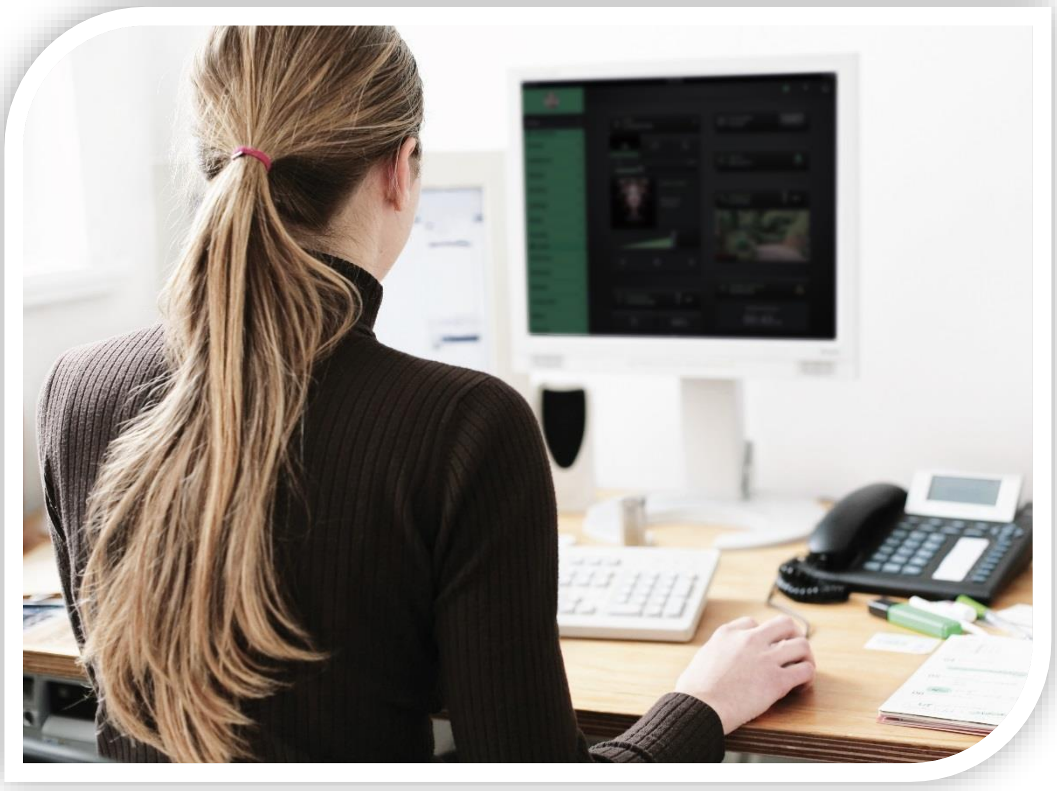
★★☆

Basic / Advanced / Expert

Application note

Programming in Lua

with Wiser for KNX



The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information that is contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.
No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.
All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.
When devices are used for applications with technical safety requirements, the relevant instructions must be followed.
Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.
Failure to observe this information can result in injury or equipment damage.

© 2017 Schneider Electric. All rights reserved

Table of Contents

1 Introduction

This application note describes programming in Lua language build in Wiser for KNX with samples codes covering basic automation needs.

Competencies

This document is intended for readers who have been trained on Wiser for KNX, Wiser for KNX products. The integration should not be attempted by someone who is new to the installation of either product.

Basic technical knowledge on software principles of Lua scripting can be obtained by courses mentioned in chapter ***Reference*** of this document.

