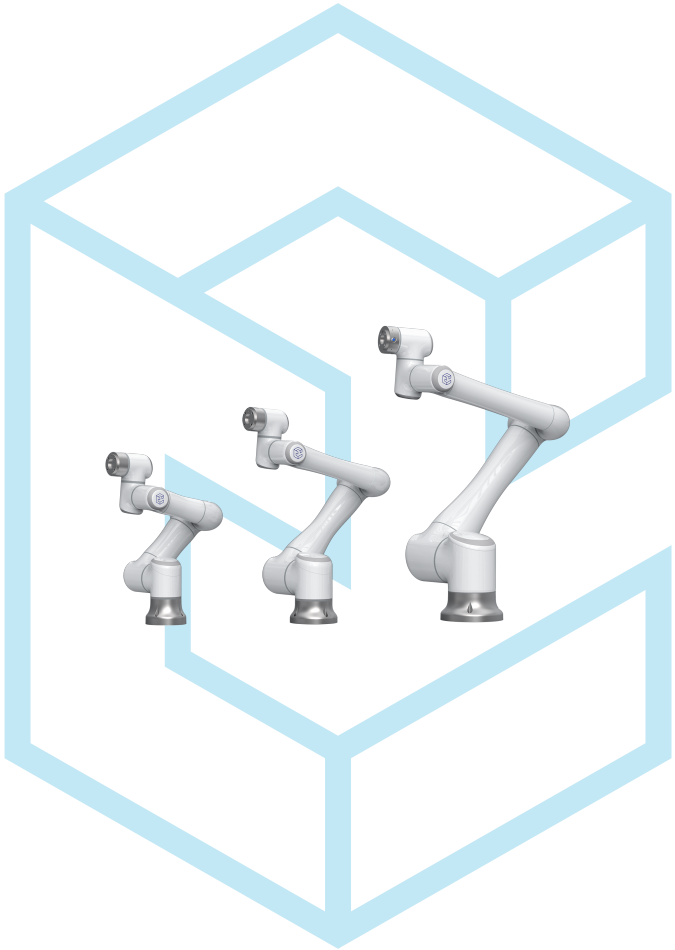
ELITE ROBOT
艾利特机器人

ELITE ROBOT EC系列

编程手册



Lua 脚本手册

苏州艾利特机器人有限公司

2021-12-24

版本：6.0.54

目录

第 1 章 语言定义

1.1 标识符与变量

标识符可以由非数字打头的任意字母，下划线和数字构成。可用于对变量，函数等命名。下列关键字是保留的，不可用于标识符命名:

and not or break return do then if else elseif end true false for while repeat until in function goto local

示例:

1. 标识符命名合法的：a _abc a_123
2. 数字常量是合法的：1 123 0xff 0xABCD
3. 合法的浮点常量：1.0 3.141592 1.6e-2 100e1
4. 字符串变量用""表示，例如："abcdef"
5. 数组变量以 {} 表示，例如：a={1.0,0.2,3.0}

全局变量与局部变量的作用域不同，声明局部变量前加上 local 关键字，例如 local a=5 ，否则就是一个全局变量。

EliteScript 中表达式语法是非常标准的，如下:

1. 算术表达式

   6+2-3

   5*2/3

   (2+3)*4/(5-6)

2. 逻辑表达式

   true or false and (2==3) 1 ＞ 2

   or 3~=4 or 5 ＜-6 not 9 ＞ =10

   and 100 ＜ =50

3. 变量赋值

   A = 100

   bar = ture

   PI = 3.1415

   name = "Lily"

   positon = {0.1,-1.0,0.2,1.0,0.4,0.5}

EliteScript 中的变量类型不需在前面修饰，而是由变量的第一个赋值推导出的。比如上例中，A 是一个整数，bar 是一个布尔值，PI 是一个浮点数，name 是一个字符串，position 是一个数组。

EliteScript 中的基本变量类型有：数字，布尔，字符串，数组。

1.2 控制流语句

程序的控制流可以通过 if, while, repeat, for 这些控制结构来实现，在 EliteScript 中，它们的语法规则都符合通常定义，语法为：

1.2.1 分支语句

if(exp1) then
    ——[[exp1 表达式为 true 时执行该语句块]]
    else if(exp2) then
    ——[[exp2 表达式为 true 时执行该语句块]]
    else
    ——[[满足其他条件时执行该语句块]]
    end
例如：

```
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 循环

while(exp) do
    —— [[exp 表达式为 true 时循环执行该语句块]]
    end
例如：

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

1.2.3 repeat 循环

repeat

—— [[exp 表达式为 false 时循环执行该语句块]]

until(exp)

如下程序打印：10 11 12 13 14 15

```
1 A = 10
2 repeat
3         print(A)
4         A = A+1
5 until(A>15)
```

1.2.4 for 循环

for var=exp1,exp2,exp3 do

—— [[执行的语句块]]

end

如下程序打印：10 9 8 7 6 5 4 3 2 1

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

可以使用 break 停止某个循环, 使用 goto 语句实现跳转，可以通过 return 直接返回。

特别注意，EliteScript 不支持 continue 语句，但可以使用 goto 间接实现。

1.3 操作符

数字操作符	含义
+	加法
*	乘法
–	减法
/	浮点除法
//	向下取整除法
%	取模
^	乘方
–	取负

位操作符	含义
&	按位与
|	按位或
~	按位异或
>>	右移
<<	左移
~	按位非

比较操作符	含义
==	等于
~=	不等于
<	小于
>	大于
< =	小于等于
> =	大于等于

逻辑操作符	含义
and	与
or	或
not	非

字符串连接符：写作两个点..，如 12..34，等价 1234。

取长度操作符：为 #，字符串的长度是它的字节数。

优先级定义（由低向高）：

or

and

<，>，<=，>=，~=，==

|

~

&

«，»

..

+，-

*，/，//，%

#，—(unary)，not

^

可以用括号来改变运算次序。连接操作符.. 和乘方操作 ^ 是从右至左的。其它所有的操作都是从左至右。

1.4 函数

1.4.1 函数定义语法

函数定义的语法如下：

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

1.4.2 自定义函数

自定义加函数：

function add(a,b)

return (a+b)

end

函数调用：

x= add(3,4)

函数可以无返回值，也可以有一个或多个返回值，例如:

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

函数调用：

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

1.4.3 分割字符串

string.sub(strccd, 起始位置, 终止位置)

—— 获取指定位置长度的字符串

string.len(目标字符串)

—— 获取字符串的长度函数调用

function string.split(str, delimiter)

—— 分割字符串，带分隔字符串，返回数组，str 要分离的字符串，delimiter 分隔符

