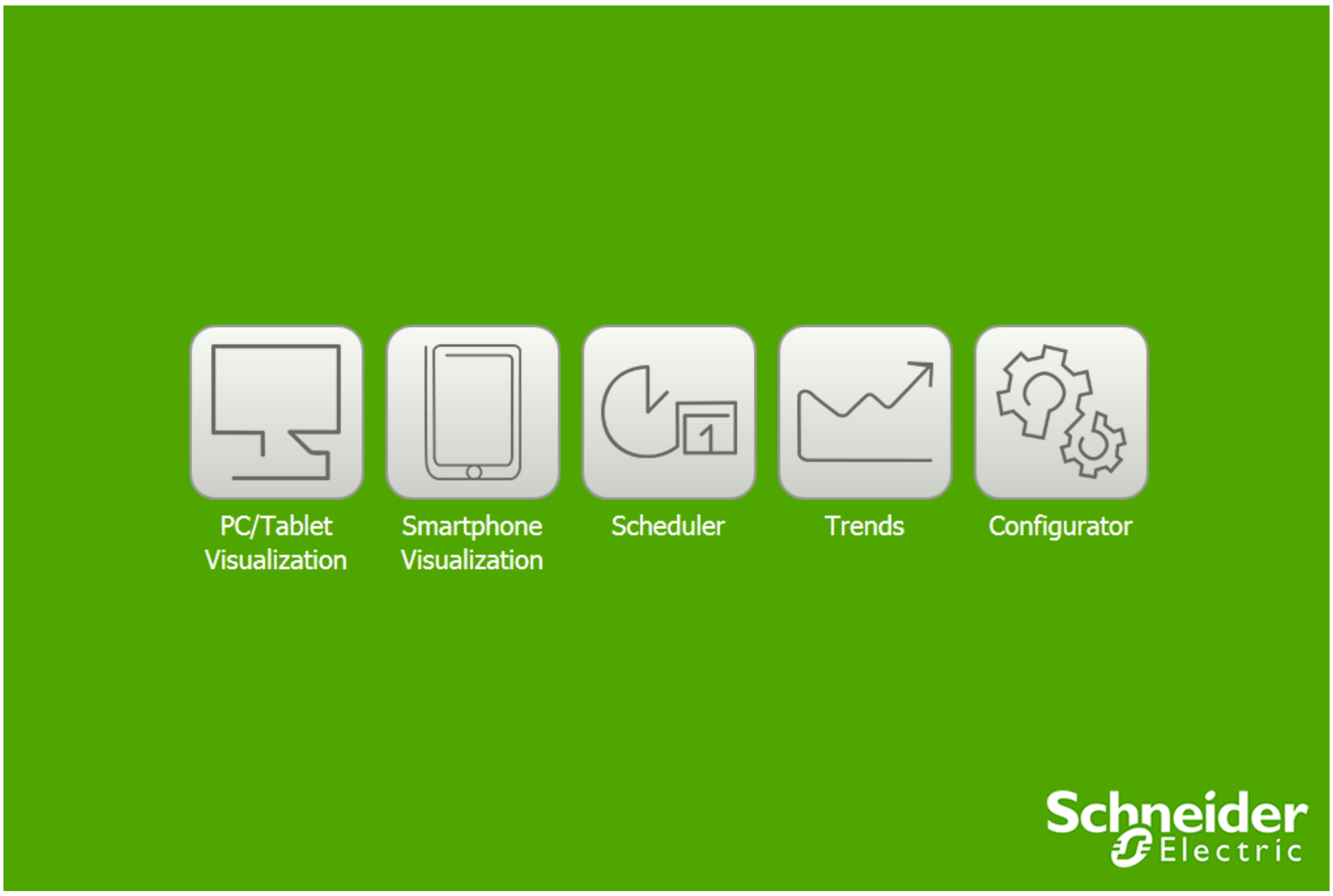
HomeLYnk
Product Manual

Contents

1 Quick start up guide

- Mount the device on DIN rail
- Connect the bus cables (KNX, ModBus, RS232) or flash drive
- Connect 24V power supply to the device
  (24V DC conductor to the red clamp, GND conductor to the blue clamp)
- Recommended accessory - power supply REG/24V DC/0,4A, article No.: **MTN693003**
- Connect Ethernet cable from the PC.
- Change the IP address of the computer to e.g. 192.168.0.9, mask 255.255.255.0
- Run Google Chrome or Mozilla Firefox (if OS Windows) Safari (if iOS) and go for 192.168.0.10.

Note: Internet Explorer is not supported.



PC/Tablet visualization – Under this icon is a visualization designed with visualization maps with objects, for PC, iPads, Android tablets (preferably 10'').

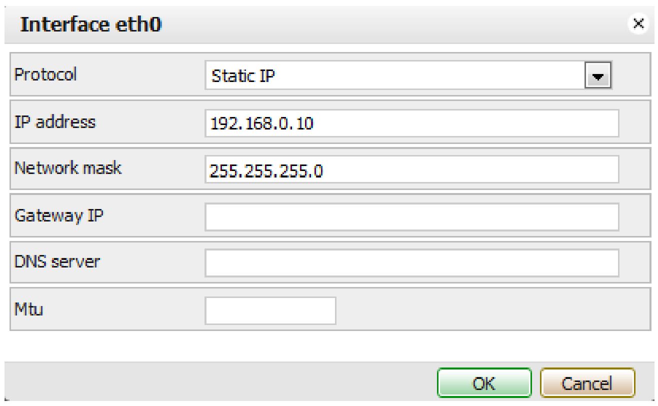
Smartphone visualization –Under this icon navigate is a Visualization designed for iPhone/iPod/iPad/Android smartphones/ Android tablets(7'' and small resolution). All objects which are added in *HomeLYnk* configurator by default are visible in this Smartphone visualization (if there is no *Hide in Smartphone* option enabled).

Scheduler – This Icon navigate to user friendly interface for end-user to manage scheduler tasks, for example, specify thermostat values depending on the day of the week, time and holidays

Trends – This icon navigate to user friendly display of Trend logs with the ability to compare data from 2 different dates. It can display trends up to 10 years.

Configurator – All programming and configurations can be performed under this icon. Access is only for admin user.

1.1 Default configuration

HomeLYnk Configurator	Login		Password
	admin		**admin**
PC/Tablet visualization/Smartphone visualization/ Schedulers/ Trends	Access right	Login	Password
	Read-only: Write: Write + admin level	**visview** **viscontrol** **visadmin**	**visview** **viscontrol** **visadmin**
IP address on LAN	**192.168.0.10**		
Networks mask on LAN	255.255.255.0		
Change IP settings In ***Configurator → Utilities → System → Network → Interfaces*** window click on the specific interface to change the IP settings.	 Interface eth0 Protocol: Static IP IP address: 192.168.0.10 Network mask: 255.255.255.0 Gateway IP: DNS server: Mtu: OK Cancel		

Protocol– Specific protocol used for addressing

- ***Static IP*** – Static IP address. By default 192.168.0.10
- ***DHCP*** – DHCP protocol used to fetch IP configuration.

IP address	IP address received from the DHCP server. This field appears only if the IP address is given or else it is 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 150

When changes are done, the following icon appears Apply changes in the top-right corner. This should be applied for changes to take effect. homeLYnk will automatically reboot after these changes are applied

1.2 Discover HomeLYnk IP address

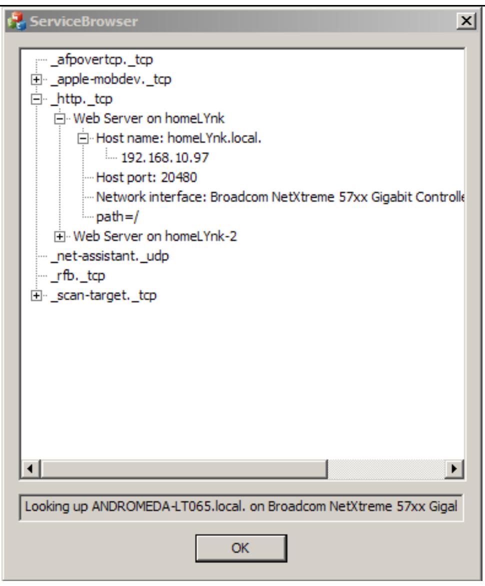
Windows PC

Easiest way is by using the utility **Service Browser** which can be downloaded here:

http://marknelson.us/2011/10/25/dns-service-discovery-on-windows/

http://marknelson.us/attachments/2011/bonjour-windows/ServiceBrowserExe.zip

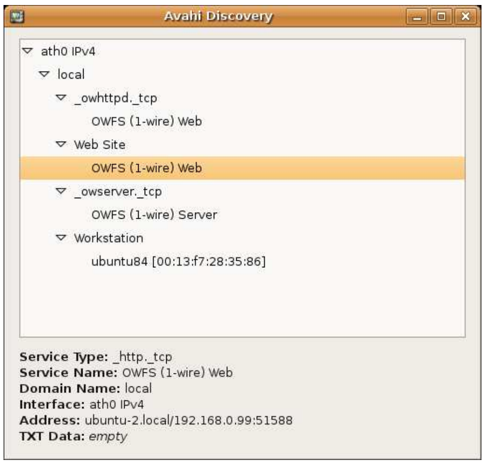
Note: Make sure that your firewall is not blocking TCP connection port :20480



```
ServiceBrowser
  _afpovertcp._tcp
+ _apple-mobdev._tcp
- _http._tcp
   - Web Server on homeLYnk
      - Host name: homeLYnk.local.
           192.168.10.97
        Host port: 20480
        Network interface: Broadcom NetXtreme 57xx Gigabit Controlle
        path=/
   + Web Server on homeLYnk-2
  _net-assistant._udp
  _rfb._tcp
+ _scan-target._tcp

Looking up ANDROMEDA-LT065.local. on Broadcom NetXtreme 57xx Gigal
OK
```

Linux PC

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

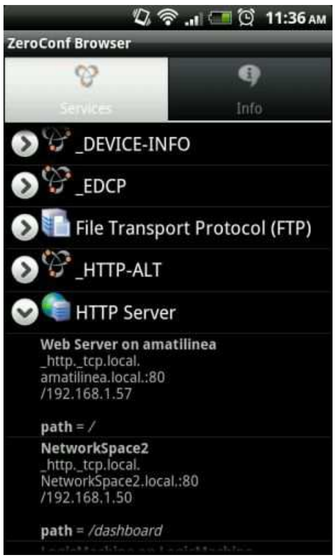


```
Avahi Discovery
▽ ath0 IPv4
  ▽ local
    ▽ _owhttpd._tcp
        OWFS (1-wire) Web
    ▽ Web Site
        OWFS (1-wire) Web
    ▽ _owserver._tcp
        OWFS (1-wire) Server
    ▽ Workstation
        ubuntu84 [00:13:f7:28:35:86]

Service Type: _http._tcp
Service Name: OWFS (1-wire) Web
Domain Name: local
Interface: ath0 IPv4
Address: ubuntu-2.local/192.168.0.99:51588
TXT Data: empty
```

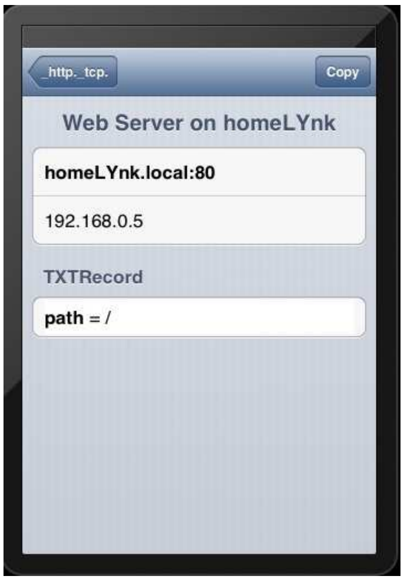
Android

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

http://play.google.com/store/apps/details?id=com.grokkt.andriod.bonjour&hl=en



```
11:36 AM
ZeroConf Browser
Services    Info
_DEVICE-INFO
_EDCP
File Transport Protocol (FTP)
_HTTP-ALT
HTTP Server
  Web Server on amatilinea
  _http._tcp.local.
  amatilinea.local.:80
  /192.168.1.57

  path = /
  NetworkSpace2
  _http._tcp.local.
  NetworkSpace2.local.:80
  /192.168.1.50

  path = /dashboard
```

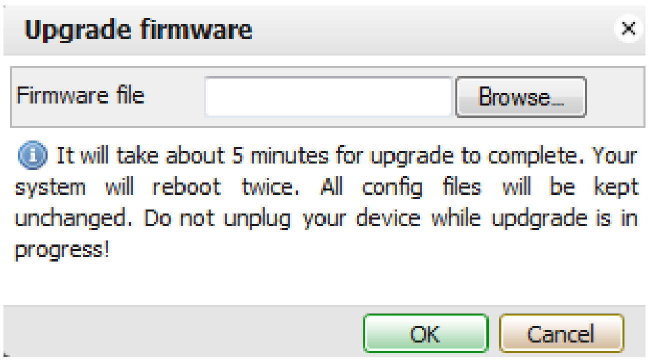
iOS/Mac OS The freely available app called **Discovery bonjour browser** can be downloaded in *App Store*: https://itunes.apple.com/en/app/discovery-bonjour-browser/id305441017?mt=8 For iPad install the iPhone/iPod version of the utility.	 _http._tcp. Copy Web Server on homeLYnk homeLYnk.local:80 192.168.0.5 TXTRecord path = /

1.3 Firmware upgrade

Note: Before each upgrade please backup the visualization, scripts and object in ***Configurator → Utilities → Backup***, as the database is cleaned during the upgrade. During the upgrade the device will not respond as it will be rebooting.

Note: After each upgrade, it is strongly recommended to clean the browser cache.

Use web browser to perform upgrade of the software of homeLYnk. Firmware is available in a form of images and can be downloaded from the support page of SE office **/Planet tool**.

Complete system upgrade ***Configurator → Utilities → System → System → Upgrade firmware***	 **Upgrade firmware** Firmware file Browse... It will take about 5 minutes for upgrade to complete. Your system will reboot twice. All config files will be kept unchanged. Do not unplug your device while updgrade is in progress! OK Cancel

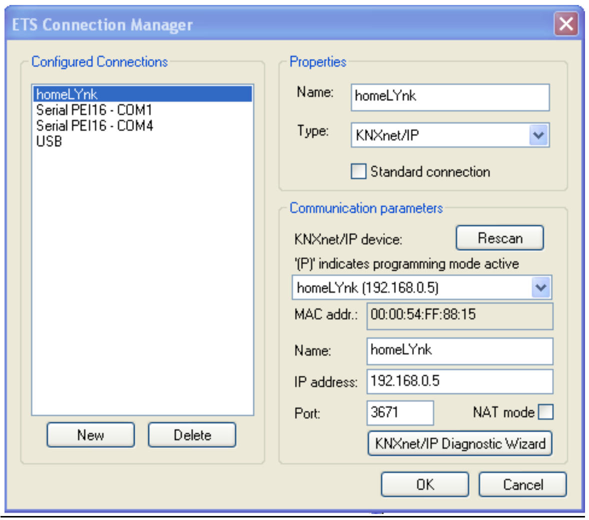
1.4 HomeLYnk for KNX/EIB network configuration management with ETS3

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

Extras → Options → Communication → Configure interfaces

Put some freely chosen *Name* for the connection
Choose ***Type = KNXnet/IP***
Press ***Rescan*** button and then choose from the drop down menu found HomeLYnk
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

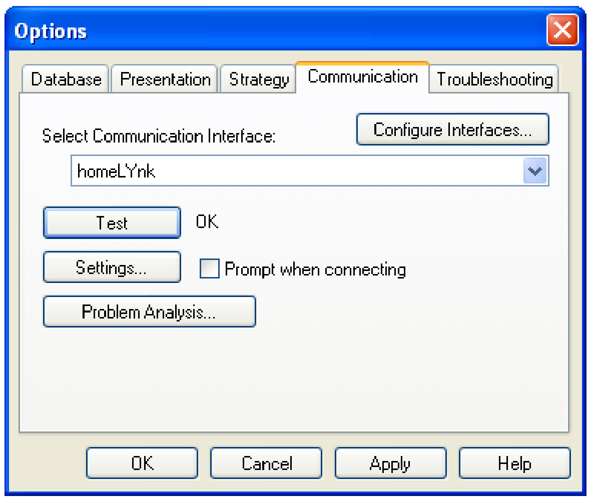
Make sure that bus status is Online – press button in ETS.


ETS Connection Manager

Configured Connections
homeLYnk
Serial PEI16 - COM1
Serial PEI16 - COM4
USB

Properties
Name: homeLYnk
Type: KNXnet/IP
☐ Standard connection

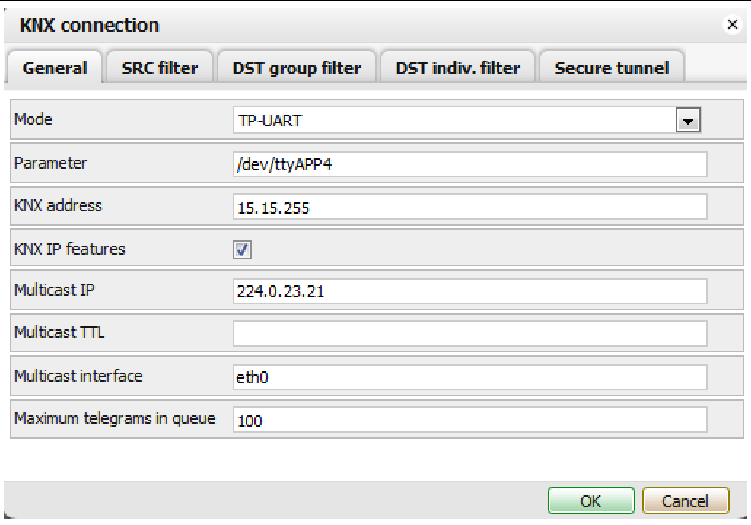
Communication parameters
KNXnet/IP device: Rescan
'(P)' indicates programming mode active
homeLYnk (192.168.0.5)
MAC addr.: 00:00:54:FF:88:15
Name: homeLYnk
IP address: 192.168.0.5
Port: 3671 NAT mode ☐


Options

Select Communication Interface: homeLYnk
Test OK
☐ Prompt when connecting

1.5 KNX and IP Router settings

KNX specific configuration is located in ***Configurator → Utilities → System → Network → KNX connection***


KNX connection

Mode	TP-UART
Parameter	/dev/ttyAPP4
KNX address	15.15.255
KNX IP features	✓
Multicast IP	224.0.23.21
Multicast TTL	
Multicast interface	eth0
Maximum telegrams in queue	100

General tab

Mode

- ➔ *TP-UART*
- ➔ *EIBnet/IP Tunneling*
- ➔ *EIBnet/IP Routing*

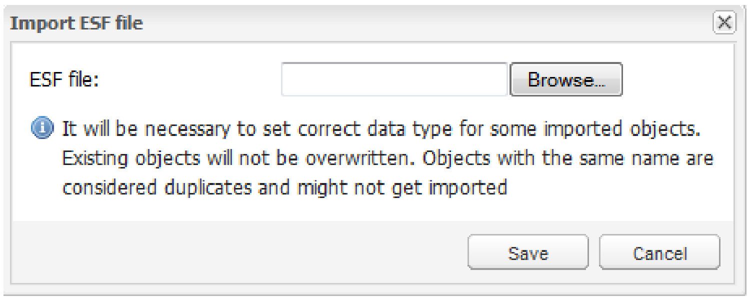
KNX connection mode. homeLYnk has TPUART interface by default built-in.

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 can be used when sending KNX telegrams to other KNX networks over UDP/IP
Multicast IP	Multicast IP address
Maximum telegrams in queue	Count of maximum telegrams in the queue

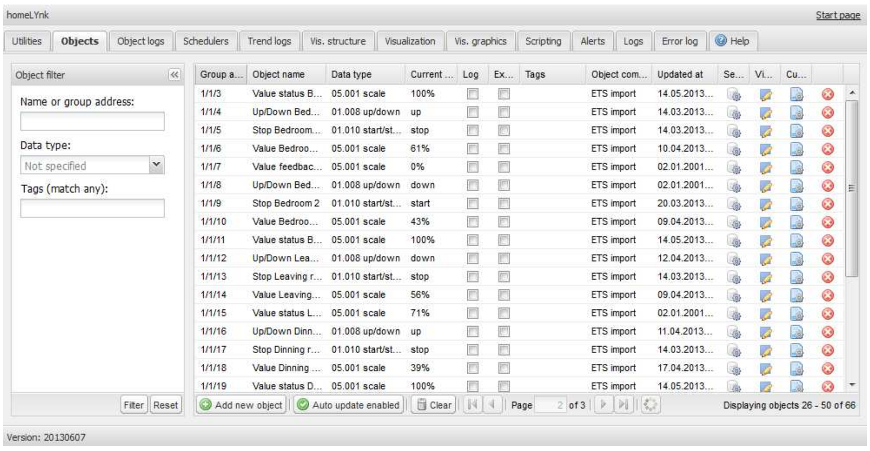
Note: If no KNX TP is connected to the device, ***Routing*** mode should be used so that the group addresses can be updated correctly. If the ***Routing*** mode is used, ***Parameter*** field should be left empty. System should be rebooted (***System*** → ***Reboot***) after each setting change in *KNX connection*.

1.6 Create visualization for iPad/PC

1.6.1 Import objects

Complete ETS project generate .ESF* file from ETS via **File → Extract data →*Export to OPC server*** Import *.ESF file to **Configurator → *Utilities* → *Import ESF file***	 **Import ESF file** ESF file: It will be necessary to set correct data type for some imported objects. Existing objects will not be overwritten. Objects with the same name are considered duplicates and might not get imported

Or connect homeLYnk to the bus and it will detect objects automatically in *Objects* tab once they are activated. This option is Enabled/Disabled **Configurator → Utilities → Configuration → Discover new objects.**
Objects can be added manually as well.


homeLYnk Start page

Utilities | Objects | Object logs | Schedulers | Trend logs | Vis. structure | Visualization | Vis. graphics | Scripting | Alerts | Logs | Error log | Help

Object filter

Name or group address:

Data type: Not specified

Tags (match any):

Filter Reset

Group a...	Object name	Data type	Current ...	Log	Ex...	Tags	Object com...	Updated at	Se...	Vi...	Cu...
1/1/3	Value status B...	05.001 scale	100%	☐	☐		ETS import	14.05.2013...			
1/1/4	Up/Down Bed...	01.008 up/down	up	☐	☐		ETS import	14.03.2013...			
1/1/5	Stop Bedroom...	01.010 start/st...	stop	☐	☐		ETS import	14.03.2013...			
1/1/6	Value Bedroo...	05.001 scale	61%	☐	☐		ETS import	10.04.2013...			
1/1/7	Value feedbac...	05.001 scale	0%	☐	☐		ETS import	02.01.2001...			
1/1/8	Up/Down Bed...	01.008 up/down	down	☐	☐		ETS import	02.01.2001...			
1/1/9	Stop Bedroom 2	01.010 start/st...	start	☐	☐		ETS import	20.03.2013...			
1/1/10	Value Bedroo...	05.001 scale	43%	☐	☐		ETS import	09.04.2013...			
1/1/11	Value status B...	05.001 scale	100%	☐	☐		ETS import	14.05.2013...			
1/1/12	Up/Down Lea...	01.008 up/down	down	☐	☐		ETS import	12.04.2013...			
1/1/13	Stop Leaving r...	01.010 start/st...	stop	☐	☐		ETS import	14.03.2013...			
1/1/14	Value Leaving...	05.001 scale	56%	☐	☐		ETS import	09.04.2013...			
1/1/15	Value status L...	05.001 scale	71%	☐	☐		ETS import	02.01.2001...			
1/1/16	Up/Down Dinn...	01.008 up/down	up	☐	☐		ETS import	11.04.2013...			
1/1/17	Stop Dinning r...	01.010 start/st...	stop	☐	☐		ETS import	14.03.2013...			
1/1/18	Value Dinning ...	05.001 scale	39%	☐	☐		ETS import	17.04.2013...			
1/1/19	Value status D...	05.001 scale	100%	☐	☐		ETS import	14.05.2013...			

Add new object | Auto update enabled | Clear | Page 2 of 3 | Displaying objects 26 - 50 of 66

Version: 20130607

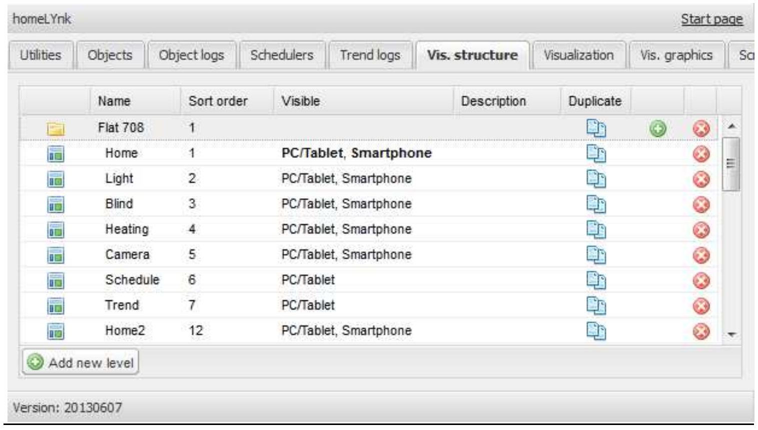
1.6.2 Create "building/floor" structure and add objects to the map

Go to **Configurator →** ***Vis.structure***
Vis. Structure allows creating all buildings levels and visualizations plans. Additionally it can create Layouts and Widgets for visualization plans.

Starting new projects, only **Layouts** and **Widgets** folders are visible. Adding new level allows defining specific **Plan** of 'flat'. **Layouts** and **Widgets** are additional tools which are not mandatory for basic visualizations; can define and implemented in many other **Plans.**

Levels

To add new Level press at Add new level. Main level usually is the project name, additional levels can be added later.


homeLYnk Start page

Utilities | Objects | Object logs | Schedulers | Trend logs | Vis. structure | Visualization | Vis. graphics | Sc

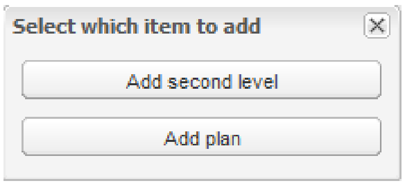
Name	Sort order	Visible	Description	Duplicate
Flat 708	1			
Home	1	PC/Tablet, Smartphone		
Light	2	PC/Tablet, Smartphone		
Blind	3	PC/Tablet, Smartphone		
Heating	4	PC/Tablet, Smartphone		
Camera	5	PC/Tablet, Smartphone		
Schedule	6	PC/Tablet		
Trend	7	PC/Tablet		
Home2	12	PC/Tablet, Smartphone		

Add new level

Version: 20130607

Second level

If additional levels are needed press on next to the main level


Select which item to add

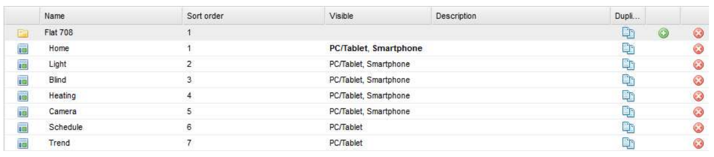
Add second level

Add plan

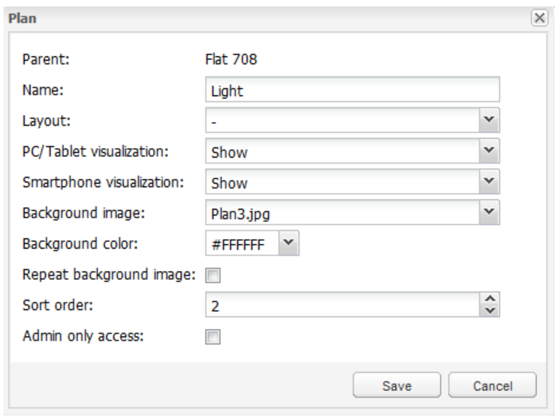
Select Add second level and provide name and sort order.
Each level can be duplicated with sublevels and plans by pressing duplicate icon next to the level.

Plans

To add Plans press on ⊕ next to a level under which another plan is to be added and select Add plan.



	Name	Sort order	Visible	Description	Dupli...		
	Flat 708	1					
	Home	1	**PC/Tablet, Smartphone**				
	Light	2	PC/Tablet, Smartphone				
	Blind	3	PC/Tablet, Smartphone				
	Heating	4	PC/Tablet, Smartphone				
	Camera	5	PC/Tablet, Smartphone				
	Schedule	6	PC/Tablet				
	Trend	7	PC/Tablet				



Plan

Parent:	Flat 708
Name:	Light
Layout:	-
PC/Tablet visualization:	Show
Smartphone visualization:	Show
Background image:	Plan3.jpg
Background color:	#FFFFFF
Repeat background image:	☐
Sort order:	2
Admin only access:	☐

Name	Name of the plan
Layout	Layout for the plan. All *Objects* from the Layout will be duplicated on the plan including background color and plan image if they are not defined separately.