2 Lua – Programming Language

2.1 About Lua

Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

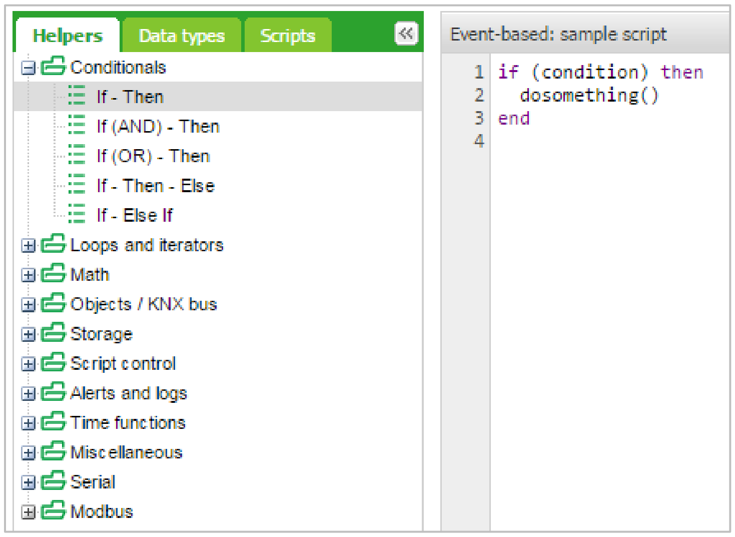
Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection,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

2.2 Lua in Wiser for KNX

Lua in Wiser for KNX provides functions focused on automation needs. Some commands are specific for Wiser for KNX only and thus cannot be used in other Lua enabled devices. Also, scripts made outside Wiser for KNX may need modifications to work properly. See chapter ***Conclusion*** for details.

2.3 Integrated Helpers

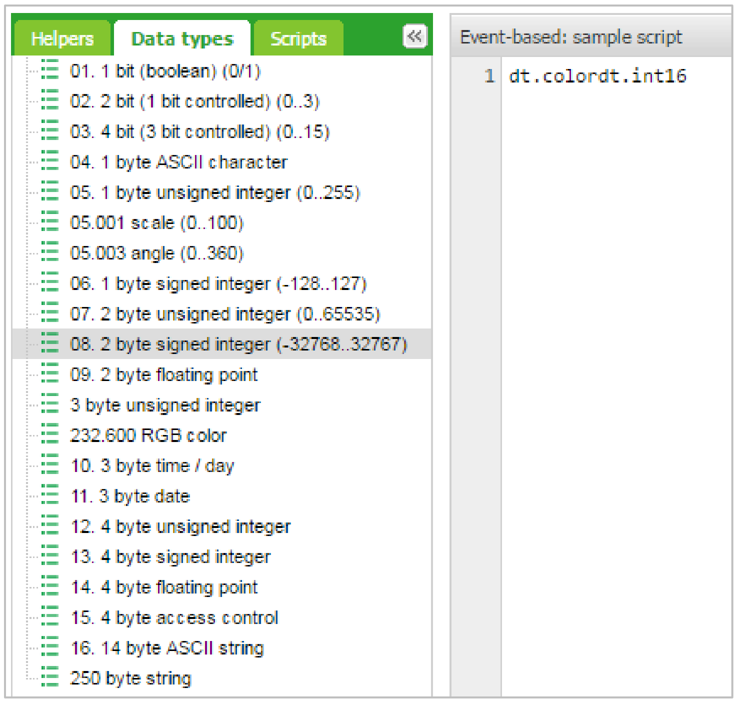
Build-in Helpers is available in in Wiser for KNX script editor. Functions, objects or other snippets can be added to the code simply by clicking on them in tab Helpers.



```
if (condition) then
  dosomething()
end
```

Picture 1 Sample of- adding snippet If-then to the code from Helpers

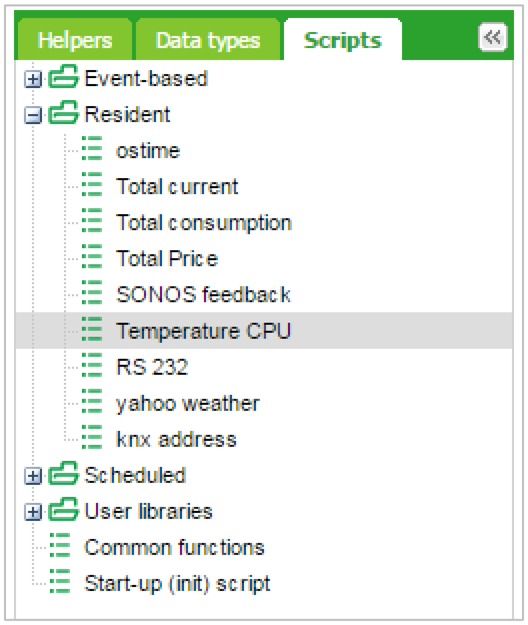
List of supported Data types is also available in tab Data types of Helpers.


