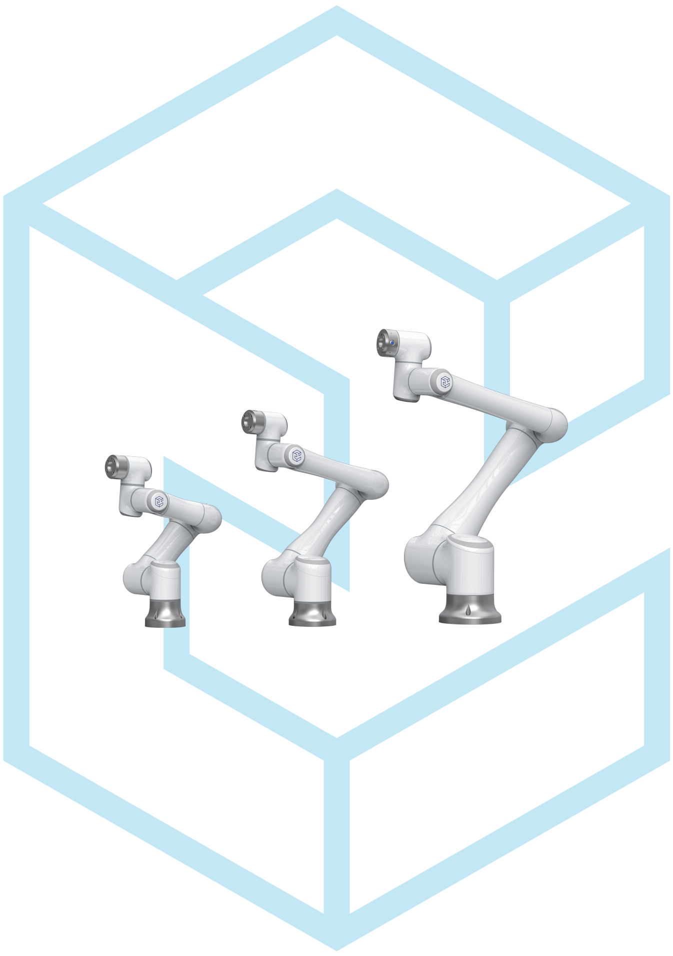
ELITE ROBOTS
艾利特机器人

ELITE ROBOTS EC Series Programming Manual



Lua Script Manual

Suzhou Elite Robot Co., Ltd

2022-10-10

Version: Ver3.6.2

www.elibot.cn

Contents

Chapter 1 Language Definition

1.1 Identifiers and Variables

Identifiers can be composed of any letter, underscore and digits that do not start with a number. It can be used to name variables, functions, etc. The following keywords are reserved and cannot be used for identifier naming :
and not or break return do then if else elseif end true false for while repeat until in function goto local
Example

1. Valid identifier name: a _abc a_123
2. Valid digital constants: 1 123 0xff 0xABCD
3. Valid float constant: 1.0 3.141592 1.6e-2 100e1
4. String variables are represented by "", for example: "abcdef"
5. Array variables are represented by{ }, for example: a={1.0,0.2,3.0}

Global variables and local variables have different scopes. Add local keyword before declaring local variables, for example, local a=5, otherwise it is a global variable.

The expression syntax in EliteScript is very standard, as follows:

1. Arithmetic expression
   6+2-3
   5*2/3
   (2+3)*4/(5-6)
2. Logical expression
   true or false and (2==3) and (1⟩2)
   or (3~=4) or (5⟨-6) not (9⟩=10)
   and 100⟨=50
3. Variable assignment
   A = 100
   bar = true
   PI = 3.1415
   name = "Lily"
   positon = {0.1,-1.0,0.2,1.0,0.4,0.5}

Variable types in EliteScript do not need to be declared before, but are derived from the first assignment of the variable. For example, in the above example, A is an integer, bar is a Boolean, PI is a float number, name is a string, position is an array.

The basic variable types in EliteScript are: number, Boolean, string, array.

1.2 Control Flow Statement

The flow control of programs can be implemented through if, while, repeat, and for structures. In EliteScript , their syntax rules all conform to the common definition and the syntax is:

1.2.1 Branch Statement

if(exp1) then

—— [execute the statement block when exp1 is true]

else if(exp2) then

—— [execute the statement block when exp2 is true]

else

—— [execute the statement block when other conditions are met]

end

For example:

```
a= -2
if(a<0) then
          print("a<0")
elseif(a>0) then
         print("a>0") else
         print("a=0");
end
```

1.2.2 while Loop

while(exp) do

—— [[exp loop the statement block when the expression is true]]

end

For example:

```
while(true) do
        print("A");
end
```

1.2.3 repeat Loop

repeat

—— [loop the statement block when the expression is false]

until(exp)

Example: Print the following program: 10 11 12 13 14 15

```
A = 10
repeat
        print(A)
        A = A+1
until(A>15)
```

1.2.4 for Loop

for init, max/min value, increment do

—— [execute statement block]

end

Example: print the following program: 10 9 8 7 6 5 4 3 2 1

```
for i=10,1,-1 do
        print(i)
end
```

Use break to stop a loop , use goto statement to jump, and return to return directly.

Note in particular that EliteScript does not support continue statement, but it can be implemented indirectly with goto .

1.3 Operator

Mathematical Operator	Meaning
+	Addition
*	Multiplication
–	Subtraction
/	Floating Point Division
//	Division Down
%	Modulo
^	Power
–	Negative

Bit operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	Bitwise XOR
⟩⟩	Right Shift
⟨⟨	Left Shift
~	Bitwise

Non-comparison Operators	Meaning
==	Equal to
~=	Not Equal to
⟨	Less than
⟩	Greater than
⟨=	Less than or Equal to
⟩=	Greater than or Equal to

Logical Operators	Meaning
and	and
or	or
not	not

String Connector: write two dots .., such as 12..34, equivalent 1234.

Length Operator: #, and the length of the string is the number of bytes.

Priority Definition (from low to high):

or

and

⟨, ⟩, ⟨=, ⟩=, ~=, ==

|

~

&

⟨⟨, ⟩⟩

..

+,-

*, /, //, %

#, −(unary), not

^

Use parentheses to change the order of operations. The append operator .. and the power operation ^ are right to left. All other operations are left to right.

1.4 Function

1.4.1 Function Definition Syntax:

The syntax to define a function is as follows:

```
function MyFunc(param)
--Do something
end
```

1.4.2 Custom Function

Customize Addition Function:

function add(a,b)

return (a+b)

end

Function Call:

x= add(3,4)

Functions can have no return value or one or more return values.

For example :

```
function add(a,b)
        return a,b,(a+b)
end
```

Function Call:

x,y,z=add(a,b)

1.4.3 Split String

string.sub(strccd, start position, end position)
—— acquire the length of string from a specified position

string.len(target string)
—— call the function to acquire the length of the string

function string.split(str, delimiter)
—— split string with delimiter and return array.

str: string to detach

```
function string.split(str, delimiter)
        if str==nil or str=='' or delimiter==nil then
                return nil
        end
        local result = {}
        for match in (str..delimiter):gmatch("(.-)"..delimiter) do
                table.insert(result, match)
        end
```

```
                return result
end
```

string:split is intrinsic for string separation

1.4.4 String

string.len(s):

return the length of string s

string.lower(s):

convert uppercase letters in s to lowercase

string.upper(s):

convert lowercase letters in s to uppercase

string.sub(s,i,j):

intercepts the string out of string s from i to j

string.char():

convert character to number

string.byte():

convert number to character

1.4.4.1 String formatting

The string.format() function is used to generate a string with a specific format. The first parameter of the function is the format, followed by various data corresponding to each code in the format.

The following example demonstrates how to format a string:

The format string may contain the following escape codes:

%c - accept a number, and convert it into the corresponding character in the ASCII code table
string.format("%c", 83) —— output S

%d, %i - accept a number and convert it into signed integer format
string.format("%+d", 17.0) —— output +17

%o - Accept a number and convert it to octal number format
string.format("%o", 17) —— output 21

%u - accept a number and convert it to unsigned integer forma string.format(“%u”, 3.14) —— output 3

%x - accept a number and convert it to hexadecimal number format with lowercase letters

string.format(“%x”, 13) —— output d

%X - accept a number and convert it to hexadecimal number format with uppercase letters

string.format(“%X”, 13) —— output D

%e - accept a number and convert it to   notation format with lowercase letter e

string.format(“%e”, 1000) —— output 1.000000e+03

%E - accept a number and convert it to   notation format, with uppercase letters E.

string.format(“%E”, 1000) —— output 1.000000E+03

%f - accept a number and convert it to   point format

string.format(“%6.3f”, 13) —— output 13.000

%g(%G) - accept a number and convert it to the shorter format of% e (% E, corresponds to% G) and % f

%q - accept a string and convert it to a format that can be read safely by Lua compiler

string.format(“%q”, “One\nTwo”) —— output “One\Two”

%s - accept a string and format it with the given parameters

string.format(“%s”, “monkey”) —— output monkey

string.format(“%10s”, “monkey”) —— output   monkey

1.5 Http Client Support

```lua
local http = require("socket.http")
local ltn12 = require("ltn12")
local request_body = [[login=user&password=123]]
local response_body = {}
local res, code, response_headers = http.request{
    url = "http://getman.cn/echo",
    method = "GET",
    headers ={
        ["Content-Type"] = "application/x-www-form-urlencoded";
        ["Content-Length"] = #request_body;
    },
    source = ltn12.source.string(request_body),
    sink = ltn12.sink.table(response_body),
}

print(res)
print(code)

if type(response_headers) == "table" then
    for k, v in pairs(response_headers) do
        print(k, v)
    end
end

print("Response body:")
if type(response_body) == "table" then
    print(table.concat(response_body))
else
    print("Not a table:", type(response_body))
end
```

Chapter 2 list of internal methods

2.1 delay

```
sleep(second)
```

Function:

sleep(second) is used for sleep waiting

Parameter:

second:

Waiting time, double type, unit: seconds

Return value:

nil

Note:

none

Example:

```
sleep(0.1)
```

2.2 Set the IO status of the robot arm body

```
set_robot_io_status(name,value)
```

Function:

set_robot_io_status(name,value) is used to set the IO status of the robot body, and the status of Y, M, and AO variables can be set.

