V-BOX Script Interface Manual

Website: http://www.we-con.com.cn/en
Technical Support: support@we-con.com.cn
Skype: fcwkkj
Phone: 86-591-87868869
QQ: 1043098682
Technical forum: http://wecon.freeforums.net/

1 Interface Description

1.1 Data type definition

Type	Description
nil	Null
boolean	Boolean, the value is true or false
number	Integer or floating point, signed or unsigned
string	String
table	Table
function	Functions

1.2 Built-in function library clipping

Full features supported: coroutine/debug/ math/ package/ string/ table/ utf8
Some features supported (available in []): os[clock/ date/ difftime/ time]
Not supported: io/ file

1.3 Return value description

The function return type multi means multiple return values (at least 2), usually:
1st: nil;
2nd: the error message;

1.4 Function parameter description

Assume the function prototype:

student(string name, number age[, number class])
Function:
Register a student

Parameters:

name: student name

age: student age

[class=1]: Student class

Return：

Succeed: true

Failed: multi

Comment

string name indicates that the first parameter *name* is a string

number age indicates that the second parameter *age* is numeric

[, number class] indicates that the third parameter *class* is a numeric value, and it is optional. Specify the default class in class 1 in the parameter description.

Any parameter in the brackets of [] is considered to be an optional parameter, and may not be transmitted when called. The default value will be given in the parameter description.

Call example

```
print(student("foo", 18))          -- foo, 18 years old, assigned to class 1 by default
print(student("bar", 19, 2))       --bar, 19 years old, assigned to class 2
print(student("bar", 18))          --bar, 18 years old, assigned to class 1 by default
local stat, err = student("bar", 18) -- Call again, use err to capture error messages
print(stat, err)
```

Output results

```
true
true
nil     student bar registered
nil     student bar registered
```

Comment

1) From the print result, the second line of the first line is successfully called and returns true; the third line fails the call, the error message is translated as: the bar student has been registered, and there is indeed an error in the code.
2) The fourth line of code uses two variables to receive the return value. The call failed, the first variable *stat* is nil, and the second variable *err* stores the error message. Then print it out using print, which is the output of the third line. This example shows how to capture and view the error message.

1.5 Modification of the Print Function

For the convenience of remote development, the print data is sent to the front end (web page) by means of network transmission, and the user can see the result of the debug output, because it consumes certain data and occupies the bandwidth of the server (or occupies server resources). So make the following restrictions:

1) **Interval limit:** When debugging, transfer once every 2~3 seconds;
2) **Data limit:** The transfer data cannot be larger than 1.5KB, and the extra part will be ignored;
3) **Transmission limit:** The data transmission will be stopped automatically after the debugging windows is not closed normally. Only when it is in the debugging window and the switch is on, there is data transmission;

Users should pay attention to avoid printing a lot of useless information, should minimize the debug output

In addition, please refer to the front-end documentation for how to use view debugging.

2 Address Operation

2.1 addr_getshort(string addr)

Function:

Read 16-bit signed decimal address

Parameters:

addr: address

Return：

Succeed: 16-bit signed decimal value

Failed: multi

2.2 addr_setshort(string addr, number num)

Function:

Write 16-bit signed decimal address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.3 addr_getword(string addr)

Function:

Read 16-bit unsigned decimal address

Parameters:

addr: address

Return：

Succeed: 16-bit unsigned decimal value

Failed: multi

2.4 addr_setword(string addr, number num)

Function:

Write 16-bit unsigned decimal address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.5 addr_getint(string addr)

Function:

Read 32-bit signed decimal address

Parameters:

addr: address

Return：

Succeed: 32-bit signed decimal value

Failed: multi

2.6 addr_setint(string addr, number num)

Function:

Write 32-bit signed decimal address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.7 addr_getdword(string addr)

Function:

Read 32-bit unsigned decimal address

Parameters:

addr: address

Return：

Succeed: 32-bit unsigned decimal value

Failed: multi

2.8 addr_setdword(string addr, number num)

Function:

Write 32-bit unsigned decimal address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.9 addr_getbit(string addr)

Function:

Read a bit of the register address

Parameters:

addr: address

Return：

Succeed: Bit address value

Failed: multi

2.10 addr_setbit(string addr, number num)

Function:

Write a bit of the register address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.11 addr_getfloat(string addr)

Function:

Read 32-bit floating address

Parameters:

addr: address

Return：

Succeed: 32-bit floating point value

Failed: multi

2.12 addr_setfloat(string addr, number num)

Function:

Write 32-bit floating address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.13 addr_getdouble(string addr)

