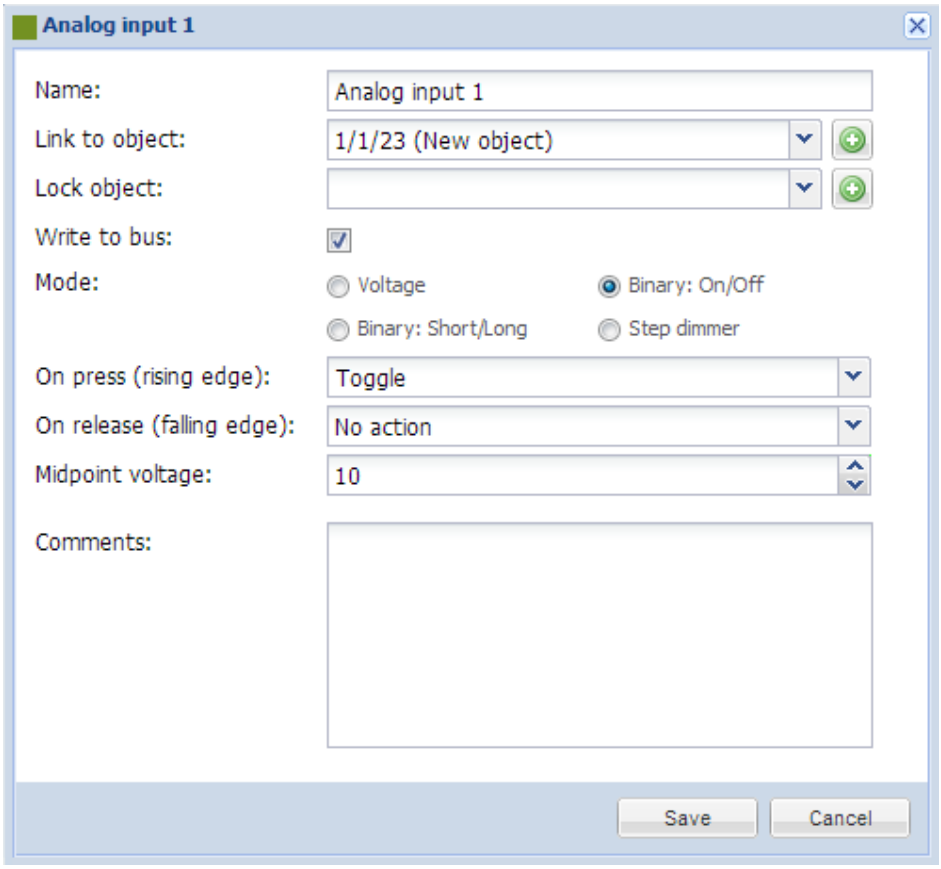
Analog inputs can be used either as Voltage, Binary On/Off, Binary Short/Long or Step dimmer.

Voltage input



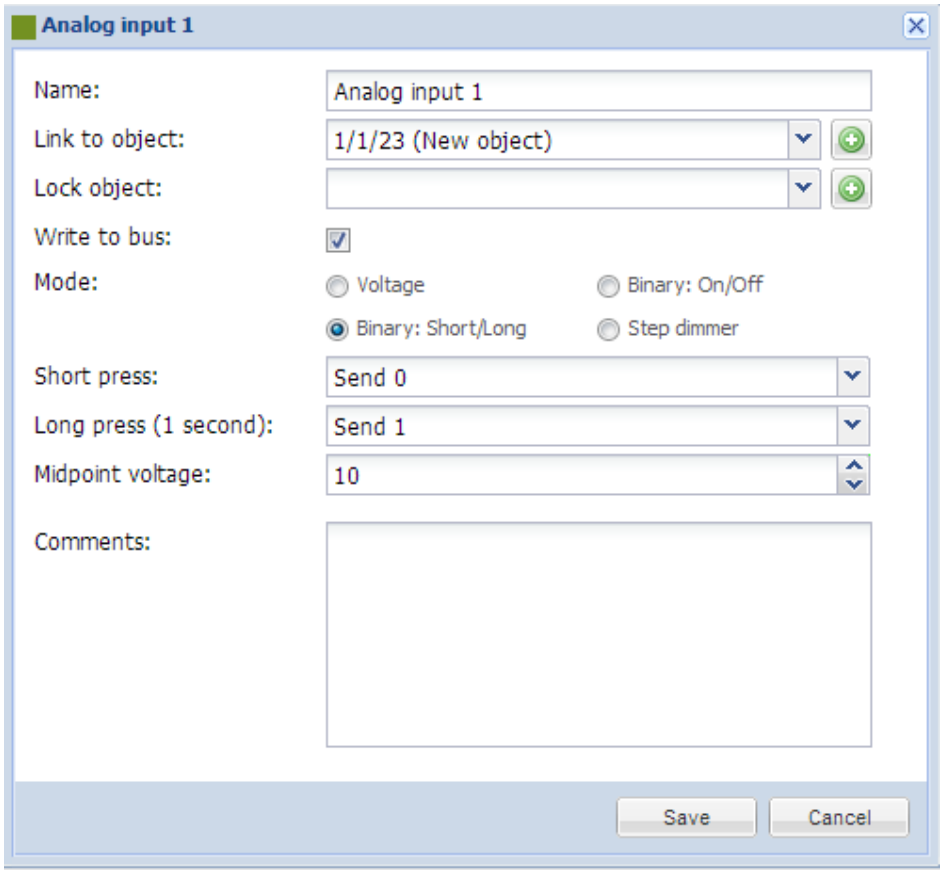
- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [Voltage; Binary On/Off; Binary Short/Long; Step dimmer]*** – input mode
- ***Send delta*** – send the data upon specific delta value has changed
- ***Send timer (seconds)*** – time interval to send the reading
- ***Value compensation*** – compensation of the value

Binary On/Off



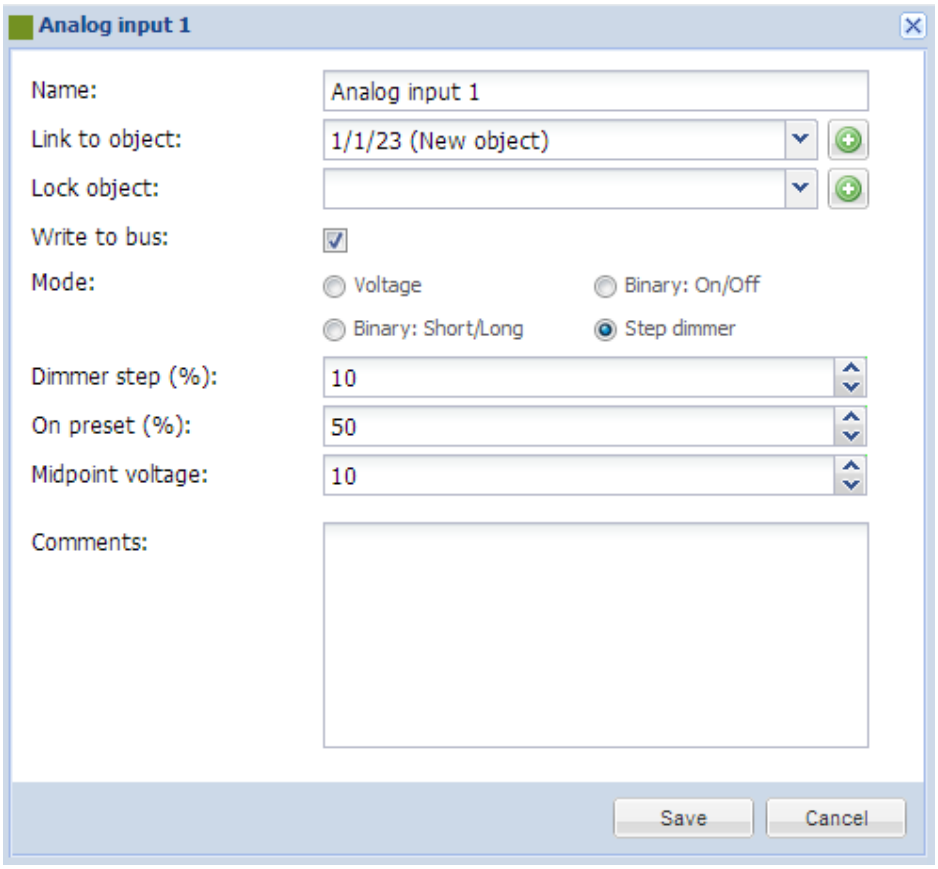
- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [Voltage; Binary On/Off; Binary Short/Long; Step dimmer]*** – input mode
- ***On press (rising edge)[Send 0; Send 1; Toggle]*** – action on rising edge
- ***On release (falling edge) [Send 0; Send 1; Toggle]*** – action on falling edge
- ***Midpoint voltage*** – voltage midpoint to determine the On/Off state

Binary Short/Long



- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [Voltage; Binary On/Off; Binary Short/Long; Step dimmer]*** – input mode
- ***Short press (rising edge)[Send 0; Send 1; Toggle]*** – action on short press
- ***Long press (1 second) [Send 0; Send 1; Toggle]*** – action on long press
- ***Midpoint voltage*** – voltage midpoint to determine the Short/Long state

Step dimmer



- ***Name*** – name of the port
- ***Link to object*** – group address to link the object with. By pressing the button, the field will be automatically filled with the next free group address.
- ***Status object*** – status object group address.
- ***Lock object*** – lock object group address
- ***Write to bus*** – defines either to write or not to bus on value change
- ***Mode [Voltage; Binary On/Off; Binary Short/Long; Step dimmer]*** – input mode
- ***Dimmer step (%)***– value on which the brightness value will change by one step
- ***On preset (%)*** – brightness preset when getting in On state
- ***Midpoint voltage*** – voltage midpoint

1.2. Scripting

Scripting menu allows adding and managing various scripts, depending on the type of the script. Lua programming language is used to implement user scripts. 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 Logic Machine 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

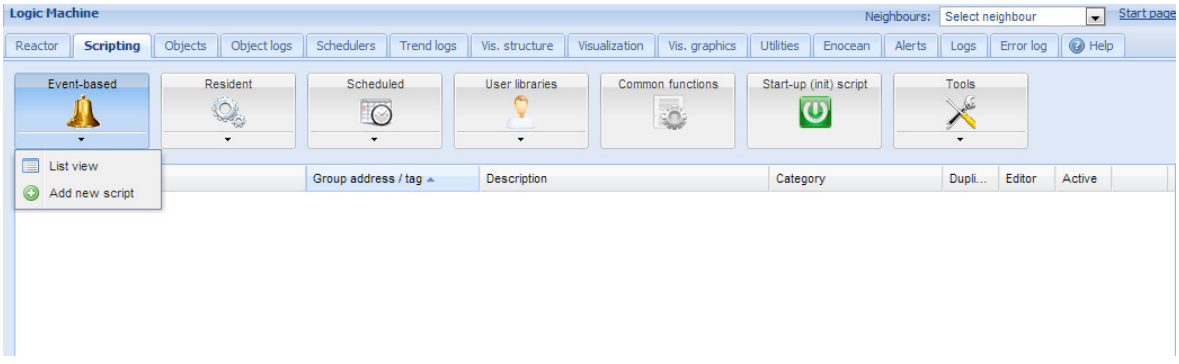
Common functions – common functions to call from other scripts

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

1.1.1. Adding a new script

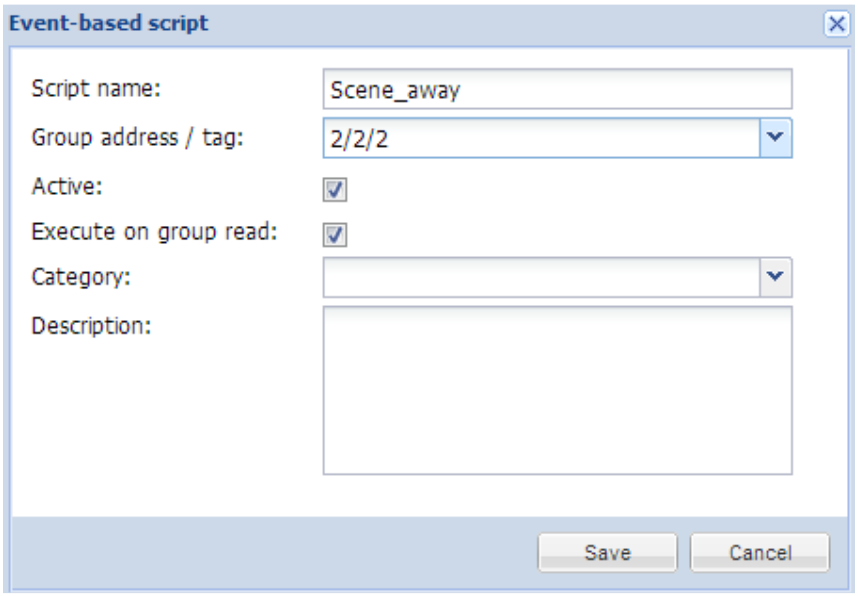
When pressing on the arrow on the lower side of the *Event-based, Resident* or *Scheduled* buttons, two possibilities appear:

List view – sort scripts in list view
Add new script – add new script to the list



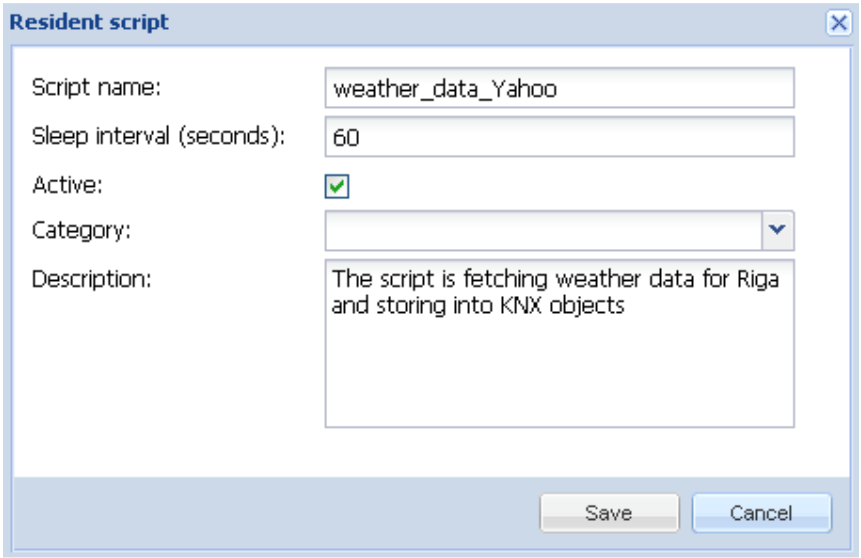
The following fields should be filled when adding a new script:

Event-based



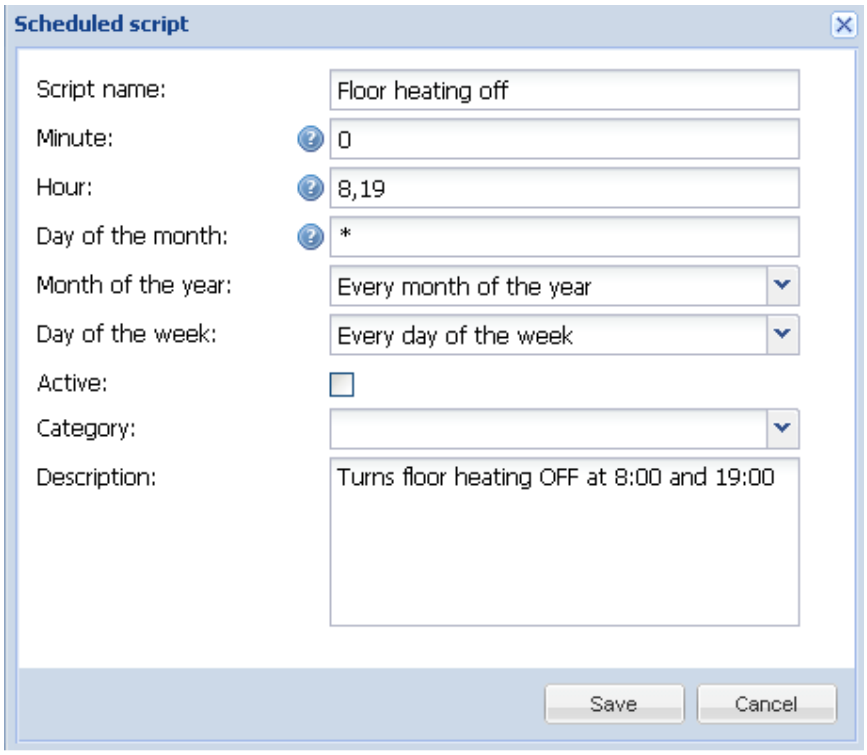
- ***Script name*** – the name of the script
- ***Group address / Tag*** – specific group address or tag name on which the script will be triggered
- ***Active***– specifies whether the script is active (green circle) or disabled (red circle)
- ***Execute on group read***– specifies whether the script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



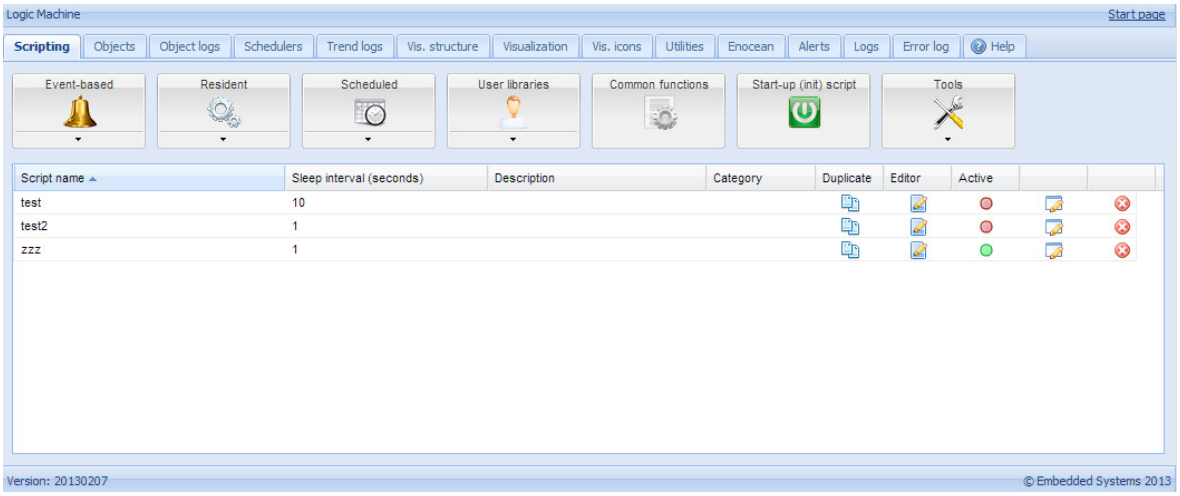
- ***Script name*** – the name of the script
- ***Sleep interval (seconds)*** – interval after which the script will be executed.
- ***Active***– specifies whether the script is active (green circle) or disabled (red circle)
- ***Category*** – a new or existing name of the category the script will be 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



- ***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
Active – Make script active (green) or deactivate it (red)
Edit – Edit script name, description, category and other parameters
Delete – Delete the script. When pressing this icon the confirmation is asked to accept the delete.

1.1.2. Event-based scripting

Event-based scripting can be used to implement custom logic for group address 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()**

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.

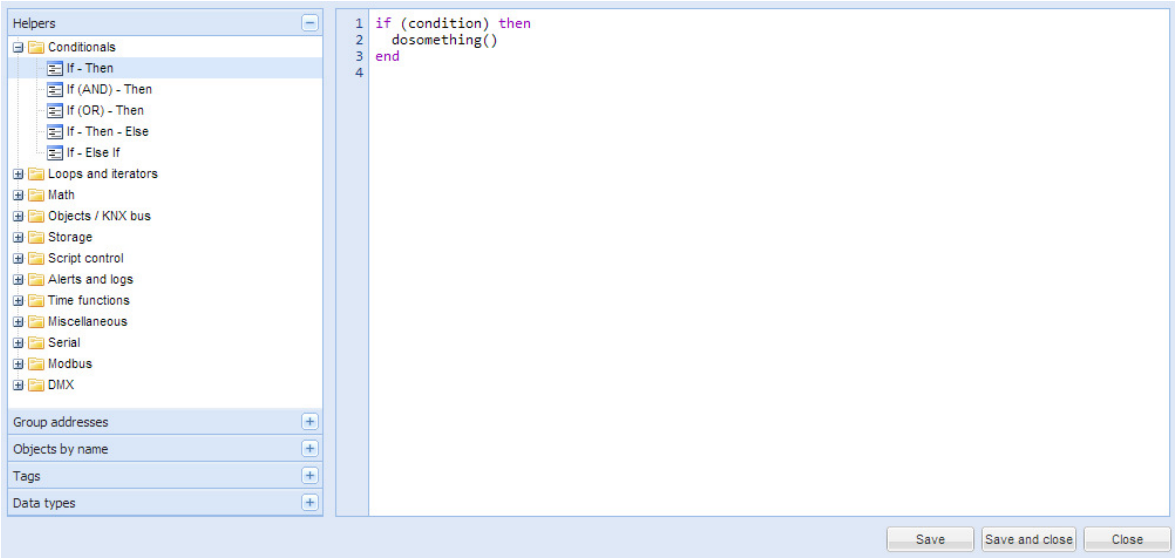
N — execute script exactly at N minute, hour or day.

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 five main groups of Script editor:

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

- *Conditionals* – If Else If, If Then etc.
- *Loops and iterators* – Array, Repeat..Untiletc
- *Math* – Random value, Ceiling, Absolute value, Round etc.
- *Objects/KNX bus* – Get object value, Group read, Group write, Update interval etc.
- *Storage* – Get data from storage, Save data to storage
- *Script control* – Get other script status, enable or disable other scripts
- *Alerts and logs* – Alert, Log variables, Formatted alert
- *Time functions* – Delay script execution
- *Miscellaneous* – Sunrise/sunset etc.
- *Serial* – Communication through internal Logic Machine IO ports
- *Modbus* – Create RTU/TCP connection, Write register, Read register etc.
- *DMX* – Communication with DMX devices

Group addresses – existing group addresses on the KNX bus
Objects by name – chose object by name
Tags – choose object by tag
Data types – choose object by data type

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.if myobject then
4.myobject:write(1)-- update object value with 1
5.end
```

Find object by address and write new value.

```
1.myobject=grp.find('1/1/15')
2.-- verify that the requested object was found
3.if myobject then
4.myobject:write(52.12, dt.float16)-- explicitly set data type to dt.float16 (2-byte
   floating point)
```

```
5. end
```

Switch all binary objects tagged 'lights' off.

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

Group write to the specified group address and data type.

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

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.

```
result =storage.get('my_stored_value_3')-- returns nil if value is not found
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.

```
objectdata={}
objectdata.temperature=23.1
objectdata.scene='default'
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.

```
objectdata=storage.get('test_object_data')
if type(objectdata)=='table'then
if objectdata.temperature> 24 then
-- do something if temperature level is too high
end
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

```
temperature = 25.3
if temperature > 24 then
-- resulting message: 'Temperature levels are too high: 25.3'
  alert('Temperature level is too high: %.1f', temperature)
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 metatable 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 string s. 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 captured 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)
      return loadstring(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 and 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 and between any number of variables

bit.bxor (x1 [, x2...])
Binary and between any number of variables

bit.lshift (value, shift)
Left binary shift

bit.rshift (value, shift)
Right binary shift

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 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 adress 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 adress. 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)

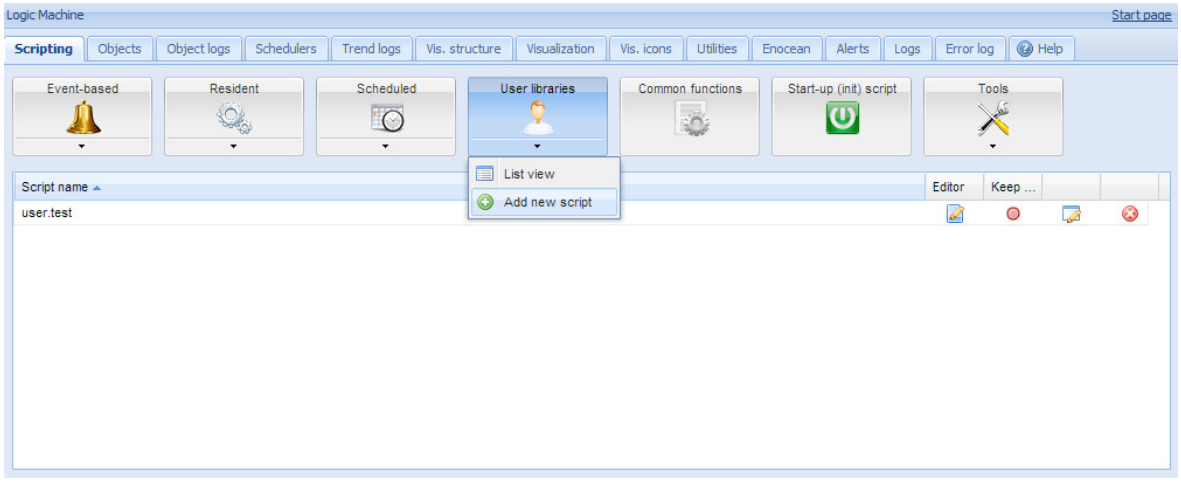
Receives an argument of any type and converts it to a string in a reasonable format. For complete control of how numbers are converted, use *string.format*.
If the metatable of e has a "__tostring" field, then *tostring* calls the corresponding value with e as argument, and uses the result of the call 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

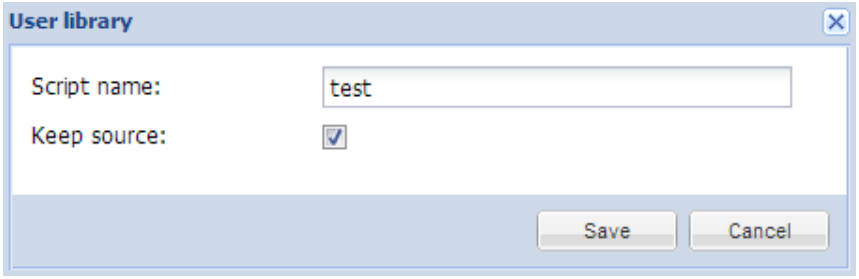
User libraries usually contain user defined functions which are later called from other scripts.



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

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.



Include the library in the scripts

To use functions defined in user library, they should be included in the beginning of the script, for example, user library with the name ‘test’ should be included like this:

require('user.test')

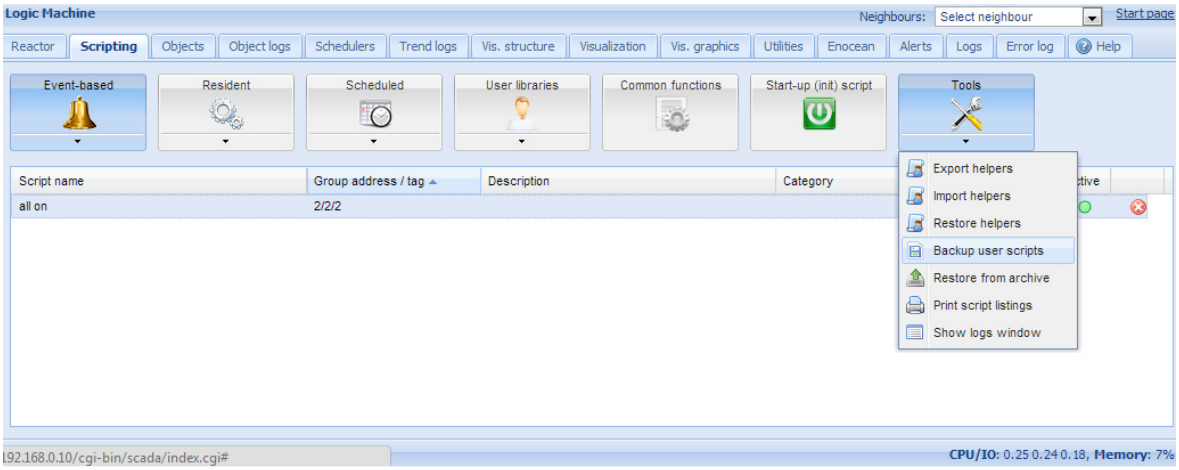
1.2.30.Common functions

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 C*ommon function*s.

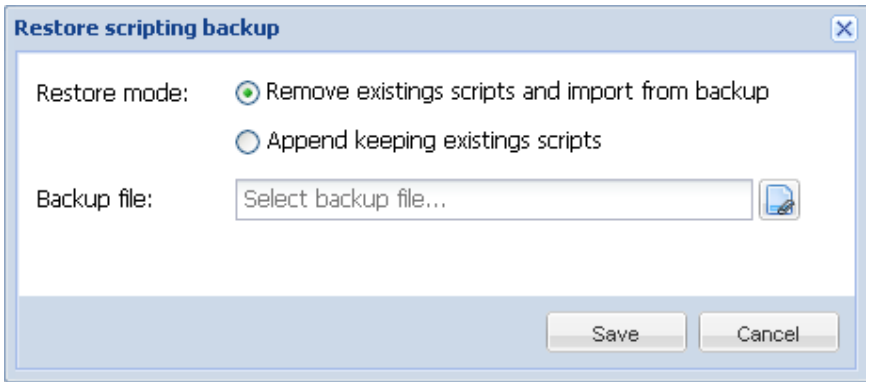
1.2.31.Start-up (init) script

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

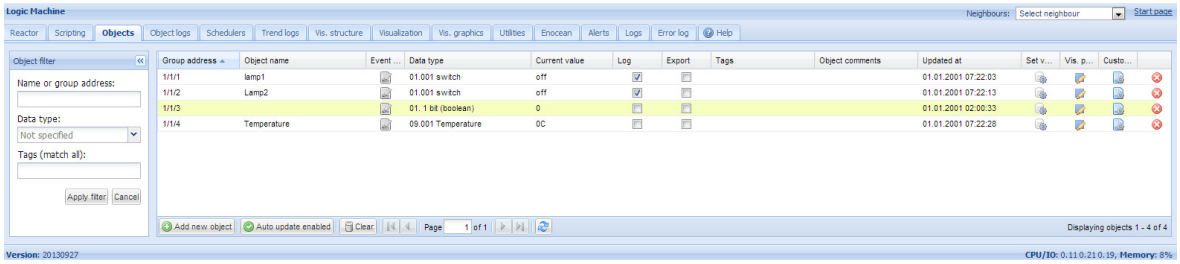
```
-- if object exists "presence" variable will be a table, nil otherwise
presence = knxobject.get('address', '0/0/1')