Parameter:

name:

IO name, string type

value:

IO state value, int type (corresponding to Y, M) or double type (corresponding to AO)

Return value:

nil

Note:

When setting the value of the Y variable, the parameter name is "o**", for example "o0" means "Y0"
When setting the analog output value, the parameter name is "a**", for example, "a0" means "AO001", "a1" means "AO002", and so on.
When setting the virtual output value, the parameter name is "M**", for example, "M528" means "VOUT528".
This command is applicable to v2.9.3 and above.

Example:

```
set_robot_io_status("M528",1)
```

2.3 Get the IO status of the robot arm body

```
ret get_robot_io_status(name)
```

Function:

get_robot_io_status(name) is used to obtain the IO status of the robot body, and the status of X, Y, M, and AI variables can be obtained

Parameter:

name:
IO name, string type

Return value:

ret:
corresponds to the status value of IO

Note:

When getting the state of the X variable, the parameter name is ”i**”, for example, ”i0” means ”X0”
When obtaining the analog input value, the parameter name is ”a**”, for example, ”a0” means ”AI001”, ”a1” means ”AI002”, and so on.
This command is applicable to v2.9.3 and above.

Example:

```
ret=get_robot_io_status("M20")
```

2.4 Get teach pendant global variable value

```
ret get_global_variable(varName)
```

Function:

get_global_variable(varName) is used to obtain the global variable value of the teach pendant, and supports to obtain B, I, D, P, V variables

Parameter:

varName:

variable name, string type

Return value:

ret:

The variable value corresponding to the variable name, the return value type depends on the type of the variable

Note:

When getting the I variable, the range is 0~8255.

Example:

```
--Get B, I, D variables, such as getting B variable
ret=get_global_variable("B20")
elite_print(ret)
--Get B, I, D variables, such as getting P variable
a1,a2,a3,a4,a5,a6=get_global_variable("P20")
elite_print(a1,a2,a3,a4,a5,a6)
```

2.5 Set teach pendant global variable value

```
set_global_variable(varName,varValue)
```

Function:

set_global_variable(varName,varValue) is used to set the global variable value of the teach pendant, and supports to obtain B, I, D, P, V variables

Parameter:

varName:

variable name, string type

varValue:

variable value, variant type

Return value:

nil

Note:

Set the I variable, the range is 0~8255.

Example:

```
--Set B, I, D variables, such as setting B variable
set_global_variable ("B20", 1)
--Set P, V variables, such as setting P variable
set_global_variable("P20",90,0,-90,80,30,60)
```

2.6 Set whether to debug mode

```
set_debug(debug)
```

Function:

set_debug(debug) is used to set whether to debug mode

Parameter:

debug:

int type

0: do not output INFO information

1: output INFO information

Return value:

nil

Note:

When this function is 1, the debugging information of the internal module is output, and the function of elite_print is not affected

Example:

```
set_debug(1)
```

2.7 Print information to the teach pendant interface

```
elite_print(var1,var2....)
```

Function:

elite_print(var1,var2....) is used to print the specified information to the information prompt window in the teaching box

Parameter:

var1:

string type

Return value:

nil

Note:

none

Example:

```
int num=10
elite_print("test",tostring(10))
```

2.8 Inverse solution function

```
ret get_inv_kinematics(var1,var2)
```

Function:

get_inv_kinematics(var1,var2) is used for inverse solution function

Parameter:

var1:

pose (target pose), table type

var2:

joint (reference point joint angle, the reference point needs to be close to the target point. If not written, it is regarded as the reference current point), table type

Return value:

ret:

inverse solution result is empty/

table:joint (target point joint angle)

Note:

pose={x,y,z,rx,ry,rz}, an array of size 6 (note: rx, ry, rz in pose are all radians, the same below)

joint={j1,j2,j3,j4,j5,j6}Array of size 6

Example:

```
pose={378.538,212.504,134.055,-2.712,-0.791,2.553}
joint={10.081,-75.007,105.449,-70.694,98.434,89.481}
ret=get_inv_kinematics(pose,joint)
```

2.9 Positive solution function

```
ret get_fwd_kinematics(var1)
```

Function:

get_fwd_kinematics(var1) for positive solution function

Parameter:

var1:

joint (target point joint angle), table type

Return value:

ret:

Positive solution is empty/

table:pose (target point pose)

Note:

pose={x,y,z,rx,ry,rz}Array of size 6

joint={j1,j2,j3,j4,j5,j6}Array of size 6

Example:

```
joint={10.081,-75.007,105.449,-70.694,98.434,89.481}

ret=get_fwd_kinematics(joint)
```

2.10 inverse of pose

```
ret pose_inv(var1)
```

Function:

pose_inv(var1) is used to return the inverse of the p pose

Parameter:

var1:

pose data, table type

Return value:

ret:

result is empty/

table:pose

Note:

pose={x,y,z,rx,ry,rz}An array of size 6

Example:

```
ret=pose_inv({0,1,2,3,4,5})
```

2.11 Multiplication of pose

```
ret pose_mul(var1,var2)
```

Function:

pose_mul(var1,var2) is used to return the multiplication of the pose

Parameter:

var1:

pose data, table type

var2:

pose data, table type

Return value:

ret:

result is empty/

table:pose

Note:

pose={x,y,z,rx,ry,rz}An array of size 6

Example:

```
ret=pose_mul({0,1,2,3,4,5},{0,1,2,3,4,5})
```

2.12 Get robot mode

```
ret get_robot_mode()
```

Function:

get_robot_mode() is used to get the robot mode

Parameter:

none

Return value:

ret:

0: Teaching mode

1: Auto mode

2: remote mode

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=get_robot_mode()
```

2.13 Get robot JBI operating mode

```
ret get_cycle_mode()
```

Function:

get_cycle_mode() is used to get the JBI running mode of the robot

Parameter:

none

Return value:

ret:

0: single step

1: single loop

2: continuous loop

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=get_cycle_mode()
```

2.14 Get robot servo enable status

```
ret get_servo_status()
```

Function:

get_servo_status() is used to get the robot servo enable status

Parameter:

none

Return value:

ret:

0: Servo enable off

1: Servo enable on

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=get_servo_status()
```

2.15 Get the robot running status

```
ret get_robot_state()
```

Function:

get_robot_state() is used to get the running state of the robot

Parameter:

none

Return value:

ret:

0: stop
1: pause
3: Running
4: Alarm
5: collision

Note:

none

Example:

```
ret=get_robot_state()
```

2.16 Get the robot coordinate system

```
ret get_current_coord()
```

Function:

get_current_coord() is used to get the robot coordinate system

Parameter:

none

Return value:

ret:

0: Joint coordinate system
1: base coordinate system
2: Tool coordinate system
3: User coordinate system
4: cylindrical coordinate system

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=get_current_coord()
```

2.17 Get robot pose

```
ret get_robot_pose()
```

Function:

get_robot_pose() is used to get the robot pose

Parameter:

none

Return value:

ret:

Robot current pose [x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.9.3 and above.
This command is not recommended.

Example:

```
ret=get_robot_pose()
```

2.18 Get the robot joint angle

```
ret get_robot_joint()
```

Function:

get_robot_joint() is used to get the robot joint angle (returned in angle)

Parameter:

none

Return value:

ret:

The current joint angle of the robot [joint1,joint2,joint3,joint4,joint5,joint6]

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=get_robot_joint()
```

2.19 Get robot moment information

```
ret get_robot_torque()
```

Function:

get_robot_torque() is used to obtain robot torque information

Parameter:

none

Return value:

ret:

The torque of each joint of the robot's current pose [torque1,torque2,torque3,torque4,torque5,torque6]

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=get_robot_torque()
```

2.20 Get the "external" force and torque in the current TCP coordinate system

```
ret get_tcp_force(ref_tcp)
```

Function:

get_tcp_force (ref_tcp) is used to get the "external" force and torque of the current TCP or base coordinate system

Parameter:

ref_tcp:

reference coordinate system, int[0,1], optional parameter, 0: base coordinate system, 1: tcp coordinate system, if not written, the default is tcp coordinate system.

Return value:

ret:

when the robot is in a singular position, it returns null, otherwise, it returns to the "external" force and torque of the robot's current TCP or base coordinate system

Note:

This command is applicable to v3.4.2 and above.

Example:

```
ret=get_tcp_force()
```

2.21 Get the current joint torque

```
ret get_joint_torques()
```

Function:

get_joint_torques() is used to get the current joint torque

Parameter:

none

Return value:

ret:

the current joint torque of the robot [J1_torq,J2_torq,J3_torq,J4_torq, J5_torq,J6_torq]

Note:

The joint torque is the motor torque minus the "torque required to drive itself", reflecting the "external" torque. This command is applicable to v3.4.2 and above.