Helpers
Data types
Scripts
01. 1 bit (boolean) (0/1)
02. 2 bit (1 bit controlled) (0..3)
03. 4 bit (3 bit controlled) (0..15)
04. 1 byte ASCII character
05. 1 byte unsigned integer (0..255)
05.001 scale (0..100)
05.003 angle (0..360)
06. 1 byte signed integer (-128..127)
07. 2 byte unsigned integer (0..65535)
08. 2 byte signed integer (-32768..32767)
09. 2 byte floating point
3 byte unsigned integer
232.600 RGB color
10. 3 byte time / day
11. 3 byte date
12. 4 byte unsigned integer
13. 4 byte signed integer
14. 4 byte floating point
15. 4 byte access control
16. 14 byte ASCII string
250 byte string
Event-based: sample script

```
dt.colordt.int16
```

Picture 2 Sample of adding 2 byte signed integer snippet from Helpers.

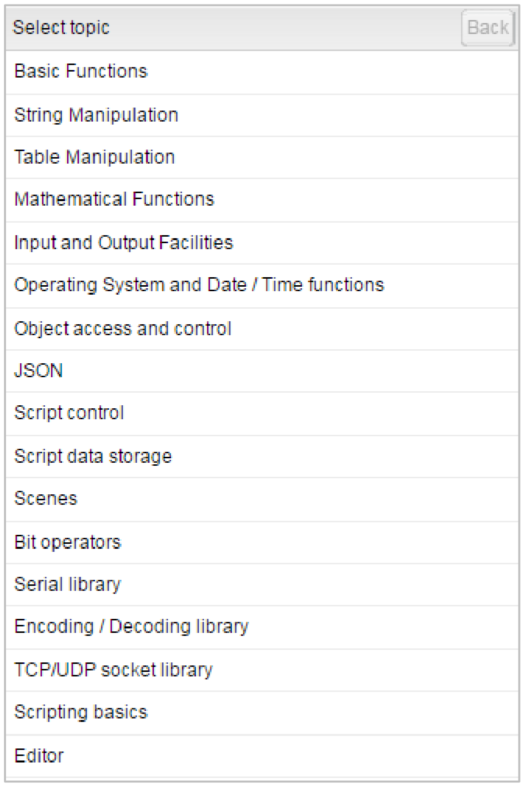
List of already created scripts sorted by their type is available in tab Scripts. Clicking on selected script will open new window in Script editor.



- Event-based
- Resident
  - ostime
  - Total current
  - Total consumption
  - Total Price
  - SONOS feedback
  - Temperature CPU
  - RS 232
  - yahoo weather
  - knx address
- Scheduled
- User libraries
- Common functions
- Start-up (init) script

Picture 3 List of available scripts in Scripts tab.

List of basic functions is available on the right side of Scripting editor.



- Basic Functions
- String Manipulation
- Table Manipulation
- Mathematical Functions
- Input and Output Facilities
- Operating System and Date / Time functions
- Object access and control
- JSON
- Script control
- Script data storage
- Scenes
- Bit operators
- Serial library
- Encoding / Decoding library
- TCP/UDP socket library
- Scripting basics
- Editor

Picture 4 List of available Basic functions

2.4 Object functions

Programming in Lua as scripting language for Wiser for KNX is primary based on the writing functions. Functions in Lua are first-class values with proper lexical scoping meaning functions can access variables of its enclosing functions.

What does it mean for functions to be first-class values? It means that, in Lua, a function is a value with the same rights as conventional values like: numbers and strings. Functions can be stored in variables (both global and local) and in tables. They can be passed as arguments, and can be returned by other functions.

The grp provides simplified access to 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 tab 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 tab 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.