Function:

Read 64-bit floating address

Parameters:

addr: address

Return：

Succeed: 64-bit floating point value

Failed: multi

2.14 addr_setdouble(string addr, number num)

Function:

Write 64-bit floating address

Parameters:

addr: address

num: value

Return：

Succeed: true

Failed: multi

2.15 addr_getstring(string addr, number length)

Function:

Read the specified length string from address

Parameters:

addr: address

length: value

Return：

Succeed: specified length string

Failed: multi

2.16 addr_setstring(string addr, string str)

Function:

Write the specified length string to address

Parameters:

addr: address

str: string

Return：

Succeed: true

Failed: multi

2.17 addr_bmov(string dst, string src, number length)

Function:

Copy data from source address to destination address

Parameters:

dst: destination address

src: source address

length: length

Return：

Succeed: true

Failed: multi

2.18 addr_fill(string addr, number num, number length)

Function:

Write the same value to consecutive addresses

Parameters:

addr: address

num: value

length: length

Return：

Succeed: true

Failed: multi

2.19 addr_newnoaddr(string addr, number offset)

Function:

Offset address value relative to *addr*

Parameters:

addr: address

offset: offset value

Return：

Succeed: New address after offset

Failed: multi

2.20 addr_newstataddr(string addr, number offset)

Function:

Offset station number relative to *addr* station number

Parameters:

addr: address

offset: offset value

Return：

Succeed: New station number after offset

Failed: multi

3 Serial port operation

Operations on the serial port such as read, write, etc. must use ':' for full mode calls, ie operations on an open serial object.

Serial port name and mode:

The serial port configured in the communication configuration window cannot be configured again using the script. RS232 and RS458 (or RS422) can be used simultaneously, but RS422 and RS485 are mutually exclusive.

Attempting to use a script to open a serial port in an unsupported mode will result in an error directly, as below

```
local setup = {
    name = "COM2",
    mode = 422, -- COM2 does not support RS422
    ...
}
serial.open(setup)
```

Data bit:

1) When the data bit is 7, the maximum value of data transmission is 127 (0x7F), and non-ASCII characters will be truncated, resulting in data errors and garbled

characters.

2) When the data bit is 8, the maximum value of data transmission is 255 (0xFF), which supports the transmission of any character.

3.1 serial.open(table setup)

Function:

Enable one serial port

Parameters:

Setup is a Lua table; it needs to contain the following fields

String setup.name, serial port name, such as: COM1/COM2 (requires uppercase)

number setup.mode, mode: RS232/RS485/RS422

number setup.baud_rate, such as 115200

number setup.stop_bit, stop bit: 1 or 2

number setup.data_len, data bit: 7 or 8

string setup.check_bit, check bit: NONE/ODD/EVEN/SPACE

number [setup.wait_timeout=300], waiting timeout

number [setup.recv_timeout=50], receive wait timeout

number [setup.flow_control=0], Flow control method, 0:XON/XOFF, 2:DSR/ER

Supported baud rate

1200/2400/4800/9600/14400/19200/38400/43000/57600/76800/115200/128000/230400/256000/460800/961000

Return：

Succeed: serial object

Failed: multi

3.2 serial.close(serial obj)

Function:

Disable the serial port

Parameters:

Obj is the object returned by serial.open

Return：

Succeed: true

Failed: multi

3.3 serial:read(number bytes[, number timeout])

Function:

Read the specified byte length serial port data

Parameters:

bytes: number of bytes

[timeout=50]: timeout for reading, in milliseconds

Return：

Succeed: true

Failed: multi

3.4 serial:write(string data)

Function:

Write the specified byte length to serial port data

Parameters:

data: serial port data

Return：

Succeed: true

Failed: multi

3.5 serial:flush([number flag])

Function:

Clear the serial port buffer

Parameters:

[flag=2] clear option: 0: read, 1: write, 2: read-write

Return：

Succeed: true

Failed: multi

3.6 serial:close()

Function:

Close the serial port object

Parameters:

None

Return：

Succeed: true

Failed: multi

4 MQTT operation

Operations on MQTT such as connect, subscribe, etc. must use ':' for full mode calls, that is, operate on a created MQTT object.

Both MQTT subscriptions and publications are asynchronous implementations that require the user to implement a callback function.

QoS value

0: Only push messages once, messages may be lost or duplicated. It can be used for environmental sensor data, it doesn't matter if lose a record, because there will be a second push message soon. This method is mainly used for normal APP push, but if the user smart device is not connected when the message is pushed, the message will be discarded, and the smart device will not be received when it is networked again.