-- 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

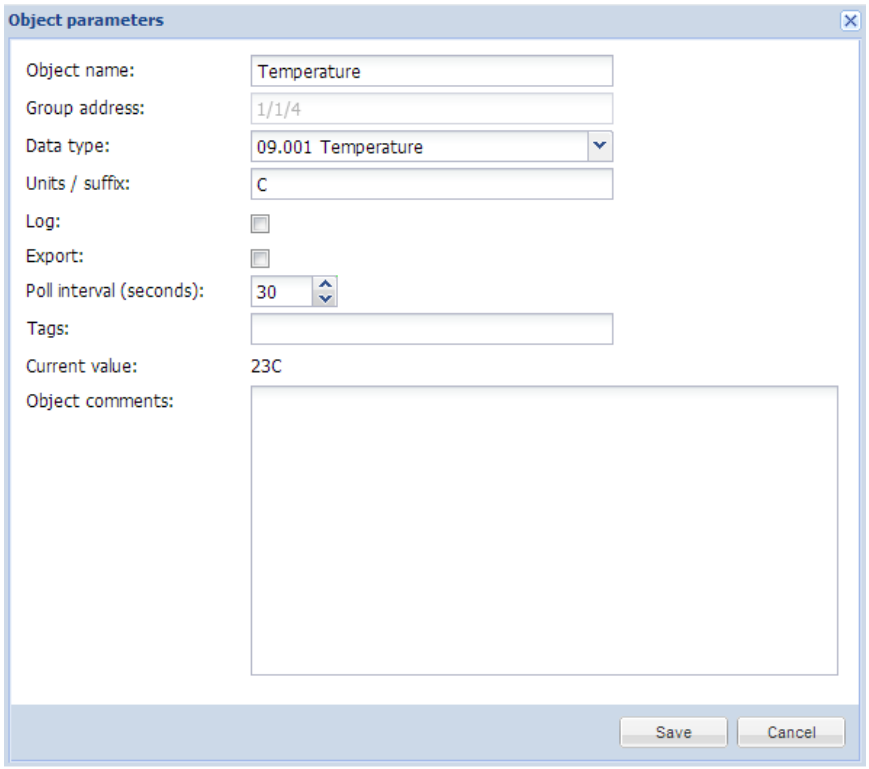
List of KNX network objects appears in *Objects* menu. The object appears in the list by way of:

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



1.3.1. Object parameters

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



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

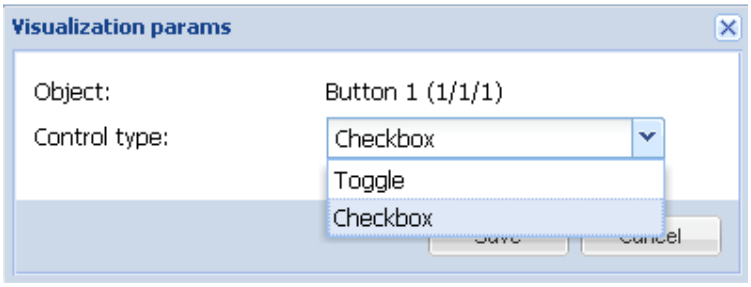
- ***Export*** – Make object visible by remote XML requests and in BACnet network (if KNX – BACnet gateway functionality is used)
- ***Poll interval (seconds)*** – perform automatic object read after some time interval
- ***Tags*** – assign this object to some tag which can be later used in writing scripts, 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

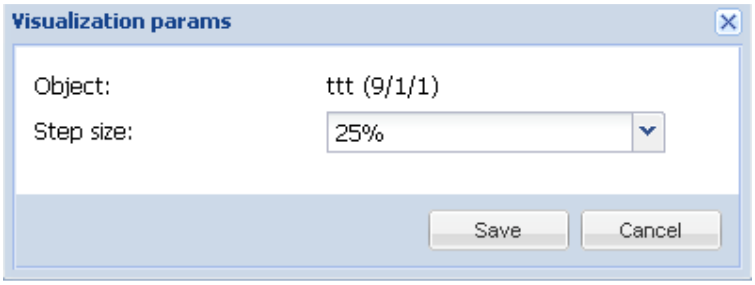
By pressing on the  button of the corresponding object you can set specific visualization parameters for this type of object.

1 bit



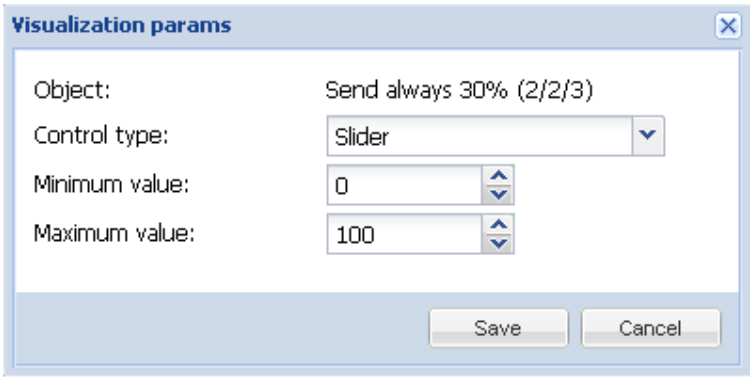
- ***Control type*** – type of the visual control element
  - Toggle 
  - Checkbox 

4 bit (3 bit controlled)



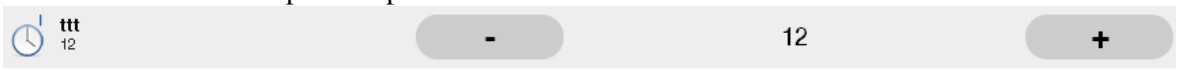
- ***Step size*** – step size for example for blinds control

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



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

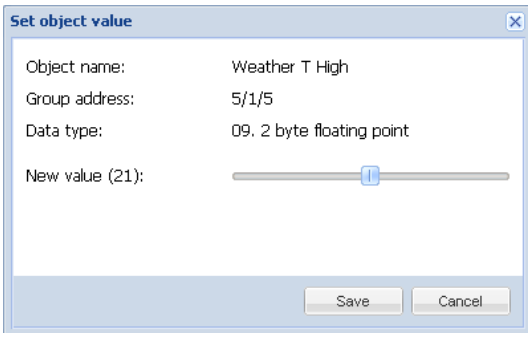
  - Direct input / Step +/-

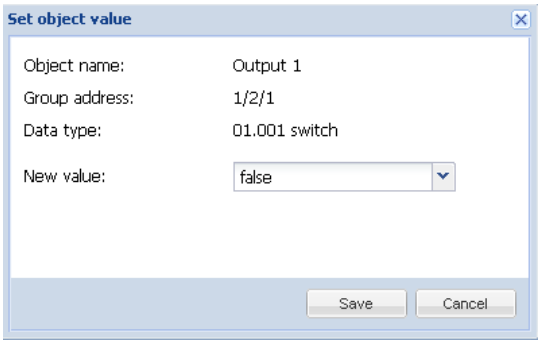


- Minimum value
- Maximum value

1.3.3. Change the object state

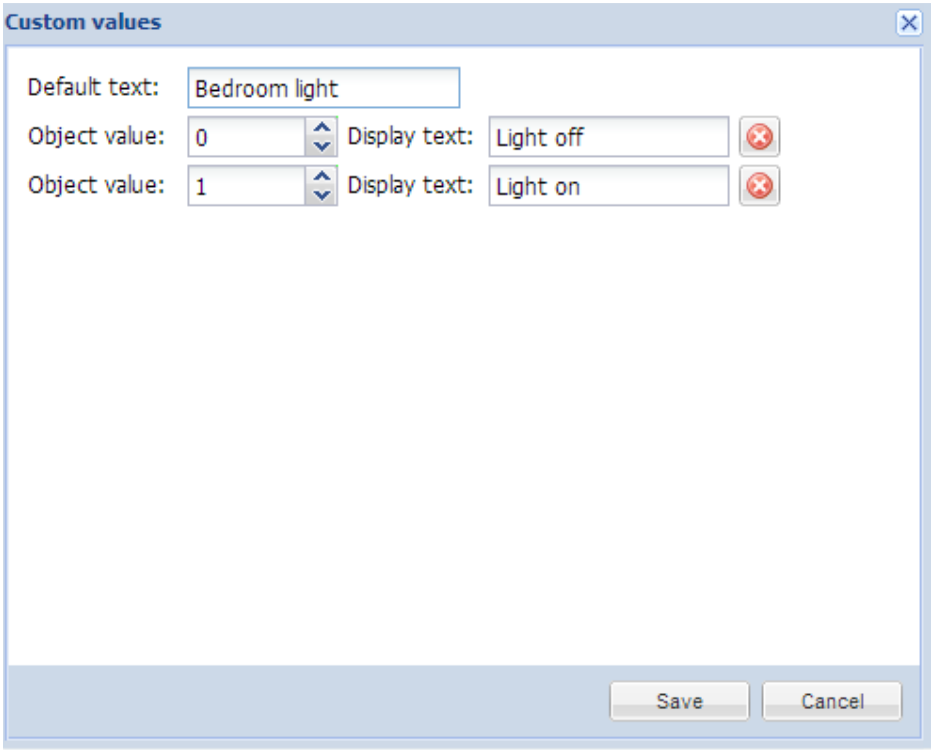
In the object list, by pressing on the  button, you can change the state of the object. The appearance of the *New value* depends on what visualization parameters are set for specific object.





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)

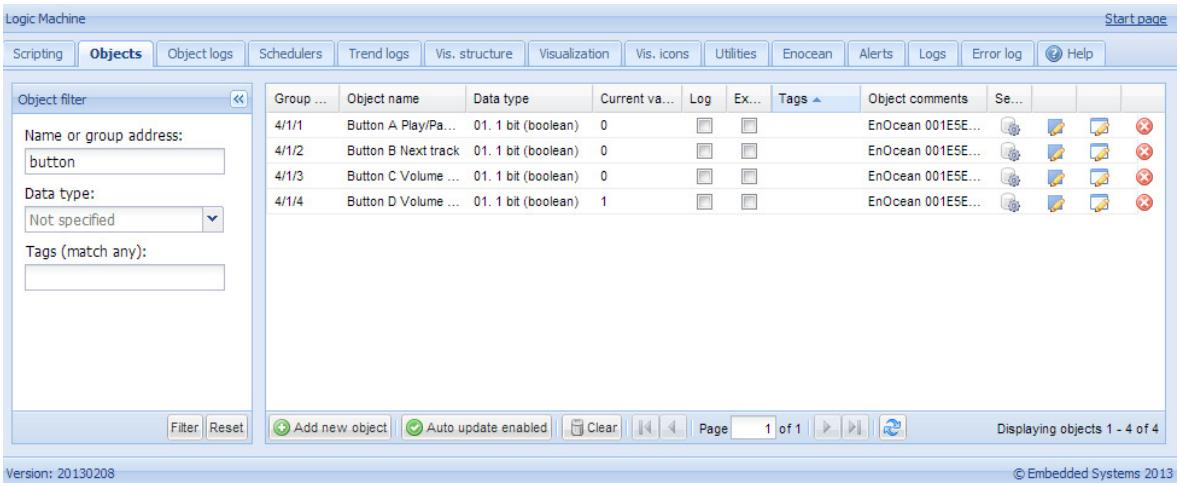


1.3.5. Object control bar

- ***Add new object*** – Manually add new object to the list
- ***Auto update enabled*** –Specifies either the object list is updated automatically or not
- ***Clear*** – Clear the list of group addresses
- ***Next/Previous page*** – move to next or previous page
- ***Refresh*** – refresh the object list

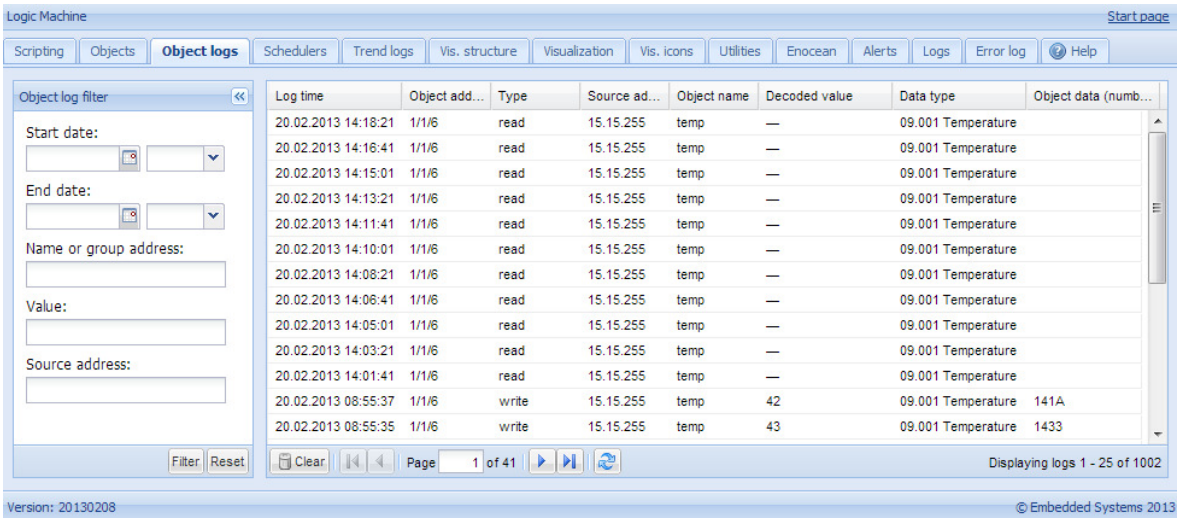
1.3.6. Filter objects

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.



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
- ***Value*** – specific object value
- ***Source address*** – specific source address

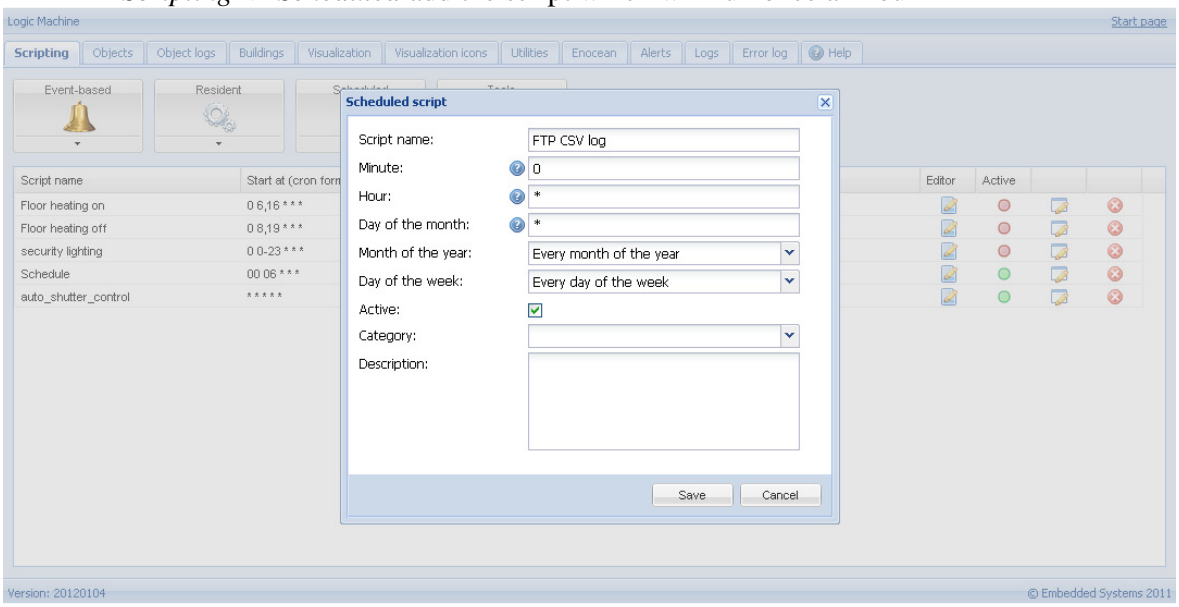
You can clear all logs by pressing on *Clear* button.

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 ‘ftplogin‘, 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.

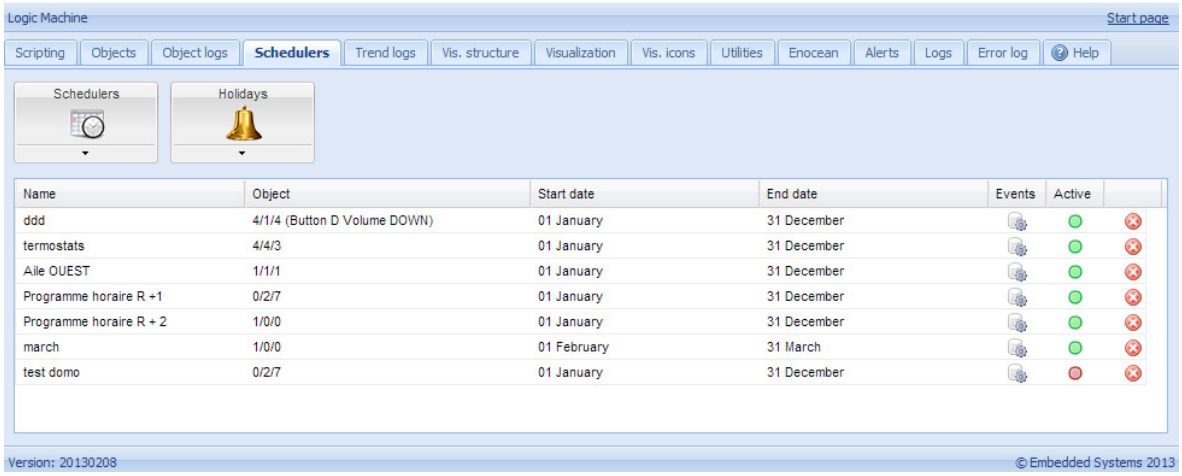
```
require('socket.ftp')

-- ftp file
ftpfile=string.format('ftp://ftplogin:ftppassword@192.168.1.11/%s.csv', os.date('%Y-%m-%d_%H-%M'))
-- get past hour data (3600 seconds)
logtime=os.time() - 60*60

-- list of objects by id
objects ={}

-- objects with logging enabled
query ='SELECT address, datatype, name FROM objects WHERE disablelog=0'
for _, object inipairs(db:getall(query))do
  objects[tonumber(object.address)]={
datatype=tonumber(object.datatype),
    name =tostring(object.name or''),
}
end

-- csv buffer
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[tonumber(row.address)]
27.
28. -- found matching object and event type is group write
29. if object androw.eventtype=='write'then
30. datatype=object.datatype
31.
32. -- check that object datatype is set
33. ifdatatypethen
34. -- decode data
35.       data =knxdatatype.decode(row.datahex, datatype)
36.
37. -- remove null chars from char/string datatype
38. ifdatatype==dt.charordatatype==dt.stringthen
39.         data =data:gsub('%z+', '')
40. -- date to DD.MM.YYYY
41. elseifdatatype==dt.datethen
42.         data =string.format('%.2d.%.2d.%.2d', data.day, data.month, data.year)
43. -- time to HH:MM:SS
44. elseifdatatype==dt.timethen
45.         data =string.format('%.2d:%.2d:%.2d', 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
```

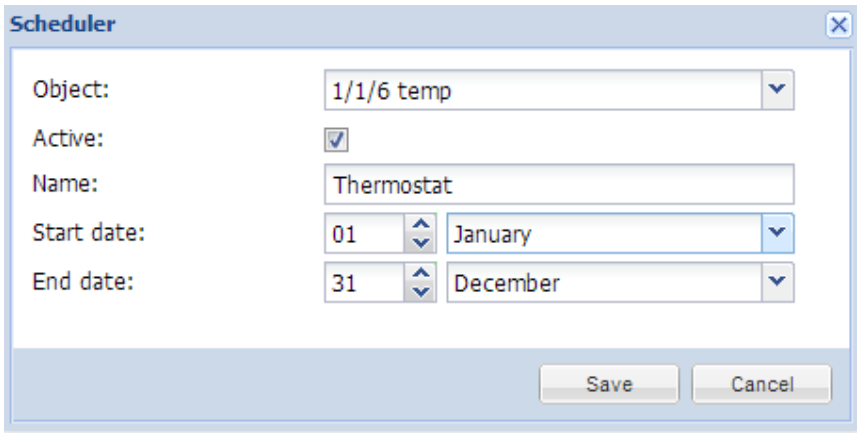
```
66. if err then
67.   alert('FTP upload failed: %s', err)
68. end
```

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.

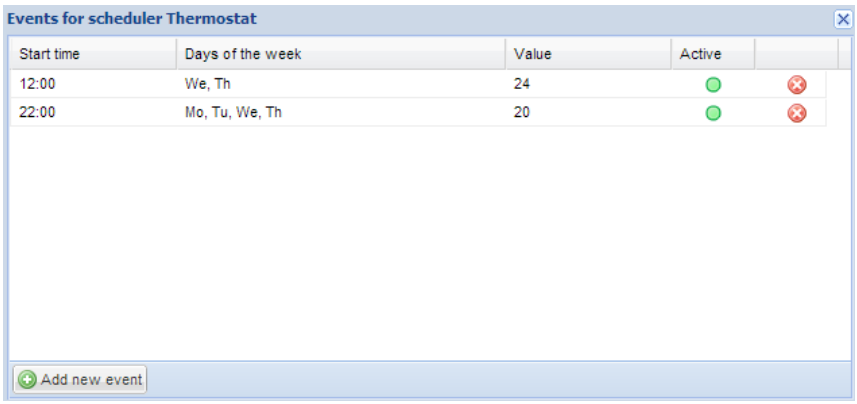


1.5.1. Add new scheduler

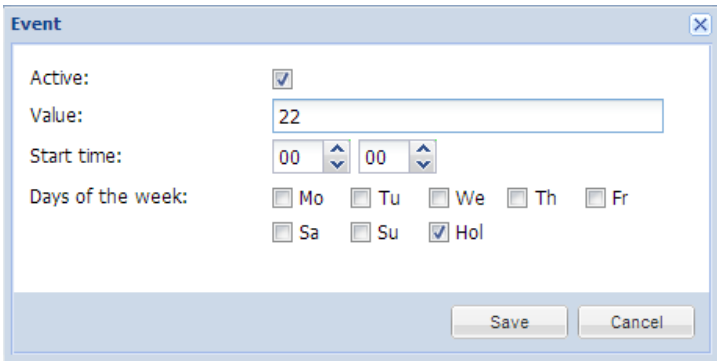


- ***Object*** – the object group address which will be controlled by scheduler
- ***Active*** – define this scheduler as active or not
- ***Name*** – name of the scheduler
- ***Start date*** – start date of the scheduler
- ***End date*** – end date of the scheduler

1.5.2. Scheduler events



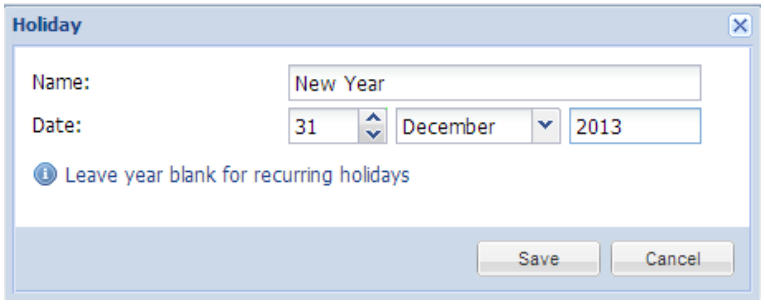
Event can be added both in administrator interface as well as by end user in the special *User mode schedulers* interface.



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

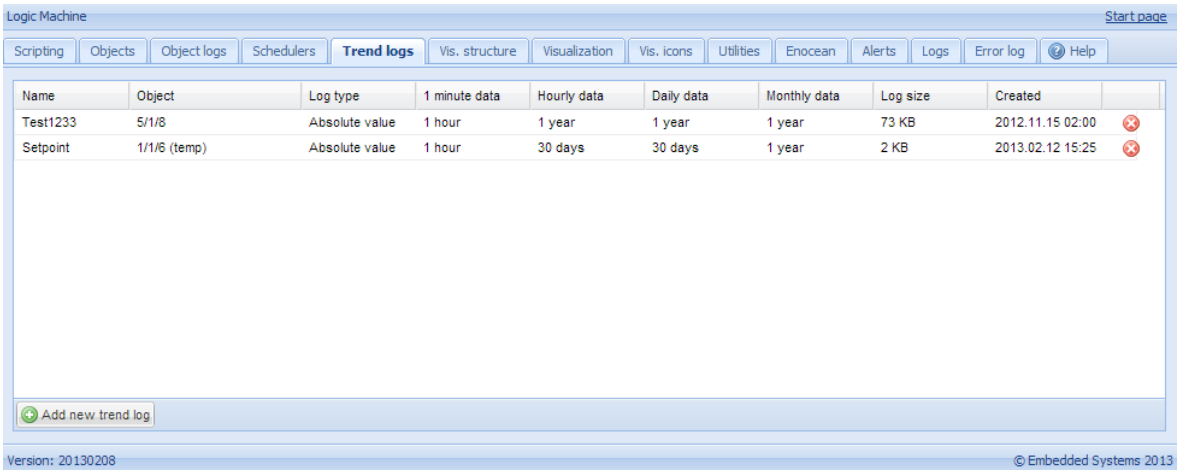
Once the event will be marked to run in *Hol*, Holiday entries will be activated.