Example:

```
ret=get_joint_torques()
```

2.22 Get the actual tcp pose

```
ret get_actual_tcp(int tool_num,int user_num)
```

Function:

get_actual_tcp(int tool_num, int user_num) is used to get the actual tcp pose data in the base coordinate system or the specified user coordinate system

Parameter:

tool_num:

user coordinate number, optional parameter, int[-1,7], if tool_num is set to -1, the current tool is used. Otherwise, the specified tool is used

user_num:

user coordinate number, optional parameter, int[-1,7], if user_num is set to -1, the user gets the pose in the base coordinate system. Otherwise, the user gets the pose in the specified user coordinate system

Return value:

ret:

table pose data

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_actual_tcp(0,0)
```

2.23 Get the target interpolation tcp pose

```
ret get_target_tcp(int tool_num,int user_num)
```

Function:

get_target_tcp(int tool_num, int user_num) is used to get the target interpolation tcp pose data in the base coordinate system or the specified user coordinate system

Parameter:

tool_num:

user coordinate number, optional parameter, int[-1,7], if tool_num is set to -1, the current tool is used. Otherwise, the specified tool is used

user_num:

user coordinate number, optional parameter, int[-1,7], if user_num is set to -1, the user gets the pose in the base coordinate system. Otherwise, the user gets the pose in the specified user coordinate system

Return value:

nil

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_target_tcp(0,0)
```

2.24 Get the linear interpolation pose

```
ret get_interp_pose(table var1,table var2,double ratio)
```

Function:

get_interp_pose(table var1, table var2, double ratio) is used to get the linear interpolation pose data between two given poses

Parameter:

var1:

pose data, double pose[6], the first three stand for position, unit x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π, π]

var2:

pose data, double pose[6], the first three stand for position, unit x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit Rx, Ry, Rz is radian, range is [-π, π]

ratio:

floating-point data, it stands for the proportional value. The range is [0,1]. When the value is equal to 0, the robot will return to the first pose. When the value is equal to 1, the robot will return to the second pose.

Return value:

ret:

table pose data

Example:

```
pose1={371.534, 101.636, 3.038, 0, -0.174, 2.861}
pose2={346.312, -256.945, -91.131, -0.142, 0.521, 1.903}
interp_pose=get_interp_pose(pose1,pose2,0.1)
```

2.25 Get the actual joint

```
ret get_actual_joint()
```

Function:

get_actual_joint() is used to get the actual joint data

Parameter:

none

Return value:

ret:

table joint data

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_actual_joint()
```

2.26 Get the target interpolation joint

```
ret get_target_joint()
```

Function:

get_target_joint() is used to get the target interpolation joint data

Parameter:

none

Return value:

ret:

table joint data

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_target_joint()
```

2.27 Get tool coordinate system pose

```
ret get_tool_frame(num)
```

Function:

get_tool_frame(num) is used to get the data in the toolframe

Parameter:

num:

Tool coordinate number, int[0,7], optional parameter, if num is not set, then get the current tool coordinate system pose.

Return value:

ret:

tool[x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.9. and above.

Example:

```
ret=get_tool_frame(0)
```

2.28 Get user coordinate system pose

```
ret get_user_frame(num)
```

Function:

get_user_frame(num) is used to get the data in userframe

Parameter:

num:

User coordinate number, int[0,7], optional parameter, if num is not set, then get the current user coordinate system pose.

Return value:

ret:

user[x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.9.4 and above.

Example:

```
ret=get_user_frame(0)
```

2.29 Get tool coordinate system number

```
ret get_tool_no()
```

Function:

get_tool_no() is used to get the current tool coordinate system number

Parameter:

none

Return value:

ret:

Current robot tool coordinate system number

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ret=get_tool_no()
```

2.30 Get user coordinate system number

```
ret get_user_no()
```

Function:

get_user_no() is used to get the current user coordinate system number

Parameter:

none

Return value:

ret:

Current robot user coordinate system number

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ret=get_user_no()
```

2.31 Check tool enable status

```
ret check_tool_frame_enable(num)
```

Function:

check_tool_frame_enable(num) is used to check whether the specified tool is enabled

Parameter:

num:

Tool coordinate number, int[0,7]

Return value:

ret:

0: The current tool number is not enabled

1: The current tool number is enabled

Note:

This command is applicable to v2.9.4 and above.

Example:

```
ret=check_tool_frame_enable(0)
```

2.32 Check User Coordinates Enable Status

```
ret check_user_frame_enable(num)
```

Function:

check_user_frame_enable(num) is used to check whether the specified user coordinates are enabled

Parameter:

num:

User coordinate number, int[0,7]

Return value:

ret:

0: The current user coordinate number is not enabled

1: The current user coordinate number is enabled

Note:

This command is applicable to v2.9.4 and above.

Example:

```
ret=check_user_frame_enable(0)
```

2.33 Get M variable register interface (0-191)

```
recv, ret get_robot_register(index)
```

Function:

get_robot_register(index) is used to get the register interface of the M variable

Parameter:

index:

byte definition, int[0,191]

Return value:

recv:

register interface value

ret:

0: success

-1: failed

Note:

This command is applicable to v2.16.2 and above.

Example:

```
recv, ret=get_robot_register(77)
```

2.34 Set M variable register interface (66-191)

```
set_robot_register(index,size,string)
```

Function:

set_robot_register(index, size, string) eight-bit register interface for setting M variables

Parameter:

index:

register address, int type[66,191]

size:

The unit is byte, int type[1,128], and the sum with the index value must be less than or equal to 192

string:

hex string

Return value:

nil

Note:

This command is applicable to v2.16.2 and above.

Example:

```
set_robot_register(66,1,"01")--Execution result, M528-M535 data is 10000000 in sequence
```

2.35 Get M variable register interface (192-575)

```
recv, ret get_robot_extra_register(index)
```

Function:

get_robot_extra_register(index) is used to get the register interface of the M variable

Parameter:

index:

byte definition, int type[192,575]

Return value:

recv:

register interface value

ret:

0: success

-1: failed

Note:

This command is applicable to v2.16.2 and above.

Example:

```
recv,ret=get_robot_extra_register(192)
```

2.36 Set M variable register interface (300-447)

```
set_robot_extra_register(index,size,string)
```

Function:

set_robot_extra_register(index, size, string) hexadecimal register interface for setting M variables

Parameter:

index:

register address, int type[300,447]

size:

int type, the unit is bytes, and the sum with the index value must be less than or equal to 448

string:

hex string

Return value:

nil

Note:

The registers occupied by the system cannot be modified. Please refer to the com-munication protocol manual for detailed interfaces.
This command is applicable to v2.16.2 and above.

Example:

set_robot_extra_register(300,2,"0F0E"). At this time, get_robot_extra_register(300), the return value is "0E0F" (the register storage mode is little endian mode, the high data bit corresponds to the high 8 bits of the address, and the low data bit corresponds to the low 8 bits of the address).

2.37 Get current tcp speed

```
ret get_current_tcp_spd()
```

Function:

get_current_tcp_spd() is used to get the current tcp speed

Parameter:

none

Return value:

ret:

Current tcp speed, unit: mm/s

Note:

This command is applicable to v2.16.2 and above.

Example:

```
ret=get_current_tcp_spd()
```

2.38 Get the pose of the flange center in the current base coordinate system

```
ret get_flange_pose_inbase()
```

Function:

get_flange_pose_inbase() is used to obtain the pose data of the flange center in the current base coordinate system

Parameter:

none

Return value:

ret:

The current pose of the robot [x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ret=get_flange_pose_inbase()
```

2.39 Get the pose of the flange center in the current user coordinate system

```
ret get_flange_pose_inuser()
```

Function:

get_flange_pose_inuser() is used to obtain the pose of the flange center in the current user coordinate system

Parameter:

none

Return value:

ret:

Robot's current user coordinate system pose [x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ret=get_flange_pose_inuser()
```

2.40 Get robot pose

```
ret get_tcp_pose()
```

Function:

get_tcp_pose() is used to get the robot pose

Parameter:

none

Return value:

ret:

The current pose of the robot [x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ret=get_tcp_pose()
```

2.41 Get the pose of the current tcp in the current user coordinate system

```
ret get_tcp_pose_inuser()
```

Function:

get_tcp_pose_inuser() is used to obtain the pose of the current tcp in the current user coordinate system

Parameter:

none

Return value:

ret:

The robot's current tcp pose in the current user coordinate system [x,y,z,rx,ry,rz]

Note: rx, ry, rz are radians

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ret=get_tcp_pose_inuser()
```

2.42 Get end 485 mode

```
ret get_terminal_485_mode()
```

Function:

get_terminal_485_mode() is used to get the terminal 485 mode

Parameter:

none

Return value:

ret:

0: initial mode

1: tci communication mode

2: modbusRTU master mode

Note:

This command is applicable to v2.17.0 and above.

Example:

```
ctx=modbus_new_rtu(2,9600,78,8,1)
sleep(2)
ret=get_terminal_485_mode()
elite_print(ret)
```

2.43 Get the distance between two points

```
ret get_point_distance(table var1, table var2)
```

Function:

get_point_distance(table var1, table var2) is used to get the distance between two giving points in the Cartesian space

Parameter:

var1: pose data, double pose [6], the first three stand for position, unit of x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit of Rx, Ry, Rz is radian, range is [-π, π]
var2: pose data, double pose [6], the first three stand for position, unit of x, y, z is mm, range is [-∞, +∞], the last three stand for pose, unit of Rx, Ry, Rz is radian, range is [-π, π]

Return value:

ret:

distance between two points, floating-point type, unit: mm

Note:

none

Example:

```
table1={499,19,423,-3.14,0,-1.57}
table2={500,122,454,-3.14,0,-1.57}
dist=get_point_distance(table1,table2)
elite_print("dist", tostring(dist))
```

2.44 Get the joint temperature

```
ret get_joint_temp()
```

Function：

get_joint_temp() is used to get the joint temperature

Parameter:

none

Return value:

ret:

double joint_temp[6]

Example:

```
ret=get_joint_temp()
elite_print(ret[1],ret[2],ret[3],ret[4],ret[5],ret[6])
```

Chapter 3 TCP communication

3.1 TCP server

3.1.1 Initialize TCP server

```
init_tcp_server(port)
```

Function:

init_tcp_server(port) is used to initialize the TCP server

Parameter:

port:

port number, int type

Return value:

nil

Note:

The port numbers (22, 80, 6680, 8055, 8056, 8058, 8059, 502, 1502 and 5900) have been occupied by the controller. Please do not use them.