```lua
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
```

内在有 string:split 进行字符串分隔

1.4.4 string 字符串

string.len(s) 返回字符串 s 的长度

string.lower(s) 将 s 中的大写字母转换成小写

string.upper(s) 将 s 中的小写字母转换成大写

string.sub(s,i,j) 函数截取字符串 s 的从第 i 个字符到第 j 个字符之间的串。

string.char() 字符转换成数字

string.byte() 数字转换成字符

1.4.4.1 字符串格式化

string.format() 函数来生成具有特定格式的字符串，函数的第一个参数是格式，之后是对应格式中每个代号的各种数据。

以下实例演示了如何对字符串进行格式化操作：

格式字符串可能包含以下的转义码:

%c - 接受一个数字, 并将其转化为 ASCII 码表中对应的字符

string.format(“%c”, 83) -- 输出 S

%d, %i - 接受一个数字并将其转化为有符号的整数格式

string.format(“%+d”, 17.0) -- 输出 +17

%o - 接受一个数字并将其转化为八进制数格式

string.format(“%o”, 17) -- 输出 21

%u - 接受一个数字并将其转化为无符号整数格式

string.format(“%u”, 3.14) -- 输出 3

%x - 接受一个数字并将其转化为十六进制数格式, 使用小写字母

string.format(“%x”, 13) -- 输出 d

%X - 接受一个数字并将其转化为十六进制数格式, 使用大写字母

string.format(“%X”, 13) —— 输出 D
%e - 接受一个数字并将其转化为科学记数法格式, 使用小写字母 e
string.format(“%e”, 1000) —— 输出 1.000000e+03
%E - 接受一个数字并将其转化为科学记数法格式, 使用大写字母 E
string.format(“%E”, 1000) —— 输出 1.000000E+03
%f - 接受一个数字并将其转化为浮点数格式
string.format(“%6.3f”, 13) —— 输出 13.000
%g(%G) - 接受一个数字并将其转化为%e(%E, 对应%G) 及%f 中较短的一种格式
%q - 接受一个字符串并将其转化为可安全被 Lua 编译器读入的格式
string.format(“%q”, “One\nTwo”) —— 输出 “One\Two”
%s - 接受一个字符串并按照给定的参数格式化该字符串
string.format(“%s”, “monkey”) —— 输出 monkey
string.format(“%10s”, “monkey”) —— 输出 monkey

1.5 http client 支持

```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
```

```
23 end
24
25 print("Response body:")
26 if type(response_body) == "table" then
27     print(table.concat(response_body))
28 else
29     print("Not a table:", type(response_body))
30 end
```

第 2 章 内部模块

2.1 延时

```
void sleep(double second);
```

功能： 睡眠等待

参数： second，等待时间，单位：秒

返回： 无

示例：
```
sleep(0.1)
```

2.2 设置机械臂本体 IO 状态

```
void set_robot_io_status(string name, value);
```

功能： 设置机械臂本体 IO 状态，可以设置 Y、M 、AO 变量的状态
机械臂本体标配 IO 名称请参考示教器 IO 设置界面

参数： name 表示 IO 名称, 字符串类型
valueIO 状态值，为 int 类型（对应 Y，M）或 double 类型（对应 AO）

返回： 无

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

注意： 设置 Y 变量的值时，参数 name 为“o**”，例如“o0”表示“Y0”
设置模拟量输出值时，参数 name 为“a**”，比如“a0”表示“AO001”，“a1”表示“AO002”，以此类推。
本命令适用于 v2.9.3 及以上版本。

2.3 获取机器人臂本体 IO 状态

```
double get_robot_io_status(string name)
```

功能： 获取机械臂本体 IO 状态，可以获取 X、Y、M、AI 变量的状态

参数： name 表示 IO 名称, 字符串类型

返回： 对应 IO 的状态值，double 类型

示例：
```
a=get_robot_io_status("M20")
print(a)
```

注意： 获取 X 变量的状态时，参数 name 为“i**”，比如“i0”表示“X0”
获取模拟量输入值时，参数 name 为“a**”, 比如“a0”表示“AI001”，“a1”表示“AI002”，以此类推。
本命令适用于 v2.9.3 及以上版本。

2.4 获取示教器全局变量值

```
variant get_global_variable(string varName)
```

功能： 获取示教器全局变量值，支持获取 B、I、D、P、V 变量

参数： varName 变量名称

返回： 对应变量名称的变量值，返回值类型取决于变量的类型

示例：
```
--获取B、I、D变量，如获取B变量
a = get_global_variable ("B20")
elite_print(a)
--获取B、I、D变量，如获取P变量
a1,a2,a3,a4,a5,a6 = get_global_variable ("P20")
elite_print(a1,a2,a3,a4,a5,a6)
```

2.5 设置示教器全局变量值

```
void set_global_variable(string varName, variant varValue)
```

功能： 设置示教器全局变量值，支持获取 B、I、D、P、V 变量

参数： varName 变量名称 varValue 变量值

返回： 无

示例：
```
--设置B、I、D变量，如设置B变量
set_global_variable ("B20", 1)
--设置P、V变量，如设置P变量
set_global_variable("P20",90,0,-90,80,30,60,0,0)
```

2.6 设置是否调试模式

```
void set_debug(int debug)
```

功能： 设置是否调试模式

参数： 0, 不输出 INFO 信息，1, 输出 INFO 信息

返回： 无

示例：
```
set_debug(1)
```

注意： 该函数为 1 时，输出内部模块的调试信息，不影响 elite_print 的功能。

2.7 打印信息至示教器界面

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

功能： 打印指定信息到示教盒中信息提示窗口

参数： string var1....

返回： 无

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

2.8 逆解函数