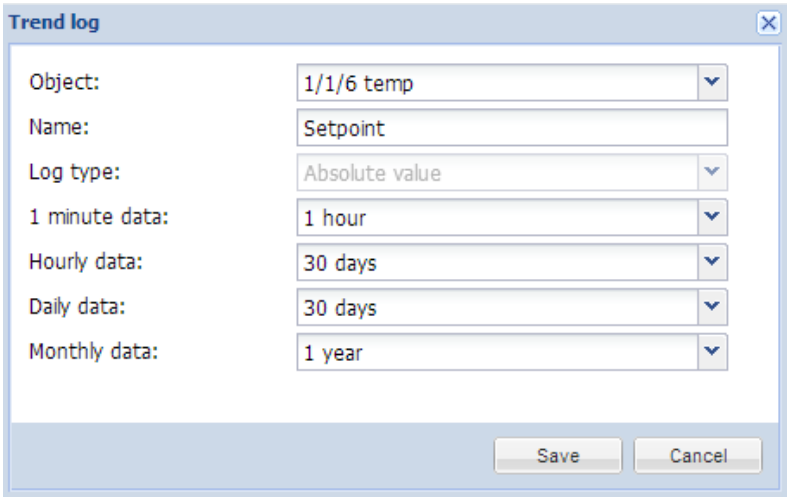
Name – the name of the holiday entry
Date – date of the holiday

1.6. Trend logs

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



1.6.1. Add new trend log



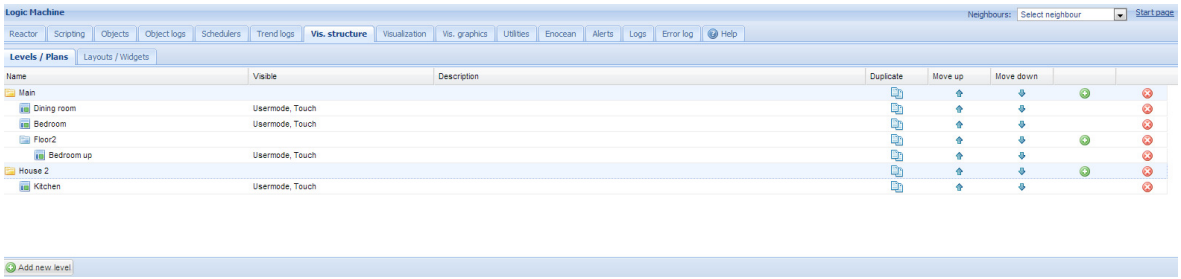
Object – choose from list of object the one to make trends for
Name – name of the trend
Log type [Counter, Absolute value] – type of the log. *Counter* type is used to count the date, *Absolute value* – saves the actual readings
1 minute data – average value of 1 minute for specific time interval data 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

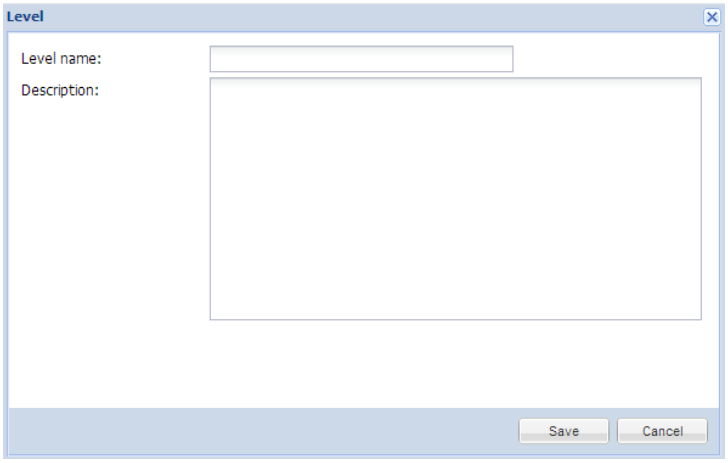
Note! One trend data point reading takes *8 bytes* of flash memory. E.g. reading some value once in every 10 minutes, will consume ~0.4MB of flash each year.

1.7. Visualization structure

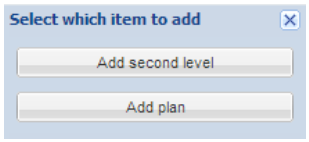
In *Vis.structure* menu the structure of the visualization is defined and visualization backgrounds are uploaded.

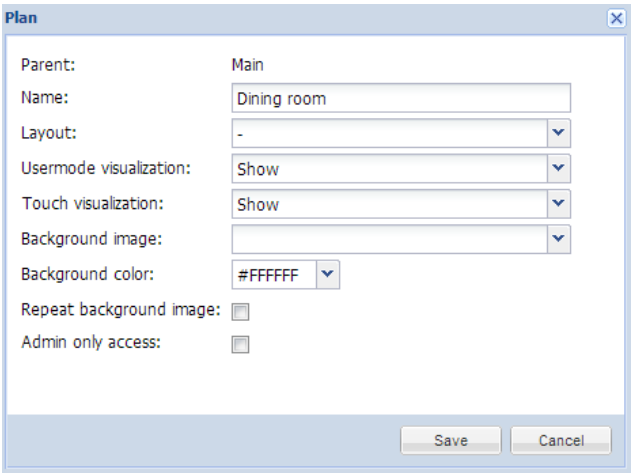


By default there is *Main* level added. To add a new level/building, press "*Add new level*" button.



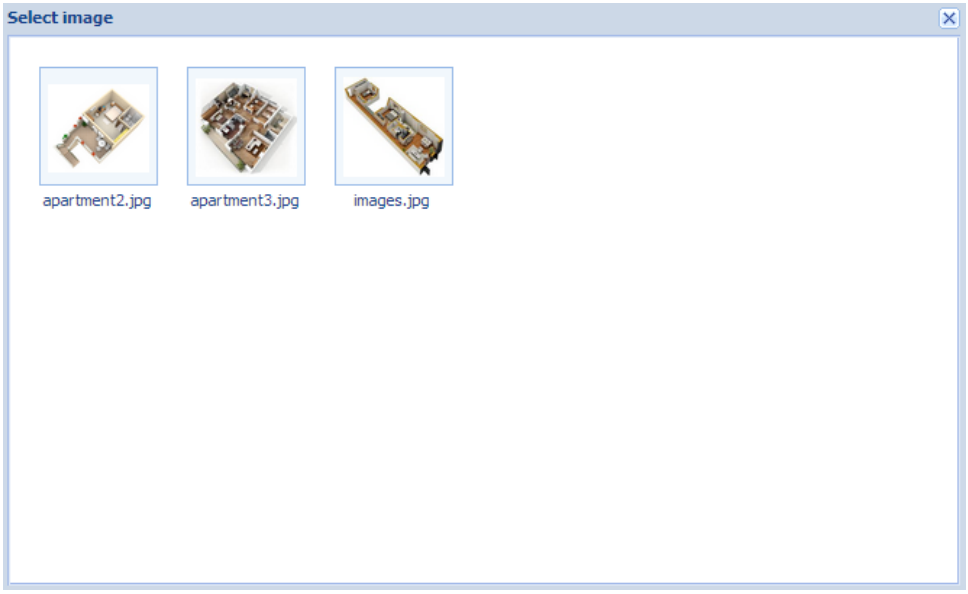
Once a new level is added, you can add second level or upload floor pictures related to this particular building. To add a new entry, click on the green icon, to delete a specific entry press on the red icon.





- ***Plan name*** – name for the plan
- ***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
- ***Background image*** – choose background image from the list added in *Vis.graphics → Images/Backgrounds*
- ***Background color*** – choose background color of the plan
- ***Repeat background image*** – either to show the image once or repeat it and fill the whole plan
- ***Admin only access*** – enable admin only access for this floor

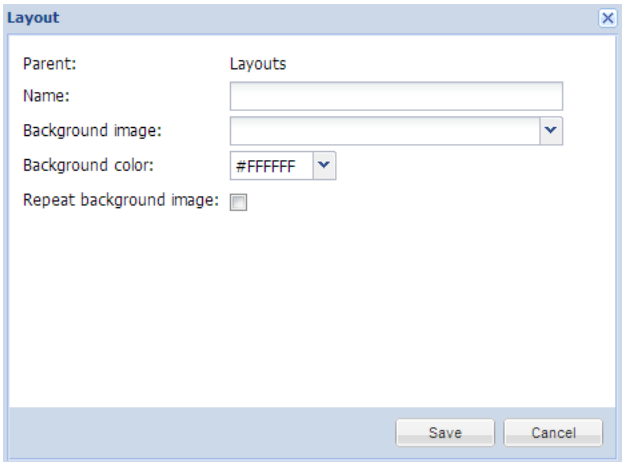
When clicking on Background image, the following window appears with background images which has to be added in *Vis.graphics → Images/Backgrounds* in advance:



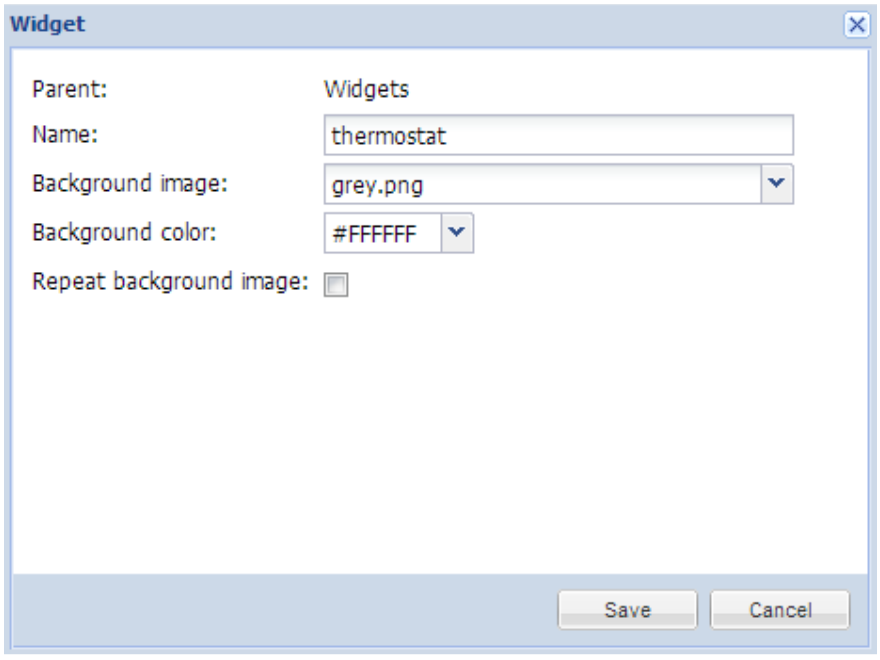
You can duplicate the plan with all its objects and settings by pressing on icon.
Levels can be sorted by pressing and icons.

1.7.1. Layouts / Widgets

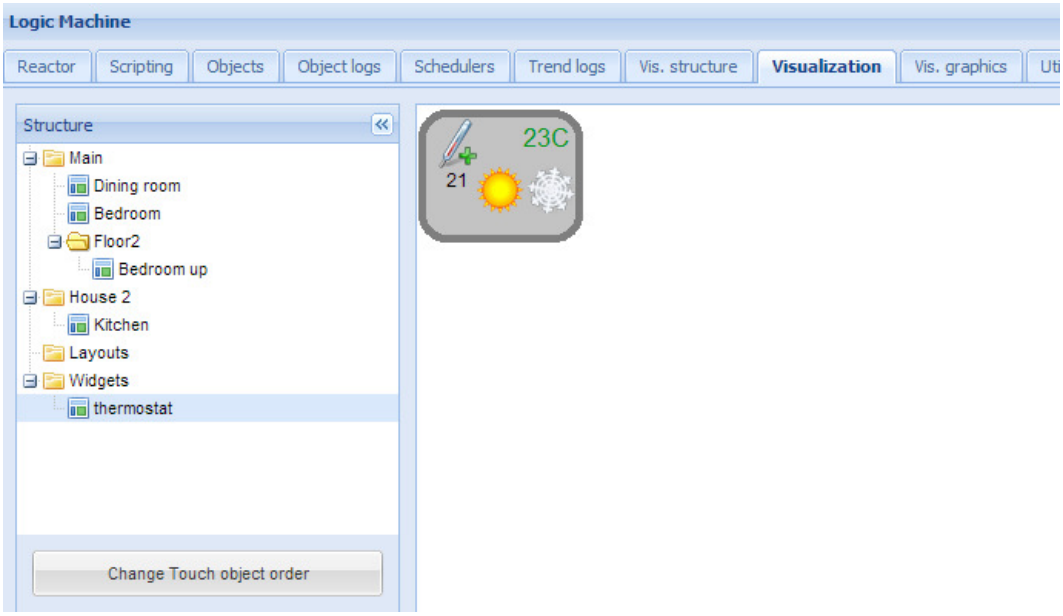
Layouts are used as templates for further use when adding *Levels* in *Levels/Plans* tab.
Layouts will not be visible from the Usermode/Touch visualizations. When you add any background, objects to layouts level in *Visualization*, they will automatically appear on all linked Levels.



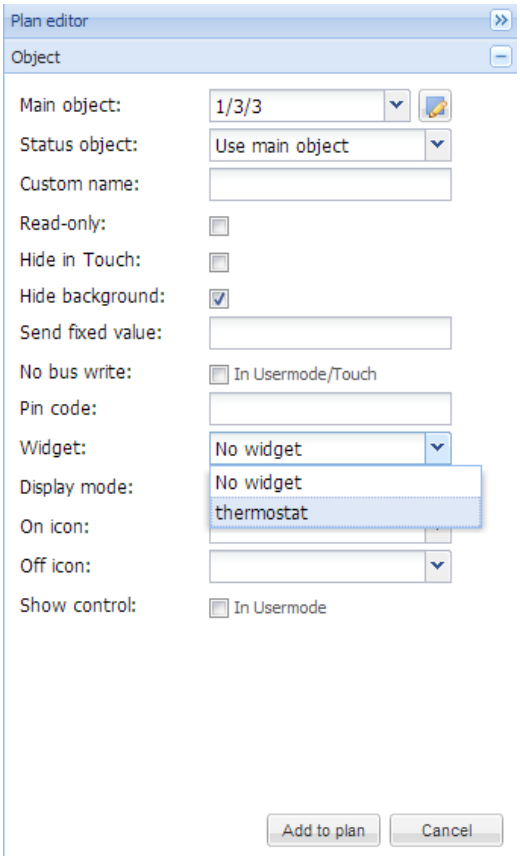
Widgets are used to combine several objects under one object in visualization.
Background image for the widget should be added in *Vis.graphics* → *Images/Backgrounds* in advance.



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.

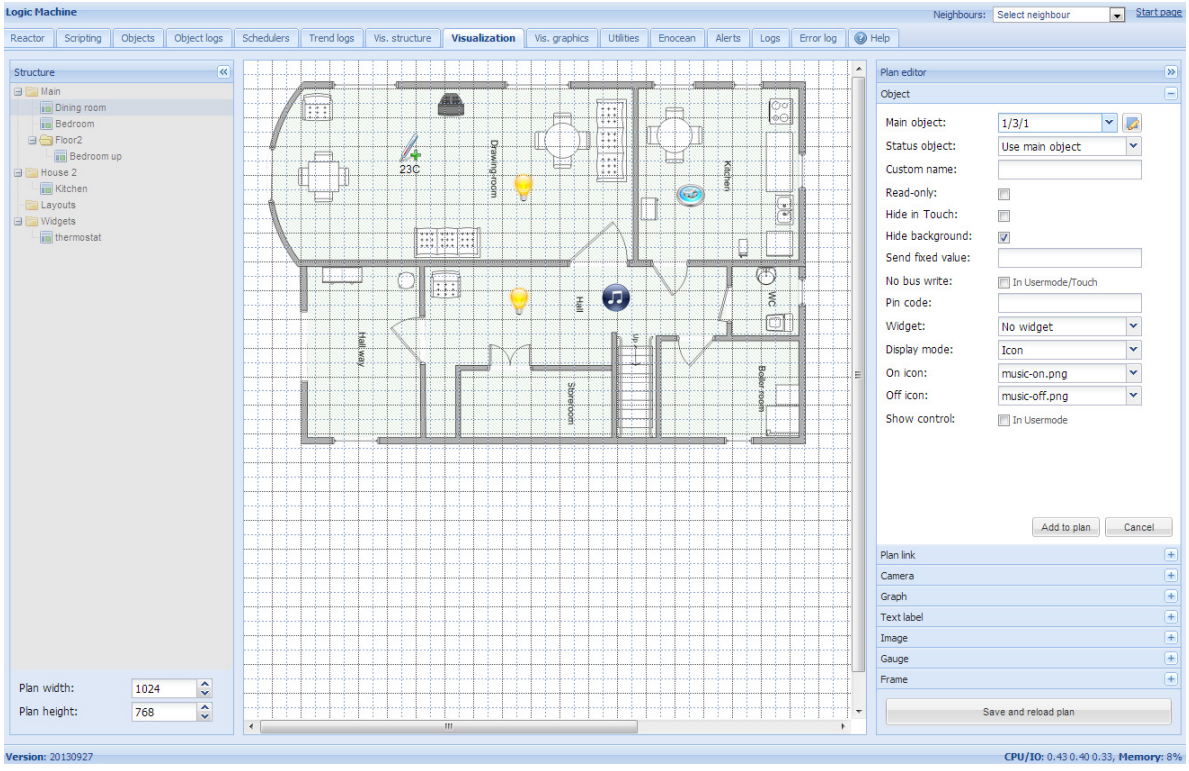


Once added, you can try out the widget in *Usermode visualization* by clicking on added object (temperature sensor icon on the left), the widget appears on click.

1.8. Visualization

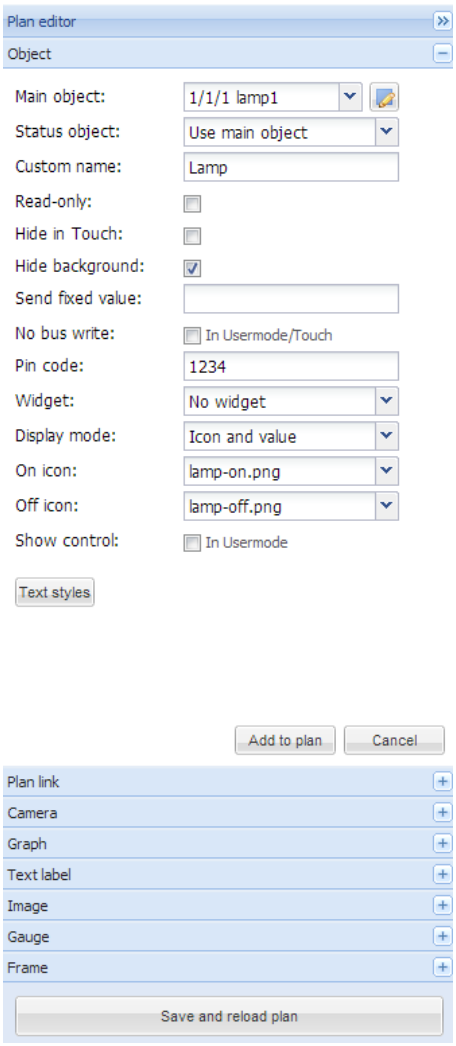
After the building and floor structure is defined in Vis.structure tab, it is visualized in *Visualization* tab. Controlled and monitored objects can be added and managed in this section.

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



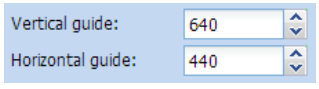
1.8.1. Plan editor

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:



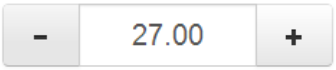
- *Object* – new object to be added to the map
- *Plan link* – linking several floors with special icons
- *Camera* – IP web camera integration into visualization
- *Graph* – Real-time graph to monitor value of scale-type objects
- *Text Label* – text label to put on visualization
- *Image* – Add specific image on the visualization
- *Gauge* – Metering gauge
- *Frame* – add frame object to the visualization

On the left side of the plan *Vertical guide* and *Horizontal guide* fields appears, once the plan editor is unlocked. This is used to see guidelines for adapting specific plan to specific device resolution.

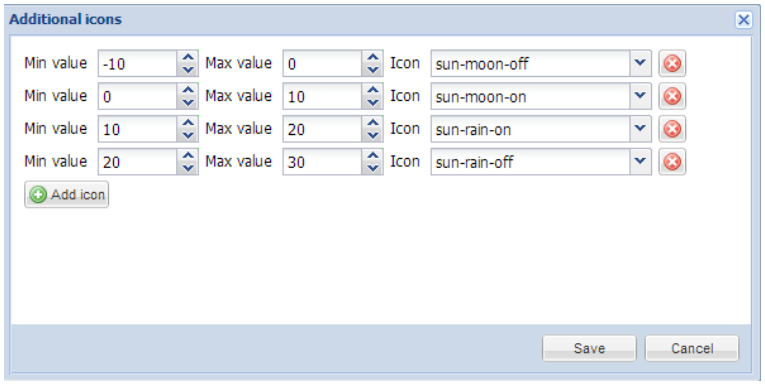


1.8.2. Object

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



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.

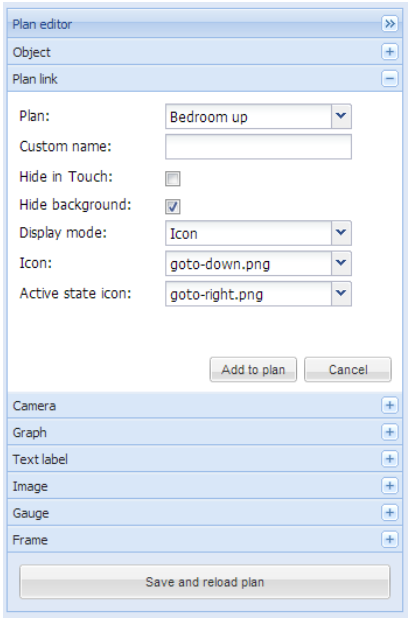


Once the object parameters are defined, press *Add to plan* button and newly created object will appear. You can move the object to the location it will be located. Note that while being in editing mode, the object will not work. When all necessary objects are added, press *Save and reload plan* button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

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

1.8.3. Plan 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.

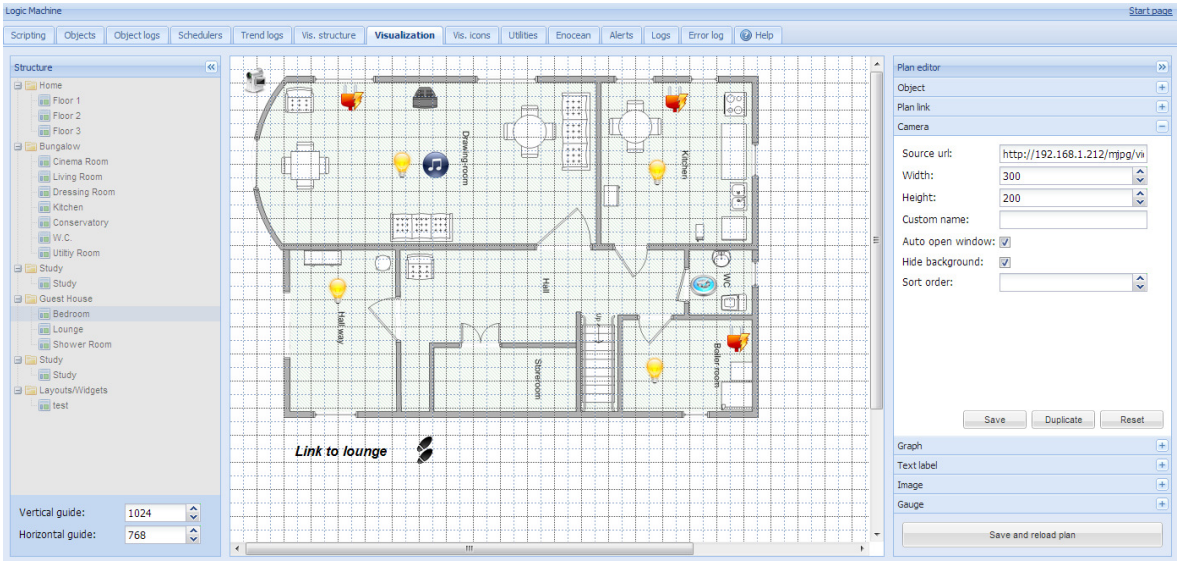


- ***Plan*** – Linked plan name or link to Schedulers / Trends
- ***Custom name*** – name for the link
- ***Hide background***– Hide icon background
- ***Display mode [Icon; Value]*** – either to show icon or its value
- ***Icon*** – Icon which will be showed in visualization (if chosen, no further parameters 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)
- ***Font size*** – size of font
- ***Text style*** – text style – bold, italic, underscore
- ***Custom font*** – font name
- ***Font color*** – font color

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. Camera

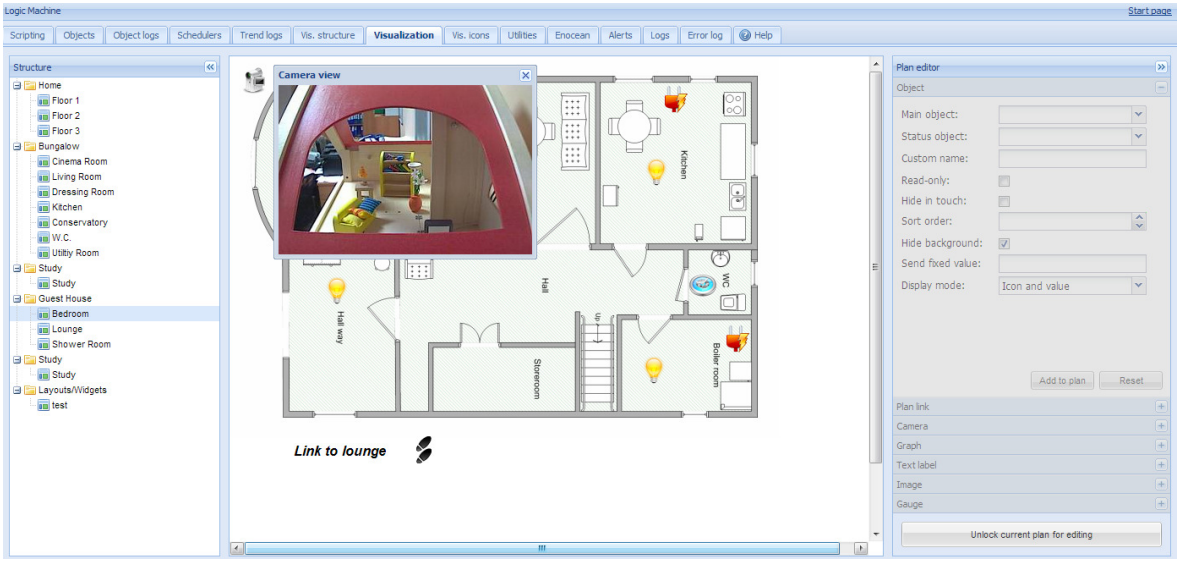
Logic Machine supports third party IP web camera integration into its visualization.