PC/Tablet visualization [Show, Show and make default, Hide]

Visibility for this particular plan in PC/Tablet visualization

Smartphone visualization [Show, Show and make default, Hide]

Visibility for this particular plan in Smartphone visualization

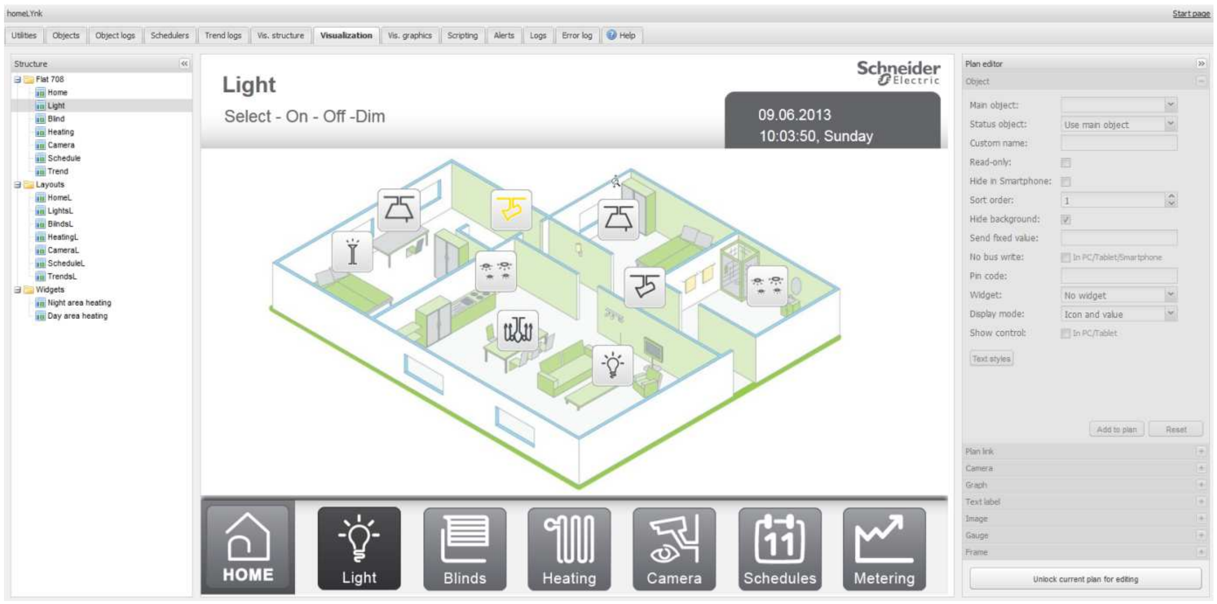
Background image	Select background previously added to Vis. graphics -> *Images/Backgrounds*
Background color	Choose background color of the plan
Repeat background image	To show the image once or repeat it and fill the whole plan
Sort order	Sort order for the plan, this is dependent on where the particular plan is located on the specific level
Admin only access	Enable admin only access for the floor (visadmin user)

Note: Each Plan can be duplicated together with all components on a plan by pressing duplicate icon next to the plan

1.6.3 Add objects to newly created visualization map

Go to **Configurator → Visualization**

After the Level and plan's structure are defined in *Vis. structure* tab, it can be visualized in the *Visualization* tab. Controlled and monitored objects can be added and managed in this section. Both the side bars can be minimized by pressing the left/right arrow icon which will make the map appear more visible especially on small displays.



Main object:	
Status object:	Use main object
Custom name:	
Read-only:	☐
Hide in Smartphone:	☐
Sort order:	1
Hide background:	✓
Send fixed value:	
No bus write:	☐ In PC/Tablet/Smartphone
Pin code:	
Widget:	No widget
Display mode:	Icon and value
Show control:	☐ In PC/Tablet

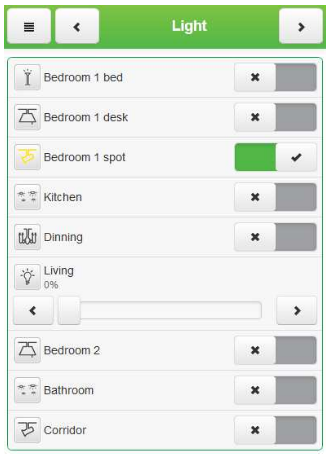
Existing objects can be added to the map by clicking on ***Unlock current floor plan for editing*** button. Once the object parameters are defined, press *Add new object* button and a newly created object will appear. This object can be moved to the desired location but whilst in editing mode the object will not work. When all the necessary objects are added, press ***Save and reload floor plan*** button so that the objects can be visualized.

1.6.4 Launching visualization on Smartphone device (iPod in this case)

- Make sure the iPod is connected wirelessly to the HomeLYnk (through separate access point – wireless router).

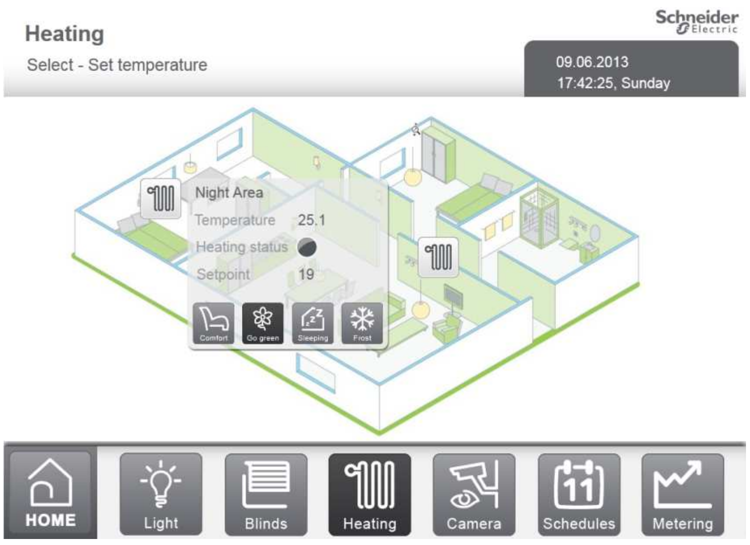
- In the browser enter HomeLYnk's IP (default 192.168.0.10).

- Click on the Smartphone *visualization* icon . Save the application as permanent /shortcut in the iPod.



1.6.5 Launching vis. on PC, Tablet or any other touch device with large screen

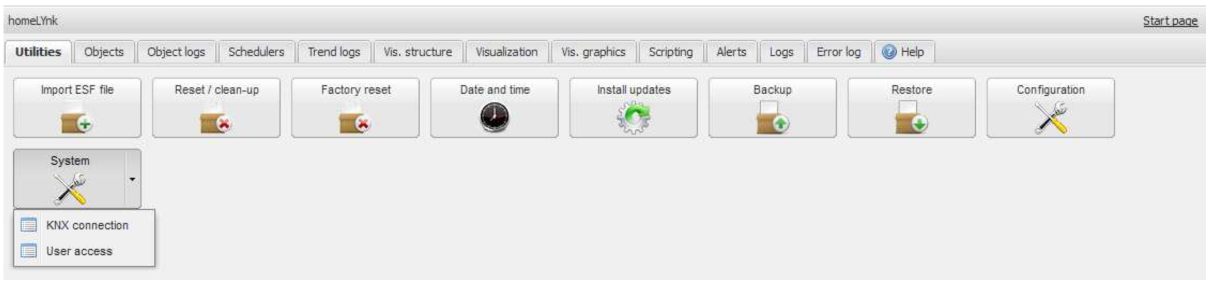
- Make sure the PC/Tablet device is able to access HomeLYnk and enter it's IP in the browser (default 192.168.0.10).
- Click on the PC/Tablet *visualization* and enter the "plan" you want to see.
- Then minimize side bar by pressing on left-arrow icon to make the map more visible.



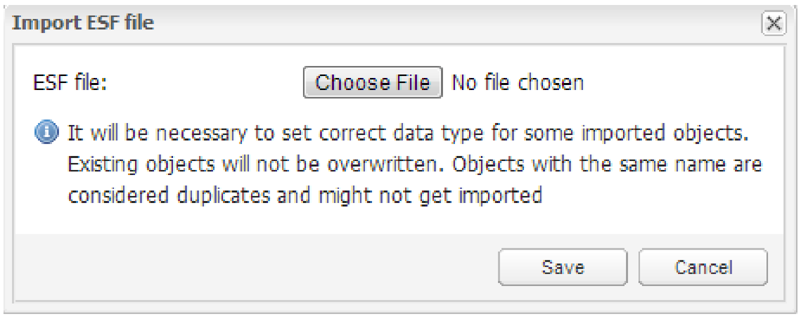
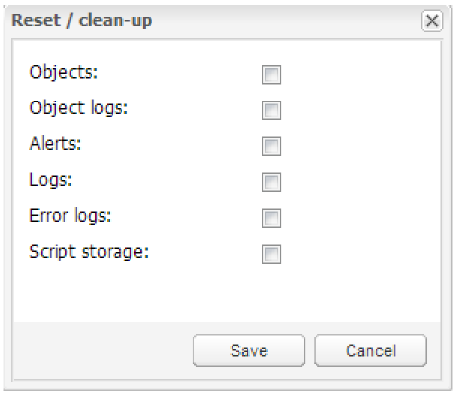
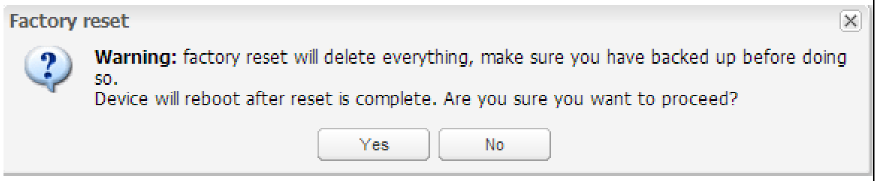
Temperature 25.1
Heating status
Setpoint 19

2 Advanced guide

2.1 Utilities



Utilities available in the tab:

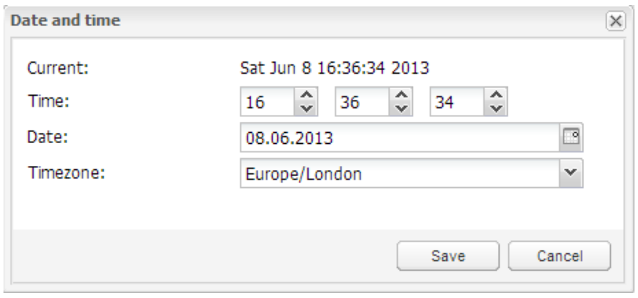
2.1.1 Import ESF file Imports the ETS object file. It is essential to set correct data types for imported objects. Existing objects would not be overwritten. Objects with the same name are considered duplicates and might not be imported.	 Import ESF file ESF file: Choose File No file chosen It will be necessary to set correct data type for some imported objects. Existing objects will not be overwritten. Objects with the same name are considered duplicates and might not get imported Save Cancel
2.1.2 Reset /clean-up Deletes all objects from the HomeLYnk, including visualization	 Reset / clean-up Objects: ☐ Object logs: ☐ Alerts: ☐ Logs: ☐ Error logs: ☐ Script storage: ☐ Save Cancel
2.1.3 Factory reset Deletes all configuration and resets to factory default settings. This feature is identical to the double long pressing of the RESET push button.	 Factory reset **Warning:** factory reset will delete everything, make sure you have backed up before doing so. Device will reboot after reset is complete. Are you sure you want to proceed? Yes No

2.1.4 Date and time

Network time protocol (NTP) is implemented. With internet connection HomeLYnk will automatically update time from servers:

0.europe.pool.ntp.org
1.europe.pool.ntp.org
2.europe.pool.ntp.org
3.europe.pool.ntp.org

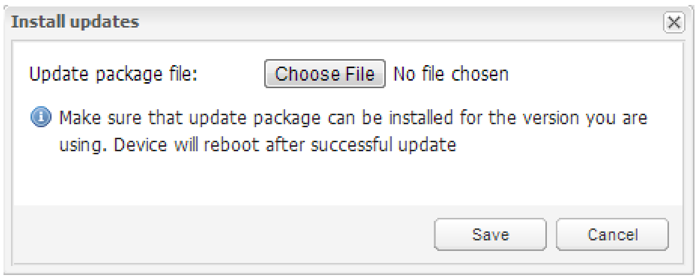
Note: It is important to select correct time zone.



Current: Sat Jun 8 16:36:34 2013
Time: 16 36 34
Date: 08.06.2013
Timezone: Europe/London

2.1.5 Install updates

Install HomeLYnk update file *.lmup. HomeLYnk will reboot after successful update



Update package file: No file chosen

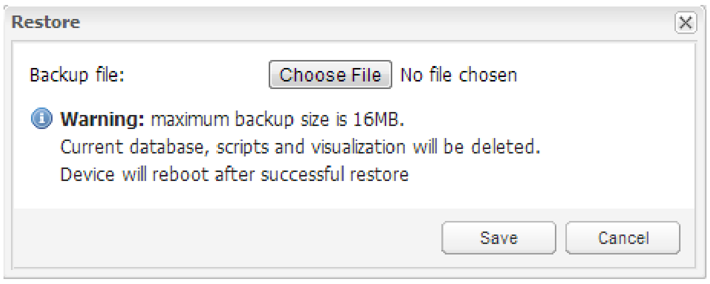
Make sure that update package can be installed for the version you are using. Device will reboot after successful update

2.1.6 Backup

Backups all the objects ,trends, logs and visualization

2.1.7 Restore

Restores configuration from backup



Backup file: No file chosen

Warning: maximum backup size is 16MB.
Current database, scripts and visualization will be deleted.
Device will reboot after successful restore

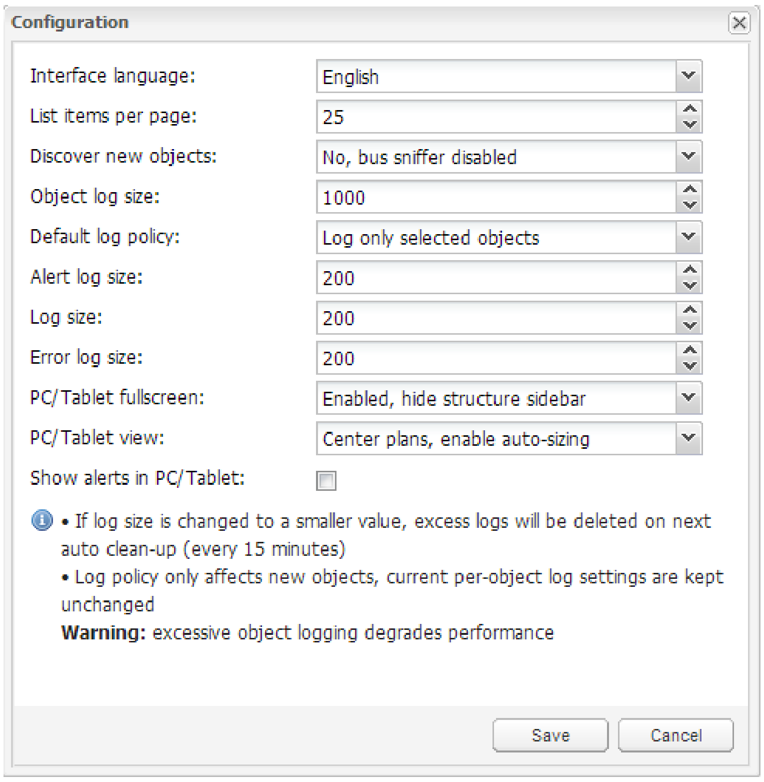
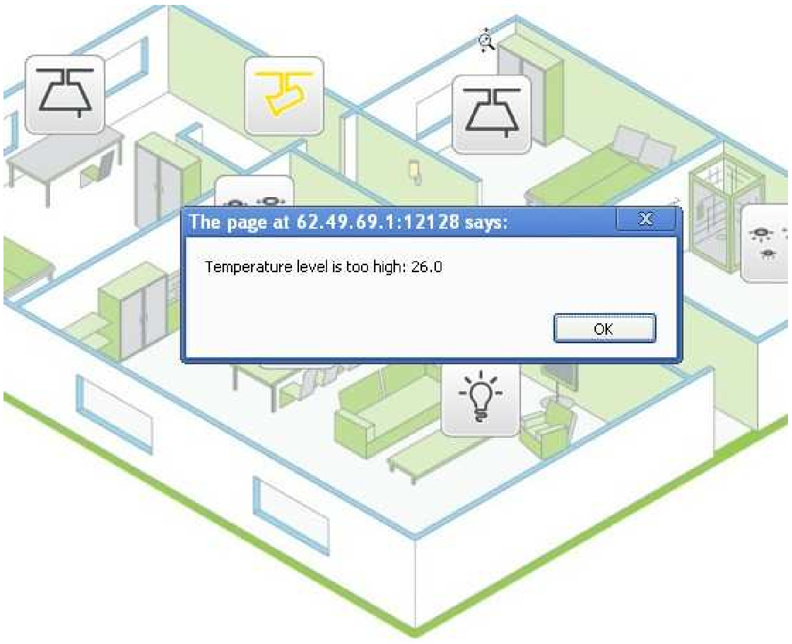
2.1.8 Configuration

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

Interface language – interface language.

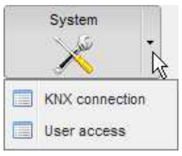
List items per page –count of lines per page e.g. *Objects, Object logs, Alerts etc.*

Discover new objects– KNX object sniffer is enabled. If YES is selected, once triggered all new objects would automatically appear in the

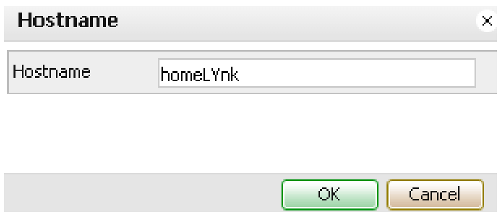
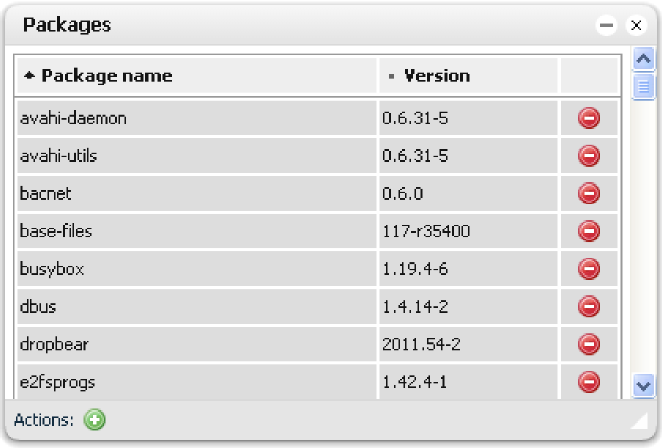
Objects list. ***Object log size*** – max count of object logs. ***Default log policy***– log status for all objects or only for checked objects can be selected. ***Alert log size*** – max count of alerts logged ***Log size***– max count of logs ***Error log size***– max count of errors logged ***PC/Tablet full screen*** – defines if the *User mode visualization* is viewed in full screen mode without any side bars. ***PC/Tablet view*** - Align plans to top left, no size limit. Centre plans, limit size, Centre plans-enable auto sizing **Note!** Auto sizing will work only in web browsers with Web Kit engine (Chrome, Safari) and Firefox ***Show alerts in PC/Tablet*** – *once new Alerts are triggered it will pop-up in PC/Tablet visualization.* **Note:** • HomeLYnk reboot is required when changing "List items per page" or "Language" parameter. **(Clear browser cache)** • HomeLYnk will keep log objects above the limit for 15 minutes; after this time elapse all records above the limit will be cleared. It is necessary to take it in to account while logging too many data in time. • Excessive object logging degrades performance	 Configuration Interface language: English List items per page: 25 Discover new objects: No, bus sniffer disabled Object log size: 1000 Default log policy: Log only selected objects Alert log size: 200 Log size: 200 Error log size: 200 PC/Tablet fullscreen: Enabled, hide structure sidebar PC/Tablet view: Center plans, enable auto-sizing Show alerts in PC/Tablet: ☐ • If log size is changed to a smaller value, excess logs will be deleted on next auto clean-up (every 15 minutes) • Log policy only affects new objects, current per-object log settings are kept unchanged **Warning:** excessive object logging degrades performance  The page at 62.49.69.1:12128 says: Temperature level is too high: 26.0

2.1.9 System

System allows managing router functionality on KNX/EIB HomeLYnk as well as access control management and firmware upgrade.



Shortcut 1

Hostname Defines host name for HomeLYnk	 Hostname Hostname: homeLYnk OK Cancel
Packages *System* → *Packages* shows the packages installed in the system. Package is functional block of the device. A new package can be added by pressing on +	 Packages Package name / Version: avahi-daemon 0.6.31-5 avahi-utils 0.6.31-5 bacnet 0.6.0 base-files 117-r35400 busybox 1.19.4-6 dbus 1.4.14-2 dropbear 2011.54-2 e2fsprogs 1.42.4-1 Actions:

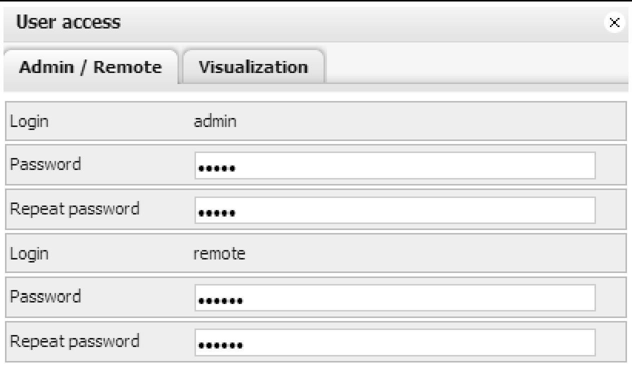
User Access

The login and password configuration window is located in *System GUI login*.

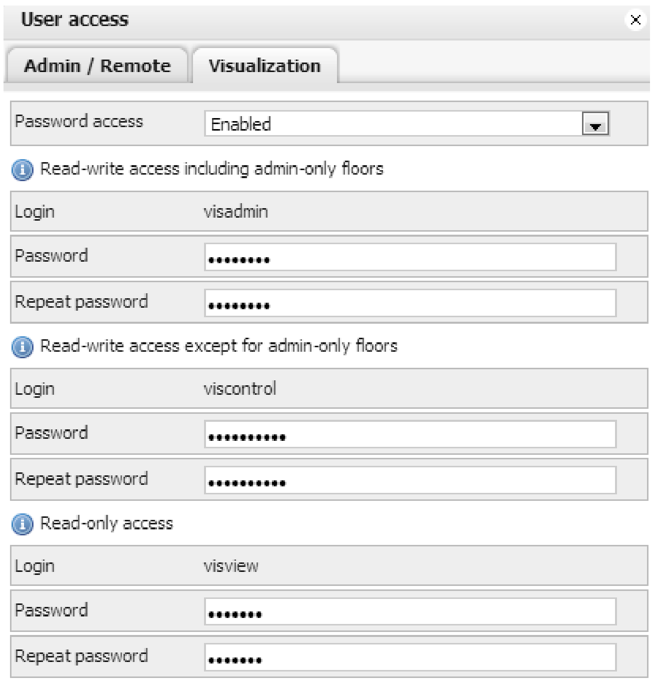
Access control is separated in 2 tabs:

Admin/Remote – access parameters for *HomeLYnk, Network, RSS* and *XML*

Visualization – access parameters for *PC/Tablet* and *Smartphone visualization*



Login	admin
Password	•••••
Repeat password	•••••
Login	remote
Password	••••••
Repeat password	••••••



Password access	Enabled

Read-write access including admin-only floors

Login	visadmin
Password	••••••••
Repeat password	••••••••

Read-write access except for admin-only floors

Login	viscontrol
Password	••••••••••
Repeat password	••••••••••

Read-only access

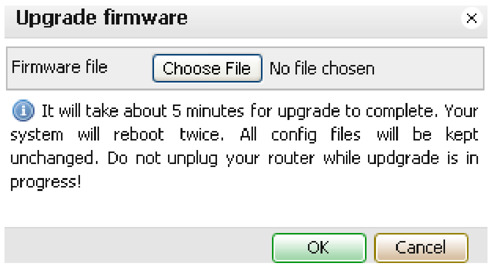
Login	visview
Password	•••••••
Repeat password	•••••••

Upgrade firmware

System → Upgrade firmware is used to perform complete upgrade of the system (both OS as well as HomeLYnk part).

Note! After each upgrade, it is strongly recommended to clean the browser cache.

Note! During firmware upgrade the device will not respond as HomeLYnk would be rebooting several times.



Firmware file — No file chosen

It will take about 5 minutes for upgrade to complete. Your system will reboot twice. All config files will be kept unchanged. Do not unplug your router while updgrade is in progress!

Reboot

By executing *System → Reboot* command HomeLYnk would restart.

Shutdown By executing *System* → *Shutdown* command HomeLYnk would shut down. **Note!** It is strongly advised to shutdown the system before the unit is powered off so that the database can be saved securely.	

Network

Interfaces

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

None– No protocol is used

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 its hidden

Network mask – network mask. By default 255.255.255.0 (/24)

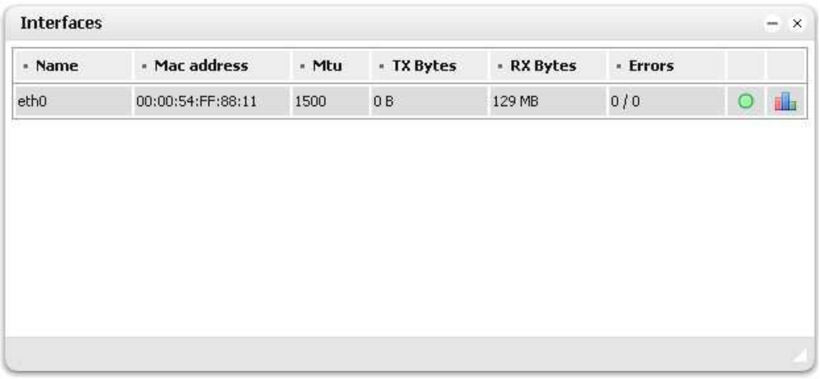
Gateway IP – gateway IP address

DNS server – DNS server IP address

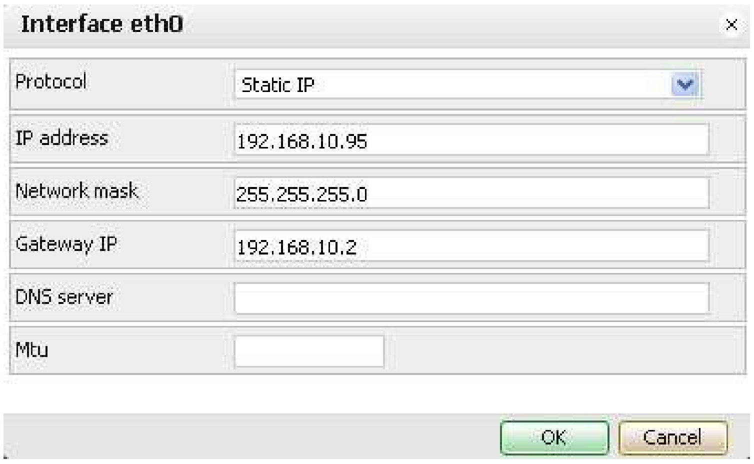
MTU– maximum transmission unit, the largest size of the packet which could be passed in the communication protocol. By default 1500

Ethernet interface data put through graph

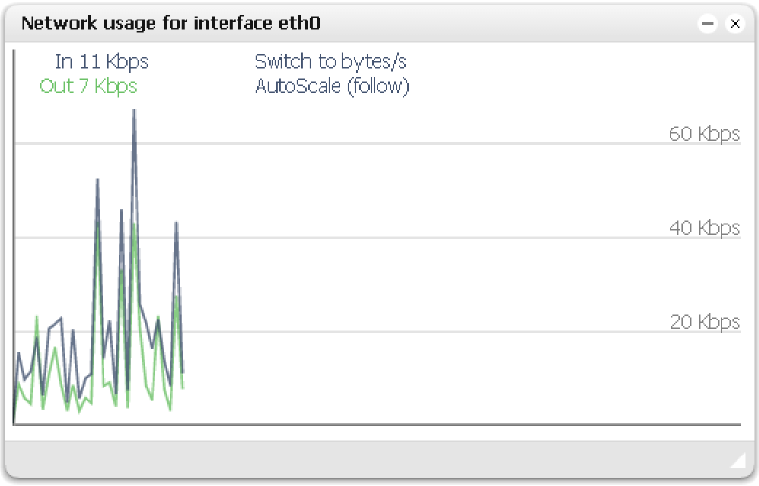
On the main window of the Ethernets tab, if you click on the button, a new window is opened. It draws a real-time graph of the traffic flow passing the interface (both In and Out). There is a possibility to switch the units of measurement – bytes/s or bytes/s.


Interfaces

Name	Mac address	Mtu	TX Bytes	RX Bytes	Errors		
eth0	00:00:54:FF:88:11	1500	0 B	129 MB	0 / 0	●	

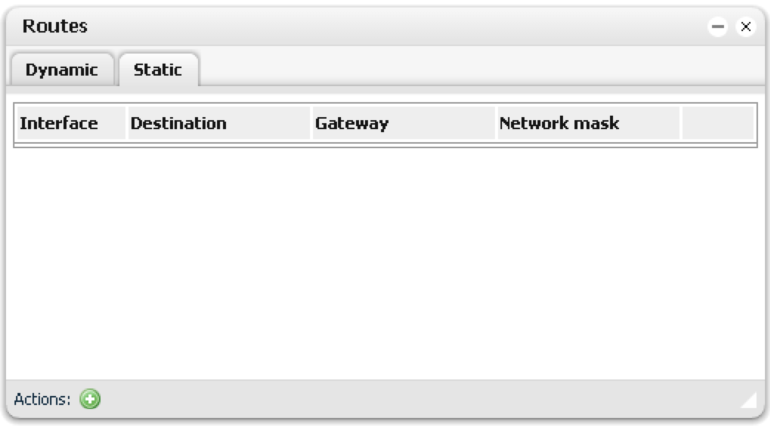

Interface eth0

Protocol	Static IP
IP address	192.168.10.95
Network mask	255.255.255.0
Gateway IP	192.168.10.2
DNS server	
Mtu	



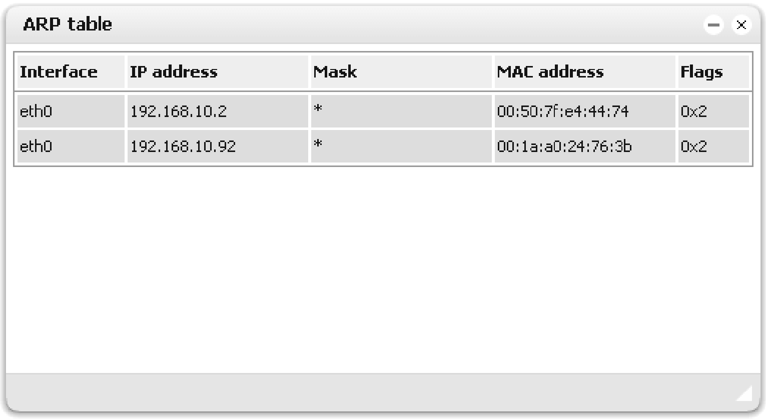
Routes

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

➔ *Dynamic routes* ***Interface*** – Interface name ***Destination***– Destination IP address ***Gateway*** – Gateway IP address ***Network mask*** – Network mask ➔ *Static routes* ***Interface*** – Interface name ***Destination***– Destination IP address ***Gateway*** – Gateway IP address ***Network mask*** – Network mask	 Routes Dynamic Static Interface Destination Gateway Network mask Actions:

ARP table

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



ARP table

Interface	IP address	Mask	MAC address	Flags
eth0	192.168.10.2	*	00:50:7f:e4:44:74	0x2
eth0	192.168.10.92	*	00:1a:a0:24:76:3b	0x2

KNX connection

KNX specific configuration is located in Configurator -> Utilities -> *Network→Network → KNX connection* window.