```
table get_inv_kinematics(table var1,table var2)
```

功能： 逆解函数

参数： var1:pose（目标点位姿）
var2:joint （参考点关节角度，参考点需接近目标点。若不写则视为参考当前点）

返回： 逆解结果空或者 table:joint（目标点关节角度）

示例：
```
pose={378.538,212.504,134.055,-2.712,-0.791,2.553}
joint={10.081,-75.007,105.449,-70.694,98.434,89.481,0.000,0
    .000}
Inv_data=get_inv_kinematics(pose,joint)
```

注意： pose={x,y,z,rx,ry,rz}，size 为 6 的数组（注意：pose 中的 rx,ry,rz 均为弧度，下同）
joint={j1,j2,j3,j4,j5,j6,j7,j8}size 为 8 的数组

2.9 正解函数

```
table get_fwd_kinematics(table var1)
```

功能： 正解函数

参数： var1:joint（目标点关节角度）

返回： 正解结果空或者 table:pose（目标点位姿）

示例：
```
joint={10.081,-75.007,105.449,-70.694,98.434,89.481,0.000,0
    .000}
fwd_data=get_fwd_kinematics(joint)
```

注意： pose={x,y,z,rx,ry,rz}size 为 6 的数组
joint={j1,j2,j3,j4,j5,j6,j7,j8}size 为 6~8 的数组

2.10 位姿的逆

```
table pose_inv(table var1)
```

功能： 返回 p 位姿的逆

参数： var1:pose

返回： 结果空或者 table:pose

示例：
```
pose_inv_data=pose_inv({0,1,2,3,4,5})
```

注意： pose={x,y,z,rx,ry,rz}size 为 6 的数组

2.11 位姿的乘

```
table pose_mul(table var1,table var2)
```

功能： 位姿的乘

参数： var1:pose
var2:pose

返回： 结果空或者 table:pose

示例：
```
pose_mul_data=pose_mul({0,1,2,3,4,5},{0,1,2,3,4,5})
```

注意： 注：pose={x,y,z,rx,ry,rz}size 为 6 的数组

2.12 获取机器人模式

```
int get_robot_mode()
```

功能： 获取机器人模式

参数： 无

返回： 0 示教模式，1 自动模式，2 远程模式

示例：
```
get_robot_mode()
```

注意： 本命令适用于 v2.9.3 及以上版本。

2.13 获取机器人 JBI 运行模式

```
int get_cycle_mode()
```

功能： 获取机器人 JBI 运行模式

参数： 无

返回： 0 单步，1 单循环，2 连续循环

示例：
```
get_cycle_mode()
```

注意： 本命令适用于 v2.9.3 及以上版本。

2.14 获取机器人伺服使能状态

```
int get_servo_status()
```

功能： 获取机器人伺服使能状态

参数： 无

返回： 0 伺服使能关闭，1 伺服使能打开

示例：
```
get_servo_status()
```

注意： 本命令适用于 v2.9.3 及以上版本。

2.15 获取机器人运行状态

```
int get_robot_state()
```

功能： 获取机器人运行状态

参数： 无

返回： 0 停止，1 暂停，3 运行中，4 报警，5 碰撞

示例：
```
get_robot_state()
```

2.16 获取机器人坐标系

```
int get_current_coord()
```

功能： 获取机器人坐标系

参数： 无

返回： 0 关节坐标系，1 直角坐标系，2 工具坐标系，3 用户坐标系，4 圆柱坐标系

示例：
```
get_current_coord()
```

注意： 本命令适用于 v2.9.3 及以上版本。

2.17 获取机器人位姿

```
table get_robot_pose()
```

功能： 获取机器人位姿

参数： 无

返回： 机器人当前位姿 [x,y,z,rx,ry,rz]
注：rx、ry、rz 为弧度

示例：
```
get_robot_pose()
```

注意： 本命令适用于 v2.9.3 及以上版本。
本命令不推荐使用。

2.18 获取机器人关节角度

```
table get_robot_joint()
```

功能： 获取机器人关节角度（以角度返回）

参数： 无

返回： 机器人当前关节角度 [joint1,joint2,joint3,joint4,joint5,joint6,joint7,joint8]

示例：
```
get_robot_joint()
```

注意： 本命令适用于 v2.9.3 及以上版本。

2.19 获取机器人力矩信息

```
table get_robot_torque()
```

功能： 获取机器人力矩信息

参数： 无

返回： 机器人当前位姿各关节的力矩 [torque1,torque2,torque3,torque4,torque5,torque6,torque7,torque8]

示例：
```
a = get_robot_torque()
```

注意： 本命令适用于 v2.9.3 及以上版本。

2.20 获取工具坐标系位姿

```
table get_tool_frame(int num)
```

功能： 获取 toolframe 中的数据

参数： 工具坐标编号：num（0-7, 为可选参数, 如果 num 未设置，那么获取当前的工具坐标系位姿）

返回： tool [x,y,z,rx,ry,rz]，注：rx、ry、rz 为弧度

示例：
```
get_tool_frame(0)
```

注意： 本命令适用于 v2.9.4 及以上版本。

2.21 获取用户坐标系位姿

```
table get_user_frame(int num)
```

功能： 获取 userframe 中的数据

参数： 用户坐标编号：num（0-7, 为可选参数, 如果 num 未设置，那么获取当前的用户坐标系位姿）

返回： user [x,y,z,rx,ry,rz]，注：rx、ry、rz 为弧度

示例：
```
get_user_frame(0)
```

注意： 本命令适用于 v2.9.4 及以上版本。

2.22 获取工具坐标系编号

```
int get_tool_no()
```

功能： 获取当前的工具坐标系编号

参数： 无

返回： 当前机器人工具坐标系编号为（0，1，2，3，4，5，6，7）

示例：
```
get_tool_no()
```

注意： 本命令适用于 v2.17.0 及以上版本。

2.23 获取用户坐标系编号

```
int get_user_no()
```

功能： 获取当前的用户坐标系编号

参数： 无

返回： 当前机器人用户坐标系编号为（0，1，2，3，4，5，6，7）

示例：
```
get_user_no()
```

注意： 本命令适用于 v2.17.0 及以上版本。

2.24 检查工具启用状态

```
int check_tool_frame_enable(int num)
```

功能： 检查指定工具是否被启用

参数： 工具坐标编号：num（0-7）

返回： 1: 当前工具号使能，0: 当前工具号未使能

示例：
```
check_tool_frame_enable(0)
```

注意： 本命令适用于 v2.9.4 及以上版本。

2.25 检查用户坐标启用状态