- ***Source url*** – source address of the video stream
- ***Width*** – sub-window width for displaying of picture
- ***Height***– sub-window height for displaying of picture
- ***Custom name*** – name for the object
- ***Icon*** – icon for the object
- ***Auto open window*** – automatically open video window
- ***Hide background***– hide icon background

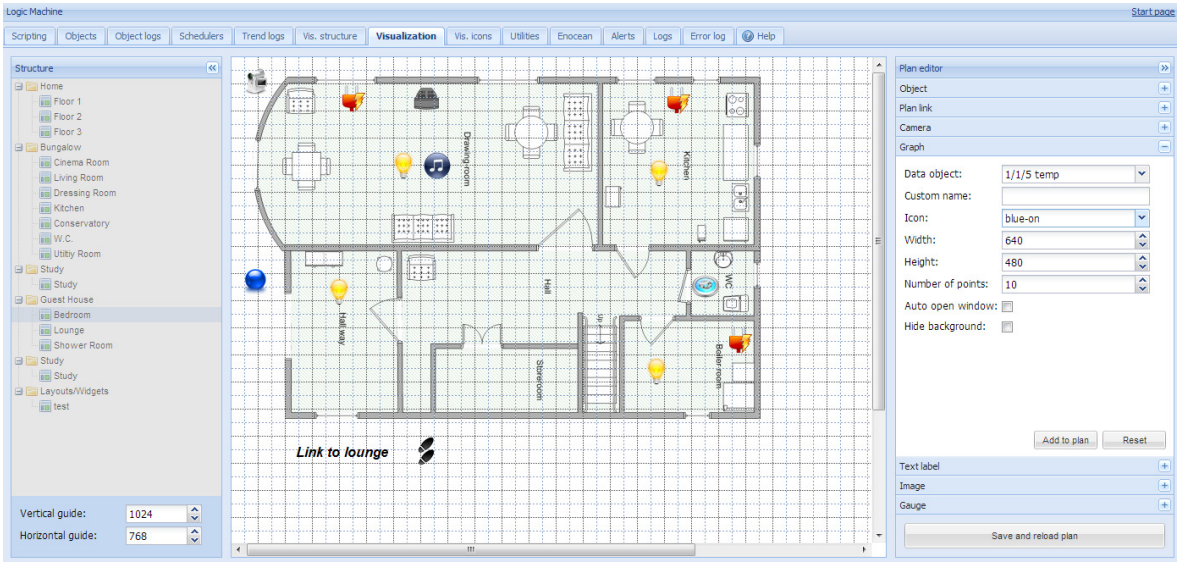
Note! If IP camera requires user name and password, enter the url in form ***http://USER:PASSWORD@IP***

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.



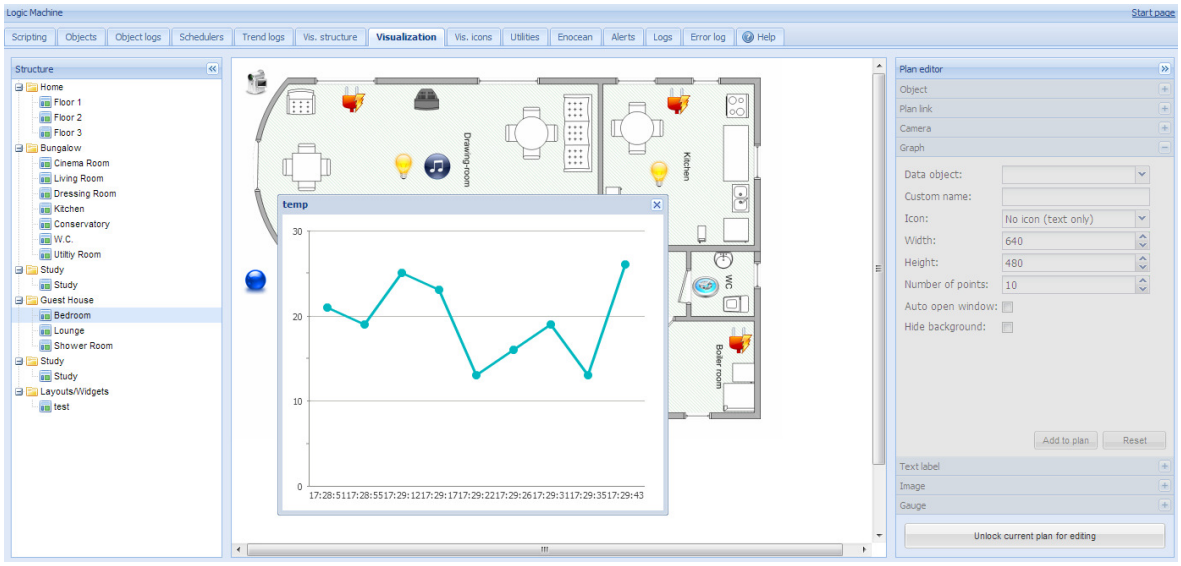
1.8.5. Graph

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.



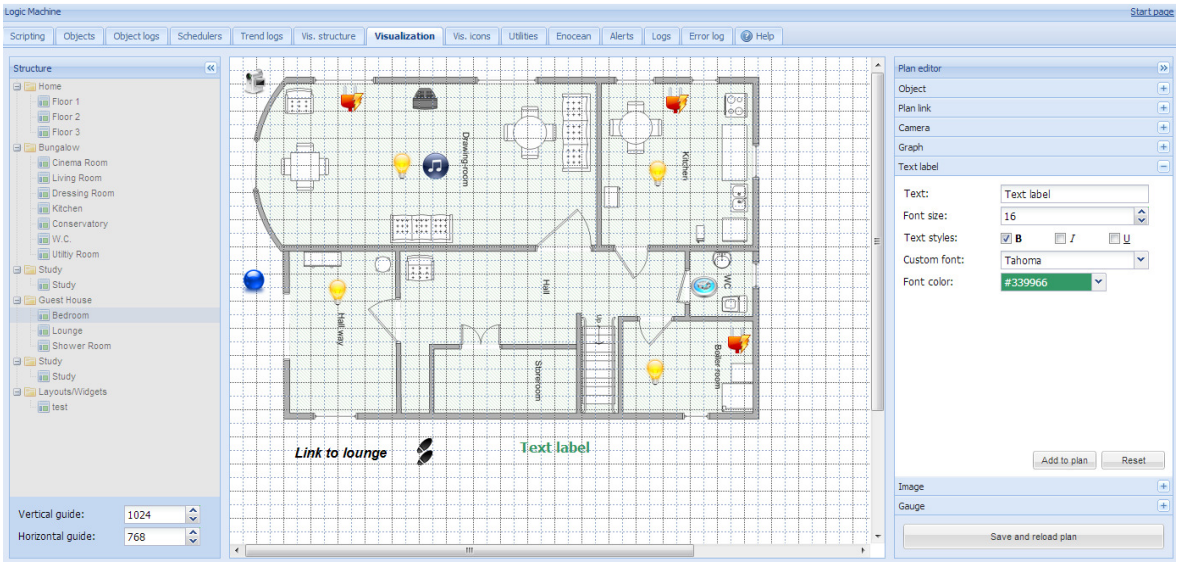
- ***Data object*** – group address of the object
- ***Custom name*** – name of the object
- ***Icon***– icon to launch the graph
- ***Width*** – sub-window width for displaying the graph
- ***Height***– sub-window height for displaying the graph
- ***Number of points*** – number of data points to show in the graph
- ***Auto open window*** – graph window is automatically opened
- ***Hide background*** – hide icon background

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.



1.8.6. Text Label

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

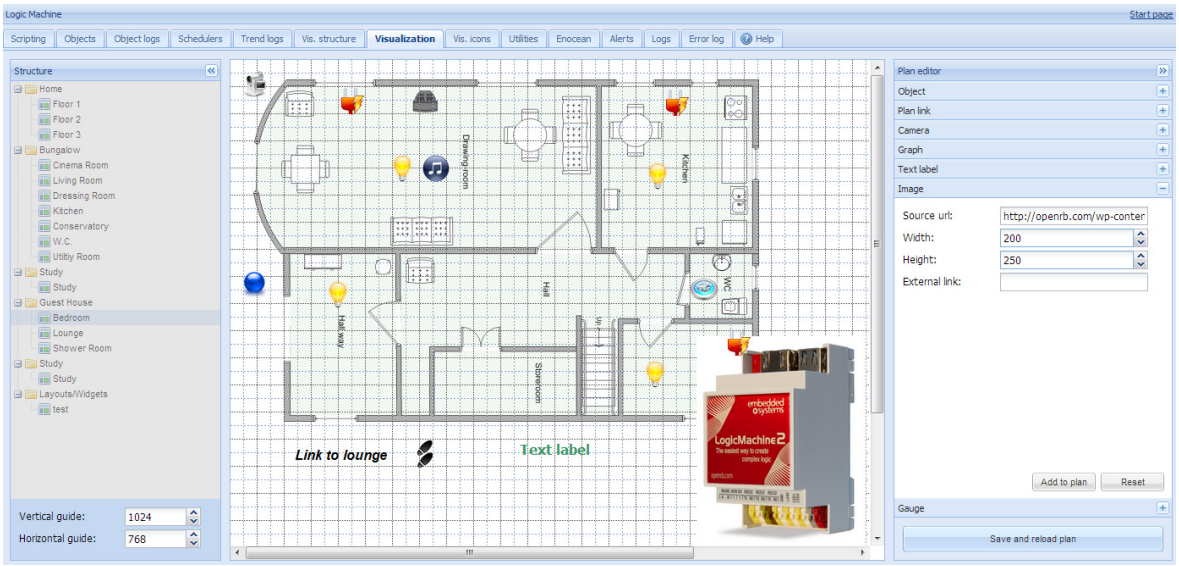


- ***Text*** – label text
- ***Font size*** – label font size
- ***Text style*** – style of the text – bold, italic, underscored
- ***Custom font*** – font name
- ***Font color***– label font color

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.

1.8.7. Image

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



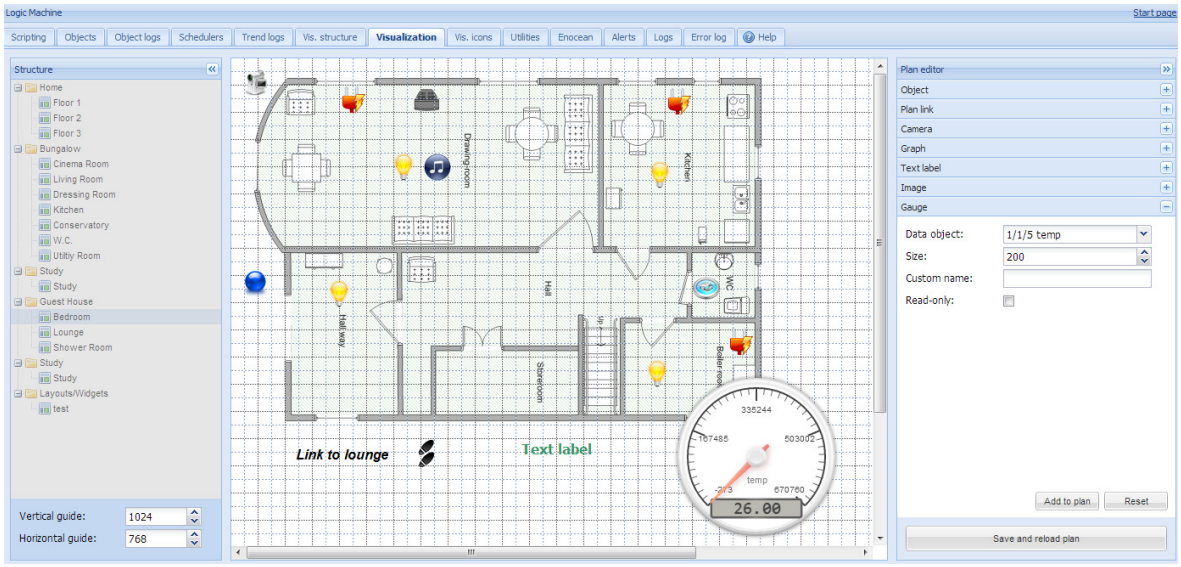
- ***Image source [Local; Remote]*** – image source location
- ***Source url / Select image*** – Source URL of the image or image from local database
- ***Width*** – width of the image
- ***Height*** – height of the image
- ***External link*** – external link URL when pressing on the image

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.

1.8.8. Gauge

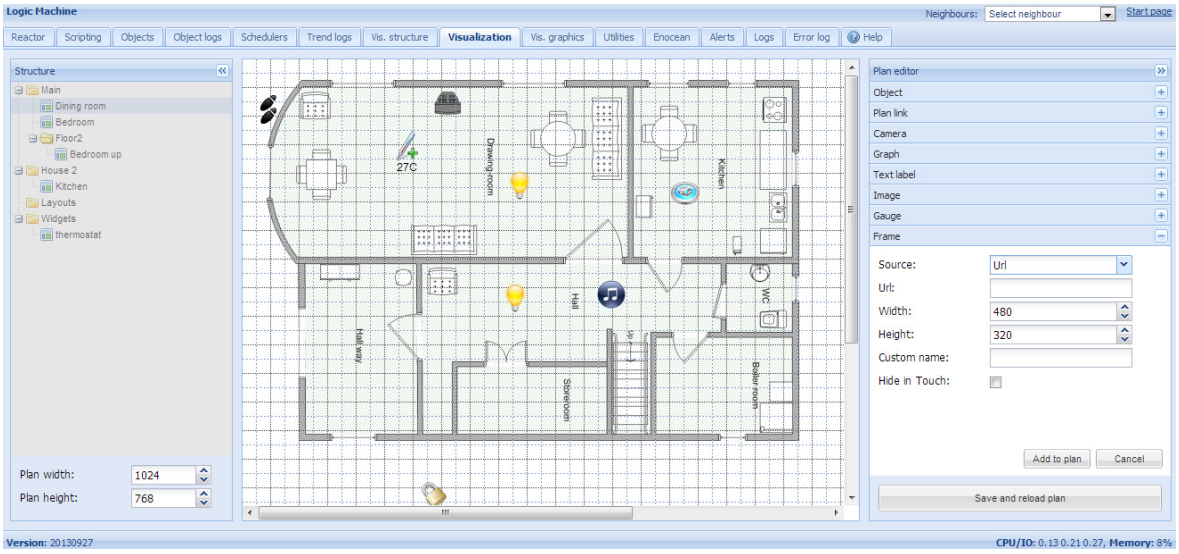
Gauge allows visualizing and changing object value in the gauge.

- ***Data object*** – KNX group address
- ***Size*** – size of the gauge
- ***Custom name*** – custom name for the object
- ***Read only*** – make the gauge read only

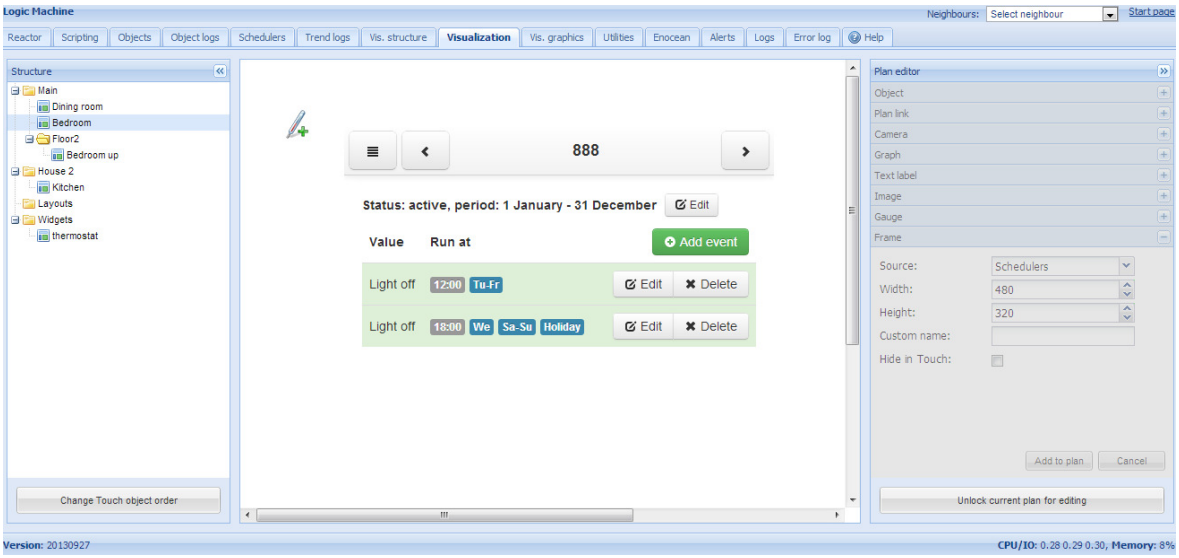


Once the gauge parameters are defined, press *Add to plan* button and newly created object will appear on the map. You can move the object to the location it will be located. Press on *Save and reload plan* button so the objects starts functioning.

1.8.9. Frame

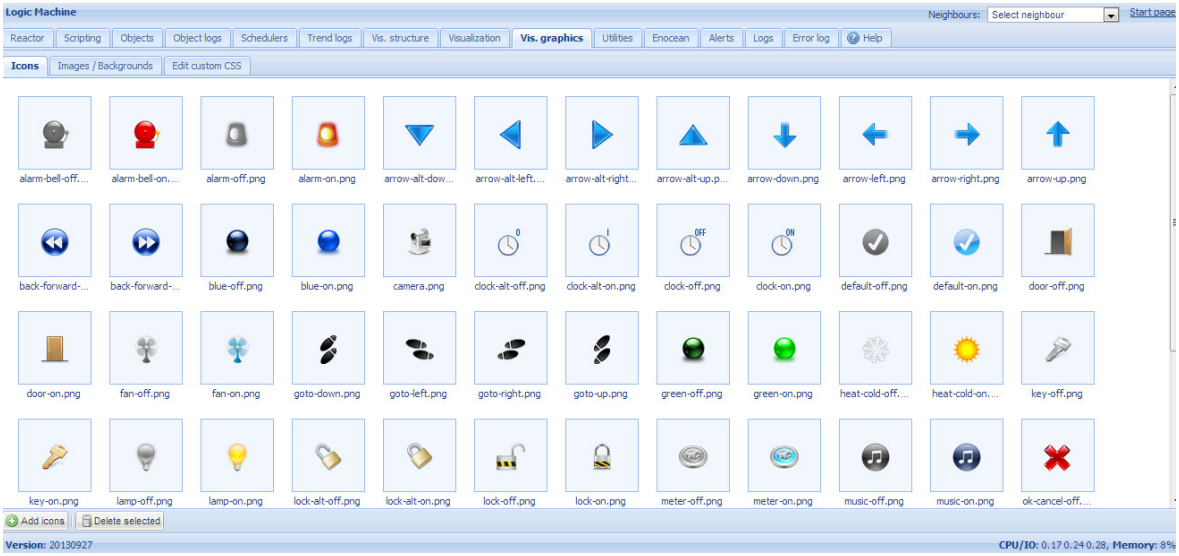


- ***Source [Url, Schedulers; Trend logs]*** – frame source
- ***Url*** – Source URL of the page to integrate
- ***Width*** – width of the frame
- ***Height*** – height of the frame
- ***Custom name*** – custom name of the frame object
- ***External link*** – external link URL when pressing on the image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

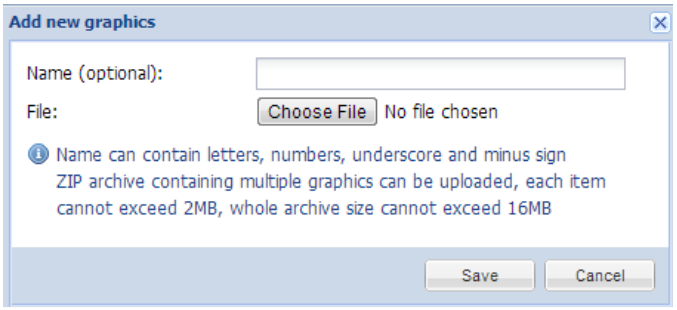


1.9. Vis. graphics

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

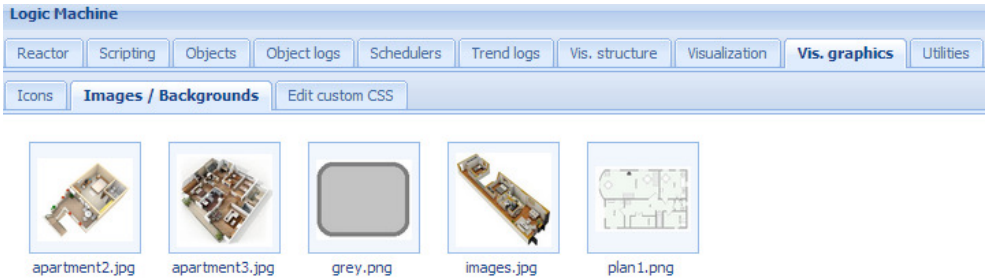


Press on *Add new icon* button to add a new entry. The system accepts any size icons. GIF is also supported.



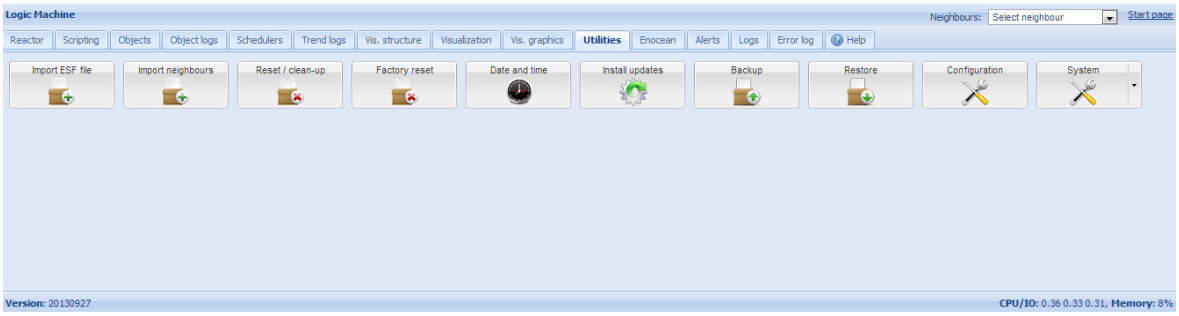
Name (optional) – the name of the icon
File – Icon file location

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

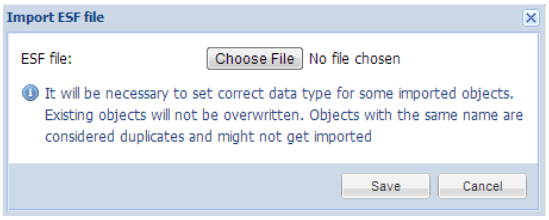


1.10. Utilities

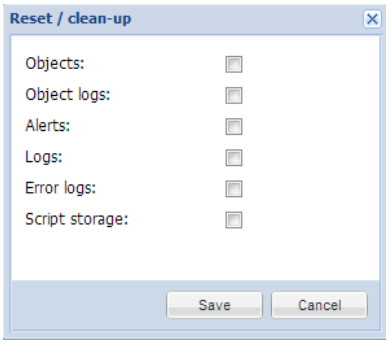
There are following utilities in the tab available:



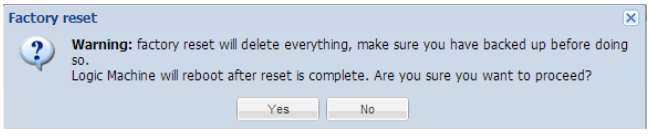
Import ESF file– imports ETS object file. It will be necessary to set correct data types for some imported objects. Existing objects will not be overwritten. Objects with the same name are considered duplicates and might not be imported



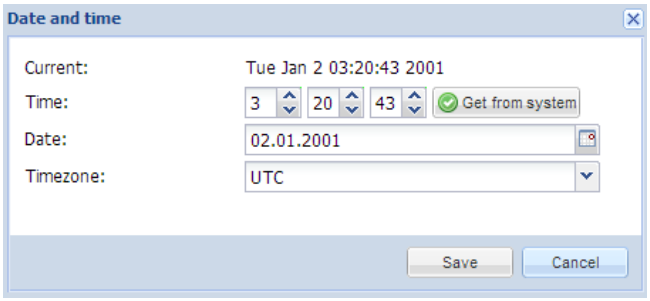
Reset object/clean-up – delete all objects from the Logic Machine, they disappear from visualization aswell



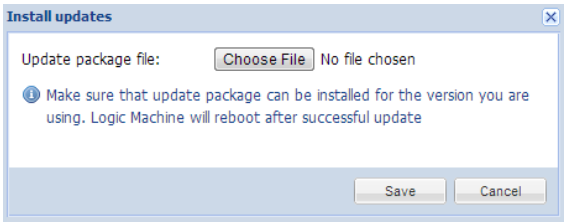
Factory reset– delete all configuration and return to factory defaults



Date and time – data and time settings

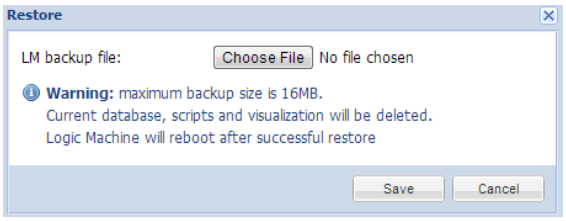


Install updates – install Logic Machine update file *.lmu. Logic Machine will reboot after successful update

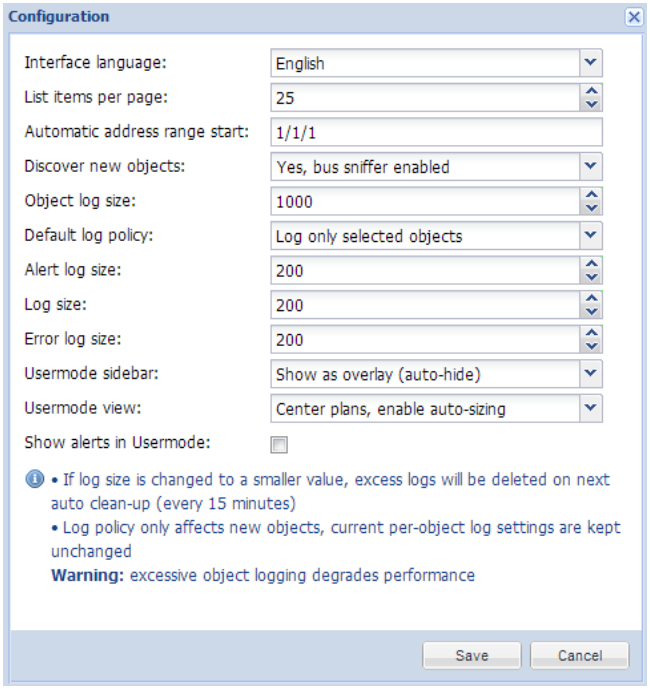


Backup – backup all objects, logs, scripts, visualization.