General

Mode *TP-UART / EIBnet IP Tunnelling (NAT mode) and IP Routing]* – KNX connection mode. HomeLYnk has TPUART interface by default built-in.

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. If not active, then all IP communication from KNX is blocked.

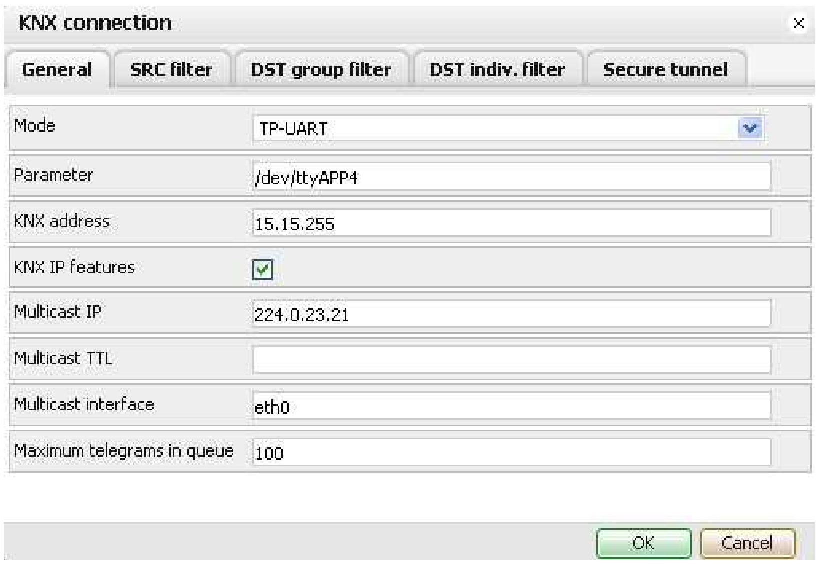
Multicast IP – multicast IP address.

Multicast TTL – default value is 1; it allows communication between different sub networks.

Multicast interface – multicast interface to use when sending KNX telegrams to other KNX networks over UDP/IP, default 224.0.23.12.

Maximum telegrams in queue – count of maximum telegrams in the queue.

Note: If KNX TP is not connected to the device, *Routing* mode should be used in order the group



Mode	TP-UART
Parameter	/dev/ttyAPP4
KNX address	15.15.255
KNX IP features	✓
Multicast IP	224.0.23.21
Multicast TTL	
Multicast interface	eth0
Maximum telegrams in queue	100

addresses are updated correctly. Once *Routing* mode is used, *Parameter* field should be empty. Please perform system reboot (*System → Reboot*) after each setting change in *KNX connection*.	

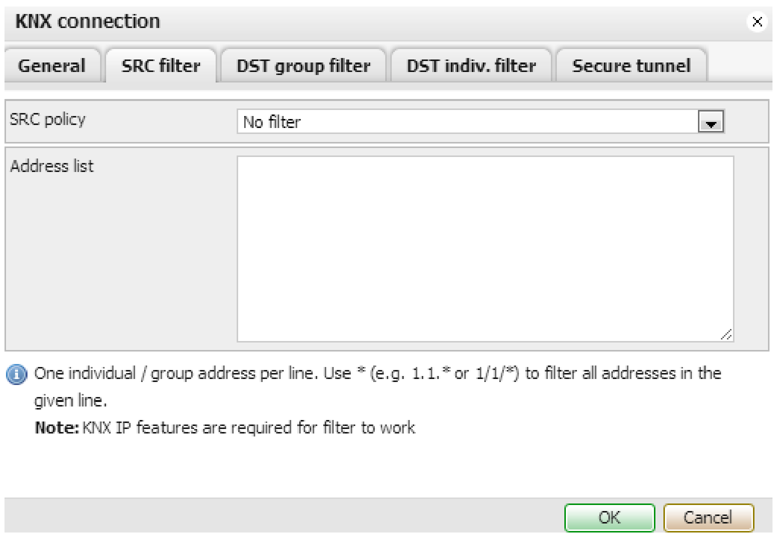
SRC filter

Source filter accepts or drops received telegrams from defined KNX devices/physical addresses. All outgoing telegrams are not filtered.

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
This applies to incoming telegrams only!



SRC policy: No filter

Address list

One individual / group address per line. Use * (e.g. 1.1.* or 1/1/*) to filter all addresses in the given line.
Note: KNX IP features are required for filter to work

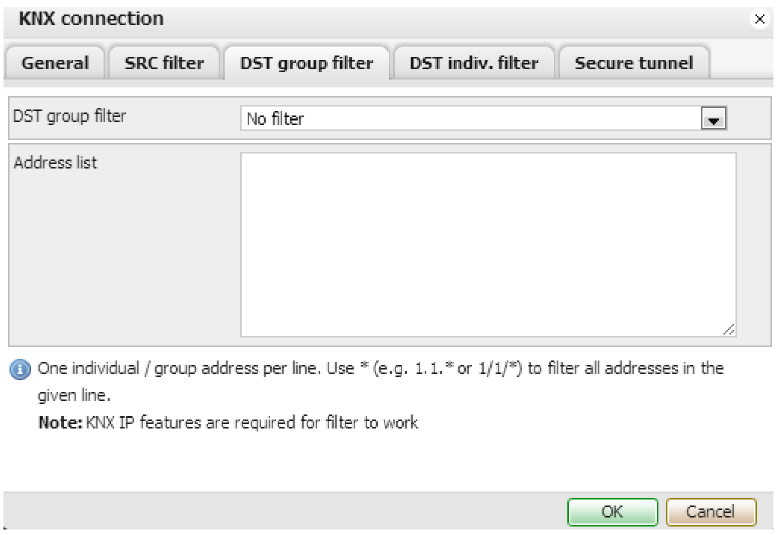
DST group filter

Destination group filter accepts or drops received telegrams belonging to one group as 1/2/3 or subgroup as 1/2/*. All outgoing telegrams are not filtered.

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.



DST group filter: No filter

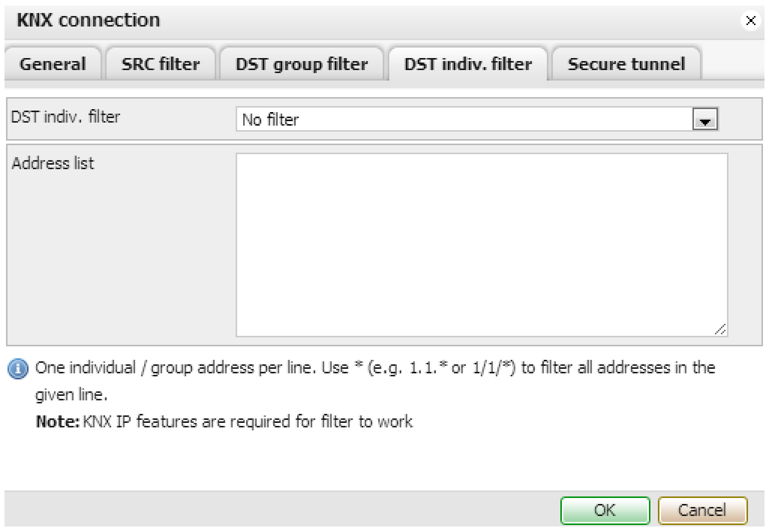
Address list

One individual / group address per line. Use * (e.g. 1.1.* or 1/1/*) to filter all addresses in the given line.
Note: KNX IP features are required for filter to work

DST indiv. filter

Destination individual filter accepts or drops received telegrams from defined KNX devices/physical addresses. All outgoing telegrams are not filtered.

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


KNX connection

DST indiv. filter: No filter

Address list

One individual / group address per line. Use * (e.g. 1.1.* or 1/1/*) to filter all addresses in the given line.
Note: KNX IP features are required for filter to work

Secure tunnel

To make a secure tunnel between two KNX networks. In comparison with standard tunnels, which use UDP protocol, this tunnelling 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.

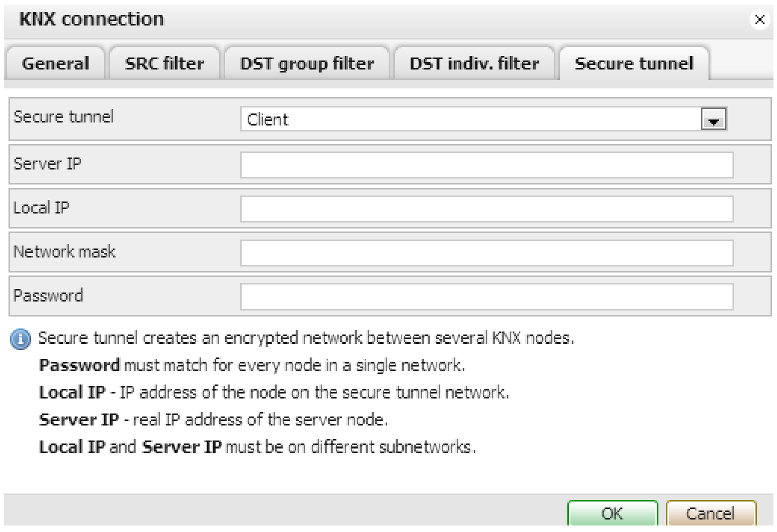
Secure tunnel *[Disabled / Client / Server]* – Secure tunnel mode.

Server IP – In case of secure client, server IP should be specified here.

Local IP– Local IP address.

Network mask – Network mask.

Password– Password.


KNX connection

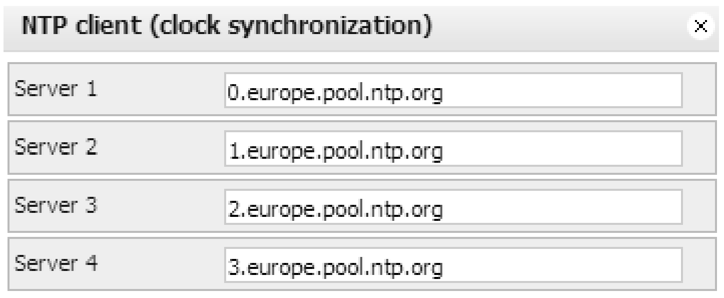
Secure tunnel: Client
Server IP
Local IP
Network mask
Password

Secure tunnel creates an encrypted network between several KNX nodes.
Password must match for every node in a single network.
Local IP - IP address of the node on the secure tunnel network.
Server IP - real IP address of the server node.
Local IP and **Server IP** must be on different subnetworks.

Services

NTP Client

Network Time Protocol (clock synchronization)
Servers 1-4
Define server where from date and time is downloaded from



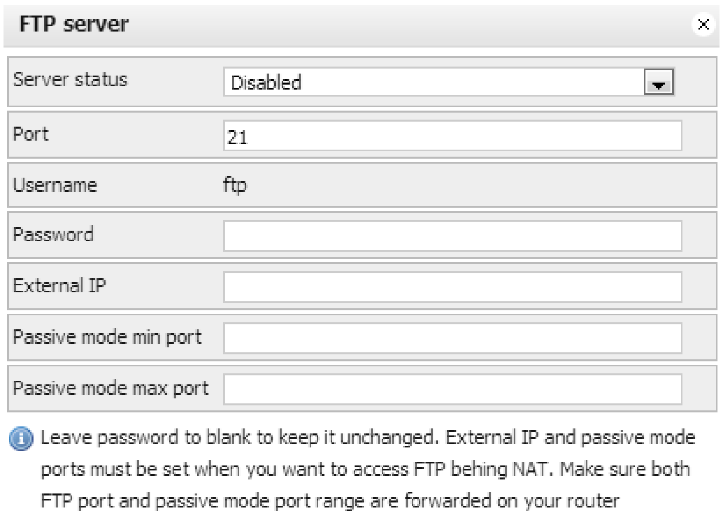
NTP client (clock synchronization)

Server 1	0.europe.pool.ntp.org
Server 2	1.europe.pool.ntp.org
Server 3	2.europe.pool.ntp.org
Server 4	3.europe.pool.ntp.org

FTP server

FTP server of HomeLYnk can be accessed by enabling Service → *FTP Server*.

Server status – secure tunnel mode
Port – port of the service
Username **–** login name, *ftp*
Password – password, length 4-20 symbol
Passive mode min port**–** minimum port of passive mode
Passive mode max port – maximum port of passive mode



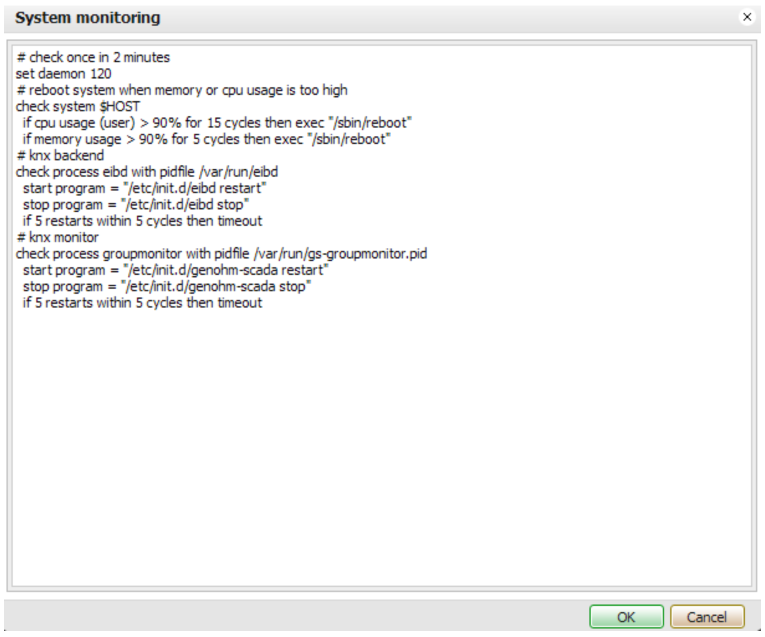
FTP server

Server status	Disabled
Port	21
Username	ftp
Password	
External IP	
Passive mode min port	
Passive mode max port	

Leave password to blank to keep it unchanged. External IP and passive mode ports must be set when you want to access FTP behing NAT. Make sure both FTP port and passive mode port range are forwarded on your router

System monitoring

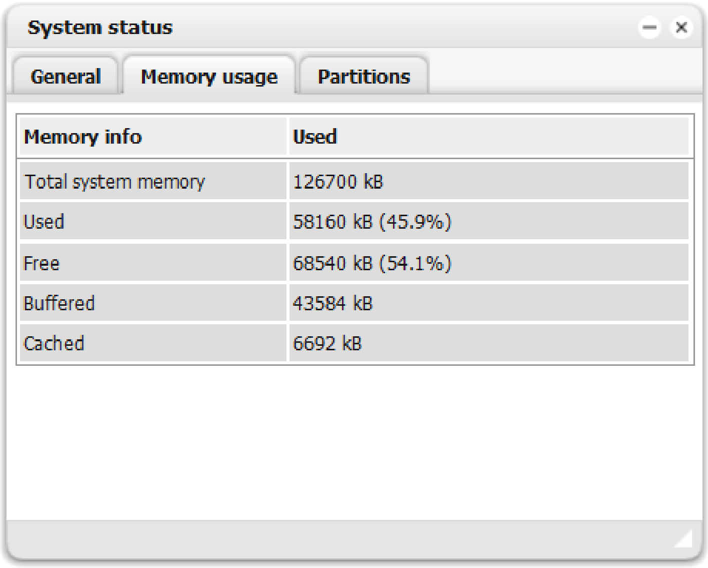
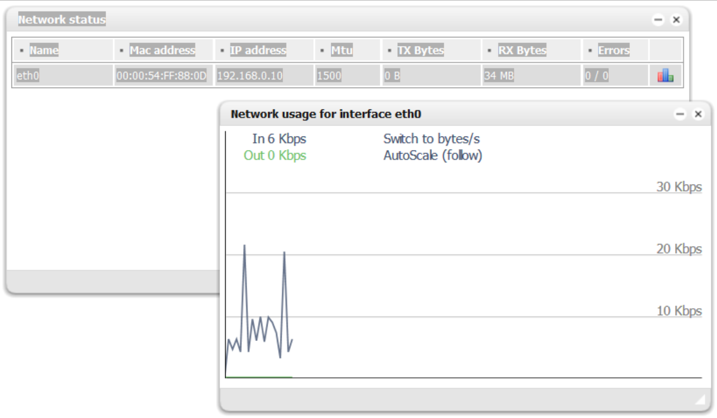
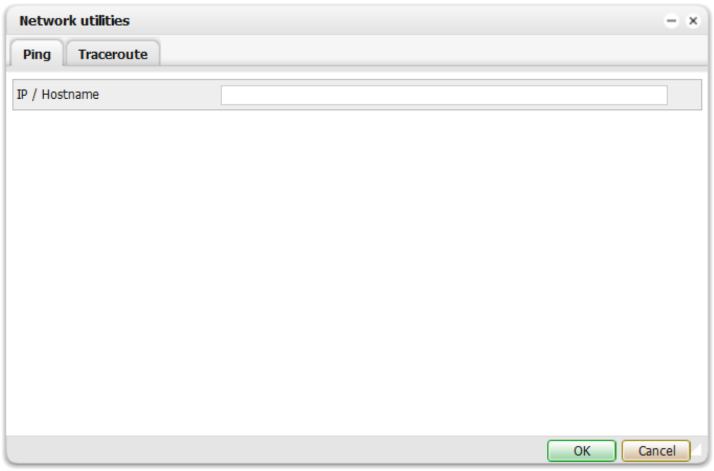
Definition of system auto check and auto reboot



System monitoring

```
# check once in 2 minutes
set daemon 120
# reboot system when memory or cpu usage is too high
check system $HOST
  if cpu usage (user) > 90% for 15 cycles then exec "/sbin/reboot"
  if memory usage > 90% for 5 cycles then exec "/sbin/reboot"
# knx backend
check process eibd with pidfile /var/run/eibd
  start program = "/etc/init.d/eibd restart"
  stop program = "/etc/init.d/eibd stop"
  if 5 restarts within 5 cycles then timeout
# knx monitor
check process groupmonitor with pidfile /var/run/gs-groupmonitor.pid
  start program = "/etc/init.d/genohm-scada restart"
  stop program = "/etc/init.d/genohm-scada stop"
  if 5 restarts within 5 cycles then timeout
```

Status

System status *General* *Memory usage* *Partitions*	
Network status	
Network utilities *Ping* *Traceroute*	

System status

General | Memory usage | Partitions

Memory info	Used
Total system memory	126700 kB
Used	58160 kB (45.9%)
Free	68540 kB (54.1%)
Buffered	43584 kB
Cached	6692 kB

Network status

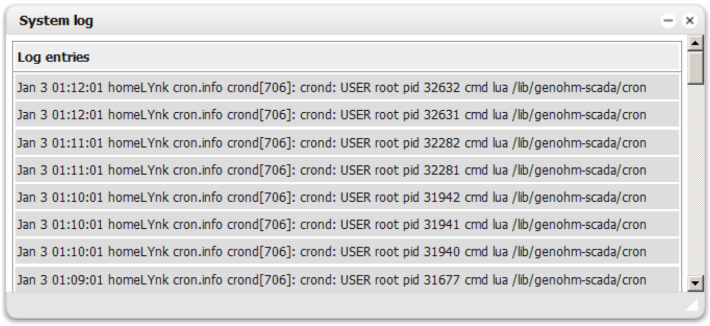
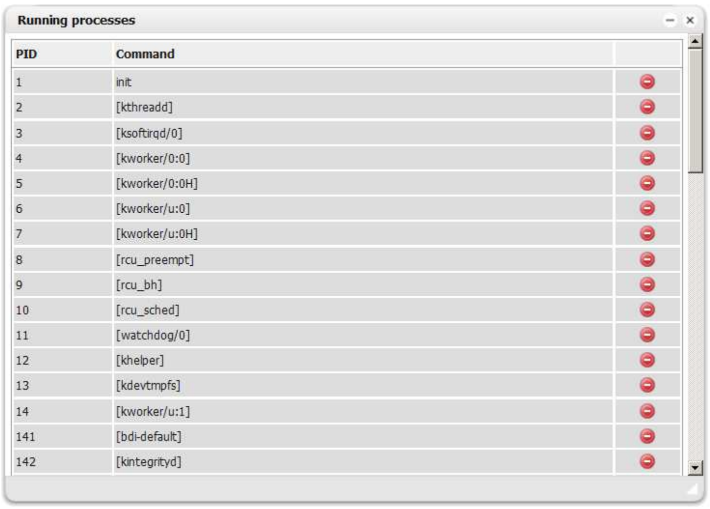
Name	Mac address	IP address	Mtu	TX Bytes	RX Bytes	Errors	
eth0	00:00:54:FF:88:0D	192.168.0.10	1500	0 B	34 MB	0 / 0	

Network usage for interface eth0

In 6 Kbps
Out 0 Kbps

Network utilities

IP / Hostname

System log	
Running processes	

System log

Log entries

```
Jan 3 01:12:01 homeLYnk cron.info crond[706]: crond: USER root pid 32632 cmd lua /lib/genohm-scada/cron
Jan 3 01:12:01 homeLYnk cron.info crond[706]: crond: USER root pid 32631 cmd lua /lib/genohm-scada/cron
Jan 3 01:11:01 homeLYnk cron.info crond[706]: crond: USER root pid 32282 cmd lua /lib/genohm-scada/cron
Jan 3 01:11:01 homeLYnk cron.info crond[706]: crond: USER root pid 32281 cmd lua /lib/genohm-scada/cron
Jan 3 01:10:01 homeLYnk cron.info crond[706]: crond: USER root pid 31942 cmd lua /lib/genohm-scada/cron
Jan 3 01:10:01 homeLYnk cron.info crond[706]: crond: USER root pid 31941 cmd lua /lib/genohm-scada/cron
Jan 3 01:10:01 homeLYnk cron.info crond[706]: crond: USER root pid 31940 cmd lua /lib/genohm-scada/cron
Jan 3 01:09:01 homeLYnk cron.info crond[706]: crond: USER root pid 31677 cmd lua /lib/genohm-scada/cron
```

Running processes

PID	Command	
1	init	⛔
2	[kthreadd]	⛔
3	[ksoftirqd/0]	⛔
4	[kworker/0:0]	⛔
5	[kworker/0:0H]	⛔
6	[kworker/u:0]	⛔
7	[kworker/u:0H]	⛔
8	[rcu_preempt]	⛔
9	[rcu_bh]	⛔
10	[rcu_sched]	⛔
11	[watchdog/0]	⛔
12	[khelper]	⛔
13	[kdevtmpfs]	⛔
14	[kworker/u:1]	⛔
141	[bdi-default]	⛔
142	[kintegrityd]	⛔

Help

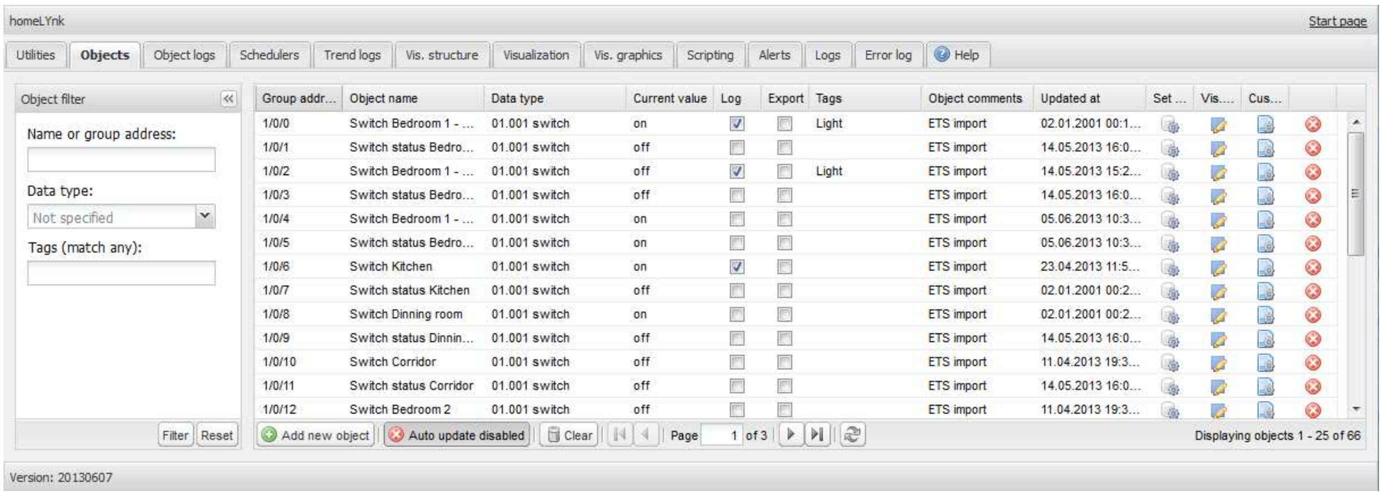
SW license description of the FlashSYS.

2.2 Objects

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

- Sniffing the bus for telegrams from unknown group addresses (if enabled in *Utilities*)
- Adding manually
- Importing ESF file (in *Utilities*)

Objects can be sorted with the following parameters– *Name, Group address, Data type, Current value, Tags, Comments* and *Updated at*.



Name or group address:

Data type: Not specified

Tags (match any):

Group addr...	Object name	Data type	Current value	Log	Export	Tags	Object comments	Updated at
1/0/0	Switch Bedroom 1 - ...	01.001 switch	on	✓	☐	Light	ETS import	02.01.2001 00:1...
1/0/1	Switch status Bedro...	01.001 switch	off	☐	☐		ETS import	14.05.2013 16:0...
1/0/2	Switch Bedroom 1 - ...	01.001 switch	off	✓	☐	Light	ETS import	14.05.2013 15:2...
1/0/3	Switch status Bedro...	01.001 switch	off	☐	☐		ETS import	14.05.2013 16:0...
1/0/4	Switch Bedroom 1 - ...	01.001 switch	on	☐	☐		ETS import	05.06.2013 10:3...
1/0/5	Switch status Bedro...	01.001 switch	on	☐	☐		ETS import	05.06.2013 10:3...
1/0/6	Switch Kitchen	01.001 switch	on	✓	☐		ETS import	23.04.2013 11:5...
1/0/7	Switch status Kitchen	01.001 switch	off	☐	☐		ETS import	02.01.2001 00:2...
1/0/8	Switch Dinning room	01.001 switch	on	☐	☐		ETS import	02.01.2001 00:2...
1/0/9	Switch status Dinnin...	01.001 switch	off	☐	☐		ETS import	14.05.2013 16:0...
1/0/10	Switch Corridor	01.001 switch	off	☐	☐		ETS import	11.04.2013 19:3...
1/0/11	Switch status Corridor	01.001 switch	off	☐	☐		ETS import	14.05.2013 16:0...
1/0/12	Switch Bedroom 2	01.001 switch	off	☐	☐		ETS import	11.04.2013 19:3...

Page 1 of 3

Displaying objects 1 - 25 of 66

Version: 20130607

2.2.1 Object parameters

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

Object name – Name for the object

Group address – Group address of this object

Data type – KNX data type for the object. This has to be set once the HomeLYnk sniffs the new object for actual work.

Unit/suffix – Add unit/suffix to value of object. Units which cannot be created from keyboard can be created in external editor and pasted in to the browser

Log – Enable logging for this object. Logs will appear in *Objects logs* tab.

