09 Lua Script

last modified by admin

on 2022/06/08 12:57

Table of Contents

Lua Relevant

Before using the Lua script function, you should go to the Internet to learn relevant grammar and usage tutorials, and master how to use the script

There is the link to the **Lua Official Reference Manual**. For the description of Lua syntax and functions, please refer to the following documents:

- English

Common Function

Here is a list of commonly used functions to facilitate quick search

- basic Basic functions: English
- string String processing: English
- table Table processing: English
- math Mathematical calculation: English

Built-In Library Deletion

Removed the Lua built-in functions (Including libraries), the details are as follows

- Completely remove: debug
- Partial removal:

io	os	basic
io.close	os.clock	~~dofile~~
~~io.flush~~	os.date	~~loadfile~~
~~io.input~~	os.difftime	
~~io.lines~~	~~os.execute~~	
io.open	~~os.exit~~	
~~io.output~~	~~os.getenv~~	
~~io.popen~~	os.remove	
~~io.read~~	os.rename	
~~io.stderr~~	~~os.setlocale~~	
~~io.stdin~~	os.time	
~~io.stdout~~	~~os.tmpname~~	
~~io.tmpfile~~		
io.type		
~~io.write~~		

#Note: The io/os library cannot write to the internal Flash

Lua Extension

On the basis of Lua, combined with the characteristics of HMI, some functional expansions have been made to assist in completing some more complex business requirements

General Class

Function	Overview	i/HMI+ series	ie series	ig series
1.1 Basic Functions	Address Read&Write, screen operations, etc.	√	√	√

1.2 Lua Object Operation	Object status and property control	√	√	√
1.3 Timer	Cycle execution function	×	√	√
1.4 Trigger	Trigger execution function	×	√	√
1.5 Flash Read&Write	Custom files Read&Write	×	√	√
1.6 Other functions	-	√	√	√

Custom Object Class

Realize the drawing and display of charts through **custom parts + functions**

Function	Overview	i/HMI+ series	ie series	ig series
2.1 Table Drawing	-	×	√	√
2.2 Curve Drawing	-	×	√	√

Network Communication

Function	Overview	ig series + series
3.1 MQTT	MQTT Client	√
3.2 Cloud Interface	Get the data of tags from Cloud Function	√
3.3 json	Conversion between json string and lua data	√

Debug

At present, it can only be debugged by the printing function print. There are two situations for debugging.

Category	Overview	i/HMI+ series	ie series	ig series
Simulation debugging	View the print information through the black window created by the simulation operation	√	√	√
HMI debugging	Create a small independent window at the top of the screen to view the print information	×	√	√

How to use HMI debugging

- Open the debug window, add the print_debug(1) in falling script to new Lua object
- Close the debug window, add the print_debug(0) in falling script to new Lua object

#Note:

- When using HMI debugging, if the printing is too frequent and the quantity is large, it may cause the screen to stuck or freeze
- If you have completed the debugging phase of development, you can close the debugging window to reduce the performance consumption of HMI debugging

Coding Environment

Like the Lua standard, the encoding environment of the simulator and the HMI is unified with the encoding of UTF8, including the definition and processing of strings

If you need to print strings in Chinese or other languages into the debugging window, please use we_u8ta for transcoding output

Appendix: Editor Shortcut Hotkeys

Modifier Key	Function Key	Function
Ctrl	A	Select all
Ctrl	B	Insert bit address
Ctrl	C	Copy
Ctrl	D	Copy cursor line
Ctrl	E	When the cursor is inside (outside) a pair of brackets, the cursor s
Shift+Ctrl	E	When the cursor is inside (outside) a pair of brackets, select the te
Ctrl	F	Find
Ctrl	H	Replace
Ctrl	L	Delete cursor line
Shift+Ctrl	L	Lowercase the selected content
Ctrl	S	Save
Ctrl	T	Swap the cursor line and the previous line position
Shift+Ctrl	U	Uppercase the selected content
Ctrl	V	Paste
Ctrl	W	Insert word address
Ctrl	X	Cut
Ctrl	Y	Redo
Ctrl	Z	Undo
None	Arrow key up ↑	Move cursor up
Shift	Arrow key up ↑	Up area selection
Shift+Alt	Arrow key up ↑	Up vertical area selection
Ctrl	Arrow keys up ↑	Window slide up
Shift+Ctrl	Arrow key up ↑	Move selected line up
None	Arrow key down ↓	Move cursor down
Shift	Arrow key down ↓	Down area selection
Shift+Alt	Arrow key down ↓	Down vertical area selection
Ctrl	Arrow key down ↓	Window down
Shift+Ctrl	Arrow key down ↓	Move selected line down
None	Left arrow key ←	Move cursor to the left
Shift	Left arrow key ←	Select a symbol to the left
Shift+Alt	Left Arrow ←	Select a symbol to the left
Ctrl	Left Arrow ←	Cursor cross over a word to the left
Shift+Ctrl	Left Arrow ←	Select a word to the left
None	Right arrow key →	Move the cursor to the right
Shift	Right arrow key →	Select a symbol to the right
Shift+Alt	Right Arrow →	Select a symbol to the right
Ctrl	Right Arrow →	Cursor cross over a word to the right

Shift+Ctrl	Right Arrow →	Select a word to the right
Ctrl	Numeric keyboard plus +	Zoom in the display
Ctrl	Numeric keyboard minus-	Zoom out the display
Ctrl	Slash'?/'	Select content comment
None	Tab	Indent 4 spaces
Shift	Tab	Reduce 4 spaces
None	Home	Move the cursor to the beginning of the line
Shift	Home	Select the cursor position to the beginning of the line
None	End	Move the cursor to the end of the line
Shift	End	Select the cursor position to the end of the line
None	PageUp	The cursor moves up one page according to the size of the windo
Shift	PageUp	Select the cursor position to the previous page
None	PageDown	The cursor moves down one page according to the size of the wir
Shift	PageDown	Select the cursor position to the next page
None	Insert	Switch input mode
		The following delete operation
None	Delete	Delete one symbol backward
Ctrl	Backspace	Delete the content of the word range to the left of the cursor
Ctrl	Delete	Delete the content of the word range to the right of the cursor
Ctrl+Shift	Backspace	Delete all content to the left of the cursor
Ctrl+Shift	Delete	Delete all content to the right of the cursor
		The following paragraph operations, span: paragraph length
Ctrl	[	Move cursor up
Ctrl+Shift	[	Move cursor up and select area
Ctrl	]	Move cursor down
Ctrl+Shift	]	Move cursor down and select area

Basic Function

Address Class

How to read or write address

Function	Description
we_bas_getshort	Read single-word(16bits) signed Decimal
we_bas_setshort	Write single-word(16bits) signed Decimal
we_bas_getword	Read single-word(16bits) unsigned Decimal
we_bas_setword	Write single-word(16bits) unsigned Decimal
we_bas_getint	Read double-word(32bits) signed Decimal
we_bas_setint	Write double-word(32bits) signed Decimal
we_bas_getdword	Read double-word(32bits) unsigned Decimal
we_bas_setdword	Write double-word(32bits) unsigned Decimal
we_bas_getfloat	Read 32bits Floating
we_bas_setfloat	Write 32bits Floating
we_bas_getdouble	Read 64bits Floating
we_bas_setdouble	Write 64bits Floating
we_bas_getbit	Read one bit

we_bas_setbit	Write one bit
we_bas_getstring	Read string
we_bas_setstring	Write string
we_bas_bmov	Continuous address copy
we_bas_fill	Consecutive address assignment
we_bas_newnoaddr	External address offset
we_bas_newstataddr	External station number offset

Path Class

Usage Notice

Function	Description
we_bas_getscriptpath	Get project script path
we_bas_getcustompath	Get the available internal flash path
we_bas_getusbpath	Detect USB
we_bas_getsdpath	Detect SD card
we_bas_file_exist	Does the file exist
we_bas_path_exist	Does the folder exist

Screen Class

Function
we_bas_jumpscreen
we_bas_popscreen
we_bas_closescreen

Others

Function
we_bas_beep
we_bas_gettickcount
we_bas_sleep
we_atu8
we_u8ta
we_bas_getmachineinfo

Address Class

How to read or write address

Let HDW0 store a 32-bit integer value, the macro instruction reads/writes the address like following:

```
@W_HDW0 = 1 'Write 1 to address HDW0
Dim a as integer
a = @W_HDW0 'Read address HDW0 to variable a
```

How to do in Lua:

```
-- The following is the wrong way to write
-- Mistake 1: Attempt to assign a value to a string
"@W_HDW0" = 1 -- The address here is a string, and an error will be reported when compiling the project
-- Mistake 2: Attempt to assign a value to a variable
W_HDW0 = 1 -- The address here is a variable, so the assignment will not be written to the real address
```

```
-- The following is the correct way to write
we_bas_setint("@W_HDW0", 1) - write 1 to the address HDW0
local a = we_bas_getint("@W_HDW0") - read address HDW0 to variable a
```

- Same point: The address is expressed in the same form, but Lua needs to be enclosed in double quotation marks. The address read and write functions, its address parameters are the same as VB, or use the combination key ctrl+W in the editor to insert the word address, ctrl+BInsert bit address
- Different point: In Lua, you need to **call the interface function** to complete the operation address, there is no **equal sign assignment** method like macro instruction

You can find all the interface functions available for reading /writing address operations from this list

we_*bas*_getshort

Prototype: we_*bas*_getshort(address [, type])

Function: Read single-word(16bits) signed Decimal

Parameters:

- address(string):address
- type(number):type, type=1 is for read-through

Return:

- Success: Single word signed decimal value
- Failure: nil

Example:

```
print(we_bas_getshort("@W_HDW0"))
```

we_*bas*_setshort

Prototype: we_*bas*_setshort(address, value [, type])

Function: Write single-word(16bits) signed Decimal

Parameters:

- address(string):address
- value(number):value
- `type(number):type, type=1` is for write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setshort("@W_HDW0", 123)
```

we_*bas*_getword

Prototype: we_*bas*_getword(address [, type])

Function: Read single-word(16bits) unsigned Decimal

Parameters:

- address(string):address
- `type(number):type, type=1` is for read-through

Return:

- Success: single-word(16bits) unsigned Decimal value
- Failure: nil

Example:

```
print(we_bas_getword("@W_HDW0"))
```

we_*bas*_setword

Prototype: we_*bas*_setword(address, value [, type])

Function: Write single-word(16bits) unsigned Decimal

Parameters:

- address(string):address
- value(number):value
- `type(number):type, type` = 1 is for write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setword("@W_HDW0", 123456)
```

we_*bas*_getint

Prototype: we_*bas*_getint(address [, type])

Function: Read double-word(32bits) signed Decimal

Parameters:

- address(string):address
- `type(number):type, type=1` is for read-through

Return:

- Success: Read double-word(32bits) signed Decimal value
- Failure: nil

Example:

```
print(we_bas_getint("@W_HDW0"))
```

we_*bas*_setint

Prototype: we_*bas*_setint(address, value [, type])

Function: Write double-word(32bits) signed Decimal

Parameters:

- address(string):address
- value(number):value
- `type(number):type, type=1` is for write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setint("@W_HDW0", -111)
```

we_*bas*_getdword

Prototype: we_*bas*_getdword(address [, type])

Function: Read double-word(32bits) unsigned Decimal

Parameters:

- address(string):address
- `type(number):type, type=1` is for read-through

Return:

- Success: Read double-word(32bits) unsigned Decimal value
- Failure: nil

Example:

```
print(we_bas_getdword("@W_HDW0"))
```

we_*bas*_setdword

Prototype: we_*bas*_setdword(address, value [, type])

Function: Write double-word(32bits) unsigned Decimal

Parameters:

- address(string):address
- value(number):value
- `type(number):type, type=1`为write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setdword("@W_HDW0", 12345678)
```

we_*bas*_getfloat

Prototype: we_*bas*_getfloat(address [, type])

Function: Read 32bits Floating

Parameters:

- address(string):address
- `type(number):type, type=1` is for read-through

Return:

- Success: 32bits Floating value
- Failure: nil

Example:

```
print(we_bas_getfloat("@W_HDW0"))
```

we_*bas*_setfloat

Prototype: we_*bas*_setfloat(address, value [, type])

Function: Write 32bits Floating

Parameters:

- address(string):address
- value(number):value
- `type(number):type, type=1` is for write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setfloat("@W_HDW0", 123.456)
```

we_*bas*_getdouble

Prototype: we_*bas*_getdouble(address [, type])

Function: Read 64bits Floating

Parameters:

- address(string):address
- `type(number):type, type=lis` for read-through

Return:

- Success: Read 64bits Floating value
- Failure: nil

Example:

```
print(we_bas_getdouble("@W_HDW0"))
```

we_*bas*_setdouble

Prototype: we_*bas*_setdouble(address, value [, type])

Function: Write 64bits Floating

Parameters:

- address(string):address
- value(number):value
- `type(number):type, type=1`为write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setdouble("@W_HDW0", 1.23456)
```

we_*bas*_getbit

Prototype: we_*bas*_getbit(address [, type])

Function: Read one bit

Parameters:

- address(string):address

- `type(number):type, type=lis` for read-through

Return:

- Success: Single-word signed decimal value
- Failure: nil

Example:

```
print(we_bas_getbit("@W_HDW0"))
```

we_*bas*_setbit

Prototype: we_*bas*_setbit(address, value [, type])

Function: Write one bit

Parameters:

- address(string):address
- `value(number):value, value=0` is `off, value=1` is on
- `type(number):type, type=1` is for write-through

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_setbit("@W_HDW0", 1)
```

we_*bas*_getstring

Prototype: we_*bas*_getstring(address, len)

Function: Read string

Parameters:

- address(string):address
- `len(number):length, Unit:` byte

Return:

- Success: the designated length of the string
- Failure: nil

Example:

```
print(we_bas_getstring("@W_HDW0", 10))
```

we_*bas*_setstring

Prototype: we_*bas*_setstring(address, str [, len])

Function: Write string

Parameters:

- address(string):address
- str(string):string
- len(number):length, if not set, it will be the bytes of the string

Return:

- Success: true

- Failure: nil

Example:

```
local str = "Hello"
we_bas_setstring("@W_HDW0", str)
```

we_bas_bmov

Prototype: we_bas_bmov(dstAddress, srcAddress, len)

Function: Copy data of the designated length from the source address to the target address

Parameters:

- dstAddress(string): Target address
- srcAddress(string): Source address
- len(number): designated length

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_bmov("@W_HDW100", "@W_HDW0", 10)
```

we_bas_fill

Prototype: we_bas_fill(address, value, len)

Function: Assign the same value to continuous addresses

Parameters:

- address(string):address
- value(number):value
- len(number):continuous `length (Unit:word)`

Return:

- Success: true
- Failure: nil

Example:

```
we_bas_fill("@W_HDW0", 0, 10)
```

we_bas_newnoaddr

Prototype: we_bas_newnoaddr(address, len)

Function: External address offset

Parameters:

- address(string):address
- len(number):length

Return:

- Success: offset address(string)
- Failure: nil

Example:

```
local x0 = "@W_X0"
for i=1, 10 do
  -- address offset x1 ~ x10
  local x_i = we_bas_newnoaddr(x0, i)
  -- todo
end
```

we_*bas*_newstataddr

Prototype: we_*bas*_newstataddr(address, len)

Function: External station number offset

Parameters:

- address(string):address
- len(number):length

Return:

- Success: offset address(string)
- Failure: nil

Example:

```
-- The address 10 of station No.1 is offset by 2 station numbers
local a = we_bas_newstataddr("@W_1:10", 2)
-- Write 20 to address 10 of station No.3
we_bas_setword(a, 20)
```

Path Class

Usage Notice

Before using this type of interface,you need to understand the structure of the File System inside the HMI.

we_*bas*_getscriptpath

Prototype: we_*bas*_getscriptpath()

Function: Get project script path

Parameters: None

Return:

- Success: The path of script(string)
- Failure: nil

Example:

```
local path = we_bas_getscriptpath()
if path then
  -- todo
end
```

we_*bas*_getcustompath

Prototype: we_*bas*_getcustompath()

Function: Get the available internal flash path

Parameters:None

Return:

- Success: Internal Flash path(string)
- Failure: nil

Example:

```
local path = we_bas_getcustompath()
if path then
  -- todo
end
```

we_bas_getusbpath

Prototype: we_*bas*_getusbpath()

Function: Detect USB, if detected, Return USB path (premise: support U disk Function)

Parameters: None

Return:

- Success: The path of USB
- Failure: nil

Example:

```
local path = we_bas_getusbpath()
if path then
  print("udisk found")
  -- todo
else
  print("udisk not found")
end
```

we_*bas*_getsdpath

Prototype: we_*bas*_getsdpath()

Function: Detect SD card, if detected, return SD card path (premise: support SD card Function)

Parameters: None

Return:

- Success: The path of SD card
- Failure: nil

Example:

```
local path = we_bas_getsdpath()
if path then
  print("sd found")
  -- todo
else
  print("sd not found")
end
```

we_*bas*_file_exist

Prototype: we_*bas*_file_exist(filepath)

Function: Does the file exist

Parameters:

- filepath(string):File path

Return:

- Success: true --exist
- Failure: false --not exist or error

Example:

```
-- Judge whether there is a file 123.txt in the root directory of the U disk
print(we_bas_file_exist("udisk:123.txt"))
```

we_bas_path_exist

Prototype: we_bas_path_exist(dirpath)

Function: Does the folder exist

Parameters:

- dirpath(string): Folder path

Return:

- Success: true --exist
- Failure: false --not exist or error

Example:

```
-- Judge whether there is a folder dir in the root directory of the U disk
print(we_bas_path_exist("udisk:dir"))
```

Screen Class

we_bas_jumpscreen

Prototype: we_bas_jumpscreen(screenid)

Function: Main screen jump

Parameters:

- screenid(number): Main screen No.

Return:

- Success: true
- Failure: false

Example:

```
-- Jump to Screen 0
we_bas_jumpscreen(0)
```

we_bas_popscreen

Prototype: we_bas_popscreen(subscreenid, x, y)

Function: Pop up a sub-screen, if a built-in screen like keyboard pops up, the screen cannot be used

Parameters:

- subscreenid(number): Sub-screen No.
- x(number):The X coordinate of upper left corner of the screen
- y(number):The Y coordinate of upper left corner of the screen

Return:

- Success: true
- Failure: false

Example:

```
-- Pop up a sub-screen, coordinates (100, 100)
we_bas_popscreen(100, 100, 100)
```

we_*bas*_closescreen

Prototype: we_*bas*_closescreen(subscreenid)

Function: Close sub-screen

Parameters:

- subscreenid(number): Sub-screen No.