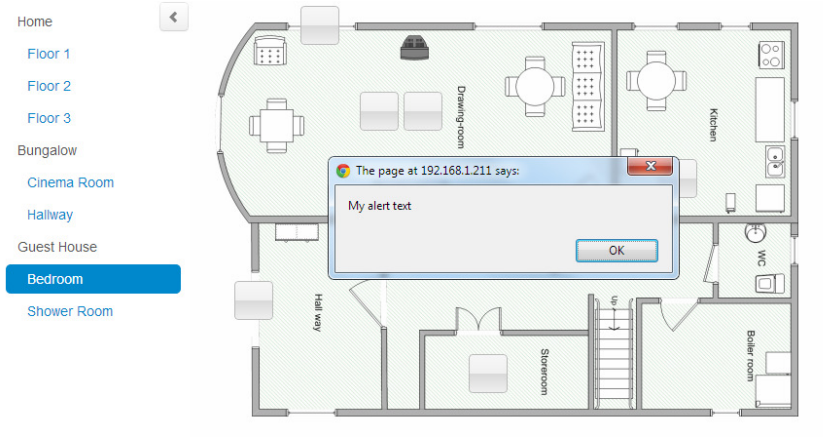
Restore– restore configuration from backup



Configuration – by clicking on the arrow, *KNX Connection* and *User Access* settings can be access. By clicking on the *Configuration button*, system general settings appear.



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
Usermode sidebar [Show docked; Show as overlay; Hide] – defines how the side bar is located in Usermode visualization
Usermode view [Align plans to top left, no size limit; Center plans, limit size; Center plans, enable auto-sizing] – defines the look of Usermode visualization
Show alerts in user mode – *once new Alerts is triggered it will pop-up in User mode visualization*



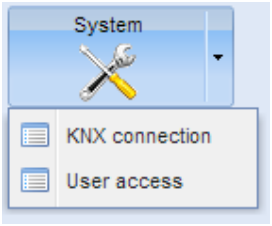
Note! Interface reload is required when changing "List items per page" or "Language" parameter

Note! If log size is changed to a smaller value, excess logs will be deleted on next auto clean-up (every 15 minutes)

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

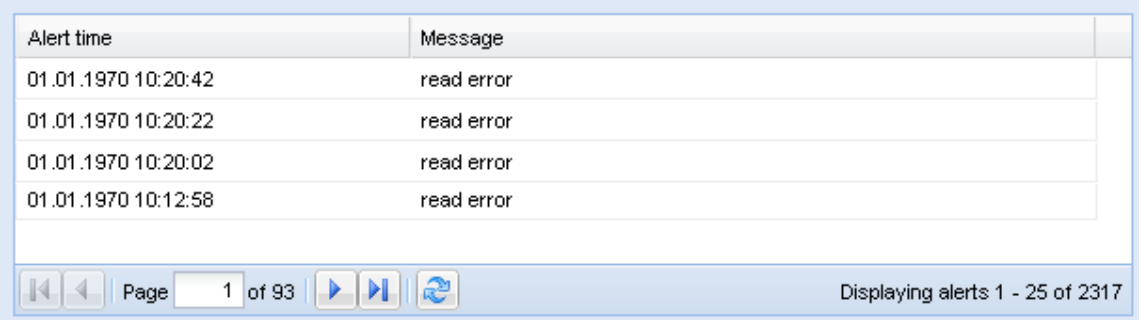
Warning! Excessive object logging degrades Logic Machine performance

System – by clicking on the arrow near System button, *KNX Connection* and *User Access* 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.



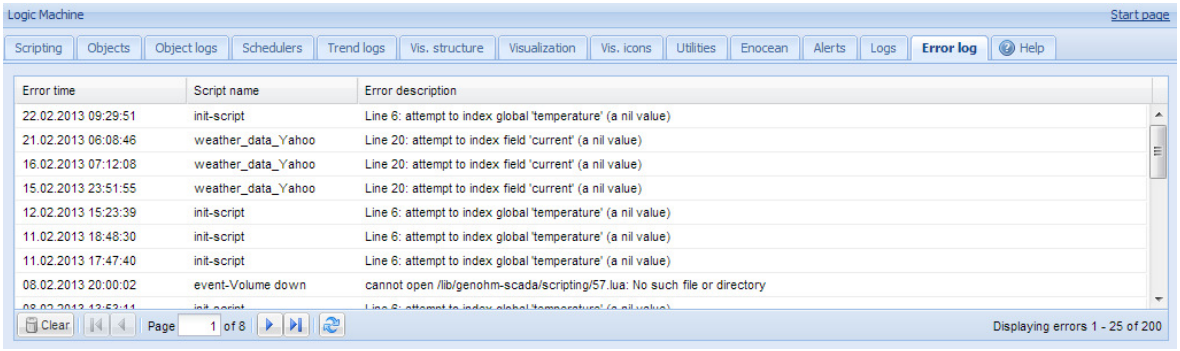
On the communication panel you can jump by pages and reload the page.