```
int check_user_frame_enable(int num)
```

功能： 检查指定用户坐标是否被启用

参数： 用户坐标编号：num（0-7）

返回： 1: 当前用户坐标号使能，0: 当前用户坐标号未使能

示例：
```
check_user_frame_enable(0)
```

注意： 本命令适用于 v2.9.4 及以上版本。

2.26 获取 M 变量寄存器接口 (0-191)

```
get_robot_register(index)
```

功能： 获取 M 变量的寄存器接口

参数： index：字节定义，int，范围：0-191

返回： 寄存器的接口值，int
0 成功，-1 失败

示例：
```
get_robot_register(77)
```

注意： 本命令适用于 v2.16.2 及以上版本。

2.27 设置 M 变量寄存器接口 (66-191)

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

功能： 设置 M 变量的寄存器接口

参数： index：字节定义，int，范围：66-191
size：int，单位是字节，范围：1-128，且与 index 值的和需小于等于 192
string：hex 字符串

返回： 无

示例：
```
set_robot_register(66,2,"0000")
```

注意： 本命令适用于 v2.16.2 及以上版本。

2.28 获取 M 变量寄存器接口（192-575）

```
get_robot_extra_register(index)
```

功能： 获取 M 变量的寄存器接口

参数： index：字节定义，int，范围：192-575

返回： 寄存器的接口值，int
0 成功，-1 失败

示例：
```
get_robot_extra_register(192)
```

注意： 本命令适用于 v2.16.2 及以上版本。

2.29 设置 M 变量寄存器接口（300-447）

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

功能： 设置 M 变量的寄存器接口

参数： index：字节定义，int，范围：300-447
size：int，单位是字节，且 size 的一半与 index 值的和需小于等于 448
string：hex 字符串

返回： 无

示例：
```
set_robot_extra_register(300,4,"00000000")
```

注意： 本命令适用于 v2.16.2 及以上版本。
系统占用的寄存器不可修改，详细接口请参见 Modbus 手册。

2.30 获取当前 tcp 速度

```
get_current_tcp_spd ()
```

功能： 获取当前 tcp 速度

参数： 无

返回： 当前 tcp 速度，单位：mm/s

示例：
```
get_current_tcp_spd ()
```

注意： 本命令适用于 v2.16.2 及以上版本。

2.31 获取法兰中心在当前基坐标系下的位姿

```
pose get_flange_pose_inbase()
```

功能： 获取法兰中心在当前基坐标系下的位姿数据

参数： 无

返回： 机器人当前位姿 [x,y,z,rx,ry,rz] 注：rx、ry、rz 为弧度

示例：
```
get_flange_pose_inbase()
```

注意： 本命令适用于 v2.17.0 及以上版本。

2.32 获取法兰中心在当前用户坐标系下的位姿

```
pose get_flange_pose_inuser()
```

功能： 获取法兰中心在当前用户坐标系下的位姿

参数： 无

返回： 机器人当前用户坐标系位姿 [x,y,z,rx,ry,rz] 注：rx、ry、rz 为弧度

示例：
```
get_flange_pose_inuser()
```

注意： 本命令适用于 v2.17.0 及以上版本。

2.33 获取机器人位姿

```
table get_tcp_pose()
```

功能： 获取机器人位姿

参数： 无

返回： 机器人当前位姿 [x,y,z,rx,ry,rz] 注：rx、ry、rz 为弧度

示例：
```
get_tcp_pose()
```

注意： 本命令适用于 v2.17.0 及以上版本。

2.34 获取当前 tcp 在当前用户坐标系下的位姿

```
table get_tcp_pose_inuser()
```

功能： 获取当前 tcp 在当前用户坐标系下的位姿

参数： 无

返回： 机器人当前 tcp 在当前用户坐标系下的位姿 [x,y,z,rx,ry,rz] 注：rx、ry、rz 为弧度

示例：
```
get_tcp_pose_inuser()
```

注意： 本命令适用于 v2.17.0 及以上版本。

2.35 获取末端 485 模式

```
int  get_terminal_485_mode()
```

功能： 获取末端 485 模式

参数： 无

返回： 0：初始模式，1：tci 通讯模式，2：modbusRTU 主站模式

示例：
```
ctx=modbus_new_rtu(2,9600,78,8,1)
sleep(2)
    a=get_terminal_485_mode()
    elite_print(a)
```

注意： 本命令适用于 v2.17.0 及以上版本。

第 3 章 TCP 通信

3.1 TCP 服务器

3.1.1 初始化 TCP 服务器

```
void init_tcp_server(int port)
```

功能： 初始化 TCP 服务器

参数： port：端口号

返回： 无

示例：

```
init_tcp_server(8888)
```

3.1.2 判断客户端是否连接了服务器

```
int is_client_connected(std::string IP,std::int port)
```

功能： 判断指定 IP 和端口号的客户端是否连接了服务器

参数： IP：客户端 IP 地址
port: 可选参数，端口号

返回： 如果参数中的 IP 已经连接了 TCP 服务器，则返回 1；否则返回-1

示例：

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

注意： port 参数仅适用于 v2.15.2 及以上版本。

3.1.3 接收客户端数据

```
int, string server_recv_data(std::string IP,std::int hex,std::int
    port)
```

功能： 接收来自指定 IP 和端口号的客户端的数据

参数： IP：客户端 IP 地址
hex：可选参数，是否为 16 进制数，1 的接收数据为 16 进制数（默认为 0）
port: 可选参数，端口号

返回： int 返回接受到的数据数量，接受错误返回-1，string 返回接受到数据

示例：
```
ret,recv=server_recv_data("192.168.1.100",8888)
```

注意： port 参数仅适用于 v2.15.2 及以上版本。

3.1.4 向客户端发送数据

```
int server_send_data(std::string IP,std::string msg,std::int hex,std
    ::int port)
```

功能： 服务器向指定 IP 和端口号的客户端发送消息 msg

参数： IP：客户端 IP 地址；
msg：发送的消息
hex：可选参数，是否为 16 进制数，1 的发送数据为 16 进制数（默认为 0）
port: 可选参数，端口号

返回： int 发送数据数量，发送失败返回-1

示例：
```
server_send_data("192.168.1.100","msg",0,8888)
```

注意： port 参数仅适用于 v2.15.2 及以上版本。

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
```

```
11                          sleep(1)
12                          Ret,recv=server_recv_data(ip)
13                          print(recv,Ret)
14                  end
15          end
16  end
```