Example:

```
init_tcp_server(8888)
```

3.1.2 Determine whether the client is connected to the server

```
ret is_client_connected(IP, port)
```

Function:

is_client_connected(IP,port) is used to determine whether the client is connected to the server.

Parameter:

IP:

client IP address, string type

port:

TCP server port number, int type

Return value:

ret:

1: The IP address in the parameter is connected to the tcp server

-1: The IP address in the parameter is not connected to the tcp server

Note:

The port parameter is only applicable to v2.15.2 and above.

Example:

```
ret=is_client_connected("192.168.1.100",8888)
```

3.1.3 Receive client data

```
ret, recv server_recv_data (IP, hex, port, recv_timeout)
```

Function:

server_recv_data(IP,hex,port,recv_timeout) is used to receive client data.

Parameter:

IP:

client IP address, string type

hex:

Optional parameter, the format of the data sent from the specified server to the client， 0: string, 1: hexadecimal, default is 0, int type

port:

TCP server port number, int type

recv_timeout:

optional parameter, double type, timeout, unit: second

Return value:

ret:

amount of data received

-1: Receive error

recv:

Received data

Note:

The port parameter is only applicable to v2.15.2 and above.

Example:

```
ret,recv=server_recv_data("192.168.1.100",0,888,0.5)
```

3.1.4 Send data to client

```
ret server_send_data(IP, msg, hex, port)
```

Function:

server_send_data(IP, msg, hex, port) is used to send data to client.

Parameter:

IP:

client IP address, string type

msg:

Sent message, string type

hex:

Optional parameter, int type, whether it is a hexadecimal number, the sent data of 1 is a hexadecimal number (default is 0)

port:

TCP server port number, int type

Return value:

ret:

Amount of data sent

-1: failed to send

Note:

The port parameter is only applicable to v2.15.2 and above.

Example:

```
ret=server_send_data("192.168.1.100","msg",0,8888)
```

3.1.5 Example

Example:

```
port = 6666
ip ="192.168.1.100"
init_tcp_server(port)
sleep(5)
while(1)do
        ret=is_client_connected(ip)
        if(ret==1)then
                server_send_data(ip,"1")
                recv="1"
                while(recv ~= "2") do
                        sleep(1)
                        Ret,recv=server_recv_data(ip,0,888,0 .5

                        print(recv,Ret)
                end
        end
end
```

3.2 TCP Client

3.2.1 connect to server

```
ret connect_tcp_server(IP, port)
```

Function:

connect_tcp_server(IP, port) is used to connect to the TCP server with the specified IP and port

Parameter:

IP:

Server IP address, string type

port:

port number, int type

Return value:

ret:

1: successful connection

0: not connected

Note:

none

Example:

```
ret=connect_tcp_server("192.168.1.100",7777)
```

3.2.2 Receive server data

```
ret, recv client_recv_data(IP, recv_timeout, hex, port)
```

Function:

client_recv_data(IP,recv_timeout,hex,port) is used by the client to receive the data sent by the server with the specified IP and port number

Parameter:

IP:

Server ip address, string type

recv_time:

Receive timeout, optional parameter (if this parameter is not filled, set the global timeout via client_set_recv_timeout, otherwise the default timeout is 4s), double type, in seconds

hex:

The format of the data sent from the specified server to the client， 0: string, 1: hexadecimal, optional parameter (default is 0), int type

port:

Port number of the server (receive data on different port numbers by specifying this parameter), optional parameter, int type, port number

Return value:

ret:

returns the number of recvs

-1: Accept failure requires reconnection

recv:

data sent from server

Note:

1: When the recv_time parameter is defaulted, the subsequent optional parameters must be defaulted.

2: When using multiple set global timeout commands and specifying the timeout time by receiving server data, if the default timeout time is used for receiving server data after this, the timeout time is the last specified timeout time at this time.
The port parameter is only applicable to v2.15.2 and above.
This command is applicable to v2.15.2 and above.

Example:

```
ret,recv=client_recv_data("192.168.1.100",0.1,0,7777)
Note: The recv_timeout parameter can not be written, but use
    the following function to set the timeout time.
```

3.2.3 Overall setting timeout

```
ret client_set_recv_timeout(IP, recv_timeout)
```

Function:

client_set_recv_timeout(IP,recv_timeout) is used to set the recv_timeout time in combination with the recv above

Parameter:

IP:

IP address, string type

recv_timeout:

Timeout time, double type, in seconds

Return value:

ret:

1: set successfully

-1: Setup failed

Note:

none

Example:

```
ret=client_set_recv_timeout("192.168.1.100",0.1)
```

3.2.4 Flush client buffer

```
client_flush(string IP, int port)
```

Function:

client_flush (sting IP, int port) is used to flush the buffer of specified TCP client

Parameter:

IP:

Server IP address, string type

port:

Server port number (the client is specified by the connected port number), optional parameter, int type

Return value:

nil

Note:

none

Example:

```
client_flush("192.168.1.100",7777)
```

3.2.5 Send data to server

```
ret client_send_data(IP, msg, hex, port)
```

Function:

client_send_data(IP, msg, hex, port) is used by the client to send data msg to the server with the specified IP and port number

Parameter:

IP:

IP address, string type

msg:

data sent to server, string type

hex:

Whether is a hexadecimal number, 1 sends data in hexadecimal character format (default is 0), int type

port:

optional parameter, port number, int type

Return value:

ret:

number of send

-1: Send failed and need to reconnect

Note:

The port parameter is only applicable to v2.15.2 and above.
This command is applicable to v2.9.4 and above.

Example:

```
ret=client_send_data("127.0.0.1","OK",0,7777)
```

3.2.6 Disconnect TCP connection

```
disconnect_tcp_server(IP,port)
```

Function:

disconnect_tcp_server(IP, port) is used to disconnect the client from the server

Parameter:

IP:

IP address, string type

port:

optional parameter, port number (if not written, all TCP connections will be disconnected by default), int type

Return value:

nil

Note:

The port parameter is only applicable to v2.15.2 and above.

Example:

```
disconnect_tcp_server("192.168.1.200",7777)
```

3.2.7 Example

Example:

```
port = 7777
ip = "192.168.1.100"
connect_tcp_server(ip,port)
sleep(1)
client_send_data(ip,"OK")
recv="1"
while(recv ~= "2") do
        sleep(1)
        Ret,recv=client_recv_data(ip,2)
        elite_print(Ret,recv)
end
disconnect_tcp_server(ip)
```

Chapter 4 UDP communication

The commands in this chapter are applicable to v2.16 and above.

4.1 UDP server

4.1.1 Initialize UDP server

```
ret init_udp_server(port)
```

Function:

init_udp_server(port) is used to initialize the UDP server

Parameter:

port:

port number, int type

Return value:

ret:

1: success

-1: failed

Note:

none

Example:

```
ret=init_udp_server(8888)
```

4.1.2 Receive UDP client data

```
ret, recvbuff, client_ip, client_port udp_server_recv_data(port, timeout
    , hex)
```

Function:

udp_server_recv_data(port,timeout,hex) is used to receive data from the client with the specified IP and port number

Parameter:

port:

server port number, int type

timeout:

Timeout time, unit: s, double type

hex:

Optional parameter, the format of the data sent from the specified server to the client, 0: string, 1: hexadecimal, default is 0, int type

Return value:

ret:

greater than or equal to 0: success

-1: failed

recvbuff:

Received data

client_ip:

source client IP address

client_port:

source client port number

Note:

none

Example:

```
ret,recvbuff,client_ip,client_port=udp_server_recv_data
    (777,0.1,0)
```

4.1.3 Send data to UDP client

```
ret udp_server_send_data(server_port,msg,client_port,client_ip,hex)
```

Function:

udp_server_send_data(server_port,msg,client_port,client_ip,hex) is used by the server to send data to the client with the specified port number and IP address

Parameter:

server_port:

server port number, int type

msg:

data sent to client, string type

client_port:

destination client port number, int type

client_ip:

target client IP, string type

hex:

Optional parameter, whether it is a hexadecimal number, the sent data of 1 is a hexadecimal number (default is 0), int type

Return value:

ret:

greater than or equal to 0: success

-1: failed to send

Note:

none

Example:

```
ret=udp_server_send_data(777,"test",10000,"192.168.1.100")
```

4.1.4 close udp server

```
ret close_udp_server(port)
```

Function:

close_udp_server(port) is used to close the udp server

Parameter:

port:

optional parameter, server port number, leave all udp servers closed, int type

Return value:

ret:

greater than or equal to 0: success

is less than: send failed

Note:

none

Example:

```
ret=close_udp_server(777)
```

4.2 UDP Client

4.2.1 Connect to UDP server

```
ret connect_udp_server(port, IP)
```

Function:

connect_udp_server(port, IP) is used by the client to connect to the server with the specified port number and IP address

Parameter:

port:

mark server port, int type

IP:

target server IP address, string type

Return value:

ret:

greater than or equal to 0: success
is less than 0: not connected

Note:

None

Example:

```
ret=connect_udp_server(777,"192.168.1.100")
```

4.2.2 Receive UDP server data

```
ret, recv udp_client_recv_data(port, IP, timeout, hex)
```

Function:

udp_client_recv_data(port, IP, timeout, hex) is used to receive data from the server with the specified port number and IP address

Parameter:

port:

target server port, int type

IP:

target server IP address, string type

recv_timeout:

Timeout time in seconds, double type

hex:

Optional parameter, the format of the data sent from the specified server to the client， 0: string, 1: hexadecimal, default is 0, int type

Return value:

ret:

greater than or equal to 0：success

is less than 0：failed

recv:

Received data

Note:

UDP adopts a connectionless mode, so the data must be sent first for the client, and the server responds with the data.