Return:

- Success: true
- Failure: false

Example:

```
-- Close sub-screen 100
we_bas_closescreen(100)
```

Others

we_*bas*_gettickcount

Prototype: we_*bas*_gettickcount()

Function: Get milliseconds that have passed since the operating system was started

Parameters: None

Return: Time

Example:

```
print(we_bas_gettickcount())
```

we_*bas*_sleep

Prototype: we_*bas*_sleep(millisecond)

Function: `Sleep,` **would block the running of the script**

Parameters:

- millisecond(number): Sleep milliseconds

Return: `None;`

Example:

```
we_bas_sleep()
```

we_*bas*_beep

Prototype: we_*bas*_beep()

Function: Beep

Parameters: None

Return: None

Example:

```
we_bas_beep()
```

we_atu8

Prototype: we_atu8(str)

Function: Character set conversion, windows to utf8

Note: Only effective for the simulator, no conversion in the real HMI

Parameters:

- str(string): string of windows character set

Return:

- Success: string of utf8 character set
- Failure: nil

Example:Reference

we_u8ta

Prototype: we_u8ta(str)

Function: Character set conversion, utf8 to windows

Note: Only effective for the simulator, no conversion in the real HMI

Parameters:

- str(string): string of utf8 character set

Return:

- Success: string of windows character set
- Failure: nil

Example:Reference

we_*bas*_getmachineinfo

Prototype: we_bas_getmachineinfo()

Function: Get HMI machine information

Parameters: None

Return:

- Success: Model, Machine code, Image list
- Failure: nil

Example:

```
local sBoardType, sMachineID, sImageList = we_bas_getmachineinfo()
local sSendData = ""

sSendData = sBoardType .. "\n" .. sMachineID .. "\n" .. sImageList
```

```
print(sSendData)
```

Lua Object Operation

Description

The library is mainly added for **Lua Object** , used to control various states of Lua Object

The prefix name of the library is: **we_*part*_xxx**

#Note:

- Do not use the functions of this type of interface for other types of components, may get unexpected results
- The state of Lua object will be cleared after leaving the screen, and no state will be retained or recorded

List

Function	Description
we_*part*_setstate	Set the state of the object
we_*part*_getstate	Get the state value of the object
we_*part*_settchenable	Set object allowed to click
we_*part*_settchunable	Set object not allowed to click
we_*part*_gettchstate	Get whether the object is allowed to be clicked
we_*part*_sethidestate	Set object hidden value
we_*part*_gethidestate	Get object hidden value
we_*part*_clickdown	Send a click and press message to the specified object, and execute the press script
we_*part*_clickup	Send a click-- up message to the specified part and execute the pop-- up script
we_*part*_setcurrc	Set current outline information of object
we_*part*_getcurrc	Get current outline information of object
we_*part*_getoriginalrc	Get the original outline information of the object

we_*part*_setstate

Prototype: we_*part*_setstate(partname, value)

Function: Set the status of object

Parameter:

- partname(string): object name
- value(number): the status value of the object

Return:

- Success: true
- Failure: false

Example:

```
-- Set the state of part 0_Lua_0 to 1
we_part_setstate("0_Lua_0", 1)
```

we_*part*_getstate

Prototype: we_*part*_getstate(partname)

Function: Get the status value of the object

Parameter:

- partname(string): object name

Return:

- Success: Status value (number)
- Failure: nil

Example:

```
print(we_part_getstate("0_Lua_0"))
```

we_*part*_settchenable

Prototype: we_*part*_settchenable(partname)

Function: Set the object allowed to click

Parameter:

- partname(string): object name

Return:

- Success: true
- Failure: false

Example:

```
print(we_part_settchenable("0_Lua_0"))
```

we_*part*_settchunable

Prototype: we_*part*_settchunable(partname)

Function: Set object not allowed to click

Parameter:

- partname(string): object name

Return:

- Success: true
- Failure: false

Example:

```
print(we_part_settchunable("0_Lua_0"))
```

we_*part*_gettchstate

Prototype: we_*part*_gettchstate(partname)

Function: Get whether the object is allowed to be clicked

Parameter:

- partname(string): object name

Return:

- Success: true allowed to click, false not allowed to click
- Failure: nil

Example:

```
print(we_part_gettchstate("0_Lua_0"))
```

we_*part*_sethidestate

Prototype: we_*part*_sethidestate(partname, value)

Function: set the hidden value of the object

Parameter:

- partname(string): object name
- value(number): set value
  0: No special processing is performed, according to the Hide function controlled by the object's control address
  1: Display and not be affected by the Hide function of the object's control address
  2: Hidden and not affected by the Hide function of the object's control address

Return:

- Success: true
- Failure: false

Example:

```
-- hide
we_part_sethidestate("0_Lua_0", 2)
```

we_*part*_gethidestate

Prototype: we_*part*_gethidestate(partname)

Function: Get the hidden value of the object

Parameter:

- partname(string): object name

Return:

- Success: true object is hidden, false object is displayed
- Failure: nil

we_*part*_clickdown

Prototype: we_*part*_clickdown(partname)

Function: Send a click-- down message to the specified component, and execute the click-- down script

Parameter:

- partname(string): object name

Return:

- Success: true
- Failure: false

we_*part*_clickup

Prototype: we_*part*_clickup(partname)

Function: Send a click-- up message to the specified component, and execute the click-- up script

Parameter:

- partname(string): object name

Return:

- Success: true
- Failure: false

we_*part*_setcurrc

Prototype: we_*part*_setcurrc(partname, left, top, right, bottom)

Function: Set the current contour info of the object

Parameter:

- partname(string): object name
- left(number): coordinate value of the left side of the object
- top(number): coordinate value of the left side of the object
- right(number): coordinate value of the right side of the object
- bottom(number): coordinate value of the bottom of the object

Return:

- Success: true
- Failure: false

Example:

```
-- Move the object to the upper left corner, and the size is 100x100
we_part_setcurrc("0_Lua_0", 0, 0, 100, 100)
```

we_*part*_getcurrc

Prototype: we_*part*_getcurrc(partname)

Function: Get the current contour info of the object

Parameter:

- partname(string): object name

Return:

- Success: left, top, right, bottom (number), four values
- Failure: nil

Example:

```
local left, top, right, bottom = we_part_getcurrc("0_Lua_0")
print(left, top, right, bottom)
```

we_*part*_getoriginalrc

Prototype: we_*part*_getoriginalrc(partname)

Function: Get the original contour info of the object

Parameter:

- partname(string): object name

Return:

- Success: left, top, right, bottom (number), four values
- Failure: nil

Example:

```
local left, top, right, bottom = we_part_getoriginalrc("0_Lua_0")
print(left, top, right, bottom)
```

Timer

Description

The timer function is similar to the timing script of the Macro, which has the effect of timing execution. The difference is that Macro executes one script, while Lua executes one function.

The name of the library is: **timer**

Features

- Trigger execution when timing arrives
- Dynamic switching enable, can disable or enable during operation
- The timing interval is variable, and the interval value can be modified during operation

#Note:

- If multiple timers are registered in one screen, they will be executed out of order, and timing arrives will be executed first
- The function registered by the timer cannot be written in an endless loop, otherwise it will be stuck in all operations, including the execution of screen polling **we*scr*poll_x**

List

Function	Description
timer.create	Create timer
timer.set_status	Enable timer
timer.get_status	Get timer status
timer.set_interval	Modify timer interval value
timer.get_interval	Get timer interval value
timer.get_func	Get the function associated with the timer

Parameter Requirements:

The value range of **screen number** in the function parameters is:

- -1: Global background
- [0,99999]: the corresponding screen number

The **ID** in the function parameters must be unique in every screen, and also unique in the global, but each screen is independent of each other. For example, a timer with ID=[1,2,3] is created on screen 0, screen 1 can also have a timer with ID=[1,2,3], and the global background can also have ID=[1,2, 3] these timers are independent and unique

timer.create

Prototype: timer.create(screenid, id, interval, f)

Function: Create a timer, execute the registered function when the timer arrives, and automatically start to run after it is created by default

Parameter:

- screenid(number): screen number
- id(number): ID
- interval(number): interval
- f(function): registered execution function

Return:

- Success: true
- Failure: nil

Example:

```
-- Method 1: On screen 0, create a timer with ID=1 and an interval of 1000ms
timer.create(0, 1, 1000, function() print("Hello world") end)
-- Method 2: Write the function first, then register it
function callback() print("Timer called") end
-- Similarly, in screen 0, pay attention to ID=2 at this time, it cannot be repeated, and the function name can be used directly for the registration function
timer.create(0, 2, 1000, callback)
-- Create a timer for global execution, which should follow the unique ID
timer.create(-- 1, 1, 1000, function() print("This is global timer test.") end)
```

timer.set_status

Prototype: timer.set_status(screenid, id, value)

Function: enable timer

Parameter:

- screenid(number): screen number
- id(number): ID
- value(number): value, 0: off, 1: on

Return:

- Success: true
- Failure: nil

Example:

```
-- Turn off the timer
timer.set_status(0, 1, 0)
-- Turn on the timer
timer.set_status(0, 1, 1)
```

timer.get_status

Prototype: timer.get_status(screenid, id)

Function: Get timer status

Parameter:

- screenid(number): screen number
- id(number): ID

Return:

- Success: 0: off, 1: on (number)
- Failure: nil

Example:

```
-- Get the status of the timer
local status = timer.get_status(0, 1)
print(status)
```

timer.set_interval

Prototype: timer.set_interval(screenid, id, num)

Function: modify the interval value of the timer

Parameter:

- screenid(number): screen number
- id(number): ID
- num(number): value, unit: ms

Return:

- Success: true
- Failure: nil

Example:

```
-- Change the interval value of screen 0, ID=1 to 2000ms
timer.set_interval(0, 1, 2000)
```

timer.get_interval

Prototype: timer.get_interval(screenid, id)

Function: Get the interval value of the timer

Parameter:

- screenid(number): screen number
- id(number): ID

Return:

- Success: Interval value (number)
- Failure: nil

Example:

```
local val = timer.get_interval(0, 1)
print(val)
```

timer.get_func

Prototype: timer.get_func(screenid, id)

Function: Get the execution function associated with the timer

Parameter:

- screenid(number): screen number
- id(number): ID

Return:

- Success: function
- Failure: nil

Example:

```
local f = timer.get_func(0, 1)
-- Print function address
print(f)
-- or execute function
f()
```

Trigger

Description

The trigger function is similar to the trigger script of Macro, which has the effect of triggering execution. The difference is that Macro executes a script, while Lua executes a function.

The name of the library is: **trigger**
Support 5 trigger types:

1. TRUE: trigger.type_true
2. FALSE: trigger.type_false
3. Bit change: trigger.type_change
4. Rising edge: trigger.type_raise
5. Falling edge: trigger.type_fall

Features

- Bit address condition triggers execution
- Dynamic switching enable, which can enable or disable during operation
- Variable trigger conditions, trigger conditions can be modified during operation

#Note:

- If multiple triggers are registered in one screen, the address monitoring will be executed out of order
- The function registered by the trigger cannot be written in an endless loop, otherwise it will be stuck in all operations, including the execution of screen polling **we*scr*poll_x**

List

Function	Description
trigger.create	Create Trigger
trigger.set_status	Enable trigger
trigger.get_status	Get trigger status
trigger.set_type	Modify the trigger type of the trigger
trigger.get_type	Get the trigger type of the trigger
trigger.get_func	Get the execution function associated with the trigger

Parameter Requirements:

The value range of **screen number** in the function parameters is:

- -1: Global background
- [0,99999]: the corresponding screen number

The **ID** in the function parameters must be unique in every screen, and also unique in the global, but each screen is independent of each other. For example, a trigger with ID=[1,2,3] is created on screen 0, screen 1 can also have a trigger with ID=[1,2,3], and the global background can also have ID=[1,2, 3] , these triggers are independent and unique

trigger.create

Prototype: trigger.create(screenid, id, triggertype, address, f)

Function: Create a trigger, execute the registered function when the trigger condition is met, and automatically start monitoring after it is created by default

Parameter:

- screenid(number): screen number
- id(number): ID
- triggertype(number): trigger condition type

- address(string): the bit address to be monitored and triggered
- f(function): registered execution function

Return:

- Success: true
- Failure: nil

Example:

```
-- Method 1: On screen 0, create a trigger with ID=1, type TRUE, and address HDX0.0
trigger.create(0, 1, trigger.type_true, "@B_HDX0.0", function() print("Hello world") end)
-- Method 2: Write the function first, then register
function callback() print("trigger called") end
-- Similarly, in screen 0, pay attention to ID=2 at this time, it cannot be repeated, and the function name can be used directly for the registration function
trigger.create(0, 2, trigger.type_true, "@B_HDX0.1", callback)
-- Create a global execution bit change trigger, which should follow the unique ID
trigger.create(-- 1, 1, trigger.type_change, "@B_HDX0.2", function() print("This is global trigger test.") end)
```

trigger.set_status

Prototype: trigger.set_status(screenid, id, value)

Function: Enable trigger

Parameter:

- screenid(number): screen number
- id(number): ID
- value(number): value, 0: off, 1: on

Return:

- Success: true
- Failure: nil

Example:

```
-- Enable the trigger
trigger.set_status(0, 1, 0)
-- Disable the trigger
trigger.set_status(0, 1, 1)
```

trigger.get_status

Prototype: trigger.get_status(screenid, id)

Function: Get trigger status

Parameter:

- screenid(number): screen number
- id(number): ID

Return:

- Success: 0: off, 1: on(number)
- Failure: nil

Example:

```
-- Get the status of the trigger
local status = trigger.get_status(0, 1)
print(status)
```

trigger.set_type

Prototype: trigger.set_type(screenid, id, num)

Function: modify the trigger type of the trigger

Parameter:

- screenid(number): screen number
- id(number): ID
- num(number):type value

Return:

- Success: true
- Failure: nil

Example:

```
-- Change the trigger type of screen 0, ID=1 to false
trigger.set_type(0, 1, trigger.type_false)
```

trigger.get_type

Prototype: trigger.get_type(screenid, id)

Function: Get the trigger type of the trigger

Parameter:

- screenid(number): screen number
- id(number): ID

Return:

- Success: trigger type value (number)
- Failure: nil

Example:

```
local val = trigger.get_type(0, 1)
print(val)
```

trigger.get_func

Prototype: trigger.get_func(screenid, id)

Function: Get the execution function associated with the trigger

Parameter:

- screenid(number): screen number
- id(number): ID

Return:

- Success: function
- Failure: nil

Example:

```
local f = trigger.get_func(0, 1)
-- Print function address
print(f)
-- or execute function
f()
```

Flash Read&Write

Description

Lua's io/os library doesn't support operating the internal Flash of the HMI. There is another interface for Flash that can implement common operations for files and folders.

The name of the library is: **flash**

The path format of the system is drive letter: location, and the available drive letters have the following 4 types:

- User directory: "user:"
- User script: "script:"
- U Disk: "udisk:"
- SD card: "sdcard:"

The following shows how to express the path:

- Example 1: If a file named 1.txt exists in the root directory of the U disk, the path expression is: udisk:1.txt
- Example 2: If a folder named dir exists in the root directory of the U disk, the path expression is: udisk:dir

#Note:

- In order to prolong the service life, there is a limited control for writing, and only 1M is allowed to be written within 1 minute. If it exceeds, an error message will be prompted
- The storage space is limited. This part of the space for reading and writing in Lua is shared with Data recording, Historical data and other functions. If the recording data is full, it may be impossible to use Lua scripts to write files, and vice versa

List

Function	Description
flash.file_write	Write file
flash.file_copy	Copy file
flash.file_rename	Rename file
flash.file_remove	Delete file
flash.file_filter	Traverse folders
flash.dir_make	Recursively create folders
flash.dir_remove	Recursively delete folders

flash.file_write

Prototype: flash.file_write(filepath, mode, data)

Function: Write data to a file, and you can write a file to the internal Flash through this interface

Parameter:

- filepath(string): file path
- mode(string): open mode, it can be non-- r mode,
- data(string): the data to be written

Return:

- Success: true
- Failure: nil

Example:

```
-- Write to internal Flash
flash.file_write("user:demo.txt", "w", "Hello world")
-- Write to U disk
```

```
flash.file_wrtie("udisk:export.txt", "w", "This is a demo file")
```

flash.file_copy

Prototype: flash.file_copy(srcpath, dstpath, failifexists)

Function: Copy files, you can copy files from the peripheral to the internal Flash through this interface, and can also be used to export

Parameter:

- srcpath(string): source file path
- dstpath(string): target file path
- failifexists(boolean): whether it fails if the target file already exists
  - true if the target file exists, copy fails
  - false If the target file exists, overwrite copy

Return:

- Success: true
- Failure: nil

Example:

```
-- Assuming that the U disk has a demo.txt file, copy it to the HMI and perform an overwrite copy
flash.file_copy("udisk:demo.txt", "user:demo.txt", false)
-- The following is the export
flash.file_copy("user:demo.txt", "udisk:demo2.txt", true)
```

flash.file_rename

Prototype: flash.file_rename(oldname, newname)

Function: Rename file

Parameter:

- oldname(string): old file name path
- newname(string): new file name path

Return:

- Success: true
- Failure: false

Example:

```
flash.file_rename("udisk:old.txt", "udisk:new.txt")
```

flash.file_remove

Prototype: flash.file_remove(filepath)

Function: delete files

Parameter:

- filepath(string): file name path

Return:

- Success: true
- Failure: false

Example:

flash.file_remove("udisk:new.txt")

flash.file_filter

Prototype: flash.file_filter(dirpath, format)

Function: traverse folders and return all file names that conform to the file format

Parameter:

- dirpath(string): folder path
- format(string): file format

Return:

- Success: an array of file names (table)
- Failure: nil

Example:

```
-- Assume that the root directory of the U disk has the following files
-- 123.txt|sheet.csv|demo.txt|zzz.txt|myword.doc
-- Execute to get all files in txt format
local data = flash.file_filter("udisk:", "*.txt")
if data and type(data) == "table" then
  for i,v in ipairs(data) do
    print(i, v)
  end
end
```

Output:

```
1 123.txt
2 demo.txt
3 zzz.txt
```

flash.dir_make

Prototype: flash.dir_make(dirpath)

Function: Recursively create folders

Parameter:

- dirpath(string): folder path

Return:

- Success: true
- Failure: false

Example:

```
flash.dir_make("udisk:dir1/dir2/dir3")
```

flash.dir_remove

Prototype: flash.dir_remove(dirpath)

Function: delete folders recursively, **including files in the folder**

Parameter:

- dirpath(string): folder path

Return:

- Success: true
- Failure: false

Example:

```
-- Assuming that there are dir1/dir2/dir3, three-- level folders in the U disk, specify dir1 to clear all
flash.dir_remove("udisk:dir1")
```

flash.file_u8ta

Prototype: flash.file_u8ta(u8file, gbfile)

Function: File character set conversion, UTF8 to GB2312

Parameter:

- u8file(string): utf8 file to convert
- gbfile(string): Save gb2312 file of converted data (**Only files in a Udisk or sd card supported)**

Return:

- = 0：Succeed
- =-1：Function parameter error
- =-2：File path error
- =-3：Failed to read utf8 file data
- =-4：utf8 file data exception
- =-5：Insufficient space
- =-6：Data conversion failure
- =-7：Failed to write the GB2312 file

Example:

```
--Convert utF8 files in the Udisk to GB2312 and save the files in the Udisk
flash.file_u8ta("udisk:utf8.txt", "udisk:new_gb2312.txt")
```

flash.file_atu8

Prototype: flash.file_atu8(gbfile, u8file)

Function: File character set conversion, UTF8 to GB2312

Parameter:

- u8file(string): gb2312 file to convert
- gbfile(string): Save utf8 file of converted data (**Only files in a Udisk or sd card supported)**

Return:

- = 0：Succeed
- =-1：Function parameter error
- =-2：File path error
- =-3：Failed to read gb2312 file data
- =-4：utf8 file data exception
- =-5：Insufficient space
- =-6：Data conversion failure
- =-7：Failed to write the utf8 file

Example:

```
--Convert GB2312 files in the Udisk to utF8 and save the files in the Udisk
flash.file_atu8("udisk:gb2312.txt", "udisk:new_u8.txt")
```

Other Functions

Description

The library provides operations for folder traversal, creation, and deletion

The name of the library is: **lfs**

List

Function	Description
lfs.attributes	Get properties
lfs.dir	Traverse folders
lfs.mkdir	Create folder
lfs.rmdir	Delete folder

lfs.attributes

Prototype: lfs.attributes(filepath [, aname])

Function: Get properties and return a table with file properties corresponding to the file path. If the second parameter is given, only the value of the naming property will be returned

Parameter:

- filepath(string): file path
- aname(string):File attribute

Return:

- Success: table
- Failure: nil

Example:

```
-- Judge whether the format of the specified file is a text file
local filemode = lfs.attributes("udisk:123.txt", "mode")
if filemode == "file" then
  print("udisk:123.txt is a normal txt file")
end
```

lfs.dir

Prototype: lfs.dir(dirpath)

Function: traverse the folder, this is an iterator function, need to use the for statement

Note: This interface function is planned to be deprecated in the future, it is recommended to use the flash.file_filter function

Parameter:

- dirpath(string): folder path

Return:

- Success: file name (string)
- Failure: nil

Example:

```
-- Example of traversing folders
local mydir = "udisk:"
```

```
for fn in lfs.dir(mydir) do
  if fn ~= "." and fn ~= ".." then
    local filepath = mydir .. fn
    local attr = lfs.attributes(filepath)
 --  The simulator window needs the windows character set to display Chinese
    --  Print the full file path and its file properties
    print(we_u8ta(filepath), attr.mode)
  end
end
```

lfs.mkdir

Prototype: lfs.mkdir(dirpath)

Function: Create a folder (recursion is not supported)

Notice: This interface function is planned to be deprecated in the future, it is recommended to use the flash.dir_make function

Parameter:

- dirpath(string): folder path

Return:

- Success: true
- Failure: nil

Example:

```
-- Create the parent folder first, then create the sub folders
if lfs.mkdir("udisk:mydir") then
 lfs.mkdir("udisk:mydir/subdir")
end
```

lfs.rmdir

Prototype: lfs.rmdir(dirpath)

Function: Delete folder (recursion is not supported)

Note: This interface function is planned to be deprecated in the future. It is recommended to use the flash.dir_remove function

Parameter:

- dirpath(string): folder path

Return:

- Success: true
- Failure: nil

Example:

```
-- Remove the sub folder first, then remove the parent folder
if lfs.rmdir("udisk:mydir/subdir") then
   lfs.rmdir("udisk:mydir")
end
```

Table Drawing (Experimental)

Description

The name of the library is: **cus_table**

The appearance of the style cannot be modified at current

The following interface example assumes that there is a object named 0_CST_0 on screen 0

#Note:

- It is currently an experimental function. If there is any modification in the future, please read the interface documentation carefully
- The status of the parts will be cleared after leaving the screen, and no status will be retained or recorded

List

Function	Description
cus_*table.set*_cell	Set the number of rows and columns of the table
cus_*table.set*_head	Set table header text
cus_*table.set*_mode	Set display mode
cus_*table.page*_up	Execute page up
cus_*table.page*_down	Execute page down
cus_*table.get*_index	Get the currently selected index
cus_*table.get*_page	Get the current page and the total number of pages
cus_table.refresh	Refresh object
cus_table.reset	Reset object
Example	

cus_*table.set*_cell

Prototype: cus_*table.set*_cell(part, row, col[, ratio])

Function: Set the number of rows and columns of the table

- part(string): object name
- row(number): number of rows
- col(number): number of columns
- ratio(string): optional parameter (default equal width), the ratio of column width

Return:

- Success: true
- Failure: nil, error message

Example:

```
-- Set the table with 20 rows and 6 columns. The width of the first column is twice times than that of the second column, and the second, third, fourth, fifth, and sixth columns are equal in width
cus_table.set_cell("0_CST_0", 20, 6, "2:1:1:1:1:1")
```

cus_*table.set*_head

Prototype: cus*table.set*head(part, head)

Function: set header text

- part(string): object name
- head(table): array, head[1] position represents column 1, and so on

Return:

- Success: true
- Failure: nil, error message

Example:

```
-- Set the header text of these 6 columns
local head = {
    [1] = "c1",
    [2] = "c2",
    [3] = "c3",
    [4] = "c4",
    [5] = "c5",
    [6] = "c6",
}
cus_table.set_head("0_CST_0", head)
```

cus_*table.set*_mode

Prototype: cus_*table.set*_mode(part, mode)

Function: Set the display mode, you can display in ascending or descending order, the default is ascending order

- part(string): object name
- mode(number): 0 ascending order, 1 descending order

Return:

- Success: true
- Failure: nil, error message

Example:

```
-- Set to display in descending order
cus_table.set_mode("0_CST_0", 1)
```

cus_*table.page*_up

Prototype: cus_*table.page*_up(part)

Function: execute page up

- part(string): object name

Return:

- Success:
    - true turn page up successfully
    - false has reached the top page
- Failure: nil, error message

Example:

```
cus_table.page_up("0_CST_0")
```

cus_*table.page*_down

Prototype: cus_*table.page*_down(part)

Function: Execute page down

- part(string): object name

Return:

- Success:
    - true turn page down successfully
    - false to the bottom page
- Failure: nil, error message

Example:

```
cus_table.page_down("0_CST_0")
```

cus_*table.get*_index

Prototype: cus_*table.get*_index(part)

Function: Get the currently selected index

- part(string): object name

Return:

- Success: Index (number)
- Failure: nil, error message

Example:

```
print(cus_table.get_index("0_CST_0"))
```

cus_*table.get*_page

Prototype: cus_*table.get*_page(part)

Function: Get the current page and the total number of pages

- part(string): object name

Return:

- Success: current page (number), total page number (number)
- Failure: nil, error message

Example:

```
print(cus_table.get_page("0_CST_0"))
```

cus_table.refresh

Prototype: cus_table.refresh(part, data)

Function: Refresh the object, after performing page turning and modifying the data source, you need to call this function once to refresh the object display

- part(string): object name
- data(table): data source, two-- dimensional array, data[1][1] position represents the content of the first row and the first column

Return:

- Success: true
- Failure: nil, error message

Example:

```
local t = {
    [1] = {1,2,3,4,5,6}, --  Row 1
    [2] = {2,3,4,5,6,7}, -- Row 2
    [3] = {3,4,5,6,7,8},
}
cus_table.refresh("0_CST_0", t)
```

cus_table.reset

Prototype: cus_table.reset(part)

Function: reset object, clear all settings

- part(string): object name

Return:

- Success: true
- Failure: nil, error message

Example:

```
cus_table.reset("0_CST_0")
```

Example

Use Lua object, add table_test() in the press script, compile and run to see the effect

```
function table_test()
  local head = {
      [1] = "order",
      [2] = "c2",
      [3] = "c3",
      [4] = "c4",
      [5] = "c5",
      [6] = "c6",
   }
  local t = {
      [1] = {1,2,3,4,5,6}, --  The first row, the table has 6 columns, so all need 6 data
      [2] = {2,3,4,5,6,7},
      [3] = {3,4,5,6,7,8},
   }
  local part = "0_CST_0"
   cus_table.set_cell(part, 10, 6, "2:1:1:1:1:1")
   cus_table.set_head(part, head)
   cus_table.refresh(part, t)
  print(cus_table.get_index(part))
  print(cus_table.get_page(part))
end
```

Curve Drawing (Experimental)

Description

The name of the library is: **cus_curve**

-- This is a unified style sheet for the curve, used to control the display of the curve local style = { color = 0xff0000, -- color [0,0xffffff]: red linetype = 0, -- Line type [0,3]: 0 solid line, 1 dotted line, 2 short dotted lines, 3 long dotted lines width = 1, -- line width [1,9] visible = 1,-- visible attribute [0,1]: 0 is invisible, 1 is visible }

The following interface example assumes that there is a object named 0_CST_0 on screen 0

#Note:

- It is currently an experimental function. If there is any modification in the future, please read the interface documentation carefully
- The status of the object will be cleared after leaving the screen, and no status will be retained or recorded
- You can draw up to 5 curves, 5 x-- axis reference lines, and 5 y-- axis reference lines at the same time

List

Function	Description
cus_curve.init	Set the number of x and y major scale divisions
cus_*curve.set*_range	Set the range of the specified axis
cus_*curve.get*_range	Get the range of the specified axis
cus_*curve.set*_scale	Set the major scale style of the specified axis
cus_*curve.get*_scale	Get the major scale style of the specified axis
cus_*curve.set*_grid	Set the main grid style of the specified axis
cus_*curve.get*_grid	Get the main grid style of the specified axis
cus_*curve.set*_line	Set the point and style of the curve
cus_*curve.get*_line	Get the point and style of the curve
cus_*curve.set*_visible	Set the visibility of the curve
cus_*curve.get*_visible	Get the visibility of the curve
cus_*curve.set*_refer	Set a reference line
cus_*curve.set*_cursor	Set the display of the click event cursor
cus_*curve.draw*_line	Draw a curve
cus_curve.refresh	Refresh object
cus_curve.reset	Reset object
Example	

cus_curve.init

Prototype: cus_curve.init(part, x, y)

Function: Set the number of x and y major scale divisions

- part(string): part name
- x(number): The number of major scale on the x-- axis [1, 10]
- y(number): the number of major scale on the y-- axis [1, 10]

Return:

- Success: true
- Failure: nil, error message

Example:

```
-- Set 3x3 major scale
cus_curve.init("0_CST_0", 3, 3)
```

cus_*curve.set*_range

Prototype: cus_*curve.set*_range(part, axis, min, max)

Function: Set the range of the specified axis

- part(string): object name
- axis(string): specify the axis, x or y
- min(number): minimum value
- max(number): maximum value

Return:

- Success: true
- Failure: nil, error message

Example:

```
-- Set the x-- axis range [1,4]
```

```
cus_curve.set_range("0_CST_0", "x", 1, 4)
-- Set the y-- axis range [1,4]
cus_curve.set_range("0_CST_0", "y", 1, 4)
```

cus_*curve.get*_range

Prototype: cus_*curve.get*_range(part, axis)

Function: Get the range of the specified axis

- part(string): object name
- axis(string): specify the axis, x or y

Return:

- Success: minimum (number), maximum (number)
- Failure: nil, error message

Example:

```
local xmin, xmax = cus_curve.get_range("0_CST_0", "x")
print(xmin, xmax)
```

cus_*curve.set*_scale

Prototype: cus_*curve.set*_scale(part, axis, style)

Function: Set the major scale style of the specified axis

- part(string): object name
- axis(string): specify the axis, x or y
- style(table): style sheet, format

Return:

- Success: true
- Failure: nil, error message

Example:

```
local style = {
  color = 0x000000,
  linetype = 0,
  width = 1,
  visible = 1,
}
cus_curve.set_scale("0_CST_0", "x", style)
cus_curve.set_scale("0_CST_0", "y", style)
```

cus_*curve.get*_scale

Prototype: cus_*curve.get*_scale(part, axis)

Function: Get the major scale style of the specified axis

- part(string): object name
- axis(string): specify the axis, x or y

Return:

- Success: style sheet (table)
- Failure: nil, error message

Example:

```
local x_scale_style = cus_curve.get_scale("0_CST_0", "x")
```

cus_*curve.set*_grid

Prototype: cus_*curve.set*_grid(part, axis, style)

Function: Set the main grid style of the specified axis

- part(string): object name
- axis(string): specify the axis, x or y
- style(table): style sheet, format

Return:

- Success: true
- Failure: nil, error message

Example:

```
local gridStyle = {
  color = 0xcccccc,
  linetype = 2,
  width = 1,
  visible = 1,
}
cus_curve.set_grid("0_CST_0", "x", gridStyle)
cus_curve.set_grid("0_CST_0", "y", gridStyle)
```

cus_*curve.get*_grid

Prototype: cus_*curve.get*_grid(part, axis)

Function: Get the main grid style of the specified axis

- part(string): object name
- axis(string): specify the axis, x or y

Return:

- Success: style sheet (table)
- Failure: nil, error message

Example:

```
local x_grid_style = cus_curve.get_grid("0_CST_0", "x")
```

cus_*curve.set*_line

Prototype: cus_*curve.set*_line(part, id, num, style)

Function: Set the number of points and style of the curve

- part(string): object name
- id(number): curve ID
- num(number): The number of points that can be [0, 1000] for each curve
- style(table): style sheet, see the beginning of this section for the format

Return:

- Success: true
- Failure: nil, error message

Example:

```
local line1Style = {
```

```
    color = 0x00868b,
    linetype = 0,
    width = 1,
    visible = 1,
}
-- Set the properties of line 1
cus_curve.set_line("0_CST_0", 1, 4, line1Style)
```

cus_*curve.get*_line

Prototype: cus_*curve.get*_line(part, id)

Function: Get the number of points and style of the curve

- part(string): object name
- id(number): curve ID

Return:

- Success: points (number), style sheet (table)
- Failure: nil, error message

Example:

```
local num, style = cus_curve.get_line("0_CST_0", 1)
```

cus_*curve.set*_visible

Prototype: cus_*curve.set*_visible(part, id, visible)

Function: Set the visibility of the curve

- part(string): object name
- id(number): curve ID
- visible(number):visibility

Return:

- Success: true
- Failure: nil, error message

Example:

```
-- Hide curve 1
cus_curve.set_visible("0_CST_0", 1, 0)
```

cus_*curve.get*_visible

Prototype: cus_*curve.get*_visible(part, id)

Function: Get the visibility of the curve

- part(string): object name
- id(number): curve ID

Return:

- Success: visibility (number)
- Failure: nil, error message

Example:

```
local visable = cus_curve.get_visible("0_CST_0", 1)
```

cus_*curve.set*_refer

Prototype: cus_*curve.set*_refer(part, axis, id, num, style)

Function: Set a reference line

- part(string): object name
- axis(string): specify the axis, x or y
- id(number): reference line ID
- num(number): the value of the reference line
- style(table): style sheet

Return:

- Success: true
- Failure: nil, error message

Example:

```
local referStyle = {
  color = 0x00468b,
  linetype = 0,
  width = 1,
  visible = 1,
}
-- Set a reference line perpendicular to the x axis and x=1.5
cus_curve.set_refer("0_CST_0", "x", 1, 1.5, referStyle)
```

cus_*curve.set*_cursor

Prototype: cus_*curve.set*_cursor(part, mode, style)

Function: Set the display of the click event cursor

- part(string): object name
- mode(number): display mode, 0 integer value, 1 floating value
- style(table): style sheet, see the beginning of this section for the format

Return:

- Success: true
- Failure: nil, error message

Example:

```
local cursorStyle = {
  color = 0xff0000, -- color
  linetype = 2, -- line type
  width = 1, -- line width
  visible = 1, -- visible or not
}
cus_curve.set_cursor("0_CST_0", 0, cursorStyle)
```

cus_*curve.draw*_line

Prototype: cus_*curve.draw*_line(part, id, data)

Function: Draw a curve and connect all the coordinates into a line in the increasing order of the array

- part(string): object name
- id(number): curve ID
- data(table): a one-- dimensional data table, odd-- numbered index represents x, even-- numbered index represents y, such as:
  (data[1], data[2]) represents the first coordinate (x1, y1),
  (data[3], data[4]) represents the second coordinate (x2, y2),

Return:

- Success: true
- Failure: nil, error message

Example:

```
local xy = {1, 3, 2, 1, 3, 4, 4, 1}
cus_curve.draw_line("0_CST_0", 1, xy)
```

cus_curve.refresh

Prototype: cus_curve.refresh(part)

Function: Refresh the object, call this interface after setting the scale, grid, label and line drawing to refresh the curve object and display it

- part(string): object name

Return:

- Success: true
- Failure: nil, error message

Example:

```
cus_curve.refresh("0_CST_0")
```

cus_curve.reset

Prototype: cus_curve.reset(part)

Function: reset object, clear all settings

- part(string): object name

Return:

- Success: true
- Failure: nil, error message

Example:

```
cus_curve.reset("0_CST_0")
```

Example

Use Lua object, add curve_test() in the falling script, compile and run to see the effect

```
function curve_test()
  local style = {
      color= 0x000000,
      linetype= 0,
      width= 1,
      visible= 1,
   }
  local gridStyle = {
      color= 0xcccccc,
      linetype= 2,
      width= 1,
      visible= 1,
   }
  local line1Style = {
      color= 0x00868b,
```

```
        linetype= 0,
        width= 1,
        visible= 1,
    }
    local referStyle = {
        color= 0x00468b,
        linetype= 0,
        width= 1,
        visible= 1,
    }
    local cursorStyle = {
        color= 0xff0000,
        linetype= 2,
        width= 1,
        visible= 1,
    }
    local xy = {1, 3, 2, 1, 3, 4, 4, 1}
    local part = "0_CST_0"
    cus_curve.init(part, 3, 3)
    cus_curve.set_range(part, "x", 1, 4)
    cus_curve.set_range(part, "y", 1, 4)
    cus_curve.set_scale(part, "x", style)
    cus_curve.set_scale(part, "y", style)
    cus_curve.set_grid(part, "x", gridStyle)
    cus_curve.set_grid(part, "y", gridStyle)
    cus_curve.set_line(part, 1, 4, line1Style)
    -- cus_curve.set_visible(part, 1, 0)
    cus_curve.set_refer(part, "x", 1, 1.5, referStyle)
    cus_curve.set_cursor(part, 0, cursorStyle)
    cus_curve.draw_line(part, 1, xy)
    cus_curve.refresh(part)

    print(cus_curve.get_range(part, "x"))
    print(cus_curve.get_scale(part, "x"))
    print(cus_curve.get_grid(part, "x"))
    print(cus_curve.get_label(part, "x"))
    print(cus_curve.get_line(part, 1))
    print(cus_curve.get_visible(part, 1))
end
```

MQTT

Description

The library provides the MQTT client function, which can let HMI connect to the MQTT server

The name of the library is: **mqtt**

Safety Authentication

Only supports the method of username/password authentication

#Note:

This library cannot be run on the simulator, please debug and run on HMI

List

Function	Description
mqtt.setup_cfg	Get the configuration information of mqtt from the HMI

mqtt.create	Create object of mqtt client
mqtt.close	Close object of mqtt client
mqtt:connect	Establish connection with the server
mqtt:disconnect	Disconnect from server
mqtt:isconnected	Whether connect to the server
mqtt:close	Close the mqtt client object
mqtt:subscribe	Subscribe topic
mqtt:unsubscribe	Unsubscribe topic
mqtt:publish	Publish message
mqtt:on	Register message callback function
Example 1	Standard usage example of mqtt library
Example 2	How to easily get configuration parameters

mqtt.setup_cfg

Prototype: mqtt.setup_cfg()

Function: Get the user MQTT parameters configured in the “Cloud" in the record settings

Parameters: None

Return:

- Success: 4 results
  Server address (string)
  Client ID (string)
  Connection (table)
  Will message (table)
- Failure: nil

Example: Example 2

```
local serverurl, clientid, conn, lwt = mqtt.setup_cfg()
print(serverurl, clientid, conn, lwt)
```

mqtt.create

Prototype: mqtt.create(serverurl, clientid)

Function: Create mqtt client object

Parameter:

- serverurl(string): server address, format: "protocol://host:port"
  protocol: tcp
  host: Hostname/IP address
  port: such as 1883
- clientid(string): client ID

Return:

- Success: mqtt object (mqtt)
- Failure: nil, err

Example: Example 1

```
-- 1. Use Get method
local serverurl, clientid, conn, lwt = mqtt.setup_cfg()
local obj, err = mqtt.create(serverurl, clientid)
if obj then
```

```
  print("mqtt create ok")
else
  print("mqtt create:", err)
end
-- 2. Use the designated IP method, if 192.168.1.123 is an mqtt server then
local obj2 = mqtt.create("tcp://192.168.1.123:1883", "demo123")
```

mqtt.close

Prototype: mqtt.close(obj)

Function: close the mqtt client object

Parameter:

- obj(mqtt): obj is the object created and returned by mqtt.create

Return:

- Success: true
- Failure: nil

Example:

```
local obj2 = mqtt.create("tcp://192.168.1.123:1883", "demo123")
if obj2 then
  print(mqtt.close(obj2))
end
```

mqtt:connect

Prototype: mqtt:connect(conn[, lwt])

Function: establish connection with the server

Parameter:

- conn(table): table, require the following fields
  conn.username(string): username
  conn.password(string): password
  conn.keepalive(number): Keep-Alive Interval, unit: second, default 60 seconds
  conn.cleansession(number): Clean Session, default 1
  This function is used to control the behavior when connecting & disconnecting. Both client and server retain session information. This information is used to ensure arrived "at least once" and "accurately once" , and includes the topic subscribed by the client. You can choose to reserve or ignore the session message. The settings are as follows:
  1 (Clean): If the session exists previously and is 1, then all messages from session on the client and server are cleaned.
  0 (Reserved): Conversely, both the client and the server use the previous session. If the previous session does not exist, start a new session.
- lwt(table): Will message, also called Last Will and Testament, require the following fields
  lwt.topic(string): Topic
  lwt.message(string): Message body
  lwt.qos(number): QoS value, default 0
  lwt.retain(number): Reserved flag bit, default 0

Return:

- Success: true
- Failure: nil, err

Example: Example 1

mqtt:disconnect

Prototype: mqtt:disconnect([timeout])

Function: Disconnect from the server

Parameter:

-timeout(number): Connection Timeout, unit: milliseconds, the default is 10000 milliseconds

Return:

- Success: true
- Failure: nil

Example: Example 1

mqtt:isconnected

Prototype: mqtt:isconnected()

Function: Whether connect to the server

Parameters: None

Return:

- Success: true Connected
- Failure: false Not connected, other unknown situations also return false

Example: Example 1

mqtt:close

Prototype: mqtt:close()

Function: Close the object of mqtt client (If it is connected to the server and not manually disconnected, it will be automatically disconnected)

Parameters: None

Return:

- Success: true
- Failure: nil, err

Example: Example 1

mqtt:subscribe

Prototype: mqtt:subscribe(topic, qos)

Function: Subscribe topic

Parameter:

- topic(string): Topic name
- qos(number): QoS value

Return:

- Success: true
- Failure: nil, err

Example: Example 1

mqtt:unsubscribe

Prototype: mqtt:unsubscribe(topic)

Function: Unsubscribed topic

Parameter:

- topic(string): Topic name

Return:

- Success: true
- Failure: nil, err

Example: Example 1

mqtt:publish

Prototype: mqtt:publish(topic, message, qos, retain[, timeout])

Function: Publish message

Parameter:

- topic(string): Topic name
- message(string): Message body
- qos(number): QoS value
- retain(number): Reserved flag
- timeout(number): Waiting Timeout of publish response, unit: milliseconds (only works when QoS > 0)

Return:

- Success: true
- Failure: nil, err

Example: Example 1

mqtt:on

Prototype: mqtt:on(method, callback)

Function: Register message callback function

Parameter:

- method(string): Totally 3 types of events (Message/Arrived/Offline)

1.message: This function will be called when a message is received

- Function prototype: function(string topic, string message)
- Parameters:
  - topic Topic name
    message Message content

2.arrived: This function will be called after the publication is arrived

- Function prototype: function()
- Parameters: None

3.offline: This function will be called after the connection is lost

- Function prototype: function(string cause)
- Function:
  - cause Reason of loss connection
- callback(function): callback function, call this function once after the event is triggered

Return:

- Success: true
- Failure: nil, err

Example: Example 1

Example 1

Effect: Demonstrate that the mqtt client sends and receives print data spontaneously

The operation steps are as follows:

1. Create a new file named mqdemo.lua, and copy the following code to the file
2. Modify the configuration parameters in the function mqtt.create and the variable local config in the script
3. Place a Lua object on screen No.0, and fill in mqtt_test() in the falling type script
4. Compile and click object, let the object execute the function mqtt_test(), observe the debug console

```
mqdemo_limits = 11
local obj
local config = {
    username = "admin",-- account
    password = "password",-- password
    keepalive = 100, -- set the Keepalive Interval to 100 seconds
    cleansession = 0, -- keep the session
}
-- mqtt execution
function mqtt_loop()
  if not obj then
      local err
      -- Create an object, please replace it with the IP address of the server you want to connect to
       obj, err = mqtt.create("tcp://192.168.40.226:1883", "clientid")
      if obj then
          -- Register callback for receiving message
           obj:on("message",
            function(topic, msg)
               print("mqtt msg:", string.format("%s:%s", topic, msg))
            end
           )
          -- Register callback for lost connection
           obj:on("offline", function(cause) print("mqtt offline:", cause) end)
          -- Register callback for sending message
           obj:on("arrived", function() print("msg arrived") end)
      else
          print("mqtt create err:", err)
      end
  else
      if obj:isconnected() then
           obj:publish("tomyself", "send message to my self", 0, 0)
      else
          -- Connect
          local stat, err = obj:connect(config)
          if stat == nil then
              -- Connection failed
              print("mqtt connect err:", err)
          else
              -- Subscribe topic after the connection is successful
               obj:subscribe("tomyself", 0)
          end
      end
      -- The following comment lines are only used as interface demonstration usage, know how to call
```

```
    -- obj:unsubscribe("tohmi") - Unsubscribe
    -- obj:disconnect() - disconnect
    -- obj:close() - close
  end
end
function mqtt_test()
  -- Create a timer on screen No.0
  timer.create(0, 0, 1000, mqtt_loop)
end
```

Example 2

Goal: Learn to use the interface function to get configuration parameters

Effect: Demonstrate that mqtt client sends and receives print data spontaneously, same as example 1

The operation steps are as follows:

1. Confirm that the User MQTT parameters are configured in the “Cloud"
2. Create a new file named mqdemo.lua, copy the following code to the file
3. Modify the configuration parameters in the function mqtt.create and the variable local config in the script
4. Place a Lua object on screen No.0, and fill in mqtt_test() in the falling type script
5. Compile and click object, let it execute the function mqtt_test(), observe the debug output

```
mqdemo_limits = 11
local obj
-- mqtt execution
function mqtt_loop()
  if not obj then
    local err
    -- Create an object, the parameters are taken from global variables
     obj, err= mqtt.create(URL, CLIENDID)
    if obj then
      -- Register callback for receiving message
       obj:on("message",
        function(topic, msg)
          print("mqtt msg:")
          print(string.format("topic: %s\\n msg: %s\\n", topic, msg))
        end
       )
      -- Register callback for lost connection
       obj:on("offline", function(cause) print("mqtt offline:", cause) end)
      -- Register callback for sending message
       obj:on("arrived", function() print("msg arrived") end)
    else
      print("mqtt create err:", err)
    end
  else
    if obj:isconnected() then
        obj:publish("tomyself", "send message to my self", 0, 0)
    else
      -- Connection, parameters are taken from global variables
      local stat, err = obj:connect(CONN, LWT)
      if stat == nil then
        -- Connection failed
        print("mqtt connect err:", err)
      else
        -- Subscribe topic after the connection is successful
         obj:subscribe("tomyself", 0)
      end
    end
```

```
    -- The following comment lines are only used as interface demonstration usage, know how to call
    -- obj:unsubscribe("tohmi") - Unsubscribe
    -- obj:disconnect() - disconnect
    -- obj:close() - close
  end
end
function mqtt_test()
  -- Get parameters from the configuration and set them as global variables
  URL, CLIENDID, CONN, LWT = mqtt.setup_cfg()
  -- Create a timer on screen No.0
  timer.create(0, 0, 1000, mqtt_loop)
end
```

Cloud Interface

Description

The library provides the function of getting data collected by "Cloud".