3.2 TCP 客户端

3.2.1 连接服务器

```
int connect_tcp_server(std::string IP, int port)
```

功能： 连接指定 IP 和 port 的 TCP 服务器

参数： IP：客户端 IP 地址；
port：端口号

返回： 0 未连接，1 连接成功

示例：
```
connect_tcp_server("192.168.1.100",7777)
```

3.2.2 接收服务器数据

```
int,string client_recv_data(std::string IP,std::double recv_timeout,
    int hex,std::int port)
```

功能： 客户端接收指定 IP 和端口号的服务器发送来的数据

参数： IP：IP 地址；
recv_timeout: timeout 时间 (秒)；
hex：是否为 16 进制数，1 的接收数据为 16 进制字符格式（默认为 0）
port: 可选参数，端口号

返回： string 返回从服务器发送来的数据,int 返回 recv 的数量,-1 表示接受失败需要重新连接

示例：
```
ret,recv=client_recv_data ("192.168.1.100",0.1,0,7777)
注：recv_timeout参数可不写，而用下文函数设置超时时间
```

注意： 本命令适用于 v2.9.4 及以上版本。
port 参数仅适用于 v2.15.2 及以上版本。

3.2.3 整体设置超时时间

```
int client_set_recv_timeout(std::string IP,std::double recv_timeout)
```

功能： 整体设置 recv_timeout 时间与上文中的 recv 组合使用

参数： IP：IP 地址；
recv_timeout: timeout 时间 (秒)

返回： time_out 超时时间

示例：
```
client_set_recv_timeout ("192.168.1.100",0.1)
```

3.2.4 向服务器发送数据

```
int client_send_data(std::string ip,std::string msg,int hex,std::int
port)
```

功能： 客户端向指定 IP 和端口号的服务器发送数据 msg

参数： IP：IP 地址；
msg：向服务器发送的数据返回: string；
hex: 是否为 16 进制数,1 发送数据为 16 进制字符格式（默认为 0）
port: 可选参数，端口号

返回： send 的数量,-1 表示接受失败需要重新连

示例：
```
client_send_data("127.0.0.1","OK",0,7777)
```

注意： 本命令适用于 v2.9.4 及以上版本。
port 参数仅适用于 v2.15.2 及以上版本。

3.2.5 断开 TCP 连接

```
void disconnect_tcp_server(string ip,std::int port)
```

功能： 断开客户端与服务器的连接

参数： IP:IP 地址
port: 可选参数，端口号（不写则默认断开所有 TCP 连接）

返回： 无

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

注意： port 参数仅适用于 v2.15.2 及以上版本。

3.2.6 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)
```

第 4 章 UDP 通信

提醒

本章命令适用于 v2.16.2 及以上版本。

4.1 UDP 服务器

4.1.1 初始化 UDP 服务器

```
int init_udp_server(int port)
```

功能： 初始化 UDP 服务器

参数： port：端口号

返回： int：-1 失败，1 成功

示例：

```
init_udp_server(8888)
```

4.1.2 接收 UDP 客户端数据

```
int,string,string,int udp_server_recv_data(int port,double timeout,
    int hex)
```

功能： 指定端口号的服务器接收客户端数据

参数： port：服务器端口号
timeout：超时时间，单位：s
hex：可选参数，是否为 16 进制数，1 的接收数据为 16 进制数（默认为 0）

返回： int: -1 失败，大于等于 0 成功
string: 接收到的数据
string: 源客户端 IP 地址
int: 源客户端端口号

示例：

```
ret1,recvbuff,client_ip,client_port = udp_server_recv_data
    (777,0.1,0)
```

4.1.3 向 udp 客户端发送数据

```
int udp_server_send_data(int server_port,string msg,int client_port,
    string client_ip,int hex)
```

功能： 服务器向指定端口号和 IP 地址的客户端发送数据

参数： server_port: 服务器端口号
msg: 向客户端发送的数据
client_port: 目标客户端端口号
client_ip: 目标客户端 IP
hex: 可选参数，是否为 16 进制数，1 的发送数据为 16 进制数（默认为 0）

返回： int: -1 失败，大于等于 0 成功

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

4.1.4 关闭 udp 服务器

```
int close_udp_server(int port)
```

功能： 关闭 udp 服务器

参数： port: 可选参数，服务器端口号，不填关闭所有 udp 服务器

返回： int: 小于 0 失败，大于等于零成功

示例：
```
ret = close_udp_server(777)
```

4.2 UDP 客户端

4.2.1 连接 UDP 服务器

```
int connect_udp_server(int port,string ip)
```

功能： 客户端连接指定端口号和 IP 地址的服务器

参数： port: 目标服务器端口
ip: 目标服务器 IP 地址

返回： int: 小于 0 失败，大于等于 0 成功

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

4.2.2 接收 UDP 服务器数据

```
int string udp_client_recv_data(int port,string ip,double timeout,int
    hex)
```

功能： 接收来自指定端口号和 IP 地址的服务器的数据

参数： port: 目标服务器端口
ip: 目标服务器 IP 地址
timeout：超时时间，单位：s
hex：可选参数，是否为 16 进制数，1 的接收数据为 16 进制数（默认为 0）

返回： int: 小于 0 失败，大于等于 0 成功
string: 接收到的数据

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

4.2.3 向 udp 服务器发送数据

```
int udp_client_send_data(int port,string ip,string msg,int hex)
```

功能： 向指定端口号的服务器发送数据

参数： port: 目标服务器端口
ip: 目标服务器 IP 地址
msg: 向服务器发送的数据
hex：可选参数，是否为 16 进制数，1 的接收数据为 16 进制数（默认为 0）

返回： int: 小于 0 失败，大于等于 0 成功

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

4.2.4 取消连接 udp 服务器

```
int disconnect_udp_server(int port,string ip)
```

功能： 取消 udp 客户端与指定端口号和 IP 地址的服务器的连接

参数： 不填参数取消所有绑定
port: 目标服务器端口
ip: 目标服务器 IP 地址

返回： int: 小于 0 失败，大于等于 0 成功

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

第 5 章 485 通信

5.1 打开 485 接口