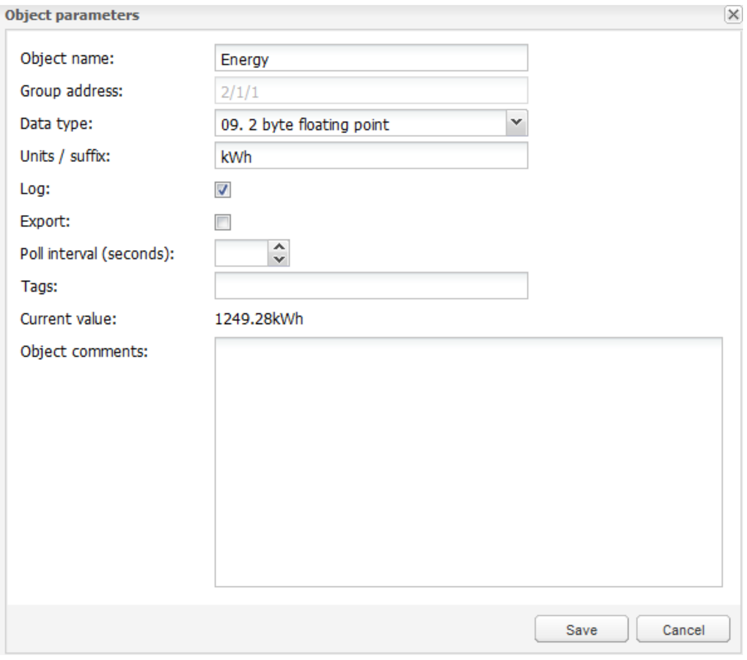
Export – Make object visible by remote XML requests

Poll interval (seconds) – Perform automatic object read after the selected time interval

Tags – Assign this object to some tag which can be later used in writing scripts, for example, *All_lights_first_floor*. (Please refer to the Script library for use cases)

Current value– Current value of the object

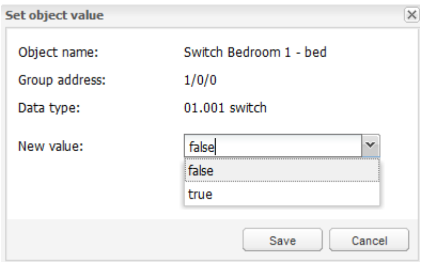
Object comments – Comment for the object



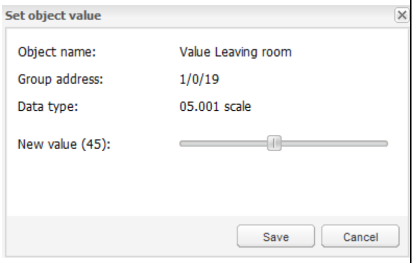
Object name:	Energy
Group address:	2/1/1
Data type:	09. 2 byte floating point
Units / suffix:	kWh
Log:	☑
Export:	☐
Poll interval (seconds):	
Tags:	
Current value:	1249.28kWh
Object comments:	

2.2.2 Set value

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



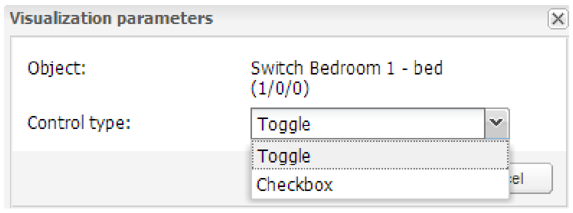
Object name:	Switch Bedroom 1 - bed
Group address:	1/0/0
Data type:	01.001 switch
New value:	false (false / true)



Object name:	Value Leaving room
Group address:	1/0/19
Data type:	05.001 scale
New value (45):	

2.2.3 Object visualization parameters

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

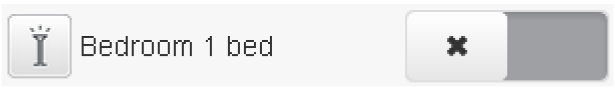

Visualization parameters

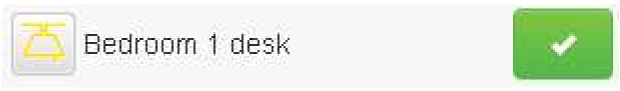
Object: Switch Bedroom 1 - bed (1/0/0)

Control type: Toggle

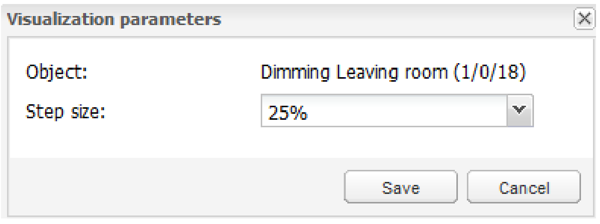
1 bit

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



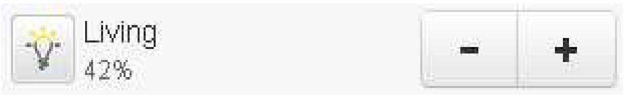


4 bit (3 bit controlled)


Visualization parameters

Object: Dimming Leaving room (1/0/18)

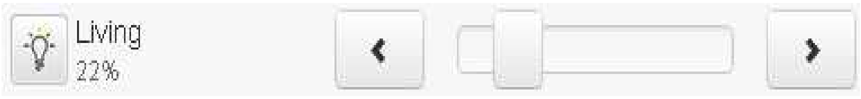
Step size: 25%



Step size – step size 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

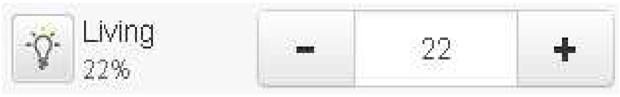
Control type – type of the visual control element

Slider



Direct input / Step

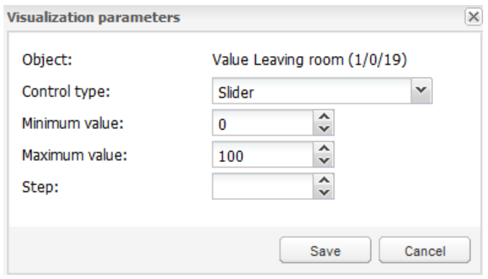
+/-



Minimum value

Maximum value

Step size – If defined value will change depend of defined step


Visualization parameters

Object: Value Leaving room (1/0/19)

Control type: Slider

Minimum value: 0

Maximum value: 100

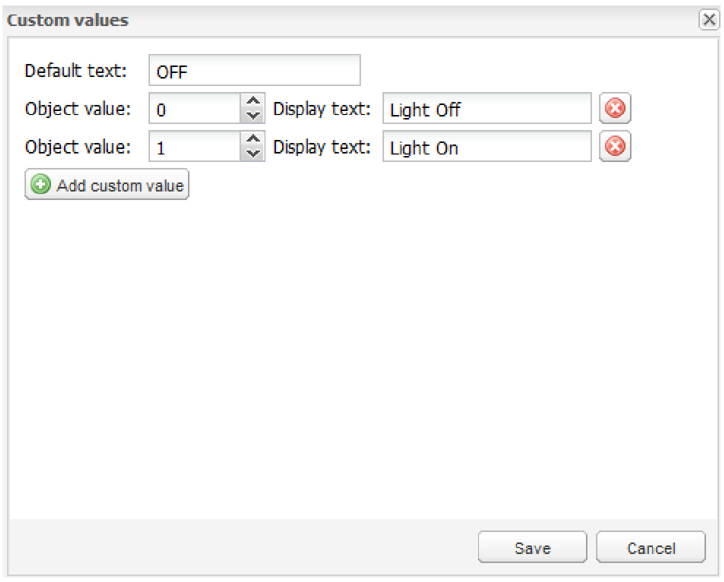
Step:

2.2.4 Custom text value

In the object list, by pressing on the button, custom text can be added to the object values.
Custom text values can be set only to Boolean and integer values.

Default text – Text which will be displayed if value is not defined

Object value – *Add custom value*, select *Object value* and define *Display text*



Custom values

Default text: OFF

Object value: 0 Display text: Light Off

Object value: 1 Display text: Light On

2.2.5 Object control bar

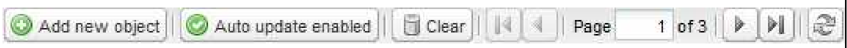
Add new object – Manually add new object to the list

Auto update enabled –Specifies either the object list is updated automatically or not

Clear – Clear the list of group addresses

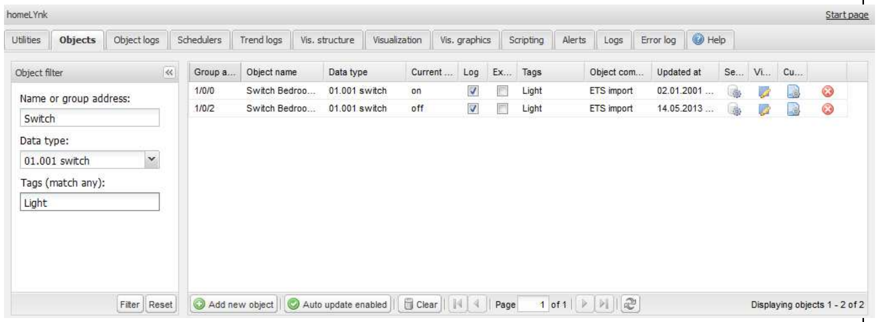
Next/Previous page – Move to next or previous page

Refresh – Refresh the object list



2.2.6 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.



Object filter

Name or group address: Switch

Data type: 01.001 switch

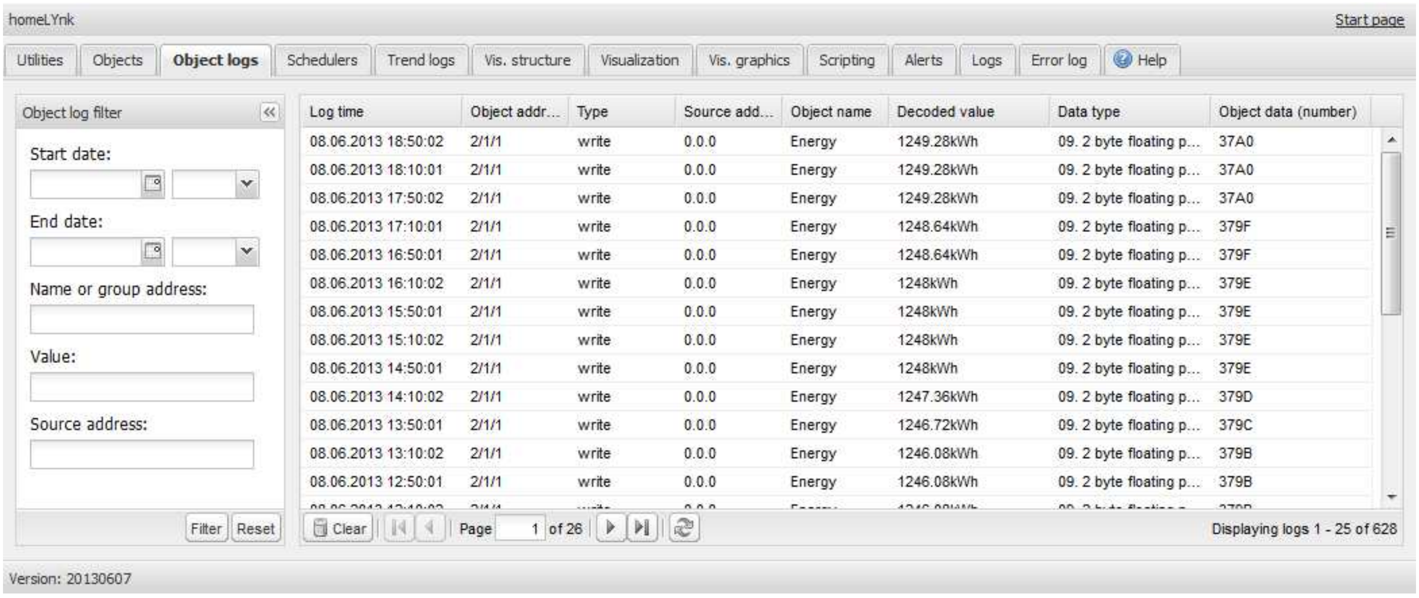
Tags (match any): Light

Group a...	Object name	Data type	Current ...	Log	Ex...	Tags	Object com...	Updated at	Se...	Vi...	Cu...	
1/0/0	Switch Bedroo...	01.001 switch	on	✓	☐	Light	ETS import	02.01.2001 ...				
1/0/2	Switch Bedroo...	01.001 switch	off	✓	☐	Light	ETS import	14.05.2013 ...				

Displaying objects 1 - 2 of 2

2.3 Object Logs

Object historical telegrams are available in *Object logs*. Once logging is enabled for object, all the historical and future data will be logged in.



homeLYnk | Start page

Object log filter

Start date:

End date:

Name or group address:

Value:

Source address:

Log time	Object addr...	Type	Source add...	Object name	Decoded value	Data type	Object data (number)
08.06.2013 18:50:02	2/1/1	write	0.0.0	Energy	1249.28kWh	09. 2 byte floating p...	37A0
08.06.2013 18:10:01	2/1/1	write	0.0.0	Energy	1249.28kWh	09. 2 byte floating p...	37A0
08.06.2013 17:50:02	2/1/1	write	0.0.0	Energy	1249.28kWh	09. 2 byte floating p...	37A0
08.06.2013 17:10:01	2/1/1	write	0.0.0	Energy	1248.64kWh	09. 2 byte floating p...	379F
08.06.2013 16:50:01	2/1/1	write	0.0.0	Energy	1248.64kWh	09. 2 byte floating p...	379F
08.06.2013 16:10:02	2/1/1	write	0.0.0	Energy	1248kWh	09. 2 byte floating p...	379E
08.06.2013 15:50:01	2/1/1	write	0.0.0	Energy	1248kWh	09. 2 byte floating p...	379E
08.06.2013 15:10:02	2/1/1	write	0.0.0	Energy	1248kWh	09. 2 byte floating p...	379E
08.06.2013 14:50:01	2/1/1	write	0.0.0	Energy	1248kWh	09. 2 byte floating p...	379E
08.06.2013 14:10:02	2/1/1	write	0.0.0	Energy	1247.36kWh	09. 2 byte floating p...	379D
08.06.2013 13:50:01	2/1/1	write	0.0.0	Energy	1246.72kWh	09. 2 byte floating p...	379C
08.06.2013 13:10:02	2/1/1	write	0.0.0	Energy	1246.08kWh	09. 2 byte floating p...	379B
08.06.2013 12:50:01	2/1/1	write	0.0.0	Energy	1246.08kWh	09. 2 byte floating p...	379B

Displaying logs 1 - 25 of 628

Version: 20130607

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

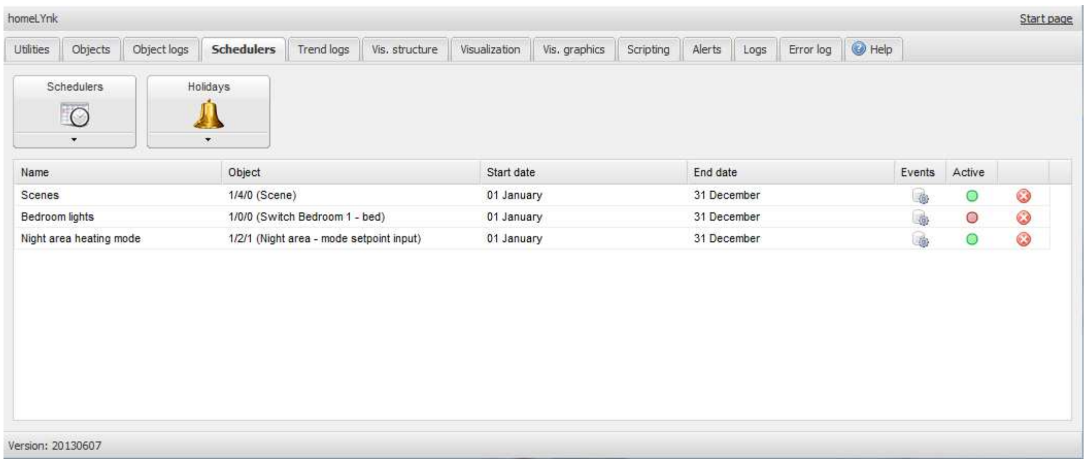
Source address – specific source address

All logs can be cleared by pressing on *Clear* button.

Note: Logging memory is set up in the *Utilities → Configurations*

2.4 Schedulers

Schedulers allow the end user to control KNX group address values based on the date or day of the week.



Name	Object	Start date	End date	Events	Active
Scenes	1/4/0 (Scene)	01 January	31 December		🟢
Bedroom lights	1/0/0 (Switch Bedroom 1 - bed)	01 January	31 December		🔴
Night area heating mode	1/2/1 (Night area - mode setpoint input)	01 January	31 December		🟢

Version: 20130607

2.4.1 Add new scheduler

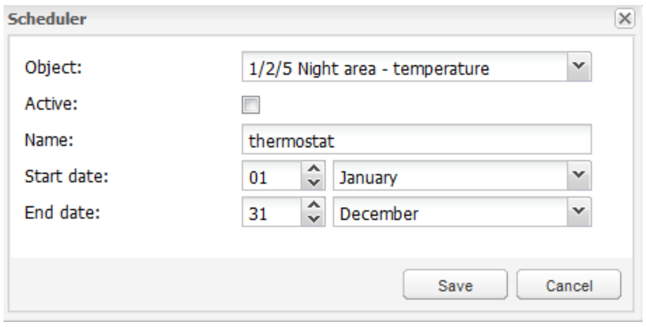
Object – The object group address which will be controlled by the scheduler

Active – Define whether a scheduler is active or not

Name – Name of the scheduler

Start date – Start date of the scheduler

End date – End date of the scheduler



Object:	1/2/5 Night area - temperature
Active:	☐
Name:	thermostat
Start date:	01 January
End date:	31 December

2.4.2 Scheduler events

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

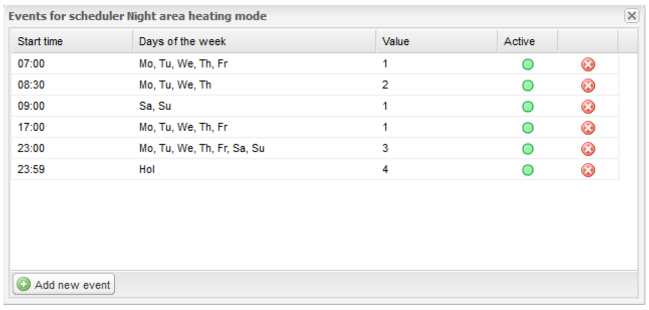
Active – Define the event to be active or not

Value – Value to send to the group address when the event will be triggered

Start time – Start time for the event

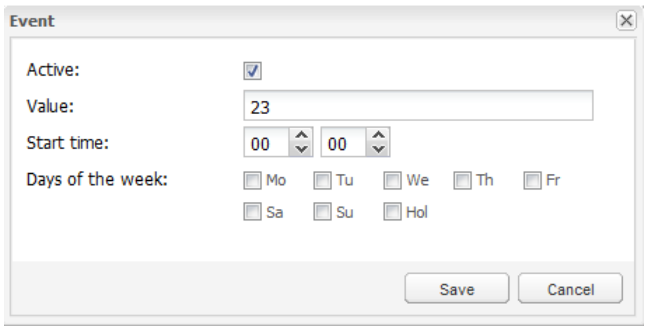
Days of the week – Days of the week when the event will be triggered.

Hol – Holidays which are defined in *Holidays* tab



Events for scheduler Night area heating mode

Start time	Days of the week	Value	Active	
07:00	Mo, Tu, We, Th, Fr	1	●	⊗
08:30	Mo, Tu, We, Th	2	●	⊗
09:00	Sa, Su	1	●	⊗
17:00	Mo, Tu, We, Th, Fr	1	●	⊗
23:00	Mo, Tu, We, Th, Fr, Sa, Su	3	●	⊗
23:59	Hol	4	●	⊗

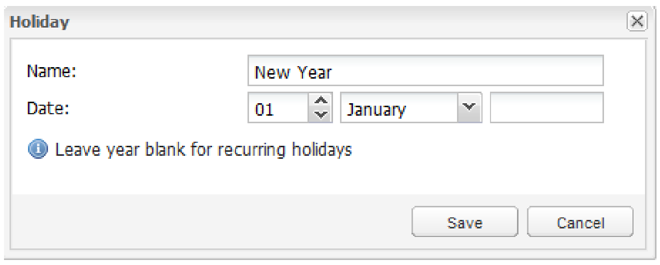


Event

Active:	✓
Value:	23
Start time:	00 00
Days of the week:	☐ Mo ☐ Tu ☐ We ☐ Th ☐ Fr ☐ Sa ☐ Su ☐ Hol

2.4.3 Scheduler holidays

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



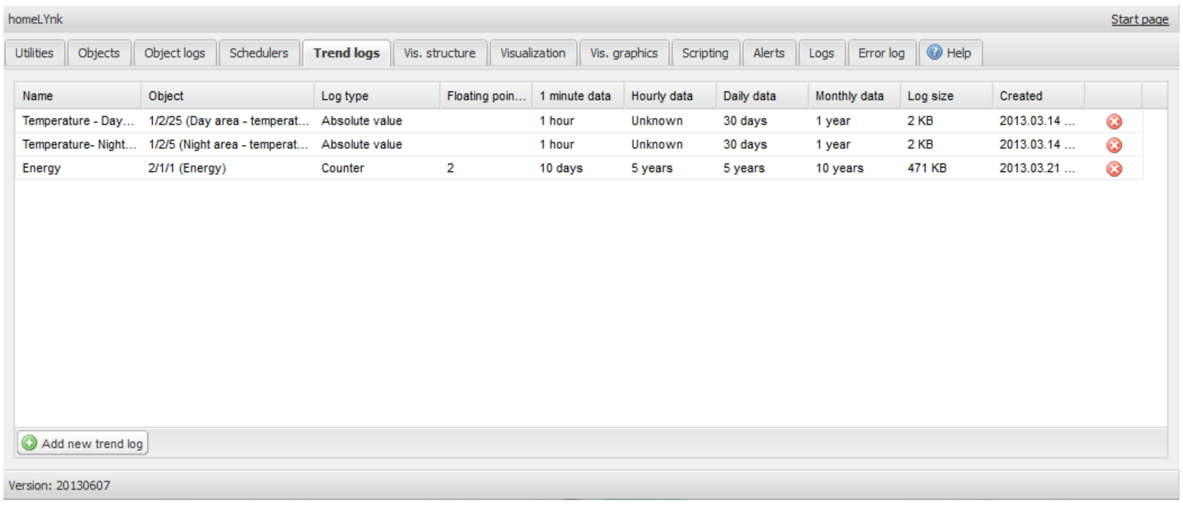
Holiday

Name:	New Year
Date:	01 January

Leave year blank for recurring holidays

2.5 Trend logs

Trends logs or so called data logging allows the end user to store selected data and compare different the time periods from the past.



homeLYnk — Start page

Utilities | Objects | Object logs | Schedulers | Trend logs | Vis. structure | Visualization | Vis. graphics | Scripting | Alerts | Logs | Error log | Help

Name	Object	Log type	Floating poin...	1 minute data	Hourly data	Daily data	Monthly data	Log size	Created
Temperature - Day...	1/2/25 (Day area - temperat...	Absolute value		1 hour	Unknown	30 days	1 year	2 KB	2013.03.14 ...
Temperature- Night...	1/2/5 (Night area - temperat...	Absolute value		1 hour	Unknown	30 days	1 year	2 KB	2013.03.14 ...
Energy	2/1/1 (Energy)	Counter	2	10 days	5 years	5 years	10 years	471 KB	2013.03.21 ...

Add new trend log

Version: 20130607

2.5.1 Add new trend log

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

Name – Name of the trend

Log type – Type of the log.
Counter type is used to count the date
Absolute value – saves the actual readings

Floating point precision – If the object is floating type, then precision needs to be selected. Example 1.1111 = precision is 4

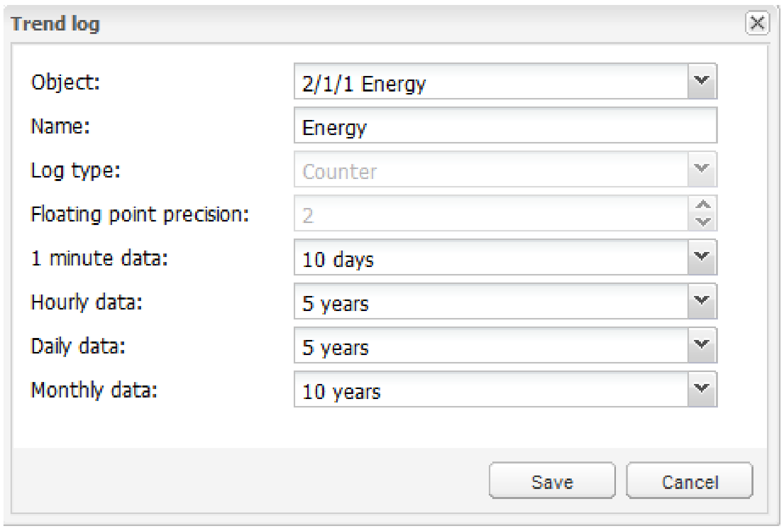
1 minute data – Average value of 1 minute for specific time interval data will be shown on the trend. E.g. if 1 hour – trend step will be 1 hour with average 60 readings data

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

Log size – Define total trend log size of the object. Trend logs are stored on 3.2 GB internal flash memory



Trend log

Object:	2/1/1 Energy
Name:	Energy
Log type:	Counter
Floating point precision:	2
1 minute data:	10 days
Hourly data:	5 years
Daily data:	5 years
Monthly data:	10 years

Save | Cancel

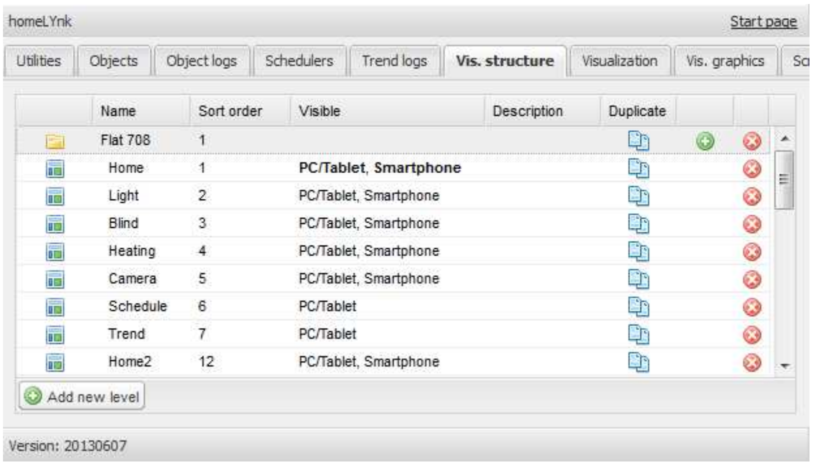
2.6 Vis. structure

Vis. Structure is used for creating all buildings levels and visualizations plans. Additionally it can create Layouts and Widgets for the plans visualization.

Starting new projects, only **Layouts** and **Widgets** folders are visible. Adding new level allows the end user to define specific **Plan** of the flat. **Layouts** and **Widgets** are additional tools which are not mandatory for basic visualizations; they can be defined and implemented in other **Plans**.

2.6.1 Levels

To add new Level press on Add new level. Main level usually is the project name. Additional levels can be added later.



Name	Sort order	Visible	Description	Duplicate
Flat 708	1			
Home	1	**PC/Tablet, Smartphone**		
Light	2	PC/Tablet, Smartphone		
Blind	3	PC/Tablet, Smartphone		
Heating	4	PC/Tablet, Smartphone		
Camera	5	PC/Tablet, Smartphone		
Schedule	6	PC/Tablet		
Trend	7	PC/Tablet		
Home2	12	PC/Tablet, Smartphone		

Version: 20130607

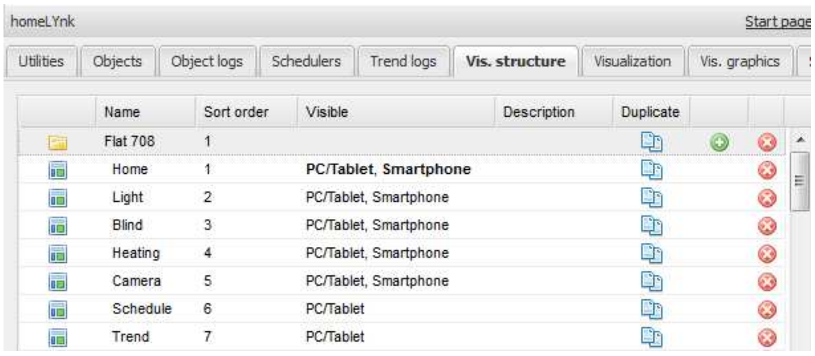
2.6.2 Second level

Second level could be used in buildings with many floors.

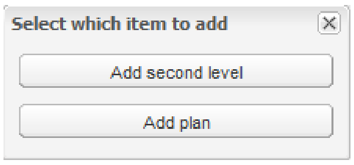
If you need additional levels press on next to your main level.

Select Add second level and give it a name and sort order.

Each level can be duplicated together with sublevels and plans via pressing duplicate icon next to the level.



Name	Sort order	Visible	Description	Duplicate
Flat 708	1			
Home	1	**PC/Tablet, Smartphone**		
Light	2	PC/Tablet, Smartphone		
Blind	3	PC/Tablet, Smartphone		
Heating	4	PC/Tablet, Smartphone		
Camera	5	PC/Tablet, Smartphone		
Schedule	6	PC/Tablet		
Trend	7	PC/Tablet		



2.6.3 Plans

A Plan could be either one room from flat or one function (as lighting or heating) for the whole flat.
To add Plans press on next to a level under which the plan is to be added and select Add plan.

Name – Name for the plan.

Layout – Layout for this specific plan. All objects from Layout will be duplicated on this particular plan including background colour and plan image if they are not defined separately for this specific plan. Layout need to be created before being added to the Plan.

PC/Tablet visualization
[Show, Show and make default, Hide] –Visibility for this particular plan in the PC/Tablet visualization

Smartphone visualization
[Show, Show and make default, Hide] – Visibility for this particular plan in the Smartphone visualization
Background image – Select background previously added to Vis. graphics -> Images/Backgrounds

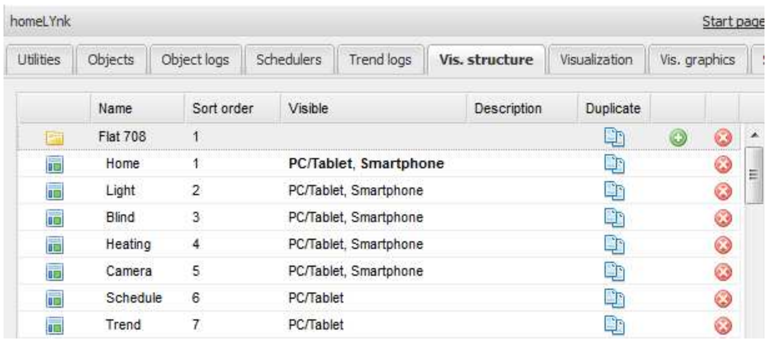
Background color – Choose background color of the plan.

Repeat background image – Either to show the image once or repeat it and fill the whole plan.