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
```

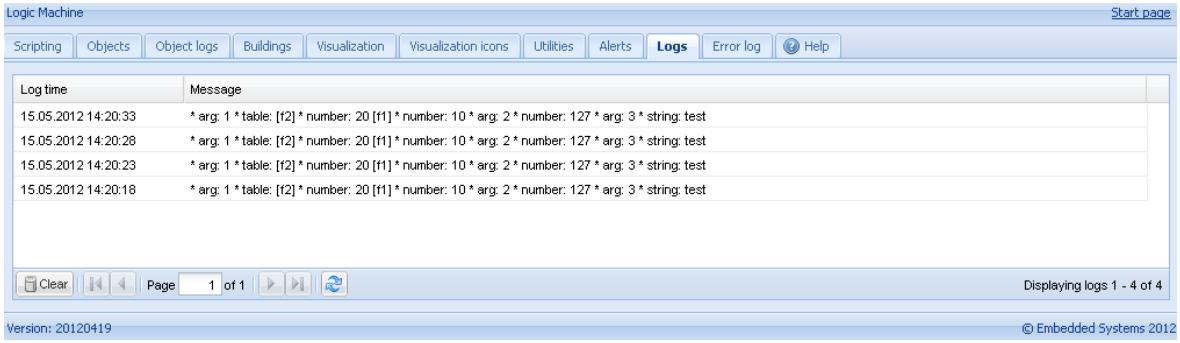
1.12. Error log

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



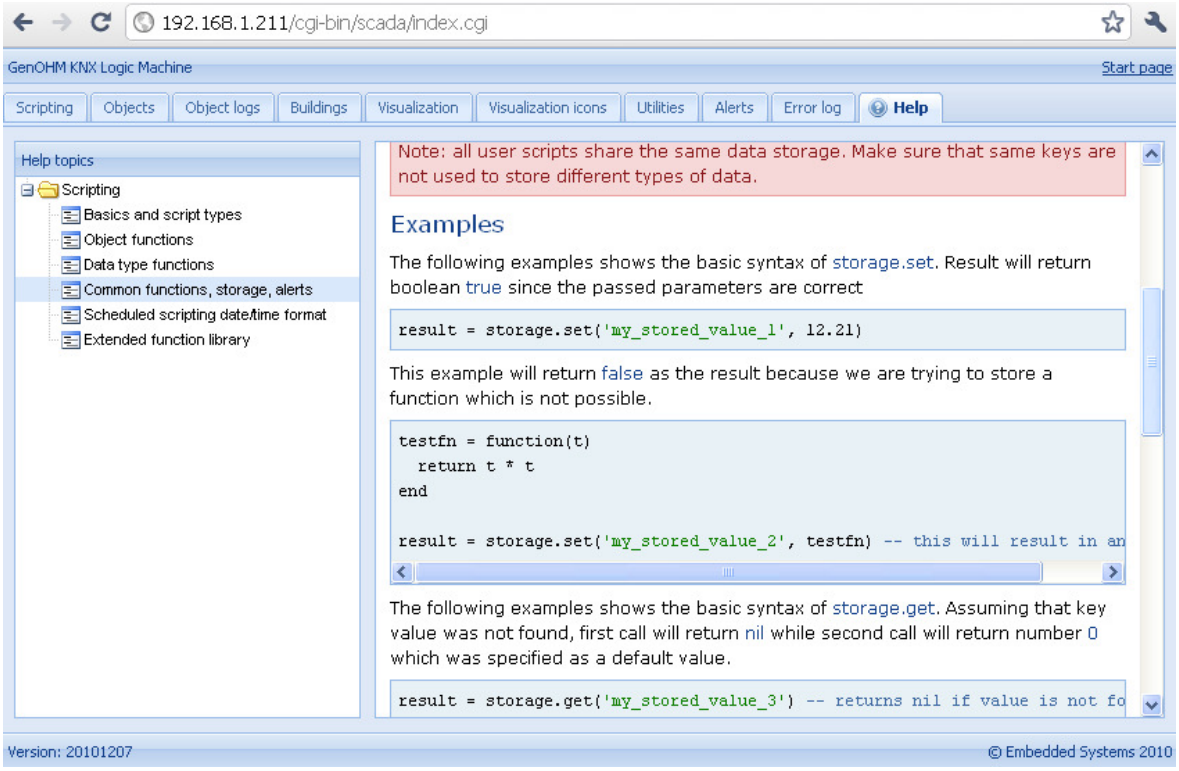
1.13. Logs

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


Logic Machine
Start page
Scripting
Objects
Object logs
Buildings
Visualization
Visualization icons
Utilities
Alerts
Logs
Error log
Help
Log time
Message
15.05.2012 14:20:33
* arg: 1 * table: [f2] * number: 20 [f1] * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
15.05.2012 14:20:28
* arg: 1 * table: [f2] * number: 20 [f1] * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
15.05.2012 14:20:23
* arg: 1 * table: [f2] * number: 20 [f1] * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
15.05.2012 14:20:18
* arg: 1 * table: [f2] * number: 20 [f1] * number: 10 * arg: 2 * number: 127 * arg: 3 * string: test
Clear
Page 1 of 1
Displaying logs 1 - 4 of 4
Version: 20120419
© Embedded Systems 2012

1.14. Help

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

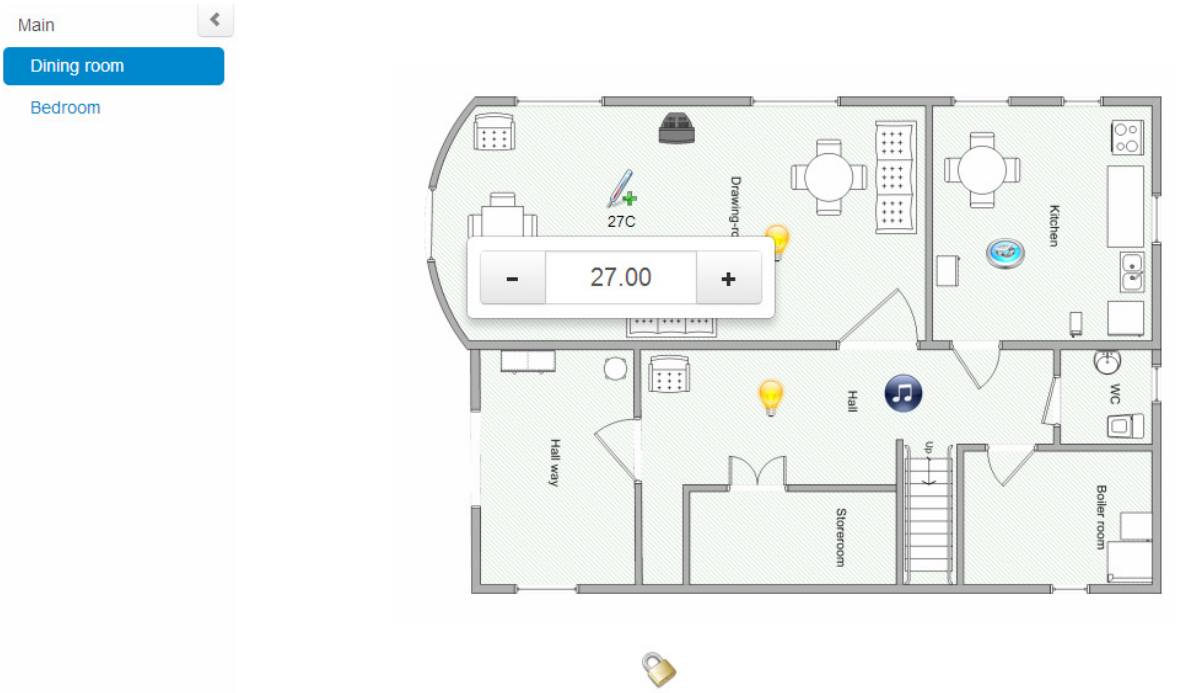


2. User mode visualization

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.

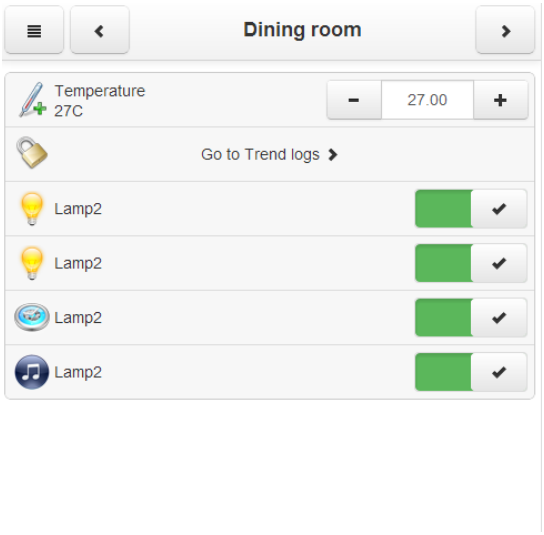
3. Touch visualization

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.

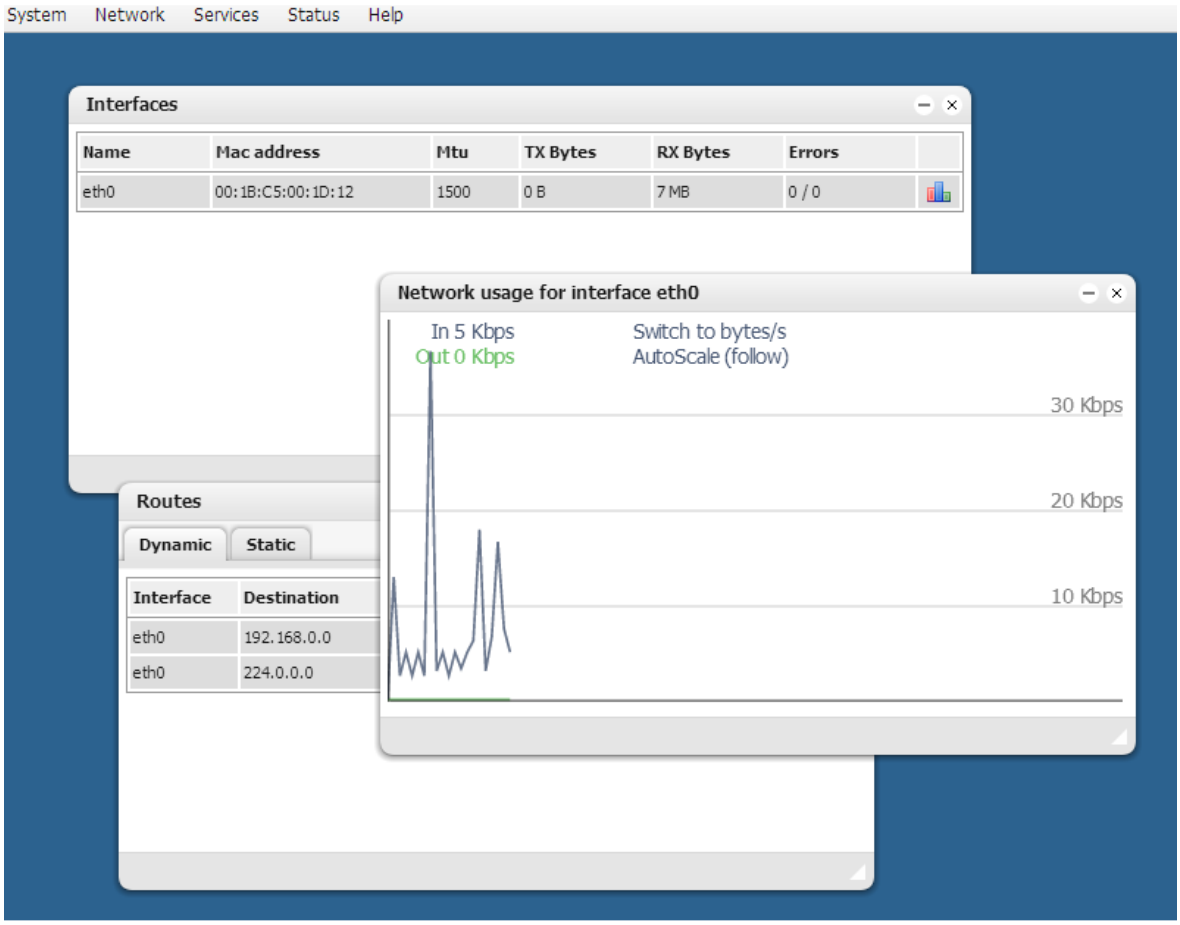


Launching visualization on touch device (iPad in this case)

- Make sure your iPad is connected wirelessly to the Logic Machine (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

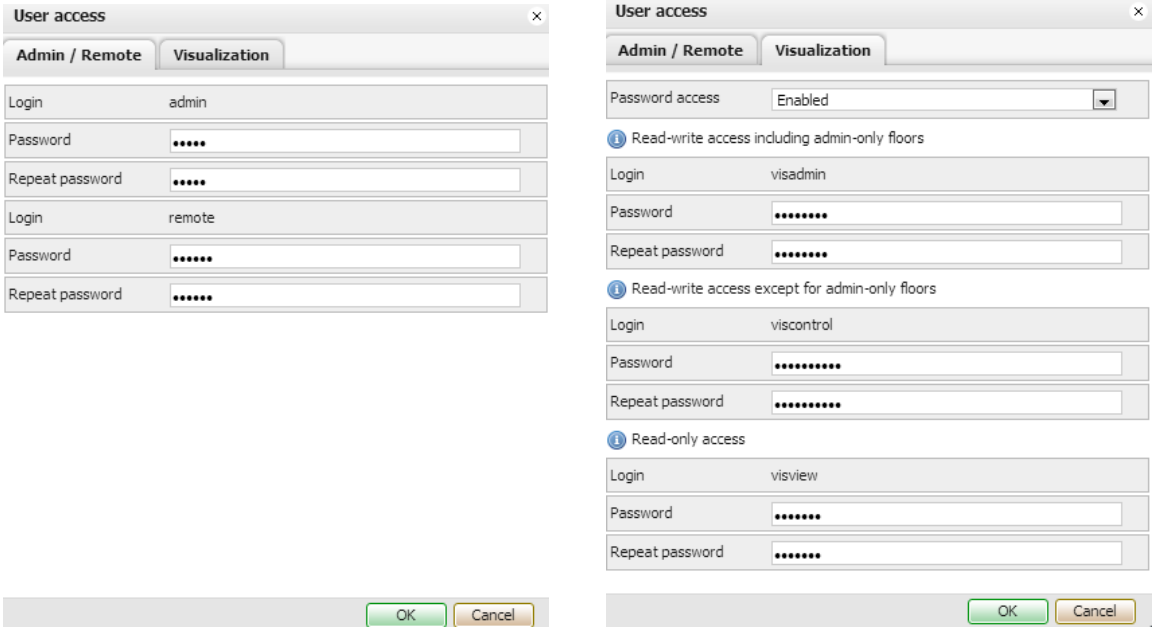
System configuration allows managing router functionality on KNX/EIB Logic Machine as well as do access control management, upgrade firmware, see network and system status and others.



Login	Password
admin	admin

4.1. Changing password

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

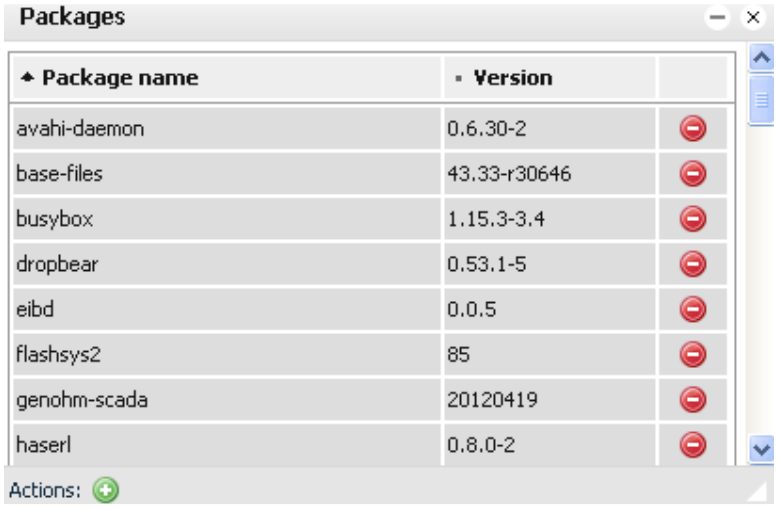


Access control is separated in 3 tabs:

Admin/Remote – access parameters for *Logic Machine, Network Configuration, RSS* and *XML*
Visualization – access parameters for *Touch* and *User mode visualization*

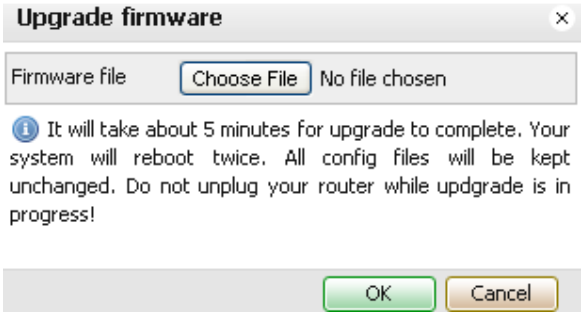
4.2. Packages

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



4.3. Upgrade firmware

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



4.4. Reboot Logic Machine

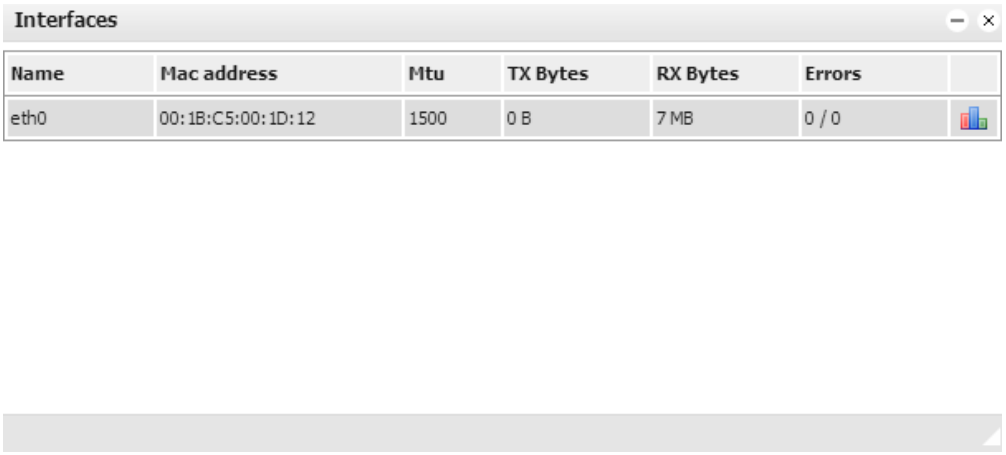
You can restart the Logic Machine by executing *System* → *Reboot* command.

4.5. Shutdown Logic Machine

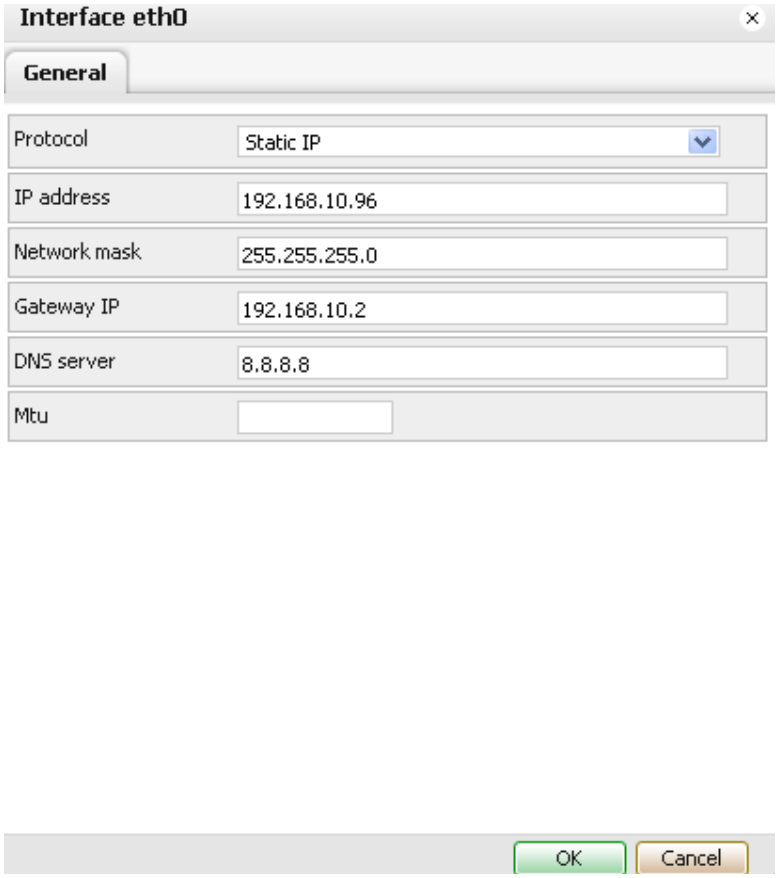
You can shutdown the Logic Machine by executing *System* → *Shutdown* command. It is advisable to shutdown the system before plug out the power, because the database is saved safely.

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.



By clicking on the interface you get to the configuration.



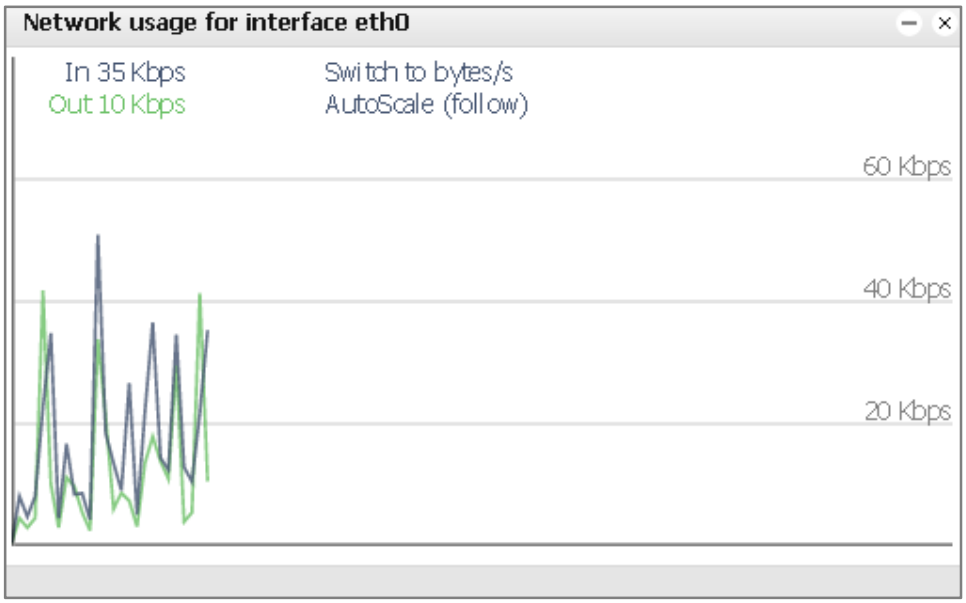
- ***Protocol***– specific protocol used for addressing
  Static IP – static IP address. By default 192.168.0.10
  DHCP – use DHCP protocol to get IP configuration.

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

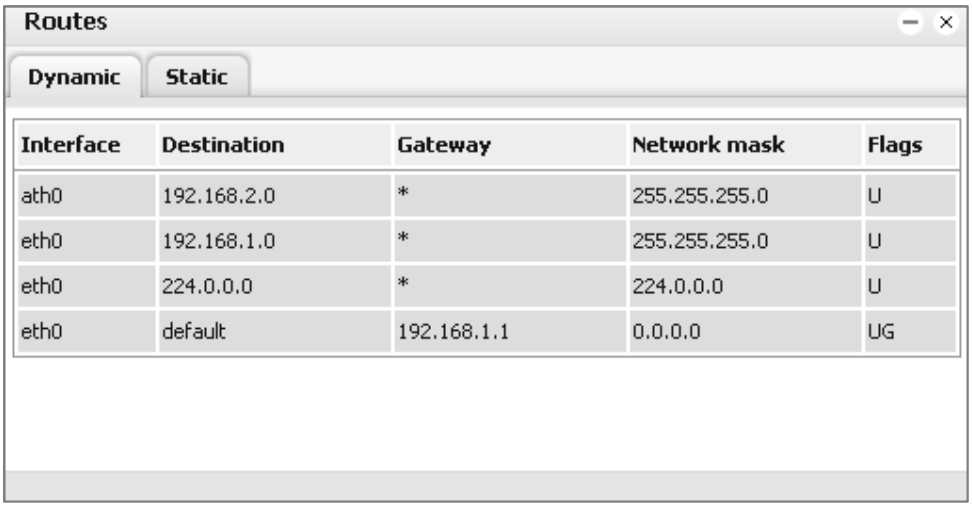
On the main window of the Ethernets tab, if you click on the  button, a new window is opened. It draws a real-time graph of the traffic flow passing the interface (both In and Out). There is a possibility to switch the units of measurement – bytes/s or bytes/s.



4.7. Routing Table

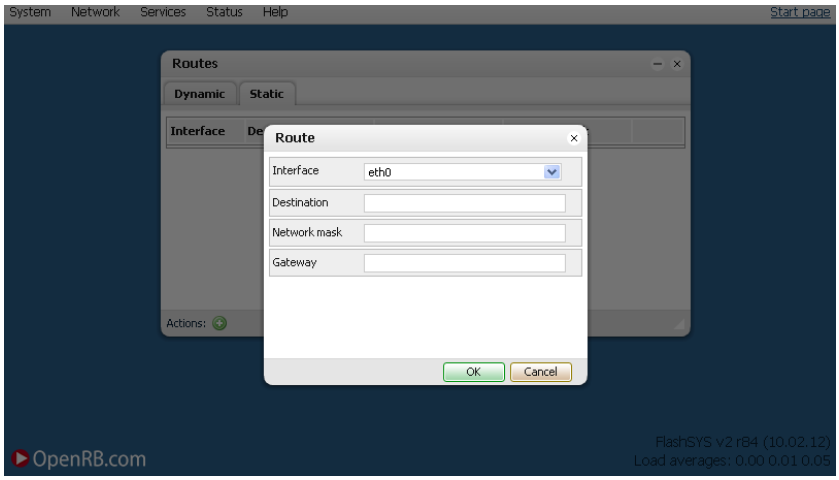
System routing table is located in *Network→Routes* menu. The window is divided in two parts – Static routes and Dynamic routes.

4.7.1. Dynamic routes



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

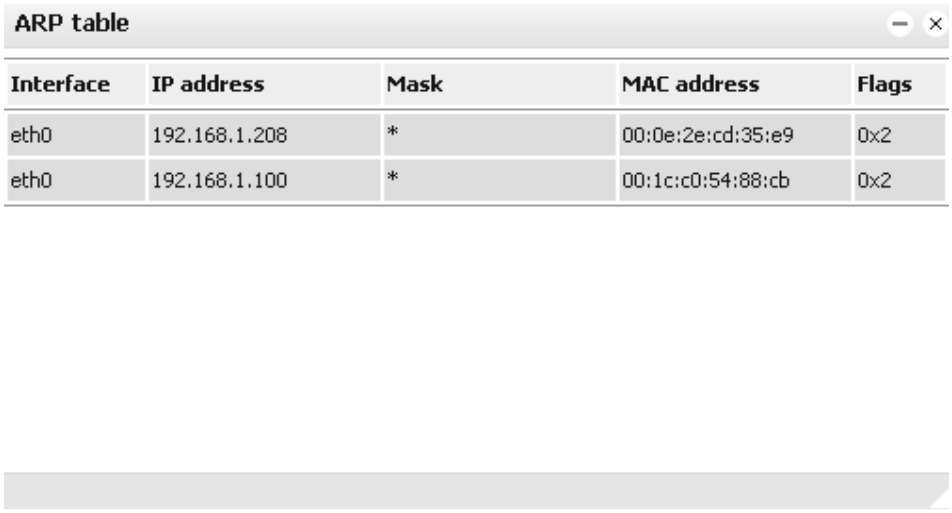
4.7.2. Static routes



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

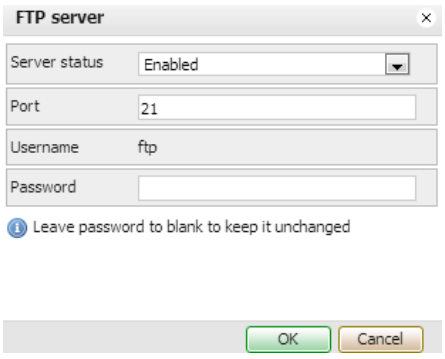
4.8. ARP table

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



4.9. FTP server

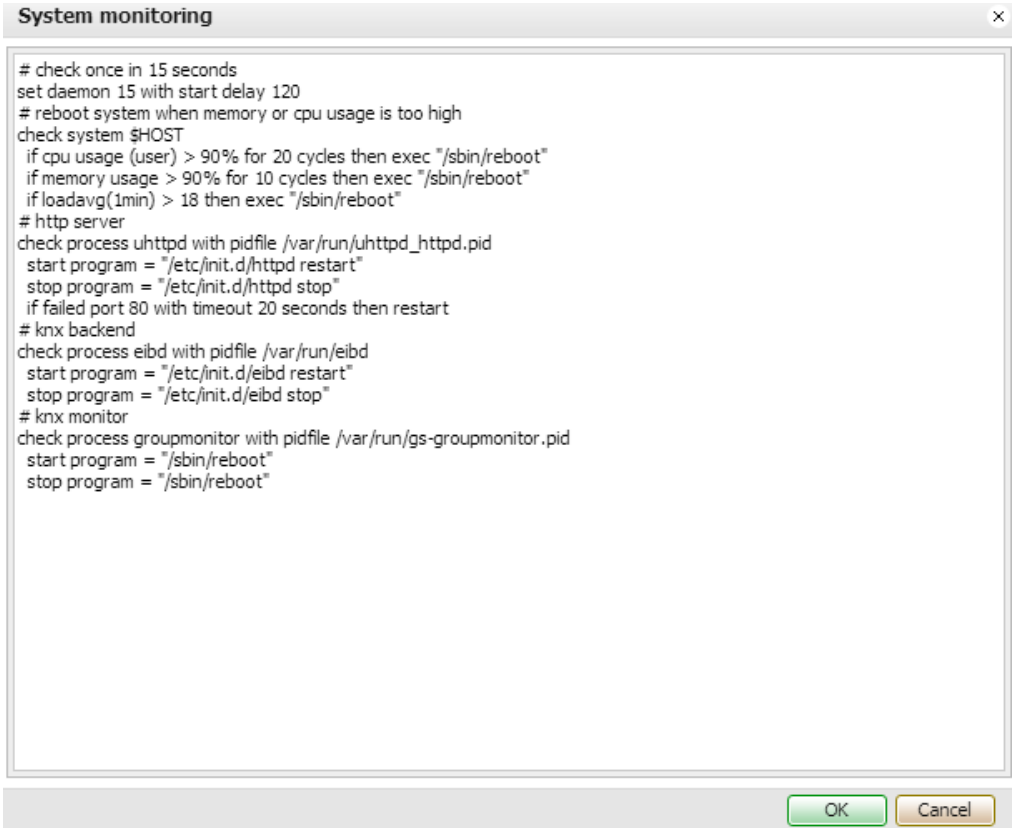
You can enable access to FTP server of Logic Machine by enabling this service in *Service → FTP Server*.



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

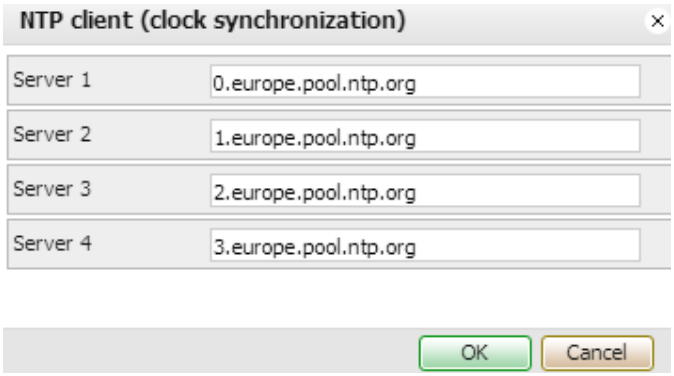
4.10. System monitoring

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



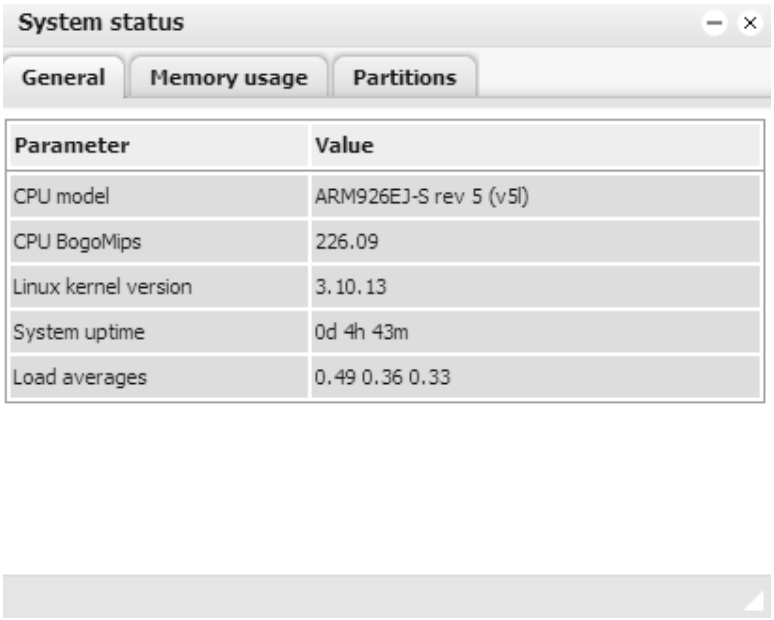
4.11. NTP client

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



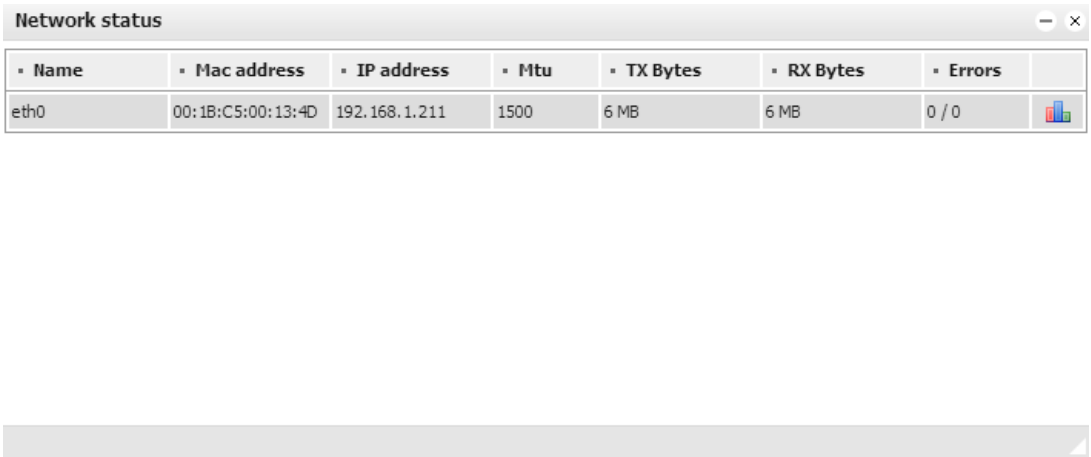
4.12. System status

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



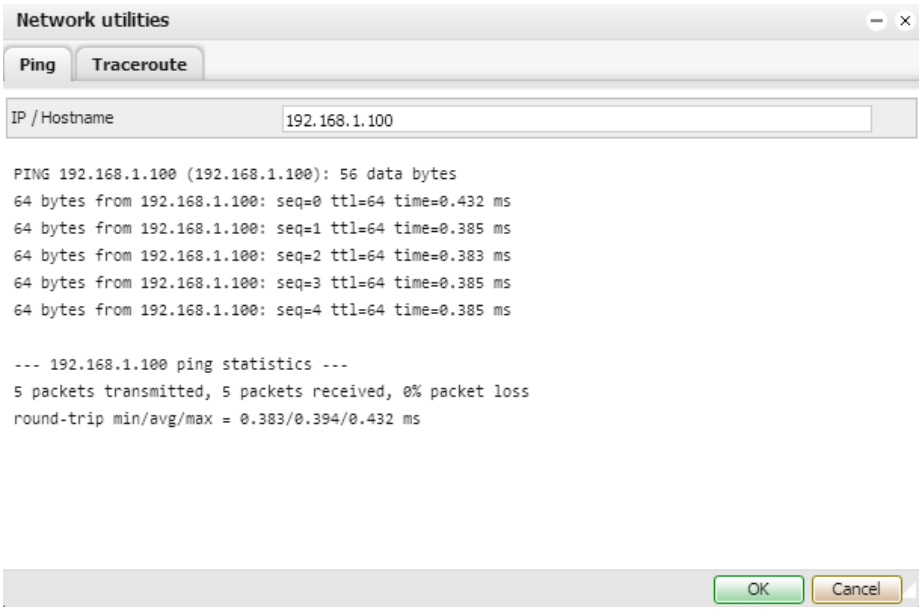
4.13. Network status

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



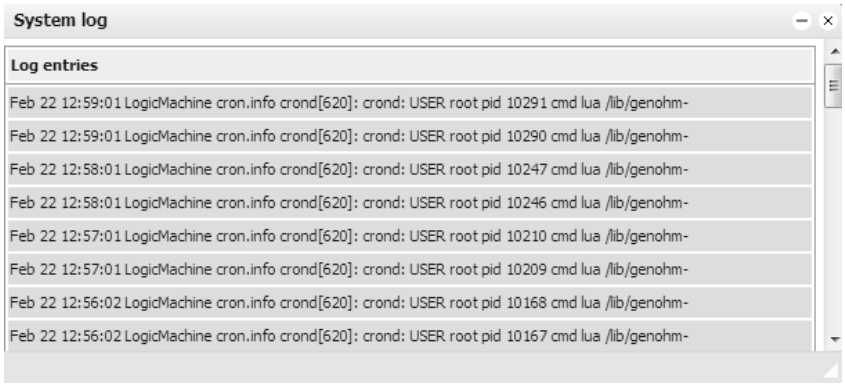
4.14. Network utilities

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



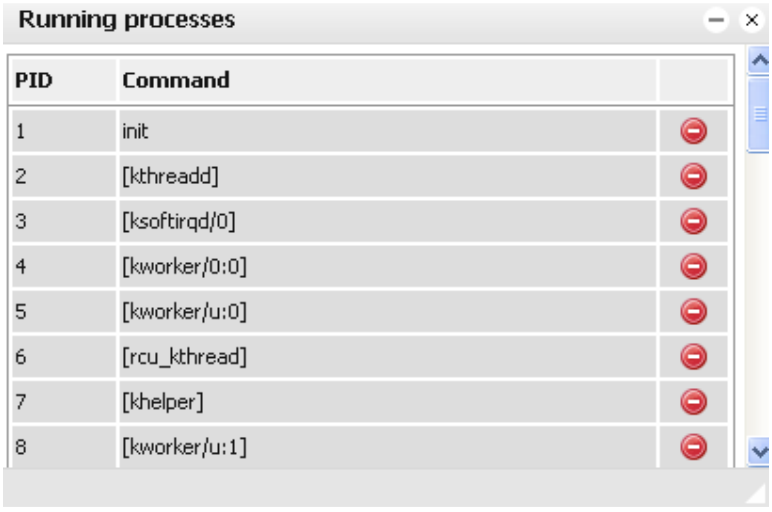
4.15. System log

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



4.16. Running processes

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

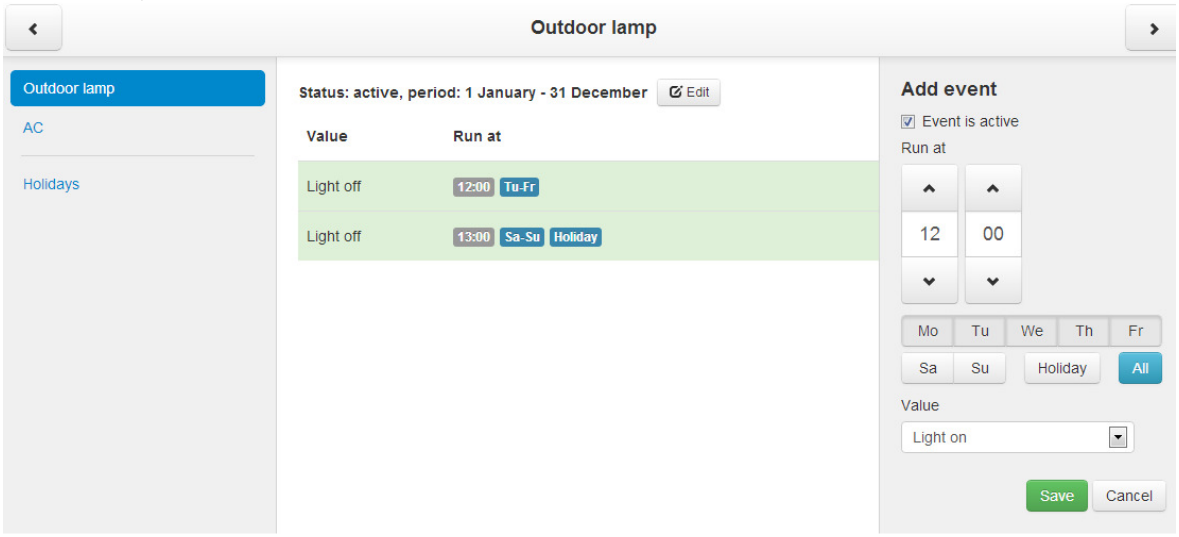


5. User mode schedulers

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

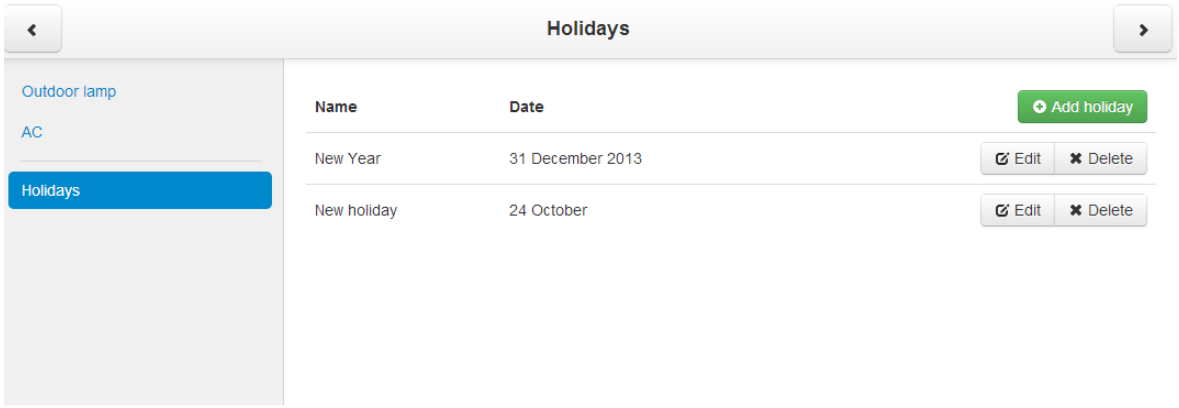
Each scheduler is mapped to specific group address in administration panel (*see section 1.4 of this manual*).



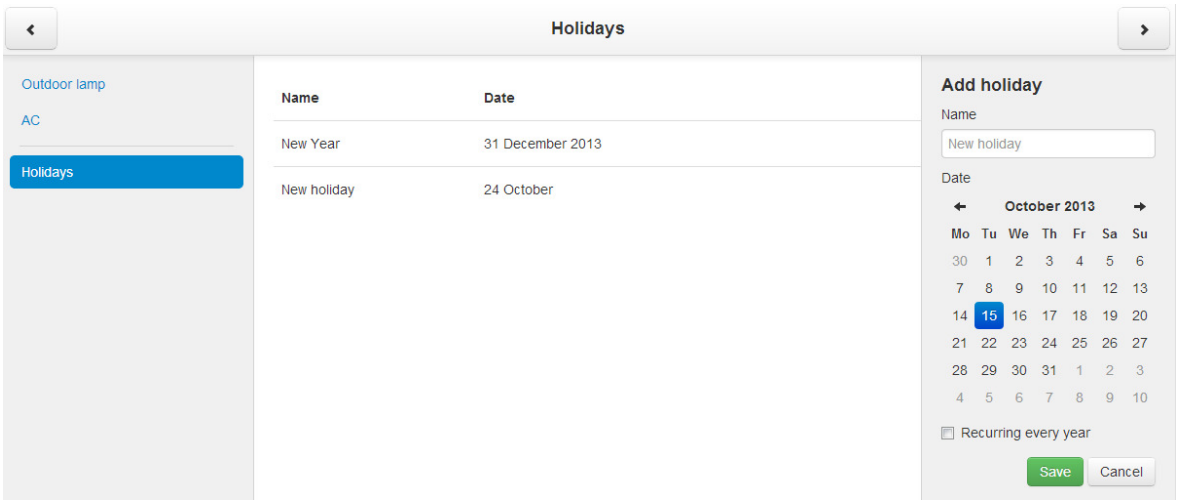
When adding the new task for specific scheduler you can specify day of the week, start time, value to send to the object.

5.2. Holidays

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

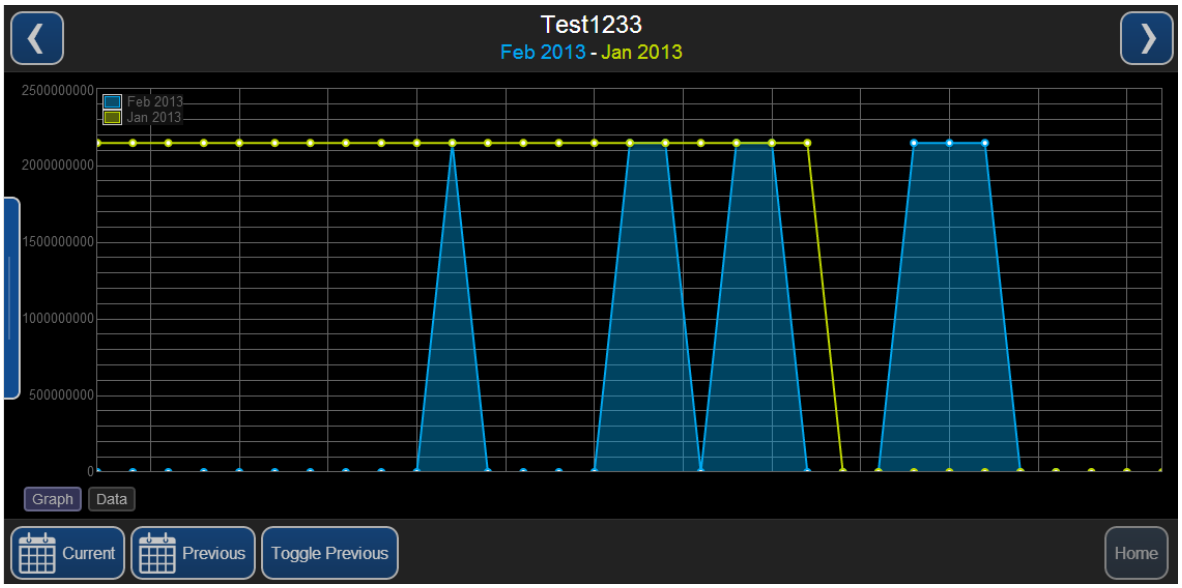


Click on *Add new holiday* button to specify a holiday.

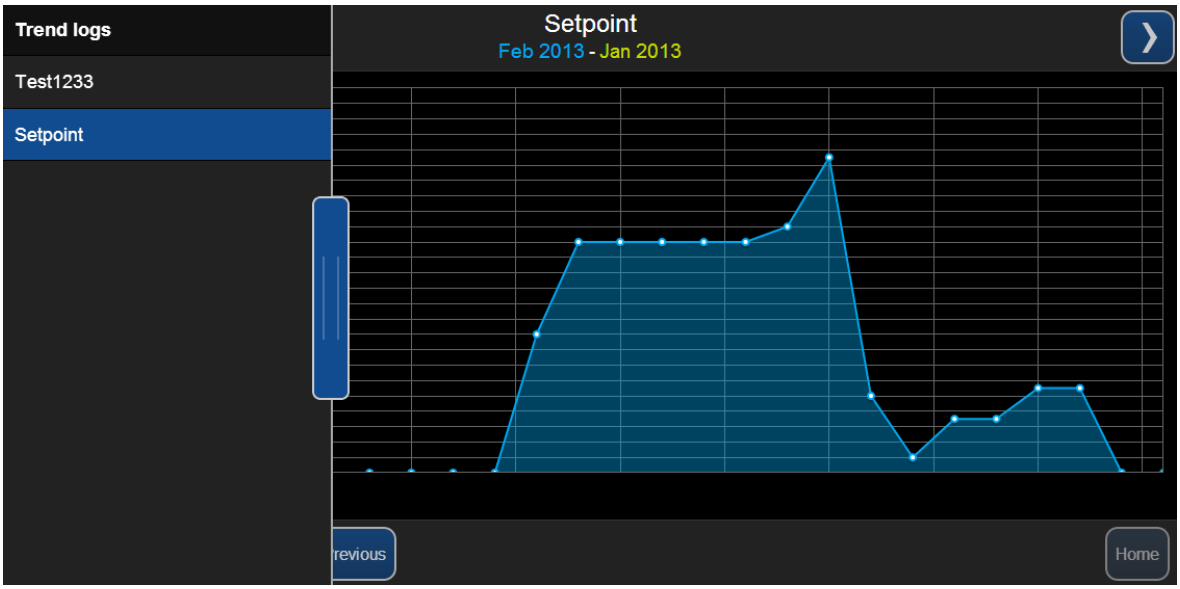


6. Trend logs

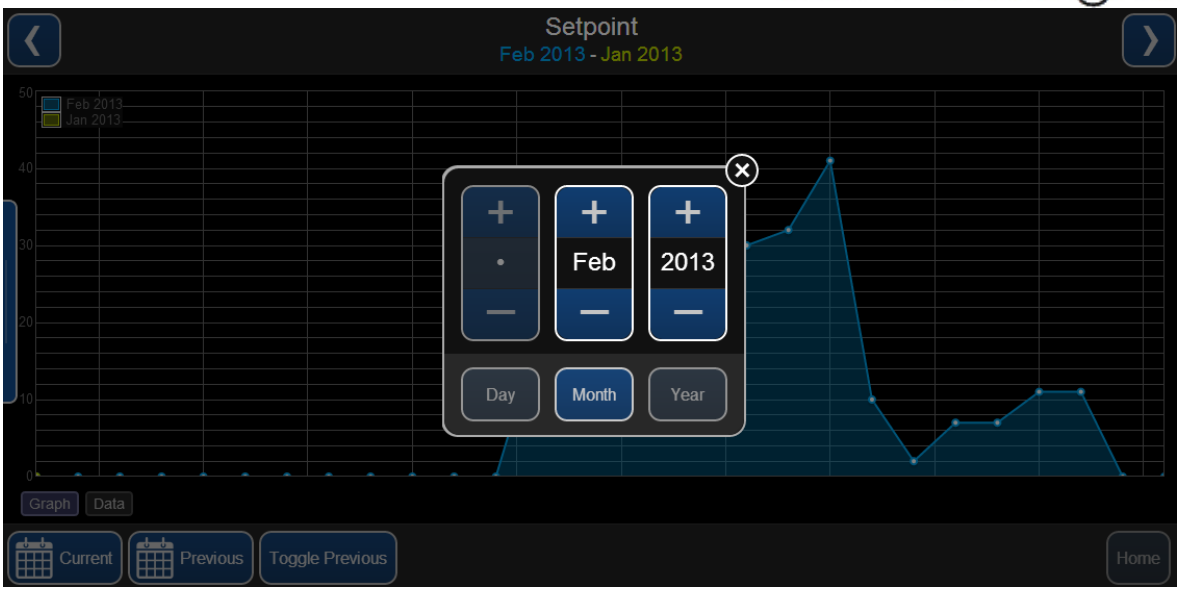
Trend logs are end user interface for trends (*defined in administrator interface in section 1.5*).



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 – Logic Machine home screen.

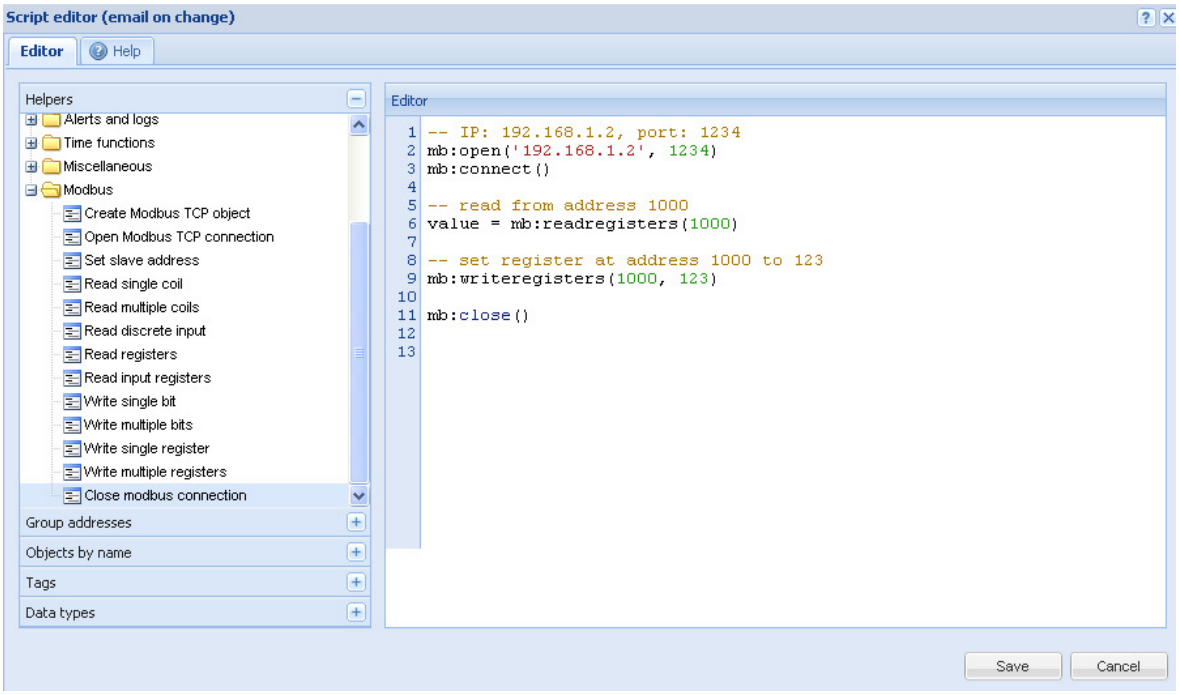


Datapoints can be shown also in a way of table which can be later exported as CSV file.

7. Modbus RTU/TCP interconnection with LM2

Modbus RTU is supported over RS485 interface. Modbus TCP is supported over Ethernet port. Modbus communication is done directly from scripts (usually resident script is used to read Modbus value after some specific time interval and write them into KNX object).

Once script is added, you can add the code in the Script Editor. There are lots of predefined code blocks in the Helpers.



7.1. Master functions

mb:setslave(slaveid)
Sets slave id to read/write data from/to

mb:readcoils(start, count)
mb:readdiscreteinputs(start, count)
mb:readregisters(start, count)
mb:readinputregisters(start, count)
Reads count registers/coils from the start address. Returns all values on success and nil, error description on error

mb:writebits(start, v1, [v2, [v3, ...]])
mb:writeregisters(start, v1, [v2, [v3, ...]])
Writes values to registers/coils from the start address. Single write will be used when only one value is supplied, multiple write otherwise. Returns all of values written on success and nil, error

description on error

mb:reportslaveid()
Reads slave internal data. Returns values on success and nil, error description on error.

7.2. Visualizing Modbus objects

Use *grp.write* to assign Modbus object to KNX object and then use this new KNX object in the visualization.

7.3. Usage example (Modbus TCP)

Task: read three registers from Modbus TCP device and write the result in Alerts.