In the configuration interface of the "Cloud", there is a checkbox for the tags for upload selection. If you make the following selections:

1. Check upload to Cloud: the real-time data, Data records, and Alarm record of the tags will be pushed to the Cloud.
2. Check upload to User MQTT: the real-time data, Data records, and Alarm record of the tags will all be transmitted through the lua callback function, and the user can upload these data to the third-party server.

That is, the choice of the upload selection of the data, can only choose one from two.

For example, if you choose upload to Cloud, all interfaces of this library can be considered invalid and cannot be used.

The name of the library is: **cloud**, some of which are callback functions

#Note: This library cannot be run on the simulator, please debug and run on HMI.

Data Type Table

Value	Description
100	16-bit binary
101	16-bit octal
102	16-bit hexadecimal
103	16-bit BCD
104	16-bit signed decimal
105	16-bit unsigned decimal
200	32-bit binary
201	32-bit octal
202	32-bit hexadecimal
203	32-bit BCD
204	32-bit signed decimal
205	32-bit unsigned decimal
206	32-bit (single precision) floating
400	64-bit binary
401	64-bit octal
402	64-bit hexadecimal
403	64-bit BCD

404	64-bit signed decimal
405	64-bit unsigned decimal
406	64-bit (double precision) floating
1000	String format

Function Table

Function	Description
cloud.get_*history*_def	Get the configuration of Data record from HMI
cloud_*history*_data	Data recording callback function
cloud.get_*alarm*_def	Get the configuration of the alarm record on the configuration software
cloud_*alarm*_data	Callback function of alarm record
cloud.get_groupdesc	Get the configurations of all groups (Only valid in "cloud business mode")
cloud.get_datadesc	Get the configurations of all collection points (Only valid in "cloud business mode")
cloud.get_groupdata	Get the collection point information of one group (Only valid in "cloud business mode")
cloud.set_data	Set a collection point (Only valid in "cloud business mode")
cloud.get_data	Get the data of a collection point(Only valid in "cloud business mode")
cloud.get_alldata	Get all the collection points that meet the conditions (Only valid in "cloud business mode")
cloud.set_onecache	Save a piece of data to the cache (Only valid in "cloud business mode")
cloud.get_allcache	Get data from the cache (Only valid in "cloud business mode")

cloud.get_*history*_def

Prototype: cloud.get_*history*_def()

Function: Get the configuration of Data record from HMI, only get channels with the Cloud checked

Parameters: None

Return:

- Success: three-dimensional array (table)
- Failure: nil

Example:

```
function get_history_config()
  local data = cloud.get_history_def()
  if data == nil then return end
  for i,m in ipairs(data) do
    print(i, m[1], m[2], m[3], m[4], m[5], m[6], m[7])
    -- [1] The group ID of the tag
    -- [2] The group name of the tag
    -- [3] Collection ID of the tag
    -- [4] The channel name of the tag
    -- [5] The address of the tag
    -- [6] The data format of the tag, see the beginning of this section
    -- [7] Data length of the tag
  end
end
```

cloud_*history*_data

Prototype: cloud_*history*_data(gid, ts, data)

Function: Data Record collection callback function, when the collection conditions are met, the collection is triggered and this interface function is called

Parameter:

- gid(number): group ID
- ts(string): time format string, for example: "2021-03-01 14:35:38"
- data(table): two-dimensional array, collected data
  The first layer is the number of data
  The second layer is the specific information of the data

Return: None

Example:

```
function cloud_history_data(gid, t, tb)
  if tb == nil then return end
  print("group id=", gid)
  print("collect time=", t)
  for i,v in pairs(tb) do
    -- i represents the i-th tag, v is the second layer table, which contains the specific information of the data
    print(i, v[1], v[2], v[3])
    -- [1] Tag ID
    -- [2] Current value
    -- [3] Tag status, 1 success, 0 failure
  end
end
```

cloud.get_*alarm*_def

Prototype: cloud.get_*alarm*_def()

Function: Get the configuration of the Alarm record from HMI, only get channels with the Cloud checked

Parameters: None

Return:

- Success: two-dimensional array (table)
- Failure: nil

Example:

```
function get_alarm_config()
  local data = cloud.get_alarm_def()
  if data == nil then return end
  for i,m in ipairs(data) do
    print(i, m[1], m[2], m[3], m[4], m[5])
    -- [1] Tag ID
    -- [2] Alarm type, 0 bit, 1 word
    -- [3] Alarm address, for example: "HDX100.0" or "HDW19"
    -- [4] Alarm condition, for example: "On" or "100≤X≤300"
    -- [5] Alarm text
  end
end
```

cloud_*alarm*_data

Prototype: cloud_*alarm*_data(data)

Function: Alarm data callback function, call this interface function when an alarm occurs

Parameter:

- data(table): two-dimensional array, Alarm data
  The first level is the number of Alarm
  The second layer is the specific information of the Alarm

Return: None

Example:

```
function cloud_alarm_data(tb)
  if tb == nil then return end
  for i,v in pairs(tb) do
    -- i means the i-th Alarm, v is the second layer table, which contains the specific information of the alarm
    print(i, v[1], v[2], v[3], v[4])
    -- [1] Tag ID
    -- [2] Current value
    -- [3] Tag status, 1 success, 0 failure
    -- [4] String time, for example: "2021-03-01 14:35:38"
  end
end
```

cloud.get_groupdesc

Prototype: cloud.*get_groupdesc()*

Function: Get the configurations of all groups.

Parameter: None

Return:

- Success: table two-dimensional array

  {

  [1] = {[1] = Trigger type, [2] = Group name, [3] = Cycle},

  ...

  [n] = {[1] = xx, [2] = xx, [3] = xx},

  }

  Trigger type:

  ? 0: Change triggered

  ? 1: Word triggered

  ? 2: No trigger (sampling by time)

  ? 3: Bit triggers record (sampling by time)

  ? 4: Bit triggers a record and resets

  ? 5:Bit triggers a record and does not reset

  Cycle: Some tags type have no cycle (Unit: `ms), then` the cycle is -1.
- Failure: table empty

Example:

```
local tableGroupDef = cloud.get_groupdesc()
local sSenddData = ""
for i, j in pairs(tableGroupDef) do
```

```
  local sData = ""
  for k, v in pairs(j) do
      sData= sData .. v
      sData= sData .. " "
  end
   sSenddData = sSenddData .. sData .. "\n"
end
print(sSenddData)
```

cloud.get_datadesc

Prototype: cloud.*get_datadesc()*

Function: Get the configurations of all collection points

Parameter: None

Return:

- Success: table three-dimensional array

  {

  [1]={------First group

  ? [1]={------The tag array of first group

  ? [1] = {[1] = Collection pointID, [2] = Collection point name, [3] = Read-write property, [4] = Type},

  ? ...

  ? [n] = {[1] = xx, [2] =xx [3] = xx, [4] = xx}

  ? },

  ? [2] = Group ID,

  ? [3] = Group name

  ? },

  .....n groups and so on

  }

  Read-write property: 0: read-only, 1: write-only, 2: read and write

  Type: 1: switch, 2: numerical, 3: strings
- Failure: table empty

Example:

```
local tablePointDef = cloud.get_datadesc()
local sSendData = ""
for i, j in pairs(tablePointDef) do
  local sData = ""
  for x, y in pairs(j) do
    if type(y) == "table" then
      for k, v in pairs(y) do
        for a, b in pairs(v) do
            sData= sData .. b
            sData= sData .. ","
        end
          sData= sData .. " "
      end
    else
```

```
            sData= sData .. y
            sData= sData .. " "
        end
    end
    sSendData = sSendData .. sData .. "\n"
end
print(sSendData)
```

cloud.get_groupdata

Prototype: cloud.*get_groupdata(name)*

Function: Get the collection point information of one group

Parameter:

- name: group name

Return:

- Success: table two-dimensional array

  {

  [1]= {[1]=Collection point ID, [2]=status, [3]=Collect pointion name, [4]=Numeric value, [5]={Cunstom content}},

  ...

  [n] = {[1] = xx, [2] = xx, [3] = xx, [4] = xx, [5] = {xx}}

  }

  Status: 0: Online, -1: Write-only, -2: Offline,
- Failure: table empty

Example:

```
local sGroup = cloud.get_groupdata("Group 3")
local sSendData = ""
for i, j in pairs(sGroup) do
    local sData = ""
    for k, v in pairs(j) do
        if type(v) == "table" then
            sData= sData .. "Custom:{"
            for x, y in pairs(v) do
                sData= sData .. x
                sData= sData .. ":"
                sData= sData .. y
                sData= sData .. ","
            end
            sData= sData .. "}"
            sData= sData .. " "
        else
            sData= sData .. v
            sData= sData .. " "
        end
    end
    sSendData = sSendData .. sData .. "\n"
end
print(sSendData)
```

cloud.set_data

Prototype: cloud.s*et_data(name, data)*

Function: Set a collection point

Parameter:

- name: collection point name
- data: value (only be `string`)

Return:

- Success: The set value
- Failure: nil

Example:

```
local sValue = cloud.set_data("Channel 1", "100")
local sSendData = ""
sSendData = sSendData .. "Set channel 1 Value : " .. sValue
print(sSendData)
```

cloud.get_data

Prototype: cloud.g*et_data(name)*

Function: Get the configurations of a collection point

Parameter:

- name: collection point name

Return:

- Success: The value of collection point, the custom content of collection point
- Failure: nil

Example:

```
local sValue, tableCustome = cloud.get_data("Channel 1")
local sSendData = ""
sSendData = sSendData .. sValue .. "\n"
for i, j in pairs(tableCustome) do
   sSendData = sSendData .. i .. j .. "\n"
end
print(sSendData)
```

cloud.get_alldata

Prototype: cloud.*get_alldata()*

Function: Get all the collection points that meet the conditions

Parameter: None

Return:

- Success: table two-dimensional array

  {

  [1]= {[1]=Collection point ID, [2]=status, [3]=Collect pointion name, [4]=Numeric value, [5]={Cunstom content}},

  ...

  [n] = {[1] = xx, [2] = xx, [3] = xx, [4] = xx, [5] = {xx}}

  }

  Status: 0: Online, -1: Write-only, -2: Offline,
- Failure: table empty

Example:

```
local tablePointsData = cloud.get_alldata()
local sSendData = ""
for i, j in pairs(tablePointsData) do
  local sData = ""
  for k, v in pairs(j) do
    if type(v) == "table" then
        sData= sData .. "Custom:{"
      for x, y in pairs(v) do
          sData= sData .. x
          sData= sData .. ":"
          sData= sData .. y
          sData= sData .. ","
      end
        sData= sData .. "}"
        sData= sData .. " "
    else
        sData= sData .. v
        sData= sData .. " "
    end
  end
   sSendData = sSendData .. sData .. "\n"
end
print(sSendData)
```

cloud.set_onecache

Prototype: cloud.set_*onecache(data)*

Function: Save a piece of data to the cache (30 seconds to write cached data to flash)

Parameter:

- data: the data to the cache

Return:

- Success: TRUE
- Failure: nil

Example:

```
local sSaveData = we_bas_getstring("@W_HDW5000", 10)
local sSendData = ""
cloud.set_onecache(sSaveData)
sSendData = sSendData .. "Save Data : " .. sSaveData
print(sSendData)
```

cloud.get_allcache

Prototype: cloud.*get_allcache()*

Function: Getdata from the cache (The cache data and flash data will be cleared after fetching))

Parameter: None

Return:

- Success: table two-dimensional array

  {

  [1]="This is the oldest message", ------ The first one is the oldest message

  .....

  [n]="This is the latest message", ------ The last one is the latest message

  }
- Failure: nil

Example:

```
local tableCacheDatas = cloud.get_allcache()
local sSendData = ""
if nil == tableCacheDatas then
  return
end
for k, v in pairs(tableCacheDatas) do
   sSendData = sSendData .. v .. "\n"
end
print(sSendData)
```

json

Description

The library provides the conversion function from json string data to lua table data

The name of the library is: **json**

List

Function	Description
json.encode	Lua data type to json string
json.decode	json string to lua data type
json.null	null
Example	

json.encode

Prototype: json.encode(object)

Function: convert lua data type to json string

Parameter:

- object(boolean, number, string, table): lua data

Return:

- Success: json format string (string)
- Failure: nil

json.decode

Prototype: json.decode(json_string)

Function: json string to lua data type

Parameter:

- json_string(string): string on json data structure

Return:

- Success: lua data type
- Failure: nil

json.null

Prototype: json.null()

Function: This function is used when encapsulating json data, which is equivalent to null in json. If you use json.null() directly, it will directly return the address of the function, please use it with encode function together

Parameters: None

Return: None

Example

```
local json = require("json")
local js_string = json.encode({1, 2,'fred', {username ='foo', password=json.null}})
print("json encode:", js_string) -- convert to json data
local data = json.decode(js_string) -- converted to script data type
if type(data) =='table' then
  print("json decode:")
  for k,v in pairs(data) do
    print(k,v)
  end
end
```

Output:

```
json encode: [1,2,"fred",{"username":"foo","password":null}]
json decode:
1 1
twenty two
3 fred
4 table: 05B30068
```

WLAN

The library provides WLAN configuration function.

The name of the library is: **hmiwifi**

List

Function	Description
hmiwifi.open	Enable
hmiwifi.close	Disable

hmiwifi.config	Configure

hmiwifi.open

Prototype: hmiwif.open()

Function: Enable wifi function

Parameters: None

Return:

- Success: ture
- Failure: nil, err

hmiwifi.close

Prototype: hmiwif.close()

Function: Disnable wifi function

Parameters: None

Return:

- Success: ture
- Failure: nil, err

hmiwifi.config

Prototype: hmiwif.close(ssid, passwd)

Function: Configure wifi

Parameters:

- ssid(string) hotspot name format: length 1~32 bytes
- Passwd(streing) hotspot password format: length is 0 or greater 8

Return:

- Success: ture
- Failure: nil, err

Example

```
local ret,err = hmiwifi.open()
if ret then
print("Wlan open ok")
else
print("Wlan open :", err)
end
local ret,err = hmiwifi.config("aaa","12345678")
if ret then
print("Wlan config ok")
else
print("Wlan config:", err)
End
//open hotspot
local ret,err = hmiwifi.config("aaa","")
if ret then
print("Wlan config ok")
else
```

```
print("Wlan config:", err)
end
local ret,err = hmiwifi.close()
if ret then
print("Wlan close ok")
else
print("Wlan close :", err)
end
```

Output:

```
Wlan open ok
Wlan config ok
Wlan config ok
Wlan close ok
```

LuaSocket

It is the network programming module of Lua, and can be used for secondary development according to requirements.

It include the following contents:

1. socket: Network basic module: basic parameter and baisic fuction
2. dns: DNS module
3. TCP: TCP protocol object
4. UDP: UDP protocol object
5. ftp: ftp protocal API: Used for file transfer. This module only supports FTP client API, and does not support server API.
6. http: http protocol API
7. https: https protocol API
8. smtp: smtp protocol API: Used for third-party email services
9. Other auxiliary modules: It is the extented function of module including ltn12 auxiliary module and mime auxiliary module.
10. Appendix: ftp parameter list

#Note:

```
local socket = require("socket") -- Load the socket module and everything it needs
local http = require("socket.http") -- Load the http module and everything it needs
local smtp = require("socket.smtp")    -- Load the smtp module and everything it needs
local ftp = require("socket.ftp") -- Load the ftp module and everything it needs
```

Modules

Socket module parameter

Parameter	Description	Value	Type
_DATAGRAMSIZE	Call the default datagram size used to receive and receive usage	8192	number
_SETSIZE	The maximum number of sockets that the select function can process	64	number
_SOCKETINVALID	invalid socket system operation value	4294967295	number
_VERSION	luasocket version	“LuaSocket 3.0-rc1”	string
BLOCKSIZE	socket receiving block size	2048	number

Socket basic function

Function	Features
socket.bind()	Bind IP address and port, and return a TCP server object to the server
socket.connect4()	Connect the object to the IP address and port and create a TCP object. (IPV4)
socket.connect6()	Connect the object to the IP address and port and create a TCP object. (IPV6)
socket.gettime()	Return the system time. Unit: scond.
socket.newtry()	Create and return a function that throws an exception
socket.protect()	Convert the throw exception function to a safety function
socket.select()	Wait for multiple sockets to change state
socket.sink()	Create an LTN12 sink from a stream socket object.
socket.sleep()	Dormancy
socket.source()	Create an LTN12 source from a stream socket object
socket.tcp()	Create and return a Tcp object
socket.udp()	Creates an empty (disconnected) UDP object

dns module

#Note:

- DNS resolution in HMI is not supported due to internal changes in HMI. Therefore, only offline simulation is supported.
- dns is a part of data socket basic module. To use the contents of this module, you need to quote the followings.

```
local socket = require("socket") -- References the socket basic module
local dns = socket.dns   -- Get DNS module content
```

Function	Features
dns.toip()	Transfer from host name to IPv4 address
dns.getaddrinfo()	Transfer from host name to address
dns.tohostname()	Transfer from IPv4 address to host name
dns.getnameinfo()	Given an address (hostname or IP address) and port, and returns all the information provided by the DNS resolver
dns.gethostname()	Returns the standard hostname of the machine as a string

TCP module

This module is a part of socket. It is the object created by socket.tcp() 、socket.tcp4() 、socket.tcp6().

This module divides Tcp objects into three types of objects: masterTcp、clientTcp、serverTcp.

Different Tcp objects have corresponding methods.

- masterTcp: close
- clientTcp: send, receive, getsockname, getpeername, settimeout and close
- serverTcp: accept, getsockname, setoption, settimeout and close

Function	Features
tcp:bind()	Bind the object to the address and port of the localhost
tcp:connect()	Attempt to connect the primary object to the remote host
tcp:listen()	Socket lisntening
tcp:accept()	Wait for a remote connection and return the server object representing the link

tcp:send()	Send data
tcp:receive()	Reads data from client object according to the specified read mode.
tcp:close()	Close a TCP object
tcp:getoption()	Gets the options for the TCP object
tcp:getpeername()	Return information about the remote end of the connected client object
tcp:getsockname	Return information about the local address associated with the object
tcp:getstats()	Return accounting information about a socket that can be used to limit bandwidth
tcp:gettimeout()	Get timeout
tcp:setoption()	Set the options for the TCP object
tcp:setstats()	Reset accounting information on sockets (useful for bandwidth limit)
tcp:settimeout()	Change the timeout value of tcp object
tcp:shutdown()	Disable some full-duplex connections.
tcp:getfamily()	Get the TCP protocol family
tcp:setpeername()	Change the main object to connect to remote host
tcp:setsockname()	Change the address and port number of the object binding

UDP module

This module is a part of socket. It is the object created by socket.udp(), socket.udp4(), socket.udp6().