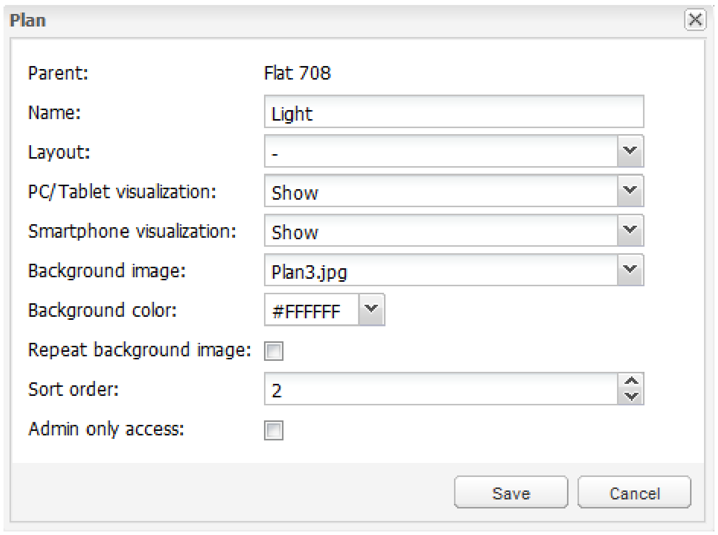
Sort order – Sort order for the plan, depends where this particular plan will be located among other in a specific level.

Admin only access – Enable admin only access for this floor (visadmin user)
Each Plan can be duplicated together with all components on a plan via pressing duplicate icon next to the plan

Content of this Plan is to be defined under Visualization tab



Name	Sort order	Visible	Description	Duplicate
Flat 708	1			
Home	1	PC/Tablet, Smartphone		
Light	2	PC/Tablet, Smartphone		
Blind	3	PC/Tablet, Smartphone		
Heating	4	PC/Tablet, Smartphone		
Camera	5	PC/Tablet, Smartphone		
Schedule	6	PC/Tablet		
Trend	7	PC/Tablet		



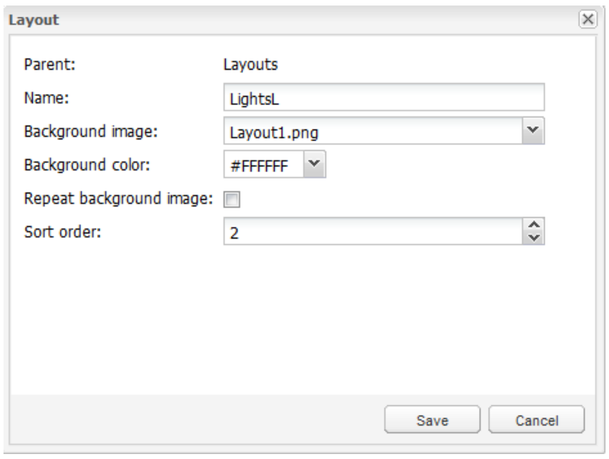
Parent:	Flat 708
Name:	Light
Layout:	-
PC/Tablet visualization:	Show
Smartphone visualization:	Show
Background image:	Plan3.jpg
Background color:	#FFFFFF
Repeat background image:	☐
Sort order:	2
Admin only access:	☐

2.6.4 Layout

To add Layout pres on next to a Layout folder.

Each Layout can be duplicated together with all components via pressing duplicate icon next to the Layout .

Content of this layout is to be defined under Visualization tab.



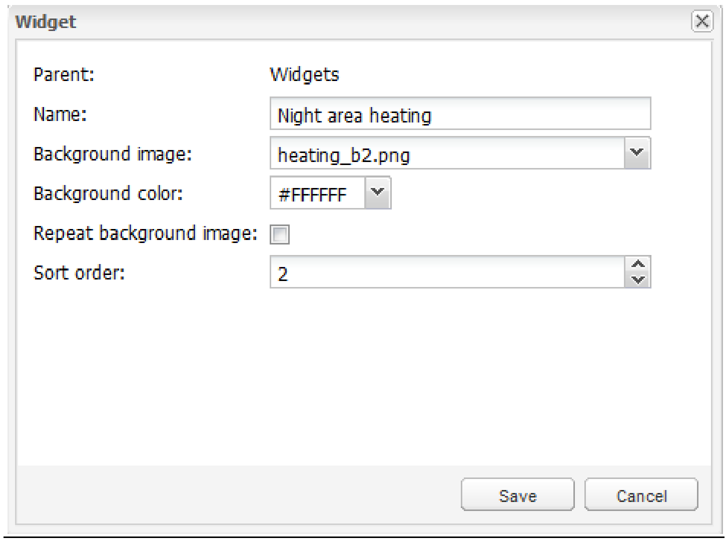
Parent:	Layouts
Name:	LightsL
Background image:	Layout1.png
Background color:	#FFFFFF
Repeat background image:	☐
Sort order:	2

2.6.5 Widgets

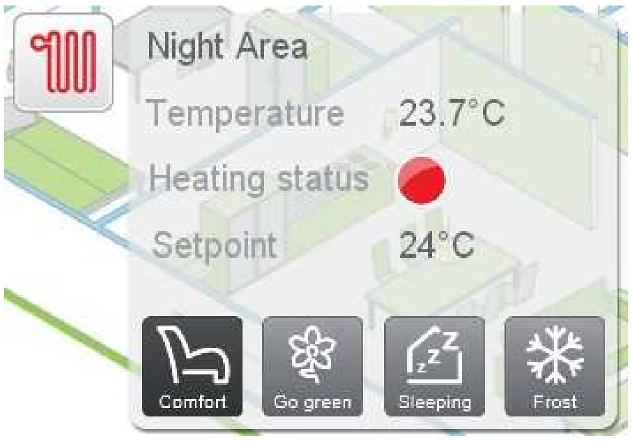
Widget is a small popup web page which can be attached to a button.

To add widgets press on next to widgets folder. Each widget can be duplicated together with all components via pressing duplicate icon next to the widget . Content of this widget is to be defined under Visualization tab.

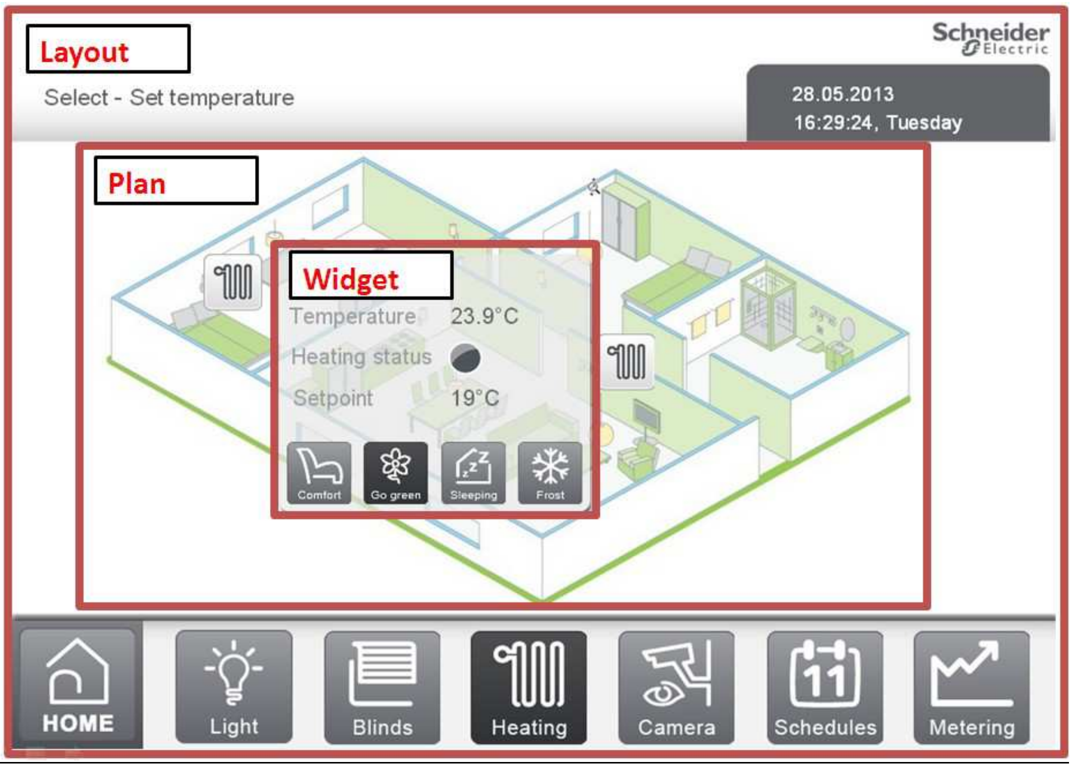
Note! Widget size always has to be smaller than plan on which it is placed on.



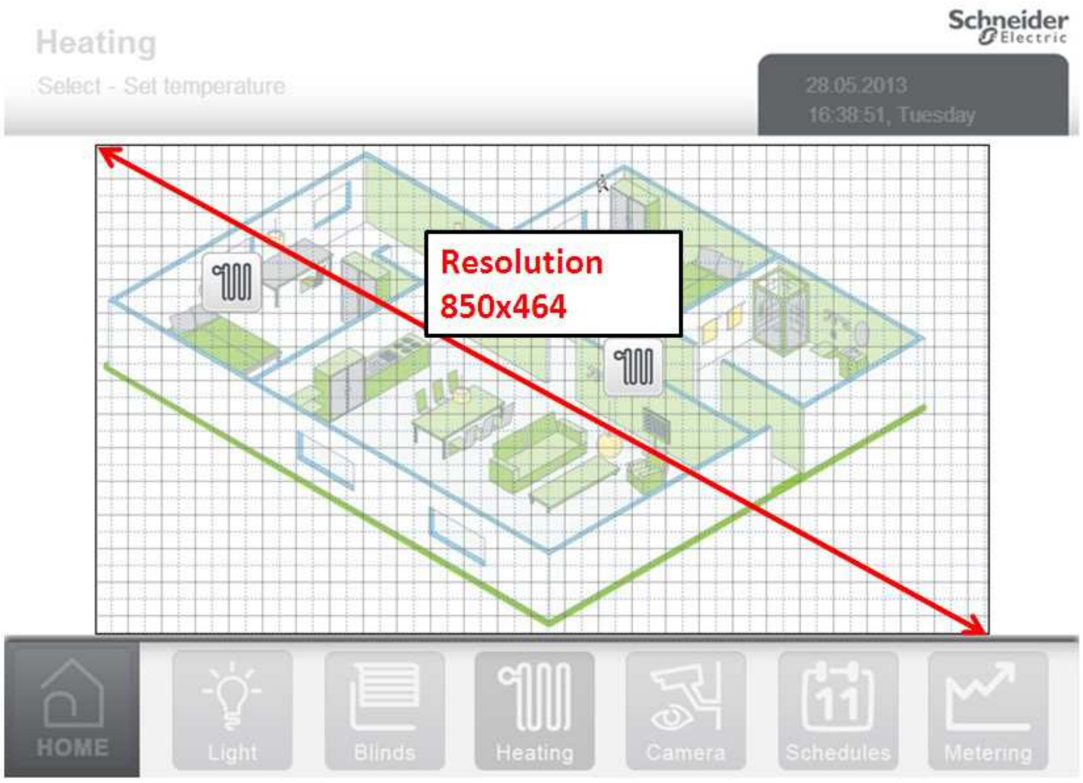
Parent:	Widgets
Name:	Night area heating
Background image:	heating_b2.png
Background color:	#FFFFFF
Repeat background image:	☐
Sort order:	2



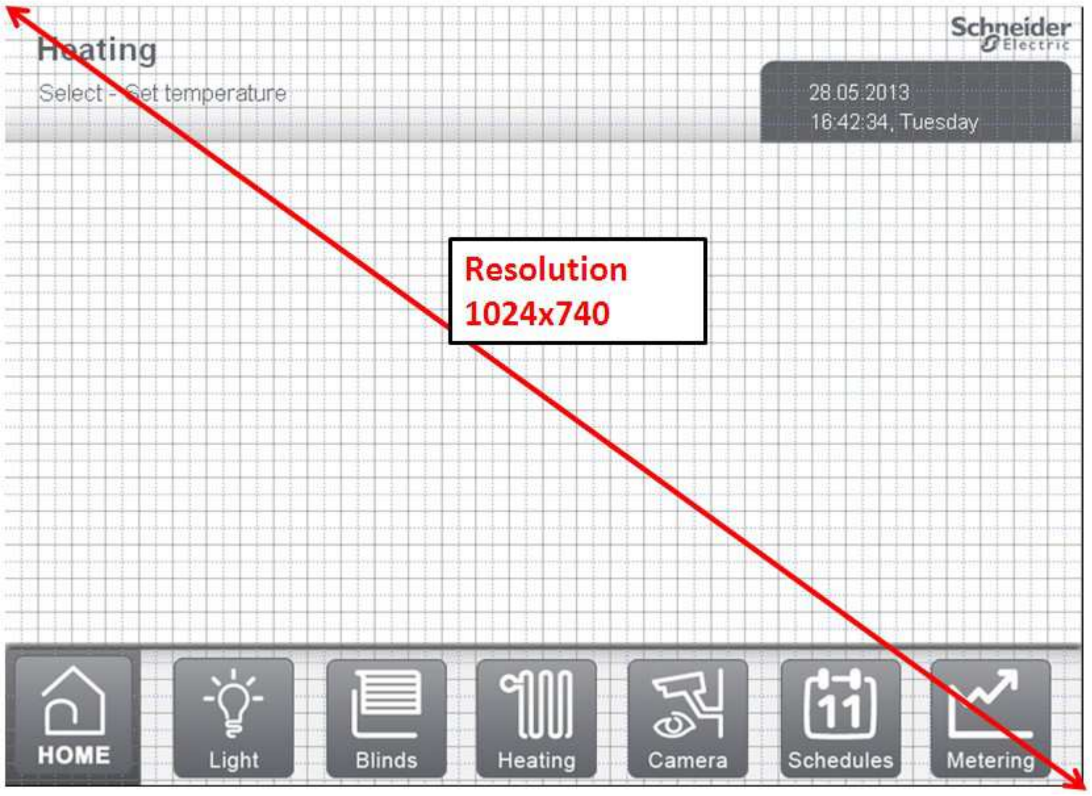
2.6.6 Visualization structure example



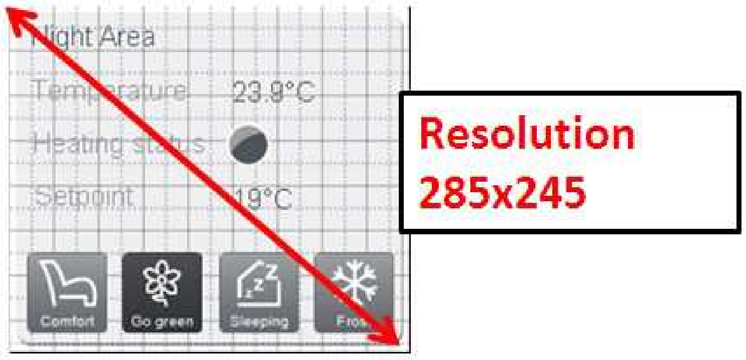
2.6.7 Plan



2.6.8 Layout



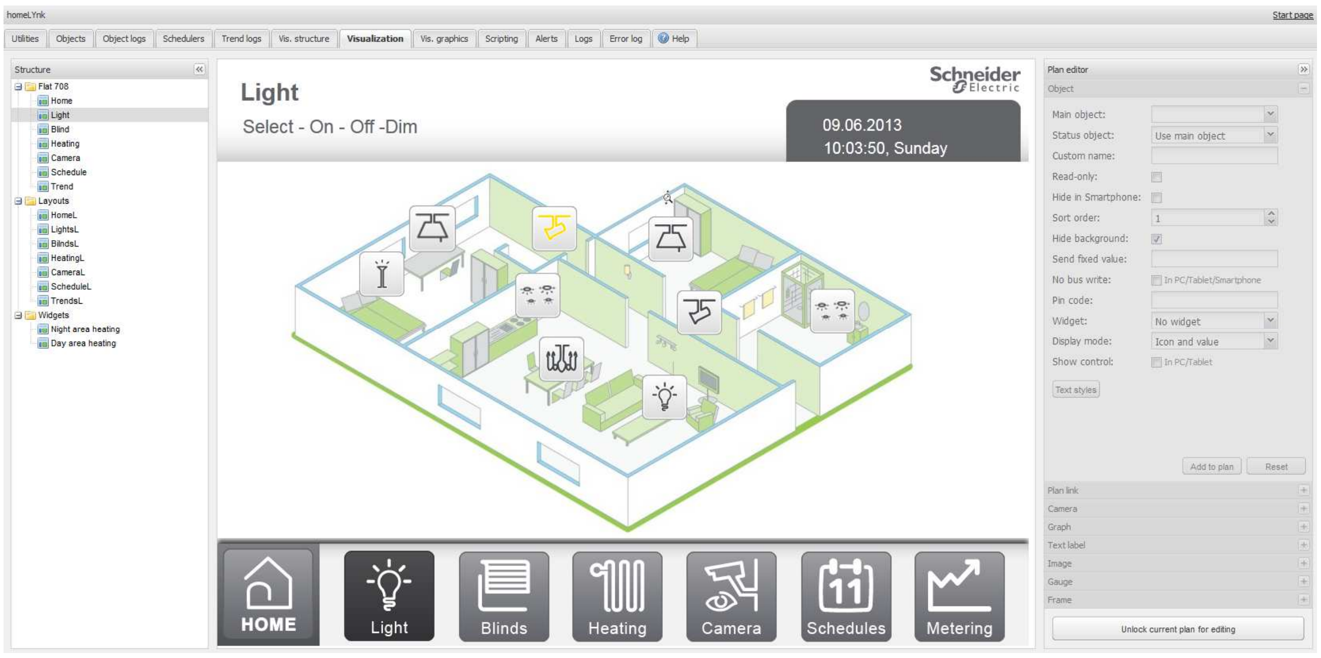
2.6.9 Widget



2.7 Visualization

This tab is split in a tree section:

- **Structure** – Navigation tree for levels, plans, widgets which were created under visualization structure tab.
- **Visualization map** – Actual visualization field where you can add all visualization components
- **Plan Editor** – all parameters of the component are set up here.



Main object:	
Status object:	Use main object
Custom name:	
Read-only:	☐
Hide in Smartphone:	☐
Sort order:	1
Hide background:	☑
Send fixed value:	
No bus write:	☐ In PC/Tablet/Smartphone
Pin code:	
Widget:	No widget
Display mode:	Icon and value
Show control:	☐ In PC/Tablet

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

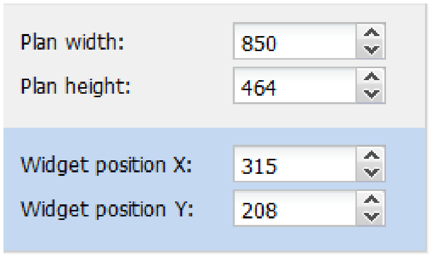
2.7.1 Structure

To navigate between plans, layouts and widgets using navigation tree in structure view.

During editing mode in bottom the below additional parameters are available

➔ Size of plans, layouts and widgets.

➔ Position of each component is also displayed here



Plan width:	850
Plan height:	464
Widget position X:	315
Widget position Y:	208

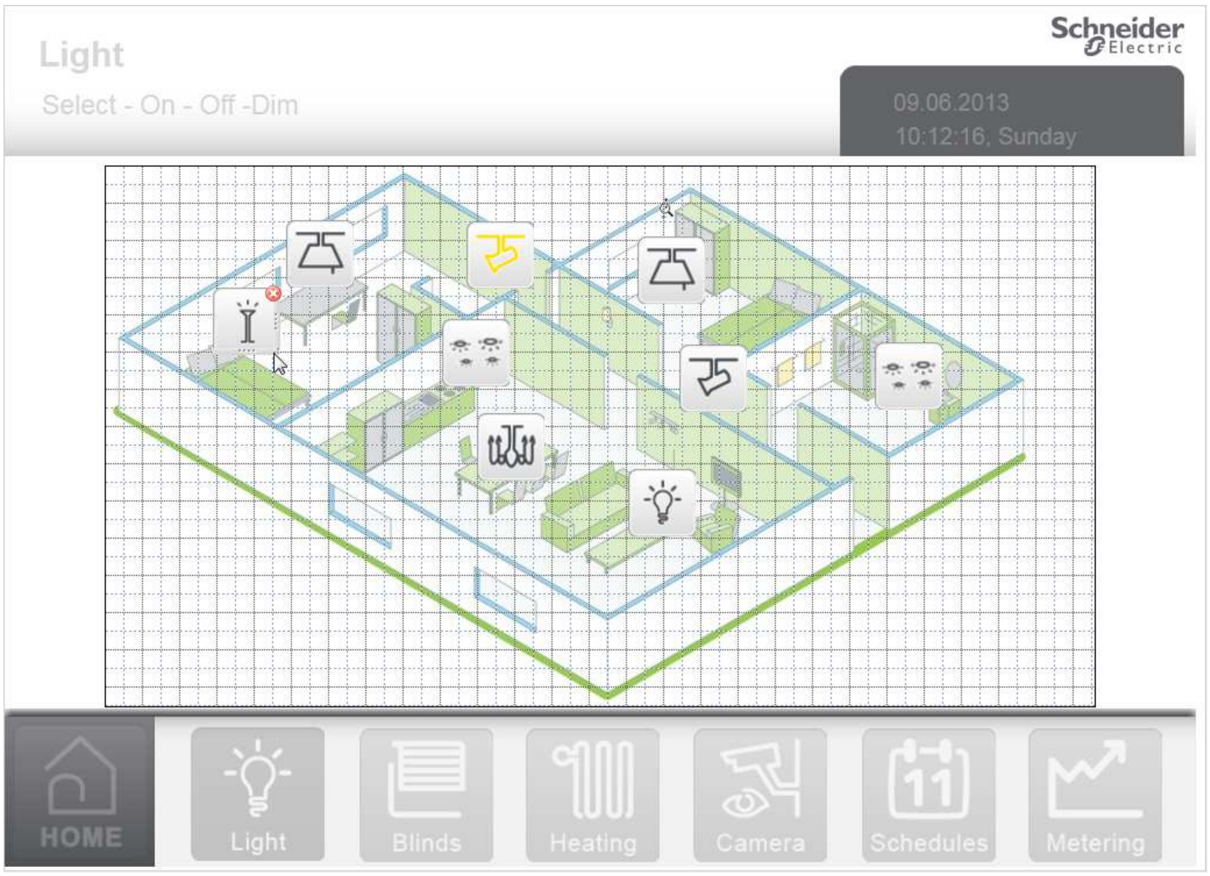
Note: Size of the plan should be positioned correctly against the background.

Widget size always has to be smaller than plan on which it is placed on.

Always use component position to align objects.

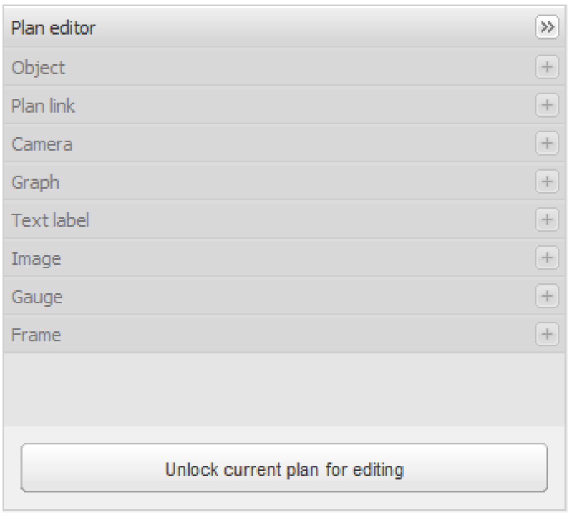
2.7.2 Visualization Map

Each added object will be placed in top left corner of plan. Move every new object to the correct position via dragging it. To delete object select it and press ⊗ which is always positioned in top right corner of every object.



2.7.3 Plan Editor

Plan editor is located on the right side of the visualization map. Editing mode can be accessed by pressing *Unlock current plan for editing*.



2.7.4 Object

Every control or monitoring objects is configured under this tab. Different data tapes have different parameters.

Main object – List of existing group addresses on KNX/EIB bus, the ones available for configuration in *Objects* tab. In order to speed up selection it is recommended to start writing group address.

Status object – List of status objects on KNX/EIB bus. Control object can also be used as status.

Custom name – Name for the object. Custom name is important for Smartphone Visualization; if the name is left blank group address name is used instead.

Read-only – The object is read-only, no write permission.

Hide in Smartphone– Do not show this object in *Smartphone Visualization.*

Sort order– Sort number for Smartphone visualization.

Hide background– Hide icon background.

Send fixed value– Allows sending specific value to the bus each time the object is pressed.

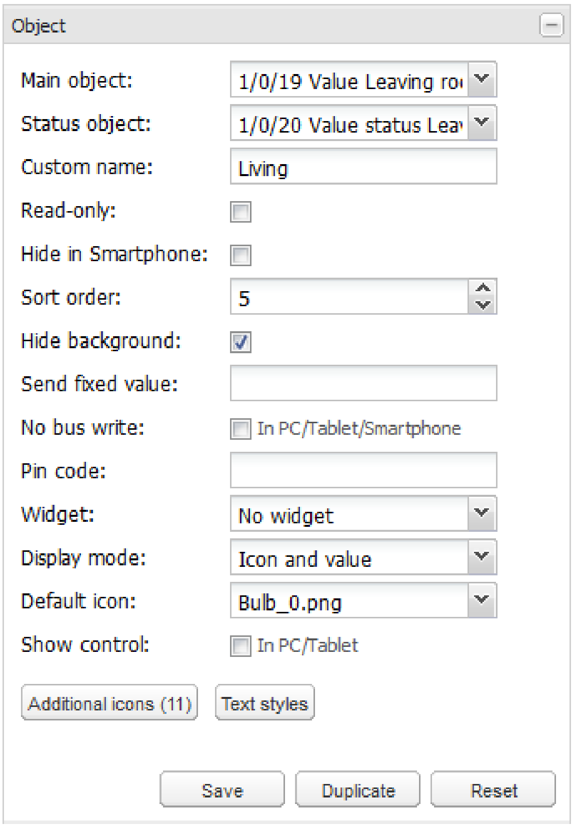
No bus write – Value would not be written in to KNX bus. Use full for triggering scripts with bus load limitation.

Pin code – Via adding a pin you can protect object. Each time value will be change pin code will be requested to write.

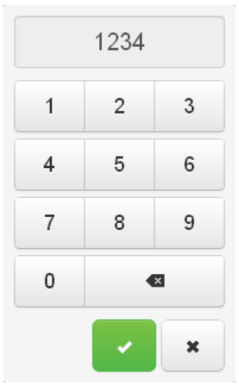
Widget –Widget can be attached to a button which needs to be created before. Widget cannot be tested under editor mode; only under PC/Tablet Visualization can it be tested.

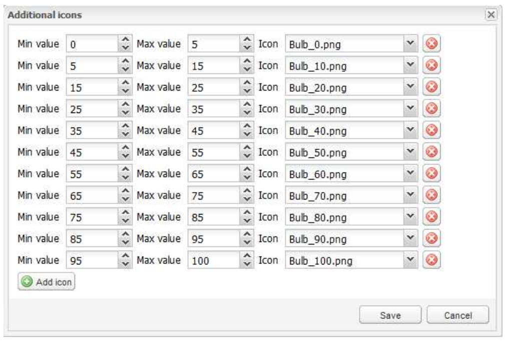
Display mode ***[icon and value; icon; value]*** – how to display the object

Default Icon– Default icon of scale-type objects


Object

Main object:	1/0/19 Value Leaving ro
Status object:	1/0/20 Value status Lea
Custom name:	Living
Read-only:	☐
Hide in Smartphone:	☐
Sort order:	5
Hide background:	☑
Send fixed value:	
No bus write:	☐ In PC/Tablet/Smartphone
Pin code:	
Widget:	No widget
Display mode:	Icon and value
Default icon:	Bulb_0.png
Show control:	☐ In PC/Tablet




Additional icons

Min value	0	Max value	5	Icon	Bulb_0.png
Min value	5	Max value	15	Icon	Bulb_10.png
Min value	15	Max value	25	Icon	Bulb_20.png
Min value	25	Max value	35	Icon	Bulb_30.png
Min value	35	Max value	45	Icon	Bulb_40.png
Min value	45	Max value	55	Icon	Bulb_50.png
Min value	55	Max value	65	Icon	Bulb_60.png
Min value	65	Max value	75	Icon	Bulb_70.png
Min value	75	Max value	85	Icon	Bulb_80.png
Min value	85	Max value	95	Icon	Bulb_90.png
Min value	95	Max value	100	Icon	Bulb_100.png

On icon – On state icon for binary-type objects

Off icon – Off state icon for binary-type objects

Show control – If enabled any control button will be always open. Visible only in PC/Tablet Visualization

For value-type objects, additional button would appear while specifying parameters – *Additional icons.* Different icons for different object values can be defined in the window.

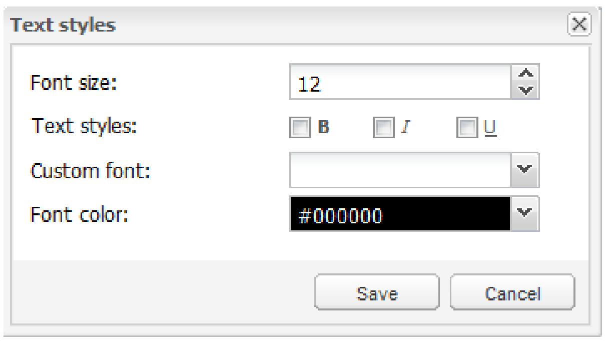
For value display text style can be defined

Once the object parameters are defined, press *Add to floor plan* button and a newly created object would appear. The object can be moved to the any location of the plan. Note that while being in editing mode, the object will not work. When all necessary objects are added, press *Save and reload floor plan* button so that the objects starts functioning.

You can edit Each added object can be edited while clicking on it in the Editing mode. Press the Save button Save after each change.

Each object can be duplicated via pressing duplicate button. Duplicate

Reset button well set object parameters to default settings. Reset



Text styles

Font size:	12
Text styles:	☐ B ☐ I ☐ U
Custom font:	
Font color:	#000000

2.7.5 Plan link

In order to make visualization more convenient, there are plan links integrated. Special icons on the map can be added which would act as a link to other plans.

Plan – Select plan link

Custom name –Name for the link

Hide background– Hide icon background

Icon – Icon which will be showed in visualization. If only text is selected, text parameters are selected.

Active state icon – If icon is selected active plan icon is available.