```
1. -- initmodbus on first script execution
2. ifnotmbthen
3. require('luamodbus')
4. mb=luamodbus.tcp()
5. end
6.
7. -- prepare connection to given ip and port
8. mb:open('192.168.1.100', 1502)
9.
10. -- open connection and check the result
11. ifmb:connect()then
12. -- read 3 input registers, function returns 3 variables
13. local x, y, z =mb:readinputregisters(1, 3)
14.
15. -- first variable will be nill if read failed
16. if x then
17. local message =string.format('1: %d; 2: %d; 3: %d', x, y, z)
18.       alert(message)
19. end
20.
21. -- end session
22. mb:close()
23. else
24.     alert('connection failed')
25. end
```

7.4. Usage example (Modbus RTU)

Task: read two parameters (3-phase system voltage, 3-phase system current) from Modbus Multimeter 32-bit registers and store the data in the KNX group addresses. Make sure to connect LM2 with Modbus device correctly, RS485 A with -, RS485 B with +.

```
1. -- initmodbus on first script execution
2. ifnotmbthen
3. require('luamodbus')
4. mb=luamodbus.rtu()
5. mb:open('/dev/ttyS2', 9600, 'E', 8, 1, 'H')
6. mb:connect()
7. end
8.
9. -- sets slave ID to read/write data from/to
10. mb:setslave(20)
11.
12. -- read 3-phase system voltage from 32-bit register
13. r1, r2 =mb:readregisters(0x1000, 2)
14. result =bit.lshift(r1, 16) + r2
15. grp.write('5/5/1',result)
16.
17. -- read 3-phase system current from 32-bit register
18. r1, r2 =mb:readregisters(0x100E, 2)
19. result =bit.lshift(r1, 16) + r2
20. grp.write('5/5/2',result)
```

Some Modbus devices keep enocded values in registers, you need to encode them first from HEX to use in the further scripts.For example, **value = 0x0cba** after executing the below commands will give temperature equal to **24.2**

```
1. hex =lmcore.inttohex(value, 2)
2. temp =knxdatatype.decode(hex, dt.float16)
```

Here is an example of function which is doing **byte shift**:

```
1. -- get single bit from a numeric value
2. function getbit(value, bnum)
3.    value = tonumber(value) or 0
4.    value = bit.rshift(value, bnum)
5.    return bit.band(value, 1)
6. end
7.
8.
9. getbit(value, 0) -- first bit, and so on
```

7.5. Modbus Slave examples

Add the following code to *Common functions*

```
-- modbus proxy
mbproxy = {
  -- supported function list
  functions = {
    'readdo',
    'readcoils',
    'readdi',
    'readdiscreteinputs',
    'readao',
    'readregisters',
    'readai',
    'readinputregisters',
    'writebits',
    'writemultiplebits',
    'writeregisters',
    'writemultipleregisters',
    'reportslaveid',
    'getcoils',
    'getdiscreteinputs',
    'getinputregisters',
    'getregisters',
    'setcoils',
    'setdiscreteinputs',
    'setinputregisters',
    'setregisters',
  },
  -- new connecton init
  new = function()
    require('rpc')
    local mb = setmetatable({}, { __index = mbproxy })

    mb.slaveid = 0
    mb.rpc = rpc.client('127.0.0.1', 28002, 'mbproxy')

    for _, fn in ipairs(mbproxy.functions) do
      mb[ fn ] = function(self, ...)
        return mb:request(fn, ...)
      end
    end

    return mb
  end
}

-- set local slave id
```

```
46. function mbproxy:setslave(slaveid)
47.   self.slaveid = slaveid
48. end
49.
50. -- send rpc request for a spefic function
51. function mbproxy:request(fn, ...)
52.   local res, err = self.rpc:request({
53.     fn = fn,
54.     params = { ... },
55.     slaveid = self.slaveid 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/ttyS2', 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.  if not 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. if not 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.     if not mb[ fn ] then
45.       return { nil, 'unknown function ' .. fn }
```

```
46.           end
47.
48.           if type(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)
```

7.6. Modbus working with several slaves on the same RS485 connection

The example was designed to interconnect with 16 VRF system in one line through 1 Logic Machine2.

Resident script

```
-- modbus init
if not mb then
  require('luamodbus')

  mb = luamodbus.rtu()

  mb:open('/dev/ttyS2', 9600, 'E', 8, 1, 'H')
  mb:connect()

  mb:setslave(1)

  -- a/c list
  aclist = {
```

```
    -- a/c: 0, id: 1
    { addrstat = '8/4/0', addrmode = '8/5/0', addrspeed = '8/6/0', addrtemp = '8/7/0' },
    -- a/c: 1, id: 2
    { addrstat = '8/4/1', addrmode = '8/5/1', addrspeed = '8/6/1', addrtemp = '8/7/1' },
  }

  -- read 8 bits and convert to single byte
  function readbyte(offset)
    local bits = mb:readdiscreteinputs(offset, 8)
    local result = 0

    for i = 1, 8 do
      if bits[ i ] then
        result = result + bit.lshift(1, i - 1)
      end
    end

    return result
  end

  -- write single byte and convert to 8 bits
  function writebyte(offset, byte)
    local bits = {}

    for i = 1, 8 do
      table.insert(bits, bit.band(1, bit.rshift(byte, i - 1)) == 1)
    end

    mb:writebits(offset, unpack(bits))
  end
end

-- local udp server init
if not server then
  require('socket')

  server = socket.udp()
  server:setsockname('127.0.0.1', 28016)
  server:settimeout(1)

  -- remote command handler
  function cmd(data)
    local id, cmd, value, ac, addr, offset

    -- command format id:cmd[:value]
    id, cmd, value = unpack(data:split(':'))
    id = tonumber(id) or 0

    -- check if ac is valid
    ac = aclist[ id ]
    if not ac then
      return
    end

    -- default offset
    offset = (id - 1) * 152

    -- on/off
    if cmd == 'ON' or cmd == 'OFF' then
      mb:writebits(offset, cmd == 'ON')
    -- temperature settings
    elseif cmd == 'TEMP' then
      value = tonumber(value)

      -- value ok
      if value then
        -- calculate register offset and write encoded value
        offset = (id - 1) * 156
        mb:writeregisters(offset, encodetemp(value))
      end
    -- operation mode
    elseif cmd == 'MODE' then
      value = tonumber(value)

      -- verify bounds
      if 0 <= value and value <= 3 then
        -- convert to a/c value and write
        writebyte(offset + 8, value + 1)
      end
    -- fan speed
    elseif cmd == 'SPEED' then
      value = tonumber(value)

      -- verify bounds
      if 0 <= value and value <= 3 then
        -- convert to a/c value and write
        writebyte(offset + 16, value + 2)
      end
    end
  end
end

-- read current status for each a/c unit
for id, ac in ipairs(aclist) do
  local stat, mode, temp, speed, offset

  -- address offset
  offset = (id - 1) * 152

  -- on/off status
  stat = mb:readdiscreteinputs(offset)
  if type(stat) == 'boolean' and ac.stat ~= stat then
```

```
    ac.stat = stat
    grp.write(ac.addrstat, stat, dt.bool)
  end

  -- operation mode
  mode = readbyte(offset + 8)
  if type(mode) == 'number' and ac.mode ~= mode then
    ac.mode = mode
    -- send proper value to knx
    if 1 <= mode && mode <= 5 then
      grp.write(ac.addrmode, mode - 1, dt.uint8)
    end
  end

  -- fan speed
  speed = readbyte(offset + 16)
  if type(speed) == 'number' and ac.speed ~= speed then
    ac.speed = speed
    -- send proper value to knx
    if 2 <= mode && mode <= 7 then
      grp.write(ac.addrspeed, speed - 2, dt.uint8)
    end
  end

  -- temperature
  offset = (id - 1) * 156 + 1
  temp = readinputregisters(offset)
  if type(temp) == 'number' and ac.temp ~= temp then
    ac.temp = temp
    grp.write(ac.addrtemp, decodetemp(temp), dt.float16)
  end
end

-- read command from client
local data = server:receive()
if data then
  cmd(data)
end
```

Common function program

```
function decodetemp(value)
  local hex = lmcore.inttohex(value, 2)
  return knxdatatype.decode(hex, dt.float16)
end

function encodetemp(value)
  local hex = knxdatatype.encode(value, dt.float16).datahex
  return tonumber(hex, 16)
end

-- send request to modbus resident
function accmd(id, cmd, value)
  local request, client

  require('socket')

  -- check if value has been passed
  value = value and tonumber(value)

  -- create request string
  request = string.format('%d:%s', id, cmd:upper())
  if value then
    request = string.format('%s:%s', request, value)
  end

  -- send udp packet
  socket.udp():sendto(request, '127.0.0.1', 28016)
end
```

Example: on/off VRF system

```
value = knxdatatype.decode(event.datahex, dt.bool)
accmd(1, value and 'ON' or 'OFF')
```

Example: set mode of VRF system

```
value = knxdatatype.decode(event.datahex, dt.uint8)
accmd(1, 'MODE', value)
```

Example: set the speed

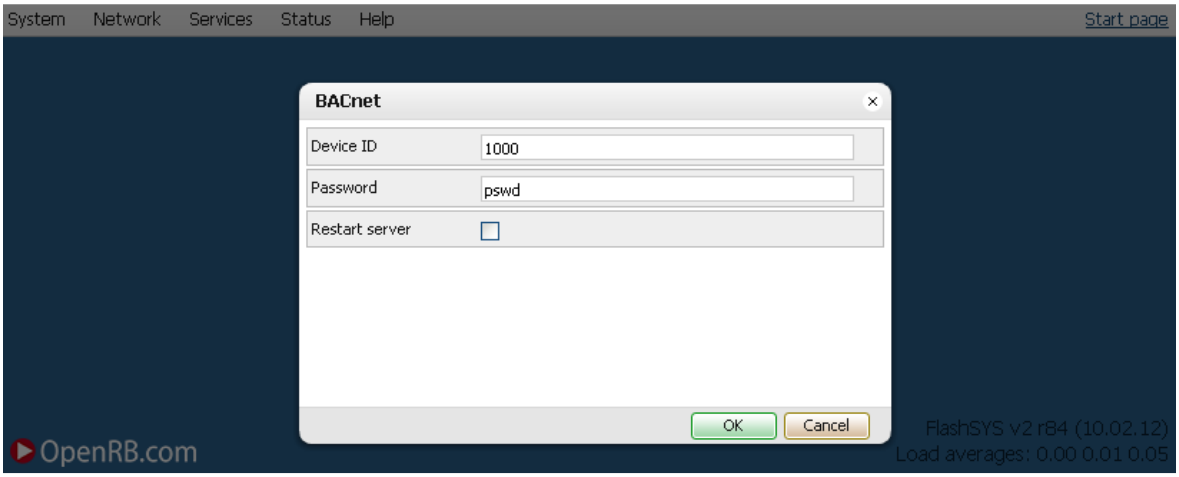
```
value = knxdatatype.decode(event.datahex, dt.uint8)
accmd(1, 'SPEED', value)
```

Example: set the temperature

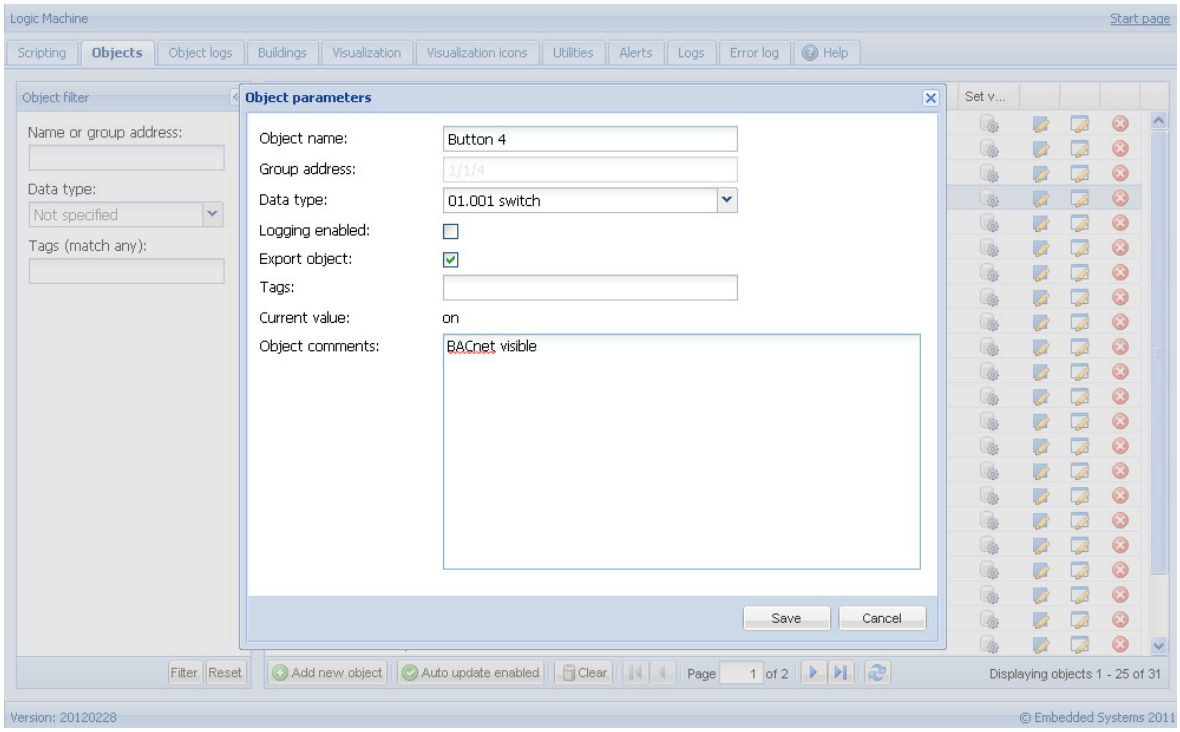
```
value = knxdatatype.decode(event.datahex, dt.float16)
accmd(1, 'TEMP', value)
```

6. BACnetIP interconnection with LM2

You can configureBACnet *Device ID* and *Password* (used for remote device reloading) in **Network Configuration → Network →BACnet.**



To make KNX/EIB objects BACnet readable/writable, mark necessary objects in Logic Machine 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 writes changes the Relinquish Default property



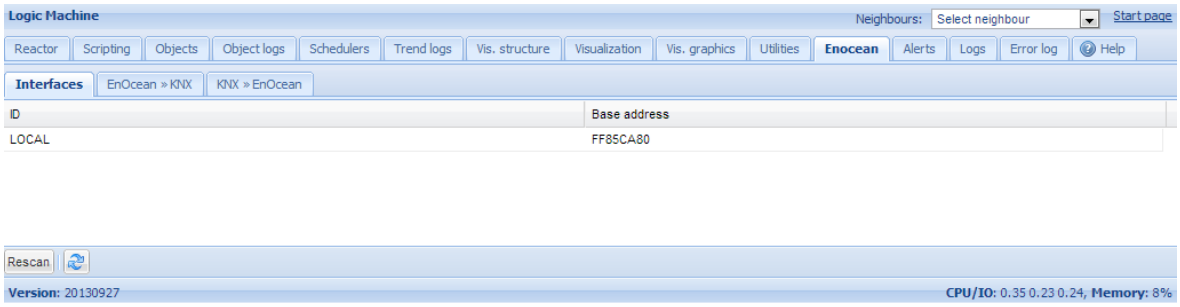
Note!BACnet service restart and Reinitialize Device requests will reload all objects, priority array will be reset to NULL.

7. EnOcean interconnection with Reactor

Logic Machine3 Reactor has EnOcean transceiver built-in with no limitation on supported count of devices.

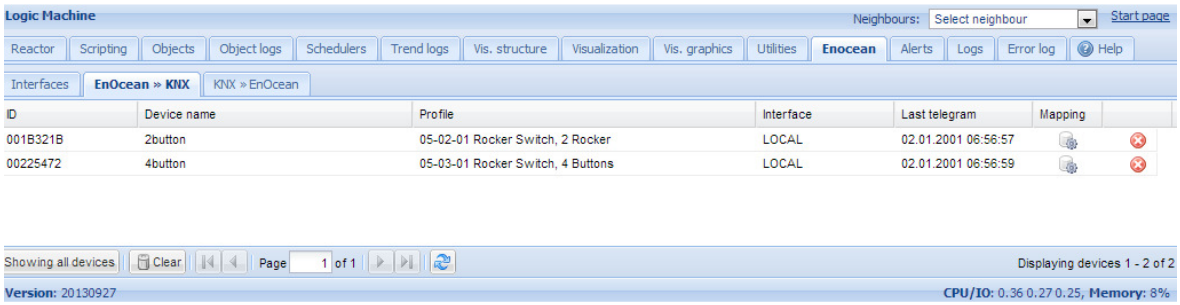
7.1. EnOcean interfaces

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

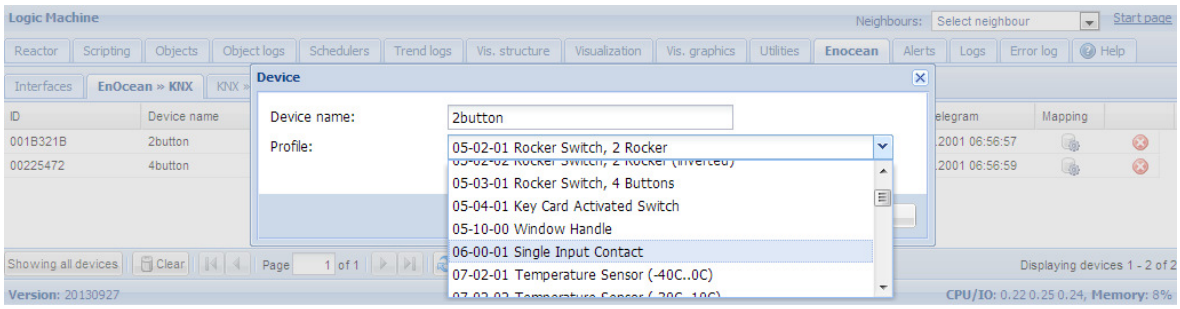


7.2. EnOceanto KNX mapping

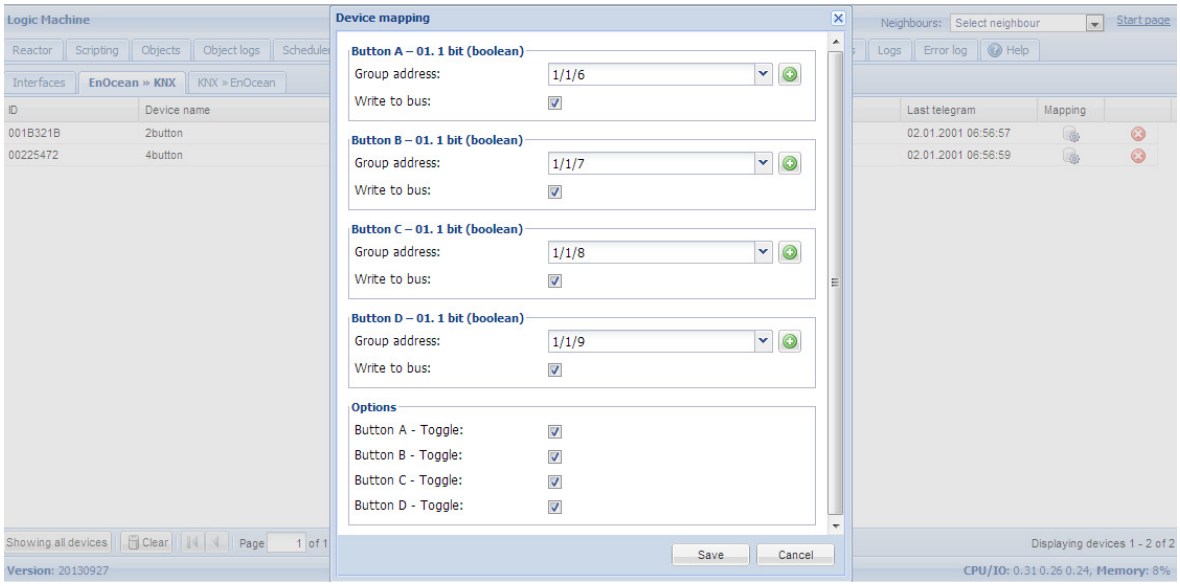
All telegrams received from EnOcean devices appears in *Enocean →KNX* section.



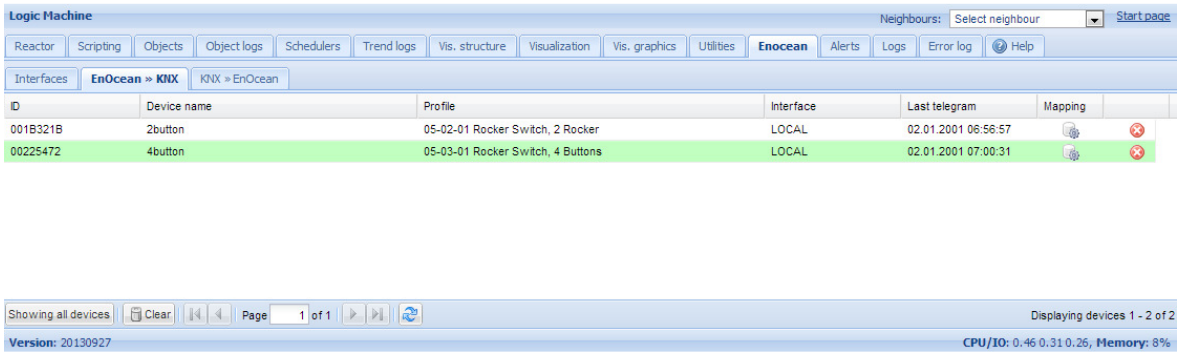
Once some specific device has to be mapped to KNX, the corresponding row has to be clicked and profile has to be chosen. There are all main profiles predefined in the list.



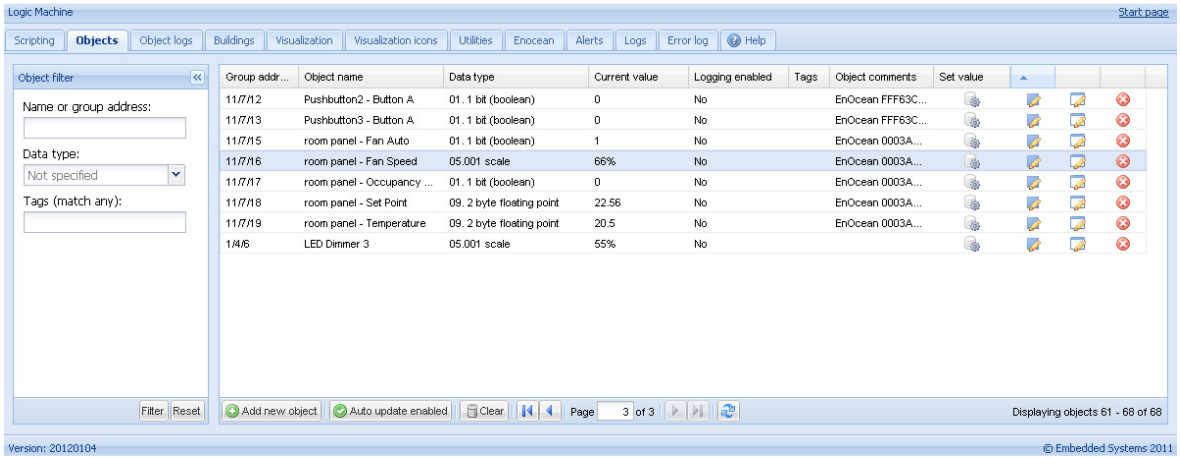
Once the device profile is set, you can map functionality of the specific device to KNX group addresses by clicking on *Mapping* icon.



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

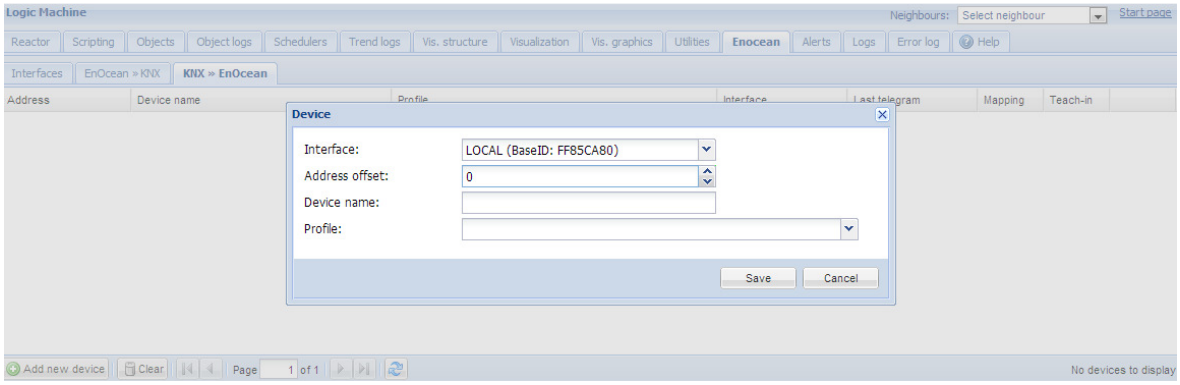


Respective KNX group addresses get updated with the new values.

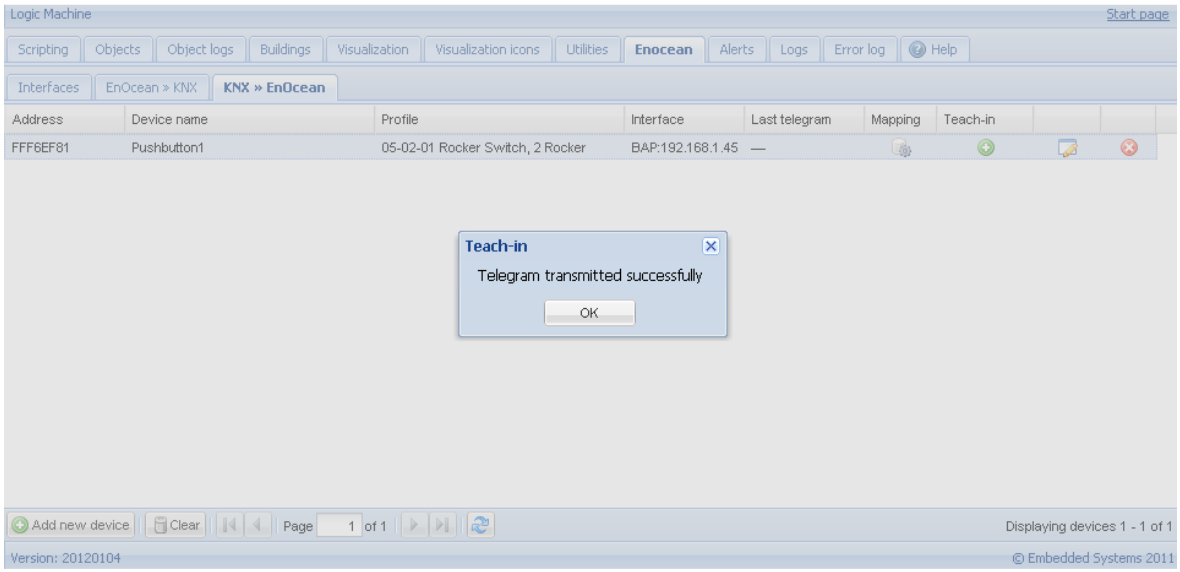


7.3. KNX to EnOcean mapping

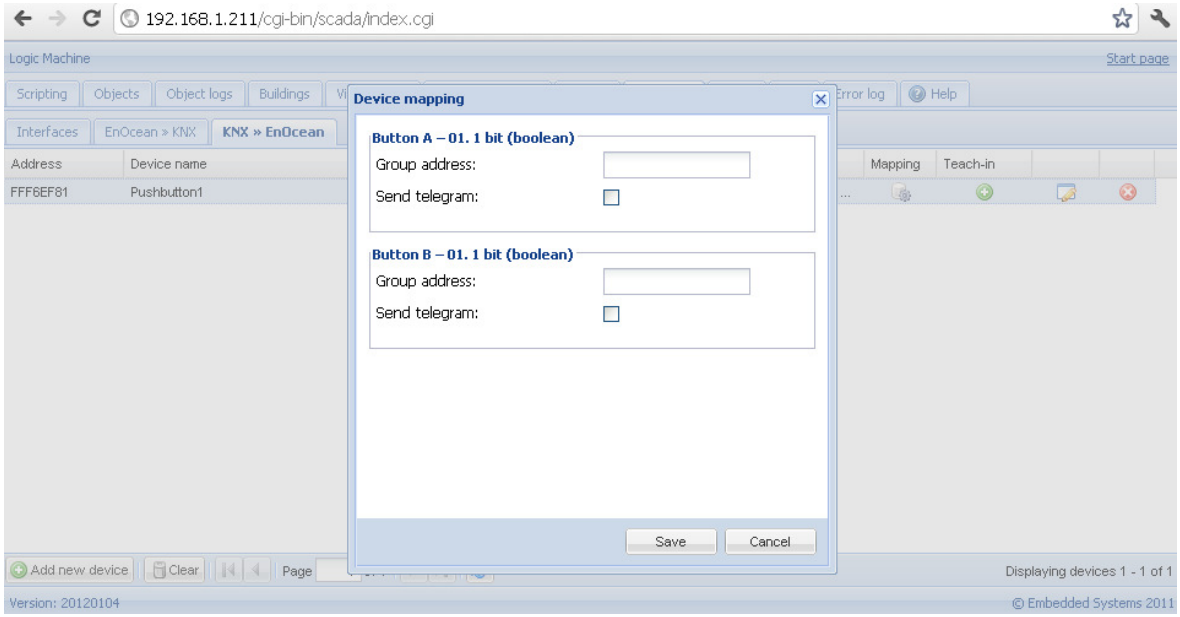
You should click on Add new device button to add EnOcean device which will be communicated from specific KNX object.



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.


192.168.1.211/cgi-bin/scada/index.cgi
Logic Machine
Start page
Scripting
Objects
Object logs
Buildings
Error log
Help
Interfaces
EnOcean » KNX
KNX » EnOcean
Address
Device name
Mapping
Teach-in
FFF6EF81
Pushbutton1
Device mapping
Button A – 01. 1 bit (boolean)
Group address:
Send telegram:
Button B – 01. 1 bit (boolean)
Group address:
Send telegram:
Save
Cancel
Add new device
Clear
Page
Displaying devices 1 - 1 of 1
Version: 20120104
© Embedded Systems 2011

8. DMX interconnection with LM2

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

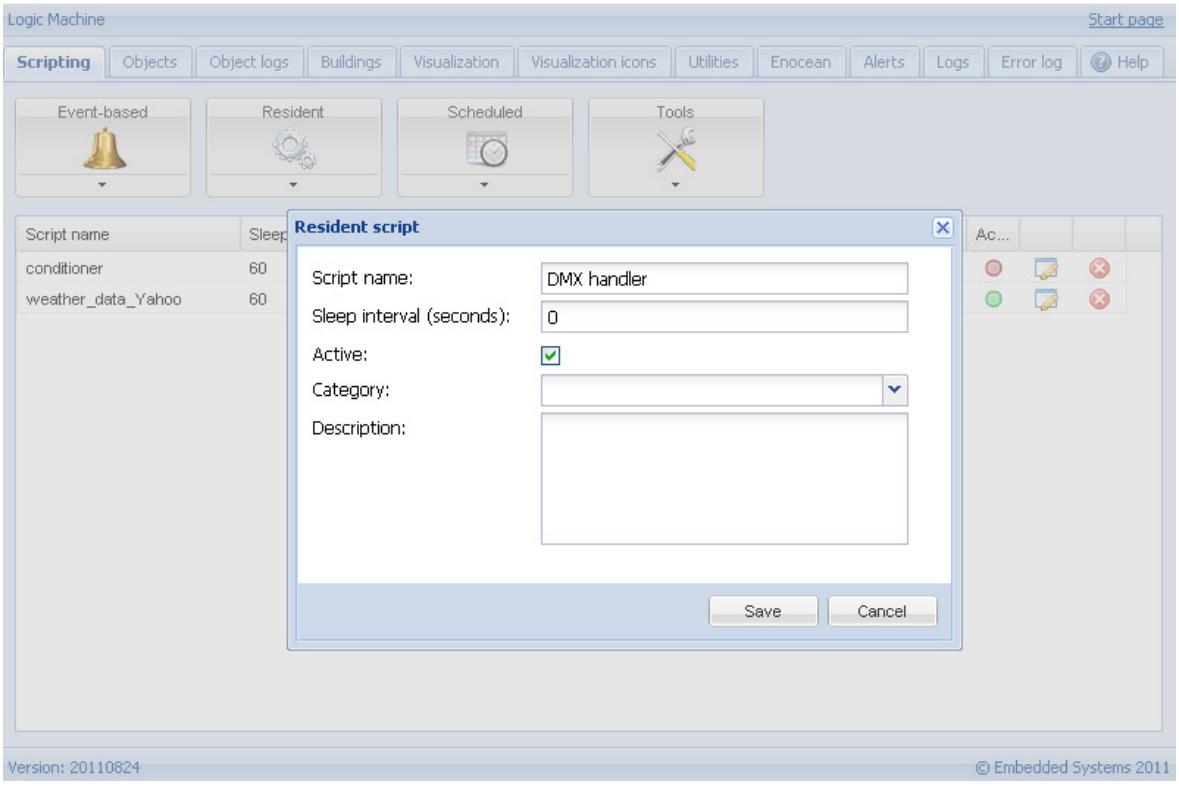
The following program has to be added in Common functions library.