```
int rs485_open()
```

功能： 打开 485 接口

参数： IP

返回： -1 打开失败，>=0 打开成功

示例：
```
rs485_open()
```

5.2 设置 485 串口配置

```
int rs485_setopt(int speed,int bits,string event,int stop)
```

功能： 设置 485 串口的配置

参数： speed: 波特率，
bits: 数据长度 7/8,
event: 奇偶校验“O”，“N”，“E”，
stop: 停止位 1/2

返回： >=0 设置成功，-1 设置失败

示例：
```
rs485_setopt(9600,8,"N",1)
```

5.3 接收数据

```
int,string rs485_recv(int time_out,int hex,int len)
```

功能： 485 的读操作

参数： time_out: 超时 (ms),
hex: 是否为 16 进制数,1 的接收到的数据为 16 进制字符格式（默认为 0）
len: 可选参数，想要获取的长度，在超过 1024 情况下，会自动被设置成 1024

返回： 读到的长度（都是转化为字符长度），0,-1 读取失败
recv_buff: 获取数据

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

5.4 发送数据

```
int rs485_send(string buff,int hex)
```

功能： 485 的发送操作

参数： buff: 需要发送的字符,
hex: 是否为 16 进制数，1 为发送的数据为 16 进制字符格式

返回： 1：发送成功，-1：发送失败

示例：
```
rs485_send("test",0)
```

5.5 关闭 485 接口

```
int rs485_close()
```

功能： 关闭 485 接口

参数： 无

返回： -1 关闭失败，>=0 关闭成功

示例：
```
rs485_close()
```

5.6 清空 485 缓冲区

```
void rs485_flush()
```

功能： 清空 485 的缓冲区

参数： 无

返回： 无

示例：
```
rs485_flush()
```

注意： 本命令适用于 v2.9.6 及以上版本。

5.7 Example

Example：

```
1  open = rs485_open()
2  if(open >= 0) then
3          set = rs485_setopt(9600,8,"N",1)
4          elite_print("set = ", set)
5          if(set >= 0) then
6                  while(1) do
7                          repeat
8                          ret,recv_buff=rs485_recv(500,0)
9                          until(ret~=0)
10                 elite_print("receive data :",recv)
11                 rs485_send(recv_buff)
12                 end
13         end
14 end
15 rs485_close()
```

第 6 章 232 通信

6.1 打开 232 接口

```
int rs232_open()
```

功能： 打开 232 接口

参数： 无

返回： -1 打开失败，>=0 打开成功

示例：
```
rs232_open()
```

6.2 设置 232 串口配置

```
int rs232_setopt(int speed,int bits,string event,int stop)
```

功能： 设置 232 串口的配置

参数： speed: 波特率，
bits: 数据长度 7/8,
event: 奇偶校验“O”，“N”，“E”，
stop: 停止位 1/2

返回： >=0 设置成功，-1 设置失败

示例：
```
rs232_setopt(9600,8,"N",1)
```

6.3 接收数据

```
int,string rs232_recv(int time_out,int hex,int len)
```

功能： 232 的读操作

参数： time_out: 超时 (ms),
hex: 是否为 16 进制数,1 接收到的数据为 16 进制字符格式（默认为 0）
len: 可选参数，想要获取的长度，在超过 1024 情况下，会自动被设置成 1024

返回： 读到的长度（都是转化为字符长度），0，-1 读取失败；
recv_buff: 获取数据

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

6.4 发送数据

```
int rs232_send(string buff,int hex)
```

功能： 232 的发送操作

参数： buff: 需要发送的字符,
hex: 是否为 16 进制数，1 为发送的数据为 16 进制字符格式

返回： 1 表示发送成功，-1 表示发送失败

示例：
```
rs232_send("test",0)
```

6.5 关闭 232 接口

```
int rs232_close()
```

功能： 关闭 232 接口

参数： 无

返回： -1 关闭失败，>=0 关闭成功

示例：
```
rs232_close()
```

提醒

上述命令适用于 v2.12.0 及以上版本。

6.6 清空 232 缓冲区

```
rs232_flush()
```

功能： 清空 232 的缓冲区

参数： 无

返回： 无

示例：

```
rs232_flush()
```

注意： 本命令适用于 v2.16.2 及以上版本。

6.7 Example

```
1  open = rs232_open()
2  if(open >= 0) then
3          set = rs232_setopt(9600,8,"N",1)
4          elite_print("set = ", set)
5          if(set >= 0) then
6                  while(1) do
7                          repeat
8                          ret,recv_buff=rs232_recv(500,0)
9                                  until(ret~=0)
10                                 elite_print("receive data :",recv_buff)

11                                 rs232_send(recv_buff)
12                 end
13         end
14 end
15 rs232_close()
```

第 7 章 TCI 通信

7.1 打开末端 485 接口

```
int tci_open()
```

功能： 打开末端 485 接口

参数： 无

返回： -1 打开失败，>=0 打开成功

示例：
```
tci_open()
```

7.2 设置 TCI 串口配置

```
int tci_setopt(int speed,int bits,string event,int stop)
```

功能： 设置 TCI 串口的配置

参数： speed：波特率，
bits：数据长度 7/8，
event：奇偶校验“O”,” N”,” E”，
stop：停止位 1/2

返回： >=0 设置成功，-1 设置失败

示例：
```
tci_setopt(9600,8,"N",1)
```

7.3 接收数据

```
int,string tci_recv(int time_out,int hex,int len)
```

功能： TCI 的读操作

参数： time_out：超时 (ms)，
hex：是否为 16 进制数,1 接受到的数据为 16 进制字符格式（默认为 0）
len: 可选参数，想要获取的长度，在超过 1024 情况下，会自动被设置成 1024

返回： 读到的长度（都是转化为字符长度），-1 读取失败 recv_buff：获取数据

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

7.4 发送数据

```
int tci_send(string buff,int hex)
```

功能： TCI 的发送操作

参数： buff：需要发送的字符，
hex：是否为 16 进制数，1 为发送的数据为 16 进制字符格式

返回： 1：发送成功，-1：发送失败

示例：
```
tci_send("test",0)
```

7.5 关闭 TCI 接口

```
int tci_close()
```

功能： 关闭 TCI 接口

参数： 无

返回： -1 关闭失败，>=0 关闭成功

示例：
```
tci_close()
```

7.6 清空 TCI 缓冲区