This module divides UDP objects into twotypes: connect and connected. The difference is whether remote ports and addresses are bound to the generated object. Please refer to the following table.

Function	Features
udp:send()	Sends datagrams to connected UDP objects
udp:sendto()	Sends a datagram to the specified IP address and port number.
udp:receive()	Receives datagrams from UDP objects
udp:receivefrom()	Receives datagrams and addresses and ports from UDP objects
udp:close()	Close udp object
udp:gettimeout()	Get timeout value
udp:settimeout()	Change timeout value of udp object
udp:getoption()	Gets the option value from the UDP object
udp:getpeername()	Search information about the peer associated with the connected UDP object
udp:getsockname()	Return information about the local address associated with the object
udp:setoption()	Set the options for the UDP object
udp:setpeername()	Change the remote information of UDP object
udp:setsockname()	Bind UDP object to local address
udp:getfamily()	Return a tcp protocol family

FTP module

- This is submodule of socket. To use the contents of this module, you need to quote the followings.

```
local ftp = require("socket.ftp")
```

For details about FTP commands, see "appendix".

Parameter	Type	Value	Description
ftp.USER	string	"ftp"	Anonymous user by default
ftp.PASSWORD	string	"anonymous@anonymous.org"	This is the default anonymous password, and used when no password is provided in the URL
ftp.TIMEOUT	number	60	Set the timeout of all I/O operations. Unit: second

Function	Features
ftp.command()	Send ftp command
ftp.genericform()	Request parameter conversion, the simple URL request mode convert to a general table parameter form
ftp.put()	Upload data
ftp.get()	Get data

http module

This is submodule of socket. To use the contents of this module, you need to quote the followings.

```
local http = require("socket.http")
```

Parameter	Type	Value	Description
http.TIMEOUT	number	60	Time out unit: second

Function	Features
http.request()	HTTP request function
http genericform()	Convert a simple URL to a generic table parameter

https module

This is submodule of socket. To use the contents of this module, you need to quote the followings.

```
local https = require("socket.https")
```

Function	Features
http.request()	HTTP request function

SMTP module

This is submodule of socket. To use the contents of this module, you need to quote the followings.

```
local smtp = require("socket.smtp")
```

Parameter	Type	Value	Description
smtp.TIMEOUT	number	60	Connection timeout period
smtp.SERVER	string	“localhost”	Default server used for sending E-mail
smtp.DOMAIN	string	“localhost”	Domain used to access the server
smtp.POST	number	25	Default port for connecting
smtp.ZONE	string	"-0000"	default time zone

Function	Features
smtp.message()	Generate mail format messages
smtp.send()	Send email

Auxiliary Module

1) LTN12 module

This is auxiliary Module of socket. To use the contents of this module, you need to quote the followings.

```
local LTN12 = require("socket.ltn12")
```

- **filter**

Function	Features
LTN12.filter.chain()	Generates a filter that connects a series of filters
LTN12.filter.cycle()	Generate a high-level filter consisting of low-level filter loops

- **pump (Data pump)**

Function	Features
LTN12.pump.all()	Extract all data from one source to one sink.
LTN12.pump.step()	Extract a chunk of data from the source to the acceptor

- **sink (acceptor)**

Function	Features
LTN12.sink.chain()	Create a new sink that transmit data through a filter before sending it to the given sink.
LTN12.sink.error()	Create and return a new sink that aborts the transmission with an error message
LTN12.sink.file()	Create a sink that send data to file
LTN12.sink.null()	Returns a sink that ignores all data received
LTN12.sink.simplify()	Given a selected sink, create and return a simple sink
LTN12.sink.table()	Create a sink that stores all the blocks in a table.

- **source**

Function	Features
LTN12.source.cat()	Create a new source that generates concatenations of data generated by multiple sourc
LTN12.source.chain()	Create a new source that transmit data through a filter before returning it.
LTN12.source.empty()	Create and return a new source
LTN12.source.error()	Create and return a source that aborts the transfer and display an error message.
LTN12.source.file()	Create a source to connect file contents.
LTN12.source.simplify()	Given a source, create and return a simple source.
LTN12.source.string()	Creates and returns a source that generates string content
LTN12.source.table()	Create and return a table data source

2) MIME module

This is auxiliary Module of socket. To use the contents of this module, you need to quote the followings.

```
local mime = require("socket.mime")
```

Function	Features
mime.decode()	Return a filter used to decode data from the given transmission content encoding
mime.encode()	Returns a filter that encodes data based on the given transmission content encoding.
mime.normalize()	Return a filter that converts the most common end-of-line tags to a specific given tag
mime.stuff()	Create and return a filter that performs SMTP message filling.
mime.wrap()	Return a filter that divides data into several lines.
mime.b64()	Create a low-level filter to perform end-of-line tag translation
mime.dot()	Create a low-level filter that performs SMTP filling and enables the transmission of mess
mime.eol()	Create a low-level filter to perform end-of-line tag translation

mime.qp()	Create a low-level filter to perform Quoted-Printable decoding
mime.qpwrp()	Create a low-level filter that divides Quoted-Printable text into several rows.
mime.unb64()	Create a low-level filter to perform Base64 decoding
mime.unqp()	Create a low-level filter that delete quoted-printable transfer content encoding from data
mime.wrp()	Create a low-level filter that uses CRLF tags to split text into several lines.

Socket basic function

socket.bind()

Prototype: socket.bind(address, port [, backlog])

Function: A shortcut function that encapsulates tcp's bind, accept, and listen calling processes. Creates and returns a TCP server object bound to a local address and port.

Parameters:

- address — local address
- port — Port
- backlog — Specifies the number of client connections that can be queued for service. If the queue is full and another client tries to connect, the connection will be rejected. (This parameter is optional. The default value is 32)

Return:

- Success: return tcp server object
- Failure: return nil and error

Example

```
local socket = require("socket")

local ip = "192.168.88.246"
local port = 8899

local fd_server = socket.bind(ip,port)
print(fd_server,type(fd_server))
--Output#tcp{server}: 06F5ED60   userdata
```

socket.connect4()

Prototype: socket.connect4(address, port [, locaddr] [, locport])

Function: This function is a shortcut that creates and returns a TCP client object connected to the remote (ipv4) address of a given port.

Parameters:

- address(string) : Remote server address
- port(number) : Remote server port
- locaddr(string) : Local address
- locport(number) : Local port

Return:

- Success: return tcp client object
- Failure: nil, error

Example

```
local socket = require("socket")
```

```
local ip = "192.168.88.246"
local port = "9966"
local locIp = "192.168.88.246"
local locPort = "8888"
local fd_client = socket.connect4(ip,port,locIp,locPort)
print(fd_client,type(fd_client))
--Output: tcp{client}: 07F627D8   userdata
```

socket.connect6()

Prototype: socket.connect6(address, port [, locaddr] [, locport])

Function: This function is a shortcut that creates and returns a TCP client object connected to the remote (ipv6) address of a given port.

Parameters:

- address(string) : Remote server address
- port(number) : Remote server port
- locaddr(string) : Local address
- locport(number) : Local port

Return:

- Success: return tcp client object
- Failure: nil, error

Example

```
local socket = require("socket")
local ip = "192.168.88.246"
local port = "9966"
local locIp = "192.168.88.246"
local locPort = "8888"
local fd_client = socket.connect6(ip,port,locIp,locPort) -- This parameter is used when the server address is ipv6
print(fd_client,type(fd_client))
--Output (test results in IPv4 addresses)# nil     nil
```

socket.gettime()

Prototype: socket.gettime()

Function: Get the system time in seconds, from January 1970.

Parameters: None

Return:

- systemtime(number): system time: second

Example

```
local socket = require("socket")

local t1 = socket.gettime() -- Get the current time
socket.sleep(5) -- Sleep for 5 seconds
local t2 = socket.gettime() -- Get2 the current time 2

print("t1 =",t1)       -- Print time 1
print("t2 =",t2)       -- Print time 2
print("t2-t1=",t2-t1)   -- Print time difference
--Output#
-- t1 =    1636878024.5057
-- t2 =    1636878029.5201
```

```
-- t2-t1=  5.0144500732422
```

socket.newtry()

Prototype: socket.newtry(finalizer)

Function: Creates and returns a try function that allows the exception to be cleaned up before it is thrown.

Parameters:

- finalizer(function): Finalizer is a function that will be called before newtry throws an exception.

Return:

- Returns a newtry function that throws an exception block

Note:

- This function is usually used with the socket.protect() function; Try throws an exception and Protect catches it.

Example

```
local socket = require("socket")
foo = socket.protect(function()
print("--------------")
local c = socket.try(socket.connect("192.168.88.241", 9090))
print(c,type(c))
-- create a try function that closes 'c' on error
local try = socket.newtry(function() c:close() end)-- Call the TCP object closure function if an exception occurs
-- do everything reassured c will be closed
 try(c:send("hello there?\r\n"))    -- formal call c of try:send()
local answer = try(c:receive())    -- formal call c of try:receive()
print(answer)
 try(c:send("good bye\r\n"))
 c:close()formal call c of try
end)
foo()
-- Analysis#Functions or more important blocks of code that may have exceptions are called in the form of try, and protection functions are called before exceptions occur.
Prevent major problems caused by exceptions.
```

socket.protect()

Prototype: socket.protect(func)

Function: Convert the throw exception function to safety function. This function only catches exceptions thrown by the try and newtry functions, and would not catch normal Lua errors.

Parameters:

- func(function): Is a function that calls try (or Assert or error) to throw an exception.

Return:

- Returns a equivalent func function that no longer throws an exception when a try call fails but returns nil and an error.

Example

```
local socket = require("socket")
foo = socket.protect(function()
print("--------------")
local c = socket.try(socket.connect("192.168.88.241", 9090))
print(c,type(c))
-- create a try function that closes 'c' on error
```

```
local try = socket.newtry(function() c:close() end)-- Call the TCP object closure function if an exception occurs
-- do everything reassured c will be closed
 try(c:send("hello there?\r\n"))            -- formal call c of try:send()
local answer = try(c:receive())   -- formal call c of try:receive()
print(answer)
 try(c:send("good bye\r\n"))
 c:close()
end)
foo()
```

socket.select()

Prototype: socket.select(recvt, sendt [, timeout])

Function: Wait for multiple sockets to change status.

Parameters:

- `recvt(arrary)`: An array with sockets that tests readable characters.
- `send(arrary)`: sendt array. Monitor the sockets of sendt array to see if they can be written immediately.
- `timeout(bunber)`: Timeout `period, nil` or a negative number is an indefinite timeout.

Return:

- Returns a equivalent func function that no longer throws an exception when a try call fails but returns nil and an error.

Note:

- Select could monitor a limited number of sockets, as defined by the constant socket._setsize. By default, this number could be as high as 1024 or as low as 64, depending on the system. You could usually change it when compiling. Calling select with a large number of sockets, an error would occur.
- Call select using the server socket in the receive parameter before calling accept does not guarantee that accept will return immediately. Using the settimeout method or accept may cause block forever.
- If a closed socket is passed to select, it will be ignored.
- Use select with non-socket objects: Any objects that implement getfd and dirty could be used with select. This allows objects from other libraries to be used in the loops of socket.select-driven.

Example

```
-- Implement timeout waits
function timeoutWait()
  local tabFd = {}
  table.insert(tabFd,fdServer)  -- Add the server socket to the Select socket monitoring table
  for i,v in pairs(tabClient) do  -- Loop traversal client
     table.insert(tabFd,tabClient[i].fd)  -- Add the client socket to the Select socket monitoring table
  end
   socket.select(tabFd)  -- select monitor socket.If the socket does not receive a connection, it remains blocked
end
```

socket.sink()

Prototype: socket.sink(mode, socket)

Function: Create an LTN12 sink from a stream socket object.

Parameters:

- mode —— Receiver mode
- socket —— A stream socket object used to receive data

Return:

- Return an LTN12 receiver

Note:

LTN12 receiver mode:

- "http-chunked": After using chunked transmission encoding, send data through socket, and close socket when completed.
- "close-when-done": Send all received data through socket, and then close the socket.
- "keep-open": Send all received data through socket, Leave it open when it is done.

socket.sleep()

Prototype: sleep(s)

Function: Sleep s seconds

Parameters:

- `n(number)`: The number of seconds to sleep

Return: None

Example

```
local socket = require("socket") print("start . . .")
local time1 = socket.gettime() socket.sleep(5)
-- Print the difference between the two acquisition times after sleeping for 5 seconds
print("socket time=",socket.gettime() - time1)
--Output#
-- start . . . -- socket time= 5.0082302093506
```

socket.source()

Prototype: socket.source(mode, socket [, length])

Function: Create LTN12 source from stream socket object.

Parameters:

- `mode(string)`: The mode of source
- `socket(userdata)`: A stream socket object used to receive data.
- `ength(number)`: Length

Return:

- Return a created LTN12 source

Note:

LTN12 source mode. The following options are available.

- "http-chunked": Receives data from the socket, and remove the chunked transmission encoding before returning data.
- "by-length": Receives a fixed number of bytes from the socket. This mode requires additional parameter lengths.
- "until-closed": Receives data from the socket until the other end closes the connection.

socket.tcp()

Prototype: `socket.tcp()`、`socket.tcp4()`、`socket.tcp6()`

Function: Create and return a Tcp main object.

Parameters: None

Return:

- Success: Return a tcp object
- Failure: nil, error

Note:

- The difference between socket.tcp4() and socket.tcp6() is whether the selected IP is ipv4 or ipv6.
- Socket. TCP divides objects into three types: main object, client object and server object. Socket.tcp creates the main object. The main object could be converted to a server object through the listen method (after calling bind), or it could be converted to a clientobject through the connect method. The only method supported by the main object is the close method.
- To choose IPv4 or IPv6 during the calling of bind or connection, it isdepending on the address family obtained from the parser.
- If the internal socket object is invalid before choosing IPv4 or IPv6, setoption will fail.

Example:

```
local socket = require("socket")
local tcp = socket.tcp()
print("tcp=",tcp,type(tcp)) -- Obtain TCP objects. For details about how to use TCP objects, see TCP Module
-----------------------------------------------
--Output#
-- tcp=    tcp{master}: 0B3074F8   userdata
```

socket.udp()

Prototype: `socket.udp()、socket.udp4()、socket.udp6()`

Function: Create an empty (disconnected) udp object.

Parameters: None

Return:

- Success: Return a new disconnected udp object.
- Failure: nil, error

Note:

- The difference between socket.tcp4() and socket.tcp6() is whether the selected IP is ipv4 or ipv6.
- To choose IPv4 or IPv6 during the calling of udp:sendto, udp:setpeeername or udp:sockname, it is depending on the address family obtained from the parser.
- If the internal socket object is invalid before choosing IPv4 or IPv6, udp: setoption will fail.

Example:

```
local socket = require("socket")
local udp = socket.udp()
print("udp=",udp,type(udp)) -- Obtain udp objects. For details about how to use udp objects, see udp Module
-----------------------------------------------
--Ouptput
-- udp=    udp{unconnected}: 0B808B98      userdata
```

dns module

dns.toip()

Prototype: dns.toip(address)

Function: Convert from hostname to IPv4 address

Parameters:

- address(string): Address. It could be an IP name or a host name.

Return:

Success: .

- address(string): IP address
- info(table): address information table

Failure: return nil and error

Example:

```
local socket = require("socket")
local dns = socket.dns
local hostname = dns.gethostname()
local ip,msg = dns.toip(hostname)
if ip then
  print("ip=",ip)
else
  print(msg)
end
--Output
-- ip=     192.168.33.149
```

dns.getaddrinfo()

Prototype: dns.getaddrinfo(address)

Function: Convert from hostname to address

Parameters:

- address(string): Address. It could be an IP name or a host name.

Return:

- Success: info(table): address information table
- Failure: return nil and error

Example:

```
local socket = require("socket")
local dns = socket.dns
local host = dns.gethostname()
  local addrinfo = dns.getaddrinfo(host)
  for i,v in pairs(addrinfo) do
     print(i,v,"----------")
     for i2,v2 in pairs(v) do
        print(i2,v2)
     end
  end
--Output#
-- 1       table: 0AB71660 ----------
-- addr    fe80::9930:10db:5b7f:1249%4
-- family  inet6
-- 2       table: 0AB715C0 ----------
-- addr    192.168.33.149
-- family  inet
```

dns.tohostname()

Prototype: dns.tohostname(address)

Function: Convert from IP4v address to hostname

Parameters:

- address(string): Address. It could be an IP name or a host name.

Return:

- Success: Return a host name string and an address information table
- Failure: return nil and error

Example:

```
local host = dns.gethostname()
local ip = dns.toip(host)
local hostname,msg = dns.tohostname(ip)
if hostname then
  print("hostname=",hostname)
else
  print(msg)
end
--Output#
-- hostname=       DESKTOP-LB0DAGV
```

dns.getnameinfo()

Prototype: dns.getnameinfo(address, port)

Function: Given address (host name or IP address) and port, and return all the information provided by dns parser.

Parameters:

- address(string): Address. It could be an IP name or a host name.
- port(number): Service port number

Return:

- Success: Return an address information table and port informantion string
- Failure: return nil and error

Example:

```
local socket = require("socket")
local json = require("json")
local host = "DESKTOP-P46LTTK"
local port = 21
local hostinfo,servinfo=socket.dns.getnameinfo(host, port)
print("hostinfo =",json.encode(hostinfo))
print("portinfo =",servinfo)
--Output#
-- hostinfo = ["DESKTOP-P46LTTK","DESKTOP-P46LTTK"]
-- portinfo = ftp
```

dns.gethostname()

Prototype: dns.gethostname()

Function: Return the standard hostname of the machine as a string

Parameters: None

Return:

- Success: Return the standard hostname of the machine as a string
- Failure: return nil and error

Example:

```
local socket = require("socket")
print("host name =",socket.dns.gethostname())
--Output#
-- host name = DESKTOP-P46LTTK
```

TCP module

tcp:bind()

Prototype: master:bind(address, port)

Function: Bind the object to the address and port of local host.

Parameters:

- `address(string)`: IP address or host name
- `port(number)`: port number

Return:

- Success: Return 1.
- Failure: return nil and error.

Note:

- If the address is "*", the system binds all local interfaces with the INADDR_ANYc constant or IN6ADDR_ANY_INIT.
- If the port is 0, the system automatically selects a temporary port.
- The socket.bind function is a shortcut to create a server socket.

tcp:connect()

Prototype: master:connect(address, port)

Function: Attempt to connect the primary object to the remote host.

Parameters:

- `address(string)`: IP address or host name
- `port(number)`: port number

Return:

- Success: Return 1.
- Failure: return nil and error.

Note:

- Convert main object to client object after using the connect function.
- The socket.connect function is available and is a shortcut to create a client socket.
- Starting with LuaSocket 2.0, the settimeout method woudl affect the action of connect, causing it to return an error in the case of a timeout. If this happens, you could still use the socket in the Sendt table to call socket.select. AFfter the connection, the socket will be writable.
- Starting with LuaSocket 3.0, hostname resolution depends on whether a socket is created by socket.tcp, socket.tcp4, or socket.tcp6. Try the addresses from the appropriate series (or both) according to the order returned by the parser until the first success or the last failure. If timeout is set to zero, only the first address is tried.

tcp:listen()

Prototype: master:listen(backlog)

Function: Listening socket, convert the specified master to server object.

Parameters:

- `backlog(number)`: Specify the number of client connections that can be queued for service. If the queue is full and another client tries to connect, the connection will be rejected.

Return:

- Success: Return 1.
- Failure: return nil and error.

tcp:accept()

Prototype: server:accept( )

Function: Wait for a remote connection and return the server object representing the connection.

Parameters: None

Return:

- Success: Return a client object.
- Failure: return nil and error. (If the timeout condition is met, return nil and error string "timeout")

Note:

- Call select using the server socket in the receive parameter before calling accept does not guarantee that accept will return immediately. Using the settimeout method or accept may cause block forever until another client appears.

tcp:send()

Prototype: client:send(data [, i [, j]])

Function: Send data.

Parameters:

- data(string): The string to be sent
- `i, j(number)`: The parameter fucntion is like string.sub. You could select a substring to send.

Return:

- Success: Return the index of the last byte in the sent [i,j]. If i is 1 or does not exist, then it is the actual total
- Failure: return nil and error.

Note:

- The output is not buffered. For small strings, it is better to connect them in Lua and send the result in one call rather than calling the method multiple times.

tcp:receive()

Prototype: client:receive([pattern [, prefix]])

Function: Read data from client object according to the specified read mode. The mode follows the Lua file I/O format, and the performance differences between all modes could be ignored.

Parameters:

1. pattern(string):

- 'a': Read from socket until the connection is closed. End-of-line translation is not performed.