```
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('luadmx')

  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 = luadmx.open(self.params.port)

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

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.if not 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]

8.1. 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
```

```
-- 3 channel scene
  scenes ={
{ 255, 0, 0 },
{ 0, 255, 0 },
{ 0, 0, 255 },
{ 255, 255, 0 },
{ 0, 255, 255 },
{ 255, 0, 255 },
{ 255, 255, 255 },
}

  current = 1
end

-- set current scene values
scene = scenes[ current ]
fori, v inipairs(scene)do
DMX.set(i, v)
end

-- switch to next scene
current = current + 1
if current > #scenes then
  current = 1
end
```

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

```
-- number of steps to use, e.g. 3 steps = { 0, 127, 255 }
steps =5
-- number of channels to set
channels =3
-- first channel number
offset = 1

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

9. 3Gmodem connection with LM2

Logic Machine 3 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. Further 3G router support will be added.

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 usign 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.

9.1. 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
```

9.2. SMS handler program

A resident script for SMS handler should be created with sleep interval 0 following code.

Note! Change white list telephone numbers and SIM card’s PIN code in the below script.

```
1.-- init
```

```
2. ifnot modem then
3. -- allowed numbers, SMS message from other number will be ignored
4.   numbers ={'1234567890', '0123456789'}
5. -- replace 0000 with SIM pin number, or remove the line below if PIN check is disabled
6. pincode='0000'
7. -- modem communication port, ttyUSB2 for Huawei E173
8.   comport ='ttyUSB2'
9. -- open serial port
10.   modem =AT:init('/dev/' .. comport)
11. -- command parser
12.   parser =function(cmd, sender)
13. local find, pos, name, mode, offset, value, jvalue, obj
14. cmd=cmd:trim()
15.     mode =cmd:sub(1, 1):upper()
16. if mode =='W'or mode =='R'then
17. cmd=cmd:sub(3):trim()
18. -- parse object name/address
19.       find =cmd:sub(1, 1)==''''and''''or' '
20.       offset = find =='''and 1 or0
21. -- pad with space when in read mode
22. if mode =='R'and find ==' 'then
23. cmd=cmd .. ' '
24. end
25. -- find name
26. pos=cmd:find(find, 1 + offset, true)
27. -- name end not found, stop
28. 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. ifpincodethen
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()
```

9.3. 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)
```

10. HDL protocol integration in Logic Machine 3

Note! Please contact Embedded Systems team to receive a special package to integrate HDL support into your LM2. Once you have the file, add it in *Network configuration -> System -> Packages*.

10.1. HDL function

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

```
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
}
}

HDL.init=function()
require('json')
require('crc16')
require('socket')

localip, chunk, chunks, data
-- read interface data
  data =json.pdecode(io.readproc('if-json'))

ifnot data ornot data.eth0 then
error('cannot get interface data')
end

-- ip header
HDL.iphdr=''
-- broadcast address
HDL.bcast= data.eth0.bcast

```

```
38. -- split ip address into chunks
39. chunks= data.eth0.inetaddr:split('.')
40.
41. -- add ip address chunks
42. fori= 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. locallen, data, src, crc
50.
51. -- primary header
52. ifpacket:sub(5, 14) ~=HDL.magicthen
53. returnnil, 'magic'
54. end
55.
56. -- leading code
57. ifpacket:sub(15, 16) ~=HDL.lcodethen
58. returnnil, 'lcode'
59. end
60.
61. -- get data length and check against
62. len=packet:byte(17)
63. iflenandlen + 16 ~=packet:len()then
64. returnnil, '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) ~=crcthen
71. returnnil, '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
  }
}
```

10.2. 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.

10.3. 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

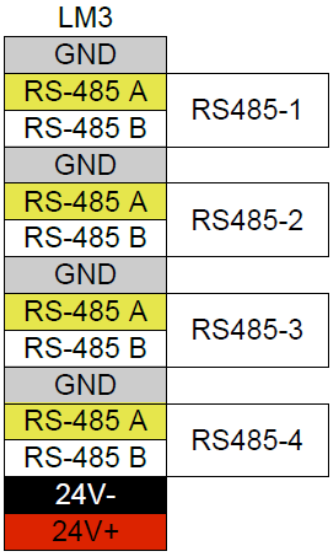
```
1. local value =dpt.decode(event.datahex, dt.bool)
2. HDL.chanreg(1, 11, 1, value))
```

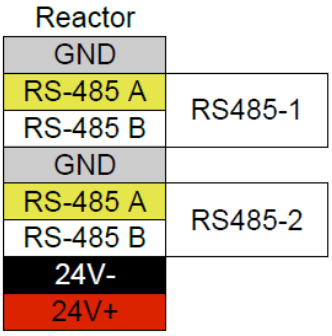
Test the program

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.

11. Communication with RS232/RS485 serial ports

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





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()
```

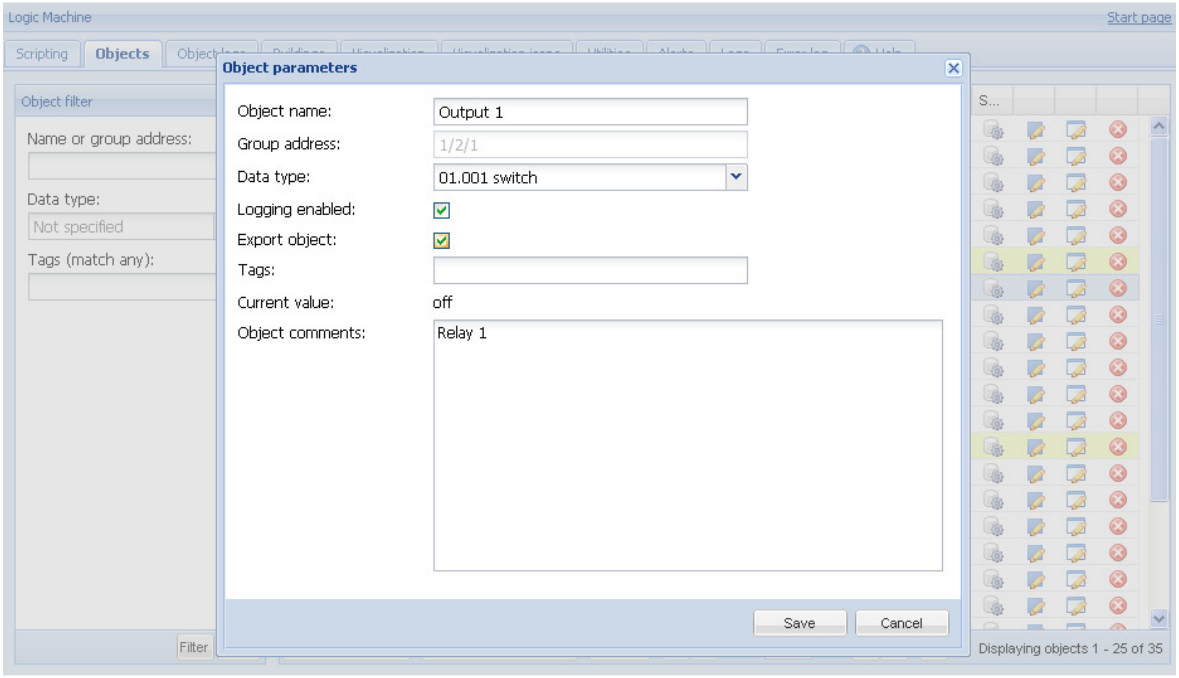
Resident script, RS-485 echo test

```
-- 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
```

12. 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

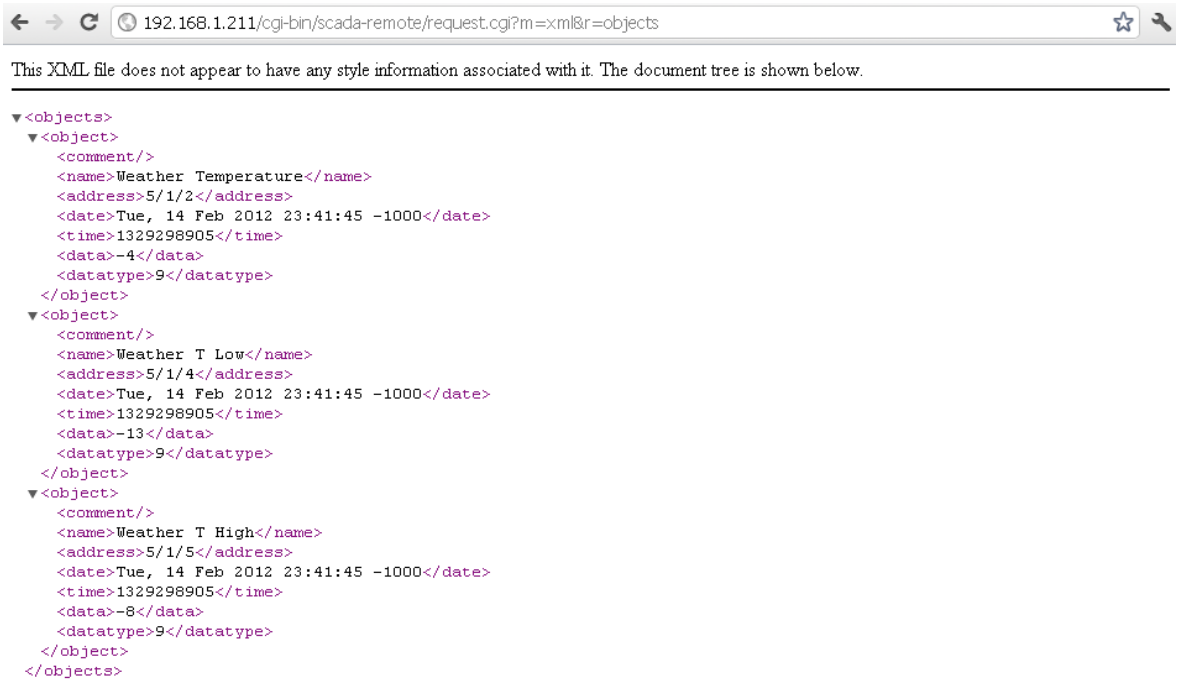
The XML request looks like this:

http://remote:remote@192.168.1.211/cgi-bin/scada-remote/request.cgi?m=xml&r=objects

Parameters:

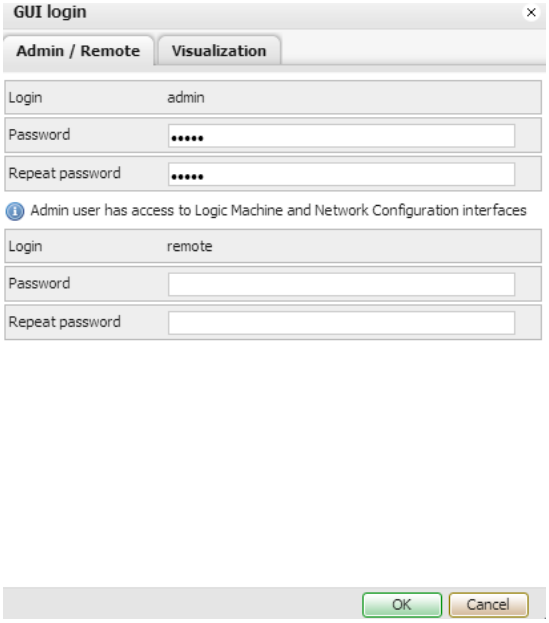
- ***address*** – object address (e.g. "1/1/1″)
- ***name*** – object name (e.g. "My object")
- ***data*** – decoded object value (e.g 42 or "01.01.2012″)
- ***datatype*** – object datatype (e.g. 1 or 5.001) – standard KNX data types
- ***time*** – object update time (UNIX timestamp)
- ***date*** – object 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.



12.1. Alerts, Errors values

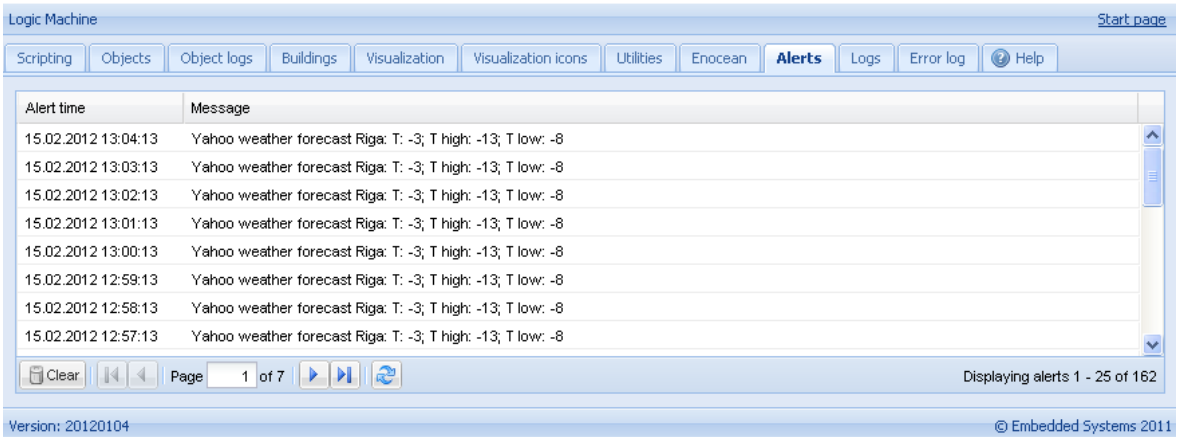
In similar way also Alerts and Errors can be read by XML requests.

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

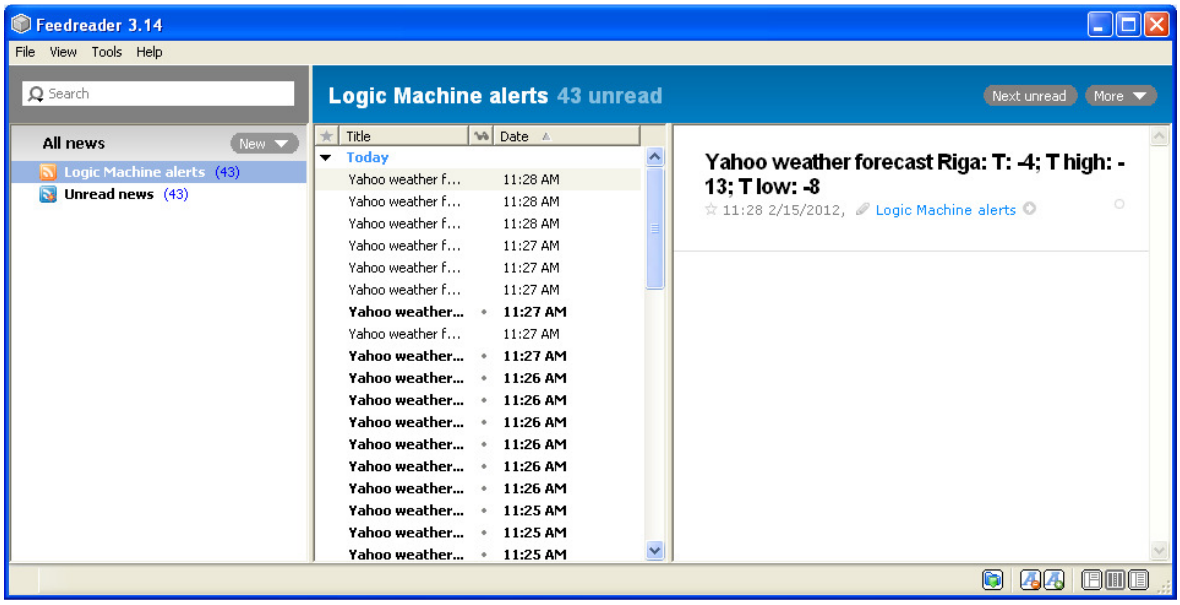
13. Read Alerts RSS feeds from Logic Machine

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.