```
void tci_flush()
```

功能： 清空 tci 的缓冲区
注：暂不建议使用

参数： 无

返回： 无

示例：
```
tci_flush()
```

注意： 本命令适用于 v2.9.6 及以上版本。

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 firmware: 20190826)")
                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()
```

第 8 章 公版扩展

HTTP 参考：http://w3.impa.br/ diego/software/luasocket/http.html
Xml 解析参考：第 10.1 节 附录 1：lua 脚本
第 10.2 节 附录 2：xml
JSON 解析参考：第 10.3 节 附录 3：lua 脚本
Xmlrpc 客户端参考：第 10.4 节 附录 4

第 9 章 MODBUS MASTER

9.1 获取 modbusrtu 句柄

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

功能： 获取 rtumodbusrtu 的句柄

参数： choose: 0 主板 485 口，2 TCI 485 口（必须）
Baud: 4800 9600 ...（非必须，可写可不写，不写默认 4800）
Parity: ‘E’，‘N’，‘O’的 ASCII 码值 69,78,79（非必须，可写可不写，不写默认 78）
Data_bit: 7/8 （非必须，可写可不写，不写默认 8）
Stop_bit: 0/1/2 （非必须，可写可不写，不写默认 1）

返回： ctx: modbus 句柄

示例：
```
ctx = modbus_new_rtu(1,9600,78,8,1)
```

注意： 本命令适用于 v2.9.3 及以上版本。

9.2 获取 modbustcp 句柄

```
ctx modbus_new_tcp(char *ip, int port)
```

功能： 获取 modbustcp 的句柄

参数： ip: modbusslave 的 IP 地址
port: modbusslave 的端口号

返回： ctx: modbus 句柄

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

注意： 本命令适用于 v2.9.3 及以上版本。

9.3 连接 modbus 句柄

```
int modbus_connect(ctx)
```

功能： 连接 modbustcp 的句柄

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

返回： 判断连接 tcp 句柄成功与否，返回-1 连接失败，其它正常。
注意：该返回值不作为连接 rtu 句柄连接成功与否的标志

示例：
```
modbus_connect(ctx)
```

注意： 本命令适用于 v2.9.3 及以上版本。

9.4 关闭 modbus 句柄

```
modbus_close(ctx)
```

功能： 关闭 modbustcp 的句柄

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

返回： 无

示例：
```
modbus_close(ctx)
```

注意： 本命令适用于 v2.9.3 及以上版本。

9.5 设置 slave 地址

```
int modbus_set_slave(ctx, int slave_id)
```

功能： 设置 slave 地址

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
slave_id: slave 地址

返回： -1 出错，其他正常

示例：
```
modbus_set_slave(ctx,0x2)
```

注意： 本命令适用于 v2.9.3 及以上版本。

9.6 往指定地址写入值

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

功能： modbus 往指定地址写入值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
reg: 寄存器地址（为十进制）
value：值

返回： -1 出错，其他正常

示例：
```
modbus_write_register(ctx, 771, 1)
```

注意： 本命令适用于 v2.9.3 及以上版本。

9.7 modbus 从指定寄存器读取信号值

```
int,int modbus_read_register(ctx, int reg)
```

功能： modbus 从指定寄存器读取信号值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
reg: 寄存器地址（为十进制）

返回： ret：读写成功与否，-1 为失败,
value：返回读到的值

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

注意： 本命令适用于 v2.9.3 及以上版本。

9.8 modbus 从指定线圈读取信号值

```
table modbus_read_bits(ctx, int reg, int len)
```

功能： modbus 从指定线圈读取信号值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
reg: 线圈地址
len：线圈个数 (<128)

返回： ret：table

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

注意： 本命令适用于 v2.12.0 及以上版本。

9.9 modbus 往指定多个线圈写入值

```
int modbus_write_bits(ctx, int reg, int size, table value)
```

功能： modbus 往指定多个线圈写入值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
reg: 线圈地址
size：数量
value：写入线圈的数据

返回： -1 出错，其他正常

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

注意： 本命令适用于 v2.12.0 及以上版本。
Modbus 不能对 M0-M527，M1472-M1536 进行写操作。

9.10 modbus 往指定线圈写入值

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

功能： modbus 往指定线圈写入值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
reg: 线圈地址
value：写入线圈的数据 1 或 0

返回： -1 出错，0,1 正常

示例：
```
modbus_write_bit(ctx, 1, 1)
```

注意： 本命令适用于 v2.12.0 及以上版本。
Modbus 不能对 M0-M527，M1472-M1536 进行写操作。

9.11 modbus 读 input 寄存器值

```
modbus_read_input_register(ctx, int addr)
```

功能： modbus 读 input 寄存器值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
addr: input register 地址

返回： ret:-1 出错，0,1 正常 reg：返回的寄存器值

示例：
```
modbus_read_input_register(ctx, 1)
```

9.12 modbus 往指定地址写入值

```
int modbus_write_registers(ctx, int reg, int size, table value)
```

功能： modbus 往指定地址写入值

参数： ctx: 前面 modbus_new_rtu, modbus_new_tcp 创建的句柄
reg: 寄存器地址
size：数量
value: 写入寄存器的数据

返回： -1 出错，其他正常

示例：
```
modbus_write_registers(ctx, 1, 3, {0x3444,0x3333,0x4444})
```

9.13 读取多个寄存器的值

```
table modbus_read_registers(ctx,int addr,int len)
```

功能： 读取多个寄存器的值

参数： ctx：创建的句柄
addr: 寄存器地址
len: 寄存器个数 [1,125]

返回： 失败：返回空，成功：返回寄存器值列表

示例：

```
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 Example

9.14.1 RTU Example

```
1  sleep(5)
2  ctx=modbus_new_rtu(0,9600,78,8,1)
3  modbus_connect(ctx)
4  ret1=modbus_set_slave(ctx,0x3)
5  if(ret1==-1)then
6          elite_print("Wrong address")
7  end
8  ret2=modbus_write_register(ctx,771,1)
9  if(ret2==-1)then
10         elite_print("Write error")
11 end
12 ret3,value=modbus_read_register(ctx,771)
13 if(ret3==-1)then
14         elite_print("Read error")
15 end
16 elite_print("value is:",value)
17 modbus_close(ctx)
```

9.14.2 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)
```