2.5 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 requests to the given alias. Data type is taken from the database if not specified as third parameter. Returns Lua Booolean 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 requests to the given alias.

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 any value to KNX TP bus. It only sends telegrams to KNX IP, when KNX IP features are turned on. Useful for objects that are used in visualization only.

2.6 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 requests 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 requests to object's group address.

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 in visualization only.

2.7 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

2.8 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 unsigned integer – **232.600 RGB color**
- 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

2.9 Data Storage Functions

Storage objects 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.

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

Example:

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

```
result =storage.get('my_stored_value_3') )--return nil if valu eis not
found
result =storage.get('my_stored_value_3', 0)--return nil if valu eis 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
```

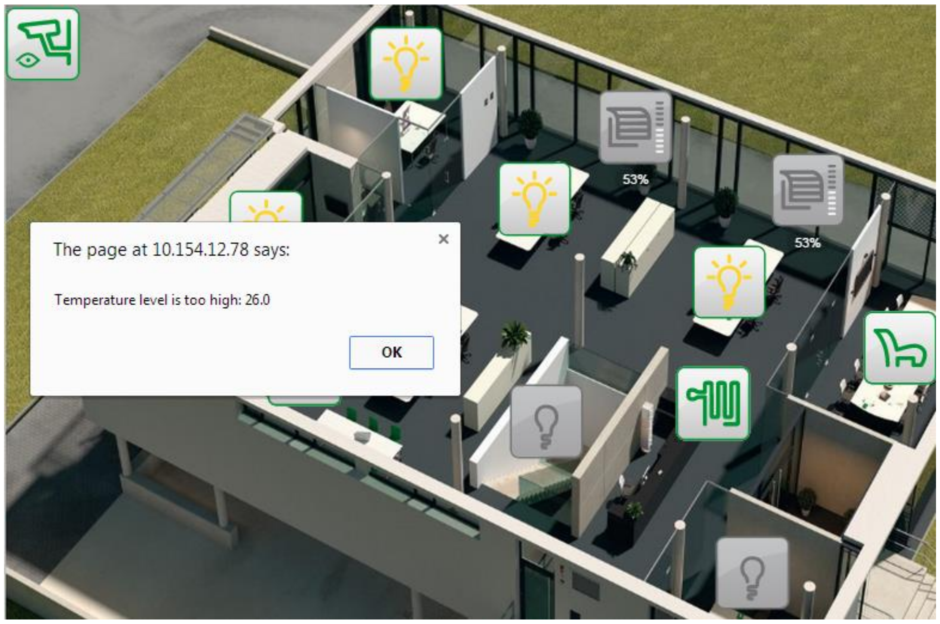
2.10 Alert Functions

```
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 10.154.12.78 says:

Temperature level is too high: 26.0
```

2.11 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
-- log all passed variables log(a, b, c)
```

2.12 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).

2.13 String Manipulation

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 language).

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.

Be aware 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.

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; the 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 the pattern being considered magic.

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 the 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:

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

Example:

```
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, 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 the match returns the captures from the pattern; otherwise, it returns nil. If the 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.

2.14 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 are 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. Henceforth, 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.

2.15 Input and Output Functions

```
io.exists (path)
```

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

```
io.ls(directory)
```

Checks if given 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.readproc(proc)
```

Reads whole process 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.

2.16 Script Control

```
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.

(Same functions available for script category).

2.17 Conversions

Compatibility layer: lmcore is an alias of cnv.

```
lmcore.strtohex (str)
```

Converts given binary string to a hex-encoded string.

```
lmcore.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.

```
lmcore.tonumber (value)
```

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

```
lmcore.hextoint(hexvalue, bytes)
```

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

```
lmcore.inttohex(intvalue, bytes)
```

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

```
lmcore.strtohex(str)
```

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

```
lmcore.hextostr(hexstr)
```

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

2.18 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

2.19 Scene control

Control of scenes as created In Scene tab.

```
scene.run(name)
```

Run selected scene

```
scene.savelive(name)
```

Save actual values of objects assigned to scene.

```
scene.set(name, active)
```

Set scene to active status. (Boolean).