Example:

```
ret,recv=udp_client_recv_data(777,"192.168.1.6",0.1)
```

4.2.3 Send data to UDP server

```
ret udp_client_send_data(port, IP, msg, hex)
```

Function:

udp_client_send_data(port,IP,msg,hex) is used to send data to the server with the specified port number

Parameter:

port:

target server port, int type

IP:

target server IP address, string type

msg:

data sent to server, string type

hex:

Optional parameter, whether it is a hexadecimal number, 1 sends data as a hexadecimal number (default is 0), int type

Return value:

ret:

greater than or equal to 0: success

is less than 0: failed

Note:

none

Example:

```
ret=udp_client_send_data(777,"192.168.1.100","test")
```

4.2.4 unconnect udp server

```
ret disconnect_udp_server(port, IP)
```

Function:

disconnect_udp_server(port, IP) is used to disconnect the client from the server

Parameter:

port:

optional parameter, port number (if not written, all UDP connections will be disconnected by default), int type

IP:

optional parameter, target server IP address, string type

Return value:

ret:

greater than or equal to 0: success

is less than 0: failed

Note:

None

Example:

```
ret=disconnect_udp_server(777,"192.168.1.100")
```

Chapter 5 485 communication

5.1 Open 485 interface

```
ret rs485_open()
```

Function:

rs485_open is used to open the 485 interface

Parameter:

none

Return value:

ret:

greater than or equal to 0: open successfully

-1: failed to open

Note:

none

Example:

```
ret=rs485_open()
```

5.2 Set 485 serial port configuration

```
ret rs485_setopt(speed,bits,event,stop)
```

Function:

rs485_setopt is used to set the configuration of the 485 serial port

Parameter:

speed:

baud rate, int type

bits:

data length 7/8, int type

event:

parity ”O”, ”N”, ”E”, string type

stop:

stop bit 1/2, int type

Return value:

ret:

greater than or equal to 0: set successfully

-1: Setup failed

Note:

none

Example:

```
ret=rs485_setopt(9600,8,"N",1)
```

5.3 Receive data

```
ret, recv_buff rs485_recv(time_out, hex, len)
```

Function:

rs485_recv is used for 485 read operations

Parameter:

time_out:

Timeout time, unit: ms, int type

hex:

Whether is a hexadecimal number, the received data of 1 is in hexadecimal character format (default is 0), int type

len:

optional parameter, the length you want to get, if it exceeds 1024, it will be automatically set to 1024, int type

Return value:

ret:

read length (all converted to character length)

0, -1: Failed to read

recv_buff:

get data

Note:

none

Example:

```
ret,recv_buff=rs485_recv(100,0,512)
```

5.4 Send data

```
ret rs485_send(buff,hex)
```

Function:

rs485_send is used for 485 send operation

Parameter:

buff:

characters to send, string type

hex:

Whether is a hexadecimal number, the data sent by 1 is in hexadecimal character format, int type

Return value:

ret:

1: Successfully sent

-1: failed to send

Note:

none

Example:

```
ret=rs485_send("test",0)
```

5.5 Close 485 interface

```
ret rs485_close()
```

Function:

rs485_close is used to close the 485 interface

Parameter:

none

Return value:

ret:

greater than or equal to 0: close successfully

-1: close failed

Note:

none

Example:

```
ret=rs485_close()
```

5.6 Empty 485 buffer

```
rs485_flush()
```

Function:

rs485_flush is used to empty the buffer of 485

Parameter:

none

Return value:

nil

Note:

none

Example:

```
ret=rs485_flush()
```

5.7 Example

Example:

```
open = rs485_open()
if(open >= 0) then
        set = rs485_setopt(9600,8,"N",1)
        elite_print("set = ", set)
        if(set >= 0) then
                while(1) do
                        repeat
                        ret,recv_buff=rs485_recv(500,0)
                        until(ret~=0)
                elite_print("receive data :",recv_buff)
                rs485_send(recv_buff)
                end
        end
end
rs485_close()
```

Chapter 6 232 communication

6.1 Open 232 interface

```
ret rs232_open()
```

Function:

rs232_open() is used to open the 232 interface

Parameter:

none

Return value:

ret:

greater than or equal to 0: open successfully

-1: failed to open

Note:

none

Example:

```
ret=rs232_open()
```

6.2 Set 232 serial port configuration

```
ret rs232_setopt(speed,bits,event,stop)
```

Function:

rs232_setopt(speed,bits,event,stop) is used to set the configuration of the 232 serial port

Parameter:

speed:

baud rate, int type

bits:

data length 7/8, int type

event:

parity ”O”, ”N”, ”E”, string type

stop:

stop bit 1/2, int type

Return value:

ret:

greater than or equal to 0: set successfully

-1: Setup failed

Note:

none

Example:

```
ret=rs232_setopt(9600,8,"N",1)
```

6.3 Receive data

```
ret, recv_buff rs232_recv(time_out, hex, len)
```

Function:

rs232_recv(time_out,hex,len) is used for 232 read operations

Parameter:

time_out:

Timeout time, unit: ms, int type

hex:

Whether is a hexadecimal number, the received data of 1 is in hexadecimal character format (default is 0), int type

len:

optional parameter, the length you want to get, if it exceeds 1024, it will be automatically set to 1024, int type

Return value:

ret:

read length (all converted to character length)

0, -1: Failed to read

recv_buff:

get data

Note:

none

Example:

```
ret,recv_buff=rs232_recv(100,0,512)
```

6.4 Send data

```
ret rs232_send(buff,hex)
```

Function:

rs232_send(buff,hex) is used for 232 send operation

Parameter:

buff:

characters to send, string type

hex:

Whether is a hexadecimal number, the data sent by 1 is in hexadecimal character format, int type

Return value:

ret:

1: Successfully sent

-1: failed to send

Note:

none

Example:

```
ret=rs232_send("test",0)
```

6.5 close 232 interface

```
ret rs232_close()
```

Function:

rs232_close() is used to close the 232 interface

Parameter:

none

Return value:

ret:

greater than or equal to 0: close successfully

-1: close failed

Note:

none

Example:

```
ret=rs232_close()
```

The above commands are applicable to v2.12.0 and above!!!

6.6 Empty 232 buffer

```
rs232_flush()
```

Function:

rs232_flush() is used to empty the 232 buffer

Parameter:

none

Return value:

nil

Note:

This command is applicable to v2.16.2 and above.

Example:

```
ret=rs232_flush()
```

6.7 Example

```
open = rs232_open()
if(open >= 0) then
        set = rs232_setopt(9600,8,"N",1)
        elite_print("set = ", set)
        if(set >= 0) then
                while(1) do
                        repeat
                        ret,recv_buff=rs232_recv(500,0)
                                until(ret~=0)
                                elite_print("receive data :",recv_buff)

                                rs232_send(recv_buff)
                end
        end
end
rs232_close()
```

Chapter 7 TCI communication

7.1 Open end 485 interface

```
ret tci_open()
```

Function:

tci_open() is used to open the terminal 485 interface

Parameter:

without

Return value:

ret:

greater than or equal to 0: open successfully

-1: failed to open

Note:

none

Example:

```
ret=tci_open()
```

7.2 Set the configuration of the TCI serial port

```
ret tci_setopt(speed,bits,event,stop)
```

Function:

tci_setopt(speed,bits,event,stop) is used to set the configuration of the TCI serial port

Parameter:

speed:

baud rate, int type

bits:

data length 8, int type

event:

parity ”O”, ”N”, ”E”, string type

stop:

stop bit 1/2, int type

Return value:

ret:

greater than or equal to 0: set successfully

-1: Setup failed

Note:

none

Example:

```
ret=tci_setopt(9600,8,"N",1)
```

7.3 Receive data

```
ret, recv_buff tci_recv(time_out, hex, len)
```

Function:

tci_recv(time_out,hex,len) is used for TCI read operation

Parameter:

time_out:

Timeout time, unit: ms, int type

hex:

Whether is a hexadecimal number, the received data of 1 is in hexadecimal character format (default is 0), int type

len:

optional parameter, the length you want to get, if it exceeds 1024, it will be automatically set to 1024, int type

Return value:

ret:

read length (all converted to character length)

0, -1: Failed to read

recv_buff:

get data

Note:

none

Example:

```
ret,recv_buff=tci_recv(100,0,512)
```

7.4 Send data

```
ret tci_send(buff,hex)
```

Function:

tci_send(buff,hex) is used for TCI send operation

Parameter:

buff:

characters to send, string type

hex:

Whether is a hexadecimal number, the data sent by 1 is in hexadecimal character format, int type

Return value:

ret:

1: Successfully sent

-1: failed to send

Note:

none

Example:

```
ret=tci_send("test",0)
```

7.5 Close TCI interface

```
ret tci_close()
```

Function:

tci_close() is used to close the TCI interface

Parameter:

none

Return value:

ret:

greater than or equal to 0: close successfully

-1: close failed

Note:

none

Example:

```
ret=tci_close()
```

7.6 Empty TCI buffer

```
tci_flush()
```

Function:

tci_flush() is used to empty the TCI buffer

Parameter:

none

Return value:

nil

Note:

none

Example:

```
ret=tci_flush()
```

7.7 Example

Example:

```
sleep(5)
local open = tci_open()
if (open >= 0) then
        local set = tci_setopt(9600,8,"N",1)
        if (set >= 0) then
                sleep(1)
                tci_send("Testing TCI (testing firmware20190826)")
                while (1) do
                        ret,recv_buff=tci_recv(500,0)
                        sleep(1)
                        if(ret>0) then
                                elite_print(recv_buff)
                                tci_send(recv_buff)
                        end
                end
        else
                elite_print("set tci failed.")
        end
else
        elite_print("open tci failed.")
end
tci_close()
```