- *'*l': Read a line of text from socket. The line is ends with the LF character (ASCII 10), and optionally preceded by the CR character (ASCII 13). CR and LF characters are not included in the returned line. In fact, the mode ignores all CR characters. This is the default mode.
- number: Read the specified number of bytes from socket.

1. prefix(string): Prefix is an optional string, and connect to the beginning of any received data before returning.

Return:

- Success: Return the received data.
- Failure: return nil and error.

tcp:close()

Prototype:

- master:close()
- clien:close()
- server:close()

Function: Close a TCP object

Parameters: None

Return: None

Note:

- It is important to close all the sockets when not needed. Because in many systems, every socket uses a file descriptor. This is a limited system resource. However, the garbage collection object is automatically closed before being destroyed.

tcp:getoption()

Prototype:

- client:getoption(option)
- server:getoption(option)

Function: Get the options of TCP object. Option is a string with option name.

Parameters:

- option(string): Option is a string with option name. `("keepalive",` "linger", "reuseaddr", "tcp-nodelay")

Return:

- Success: Return option value.
- Failure: return nil and error.

tcp:getpeername()

Prototype: client:getpeername()

Function: Return remote information of the connected client object.

Parameters: None

Return:

- return a string, including the IP address of the peer, the port number used by the peer to connect and a string with series ("inet" or "inet6"). If an error occurs, return nil.

tcp:getsockname()

Prototype:

- master:getsockname()
- client:getsockname()
- server:getsockname()

Function: Return information about the local address associated with the object.

Parameters: None

Return:

- Success: Return a string with local IP address, a number with local port and a string with a series "inet" or "inet6".
- Failure: return nil and error.

tcp:getstats()

Prototype:

- master:getstats()
- client:getstats()
- server:getstats()

Function: Return accounting information about socket. It could be used to limit bandwidth.

Parameters: None

Return:

- Success: Return the number of the bytes received, he number of the bytes sent and the time of socket object. (Unit: second)
- Failure: return nil and error.

tcp:gettimeout()

Prototype:

- master:settimeout*value [, mode])
- client:settimeout(value [, mode])server:settimeout(value [, mode])

Function: Change the timeout value of tcp object.

Parameters:

- value(number): The amount of time to wait is specified as the value parameter in seconds. nil and negative value allow operations to block indefinitely.
- `mod(string):` timeout mode

Return: None

Note:

1. By default, all the I/O operations are blocked. That is, any call to the method send, receive and accept will indefinitely block until the operation is complete. The method settimeout define a limit on the amount of time that n I/O method can block. When timeout is set and after a specified amount of time, the affected method aborts and fails with an error code.
2. There are two timeout modes:

- 'B': block timeout. It specifies the maximum time limit that the operating system can block LuaSocket while waiting for any single I/O operation to complete. This is the default mode.
- 't' : total timeout. It specifies the maximum time limit that LuaSocket can block Lua scripts before the call returns

1. Although the timeout value in LuaSocket has millisecond precision, large blocks can cause I/O functions not to respect timeout values due to the time the library takes to transfer blocks to and from the OS and to and from the Lua interpreter. Also, function that accept host names and perform automatic name resolution might be blocked by the resolver for longer than the specified timeout value.

tcp:setoption()

Prototype:

- client:setoption(option [, value])
- server:setoption(option [, value])

Function: Set the option of TCP object. Only low-level or time-critical applications require options. If you are sure that you need it, you should only modify one option.

Parameters:

- option(string): option name
- valu(number): option value

Return:

- Success: Return 1.
- Failure: return nil and error.

Note:

- 'keepalive': Set this option to true to transmit messages regularly on the connected socket. If the connection fails to respond to these messages, then it is considered disconnected and the process using the socket is notified.
- 'linbger': Controls the action taken when unsent data is queued on the socket and the action taken when it is closed. The value is a table of numeric entries with a Boolean entry "on" and a time interval "timeout" (in seconds). If the "on" field is set to true, the system blocks the process on a shutdown attempt until it can transfer data or until the "timeout" passes. If "on" is false and a shutdown is issued, the system will process it in a way that allows the process to continue as soon as possible. It is not recommended to set it to anything value other than 0.
  Shut down. I do not recommend that you set it to anything other than zero
- 'reuseaddr': Set this option indicates that the rule used to validate the address provided in the binding call should allow the reuse of the local address.
- 'tcp-nodelay': Set this option to true would disable the Nagle algorithm for connection.
- 'tcp-keepidle': Only applicable to the value of TCP_KEEPIDLE Linux. (Unit: second)
- 'tcp-keepcnt': Only applicable to the value of TCP_KEEPCNT Linux.
- 'tcp-keepintv1': Only applicable to the value of TCP_KEEPINTVL Linux.
- 'ipv6-v6only': Set this option to true would restrict the inET6 socket to only sending and receiving IPv6 packets.

tcp:setstats()

Prototype:

- master:setstats(received, sent, age)
- client:setstats(received, sent, age)
- server:setstats(received, sent, age)

Function: Reset accounting information on sockets. It is useful for bandwidth limiting.

Parameters:

- received: Received is a number with the number of new received bytes
- shend: Sent is a number with the number of new Sent bytes.
- age: Age is the new Age in seconds.

Return:

- Success: Return 1.
- Failure: return nil

tcp:settimeout()

Prototype:

- master:settimeout*value [, mode])
- client:settimeout(value [, mode])server:settimeout(value [, mode])

Function: Change the timeout value of tcp object.

Parameters:

- `value(number)`: The amount of time to wait. nil and negative value are indefinite block. (Unit: second)
- `mod(string)`: timeout mode

Return: None

Note:

1. By default, all the I/O operations are blocked. That is, any call to the method send, receive and accept will indefinitely block until the operation is complete. The method settimeout define a limit on the amount of time that n I/O method can block. When timeout is set and after a specified amount of time, the affected method aborts and fails with an error code.
2. There are two timeout modes:

- 'B': block timeout. It specifies the maximum time limit that the operating system can block LuaSocket while waiting for any single I/O operation to complete. This is the default mode.
- 't' : total timeout. It specifies the maximum time limit that LuaSocket can block Lua scripts before the call returns

1. Although the timeout value in LuaSocket has millisecond precision

tcp:shutdown()

Prototype: client:shutdown(mode)

Function: Disable some full-duplex connections

Parameters:

- mode: Close connection mode
  - "both": Close send and receive connection (default mode)
  - "send": Close send connection
  - "receive": Close receive connection

Return:

- Success: Return 1.
- Failure: Return nil and error.

tcp:getfamily()

Prototype:

- master:getfamily()
- client:getfamily()
- server:getfamily()

Function: Obtain tcp protocol family

Parameters: None

Return:

- Return a protocol family string

tcp:setpeername()

Prototype: client:setpeername(address, port)

Function: Change the host object to connect to a remote host.

Parameters:

- address(string): IP address or host name
- port(number): port number

Return:

- Success: Return 1.
- Failure: Return nil and error.

tcp:setsockname

Prototype: server:setsockname(address, port)

Function: Change the address and port number of the objectified binding.

Parameters:

- address(string): IP address or host name
- port(number): port number

Return:

- Success: Return 1.
- Failure: Return nil and error.

UDP module

udp:send()

Prototype: connected:send(datagram)

Function: Sends datagrams to connected udp objects.

Parameters:

- datagram(string): Datagram is a string with the datagram contents

Return:

- Success: Returns the number of bytes sent.
- Failure: return nil and error.

Note:

- The maximum datagram size for UDP is 64K minus IP layer overhead. However datagrams larger than the link layer packet size will be fragmented, which may deteriorate performance and reliability.
- In UDP, the send method never blocks and the only way it can fail is if the underlying transport layer refuses to send a message to the specified address (i.e. no interface accepts the address).
- If the destination does not exist, the packet is discarded and no error is returned.

udp:sendto()

Prototype: unconnected:sendto(datagram, ip, port)

Function: Sends datagrams to connected udp objects.

Parameters:

- datagram(string): Datagram is a string with the datagram contents
- ip(string): Destination IP. Host names are not allowed due to performance.
- port(number): Destination port

Return:

- Success: Returns 1.
- Failure: return nil and error.

Note:

- The maximum datagram size for UDP is 64K minus IP layer overhead. However datagrams larger than the link layer packet size will be fragmented, which may deteriorate performance and reliability.
- In UDP, the send method never blocks and the only way it can fail is if the underlying transport layer refuses to send a message to the specified address (i.e. no interface accepts the address).

udp:receive()

Prototype:

- connected:receive([size])
- unconnected:receive([size])

Function: Receives a datagram from the UDP object. If the UDP object is connected, only datagrams coming from the peer are accepted. Otherwise, the returned datagram can come from any host.

Parameters:

- size(number): Specifies the maximum size of the datagram to be retrieved

Return:

- Success: Returns the datagram received.
- Failure: return nil and error.

Note:

- If there are more than size bytes available in the datagram, the excess bytes are discarded. If there are less then size bytes available in the current datagram, the available bytes are returned. If size is omitted, the compile-time constant socket._DATAGRAMSIZE is used (it defaults to 8192 bytes). Larger sizes will cause a temporary buffer to be allocated for the operation.

udp:receivefrom()

Prototype: unconnected:receivefrom([size])

Function: It works exactly as the receive method, except it returns the IP address and port as extra return values, and is therefore slightly less efficient.

Parameters:

- size(number): Specifies the maximum size of the datagram to be retrieved

Return:

- `dgram(string)`: received data
- `adderss(string)`: IP address
- `port(number)`: Port

udp:close()

Prototype:

- connected:close()
- unconnected:close()

Function: Closes a UDP object. The internal socket used by the object is closed and the local address to which the object was bound is made available to other applications. No further operations (except for further calls to the close method) are allowed on a closed socket.

Parameters: None

Return: None

Note:

- It is important to close all used sockets once they are not needed, since, in many systems, each socket uses a file descriptor, which are limited system resources. Garbage-collected objects are automatically closed before destruction, though.

udp:gettimeout()

Prototype:

- connected:settimeout( )
- unconnected:settimeout( )

Function: Obtains the current timeout value.

Parameters: None

Return:

- Returns the current timeout value.

udp:settimeout()

Prototype:

- connected:settimeout(value)
- unconnected:settimeout(value)

Function: Change the timeout value of UDP object.

Parameters:

- value(number): The amount of time to wait is specified as the value parameter in seconds. nil and negative value allow operations to block indefinitely.

Return:

- Returns 1

Note:

- By default, the operations of receive and receiveform are blocked.
- In UDP, the send and sendto methods never block (the datagram is just passed to the OS and the call returns immediately). Therefore, the settimeout method has no effect on them.

udp:getoption()

Prototype:

- connected:getoption()
- unconnected:getoption (optionName)

Function: Gets an option value from the UDP object

Parameters:

- optionName: option name.

Return:

- Success: Returns the option value.
- Failure: Returns nil and error.

Note:

Option names are as follows:

- 'dontroute': Indicates that outgoing messages should bypass standard routing facilities.
- 'broadcast': Requests permission to send broadcast datagrams on sockets.
- 'reuseaddr': Rules that represent the validation of addresses provided in a bind() call should allow the reuse of local addresses.
- 'reuseport': If multiple processes have set 'reusePort' before binding the port, they are allowed to double bind completely.
- 'ip-multicast-loop': Specifies whether a copy of an outgoing multicast datagram is sent to the sending host as long as it is a member of a multicast group.
- 'ipv6-v6only': Specifies whether inet6 sockets are restricted to sending and receiving only IPv6 packets.
- 'ip-multicast-if': Sets the interface for sending multicast datagrams.
- 'ip-multicast-ttl': Sets the lifetime in the IP header for outgoing multicast datagrams.
- 'ip-add-membership': Added to the specified multicast group.
- 'ip-drop-membership': Leaves the specified multicast group.

udp:getpeername()

Prototype:

- connected:getpeername()
- unconnected:getpeername()

Function: Retrieves information about the peer associated with a connected UDP object.

Parameters: None

Return:

- Success: Returns a string with the IP address of the peer, the port number that peer is using for the connection, and a string with the family ("inet4" or "inet6").
- Failure: return nil and error.

Note:

- It makes no sense to call this method on unconnected objects.

udp:getsockname()

Prototype:

- connected:getsockname()
- unconnected:getsockname()

Function: Returns the local address information associated to the object.

Parameters: None

Return:

- Success: Returns a string with local IP address, a number with the local port, and a string with the family ("inet4" or "inet6").
- Failure: return nil and error.

Note:

- UDP sockets are not bound to any address until the setsockname or the sendto method is called for the first time (in which case it is bound to an ephemeral port and the wild-card address).

udp:setoption()

Prototype:

- connected:setoption(option [, value])
- unconnected:setoption(option [, value])

Function: Sets options for the UDP object.

Parameters:

- option(string): Option name
- value: Option value and it depends on the option being set.

Return:

- Success: Returns 1
- Failure: return nil and error.

Note:

Options are only needed by low-level or time-critical applications. You should only modify an option if you are sure you need it.

Option name are as follows:

- 'dontroute': Indicates that outgoing messages should bypass the standard routing facilities. Receives a boolean value.
- 'broadcast': Requests permission to send broadcast datagrams on the socket. Receives a boolean value.
- 'reuseaddr': Indicates that the rules used in validating addresses supplied in a bind() call should allow reuse of local addresses. Receives a boolean value.
- 'reuseport': Allows completely duplicate bindings by multiple processes if they all set 'reuseport' before binding the port. Receives a boolean value.
- 'ip-multicast-loop': Specifies whether or not a copy of an outgoing multicast datagram is delivered to the sending host as long as it is a member of the multicast group. Receives a boolean value.
- 'ipv6-v6only': Specifies whether to restrict inet6 sockets to sending and receiving only IPv6 packets. Receive a boolean value.
- 'ip-multicast-if': Sets the interface over which outgoing multicast datagrams are sent. Receives an IP address.
- 'ip-multicast-ttl': Sets the Time To Live in the IP header for outgoing multicast datagrams. Receives a number.
- 'ip-add-membership': Joins the multicast group specified. Receives a table with fields multiaddr and interface, each containing an IP address.
- 'ip-drop-membership': Leaves the multicast group specified. Receives a table with fields multiaddr and interface, each containing an IP address.

udp:setpeername()

Prototype:

- connected:setpeername('*')
- unconnected:setpeername(address, port)

Function: Changes the peer of a UDP object.

Parameters:

- "* "(string): If address is '*' and the object is connected, the peer association is removed and the object becomes an unconnected object again.
- address(string), port(number) —— address could be IP address or hostname. Port is the port number.

Return:

- Success: Returns 1.
- Failure: Returns nil and error.

Note:

- This method converts an unconnected UDP object into a connected UDP object or vice versa.
- For connected objects, the outgoing datagrams will be sent to the specified peer, and datagrams received from other peers will be discarded by the OS. The connected UDP objects must use the sand and receive methods instead of sendto and receiveform.
- Since the address of the peer does not have to be passed to and from the OS, the use of connected UDP objects is recommended when the same peer is used for several transmissions and can result in up to 30% performance gains.
- Starting with LuaSocket 3.0, the host name resolution depends on whether the socket was created by socket.udp and socket.udp6. Addresses from the appropriate family are tried in succession until the first success or until the last failure.

udp:setsockname()

Prototype: unconnected:setsockname(address, port)

Function: Binds the UDP object to a local address.

Parameters:

- address(string): Bind address. The address can be an IP address or a host name. If address is '*', the system will bind to all local interfaces using the constant INADDR_ANY.
- port(number): Port number. It the port is 0, the system chooses an ephemeral port.

Return:

- Success: Returns 1.
- Failure: Returns nil and error.

Note:

- This method could only be called before any datagram is sent through the UDP object, and only once. Otherwise, the system automatically binds the object to all local interfaces and chooses an ephemeral port as soon as the first datagram is sent. After the local address is set, either automatically by the system or explicitly by socketname, it cannot be changed.

udp:getfamily()

Prototype:

- connect:getfamily
- unconnect:getfamily

Function: Returns a TCP family

Parameters: None

Return:

- Success: Returns a protocol family string
- Failure: Returns nil and error.

ftp module

ftp.command()

Prototype: ftp.command(cmdt)

Function: Send ftp command

Parameters:

- `cmdt(table)`: ftp command list

  {

  ? host = `string`, -- address

  ? command = `string`, -- ftp command

  ? port = number, -- port

  ? user = string, -- username

  ? password = string or table , -- password

  ? argument = string or table, -- upload parameter

  ? check = `function`, -- Check function

  ? create = function, -- Create a connection function

  }

Return:

- Success: Returns 1.
- Failure: Returns nil and error.

Note:

- This function only sends commands and does not receive data. To receive data, you need to create another channel.

Example

```
-- 1. Introduce modules
-- local ftp = require("socket.ftp")
-- local LTN12 = require("ltn12")

-- 2. Send command
local tabRecv = {}              -- Create a table to receive data
local state,err = ftp.command({
      host= "192.168.50.225",      -- Set ftp server address to "192.168.50.225"
      port= 21,                    -- Set ftp server port to 21
--      user = ftp.USER,             -- The password is set to the default anonymous account, so it can be omitted
--      password = ftp.PASSWORD,
      command= "DELE",        -- Set to send ftp command "DELE". For details about the command, see the appendix
      argument= "/test.png",    -- Set the Delete command parameter "/test.png", (delete "/test.png" file)
      sink= LTN12.sink.table(tabRecv) -- Configure the receiver
   })
-- 3. Print result
if state and err==nil then
  print("dele '/test.png' success ! ")
else
  print("faild !",err)
```

```
end
```

ftp.genericform()

Prototype: ftp.genericform(u, b)

Function: Request parameter conversion. The simple URL request mode convert to a general table table parameter form.

Parameters:

- u(string): The requested URL field
- b(string): The requested body field

Return:

- General request table parameter

Example

```
local ftp = require("socket.ftp")
local json = require("json")

local server_ip = "192.168.50.225"
local server_file1 = "/chenGH/ftpput.txt"

local url = "ftp://"..server_ip..server_file1
local t = assert(ftp.genericform(url))
print(type(t),json.encode(t))
--------------------------------------------------------------------------------Output#
-- table   {"path":"/chenGH/ftpput.txt","scheme":"ftp","host":"192.168.50.225","authority":
--"192.168.50.225"}
```

ftp.put()

Prototype 1: ftp.**put(** url, content **)**

Function: Upload url data

Parameters:

- url: Resource identification locator
- content: Data to be uploaded

Return:

- Success: Return 1.
- Failure: Return nil and error.

Prototype 2:

ftp.**put{**
host = *string*,
source = *LTN12 sink*,
argument *or* path = *string*,
[user = *string*,]
[password = *string*]
[command = *string*,]
[port = *number*,]
[type = *string*,]
[step = *LTN12 pump step*,]
[create = *function*]
}

Function: Data upload is controlled by fields

Parameters:

- host: houstname, IP address (Required parameters)
- source: LTN12 object (Required parameters)
- argument or path: Upload parameter/request path (Required parameters)
- user: username
- password: password
- command: Used to send data. Defaults to stor.
- port: Used for the control connection. Defaults to 21
- type: The transfer mode. Can take values "i" or "a" (Binary or ASCII). The default value is the server default value.
- step: Used to tranmist data from the server to the sink. Defaults to the LTN12 pump.step function.
- create: An optional function to be used instead of socket.tcp when the communications socket is created.

Return:

- Success: Return 1.
- Failure: Return nil and error.

Example

```
local ftp = require("socket.ftp")
local ltn12 = require("ltn12")

local server_ip = "192.168.50.225"
-- Create a file if the file does not exist, overwrites the original data if the file exists,
and reports an error if the folder does not exist
local server_file1 = "/chenGH/ftpput.txt"
local server_file2 = "/chenGH/ftpput.dll"

-- ftp.put Data push format 1, simple mode, push simple data
local url = "ftp://"..server_ip..server_file1
local ret,msg = ftp.put(url,"hello server ! this is put data !")
if ret then
  print("put data success !")
else
  print(msg)
end

-- ftp.put Data push format 2, You can configure parameters to push complex data or files
local put_param = {}  -- Creatt a parameter table
put_param.host = server_ip
put_param.path = server_file2
put_param.source = ltn12.source.file(io.open("user:ftpTest.dll","rb"),"connot open file !")
local ret = assert(ftp.put(put_param))
if ret then
  print("put data success ! ")
end
```

ftp.get()

Prototype 1: ftp.**get(** url, content **)**

Function: Download the contents of a URL and returns it as a string.

Parameters:

- 【string】 url —— url resource identifier

Return:

- Success: Return the string content that the url points to.
- Failure: Return nil and error.

Note:

- The cached string size cannot exceed 1 MB. If the string size exceeds 1 MB, data processing is performed throuhg LTN2.

Prototype 2:

ftp.**get{**
host = *string*,
sink = *LTN12 sink*,
argument *or* path = *string*,
[user = *string*,]
[password = *string*]
[command = *string*,]
[port = *number*,]
[type = *string*,]
[step = *LTN12 pump step*,]
[create = *function*]
}

Function: Download is controlled by parameter fields.