1: The message is delivered at least once, but the message may be delivered repeatedly.

2: The message was delivered exactly once. This level can be used in a billing system. In a billing system, repeated or missing messages can lead to incorrect results. This highest quality message push service can also be used for instant messaging APP pushes, ensuring that users only receive messages once.

Retain flag:

0: not reserved;

1: reserved

4.1 mqtt.create(string serverurl, string clientid)

Function:

Create an MQTT object

Parameters:

serverurl Server url

Format: "*protocol://host:port*"

protocol: tcp/ssl

host: Host name/IP

port: such as 1883

clientid: Client ID

Return：

Succeed: MQTT object

Failed: multi

4.2 mqtt.close(mqtt obj)

Function:

Close the specified MQTT object (if the connected server will be disconnected automatically)

Parameters:

Obj is the objeced returned by mqtt.create

Return：

Succeed: true

Failed: multi

4.3 mqtt:connect(table conn[, table lwt])

Function:

Establish a connection to the server

Parameters:

conn is a Lua table and needs to contain the following fields

string conn.username, user name

string conn.password, password

number [conn.netway=0], networking method, if set error number will use Ethernet method

- 0: Ethernet

- 1: WIFI
- 2: 4G
- 3: 2G

number [conn.keepalive=60], keep connected heartbeat interval, in seconds

number [conn.cleansession=1], empty the session as described below:

This function is used to control the behavior when connecting and disconnecting, and the client and server will retain the session information. This information is used to guarantee "at least once" and "accurately once" delivery, as well as the subject of the client subscription, the user can choose to keep or ignore the session message, set as follows:

- 1 (Empty): If a session exists and is 1, the previous session messages on the client and server are emptied.
- 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 (Last Will and Testament) is a Lua table and needs to contain the following fields

string lwt.topic, topic

string lwt.message, message

number [lwt.qos=0], qos value

number [lwt.retain=0], retain flag

Return：

Succeed: true

Failed: multi

4.4 mqtt:disconnect([number timeout])

Function:

Disconnect from the MQTT server

Parameters:

[timeout=10000] Disconnect waiting timeout, in milliseconds

Return：

Succeed: true

Failed: multi

4.5 mqtt:isconnected()

Function:

Test whether or not a client is currently connected to the MQTT server

Parameters:

None

Return：

Succeed: true --Connected

Failed: false -- Unconnected and other unknowns

4.6 mqtt:subscribe(string topic, number qos)

Function:

Subscribe to the topic (before the subscription, the user must first call the connect method to connect to the server)

Parameters:

topic, topic name

qos, quality of service

Return：

Succeed: true

Failed: multi

4.7 mqtt:unsubscribe(string topic)

Function:

Unsubscribe topic

Parameters:

topic, topic name

Return：

Succeed: true

Failed: multi

4.8 mqtt:publish(string topic, string message, number qos, number retain[, number timeout])

Function:

Publish message

Parameters:

topic, topic name

message, message

qos, quality of service

retain, retain flag

[timeout=1000], waiting for response timeout, in milliseconds (only valid when qos is greater than 0)

Return：

Succeed: true

Failed: multi

4.9 mqtt:close()

Function:

Close the mqtt object (the connection to the server will be automatically disconnected)

Parameters:

None

Return：

Succeed: true

Failed: multi

4.10 mqtt:on(string method, function callback)

Function:

Register event callback function

Parameters:

method, It can be message/arrived/offline, these 3 types of events

callback, It is a callback function that needs to pass in a function

1) "message" will call this function after receiving the message

   Callback prototype: *function (string topic, string message)*

   Parameter:

   Topic, topic name

   Message, content

2) "arrived" is published by publish, this function will be called after the publication arrives

Callback prototype: *function ()*

Parameter:

None

3) This function will be called after the "offline" connection is lost

Callback prototype: *function (string cause)*

Parameter:

cause, reason for loss of connection

Return：

Succeed: true

Failed: multi

5 JSON operation

Lua only has a table data structure, so all arrays and key-value objects of json will be returned as a table.

5.1 json.encode(lua_object)

Function:

Convert lua data type to json string

Parameters:

Lua data type (including boolean, number, string, table)

Return：

Json format string

5.2 json.decode(string json_string)

Function:

Convert json string to lua data type

Parameters:

json_string, string of json data structure

Return：

Lua data type

5.3 json.null

Function:

This method is used when assembling json data, which is equivalent to null in json. If the user directly uses json.null() to return the address of the function, it must be valid with the use of encode.

Parameters:

None

Return：

None