Chapter 8 Public Extension

HTTP Reference: http://w3.impa.br/ diego/software/luasocket/http.html

Xml parsing reference: Section 11.1 Appendix 1:lua script

Section 11.2 Appendix 2:xml

JSON parsing reference: Section 11.3 Appendix 3:lua script

Xmlrpc client reference: Section 11.4 Appendix 4

Chapter 9 MODBUS MASTER

9.1 Get modbusrtu handle

```
ctx modbus_new_rtu(choose,baud,parity,data_bit,stop_bit)
```

Function:

modbus_new_rtu(choose,baud,parity,data_bit,stop_bit) is used to get the handle of modbusrtu

Parameter:

choose:

0/2, 0 is 485 ports on the motherboard, 2 is TCI 485 port, int type

baud:

4800, 9600... optional parameters, if not written, the default is 4800, int type

parity:

The ASCII code values of ’E’, ’N’, ’O’ are 69, 78, 79, optional parameters, the default is 78 if not written, int type

data_bit:

8, optional parameter, default 8 if not written, int type

stop_bit:

1/2, optional parameter, default 1 if not written, int type

Return value:

ctx:

modbus handle

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ctx=modbus_new_rtu(2,9600,78,8,1)
```

9.2 Get modbustcp handle

```
ctx modbus_new_tcp(IP,port)
```

Function:

modbus_new_tcp(IP,port) is used to get the handle of modbustcp

Parameter:

IP:

IP address of modbusslave, string type

port:

port number of modbusslave, int type

Return value:

ctx:

modbus handle

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ctx=modbus_new_tcp("192.168.1.15",502)
```

9.3 connect modbus handle

```
ret modbus_connect(ctx)
```

Function:

modbus_connect(ctx) handle for connecting to modbustcp

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front

Return value:

ret:

-1: connection failed

Others: normal connection

Note:

The return value is not used as a sign of whether the connection of the connection rtu handle is successful or not.
This command is applicable to v2.9.3 and above.

Example:

```
ret=modbus_connect(ctx)
```

9.4 close the modbus handle

```
modbus_close(ctx)
```

Function:

modbus_close(ctx) is used to close the handle of modbustcp

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front

Return value:

nil

Note:

This command is applicable to v2.9.3 and above.

Example:

```
modbus_close(ctx)
```

9.5 Set slave address

```
ret modbus_set_slave(ctx,slave_id)
```

Function:

modbus_set_slave(ctx,slave_id) is used to set the slave address

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front of

slave_id:

slave address, int type

Return value:

ret:

-1: error

other: normal

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=modbus_set_slave(ctx,0x2)
```

9.6 modbus writes a value to the specified register address

```
ret modbus_write_register(ctx,reg,value)
```

Function:

modbus_write_register(ctx,reg,value) is used by modbus to write the value to the specified register address

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front of

reg:

register address (in decimal), int type

value:

value, int type

Return value:

ret:

-1: error

other: normal

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret=modbus_write_register(ctx, 771, 1)
```

9.7 modbus reads the signal value from the specified register

```
ret,value modbus_read_register(ctx,reg)
```

Function:

modbus_read_register(ctx,reg) is used by modbus to read the signal value from the specified register

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front of

reg:

register address (in decimal), int type

Return value:

ret:

Read and write success or fail, -1 is failed

value:

returns the value read

Note:

This command is applicable to v2.9.3 and above.

Example:

```
ret,value=modbus_read_register(ctx,771)
```

9.8 modbus reads the signal value from the specified coil

```
ret modbus_read_bits(ctx,reg,len)
```

Function:

modbus_read_bits(ctx,reg,len) is used by modbus to read the signal value from the specified coil

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front of

reg:

coil address, int type

len:

number of coils (<128), int type

Return value:

ret:

table

Note:

This command is applicable to v2.12.0 and above.

Example:

```
ret=modbus_read_bits(ctx, 771, 1)
```

9.9 modbus writes values to specified multiple coils

```
ret modbus_write_bits(ctx,reg,size,value)
```

Function:

modbus_write_bits(ctx,reg,size,value) is used by modbus to write values to multiple specified coils

Parameter:

ctx::
The handle created by modbus_new_rtu, modbus_new_tcp in front of

reg:
Coil Address, int type

size:
Quantity, int type

value:
write data to coil, table type, the data in the table is 0 or 1

Return value:

ret:
-1: error
other: normal

Note:

Modbus cannot write to M0-M527, M1472-M1536.
This command is applicable to v2.12.0 and above.

Example:

```
value_array = {1,1,1}
ret=modbus_write_bits(ctx,1,3,value_array)
```

9.10 modbus writes value to specified coil

```
ret modbus_write_bit(ctx,reg,value)
```

Function:

modbus_write_bit(ctx,reg,value) is used by modbus to write the value to the specified coil

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front of

reg:

Coil Address, int type

value:

write data 1 or 0 to the coil, int type

Return value:

ret:

-1: error

0, 1: normal

Note:

Modbus cannot write to M0-M527, M1472-M1536.
This command is applicable to v2.12.0 and above.

Example:

```
ret=modbus_write_bit(ctx,1,1)
```

9.11 modbus reads input register value

```
ret,reg modbus_read_input_register(ctx,addr)
```

Function:

modbus_read_input_register(ctx,addr) is used for modbus to read the input register value

Parameter:

ctx::

before modbus_new_rtu, handle created by modbus_new_tcp

addr:

input register address, int type

Return value:

ret:

-1: error

0, 1: normal

reg:

Register value returned by

Note:

none

Example:

```
ret,reg=modbus_read_input_register(ctx, 1)
```

9.12 modbus writes values to specified multiple register addresses

```
ret modbus_write_registers(ctx,reg,size,value)
```

Function:

modbus_write_registers(ctx,reg,size,value) is used by modbus to write the value to the specified register address

Parameter:

ctx::

The handle created by modbus_new_rtu, modbus_new_tcp in front of

reg:

register address, int type

size:

Quantity, int type

value:

Write data to register, table type

Return value:

ret:

-1: error

other: normal

Note:

none

Example:

```
ret=modbus_write_registers(ctx,1,3,{0x3444,0x3333,0x4444})
```

9.13 Read the values of multiple registers

```
ret modbus_read_registers(ctx,addr,len)
```

Function:

modbus_read_registers(ctx,addr,len) is used to read the value of multiple registers

Parameter:

ctx:

handle created by

addr:

register address, int type

len:

Number of registers [1,125], int type

Return value:

ret:

empty: failed

list of register values: success

Note:

none

Example:

```
sleep(5)
ctx=modbus_new_rtu(2,9600,78,8,1)
sleep(1)
modbus_set_slave(ctx,0x1)
elite_print(value)
table_c=modbus_read_register(ctx,1,12)
sleep(1)
i=1
for i,v in pairs(table_c) do
    elite_print(v)
end
```

9.14 Get input coil state

```
ret modbus_read_input_bits(ctx,reg,len)
```

Function:

modbus_read_input_bits(ctx,reg,len) is used to get the status of single or multiple input coils

Parameter:

ctx:

The handle created by modbus_new_rtu and modbus_new_tcp in front of

reg:

Coil Address, int type

len:

number of coils (<=128), int type

Return value:

ret:

Status of single or multiple input coils

Note:

none

Example:

```
ret=modbus_read_input_bits(ctx, 0, 1)
```

9.15 Example

9.15.1 RTU Example

```
sleep(5)
ctx=modbus_new_rtu(0,9600,78,8,1)
modbus_connect(ctx)
ret1=modbus_set_slave(ctx,0x3)
if(ret1==-1)then
        elite_print("Wrong address")
end
ret2=modbus_write_register(ctx,771,1)
if(ret2==-1)then
        elite_print("Write error")
end
ret3,value=modbus_read_register(ctx,771)
if(ret3==-1)then
        elite_print("Read error")
end
elite_print("value is:",value)
modbus_close(ctx)
```

9.15.2 TCP Example

```
sleep(5)
ip="192.168.1.7"
port=502
ctx=modbus_new_tcp(ip,port)
repeat
        ret=modbus_connect(ctx)
until(ret~=-1)
ret2=modbus_write_register(ctx,771,1)
if(ret2==-1)then
        elite_print("Write error")
end
ret3,value=modbus_read_register(ctx,771)
if(ret3==-1)then
        elite_print("Read error")
end
elite_print("value is:",value)
modbus_close(ctx)
```

Chapter 10 Profinet

10.1 Get the value of the int input register

```
ret get_profinet_int_input_registers(addr,size)
```

Function:

get_profinet_int_input_registers(addr,size) is used to get the value of the profinet int input register

Parameter:

addr:

Register start address, range: [0,31], int type

size:

Number of registers, range: [1,32], int type

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

without

Example:

```
ret=get_profinet_int_input_registers(0,2)
```

10.2 Get the value of the int output register

```
ret get_profinet_int_output_registers(addr,size)
```

Function:

get_profinet_int_output_registers(addr,size) is used to get the value of the profinet int output register

Parameter:

addr:

Register start address, range: [0,31], int type

size:

Number of registers, range: [1,32], int type

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

without

Example:

```
ret=get_profinet_int_output_registers(0,2)
```

10.3 Get the value of the float type input register

```
ret get_profinet_float_input_registers(addr,size)
```

Function:

get_profinet_float_input_registers(addr,size) is used to get the value of the profinet float type input register