Parameters:

- host: houstname, IP address
- source: LTN12 object
- argument or path: Request parameter/request path
- user: username. Default to "ftp".
- password: password. Default to "anonymous@anonymous.org".
- command: Used to get ftp command of data. Defaults to retr.
- port: Used for the control connection. Defaults to 21.
- type: The transfer mode. Can take values "i" or "a". The default value is the server default value.
- step: Used to tranmist data from the server to the sink. Defaults to the LTN12 pump.step()
- create: An optional function to be used instead of socket.tcp when the communications socket is created.

Return:

- Success: Return 1.
- Failure: Return nil and error.

Note:

- If the size of the retrieved file exceeds the size of the cache (1M), the receiver (prototype 2) is used to block the data or the retrieved data will fail.
- The IO module of Lua is required to use the file receiver. Currently, the IO module of Hmi is not open, so files exceeding 1M cannot be obtained temporarily.

Example

```
local ftp = require("socket.ftp")
local ltn12 = require("ltn12")

local server_ip       = "192.168.50.225" -- server address
local server_port     = "21"    -- default to 21#If there is special, it can be changed.
local server_user     = "ftp"   -- server name, default to "ftp"
-- server passpword, default to "anonymous@anonymous.org"
local server_password  = "anonymous@anonymous.org"
local server_file1    = "/chenGH/ftpTest.txt"       -- Server destination file#File 1 you want to download#
local server_file2    = "/chenGH/ftpTest.dll"       -- Server destination file#File 2 you want to download#
-- ftp.get get format 1#Enter the URL of the file directly
-- Note: The size of the file cannot exceed the buffer size (1M), and exceeding will result in acquisition failure.
```

```
local ftpUrl = "ftp://"..server_ip..server_file1
local data,msg = ftp.get(ftpUrl)
if not data then
  print("ftp_get",msg)
  return nil
else
  print(data)
  assert(flash.file_write("user:ftpTest.txt", "w+", data))    -- Create a local ftptest.txt file and write data to it
end
-- ftp.get format 2#configure ftp.get paramter
local dataTab = {}
local getRequest = {}
getRequest.host = server_ip
getRequest.path = server_file2
-- io operation temporarily blocks I/O file operations in HMI. Therefore, files that exceed 1 MB cannot be written.
getRequest.sink = ltn12.sink.file(io.open("user:ftpTest.dll", "wb"))
assert(ftp.get(getRequest))
```

http module

http.request()

Prototype 1: http.request(url [, body])

Function: http request function

Parameters:

- url(string): Website address, uniform resource locator.
- body: The requested data body

Return:

- The simple form returns the response body as a string, followed by a response status code, a response header, and a response status line.

Example

```
-- 1. Introduce modules
local http = require("socket.http")
local json = require("json")

-- 2. Constant and variable definitions
local weather_url = "http://t.weather.itboy.net/api/weather/city"
local city_code = "101010100"   -- Beijing

-- 3. Get weather information
local url = weather_url.."/"..city_code
print(url)
str_weatherInfo = http.request(url)
-- print(str_weatherInfo)
tab_weatherInfo = json.decode(str_weatherInfo)  -- Convert the weather information in JSON format to a table
print("wendu =",tab_weatherInfo["data"]["wendu"])        -- Get the temperature information in the table
-----------------------------------------------------Output#
--http://t.weather.itboy.net/api/weather/city/101010100
--wendu = 7
```

Prototype 2: http.**request{**
url = *string*,
[sink = *LTN12 sink*,]
[method = *string*,]

[headers = *header-table*,]
[source = *LTN12 source*],
[step = *LTN12 pump step*,]
[proxy = *string*,]
[redirect = *boolean*,]
[create = *function*]
}

Function: http request function

Parameters:

- url: website address, uniform resource locator. (required parameter)
- sink: LTN12 reveiver (optional parameter)
- method: http request method, default to "get". (optional parameter)
- headers: Any other HTTP headers sent with the request. (optional parameter)
- source: *simple* LTN12 source to provide the request body. If there is a body, you need to provide an appropriate "content-length" request header field, or the function will attempt to send the body as "chunked" (something few servers support). Defaults to the empty source. (optional parameter)
- step: Used to tranmist data from the server to the sink. Defaults to the LTN12 pump.step function. (optional parameter)
- proXY: The URL of the proxy server to use. The default is no proxy. (optional parameter)
- tredirect: Set to false to prevent this feature from automatically following 301 or 302 server redirection messages. (optional parameter)
- create: An optional function to be used instead of socket.tcp when the communications socket is created. (optional parameter)

Return:

- Success: Return 1, and stores the data to the receiver.
- Failure: Return nil and error.

Example

```
-- 1. Introduce modules
local http = require("socket.http")
local json = require("json")
local LTN12= require("ltn12")
-- 2. Constant and variable definitions
local weather_url = "http://t.weather.itboy.net/api/weather/city"
local city_code = "101010100"   -- Beijing
local tabWeatherInfo = {}

-- 3. Get weather information
local url = weather_url.."/"..city_code
print(url)

local tabRequest = {}
tabRequest.url = url
tabRequest.sink = LTN12.sink.table(tabWeatherInfo)

status,str_weatherInfo = http.request(tabRequest)
print(type(tabWeatherInfo[1]))
print("str_weatherInfo =",tabWeatherInfo[1])
-- Convert the weather information in JSON format to a table. Since the LTN12.sink.table receiver is used to receive,
the accepted result is present in the tabWeatherInfo table.The receive occurs only once, and it is stored at the tabWeatherInfo[1].
local tab_weatherInfo = json.decode(tabWeatherInfo[1])
print("wendu =",tab_weatherInfo["data"]["wendu"])      -- Get the temperature information in the table.
```

http.genericform()

Prototype : http.genericform(u, b)

Function: Convert a simple URL to a generic table parameter

Parameters:

- `u(string)`: The URL field that http requested
- `b(string)`: The body field that http requested

Return:

- Generic table parameter

Example

```
-- 1. Introduce modules
local http = require("socket.http")

-- 2. Constant and variable definitions
local url = "http://t.weather.itboy.net/api/weather/city/101010100"

ta = http.genericform(url)
print(type(ta))
print("ta.utl     =",ta.url)
print("ta.sink    =",ta.sink)
print("ta.method  =",ta.method)
print("ta.heards  =",ta.heards)
print("ta.source  =",ta.source)
print("ta.step    =",ta.step)
print("ta.proxy   =",ta.proxy)
print("ta.redirect=",ta.redirect)
print("ta.create  =",ta.create)
------------------------------Output#
--table
--ta.utl     =    http://t.weather.itboy.net/api/weather/city/101010100
--ta.sink    =    function: 060405E0
--ta.method  =    nil
--ta.heards  =    nil
--ta.source  =    nil
--ta.step    =    nil
--ta.proxy   =    nil
--ta.redirect=    nil
--ta.create  =    nil
```

https module

https.request()

Prototype 1: https.request(url [, body])

Function: https request fucntion

Parameters:

- url(string): Website address, uniform resource locator.
- body: The requested data body

Return:

Return value position	Meaning
1	The returned data result
2	The access result code. 200 indicates that the connection is successful
3	The returned response header line
4	The returned response status

Example

```
-- 1. Introduce modules
local https = require("socket.https")
local LTN12= require("ltn12")
function test()
  print_debug(1)
 print("wait request . . . ")
 local res, code, headers, status = https.request{
      url= "https://www.we-con.com.cn/"
   }

  print("res=",res)--The returned data result
  print("code=",code)--The access result code. 200 indicates that the connection is successful
  for i,v in pairs(headers) do --The returned response header line
     print("i,v=",i,v)
  end
  print("state=",status)--The returned response status
   we_bas_setstring("@W_HDW4600", status)
end
```

Prototype 2: http.**request{**
url = *string*,
[sink = *LTN12 sink*,]
}

Function: https request fucntion

Parameters:

- url(string): Website address, uniform resource locator. (required paramter)
- sink: The requested data body. (optional parameter)

Return:

Return value position	Meaning
1	The returned data result
2	The access result code. 200 indicates that the connection is successful
3	The returned response header line
4	The returned response status

Example

```
-- 1. Introduce modules
local https = require("socket.https")
local LTN12= require("ltn12")
function test2()
   print_debug(1)
  local url = "https://www.we-con.com.cn/"
  local Result = {}

  local tabRequest = {}
```

```
    tabRequest.url = url
    tabRequest.sink = LTN12.sink.table(Result)

  print("wait request . . . ")
  local res, code, headers, status = https.request(tabRequest)
  --print(type(Result[1]))
  print("Result =",table.concat(Result))
  print("res",res)--The returned data result
  print("code =", code)--The access result code. 200 indicates that the connection is successful
  for i,v in pairs(headers) do --The returned response header line
     print(i,v)
  end
  print("status=",status)--The returned response status

    we_bas_setstring("@W_HDW4600", status)
end
```

SMTP module

smtp.message()

Prototype: smtp.message(mesgt)

Function: Generate mail format messages

Parameters:

- mesgt(table): Message format

```
-- Message format:
mesgt = {
  headers = header-table,    -- message header
  body = LTN12 source or string or multipart-mesgt -- message text
}
multipart-mesgt = {
  [preamble = string,] -- preface
  [1] = mesgt,   -- text content 1
  [2] = mesgt,   -- text content 2
  ...
  [n] = mesgt,   -- text content n
  [epilogue = string,] -- Conclusion
}
```

Return:

- Returns a mail message source used to send messages

```
source = smtp.message
{
 headers =                              -- mail header
    {
      -- Note: smtp.send ignores headers
       from= "chenGH <chen__GH@163.com>",          -- sender
       to  = "chenGH <chengh@we-con.com.cn>",      -- recipient
       cc  = "<3388545257@qq.com>",                -- cc
       subject= "Test the mail sending function of the LUa SMTP module!"    -- title
    },
    body =                              -- mail content
    {
       preamble= "Thanks for participating in the Lua SMTP mail test!",      -- preface
       [1] =                                -- text 1#text#
```

```
    {
      -- The first part has no header, which means it is plain text, encoded in ASCII.
      -- mime.eol is an underlying function, and used to normalize trailing line break marks
      body= mime.eol(0, [[
          Lines in the message body should always end in CRLF (carriage return line feed)
          The Smtp module does not process the message content。 The execution process is as follows
           A: Package the message content via smtp.message
           B: Package the message content via smtp.send and send the message.(Contains attachment data fill
sections)
          ]])
    },
    [2] =                                   -- text 2 (picture)
    {
      -- Part two: The title describes the content as a png image.
      -- Transfer content based on base64 encoding format
      -- Note: Nothing happens until the message is actually sent, and the data is assembled just after it is sent.
      headers=
      {
        -- Declare the type and name of the content of the second part of the message#type#image/png#name#
image.png"#
          ["content-type"] = 'image/png; name="image.png"',
          ["content-disposition"] = 'attachment; filename="image.png"',   -- content description
          ["content-description"] = 'a beautiful image',                  -- mail description
          ["content-transfer-encoding"] = "BASE64"                        -- Mail encoding format
      },
      body= LTN12.source.chain(                                 -- Data source connection
          LTN12.source.file(io.open("user:smtp.png","rb")),          -- Open a file as the data source
          LTN12.filter.chain(                                   -- Encoding format is base64
              mime.encode("base64"),
              mime.wrap()
          ))
    },
    epilogue= "This might also show up, but after the attachments"     -- Conclusion
  }
}
```

smtp.send()

Prototype : smtp.**send{**
from = *string*,
rcpt = *string* or *string-table*,
source = *LTN12 source*,
[user = *string*,]
[password = *string*,]
[server = *string*,]
[port = *number*,]
[domain = *string*,]
[step = *LTN12 pump step*,]
[create = *function*]
}

Function: send email

Parameters:

- from: sender
- rcpt: recipient list
- source: message source. The message after packaging
- user: account. It is for mailbox authentication
- password: password. It is for mailbox authenticatio. (base64 format)

- server: smtp server address. e.g., the server of 163 mail is smtp.163.com; the server of QQ mail is smtp.qq.com.
- port: smtp server port, default to 25. 163 Mailbox smtp port is 25, qq mailbox smtp server port number is 587/465.
- domain: domain information. Default to "localhost".
- step: Used to pass data from the source to the server. Defaults to the LTN12 pump.step function.
- create: Used to create socket connection. Default to socket.tcp().

Return:

- Success: Return 1.
- Failure: Return nil and error.

Note:

1. SMTP servers can be very picky with the format of e-mail addresses. To be safe, use only addresses of the form "<fulano@example.com>" in the from and rcpt arguments of the send function. In headers, e-mail addresses can take whatever form you like.
2. Only recipients specified in the rcpt list will receive a copy of the message. Each recipient of an SMTP mail message receives a copy of the message body along with the headers, and nothing more. The headers *are* part of the message and should be produced by the LTN12 source function. The rcpt list is *not* part of the message and will not be sent to anyone.

- To: Contains the address(es) of the primary recipient(s) of the message.
- Cc: Contains the addresses of others who are to receive the message, though the content of the message may not be directed at them (where the "Cc" means "Carbon Copy" in the sense of making a copy on a typewriter using carbon paper).
- Bcc: Contains addresses of recipients of the message whose addresses are not to be revealed to other recipients of the message (where the "Bcc" means "Blind Carbon Copy")

1. The LuaSocket send function does not care or interpret the headers you send, but it gives you full control over what is sent and to whom it is sent:

- If someone is to receive the message, the e-mail address *has* to be in the recipient list. This is the only parameter that controls who gets a copy of the message.
- If there are multiple recipients, none of them will automatically know that someone else got that message. That is, the default behavior is similar to the Bcc field of popular e-mail clients;
- It is up to you to add the To header with the list of primary recipients so that other recipients can see it.
- It is also up to you to add the Cc header with the list of additional recipients so that everyone else sees it.
- Adding a header Bcc is nonsense, unless it is empty. Otherwise, everyone receiving the message will see it and that is exactly what you *don't* want to happen.

Example

```
local smtp = require("socket.smtp")
local LTN12= require("ltn12")
local mime = require("mime")
from = "<chen__GH@163.com>"        -- sender#Angle brackets must be added or some errors may occur#
rcpt = {                           -- send list#Angle brackets must be added or some errors may occur#
  "<chengh@we-con.com.cn>",
  "<1125920709@qq.com>",
  "<3388545257@qq.com>"
}
-- Create a sending source that is divided into two parts, one for plain text and the other for the PNG image#
source = smtp.message
{
   headers=                        -- mail header
   {
     -- Note: smtp.send ignores headers
      from= "chenGH <chen__GH@163.com>",      -- sender
      to  = "chenGH <chengh@we-con.com.cn>",  -- recipient
      cc  = "<3388545257@qq.com>",            -- CC
      subject= "Test the mail sending function of the LUa SMTP module!! "   -- title
```

```
    },
    body =                              -- mail text
    {
        preamble= "Thanks for participating in the Lua SMTP mail test!", -- preface
        [1] =                           -- text 1#plain text#
        {
          -- The first part has no header, which means it is plain text, encoded in ASCII.
          -- mime.eol is an underlying function, and used to normalize trailing line break marks
            body= mime.eol(0, [[
                Lines in the message body should always end in CRLF (carriage return line feed)
                The Smtp module does not process the message content。 The execution process is as follows
                 A: Package the message content via smtp.message
                 B: Package the message content via smtp.send and send the message.(Contains attachment data fill
sections)
                ]])
        },
        [2] =                           -- text 2 (picture)
        {
          -- Part two: The title describes the content as a png image.
          -- Transfer content based on base64 encoding format
          -- Note: Nothing happens until the message is actually sent, and the data is assembled just after it is sent.
            headers=
            {
              -- Declare the type and name of the content of the second part of the message#type#image/png#name#
image.png"#
                ["content-type"] = 'image/png; name="image.png"',
              -- content description
                ["content-disposition"] = 'attachment; filename="image.png"',
              -- mail description
                ["content-description"] = 'a beautiful image',
              -- Message encoding format
                ["content-transfer-encoding"] = "BASE64"
            },
            body= LTN12.source.chain(           -- Data source connection
                LTN12.source.file(io.open("user:smtp.png","rb")), -- Open a file as the data source
                LTN12.filter.chain(
                    mime.encode("base64"),   -- Encoding format is base64
                    mime.wrap()
                ))
        },
        epilogue= "This might also show up, but after the attachments"  -- Conclusion
    }
}
-- send email
r, e = assert(smtp.send{
        from= from,         -- mail sender
        rcpt= rcpt,         -- Mail receiving list
       -- smtp server. e.g.,qq mail is smtp.qq.com; enterprise mail is smtp.qiye.com#
        server= "smtp.163.com",
        user= "chen__GH@163.com",      -- account
        password= "HUCOYKAUIKLSGZFJ",  -- password#the password after conversion to base64#

       -- smtp server port,You need to look it up yourself#qq mailbox open 587/465, 163 mailbox port 25
       --#25 is the smtp default port#
       -- port = 587,
        source= source,             -- message source    } )
print("end success!! ")
```

Auxiliary modules

LTN12.filter.chain()

Prototype : ltn12.filter.chain(filter1, filter2 [, ... filterN])

Function: Generates a set of connected filters, and the nesting of filters can be arbitrary

Parameters:

- filter1 to filter N are simple filters

Return:

- Returns the connected filter

LTN12.filter.cycle()

Prototype : ltn12.filter.cycle(low [, ctx, extra])

Function: Returns a high-level filter that cycles though a low-level filter by passing it each chunk and updating a context between calls.

Parameters:

- `low:` The low-level filter to be cycled.
- `ctx:` The initial context.
- `extra:` Any extra argument the low-level filter might take.

Return:

- Returns the high-level filter.

LTN12.pump.all()

Prototype : ltn12.pump.all(source, sink)

Function: Pumps all data from a source to a sink.

Parameters:

- `source:` data source
- `sink:` sink

Return:

- Success: Return true.
- Failure: Return false and error.

LTN12.pump.step()

Prototype : ltn12.pump.step(source, sink)

Function: Pumps a large chunk of data from the source to the sink.

Parameters:

- `source:` source
- `sink:` sink

Return:

- Success: Return true.
- Failure: Return false and error.

LTN12.sink.chain()

Prototype : ltn12.sink.chain(filter, sink)

Function: Creates a new sink that passes data through a filter before sending it to a given sink.

Parameters:

- source: filter
- sink: sink

Return:

- Return the created sink.

LTN12.sink.error()

Prototype : ltn12.sink.chain(filter, sink)

Function: Creates a new sink that passes data through a filter before sending it to a given sink.

Parameters:

- `filter:` filter
- `sink:` sink

Return:

- Returns the new sink that was created.

LTN12.sink.file()

Prototype : ltn12.sink.file(handle, message)

Function: Create a sink that sends data to a file

Parameters:

- handle: file handle
- message: If handle is **nil**, message should give the reason for failure

Return:

- Returns the new sink that was created.

Note:

- The function returns a sink that sends all data to the given handleThe fu and closes the file when done, or a sink that aborts the transmission with the error message.
- This function is used with io.open(): ltn12.sink.file(io.open("copy.png", "wb"))

LTN12.sink.null()

Prototype : ltn12.sink.null()

Function: Returns a sink that ignores all data it receives.

Parameters: None

Return:

- Returns the new sink that was created.

LTN12.sink.simplify()

Prototype : ltn12.sink.simplify(sink)

Function: Creates and returns a simple sink given a fancy sink.

Parameters:

- `sink:` sink

Return:

- Returns the new sink that was created.

LTN12.sink.table()

Prototype : ltn12.sink.table([table])

Function: Creates a sink that stores all chunks in a table. The chunks can later be efficiently concatenated into a single string.

Parameters:

- table: Used to hold the chunks. If it is nil, the function creates its own table.

Return:

- Returns the sink and the table used to store the chunks.

LTN12.source.cat()

Prototype : ltn12.source.cat(source1 [, source2, ..., sourceN])

Function: Creates a new source that produces the concatenation of the data produced by a number of sources.

Parameters:

- source 1 to source n are the original sources.

Return:

- Returns the new source created.

LTN12.source.chain()

Prototype : ltn12.source.chain(source, filter)

Function: Creates a new source that passes data through a filter before returning it.

Parameters:

- source: original source
- filter: filter

Return:

- Returns the new source created.

LTN12.source.empty()

Prototype : ltn12.source.empty()

Function: Creates and return a new source.

Parameters: None

Return:

- Returns the new source created.

LTN12.source.error()

Prototype : ltn12.source.**error(message**)

Function: Creates and returns a source that aborts transmission with the error message.

Parameters:

- message: error message

Return:

- Returns the new source created.

LTN12.source.file()

Prototype : ltn12.source.file(handle, message)

Function: Creates a source that produces the contents of a file.

Parameters:

- handle: file handle
- message: If handle is **nil**, message should give the reason for failure

Return:

- Returns the new source created.

Note:

- The function returns a source that reads chunks of data from given handle and returns it to the user, closing the file when done, or a source that aborts the transmission with the error message.

LTN12.source.simplify()

Prototype : ltn12.source.simplify(source)

Function: Given a source, create and return a simple source.

Parameters:

- source: the given initial source

Return:

- Returns the new source created.

LTN12.source.string()

Prototype : ltn12.source.string(string)

Function: Creates and returns a source that produces the contents of a string, chunk by chunk.

Parameters:

- string: string source

Return:

- Returns the new source created.