```
scene.disable(name)
```

Disable scene (Boolean).

```
scene.disable(name)
```

Get status of scene (active/disabled).

2.20 Input and Output Facilities

The Input/Otput (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 the 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.

2.21 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 π.

`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 standard. (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.

2.22 Table manipulation

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

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

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

2.23 Operating System and Date / Time functions

`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 the 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

2.24 Extended Function Library

`toBoolean(value)`

Converts the given value to Boolean using the 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.

Compatibility layer: buslib is an alias of former knxlib.

`buslib.decodeia(indaddressa, indaddressb)`

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

`buslib.decodega(groupaddressa, groupaddressb)`

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

`buslib.encodega(groupaddress, separate)`

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

`ipairs (t)`

Returns three values: an iterator function, the table t, and 0, so that the construction will iterate over the pairs (1,t[1]), (2,t[2]), ···, up to the first integer key absent from the table.

Example:

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

`next (table [, index])`

Allows a program to traverse all fields of a table. The first argument is a table, and the 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 behaviour of next is undefined if, during the traversal, you assign any value to a non-existent field in the table. You may however modify the existing fields. In particular, you may clear existing fields.

`pairs (t)`

Returns the three values: the next function, the table t, and nil, so that the construction will iterate over all key–value pairs of table t.

Example:

```
    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".

2.25 TCP/UDP socket library

```
ssl.https.request(url)
require('ssl.https') res, err = ssl.https.request('https://...')
```

For HTTPS requests please use ssl.https.request instead of socket.http.request. This function behaves exactly the same.

3 Script Examples

3.1 Binary Filter

Create two 1-bit 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 the following code to the 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
```

3.2 Binary Gate with Bit Gate

Create three 1-bit 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 the following code to the 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
```

3.3 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 the following code to the 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
```

3.4 Or - Port (2 in 1 Out)

Create three 1-bit 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 the following code to the 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
```

3.5 And - Port (2 in 1 0ut)

Create three 1-bit 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 the following code to the 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
```

3.6 Or - Port (5 in 2 Out)

Create group addresses under Objects 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 the following code to the 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
```

3.7 And - Port (5 in 2 Out)

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 the following code to the 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
```

3.8 Telegram Transformer (0/1 bit to 0-255 byte)

Create two group addresses under Objects tab where:

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

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

Add the following code to the 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
```

3.9 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
```

3.10 Save Scene 1 (RGB value)

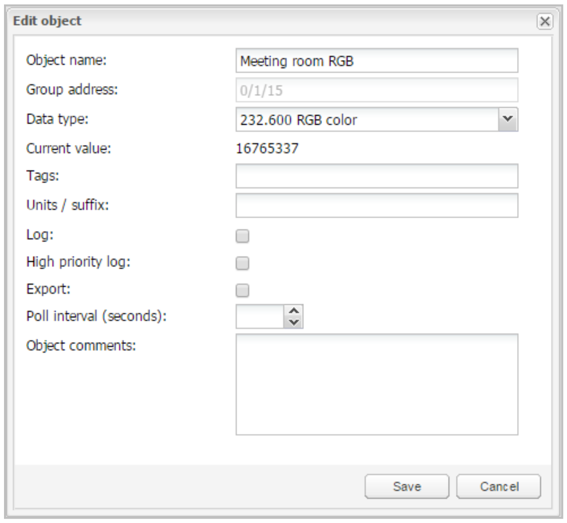
```
value_1 = grp.getvalue('1/1/1') --RED
value_2 = grp.getvalue('1/1/2') --GREEN
value_3 = grp.getvalue('1/1/3') --BLUE
storage.set('Scene1_Red', value_1)
storage.set('Scene1_Green', value_2)
storage.set('Scene1_Blue', value_3)