Parameter:

addr:

Register start address, range: [0,31], int type

size:

Number of registers, range: [1,32], int type

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

without

Example:

```
ret=get_profinet_float_input_registers(0,2)
```

10.4 Get the value of the float type output register

```
ret get_profinet_float_output_registers(addr,size)
```

Function:

get_profinet_float_output_registers(addr,size) is used to get the value of the profinet float type output register

Parameter:

addr:

Register start address, range: [0,31], int type

size:

Number of registers, range: [1,32], int type

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

without

Example:

```
ret=get_profinet_float_output_registers(0,2)
```

10.5 Set the value of the int output register

```
ret set_profinet_int_output_registers(addr,size,value)
```

Function:

set_profinet_int_output_registers(addr,size,value) is used to set the value of the profinet int output register

Parameter:

addr:

Register start address, range: [0,31], int type

size:

Number of registers, range: [1,32], int type

Note: the sum of addr and size should be less than or equal to 32

value:

list of register values, table type, list element type, int[-2147483648,2147483647]

Return value:

ret:

-1: failed

1: success

Note:

without

Example:

```
ret=set_profinet_int_output_registers(0,2,{123,-123})
```

10.6 Set the value of the float type output register

```
ret set_profinet_float_output_registers(addr,size,value)
```

Function:

set_profinet_float_output_registers(addr,size,value) is used to set the value of the profinet float type output register

Parameter:

addr:

Register start address, range: [0,31], int type

size:

Number of registers, range: [1,32], int type

Note: the sum of addr and size should be less than or equal to 32

value:

list of register values, table type, list element type, number[-3.40E+38,3.40E+38]

Return value:

ret:

-1: failed

1: success

Note:

without

Example:

```
ret=set_profinet_float_output_registers(0,2,{2.33,-2.33})
```

Chapter 11 Ethernet/IP

11.1 Get the value of the int input register

```
ret get_eip_int_input_registers(addr,size)
```

Function:

get_eip_int_input_registers(addr,size) is used to get the value of the Ethernet/IP int input register

Parameter:

addr:

Register start address, range: [0,31]

size:

Number of registers, range: [1,32]

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_eip_int_input_registers(0,2)
```

11.2 Get the value of the int output register

```
ret get_eip_int_output_registers(addr,size)
```

Function:

get_eip_int_output_registers(addr,size) is used to get the value of the Ethernet/IP int output register

Parameter:

addr:

Register start address, range: [0,31]

size:

Number of registers, range: [1,32]

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_eip_int_output_registers(0,2)
```

11.3 Get the value of the float type input register

```
ret get_eip_float_input_registers(addr,size)
```

Function:

get_eip_float_input_registers(addr,size) is used to get the value of the Ethernet/IP float type input register

Parameter:

addr:

Register start address, range: [0,31]

size:

Number of registers, range: [1,32]

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_eip_float_input_registers(0,2)
```

11.4 Get the value of the float type output register

```
ret get_eip_float_output_registers(addr,size)
```

Function:

get_eip_float_output_registers(addr,size) is used to get the value of the Ethernet/IP float type output register

Parameter:

addr:

Register start address, range: [0,31]

size:

Number of registers, range: [1,32]

Note: the sum of addr and size should be less than or equal to 32

Return value:

ret:

empty: failed

list of register values: success

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_profinet_float_output_registers(0,2)
```

11.5 Set the value of the int output register

```
ret set_eip_int_output_registers(addr,size,value)
```

Function:

set_eip_int_output_registers(addr,size,value) is used to set the value of the Ethernet/IP int output register

Parameter:

addr:

Register start address, range: [0,31]

size:

Number of registers, range: [1,32]

Note: the sum of addr and size should be less than or equal to 32

value:

list of register values, table type, list element type, int[-2147483648,2147483647]

Return value:

ret:

-1: failed

1: success

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=set_eip_int_output_registers(0,2,{123,-123})
```

11.6 Set the value of the float type output register

```
ret set_eip_float_output_registers(addr,size,value)
```

Function:

set_eip_float_output_registers(addr,size,value) is used to set the value of the Ethernet/IP float type output register

Parameter:

addr:

Register start address, range: [0,31]

size:

Number of registers, range: [1,32]

Note: the sum of addr and size should be less than or equal to 32

value:

list of register values, table type, list element type, number[-3.40E+38,3.40E+38]

Return value:

ret:

-1: failed

1: success

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=set_eip_float_output_registers(0,2,{2.33,-2.33})
```

Chapter 12 External force sensor

12.1 Mark the start of the torque data transfer

```
ret start_push_force()
```

Function：

start_push_force() is used to mark the start of the torque data transfer

Parameter:

none

Return value:

ret:

0: success

-1: failed

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=start_push_force()
```

12.2 Transfer the torque data

```
ret push_external_force(index,torque_arry)
```

Function:

push_external_force(index, torque_arry) is used to transfer the torque data

Parameter:

index:

serial number, indicates the transfer sequence, int type, range is int[0,65535]

torque_arry*:

arry of the torque data, double torques[6]

*The definitions of the parameter torque_arry are given as below: if the data name is the force in the X-axis direction and the data type is double, it indicates the force in the X-axis direction under the output coordinate system of the force sensor; if the data name is the force in the Y-axis direction and the data type is double, it indicates the force in the Y-axis direction under the output coordinate system of the force sensor; if the data name is the force in the Z-axis direction and the data type is double, it indicates the force in the Z-axis direction under the output coordinate system; if the data name is the torque on the X axis and the data type is double, it indicates the torque on the X axis under the output coordinate system of the force sensor; if the data name is the torque on the Y axis and the data type is double, it indicates the torque on the Y axis under the output coordinate system of the force sensor; if the data name is the torque on the Z axis and the data type is double, it indicates the torque on the Z axis under the output coordinate system of the force sensor. The unit of the force in the X, Y, Z-axis direction is kg and the unit of the torque is kgM. If the unit of the original data and the coordinate system are different from the defined ones, please change the parameter first and then pass in them.

Return value:

ret:

0: success

-1: failed

Note:

Please call the parameter start_push_force to mark the start of the external torque data transfer. This command is applicable to v3.5.2 and above.

Example:

```
ret=start_push_force()
if(ret==0)then
ret=push_external_force(0,{0,1,2,3,4,5})
end
```

12.3 Stop the transfer of the current torque data

```
ret  stop_push_force()
```

Function:

stop_push_force() is used to end the transfer of the current torque data

Parameter:

none

Return value:

ret:

0: success

-1: failed

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=stop_push_force()
```

12.4 Get the source of the current torque data

```
ret get_force_ctrl_mode()
```

Function:

get_force_ctrl_mode() is used to get the source of the current torque data

Parameter:

none

Return value:

ret:

int[0,4], 0 and 1 indicate that the torque data is from the inside, 2 indicates that the torque data is from SDK, 3 indicates that the torque data is from LUA 4 indicates that the torque data is from the terminal end

Note:

This command is applicable to v3.5.2 and above.

Example:

```
ret=get_force_ctrl_mode()
```

12.5 Example

Example：

```
sleep(1)
ret = get_force_ctrl_mode()
if (ret ~= 3) then
    elite_print ("please change the mode to lua(3)")
else
    -- Mark the start of the data transfer
    ret = start_push_force()
    if (ret == 0) then
        torque = {1.1,2.2,3.3,4.4,5.5,6.6}
        index = 0
        cycle_time = 0.001
        host_time = 5
        loop_cnt = host_time / cycle_time
        while (loop_cnt > 0) do
            -- Periodic torque data transfer
            ret = push_external_force(index, torque)
            if (ret < 0) then
                break
            end
            index = index + 1
            torque[1] = torque[1] + 1
            sleep(cycle_time)
            loop_cnt = loop_cnt - 1
        end
        -- Mark the end of the data transfer
        ret = stop_push_force()
    else
        elite_print("start_push_force failed.")
    end
end
elite_print("script finished!")
```

Chapter 13 Appendix

13.1 Appendix 1

```lua
xml = require('LuaXml')
-- load XML data from file "test.xml" into local table
local xfile = xml.load("test.xml")
-- search for substatement having the tag "scene" local
xscene = xfile:find("scene")
if xscene ~= nil then
        print(xscene)
-- print tag, attribute id and first
substatementprint( xscene:tag(), xscene.id, xscene[1] )
end
xfile:save"t.xml" print("---\nREADY.")
```