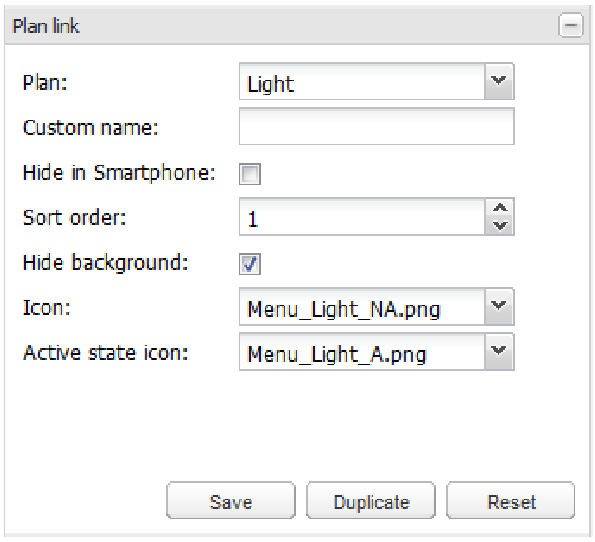
Font size – Size of font

Text style – Text style – bold, italic, underscore

Custom font – Font name

Font color – Font color

Note It is recommended to use Layout for menu and plan link creation. You can save time while adding it to different plans and later when making changes. By adding it to different plans it would save time and be beneficial when changes are required.



Plan link

Plan:	Light
Custom name:	
Hide in Smartphone:	☐
Sort order:	1
Hide background:	✓
Icon:	Menu_Light_NA.png
Active state icon:	Menu_Light_A.png

2.7.6 Camera

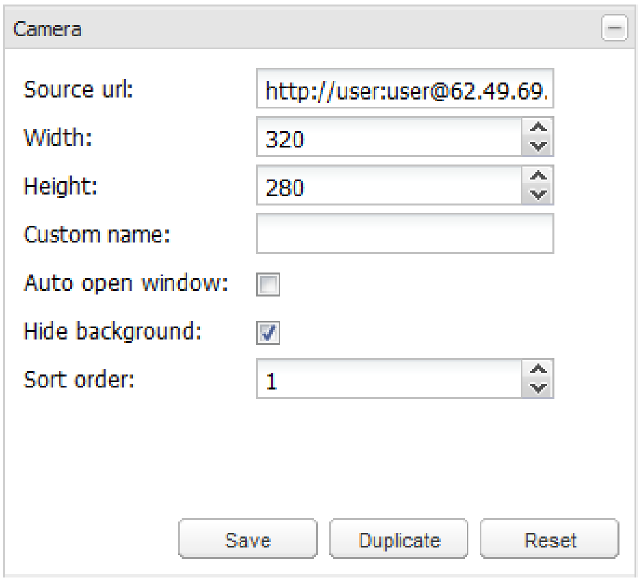
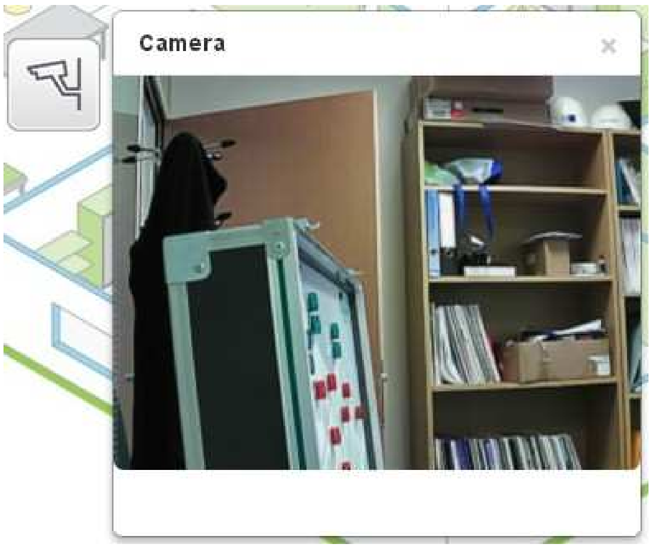
HomeLYnk supports third party IP web camera integration into its visualization.

Note Only cameras which support HTTP MJPEG streaming in web browser.

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. ***Auto open window*** – automatically open video window. ***Hide background***– Hide icon background. ***Sort order*** – Order cameras for touch visualization **Note** • If IP camera requires user name and password, enter the url in form ***http://USER:PASSWORD@IP*** • HomeLYnk is only redirecting stream from camera to the browser. If stream does not work it is web browser issue not HomeLYnk. • If there is cameras issue please check if video stream is available in browser • If camera wants to be available from external, IP of camera need to be port forwarded trough the router. While adding external camera, IP with correct port has to be used (IP:port). If local IP is used then camera would not be available from external • Contact Technical support of camera manufacturer if direct video stream is hidden by the manufacturer. • Camera image (*.png only!) can be changed via replacing camera image under Vis. graphics tab. Name of image has to be 'camera'.	 Camera Source url: http://user:user@62.49.69. Width: 320 Height: 280 Custom name: Auto open window: ☐ Hide background: ☑ Sort order: 1 

2.7.7 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.

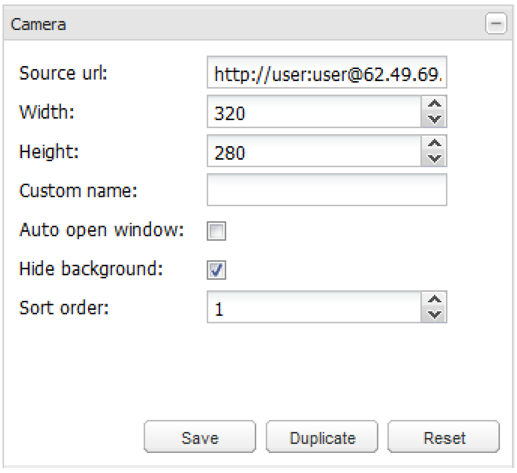
Height– Sub-window height for displaying the graph

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. The object can be moved to the desired location. Note that while being in editing mode, the object will not work. Press on *Save and reload plan* button so that the objects starts functioning.


Camera

Source url:	http://user:user@62.49.69.
Width:	320
Height:	280
Custom name:	
Auto open window:	☐
Hide background:	☑
Sort order:	1

2.7.8 Text label

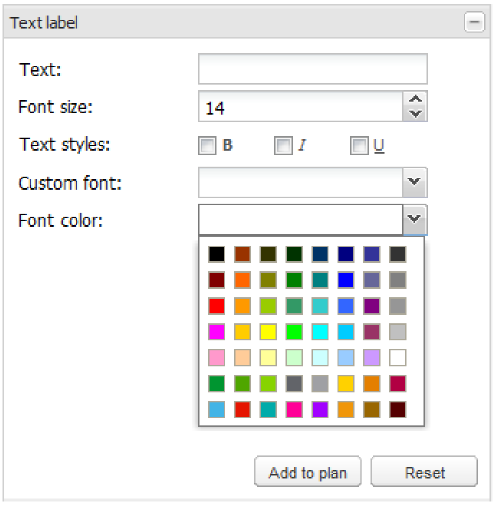
Text labels can be added and moved across the visualization map.

Text – Label text

Font size – Label font size

Text style – style of the text – bold, italic, underscored.

Custom font – font name.


Text label

Text:	
Font size:	14
Text styles:	☐ B ☐ I ☐ U
Custom font:	
Font color:	

Font color– label font color Once the label parameters are defined, press *Add new object* button and newly created object will appear on the map. The object can be moved to the desired location. Press on *Save and reload floor plan* button so the objects starts functioning. Last two rows in the color palette refer to the predefined Schneider Electric corporate colors.	

2.7.9 Image

Image section allows adding images from Local storage or from the internet into the visualization map. External image is useful for example, to grab dynamic weather cast images.

***Image source* [Local, Remote] –** Select image source

***Select image* –** Select image previously added to Vis. graphics -> Images/Backgrounds

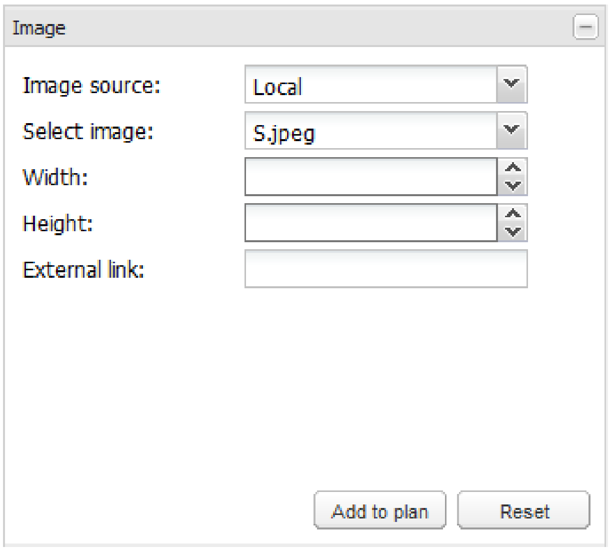
Image url – Source URL of the image

Width – Width of the image

Height – Height of the image

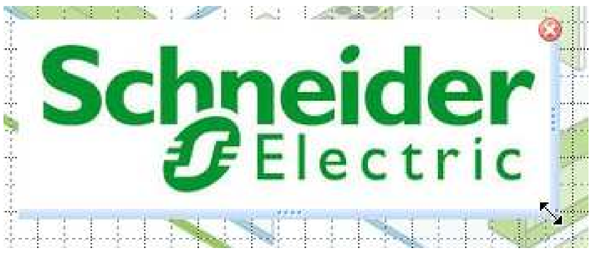
External link – External link URL when pressing on the image

Once the image parameters are defined, press *Add to plan* button and newly created object will appear on the map. The object can be moved to the desired location. Press on *Save and reload plan* button so the objects starts functioning. Image can be freely resized via catching edge of image and move.



Image

Image source:	Local
Select image:	S.jpeg
Width:	
Height:	
External link:	



2.7.10 Gauge

Gauge allows dynamic way of visualization and changing object value in the gauge.

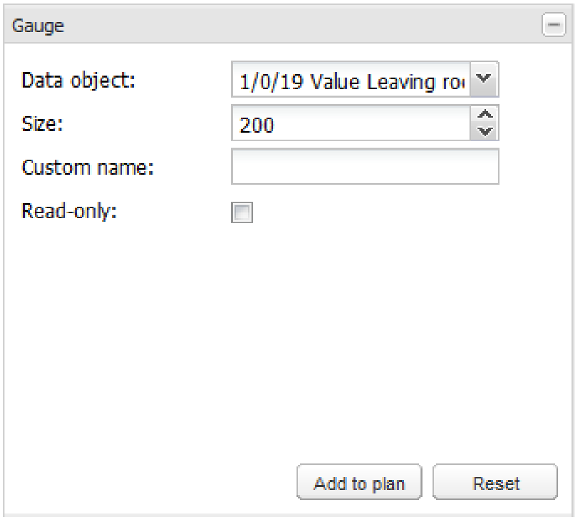
Data object – KNX group address

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. The object can be moved to the desired location. Press on *Save and reload plan* button so that the objects starts functioning.


Gauge

Data object:	1/0/19 Value Leaving ro
Size:	200
Custom name:	
Read-only:	☐

2.7.11 Frame

Frame allows displaying internal or external webpage in visualization. *Schedulers* and *Trends* are integrated in to the frame.

Source – Select Scheduler, Trend log or external URL
Url: - Source URL of external webpage

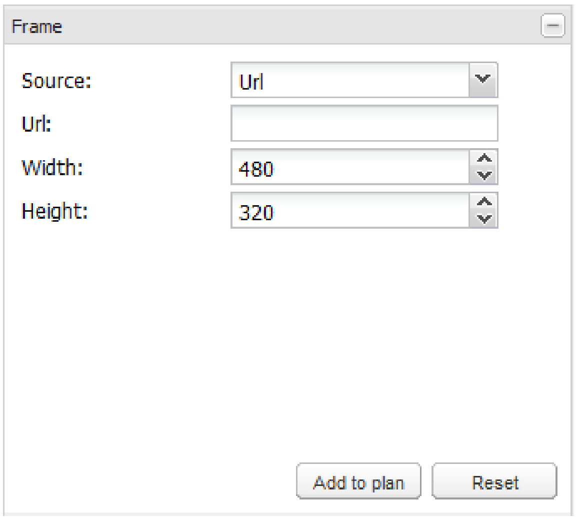
Width – width of frame

Height – height of frame

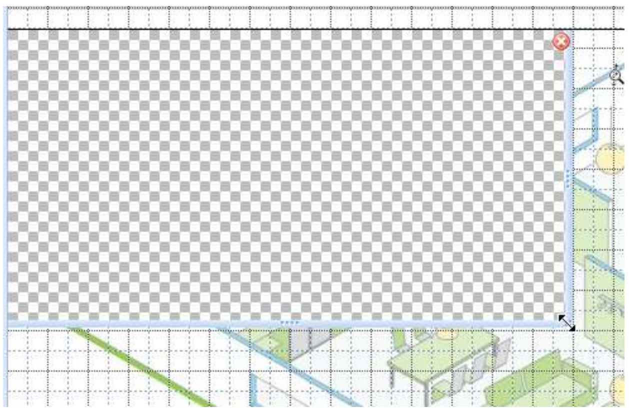
Once the Frame parameters are defined, press *Add to plan* button and newly created object will appear on the map. The object can be moved to the desired location. Press on *Save and reload plan* button so the objects starts functioning. Frame can be freely resized via catching edge of Frame and move.

Note

- Some web pages have java script which prevent from using frame, if this is implemented webpage will open in full screen rather in frame
- It is recommended to stretch the frame to maximum width if Scheduler or Trend is used. Recommended minimum width is 1024.
- Frame is only visible under PC/Tablet Visualization.
- Do not allow Scheduler or Trend to be viewed from Smartphone visualization. Settings are available in Vis. structure under dedicated plan.

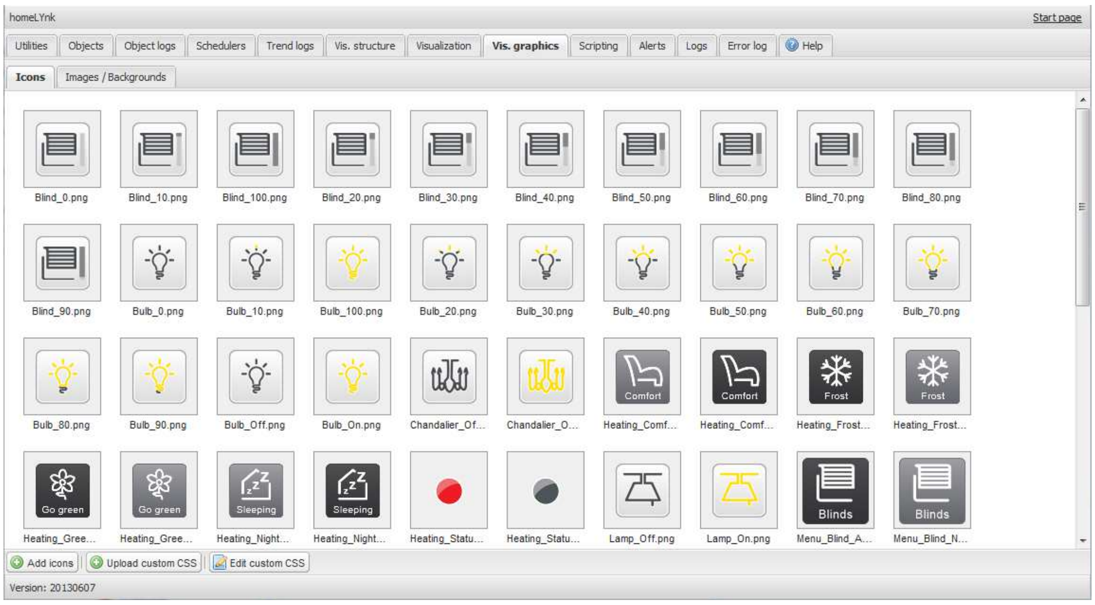

Frame

Source:	Url
Url:	
Width:	480
Height:	320



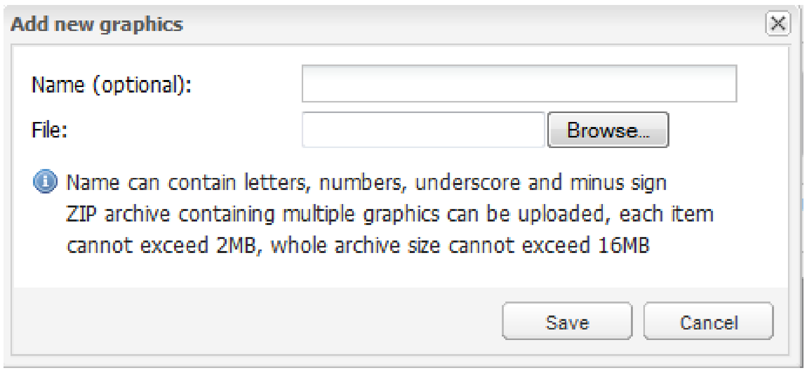
2.8 Vis. graphics

This tab is split into two sections, icons where all object icons are located and Images/Backgrounds.



Press on *Add new icon* button to add a new entry. The system accepts any size of icon.

Jpeg, Gif and PNG formats are supported. Name can contain letters, numbers, underscore and minus sign
ZIP archive containing multiple graphics can be uploaded, each item cannot exceed 2MB, and whole archive size cannot exceed 16MB.


Add new graphics

Name (optional):

File: Browse...

Name can contain letters, numbers, underscore and minus sign
ZIP archive containing multiple graphics can be uploaded, each item cannot exceed 2MB, whole archive size cannot exceed 16MB

Save Cancel

Name (optional) – The name of the icon. It will appear in the list when adding new object. It can contain letters, numbers, underscore and minus sign

File – Icon file location

CSS style can be changed via uploading new file. CSS define all control buttons, Smartphone visualization, Scheduler and Trend. For more information how to modify CSS file please contact your local front office for additional document.

Note! Please clear cache of the browser after uploading new css file.

2.9 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 HomeLYnk parts using hex-encoded strings (2 bytes per 1 byte of data).*

2.9.1 Event based

These are scripts that are executed when a group event occurs on the bus. Usually used when real-time response is required.
When pressing on the arrow on the lower side of the *Event-based, Resident* or *Scheduled* buttons, two options 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:

Script name – The name of the script

Event group address – Allows to enter only digits from 0.9 and / as a separator. When ❗ icon appears on the right side of the text-box, wrong address form is used. Correct form of the group-address is, for example, 1/1/1.

Tag – Script can run on tags. If group addresses has tag attached to and script is using tag then any telegram which is send to the group with this tag will execute script.

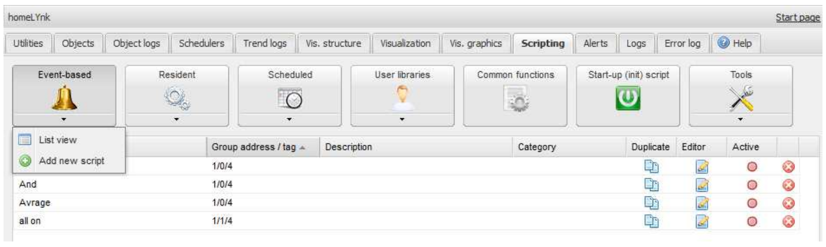
Active– Specifies whether the script is active (green circle) or disabled (red circle)

Execute on group read– Specifies whether the script is executed on KNX group read telegram.

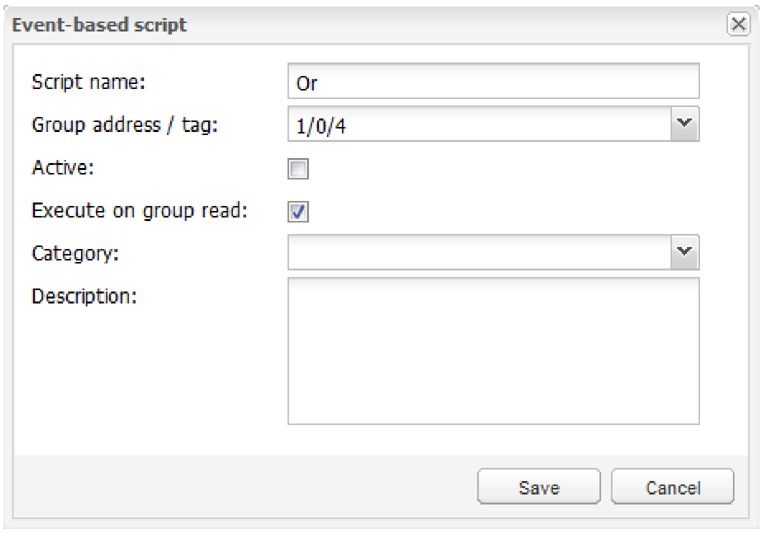
Category – A new or existing name of the category the script will be included. This will not affect on script action, helps only by grouping the scripts and watching by categories in *Tools* ⍰ *Print* script listings page.

Description– description of the script

Note! If the script is to be run only on read request ,use following script example:
if event.type == 'groupread' then
-- *script here*
end



	Group address / tag	Description	Category	Duplicate	Editor	Active
	1/0/4					●
And	1/0/4					●
Avrage	1/0/4					●
all on	1/1/4					●



Script name:	Or
Group address / tag:	1/0/4
Active:	☐
Execute on group read:	☑
Category:	
Description:	

2.9.2 Resident

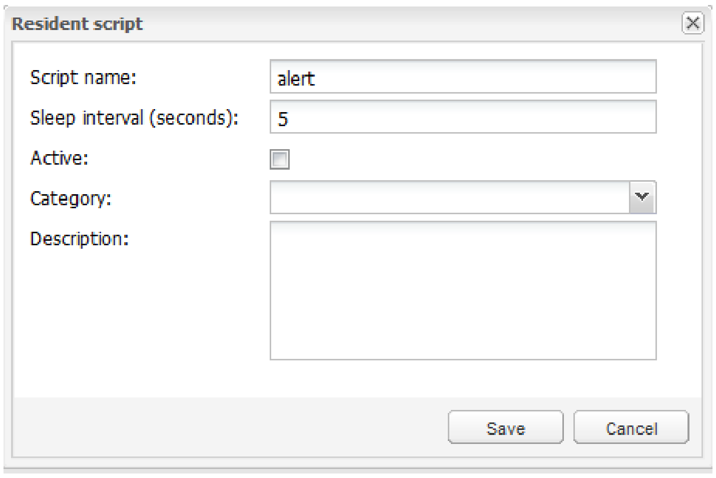
Script name – The name of the script

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


Resident script

Script name:	alert
Sleep interval (seconds):	5
Active:	☐
Category:	
Description:	

2.9.3 Scheduled

Script name – The name of the script

Minute – Minute

Hour – Hour

Day of the month – Day of the month

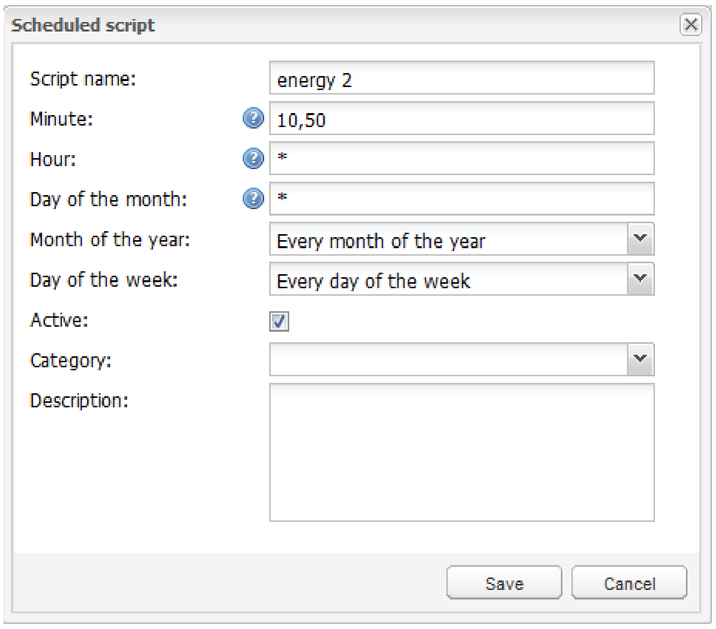
Month of the year – Month of the year

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.

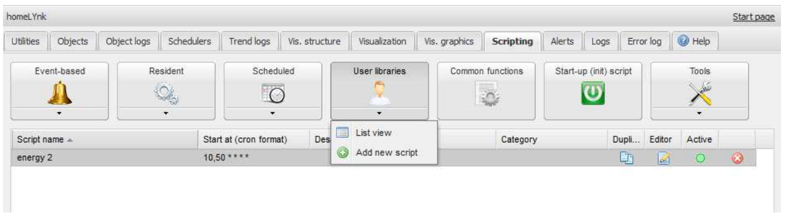
Description– description of the script


Scheduled script

Script name:	energy 2
Minute:	10,50
Hour:	*
Day of the month:	*
Month of the year:	Every month of the year
Day of the week:	Every day of the week
Active:	☑
Category:	
Description:	

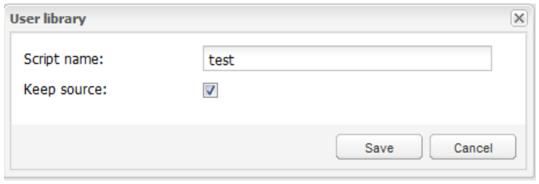
2.9.4 User libraries

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



Script name	Start at (cron format)
energy 2	10,50 * * * *

<u>Secure the code</u>



Script name: test
Keep source: ✓

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.

<u>Include the library in the scripts</u>

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 as below :

require('user.test')

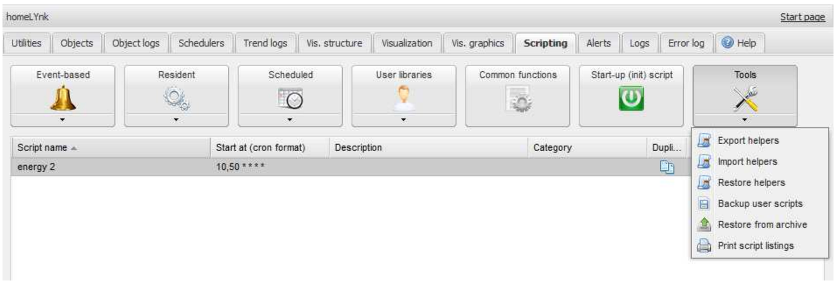
2.9.5 Common functions

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

2.9.6 Start-up 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 (power up, reboot in the SW or via RESET push button).

2.9.7 Tools



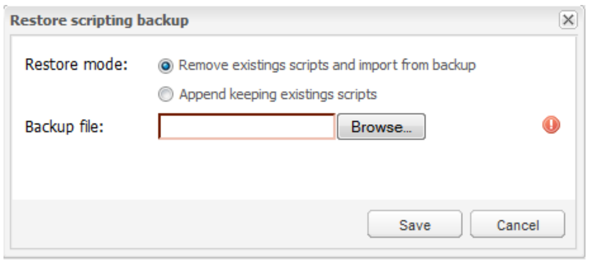
Script name	Start at (cron format)
energy 2	10,50 * * * *

Export helpers – Export scripting helpers

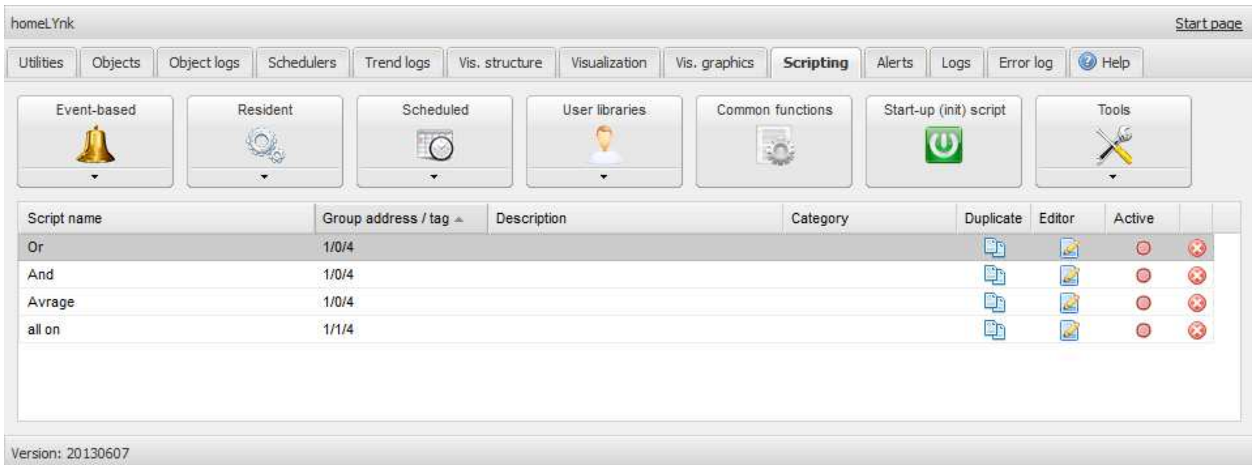
Import helpers – Import scripting helpers

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.	 Restore scripting backup Restore mode: [✓ Remove existings scripts and import from backup] Append keeping existings scripts Backup file:

2.9.8 General scripting description



Script name	Group address / tag	Description	Category	Duplicate	Editor	Active
Or	1/0/4					
And	1/0/4					
Avrage	1/0/4					
all on	1/1/4					

Version: 20130607

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.

2.9.9 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. Code snippets save time and make the coding convenient. After clicking on appropriate snippet, it automatically adds code to the editor field.
Keyboard shortcuts are implemented for help with script writing

Ctrl + F – Find syntax in a code, text will be highlighted in yellow.

Ctrl + G – after finding a text via Ctrl+F we can use Ctrl +G to select next syntax in a script.

Shift + Ctrl + G – select previous syntax.