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
```

3.11 RGB object

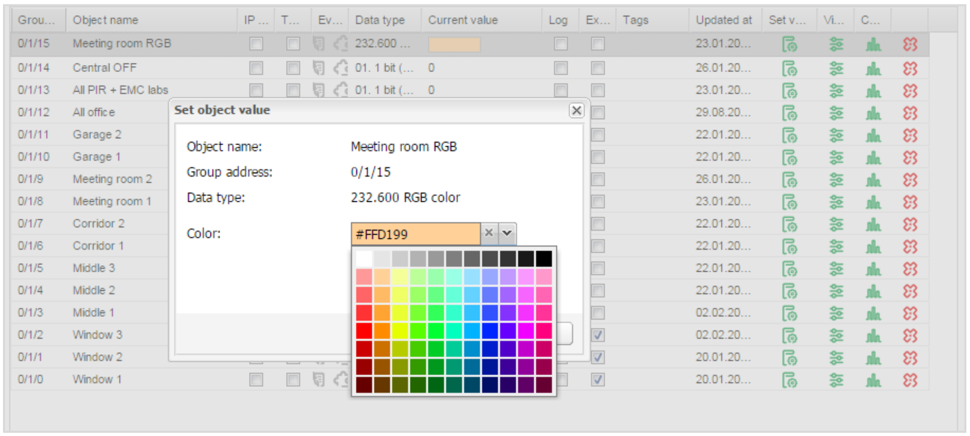
How to operate the RGB objects:

- Create object and define *Object parameters* in *Configurator/Objects/double click on the object*


Edit object

Object name:	Meeting room RGB
Group address:	0/1/15
Data type:	232.600 RGB color
Current value:	16765337
Tags:	
Units / suffix:	
Log:	☐
High priority log:	☐
Export:	☐
Poll interval (seconds):	
Object comments:	

- Set RGB color in Configurator/Objects/Set value



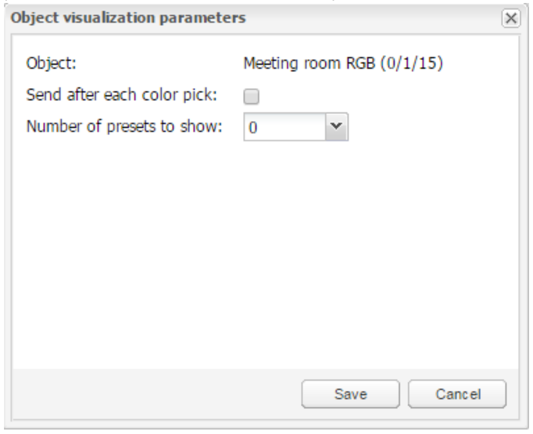
Grou...	Object name	IP ...	T...	Ev...	Data type	Current value	Log	Ex...	Tags	Updated at
0/1/15	Meeting room RGB	☐	☐		232.600 ...	🟧	☐	☐		23.01.20...
0/1/14	Central OFF	☐	☐		01. 1 bit (...	0	☐	☐		26.01.20...
0/1/13	All PIR + EMC labs	☐	☐		01. 1 bit (...	0	☐	☐		23.01.20...
0/1/12	All office							☐		29.08.20...
0/1/11	Garage 2							☐		22.01.20...
0/1/10	Garage 1							☐		22.01.20...
0/1/9	Meeting room 2							☐		26.01.20...
0/1/8	Meeting room 1							☐		23.01.20...
0/1/7	Corridor 2							☐		22.01.20...
0/1/6	Corridor 1							☐		22.01.20...
0/1/5	Middle 3							☐		22.01.20...
0/1/4	Middle 2							☐		22.01.20...
0/1/3	Middle 1							☐		02.02.20...