13.2 Appendix 2

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<XperiML version="2.0">
<!-- application info -->
<applicationInfo>
        <string id="version">0.5.0</string>
        <string id="date">2004-11-23 - 2006-08-18 </string>
        <string id="shortDescr"> a demonstration application for LuaXML
            scripting </string>
        <string id="usage">[ -i(ini.xml) ]</string>
</applicationInfo>
<!-- initialization of the window -->
<deviceWindow id="0" name="window" deviceContainer="input">
        <string id="winTitle"> my window </string>
        <float id="mouseRelative"> 0 </float>
        <float id="mouseNeutral"> 0.0 </float>
        <bool id="mouseVisible"> 0 </bool>
        <bool id="fullScreen"> 0 </bool>
        <int id="zBufferBits"> 24 </int>
        <int id="stencilBufferBits"> 0 </int>
        <int id="winSizeX"> 800 </int>
        <int id="winSizeY"> 600 </int>
        <floatArray id="bgColor"> 0.5 0.6 0.7 </floatArray>
        <!-- 2d visualization / overlay initialization -->
        <overlay>
                <floatArray id="bgNormalColor"> 1.0 1.0 1.0 0.2 </
                   floatArray>
                <floatArray id="fgNormalColor"> 1.0 0.0 1.0 0.7 </
                   floatArray>
                <floatArray id="bgSelectColor"> 0.0 0.5 1.0 0.7 </
                   floatArray>
                <floatArray id="fgSelectColor"> 1.0 1.0 1.0 1.0 </
                   floatArray>
                <!-- overlay plane minimum and maximum coordinates -->
                <float id="minX"> -1.0 </float>
                <float id="minY"> -1.0 </float>
                <float id="maxX"> 1.0 </float>
                <float id="maxY"> 1.0 </float>
        </overlay>
        <!-- X  Y  Z  H  P  R (output axes) -->
        <axesInputMapping> 0 1 2 3 4 5 </axesInputMapping>
        <axesInputScale> 1 1 1 1 1 1 </axesInputScale>
        <axesInputShift> 0 0 0 0 0 0 </axesInputShift>
</deviceWindow>
<deviceGraphics d="0">
        <float id="nearClipping"> 0.1 </float>
        <float id="farClipping"> 2000 </float>
        <float id="frustumLeft"> -1.0 </float>
        <float id="frustumRight"> 1.0 </float>
        <float id="frustumBottom"> -.75 </float>
        <float id="frustumTop"> .75 </float>
        <bool id="backFaceCulling"> 1 </bool>
        <!-- camera initialization -->
        <camera object="observer" pos="0 -3 1.6 0 0 0"/>
</deviceGraphics>
<!-- audio device initialization -->
<deviceAudio id="0" class="deviceAudioAL">
        <!-- specific global settings for OpenAL -->
        <int id="distanceModel"> 1 </int>
        <!-- valid arguments are NONE=0, INVERSE\_DISTANCE=1 (default),
            INVERSE\_DISTANCE\_CLAMPED=2, see OpenAL ref. man. p.21 -->
        <float id="dopplerVelocity"> 330.0 </float>
        <!-- corresponds to sonic speed for doppler effect calculations
            -->
        <float id="dopplerFactor"> 1.0 </float>
        <!-- additional scaling factor for doppler effect calculations
           -->
</deviceAudio>
<!-- internal script definitions -->
<scripts>
        <script name="main" mime="application/x-lua">
                <![CDATA[
                -- initialization
                if nFrames==nil then
                        nFrames=0;  tLastFrames=ve.now();
                end
                -- termination
                if obj.getFlag(INPUT,27) then
                        ve.exit();
                end
                -- update info line:
                if ve.now()>=tLastFrames+1.0 then
                        tLastFrames=ve.now()
                        ovl.text(1,-1,ALIGN\_RIGHT,ALIGN\_BOTTOM,
                           nFrames," fps");
                        nFrames=0;
                end
                nFrames=nFrames+1;
                ]]>
        </script>
</scripts>
<!-- resource definition -->
<resources>
        <resource mime="model/vrml" id="1" name="virtualab"
        url="resources/virtualab.wrl"/>
        <resource mime="model/vrml" id="2" name="surface"
        url="resources/virtualab\_surface.wrl"/>
        <resource mime="model/vrml" id="3" name="cube"
        url="resources/box.wrl"/>
        <resource mime="model/vrml" id="5" name="ball"
        rl="resources/ball01.wrl"/>
        <resource mime="image/png" id="6" name="veRner"
        url="resources/veRner\_small.png" sizeX="6" sizeY="3"/>
        <resource mime="image/png" id="7" name="explo"
        url="resources/explo.png" sizeX="3" sizeY="3" timeSpan="1.0"
           loop="true" tileX="4" tileY="4" usePitch="true"/>
        <resource mime="font/txf" id="10" name="default"
        url="default.txf" size="24"/>
        <resource mime="audio/wav" id="11" loop="true"
        url="resources/ding.wav"/>
        <resource  mime="audio/wav" id="12" pitch="1.0" loop="true"
           attenuationDist="3.0"
        url="resources/riff01.wav"/>
        <resource mime="audio/wav" id="13" pitch="1.0" loop="true"
        url="resources/river.wav"/>
        <container id="20" name="box" shape="3" sound="12"/>
        <container id="21" name="ufo" shape="5" sound="13"/>
</resources>
<!-- scene and simulation initialization -->
<scene id="0" script="main">
        <object id="0" name="observer" script="camera.lua" input="
           window"/>
        <object id="1" shape="1" surface="2" pos="0 0 0"/>
        <object id="3" shape="20" sound="20" pos="-5 5 0 225 0 0"/>
        <object id="5" shape="21" sound="21" pos="2 2 2" speed="0 2 0
           30 0 0"/>
        <object id="6" shape="6" pos="0 7 5"/>
        <object id="7" shape="7" pos="-4.5 4.5 1.7"/>
        <object id="11" sound="11" pos="10 -10 0"/>
        <light id="0" enabled="1" position="-.5 -.75 1.0 0.0"
        ambient="0.3 0.3 0.3 1.0" diffuse="0.7 0.7 0.7 1.0" specular="1
           .0 1.0 1.0 1.0"/>
</scene>
<cdata_test>
        <chars><![CDATA[x<works>]]></chars>
        <tagged><![CDATA[<works>]]></tagged>
        <open><![CDATA[<]]></open>
        <close><![CDATA[>]]></close>
        <empty><![CDATA[]]></empty>
</cdata_test>
</XperiML>
```

13.3 Appendix 3

```
-- Additional path that may be required
require("json")  local testStrings = {
        [[{1:[1213.3e12, 123 , 123, "hello", [12, 2], {1:true /*test
           */}]}]],
        [[{"username":"demo1","message":null,"password":""}]],
        [[{"challenge":"b64d-fnNQ6bRZ7CYiNIKwmdHoNgl9JR9MIYtz
        jBhpQzYXCFrgARt9mNmgUu07
   FoODGr1NieT9yTeB2SLztGkvIA4NXmN9Bi27hqx1ybJIQq6S2
        L-AjQ3VTDClSmCsYFPOm9EMVZDZ0j hBX1fXw3o9VYj1j9KzSY5VCSAzGqYo-
   cBPY\n.b64","cert":"
        b64MIIGyjCCBbKgAwIBAgIKFAC1ZgAAAAAUYzANBgkqh
        kiG9w0BAQUFADBZMRUwEwYKCZImiZP
        yLGQBGRYFbG9tp8uQuFjWGS_KxTHXz9v
        kLNFjOoZY2bOwzsdEpshuYSdvX-9
   bRvHTQcoMNz8Q9nXG1aMl5x1nbV5byQNTCJlz4gzMJeNfe
        KGci pdCj7B6e_VpF-n2P-dFZizUHjxMksCVZ3nTr51x3Uw\n.b64",
        "key":"D79B30BA7954DF520B44897A 6FF58919"}]],
        [[{"key":"D79B30BA7954DF520B44897A6FF58919"}]],
        [[{"val":undefined}]],
        [[{
        "Image": {
                "Width": 800,
                "Height": 600,
                "Title": "View from 15th Floor",
                 "Thumbnail": {
                    "Url":
                    "http://www.example.com/image/481989943",
                     "Height": 125,
                 "Width": "100"
                                },
                  "IDs": [116, 943, 234, 38793]
                 }
             }]], [[ [
        {
           "precision": "zip",
       "Latitude": 37.7668,
           "Longitude": -122.3959,
       "Address": "",
       "City": "SAN FRANCISCO",
       "State": "CA",
           "Zip":    "94107",
       "Country": "US"
        },
        {
           "precision": "zip",
           "Latitude": 37.371991,
           "Longitude": -122.026020,
        "Address":"",
            "City": "SUNNYVALE",
        "State":"CA",
            "Zip":"94085",
            "Country":"US"
         }
           ] ]],
         [[[null,true,[1,2,3],"hello\"],[world!"] ]],
         [[ [{"0":"tan\\\\","model\\\\":"sedan"},{"0":"red","model":"sp
            orts"}] ]],
         [[ {"1":"one","2":"two","5":"five"} ]],
         [=[ [[[[[[[[[[[[[[[[[[["Not too deep"]]]]]]]]]]]]]]]]]]] ]=]
}
for i, v in ipairs(testStrings) do
        print("Testing: #" .. i)
        local dec = json.decode(v)
        json.util.printValue(dec, "JSONVALUE")
        local reenc = json.encode(dec)
        print("RE_ENC: ", reenc)
        local redec = json.decode(reenc)
        json.util.printValue(redec,"REDECJSONVALUE")
end
local testValues = {
        {[300] = {nil, true, 1,2,3, nil, 3}}
}
for i, v in ipairs(testValues) do
        local ret = json.encode(v)
        print(ret)
        local dec = json.decode(ret)
json.util.printValue(dec, "Encoded value")
print("Re-encoded", json.encode(dec))
end
```

13.4 Appendix 4

```
sleep(0.5)
local xmlrpchttp=require("xmlrpc.http")
-- xmlrpchttp.callparameter 1: xmlrpc server address, parameter 2: the
   name of the requested method, other parameters: the parameters that
   need to be entered for the corresponding method (some write a few)
```

```
local ok, res = xmlrpchttp.call("http://172.19.0.102:8080/RPC2", "
   dataSum", 1, 2)
elite_print(tostring(ok))
elite_print("Result: ",res)
```

ELITE ROBOTS
艾利特机器人

ALWAYS EASIER THAN BEFORE

- Contact Us

Sales & Service: market@elibot.cn
Technical Support: tech@elibot.cn

- Shanghai

Building 18, Lane 36,
Xuelin Road, Shanghai

- Suzhou

1F, Building 4,
No 259 Changyang Street, Suzhou
+86-400-189-9358
+86-0512-83951898

- Beijing

Room 1102, Building 6, No. 2,
Ronghua South Road, Beijing

- Shenzhen

Room 202, Building 1A,
Hangkong Road, Shenzhen

WeChat
Official Account

www.elibot.cn