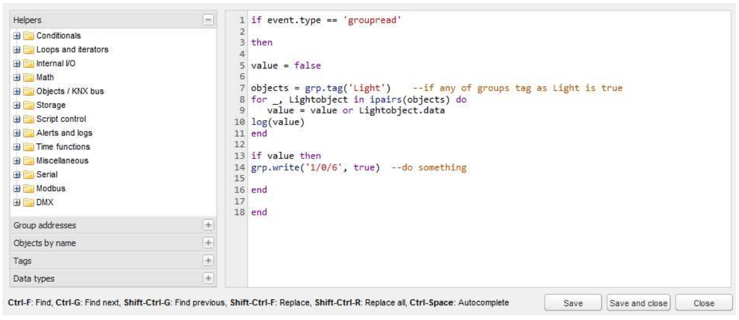
Shift + Ctrl + F – replace syntax in a script by another one. You will be allow to chose one by one if you want to change it.

Shift + Ctrl + R - replace all syntaxes in a script by another one at once.

Ctrl + Space – helps to auto detect code and write for you. Press Ctrl + Space and write first letter of a command then select correct one from the list

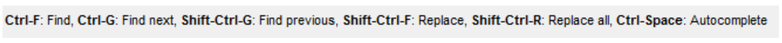
There are five main groups of Script editor:

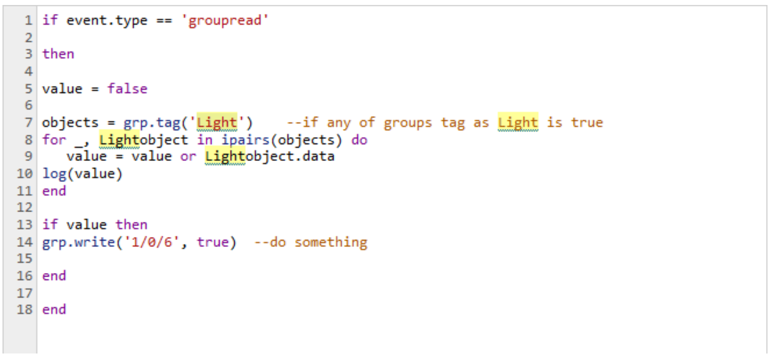
Helpers – Predefined code snippets, like if-then statement. Helpers consist of three main sub-groups:
Conditionals – If Else If, If Then etc.
Loops and iterators – Array, Repeat...Until etc
Math – Random value, Ceiling, Absolute value, Round etc.
Objects/KNX bus – Get object value, Group read, Group write, Update interval etc.
Storage – Get data from storage, Save data to storage
Script control – Get other script status, enable or disable other scripts
Alerts and logs – Alert, Log variables, Formatted alert
Time functions – Delay script execution
Miscellaneous – Sunrise/sunset etc.



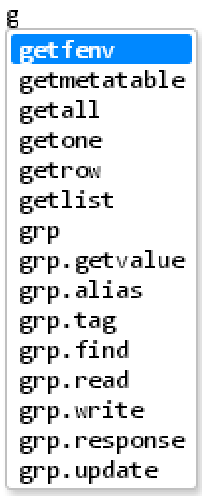
```
if event.type == 'groupread'

then

value = false

objects = grp.tag('Light')    --if any of groups tag as Light is true
for _, Lightobject in ipairs(objects) do
   value = value or Lightobject.data
log(value)
end

if value then
grp.write('1/0/6', true)  --do something

end

end
```

Ctrl-F: Find, **Ctrl-G**: Find next, **Shift-Ctrl-G**: Find previous, **Shift-Ctrl-F**: Replace, **Shift-Ctrl-R**: Replace all, **Ctrl-Space**: Autocomplete


Ctrl-F: Find, **Ctrl-G**: Find next, **Shift-Ctrl-G**: Find previous, **Shift-Ctrl-F**: Replace, **Shift-Ctrl-R**: Replace all, **Ctrl-Space**: Autocomplete



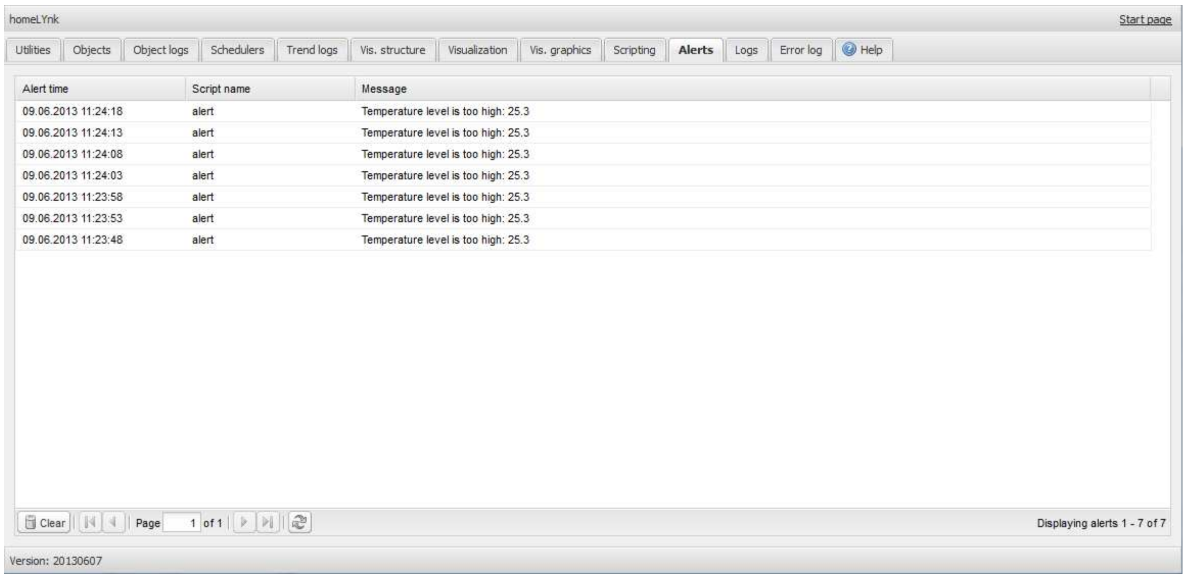
```
if event.type == 'groupread'

then

value = false

objects = grp.tag('Light')    --if any of groups tag as Light is true
for _, Lightobject in ipairs(objects) do
   value = value or Lightobject.data
log(value)
end

if value then
grp.write('1/0/6', true)  --do something

end

end
```


```
g
getfenv
getmetatable
getall
getone
getrow
getlist
grp
grp.getvalue
grp.alias
grp.tag
grp.find
grp.read
grp.write
grp.response
grp.update
```

Serial – Communication through internal HomeLYnk 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.	

2.10 Alerts

In *Alerts* tab a list of alert messages defined with ***alert*** function in scripts is located. The messages are stored in the main database.



Alert time	Script name	Message
09.06.2013 11:24:18	alert	Temperature level is too high: 25.3
09.06.2013 11:24:13	alert	Temperature level is too high: 25.3
09.06.2013 11:24:08	alert	Temperature level is too high: 25.3
09.06.2013 11:24:03	alert	Temperature level is too high: 25.3
09.06.2013 11:23:58	alert	Temperature level is too high: 25.3
09.06.2013 11:23:53	alert	Temperature level is too high: 25.3
09.06.2013 11:23:48	alert	Temperature level is too high: 25.3

Displaying alerts 1 - 7 of 7

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

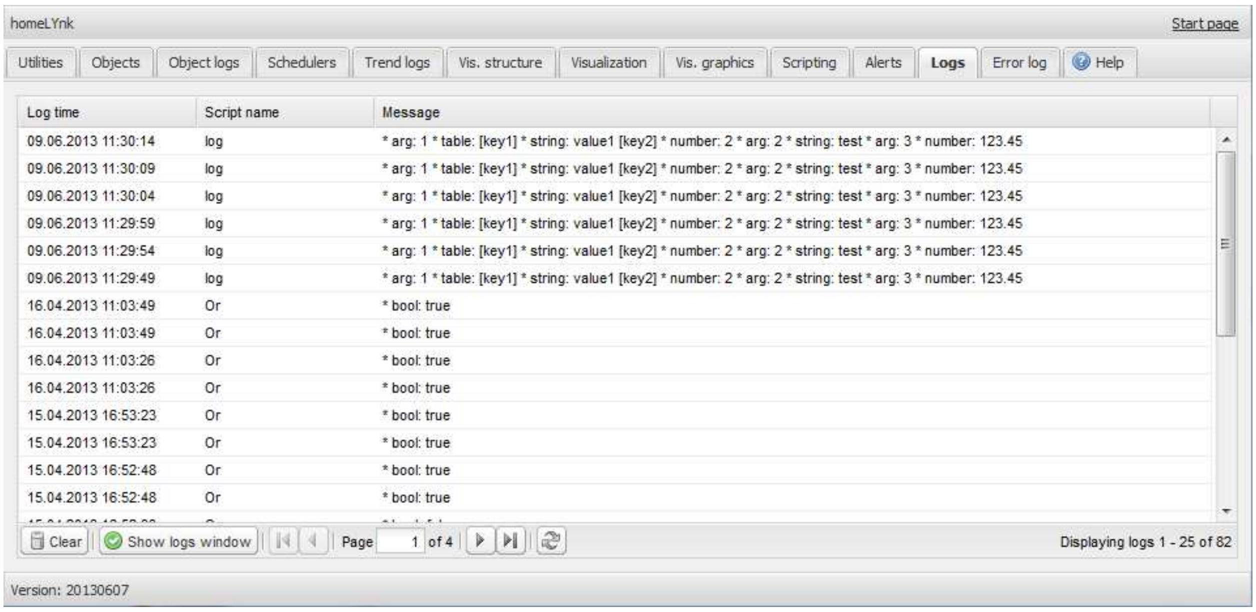
Stores alert message and current system time in the main database

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
```

2.11 Logs

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



Log time	Script name	Message
09.06.2013 11:30:14	log	* arg: 1 * table: [key1] * string: value1 [key2] * number: 2 * arg: 2 * string: test * arg: 3 * number: 123.45
09.06.2013 11:30:09	log	* arg: 1 * table: [key1] * string: value1 [key2] * number: 2 * arg: 2 * string: test * arg: 3 * number: 123.45
09.06.2013 11:30:04	log	* arg: 1 * table: [key1] * string: value1 [key2] * number: 2 * arg: 2 * string: test * arg: 3 * number: 123.45
09.06.2013 11:29:59	log	* arg: 1 * table: [key1] * string: value1 [key2] * number: 2 * arg: 2 * string: test * arg: 3 * number: 123.45
09.06.2013 11:29:54	log	* arg: 1 * table: [key1] * string: value1 [key2] * number: 2 * arg: 2 * string: test * arg: 3 * number: 123.45
09.06.2013 11:29:49	log	* arg: 1 * table: [key1] * string: value1 [key2] * number: 2 * arg: 2 * string: test * arg: 3 * number: 123.45
16.04.2013 11:03:49	Or	* bool: true
16.04.2013 11:03:49	Or	* bool: true
16.04.2013 11:03:26	Or	* bool: true
16.04.2013 11:03:26	Or	* bool: true
15.04.2013 16:53:23	Or	* bool: true
15.04.2013 16:53:23	Or	* bool: true
15.04.2013 16:52:48	Or	* bool: true
15.04.2013 16:52:48	Or	* bool: true

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

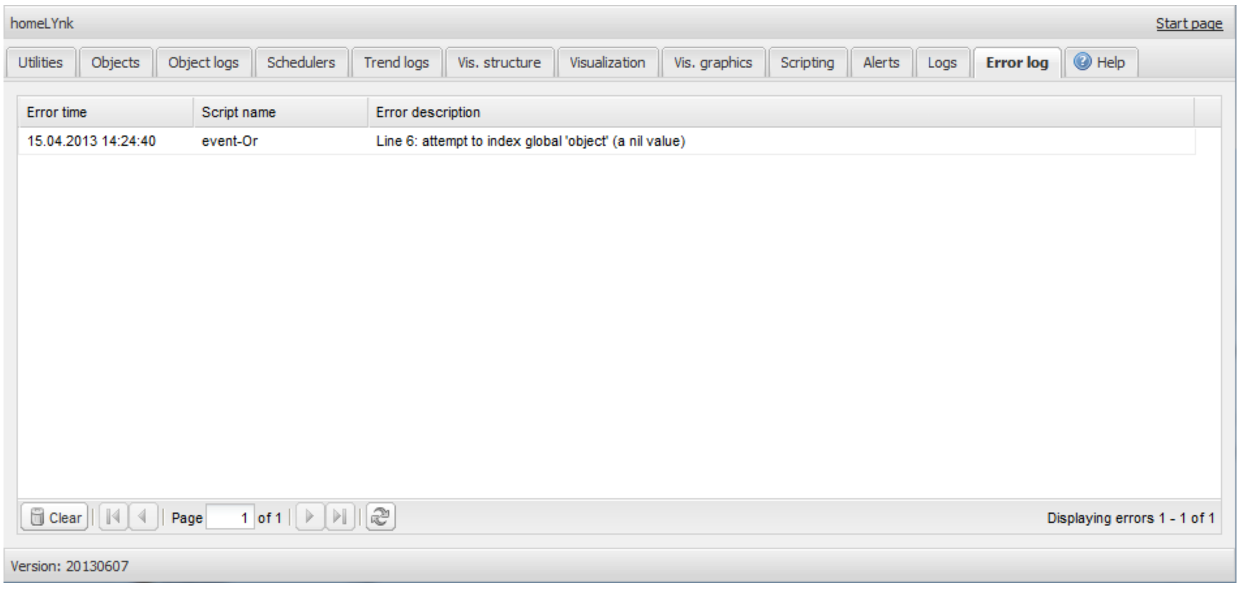
Converts variables to human-readable form and stores them.

Example

```
-- log function accepts Lua nil, boolean, number and table (up to 5 nested levels) type variables

a ={ key1 ='value1', key2 =2}

b ='test'

c =123.45

-- logs all passed variables

log(a, b, c)
```

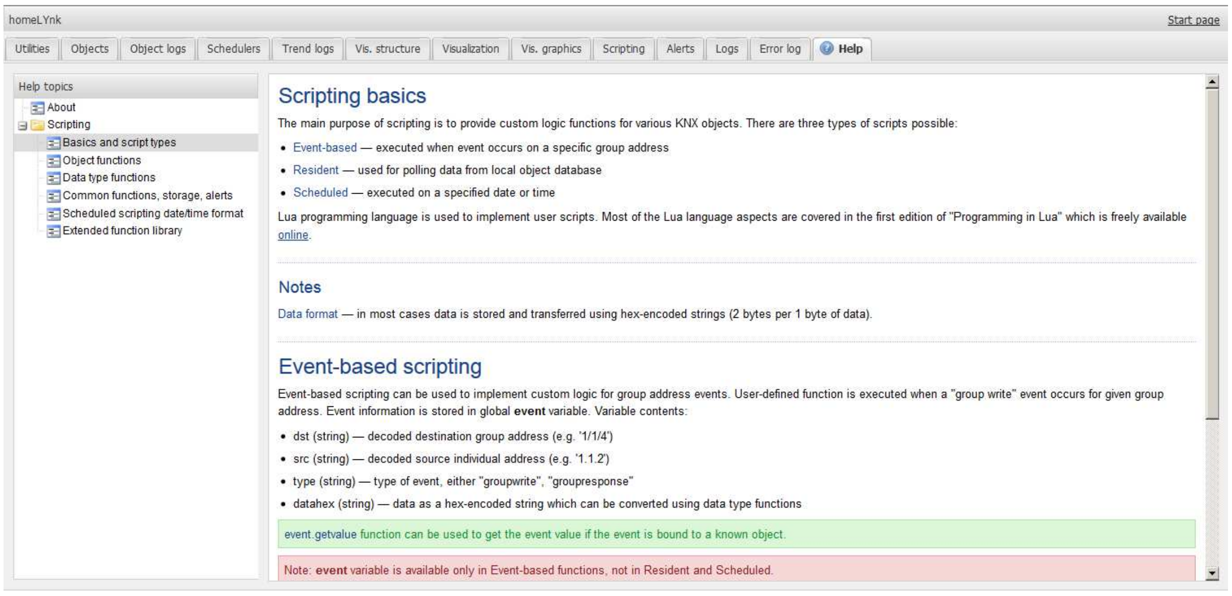
2.12 Error Log

Error messages from scripts are displayed in Error log tab.



Error time	Script name	Error description
15.04.2013 14:24:40	event-Or	Line 6: attempt to index global 'object' (a nil value)

2.13 Help



Scripting basics

The main purpose of scripting is to provide custom logic functions for various KNX objects. There are three types of scripts possible:

- Event-based — executed when event occurs on a specific group address
- Resident — used for polling data from local object database
- Scheduled — executed on a specified date or time

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 online.

Notes

Data format — in most cases data is stored and transferred using hex-encoded strings (2 bytes per 1 byte of data).

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" event occurs for given group address. Event information is stored in global **event** variable. Variable contents:

- dst (string) — decoded destination group address (e.g. '1/1/4')
- src (string) — decoded source individual address (e.g. '1.1.2')
- type (string) — type of event, either "groupwrite", "groupresponse"
- datahex (string) — data as a hex-encoded string which can be converted using data type functions

event.getvalue function can be used to get the event value if the event is bound to a known object.

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

3 Modbus/RS-485

3.1 Characteristics

The Modbus open standard allows you to receive a more in-depth analysis of consumption in all areas of your building.
You can connect up to 31 Modbus devices/slaves of the following types of meters based on Modbus remote terminal unit (RTU) within one Modbus line:

• Schneider Electric energy meters
• Schneider Electric power meters
• Schneider Electric Smart Interface Modules (SIM10M module)
• Non-Schneider Electric Modbus TCP/RTU devices (offering you greater flexibility)

With the information which the homeLYnk provides you can visualize energy or media consumption. This can also be used to reduce consumption through the use of control strategies within the KNX/IP network.

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 or visualization).

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

Application Example

Requirements

• measure and visualize how much energy is used lighting an office building
• measure the gas and water consumption in the building
• monitor the quality of the network to ensure the operational safety of IT equipment

Solution

• install an iEM3150 meter to record the energy consumed by the lighting
• install an iEM3255 meter to determine the power mains quality
• install a SIM10M module to measure gas and water consumption via impulse
• connect the devices to each other via Modbus

3.1.1 Modbus RTU Interface

- Modbus/RTU Master
- Modbus/RTU Slave
- Gateway Modbus TCP / RTU / KNX TP1/ KNX IP
- Copper Ethernet interface 10Mb, 100Mb
- Web server
- Supported Function Codes: #01, #02, #03, #04, #05, #06, #07, #0F, #10

RS 485 Characteristics

Connection Type	● point-to-point connections ● point-to-multipoint connections
Type of Trunk Cable	shielded cable with 1 twisted pair and at least a third conductor
Maximum Length of Bus	1,000 m (3,280 ft) at 19,200 bit/s with the Telemecanique TSX CSA• cable
Maximum Number of Devices (without repeater)	32 (1 UL) devices, i.e. 31 slaves
Maximum Length of Tap Links	● 20 m (65 ft) for one tap link ● a total of 40 m (131 ft) for all tap links available on the bus

3.1.2 Modbus TCP Interface

- Modbus/TCP-IP Client
- Modbus/TCP-IP Server
- Gateway Modbus TCP / RTU / KNX TP1/ KNX IP
- Copper Ethernet interface 10Mb, 100Mb
- DHCP support
- Web server
- Max. open TCP connections 100
- Supported Function Codes: #01, #02, #03, #04, #05, #06, #07, #0F, #10

Grounding-Isolation

- RS485 interface is not isolated!
- Metal cover of the RJ45 socket is connected to device ground

3.2 Configuration commands

Create Modbus TCP object

```
require('luamodbus')
mb = luamodbus.tcp()
```

Create Modbus RTU object

```
require('luamodbus')
mb = luamodbus.rtu()
```

Open Modbus TCP connection

IP: 192.168.1.2, port: 1234

```
mb:open('192.168.1.2', 1234)
mb:connect()
```

Open Modbus RTU connection

38400 baud rate, even parity, 8 data bits, 1 stop bit, half duplex

```
mb:open('/dev/RS485', 19200, 'E', 8, 1, 'H')
mb:connect()
```

Terminal name:

'/dev/RS485'

Supported Baud rates:

300	bit/s
600	bit/s
1200	bit/s
2400	bit/s
4800	bit/s
9600	bit/s
19200	**bit/s**
38400	bit/s
57600	bit/s
115200	bit/s
230400	bit/s

Parity:

„N“	None
„E“	**Even**
„O“	Odd

Data bits: [Number of data bits = 5, 6, 7, **8**]
Stop bits: [Number of stop bits **1**, 2]

Duplex:

„H“	**Half duplex**
“F”	Full duplex (not supported in RS-485)

The Baud rate is set depending on the distance between Modbus RTU devices. For instance with a Baud rate of 9600 bit/sec the maximum communication distance between 1 - 15 Modbus RTU devices is 1,200 metres. With the Baud rate of 19200 bit/sec the maximum communication distance is 900 metres, as the table shows below:

Baudrate setting	Maximum communication distance for 1 to 15 Modbus RTU devices (Typical with Belden 3105A cables)
9600 bit/sec	1200 m
19200 bit/sec	900 m

Parity refers to the technique of checking if transmission has been successful when transmitting between devices. It lets you know if some data has been lost during transmission.

Setting of parity

The Modbus supports only 11 bit frames. The ETS application sets stop bits automatically depending on the parity setting. "Parity" refers to the number of 1s in a given binary number. Odd parity means there are an odd number of 1s and even parity means that there is an even number of 1s. Parity bits are used as a means of error detection as digital data is transmitted and received.
Both the Gateway and Meter must always be set to the same as one another, odd, even or none.
The default parity mode of Modbus is "even" parity.
• Parity = None: choose between one and two stop bits
• Parity = Even: one stop bit is set
• Parity = Odd: one stop bit is set

Delay between frames

Some devices require considerable time after end of response until they are ready to receive the following request from the master. In particular it applies to Schneider Electric SEPAM power devices and legacy slave devices. As they are slow in dealing with the original request they may miss the following request.
The time between requests should be greater than 3.5 characters according to the Modbus specification. However, these legacy devices need more time. Please use delay command appropriately:

Wait for 1.5 seconds
os.sleep(1.5)
Communication itself takes care of minimal 3, 5 character delay.

Example:

```
init modbus on first script execution
        if not mb then
                require('luamodbus')
                mb = luamodbus.rtu()
                mb:open('/dev/RS485', 38400, 'E', 8, 1, 'H')
                mb:connect()
        end
        mb:setslave(30)
        mb:flush()
```

Timeout interval between two consecutive bytes of the same message

```
        mb:getbytetimeout()
        mb:setbytetimeout(timeout)
```

Timeout interval used to wait for a response:

```
        mb:getresponsetimeout()
        mb:setresponsetimeout(timeout)
```

Timeout interval used to for an incoming indication from master (slave mode only):