0/1/2	Window 3							☑		02.02.20...
0/1/1	Window 2							☑		20.01.20...
0/1/0	Window 1	☐	☐					☑		20.01.20...

Set object value

Object name:	Meeting room RGB
Group address:	0/1/15
Data type:	232.600 RGB color
Color:	#FFD199

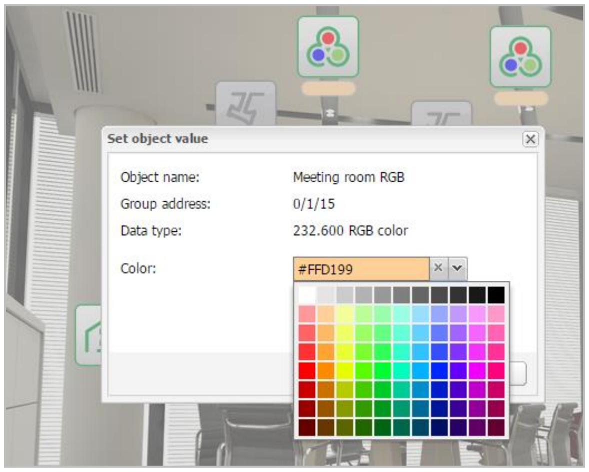
- Set Object visualization parameters in Configurator/Objects/Vis. Params

If the option *Send after each color pick* is ticked, a new updated object with selected color will be automatically sent to bus after releasing the left mouse button (PC) or release finger (touch screen) in the Visualization screen (see PC/Tablet Visualization)


Object visualization parameters

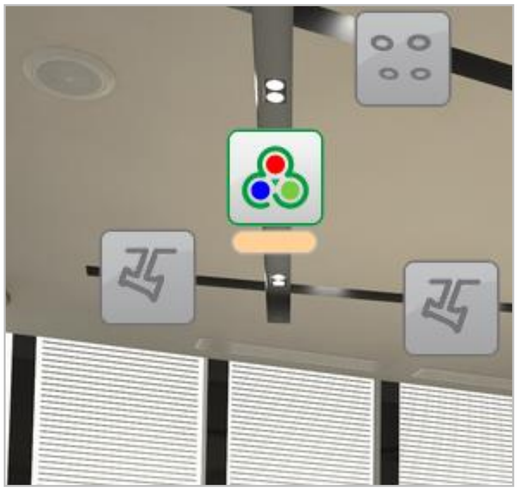
Object:	Meeting room RGB (0/1/15)
Send after each color pick:	☐
Number of presets to show:	0

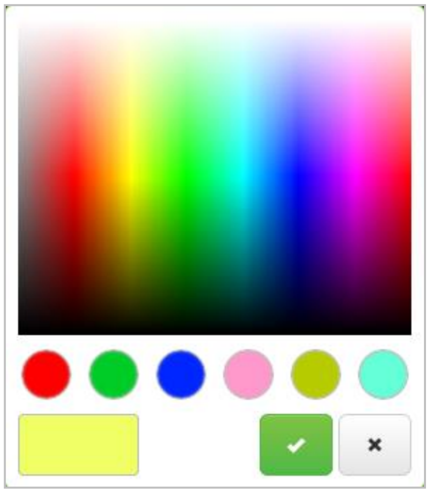
- Set RGB color in Configurator/Visualization (identical to Configurator/Objects/Set value*)*


Set object value

Object name: Meeting room RGB
Group address: 0/1/15
Data type: 232.600 RGB color
Color: #FFD199

- Set color as a User in PC/Tablet Visualization





3.12 Script for controlling RGB LED color

This script splits one 3 byte RGB object into three 1 byte value objects what control the single reg/green/blue colors.