LTN12.source.table()

Prototype : ltn12.source.table(table)

Function: Creates and returns a source that produces the numerically-indexed values of a table successively beginning at 1. The source returns nil (end-of-stream) whenever a nil value is produced by the current index, which proceeds forward regardless.

Parameters:

- table: table data

Return:

- Returns the new source created.

mime.decode()

Prototype :

- mime.decode("base64")
- mime.decode("quoted-printable")

Function: Returns a filter that decodes data from a given transfer content encoding.

Parameters:

- The two encoding formats above.

Return:

- Returns decoder filter.

mime.encode()

Prototype :

- mime.encode("base64")
- mime.encode("quoted-printable" [, mode])

Function: Returns a filter that encodes data according to a given transfer content encoding.

Parameters:

- mode(string): User can specify whether the data is textual or binary, by passing the mode strings "text" or "binary". Mode defaults to "text".

Return:

- Returns code filter.

Note:

- Although both transfer content encodings specify a limit for the line length, the encoding filters do not break text into lines (for added flexibility). Below is a filter that converts binary data to the Base64 transfer content encoding and breaks it into lines of the correct size.

mime.normalize()

Prototype : mime.normalize([marker])

Function: Converts most common end-of-line markers to a specific given marker.

Parameters:

- marker(string): The new marker. It defaults to CRLF. This is the standard end-of-line tag defined by MIME standards.

Return:

- returns a filter that performs the conversion

mime.stuff()

Prototype : mime.stuff()

Function: Creates and returns a filter that performs stuffing of SMTP messages.

Parameters: None

Return:

- Returns a filter that performs stuffing of SMTP messages.

Note:

- The smtp.send function uses this filter automatically. You don't need to chain it with your source, or apply it to your message body.

mime.wrap()

Prototype :

- mime.wrap("text" [, length])
- mime.wrap("base64")
- mime.wrap("quoted-printable")

Function: Returns a filter that breaks data into lines.

Parameters:

- length(number):

Return:

- Returns a filter that performs stuffing of SMTP messages.

Note:

- The smtp.send function uses this filter automatically. You don't need to chain it with your source, or apply it to your message body.

mime.b64()

Prototype : A, B = mime.b64(C [, D])

Function: Low-level filter to perform Base64 encoding.

Parameters:

- C: String that needs to be encoded.
- D: String 2 that needs to be encoded.

Return:

- A is the encoded version of the largest prefix of C..D that can be encoded unambiguously. B has the remaining bytes of C..D, *before* encoding. If D is **nil**, A is padded with the encoding of the remaining bytes of C.

Note:

- The simplest use of this function is to encode a string into it's Base64 transfer content encoding. Notice the extra parenthesis around the call to mime.b64 to discard the second return value.

Example

```
print((mime.b64("diego:password")))
--> ZGllZ286cGFzc3dvcmQ=
```

mime.dot()

Prototype : A, n = mime.dot(m [, B])

Function: Low-level filter to perform SMTP stuffing and enable transmission of messages containing the sequence "CRLF.CRLF".

Parameters:

- m: The data fast value used to determine whether the text concatenation belongs to the same block.
- B: Data that needs to be populated.

Return:

- A is the stuffed version of B. 'n' gives the number of characters from the sequence CRLF seen in the end of B. 'm' should tell the same, but for the previous chunk.

Note:

- The message body is defined to begin with an implicit CRLF. Therefore, to stuff a message correctly, the first m should have the value 2.

Example

```
print((string.gsub(mime.dot(2, ".\r\nStuffing the message.\r\n.\r\n."), "\r\n", "\\n")))
--> ..\nStuffing the message.\n..\n..
```

mime.eol()

Prototype : A, B = mime.eol(C [, D, marker])

Function: Low-level filter to perform end-of-line marker translation.

Parameters:

- C: C is the ASCII value of the last character of the previous chunk, if it was a candidate for line break, or 0 otherwise.
- D: D Is the block data that needs to be translated
- marker: the new end-of-line marker and defaults to CRLF.

Return:

- A:The translated version of D.
- B is the same as C, but for the current chunk.

Example

```
-- translates the end-of-line marker to UNIX
unix = mime.eol(0, dos, "\n")
```

mime.qp()

Prototype : A, B = mime.qp(C [, D, marker])

Function: Low-level filter to perform Quoted-Printable encoding.

Parameters:

- A is the encoded version of the largest prefix of C..D that can be encoded unambiguously. B has the remaining bytes of C..D, *before* encoding. If D is **nil**, A is padded with the encoding of the remaining bytes of C. Throughout encoding, occurrences of CRLF are replaced by the marker, which itself defaults to CRLF.

Return: None

Note:

- The simplest use of this function is to encode a string into it's Quoted-Printable transfer content encoding. Notice the extra parenthesis around the call to mime.qp, to discard the second return value.

mime.qpwrp()

Prototype : A, m = mime.qpwrp(n [, B, length])

Function: Low-level filter to break Quoted-Printable text into lines.

Parameters:

- n: The number of bytes remaining in the first line
- B: Data block
- length: The length of the line, defaults to 76.

Return:

- A : a copy of B.
- m: Returns the number of bytes left in the last line of A.

Note:

- Besides breaking text into lines, this function makes sure the line breaks don't fall in the middle of an escaped character combination. Also, this function only breaks lines that are bigger than length bytes.

mime.unb64()

Prototype : A, B = mime.unb64(C [, D])

Function: Low-level filter to perform Base64 decoding.

Parameters:

- C: Data block 1
- D: Data block 2

Return:

- A is the decoded version of the largest prefix of C..D that can be decoded unambiguously. B has the remaining bytes of C..D, *before* decoding. If D is **nil**, A is the empty string and B returns whatever couldn't be decoded.

Note:

- The simplest use of this function is to decode a string from it's Base64 transfer content encoding. Notice the extra parenthesis around the call to mime.unqp, to discard the second return value.

Example

```
print((mime.unb64("ZGllZ286cGFzc3dvcmQ=")))
--> diego:password
```

mime.unqp()

Prototype : A, B = mime.unqp(C [, D])

Function: Low-level filter to remove the Quoted-Printable transfer content encoding from data.

Parameters:

- C: Code block 1
- D: Code block 2

Return:

- A is the decoded version of the largest prefix of C..D that can be decoded unambiguously. B has the remaining bytes of C..D, *before* decoding. If D is **nil**, A is augmented with the encoding of the remaining bytes of C.

Note:

- The simplest use of this function is to decode a string from it's Quoted-Printable transfer content encoding. Notice the extra parenthesis around the call to mime.unqp, to discard the second return value.

Example

```
print((mime.qp("ma=E7=E3=")))
--> ma??
```

mime.wrp()

Prototype : A, m = mime.wrp(n [, B, length])

Function: Create a low-level filter that uses CRLF tags to split text into several lines.

Parameters:

- n: The number of bytes remaining in the first line
- B: Code block 2
- length: The length of the line, defaults to 76.

Return:

- A is a copy of B, broken into lines of at most length bytes (defaults to 76). 'n' should tell how many bytes are left for the first line of B and 'm' returns the number of bytes left in the last line of A.

Note:

- This function only breaks lines that are bigger than length bytes. The resulting line length does not include the CRLF marker.

Appendix

ftp command list

1. **Access commands**

Command	Format	Instruction
USER	USER	Specifies the login user name for authentication.
PASS	PASS	Specifies the user password, which must be followed by the login user name command
REIN	REIN	Reinitialize the user information. This command terminates the transfer of the current USER and the data being transferred, and then resets all parameters and enables the control connection so that the USER command occurs again on the client.
QUIT	QUIT	Disable the connection to the server

1. **Mode setting command**

Command	Format	Instruction
PASV	PASV	This command tells the FTP server to listen on the specified data port and passively accept requests from clients. If no mode is specified, the FTP server uses PASV mode by default.
PORT	PORT	This command tells the FTP server that the port number monitored by the client is address, and enables the FTP server to connect to the client in active mode.
TYPE	TYPE	This command specifies the data type to be transferred: ASCII and BINARY.
MODE	MODE	The command specifies the transmission mode. S denotes stream, B denotes block, and C denotes compression.

1. **File manegement command**

Command	Format	Instruction
CWD	CWD	Change the working directory; This command allows users to work in different directories or data sets without changing their login information. Directory is usually a directory name or a collection of files related to the system.
PWD	PDW	Returns the current working directory.
MKD	MKD	Creates a new directory in the specified path. Directory is a string for a specific directory.
CDUP	CDUP	Returns to the upper-layer directory.
RMD	RMD	Deletes a specified directory. Directory is a string representing a specific directory.
LIST	LIST	Returns a list of subdirectories and files under the specified path. Name is the path. If the path is omitted, the list of files in the current path is returned.
NLST	NLST	Returns a list of directories under the specified path. If the path is omitted, the current directory is returned.
RNFR	RNFR	Renames the file, and the next command in this command specifies the new file name with RNTO.
RNTO	RNTO	This command works with the RNFR command to rename a file.
DELE	DELE	Used to delete files in the specified path.

1. **File tranfer command**

Command	Format	Instruction
RETR	RETR	Downloads the file in the specified path
STOR	STOR	Uploads a specified file and stores it in the specified location. If the file already exists, the original file will be overwritten. If the file does not exist, a new file will be created.
SYST	SYST	Returns to the operating system used by the server

ftp common response code

Response Code	Meaning
110	Restart the token repl. In this case the text is deterministic, it must be MARK yyyy=mmmm. YYYY is the user proce server marker.
120	Service is ready in N minutes
125	Data connection open, ready for transfer
150	The file is in good condition and will open a data connection
200	Command successful
202	Command not executed
211	System status or system help response
212	Directory status reply
213	File status reply
214	Help message Reply
215	System type reply
220	Service is ready
221	The service closes the control connection and can log out
225	Data connection open, no transfer in progress
226	Closed data connection. Requested file operation succeeded
227	In passive mode
230	User logged in
250	Requested file action completed
257	Create pathname
331	Correct user name, password required
332	Account information is required for login
350	The requested file operation requires further command
421	Unable to provide service, close control connection
425	Unable to open data link
426	Close the connection and terminate the transmission
450	Unavailable file
451	Requested action aborted: local error
452	Requested action not performed: Insufficient system storage space
500	Invalid command
501	syntax error
502	The command was not executed
503	Command order error
504	Invalid command parameter
530	Not logged in
532	Account information is required to store files
550	Requested action not performed
551	Request action aborted: unknown page type
552	Requested file action aborted: storage allocation overflowed
553	Requested action not performed: filename is invalid

LuaSqlite module

Introduction

- Luasqlite is a third-party library of Lua that provides lua operations on sqLite databases.
- Luasqlite is a built-in module that can be used without reference. Module name is luasql_sqlite3.

- It is roughly divided into three parts: SQLite environment Settings, database file operations, database file cursor operations.
- Since sqLite's extensive read and write operations shorten Flash life, database files can only be created on Udisk or SD cards.

1. **Get sqlite environment**

```
env = luasql_sqlite3.sqlite3()
```

1. **sqlite environment operation**

```
-- sqlite environment
env = luasql_sqlite3.sqlite3()
```

1. **db file operation**

```
-- Get objects in database files
env = luasql_sqlite3.sqlite3()-- Initialize the environment before obtaining the db file object
db = env:connect("udisk:test.db")-- Connect a db file
```

Function	Introduction
db:close()	Close the database file
db:escape()	Escape encoding of string data
db:execute()	Execute the sql statement
db:commit()	Commit the data to the database
db:rollback()	Roll back operations on the database before committing
db:setautocommit()	Set the automatic submission function
db:getlastautoid()	Get the total number of data records for the current database +1

1. **Cursor operation**

```
-- Get cursor object
env = luasql_sqlite3.sqlite3()  -- Initialize the environment before obtaining the db file object
db = env:connect("udisk:test.db") -- Connect a db file
cursor = db:execute([[select * from students]])
```

Function	Introduction
cursor:close()	Close the cursor
cursor:getcolnames()	Get the name of the header
cursor:getcoltypes()	Get the type of the header
cursor:fetch()	Similar to iterators, facilitate row nodes

Environment operation

Get sqlite environment

```
sql = require "Luasqlite" -- Introducing the lua libraries
env = sql.sqlite3()  -- Get the environment class
```

env:connect()

Prototype : env:connect(dbFilePath)

Function: Connect to a DB file.(If the file does not exist)

Parameters:

- dbFilePath (string): Full path to the DB file to connect to (create)

Return:

- Success: database object
- Failure: Returns nil and error.

Example

```
env = luasql_sqlite3.sqlite3()-- Get the environment class
db = env:connect("udisk:test.db") -- Connect to a DB file
db:close()-- Close the db Connection
env:close()-- Close the database environment
```

env:close()

Prototype : env:close()

Function: Close the environment

Parameters: None

Return: None

Example

```
env = luasql_sqlite3.sqlite3()-- Get the environment class
db = env:connect("udisk:test.db") -- Connect to a DB file
db:close()-- Close the db Connection
env:close()-- Close the database environment
```

db:close()

Prototype : db:close()

Function: Close connection

Parameters: None

Return: None

Example

```
env = luasql_sqlite3.sqlite3()-- Get the environment class
db = env:connect("udisk:test.db") -- Connect to a DB file
db:close()-- Close the db Connection
env:close()-- Close the database environment
```

db:escape

Prototype : db:escape(szValue)

Function: Connect to a DB file.(If the file does not exist)

Parameters:

- szValue(string): The string that needs to be encoded

Return:

- string: Converted string

Example

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db") -- Connect to a DB file

strSql=db:escape("INSERT INTO student values(1,'lili')")

db:close()      -- Close the db Connection
env:close()      -- Close the database environment
-- printout: INSERT INTO student values(1,"lili")
```

db:execute()

Prototype : db:execute(sql)

Function: execute a SQL statement

Parameters:

- SQL statement executed

Return: It depends on the type of SQL

- If the information is retrieved from a database, the return value is a user-defined data object. (Extracted target data)
- If the statement is a non-fetch data type, return a status to verify that whether the SQL statement was successfully executed

Example

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db") -- Connect to a DB file.
local status=db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)") -- Create a table "student"
status=db:execute("INSERT INTO students values('1','AA')")   -- Insert information  1 AA
status=db:execute("INSERT INTO students values('2','BB')")   -- Insert information  2 BB
status=db:execute("INSERT INTO students values('3','CC')")   -- Insert information  3 CC
status=db:execute([[UPDATE students SET name = 'DD' WHERE id = '3']]) -- Update information
status=db:execute([[DELETE FROM students WHERE id='2']])   -- Delete statement
local cursor=db:execute("SELECT * FROM students") --Query the database
local row = cursor:fetch({},"a")
while row do
print(row.id,row.name)
 row = cursor:fetch({},"a")
end
cursor:close()
db:close()
env:close()
----------Output----------#
```

db:commit()

Prototype : db:commit()

Function: Commit data to the database (executing the execture () function simply stores the data into memory, automatically commit by default (can be set))

Parameters: None

Return: None

Example

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db") -- Connect a db file
local status=db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)") -- Create a table "student"
status=assert(db:setautocommit(false))  -- Set non-auto commit
status=assert(db:execute("INSERT INTO students values('4','AA')"))   -- Insert information 4 AA
status=assert(db:execute("INSERT INTO students values('5','BB')"))   -- Insert information 5 BB
status=assert(db:execute("INSERT INTO students values('6','CC')"))   -- Insert information 6 CC
--statue=assert(db:commit())      -- Manually commit the database
--while 1 do
--At this point, cut out and view the database content, you will find that the data is not commited to the database
--Put the commit() function in front of the dead loop and re-verify that the data has been updated into the database
--end
```

```
statue=assert(db:commit())      -- Manually commit the database
db:close()
```

db:rollback()

Prototype : db:rollback()

Function: Roll back operations on the database until before commit

Parameters: None

Return: None

Exmaple:

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db") -- Connect a db file
local status=db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)") -- Create a table "student"
local status=assert(db:setautocommit(false))  -- Manually commit the database
status=assert(db:execute("INSERT INTO students values('7','AA')"))   -- Insert information 7 AA
statue=assert(db:commit())      -- Commit
status=assert(db:execute("INSERT INTO students values('8','BB')"))   -- Insert information 8 BB
statue=assert(db:rollback())    -- Rollback
status=assert(db:execute("INSERT INTO students values('9','CC')"))   -- Insert information 9 CC
statue=assert(db:commit())      -- Commit
db:close()
env:close()
--------------------------------------------------------------------
-- Only 7 and 9 are inserted in the final result because after 8 we roll back the operation to before commit.
```

db:setautocommit()

Prototype : db:setautocommit(flag)

Function: Set the automatic commit function

Parameters:

- flag(boolean) : Whether to set auto-commit
  - true —— auto-commit
  - false —— Commit mannully

Return: None

Exmaple:

```
env = luasql_sqlite3.sqlite3()  -- Get the environment class
db = env:connect("udisk:test.db") -- Connect a db file
status=assert(db:setautocommit(false))  -- Set non-auto commit
status=assert(db:execute("INSERT INTO students values('10','AA')"))   -- Insert information 10 AA
statue=assert(db:commit())      -- Commit
status=assert(db:execute("INSERT INTO students values('11','BB')"))   -- Insert information 11 BB
statue=assert(db:rollback())    -- Rollback
status=assert(db:setautocommit(true))  -- Set non-auto commit
status=assert(db:execute("INSERT INTO students values('12','CC')"))   -- Insert information 12 CC
statue=assert(db:rollback())    -- Rollback
statue=assert(db:commit())      -- Commit
cursor:close()
db:close()
env:close()
-----------------------------------------
--By comparing the two rollbacks and combining the information actually written to the database (11 is not written to the database, 12 is written to the database)
```

```
--You can see that setting automatic commit as it literally means, commits to the database after the SQL statement is executed.
--(ps: The flash of embedded machine has a life limit. To increase the service life, the operation frequency of flash should be reduced as much as possible#
```

db:getlastautoid()

Prototype : db:getlastautoid()

Function: Gets the total number of data records for the current database

Parameters: None

Return: Total number of data records (number)

Exmaple:

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db") -- Connect a db file
local status=db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)") -- Create a table "student"
status=assert(db:execute("INSERT INTO students values('1','AA')"))   -- Insert information 1 AA
status=assert(db:execute("INSERT INTO students values('2','BB')"))   -- Insert information 2 BB
status=assert(db:execute("INSERT INTO students values('3','CC')"))   -- Insert information 3 CC
status=assert(db:getlastautoid())
print(type(status),status) -- number  4.0
db:close()
env:close()
```

cursor:close()

Prototype : cursor:close()

Function: Close the cursor

Parameters: None

Return: None

Exmaple:

```
cursor:close()
```

cursor:getcolnames()

Prototype : cursor:getcolnames()

Function: Gets the name of the header

Parameters: None

Return: Header (table): the database table header

Exmaple:

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db")     -- Connect a db file
local status=db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)")
local cursor,errorString = db:execute([[select * from students]])
local colNmaes=cursor:getcolnames()
for i,v in ipairs(colNmaes) do
  print(v)
end
cursor:close()
db:close()
```

```
env:close()
----------Output
--id
--name
```

cursor:getcoltypes()

Prototype : cursor:getcoltypes()

Function: Gets the type of the header

Parameters: None

Return:

- headerType(table): Returns a table containing header type information

Exmaple:

```
local env = luasql_sqlite3.sqlite3()  --Get the environment class
local db = env:connect("udisk:test.db")    --Connect a db file
local status=db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)")
status=db:execute("INSERT INTO students values('1','AA')")   --Insert information 1 AA
status=db:execute("INSERT INTO students values('2','BB')")   --Insert information 2 BB
status=db:execute("INSERT INTO students values('3','CC')")   --Insert information 3 CC
local cursor,errorString = assert(db:execute([[select * from students]]))
local colTypes = cursor:getcoltypes()
for i,v in ipairs(colTypes) do
  print(i,v)
end
cursor:close()
db:close()
env:close()
----------Output
--1 TEXT
--2 TEXT
```

cursor:fetch()

Prototype : cursor:fetch(tab, opt)

Function: Facilitate row nodes (Similar to iterators)

Parameters:

- tab(table)： Row node
- `opt(string)`:

  "n" —— The index of the table is the header of the table, column id

  "a" —— The index of the table is the header of the table, column value

Return:

- row(table) : Get the extracted data

Exmaple:

```
local env = luasql_sqlite3.sqlite3()  -- Get the environment class
local db = env:connect("udisk:test.db") -- Connect a db file
local status = db:execute("CREATE TABLE students('id' TEXT,'name' TEXT)")
status = assert(db:execute([[INSERT INTO students values('1','AA')]]))
status = assert(db:execute([[INSERT INTO students values('2','BB')]]))
status = assert(db:execute([[INSERT INTO students values('3','CC')]]))
```

```lua
local cursor = assert(db:execute([[select * from students]]))
local row = cursor:fetch({},"a")
print(string.format("id = %s    name = %s ",row.id,row.name))     -- Index by table header name
row = cursor:fetch({},"n")
print(string.format("id = %s    name = %s ",row[1], row[2]))     -- Index by numerical number
cursor:close()
db:close()
env:close()
----------Output:
--id = 1 , name = AA
--id = 2 , name = BB
```