```
        mb:getreceivetimeout()
        mb:setreceivetimeout(timeout)
```

Set slave address

mb:setslave(123)
[1..247]

Read registers

read from address 1000
value = mb:readregisters(1000)

Close modbus connection

mb:close()

3.3 Function codes (0..127)

FC#01 Read Coils

Name	**“Read single coil”**
Command	**coil = mb:readcoils(1000)**

[address]

Name	**“Read Multiple coil”**
Command	**coil1, coil2, coil3 = mb:readcoils(1000, 3)**

[Starting address, Quantity of coils / max 2000 bits]
1 = ON, 0 = OFF

This function code is used to read from 1 to 2000 contiguous status of coils in a remote device. The Request PDU specifies the starting address, i.e. the address of the first coil specified, and the number of coils. In the PDU Coils are addressed starting at zero. Therefore coils numbered 1-16 are addressed as 0-15.
The coils in the response message are packed as one coil per bit of the data field. Status is indicated as 1= ON and 0= OFF. The LSB of the first data byte contains the output addressed in the query. The other coils follow toward the high order end of this byte, and from low order to high order in subsequent bytes.
If the returned output quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request

Function code	1 Byte	**0x01**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of coils	2 Bytes	1 to 2000 (0x7D0)

Response

Function code	1 Byte	**0x01**
Byte count	1 Byte	**N***
Coil Status	**n** Byte	n = N or N+1

*N = Quantity of Outputs / 8, if the remainder is different of 0 ⇒ N = N+1

Error

Function code	1 Byte	**Function code + 0x80**
Exception code	1 Byte	01 or 02 or 03 or 04

FC#02 Read Discrete Inputs

Name **"Read discrete input"**
Command **value = mb:readdiscreteinputs(1000)**
[address]

Name **"Read discrete inputs"**
Command **value = mb:readdiscreteinputs(1000,x)**
[address of the first input specified, number of inputs]

This function code is used to read from 1 to 2000 contiguous status of discrete inputs in a remote device. The Request PDU specifies the starting address, i.e. the address of the first input specified, and the number of inputs. In the PDU Discrete Inputs are addressed starting at zero. Therefore Discrete inputs numbered 1-16 are addressed as 0-15.
The discrete inputs in the response message are packed as one input per bit of the data field. Status is indicated as 1= ON; 0= OFF. The LSB of the first data byte contains the input addressed in the query. The other inputs follow toward the high order end of this byte, and from low order to high order in subsequent bytes.
If the returned input quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request

Function code	1 Byte	**0x02**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Inputs	2 Bytes	1 to 2000 (0x7D0)

Response

Function code	1 Byte	**0x02**
Byte count	1 Byte	**N***
Input Status	**N*** x 1 Byte	

*N = Quantity of Inputs / 8 if the remainder is different of 0 ⇒ N = N+1

Error

Error code	1 Byte	**0x82**

Exception code	1 Byte	01 or 02 or 03 or 04

FC#03 Read holding registers

Name **“Read registers”**
Command **value = mb:readregisters(1015,6)**
[starting address, quantity of registers 1..125]

This function code is used to read the contents of a contiguous block of holding registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore registers numbered 1-16 are addressed as 0-15.
The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request

Function code	1 Byte	**0x03**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	**0x03**
Byte count	1 Byte	2 x **N***
Register value	**N*** x 2 Bytes	

*N = Quantity of Registers

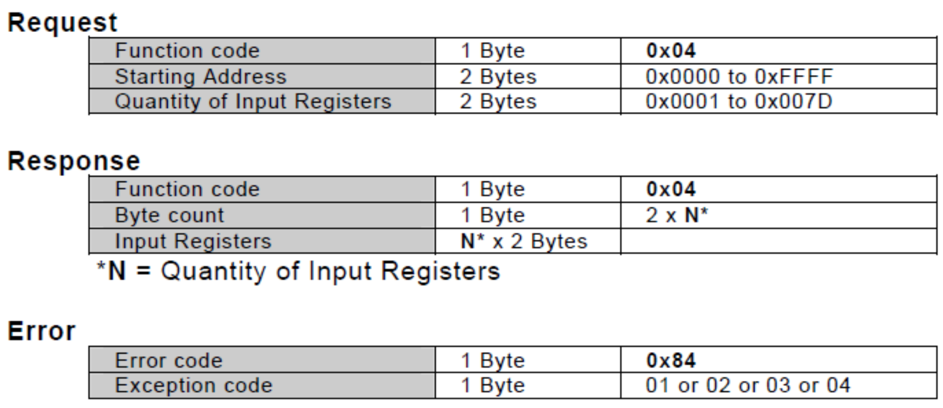
Error

Error code	1 Byte	**0x83**
Exception code	1 Byte	01 or 02 or 03 or 04

FC#04 Read Input Registers

Name **“Read input registers”**
Command **value = mb:readinputregisters(1015,6)**
[starting address, quantity of registers 1..125]

This function code is used to read from 1 to 125 contiguous input registers in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU Registers are addressed starting at zero. Therefore input registers numbered 1-16 are addressed as 0-15.
The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.



Request

Function code	1 Byte	**0x04**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Input Registers	2 Bytes	0x0001 to 0x007D

Response

Function code	1 Byte	**0x04**
Byte count	1 Byte	2 x **N***
Input Registers	**N*** x 2 Bytes	

*N = Quantity of Input Registers

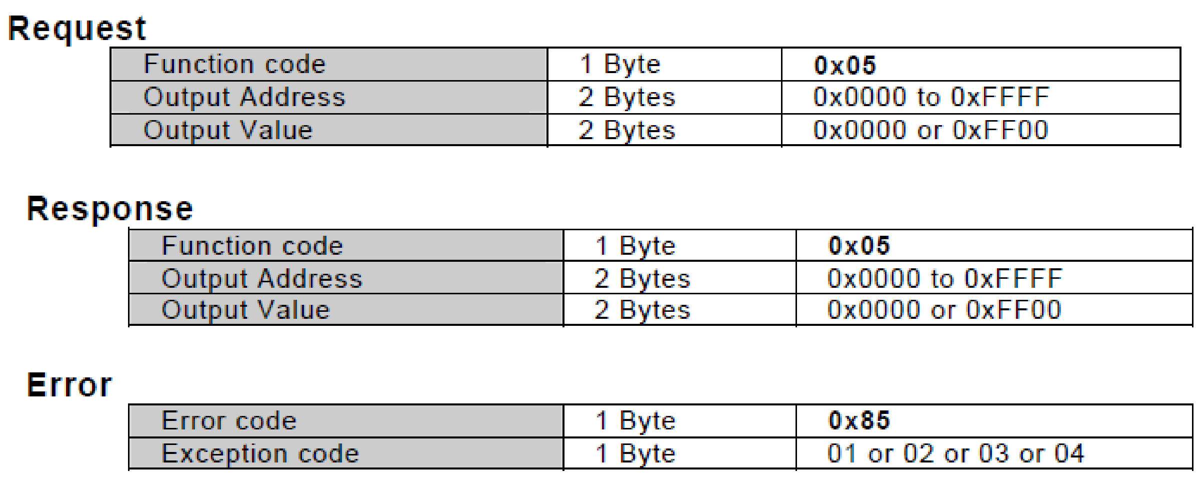
Error

Error code	1 Byte	**0x84**
Exception code	1 Byte	01 or 02 or 03 or 04

FC#05 Write Single Coil

Name **"Write single bit"**
Command **value = mb:writebits(1000, true)**
[starting address, value "true" or "false"/"0"]

This function code is used to write a single output to either ON or OFF in a remote device. The requested ON/OFF state is specified by a constant in the request data field. A value of FF 00 hex requests the output to be ON. A value of 00 00 requests it to be OFF. All other values are illegal and will not affect the output.
The Request PDU specifies the address of the coil to be forced. Coils are addressed starting at zero. Therefore coil numbered 1 is addressed as 0. The requested ON/OFF state is specified by a constant in the Coil Value field. A value of 0XFF00 requests the coil to be ON.
A value of 0X0000 requests the coil to be off. All other values are illegal and will not affect the coil.
The normal response is an echo of the request, returned after the coil state has been written.



Request

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Response

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Error

Error code	1 Byte	**0x85**
Exception code	1 Byte	01 or 02 or 03 or 04

FC#06 Write Single Register

Name **"Write single register"**
Command **value = mb:writeregisters(1000, 123)**
[address, value]

This function code is used to write a single holding register in a remote device. The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register numbered 1 is addressed as 0. The normal response is an echo of the request, returned after the register contents have been written.

Request

Function code	1 Byte	**0x06**
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 to 0xFFFF

Response

Function code	1 Byte	**0x06**
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 to 0xFFFF

Error

Error code	1 Byte	**0x86**
Exception code	1 Byte	01 or 02 or 03 or 04

FC#0F Write Multiple Coils

Name **“Write multiple bits”**
Command **value = mb:writebits(1000, true, false,true,…)**
[address, bit value1, bit value2,..{max 1968 bits}]

This function code is used to force each coil in a sequence of coils to either ON or OFF in a remote device. The Request PDU specifies the coil references to be forced. Coils are addressed starting at zero. Therefore coil numbered 1 is addressed as 0. The requested ON/OFF states are specified by contents of the request data field. A logical '1' in a bit position of the field requests the corresponding output to be ON. A logical '0' requests it to be OFF. The normal response returns the function code, starting address, and quantity of coils forced.

Request PDU

Function code	1 Byte	**0x0F**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Outputs	2 Bytes	0x0001 to 0x07B0
Byte Count	1 Byte	**N***
Outputs Value	**N*** x 1 Byte	

***N** = Quantity of Outputs / 8, if the remainder is different of 0 $\Rightarrow$ N = N+1

Response PDU

Function code	1 Byte	**0x0F**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Outputs	2 Bytes	0x0001 to 0x07B0

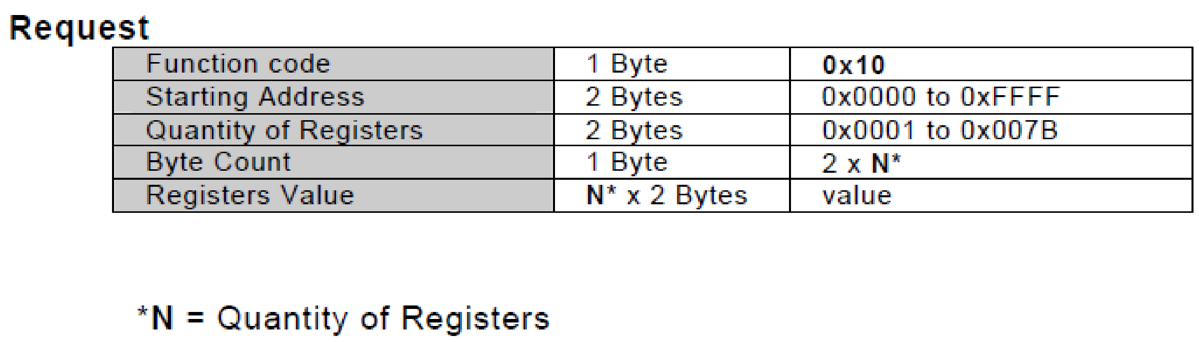
Error

Error code	1 Byte	**0x8F**
Exception code	1 Byte	01 or 02 or 03 or 04

FC#10 Write Multiple Registers

Name **“Write multiple registers”**
Command **value = mb:writeregisters(1000, 123, 321,222,..)**
[address, value1, value2, ..{max 123 registers}]

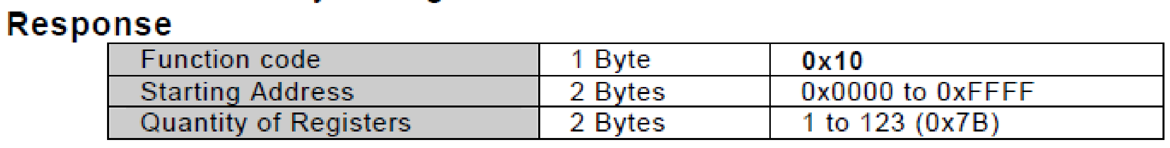
This function code is used to write a block of contiguous registers (1 to 123 registers) in a remote device. The requested written values are specified in the request data field. Data is packed as two bytes per register. The normal response returns the function code, starting address, and quantity of registers written.



Request

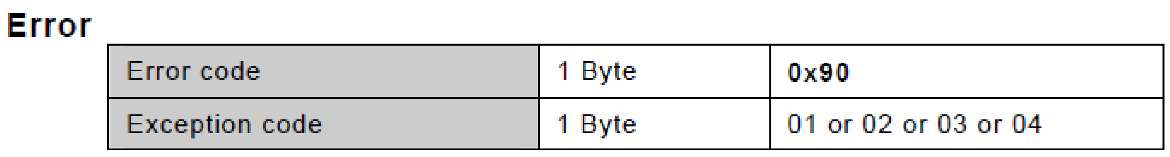
Function code	1 Byte	**0x10**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	0x0001 to 0x007B
Byte Count	1 Byte	2 x **N***
Registers Value	**N*** x 2 Bytes	value

*N = Quantity of Registers



Response

Function code	1 Byte	**0x10**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 123 (0x7B)



Error

Error code	1 Byte	**0x90**
Exception code	1 Byte	01 or 02 or 03 or 04

Exception codes (128..255)

mb:readcoils(start, count)

mb:readdiscreteinputs(start, count)

mb:readregisters(start, count)

mb:readinputregisters(start, count)

These commands read one or many registers/coils from the start address and return all values on success. In case of error, three variables are sent back:

- Nill
- Exception code description
- Exception code

MODBUS Exception Codes		
Code	Name	Meaning
01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
02	ILLEGAL DATA ADDRESS	The data address received in the query is not an allowable address for the server (or slave). More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, the PDU addresses the first register as 0, and the last one as 99. If a request is submitted with a starting register address of 96 and a quantity of registers of 4, then this request will successfully operate (address-wise at least) on registers 96, 97, 98, 99. If a request is submitted with a starting register address of 96 and a quantity of registers of 5, then this request will fail with Exception Code 0x02 "Illegal Data Address" since it attempts to operate on registers 96, 97, 98, 99 and 100, and there is no register with address 100.
03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.
04	SLAVE DEVICE FAILURE	An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
05	ACKNOWLEDGE	Specialized use in conjunction with programming commands. The server (or slave) has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client (or master). The client (or master) can next issue a Poll Program Complete message to determine if processing is completed.
06	SLAVE DEVICE BUSY	Specialized use in conjunction with programming commands. The server (or slave) is engaged in processing a long–duration program command. The client (or master) should retransmit the message later when the server (or slave) is free.
08	MEMORY PARITY ERROR	Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server (or slave) attempted to read record file, but detected a parity error in the memory. The client (or master) can retry the request, but service may be required on the server (or slave) device.
0A	GATEWAY PATH UNAVAILABLE	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

3.4 Master mode functions

mb:setslave(slaveid)

sets slave id to read/write data from/to

mb:readcoils(start, count) [01]

mb:readdiscreteinputs(start, count) [02]

mb:readregisters(start, count) [03]

mb:readinputregisters(start, count) [04]

reads one or many registers/coils from the start address

returns all values on success and nil, error description on error

mb:writebits(start, v1, [v2, [v3, ...]]) [05]

mb:writeregisters(start, v1, [v2, [v3, ...]]) [06]

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

3.5 Slave mode functions

mb:receive()
receives data from master with 1 minute timeout

returns data as a binary string on success and nil, error description on error

mb:setmapping(coils, inputs, holding_regs, input_regs)
creates memory mapping for the registers with size specified for each type

mb:handleslave()
waits for an incoming indication from master and sends a reply when necessary

mb:getcoils(start, count)

mb:getdiscreteinputs(start, count)

mb:getinputregisters(start, count)

mb:getregisters(start, count)
gets one or many register/coil values from mapping from the start address

returns all values on success and nil, error description on error, exception code if applicable

mb:setcoils(start, v1, [v2, [v3, ...]])

mb:setdiscreteinputs(start, v1, [v2, [v3, ...]])

mb:setinputregisters(start, v1, [v2, [v3, ...]])

mb:setregisters(start, v1, [v2, [v3, ...]])
sets values to register/coil mapping from the start address

returns true on success and nil, error description on error, exception code if applicable

mb:setwritecoilcb(fn)

mb:setwriteregistercb(fn)
sets a callback function for coil/register write event

callback should accept two parameters - coil/register address and value (boolean or number)

for multiple writes callback is executed for each coil/register separately

use nil to remove a callback

4 RS-232

4.1 Characteristics

The RS-232 serial interface communications standard has been in use for very many years and is one of the most widely used standards for serial data communications as a result of it being simple and reliable.

The RS232 serial interface standard still retains its popularity and remains in widespread use. It is still found on some computers and on many interfaces, often being used for applications ranging from data acquisition to supplying a serial data communications facility in general computer environments.

The long term widespread use of the RS232 standard has meant that products are both cheap and freely available, and in these days of new higher speed standards, the reliable, robust RS232 standard still has much to offer. The interface is intended to operate over distances of up to 15 meter; it is based on one Master/ one Slave rule.

Application example

- Connection to simple devices or other bus sub systems
- Audio/video, IR system integration

4.2 Configuration commands

Configuration commands with parameters are the same as the Modbus RS-485 serial connection

Open connection

```
require('serial')
port = serial.open('/dev/RS232', {baudrate = 9600})
```

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()
```

5 USB 2.0

5.1 Characteristics

- USB 2.0 provides a bandwidth of 480 Mbit/s, corresponding to an effective image data rate of 40 MB/s.
- Integrated voltage supply (5 VDC) for devices in the 4-pole cable. Devices complying with the USB specification may consume a total of 500 mA from the bus. Devices with a power of up to 2.5 W can therefore be supplied via the bus.
- USB cable must only be 4.5 m long at the maximum.
- Data transmission is possible in both directions

Application example

USB interface can be used for extending memory capacity via attaching USB flash drive.

5.2 Configuration commands

Read whole file at once. Returns file contents as a string on success or nil on error.

```
io.readfile (file)
```

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.

```
io.writefile (file, data)
```

Note: USB flash drive supports FAT, FAT32 and NTFS file system. Maximum size of Flash drive is 32GB.

Send and receive SMS messages via attaching USB GSM adapter.

- Use Huawei E173 modem
- The modem has to be plugged into any of USB ports of LM2 and it starts operating immediately
- Specific functions should be added into user script library with PIN code setting and telephone number white-list which will be able to receive and send in SMS messages

Command syntax

Write to bus:

- W ALIAS VALUE

Read from bus:

- R ALIAS
- On read request, script will reply with SMS message containing current value of selected object

ALIAS can be:

- Group address (e.g. 1/1/1)
- Name (e.g. Obj1). If name contains spaces then it must be escaped using double quotes (e.g. “Room Temperature”)

Note: Object data type and name must be set in Configurator -> Objects tab. Otherwise script won’t be able to read and write to object

Note: Only ASCII symbols are accepted in the message

6 LUA – Programming Language

LUA is a powerful, fast, lightweight, embeddable scripting language. LUA combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. LUA is dynamically typed, runs by interpreting byte code for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

6.1 Object functions

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

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

grp.getvalue(alias)
Returns value for the given alias or LUA *nil* when object cannot be found.

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

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

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

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

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

grp.alias (alias)
Converts group address to object name or name to address. Returns LUA *nil* when object cannot be found.

6.2 Group communication functions

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

grp.write (alias, value, datatype)
Sends group write request to the given alias. Data type is taken from the database if not specified as third parameter. Returns LUA *boolean* as the result.

grp.response (alias, value, datatype)
Similar to *grp.write.* Sends group response request to the given alias.

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

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

6.3 Returned object functions

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

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

object:write(value, datatype)
Sends group write request to object's group address. Data type is taken from the database if not specified as second parameter. Returns LUA *boolean* as the result.

object:response(value, datatype)
Similar to *object:write.* Sends group response request to object's group address.

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

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

6.4 Data type functions

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

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

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

6.5 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:

- ***bool** 1 bit (boolean) - **dt.*** — boolean
- *2 bit (1 bit controlled) - **dt.bit2*** — number
- *4 bit (3 bit controlled) - **dt.bit4*** — number
- *1 byte ASCII character - **dt.char*** — string
- *1 byte unsigned integer - **dt.uint8*** — number
- *1 byte signed integer - **dt.int8*** — number
- *2 byte unsigned integer - **dt.uint16*** — number

- *2 byte signed integer - **dt.int16*** — number
- *2 byte floating point - **dt.float16*** — number
- *3 byte time / day - **dt.time*** — table with the following items:
    - day — number (0-7)
    - hour — number (0-23)
    - minute — number (0-59)
    - second — number (0-59)
- *3 byte date - **dt.date*** — table with the following items:
    - day — number (1-31)
    - month — number (1-12)
    - year — number (1990-2089)
- *4 byte unsigned integer - **dt.uint32** —* number
- *4 byte signed integer - **dt.int32*** — number
- *4 byte floating point - **dt.float32*** — number
- *4 byte access control - **dt.access*** — number, currently not fully supported
- *14 byte ASCII string - **dt.string*** — string, null characters ('\0') are discarded during decoding

6.6 Data storage functions

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.

```
testfn=function(t)
return t * t
end
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
```

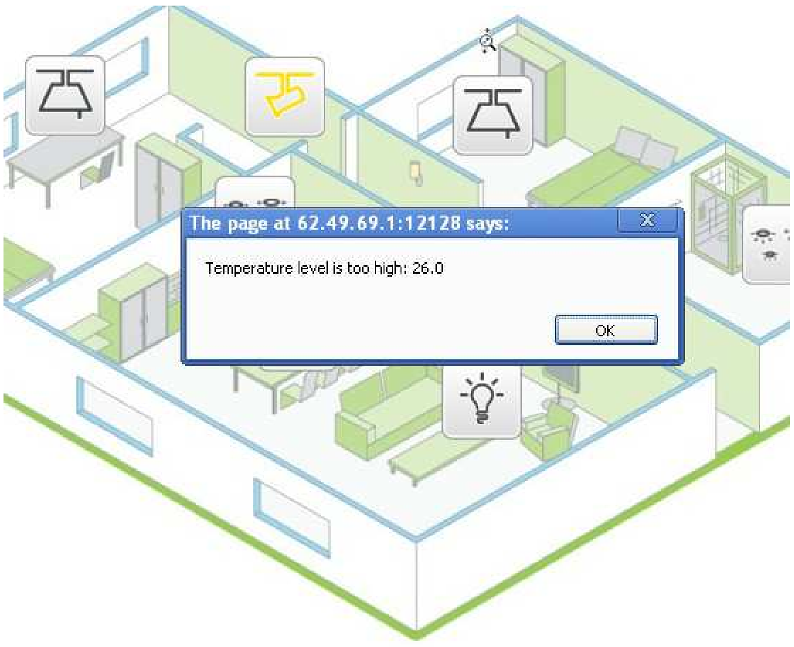
6.7 Alert functions

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
```


The page at 62.49.69.1:12128 says:

Temperature level is too high: 26.0

6.8 Log functions

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

```
-- log function accepts LUA nil, boolean, number and table (up to 5 nested levels) type variables
a ={ key1 ='value1', key2 =2}
b ='test'
c =123.45
-- logs all passed variables
log(a, b, c)
```

6.9 Time functions

os.sleep(delay)
Delay the next command execution for the delay seconds.

os.microtime ()
Returns two values: current timestamp in seconds and timestamp fraction in nanoseconds

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

6.10 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

```
s = "hello world from LUA"
for w in string.gmatch(s, "%a+") do
  print(w)
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:

```
t = {}
s = "from=world, to=LUA"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
  t[k] = v
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.

6.11 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.

6.12 Script control function

script.enable('scriptname')

Enable the script with the name scriptname.

script.disable('scriptname')

Disable the script with the name scriptname.

status = script.status('scriptname')

Returns true/false if script is found, nil otherwise

6.13 Conversions

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.

6.14 Bit operators

bit.bnot (value)
Binary not

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

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

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

bit.lshift (value, shift)
Left binary shift

bit.rshift (value, shift)
Right binary shift

6.15 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 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.

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

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

io.open (filename [, mode])

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

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

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

io.output ([file])

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

6.16 Mathematical functions

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

math.abs (x)

Returns the absolute value of x.

math.acos (x)

Returns the arc cosine of x (in radians).

math.asin (x)

Returns the arc sine of x (in radians).

math.atan (x)

Returns the arc tangent of x (in radians).

math.atan2 (y, x)

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

math.ceil (x)

Returns the smallest integer larger than or equal to x.

math.cos (x)

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

math.cosh (x)

Returns the hyperbolic cosine of x.

math.deg (x)

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

math.exp (x)

Returns the value e^x.

math.floor (x)

Returns the largest integer smaller than or equal to x.

math.fmod (x, y)

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

math.frexp (x)

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

math.huge

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

math.ldexp (m, e)

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

math.log (x)

Returns the natural logarithm of x.

math.log10 (x)

Returns the base-10 logarithm of x.

math.max (x, ···)

Returns the maximum value among its arguments.

math.min (x, ···)

Returns the minimum value among its arguments.

math.modf (x)

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

math.pi

The value of pi.

math.pow (x, y)

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

math.rad (x)

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

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

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

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

math.randomseed (x)

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

math.sin (x)

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

math.sinh (x)

Returns the hyperbolic sine of x.

math.sqrt (x)

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

math.tan (x)

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

math.tanh (x)

Returns the hyperbolic tangent of x.

6.17 Table manipulation

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.

6.18 Operating system facilities

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

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

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

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

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

os.difftime (t2, t1)

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

os.execute ([command])

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

os.exit ([code])

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

os.getenv (varname)

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

os.remove (filename)

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

os.rename (oldname, newname)

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

os.time ([table])

Returns the current time when called without arguments, or a time representing the date and time specified by the given table. This table must have fields year, month, and day, and may have fields hour, min, sec, and *isdst* (for a description of these fields, see the *os.date* function).

The returned value is a number, whose meaning depends on your system. In POSIX, Windows, and some other systems, this number counts the number of seconds since some given start time (the "epoch"). In other systems, the meaning is not specified, and the number returned by time can be used only as an argument to date and *difftime*.

os.tmpname ()

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

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

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

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

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

for k,v in pairs(t) do body end

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".

7 Script examples

7.1 Binary filter

Create two 1bit group addresses under Object tab where

1/1/1 input

1/1/2 output

Create event –based script and attach it to group 1/1/1. Script will run each time group 1/1/1 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1')
if value_1 == true then
-- do nothing
elseif value_1 == false then
grp.write('1/1/2', false)
end
```

7.2 Binary gate with bit gate

Create three 1bit group addresses under Object tab where

1/1/1 input

1/1/2 gate

1/1/3 output

Create event –based script and attach it to group 1/1/1. Script will run each time group 1/1/1 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1') --input
value_2 = grp.getvalue('1/1/2') --gate
if value_2 == true then
-- do nothing
elseif value_2 == false then
grp.write('1/1/3', value_1) --output
end
```

7.3 Gate with byte gate

Create three group addresses under Object tab where

1/1/1 input – any type but the same as output

1/1/2 gate- byte object

1/1/3 output – the same as input

Create event –based script and attach it to group 1/1/1. Script will run each time group 1/1/1 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1') -- input
value_2 = grp.getvalue('1/1/2') --gate
if value_2 == 0 then
-- do nothing
elseif  value_2 < 0 or value_2 > 0   then
grp.write('1/1/3', value_1) --output
end
```

7.4 Or - Port (2 in 1 0ut)

Create three 1bit group addresses under Object tab where

1/1/1 value 1

1/1/2 value 2

1/1/3 output

Add tag OR1 to value1 and value2 group addresses.

Create event –based script and attach it to Tag OR1. Script will run each time group 1/1/1 or group 1/1/2 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2')
if value_1 == true or value_2 == true then
grp.write('1/1/3', true)
else
grp.write('1/1/3', false)
end
```

7.5 And - Port (2 in 1 0ut)

Create three 1bit group addresses under Object tab where

1/1/1 value 1

1/1/2 value 2

1/1/3 output

Add tag AND1 to value1 and value2 group addresses.

Create event –based script and attach it to Tag AND1. Script will run each time group 1/1/1 or group 1/1/2 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2')
if value_1 == true and value_2 == true then
grp.write('1/1/3', true)
else
grp.write('1/1/3', false)
end
```

7.6 Or - Port (5 in 2 0ut)

Create group addresses under Object tab where

1/1/1 value 1 - 1bit

1/1/2 value 2 - 1bit

1/1/3 value 3 - 1bit

1/1/4 value 4 - 1bit

1/1/5 value 5 - 1bit

1/1/6 bit_output - 1bit

1/1/7 byte_output - 1byte

Add tag OR2 to group addresses value1, value2, value3, value4 and value 5.

Create event –based script and attach it to Tag OR2. Script will run each time groups 1/1/1, 1/1/2, 1/1/3, 1/1/4, 1/1/5 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2')
value_3 = grp.getvalue('1/1/3')
value_4 = grp.getvalue('1/1/4')
value_5 = grp.getvalue('1/1/5')
if value_1 == true or value_2 == true or value_3 == true or value_4 == true or value_5 == true then
grp.write('1/1/6', true) -- bit to 1
grp.write('1/1/7', 255) -- byte to 255
else
grp.write('1/1/6', false) -- bit to 0
grp.write('1/1/7', 0) -- byte to 0
end
```

7.7 And - Port (5 in 2 0ut)

Create group addresses under Object tab where

1/1/1 value 1 - 1bit
1/1/2 value 2 - 1bit
1/1/3 value 3 - 1bit

1/1/4 value 4 - 1bit
1/1/5 value 5 - 1bit
1/1/6 bit_output - 1bit
1/1/7 byte_output - 1byte

Add tag AND2 to group addresses value1, value2, value3, value4 and value 5.

Create event –based script and attach it to Tag AND2. Script will run each time groups 1/1/1, 1/1/2, 1/1/3, 1/1/4, 1/1/5 receive telegram

Add fallowing code to Script editor

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2')
value_3 = grp.getvalue('1/1/3')
value_4 = grp.getvalue('1/1/4')
value_5 = grp.getvalue('1/1/5')
if value_1 == true and value_2 == true and value_3 == true and value_4 == true and value_5 == true
then
grp.write('1/1/6', true) -- bit to 1
grp.write('1/1/7', 255) -- byte to 255
else
grp.write('1/1/6', false) -- bit to 0
grp.write('1/1/7', 0) -- byte to 0
end
```

7.8 Telegram transformer (0/1 bit to 0-255 byte)

Create two group addresses under Object tab where

1/1/1 input – 1bit
1/1/2 output – 1byte

Create event –based script and attach it to group 1/1/1. Script will run each time group 1/1/1 receive telegram

Add fallowing code to Script editor:

```
value_1 = grp.getvalue('1/1/1')
if value_1 == true then -- bit value (in)
grp.write('1/1/2', 255) -- byte value (out)
else
grp.write('1/1/2', 0) -- byte value (out)
end
```

7.9 Compare value

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2')
if value_1 == value_2 then
grp.write('1/1/3', true) -- bit to 1
grp.write('1/1/4', 255) -- byte to 255
else
```

```
grp.write('1/1/3', false) -- bit to 0
grp.write('1/1/4', 0) -- byte to 0
end
```

7.10 Save Scene 1 (RGB value)

```
value_1 = grp.getvalue('1/1/1') --RED
value_2 = grp.getvalue('1/1/2') --GREEN
value_3 = grp.getvalue('1/1/3') --BLUE
storage.set('Scene1_Red', value_1)
storage.set('Scene1_Green', value_2)
storage.set('Scene1_Blue', value_3)
```

7.11 Call Scene 1 (RGB value

```
value_1 = storage.get('Scene1_Red')
value_2 = storage.get('Scene1_Green')
value_3 = storage.get('Scene1_Blue')
if not value_1 then
--if storage value does not exist do nothing
else
grp.write('1/1/1', value_1) --RED
end
if not value_2 then
--if storage value does not exist do nothing
else
grp.write('1/1/2', value_2) --GREEN
end
if not value_3 then
--if storage value does not exist do nothing
else
grp.write('1/1/3', value_3) --BLUE
end
```

7.12 Hysteresis

(do not change object 1/1/2 when value of object 1/1/1 is between 100 and 200)

```
value_1 = grp.getvalue('1/1/1') -- byte value
if value_1 < 100 then
grp.write('1/1/2', false) -- bit to 0

elseif value_1 > 200 then
grp.write('1/1/2', true) -- bit to 0
end
```

7.13 Random byte value

```
steps = 255 -- possible steps change this value to lower value to make bigger steps
random = math.random(0, (steps - 1)) * 255 / (steps - 1)
outcome = (math.floor(random))
value_1 = grp.getvalue('1/1/1')
grp.write('1/1/1', outcome) -- Write random byte value to object
```

7.14 Cyclic Repeater (delay 60 seconds)

```
value_1 = grp.getvalue('1/1/1')
if value_1 == true then
repeat
value_1 = grp.getvalue('1/1/1')
if value_1 == true then
grp.write('1/1/2', true)
-- wait for 60 seconds
os.sleep(60)
end
until value_1 == false
end
```

7.15 Stepper / Counter Positive input

```
value_1 = grp.getvalue('1/1/1') -- Positive input
if value_1 == true then
Stepper_Value = storage.get('Value_Stepper_1')
if not Stepper_Value then
Stepper_Value = 0
end
if Stepper_Value == 255 then
else
Stepper_Value = Stepper_Value + 1
end
storage.set('Value_Stepper_1', Stepper_Value)
grp.write('1/1/4', Stepper_Value)
end
```

7.16 Stepper / Counter Negative input

```
value_1 = grp.getvalue('1/1/2') -- Negative input
if value_1 == true then
Stepper_Value = storage.get('Value_Stepper_1')
if not Stepper_Value then
Stepper_Value = 0
end
if Stepper_Value == 0 then
else
Stepper_Value = Stepper_Value - 1
end
storage.set('Value_Stepper_1', Stepper_Value)
grp.write('1/1/4', Stepper_Value)
end
```

7.17 Reset Stepper / Counter

```
value_1 = grp.getvalue('1/1/3')
if value_1 == true then
storage.set('Value_Stepper_1', 0)
grp.write('1/1/4', 0)
end
```

7.18 On Delay (button set to "update only internal")

```
value_1 = grp.getvalue('1/1/1')
if value_1 == true then
os.sleep(3) -- Delay time
grp.write('1/1/1', true)
end
```

7.19 Average

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2')
Average = value_1 + value_2
Average = (Average / 2)
value_3 = grp.getvalue('1/1/3')
grp.write('1/1/3', Average)
```

7.20 Off Delay

```
value_1 = grp.getvalue('1/1/1')
if value_1 == true then
os.sleep(3) -- Delay time
grp.write('1/1/1', false)
end
```

7.21 Stare case timer (with variable time object)

```
value_1 = grp.getvalue('1/1/1')
value_2 = grp.getvalue('1/1/2') -- Variable value
if value_1 == true then
os.sleep(value_2)
grp.write('1/1/1', false)
end
```

7.22 Value memory (write to storage)

```
value_1 = grp.getvalue('1/1/1')
storage.set('Storage_Value_Memory_1', value_1)
```

7.23 Value memory (get from storage)

```
Value_Memory_1 = storage.get('Storage_Value_Memory_1')
if not Value_Memory_1 then
-- do nothing
else
grp.write('1/1/1', Value_Memory_1)
end
```

7.24 Multiplexer (1 in / 3 out) Notice: Object type needs to be the same

```
 value_1 = grp.getvalue('1/1/1')
 grp.write('1/1/2', Value_1)
 grp.write('1/1/3', Value_1)
 grp.write('1/1/4', Value_1)
```

7.25 Round function using Common functions

Add following code to common functions

```
-- Rounds a number to the given number of decimal places...
function round(num, idp)
local mult = 10^(idp or 0)
return math.floor(num * mult + 0.5) / mult
end
```

Create script in script editor

```
-- Round function (with global function)
value_1 = grp.getvalue('1/1/1')
round(value_1, 2)  -- using function round from common functions
grp.write('1/1/1', Value_2)
```

7.26 Write data and time to KNX group addresses

```
-- get current data as table
now = os.date('*t')
-- system week day starts from sunday, convert it to knx format
wday = now.wday == 1 and 7 or now.wday - 1
-- time table
time = {
day = wday,
hour = now.hour,
minute = now.min,
second = now.sec,
}
-- date table
date = {
day = now.day,
month = now.month,
year = now.year,
}
-- write to bus
grp.write('1/1/2', time, dt.time)
grp.write('1/1/1', date, dt.date)
```

7.27 Write data to groups with tags

Create few 1bit group addresses and add tag 'Light' to them
Create one more group different one from the others to trigger script.

1/1/1 – Lihgt1 – Tag 'Light'
1/1/2 – Lihgt2 – Tag 'Light'
1/1/3 – Lihgt3 – Tag 'Light'
1/1/4 – Lihgt4 – Tag 'Light'
1/1/5 – Lihgt5 – Tag 'Light'
1/1/6 – Lihgt6 – Tag 'Light'

1/1/10 – Scene active group –no tag attached!

Create event –based script and attach it to group 1/1/10. Script will run each time group 1/1/10 receive telegram

Add fallowing code to Script editor:

```
AllLights = grp.tag('Light')
AllLights: write(true)
```

All lights will be switched on each time group 1/1/10 receive telegram.

Note: Do not start script from the same tag or group addresses containing the same tag. This will create infinite loop which will generate lots of bus traffic and high load on processor.
If infinite loop is created stop the script and reboot HomeLYnk.