```
-- This script is splitting 1 x 3byte RGB object in to 3 x 1byte value
objects
-- Create event based script which will run from the 3 byte RGB object
-- Fill configurable parameters with group addresses or groups names
-- RGB and 1 byte objects must have correct data type

------------------- Configurable parameters -----------------------------

redGroup   = '1/1/5' -- modify ether group address or name of group
greenGroup = 'LED1 Green Value' -- modify ether group address or name of
group
blueGroup  = '1/1/7' -- modify ether group address or name of group
-----------------------------------------------------------------------------
value = event.getvalue()

Blue= bit.band(value, 0xFF)
Green = bit.rshift(bit.band(value, 0xFF00), 8)
Red = bit.rshift(bit.band(value, 0xFF0000), 16)

grp.write(redGroup, Red, dt.uint8)
grp.write(greenGroup, Green, dt.uint8)
grp.write(blueGroup, Blue, dt.uint8
```

3.13 Script for reading status from the RGB LED

This script is converting 3 x 1byte status objects in to 1 x 3byte RGB object. Create event based script which will run from a unique tag, this tag must be attached to red, green, blue 1-byte status objects. Fill configurable parameters with group addresses or groups names. It is recommended to use 1-byte status object for the inputs. RGB object must have correct data type set.

```
redGroup   = '1/1/1' -- modify ether group address or name of group
greenGroup = 'LED1 Green Status' -- modify ether group address or name of
group
blueGroup  = '1/1/3' -- modify ether group address or name of group
rgbGroup   = 'RGB Value' -- modify ether group address or name of group
red    = grp.find(redGroup)
green  = grp.find(greenGroup)
blue   = grp.find(blueGroup)
redHex = red.datahex
greenHex = green.datahex
blueHex = blue.datahex
RGB = lmcore.hextoint(redHex..greenHex..blueHex)
grp.write(rgbGroup, RGB)
```

3.14 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
```

3.15 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
```

3.16 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
```

3.17 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
```

3.18 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
```

3.19 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
```

3.20 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
```

3.21 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)
```

3.22 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
```

3.23 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
```

3.24 Value Memory (write to storage)

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

3.25 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
```

3.26 Multiplexer (1 in / 3 out)

Objects type must 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)
```

3.27 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)
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
```

3.28 Write Data to Groups with Tags

Create few 1-bit 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 the following code to the Script editor:

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

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

Do not start the script from the same tag or group addresses containing the same tag. This will create an infinite loop, which will generate traffic on a bus and high load on processor.
If infinite loop is created, stop the script and reboot Wiser for KNX.

3.29 Enabling/disabling KNX IP features

This script will enable/disable KNX IP features of Wiser for KNX with restart.

```
require('uci')
-- disable
uci.set('eibd.@eibd[0].knxip=0')
-- enable
uci.set('eibd.@eibd[0].knxip=1')
uci.commit('eibd')
os.execute('/etc/init.d/eibd restart')
```

3.30 Ping checker

Script check if device responses on ping request. Result of the response is saved into selected group address.

```
ip_add= "192.168.0.10" --IP address of device you want check
grp_addr= "0/1/1"   --group address, where result is stored
--*******************************************
ping_result=os.execute('ping -c3 '..ip_add)

if ping_result == 0 then
  grp.write(grp_addr,true)
else
  grp.write(grp_addr,false)
end
```

3.31 Get Wiser for KNX's KNX address

This script will obtain KNX address of Wiser for KNX and send it to the group address (use of virtual address recommended):

```
addr = require('uci').get('eibd.@eibd[0].eibaddr')
grp.write('32/1/2', addr)
```

3.32 Get Wiser for KNX's MAC address

This script will obtain MAC address and send it to the group address (use of virtual address recommended):

```
mac = io.readfile('/sys/class/net/eth0/address'):trim()
grp.write('32/1/2', mac)
```

3.33 Displaying time in short format

This script will display time without seconds. This is useful to lower CPU load and for smooth visualization as seconds in time may be not refreshed evenly.

```
-- obtaining actual data
now = os.date('*t')
-- monday to sunday format
wday = now.wday == 1 and 7 or now.wday -1
time = {
  day = wday,
  hour = now.hour,
  minute = now.min,
  second = now.sec,
  }

date = {
  day = now.day,
  month = now.month,
  year = now.year,
  }
grp.update('1/1/98', date, dt.date)
grp.update('1/1/99', time, dt.time)
```

4 Conclusion

Wiser for KNX operate with Lua version LuaJIT 2.0.4. Always observe compatibility issues when using code made for different devices/software versions. See chapter Installation / Cross-compiling at http://Luajit.org/

5 Appendix

5.1 Reference

Document title	Reference
Lua documentation	https://www.Lua.org/docs.html
Lua Reference manual	https://www.lua.org/manual/5.3/
Lua course	http://Luatut.com/crash_course.html
Learn Lua in 15 minutes	http://tylerneylon.com/a/learn-Lua/
LuaJIT	http://Luajit.org/

Table 1: reference

Schneider Electric Industries SAS

Head Office

35, rue Joseph Monier

92506 Rueil-Malmaison Cedex

FRANCE

www.schneider-electric.com