第 10 章 附录

10.1 附录 1

```
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.")
```

10.2 附录 2

```
<?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>
```

```
19          <int id="winSizeX"> 800 </int>
20          <int id="winSizeY"> 600 </int>
21          <floatArray id="bgColor"> 0.5 0.6 0.7 </floatArray>
22          <!-- 2d visualization / overlay initialization -->
23          <overlay>
24                  <floatArray id="bgNormalColor"> 1.0 1.0 1.0 0.2 </
                       floatArray>
25                  <floatArray id="fgNormalColor"> 1.0 0.0 1.0 0.7 </
                       floatArray>
26                  <floatArray id="bgSelectColor"> 0.0 0.5 1.0 0.7 </
                       floatArray>
27                  <floatArray id="fgSelectColor"> 1.0 1.0 1.0 1.0 </
                       floatArray>
28                  <!-- overlay plane minimum and maximum coordinates -->
29                  <float id="minX"> -1.0 </float>
30                  <float id="minY"> -1.0 </float>
31                  <float id="maxX"> 1.0 </float>
32                  <float id="maxY"> 1.0 </float>
33          </overlay>
34          <!-- X  Y  Z  H  P  R (output axes) -->
35          <axesInputMapping> 0 1 2 3 4 5 </axesInputMapping>
36          <axesInputScale> 1 1 1 1 1 1 </axesInputScale>
37          <axesInputShift> 0 0 0 0 0 0 </axesInputShift>
38  </deviceWindow>
39  <deviceGraphics d="0">
40          <float id="nearClipping"> 0.1 </float>
41          <float id="farClipping"> 2000 </float>
42          <float id="frustumLeft"> -1.0 </float>
43          <float id="frustumRight"> 1.0 </float>
44          <float id="frustumBottom"> -.75 </float>
45          <float id="frustumTop"> .75 </float>
46          <bool id="backFaceCulling"> 1 </bool>
47          <!-- camera initialization -->
48          <camera object="observer" pos="0 -3 1.6 0 0 0"/>
49  </deviceGraphics>
50  <!-- audio device initialization -->
51  <deviceAudio id="0" class="deviceAudioAL">
52          <!-- specific global settings for OpenAL -->
53          <int id="distanceModel"> 1 </int>
```

```
54          <!-- valid arguments are NONE=0, INVERSE\_DISTANCE=1 (default),
                INVERSE\_DISTANCE\_CLAMPED=2, see OpenAL ref. man. p.21 -->
55          <float id="dopplerVelocity"> 330.0 </float>
56          <!-- corresponds to sonic speed for doppler effect calculations
                -->
57          <float id="dopplerFactor"> 1.0 </float>
58          <!-- additional scaling factor for doppler effect calculations
               -->
59  </deviceAudio>
60  <!-- internal script definitions -->
61  <scripts>
62          <script name="main" mime="application/x-lua">
63                  <![CDATA[
64                  -- initialization
65                  if nFrames==nil then
66                          nFrames=0;  tLastFrames=ve.now();
67                  end
68                  -- termination
69                  if obj.getFlag(INPUT,27) then
70                          ve.exit();
71                  end
72                  -- update info line:
73                  if ve.now()>=tLastFrames+1.0 then
74                          tLastFrames=ve.now()
75                          ovl.text(1,-1,ALIGN\_RIGHT,ALIGN\_BOTTOM,
                                nFrames," fps");
76                          nFrames=0;
77                  end
78                  nFrames=nFrames+1;
79                  ]]>
80          </script>
81  </scripts>
82  <!-- resource definition -->
83  <resources>
84          <resource mime="model/vrml" id="1" name="virtualab"
85          url="resources/virtualab.wrl"/>
86          <resource mime="model/vrml" id="2" name="surface"
87          url="resources/virtualab\_surface.wrl"/>
88          <resource mime="model/vrml" id="3" name="cube"
```

```
89          url="resources/box.wrl"/>
90          <resource mime="model/vrml" id="5" name="ball"
91          rl="resources/ball01.wrl"/>
92          <resource mime="image/png" id="6" name="veRner"
93          url="resources/veRner\_small.png" sizeX="6" sizeY="3"/>
94          <resource mime="image/png" id="7" name="explo"
95          url="resources/explo.png" sizeX="3" sizeY="3" timeSpan="1.0"
               loop="true" tileX="4" tileY="4" usePitch="true"/>
96          <resource mime="font/txf" id="10" name="default"
97          url="default.txf" size="24"/>
98          <resource mime="audio/wav" id="11" loop="true"
99          url="resources/ding.wav"/>
100         <resource  mime="audio/wav" id="12" pitch="1.0" loop="true"
               attenuationDist="3.0"
101         url="resources/riff01.wav"/>
102         <resource mime="audio/wav" id="13" pitch="1.0" loop="true"
103         url="resources/river.wav"/>
104         <container id="20" name="box" shape="3" sound="12"/>
105         <container id="21" name="ufo" shape="5" sound="13"/>
106 </resources>
107 <!-- scene and simulation initialization -->
108 <scene id="0" script="main">
109         <object id="0" name="observer" script="camera.lua" input="
               window"/>
110         <object id="1" shape="1" surface="2" pos="0 0 0"/>
111         <object id="3" shape="20" sound="20" pos="-5 5 0 225 0 0"/>
112         <object id="5" shape="21" sound="21" pos="2 2 2" speed="0 2 0
               30 0 0"/>
113         <object id="6" shape="6" pos="0 7 5"/>
114         <object id="7" shape="7" pos="-4.5 4.5 1.7"/>
115         <object id="11" sound="11" pos="10 -10 0"/>
116         <light id="0" enabled="1" position="-.5 -.75 1.0 0.0"
117         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"/>
118 </scene>
119 <cdata_test>
120         <chars><![CDATA[x<works>]]></chars>
121         <tagged><![CDATA[<works>]]></tagged>
122         <open><![CDATA[<]]></open>
```

```
123         <close><![CDATA[>]]></close>
124         <empty><![CDATA[]]></empty>
125 </cdata_test>
126 </XperiML>
```

10.3 附录 3

```
1  -- Additional path that may be required
2  require("json")  local testStrings = {
3          [[{1:[1213.3e12, 123 , 123, "hello", [12, 2], {1:true /*test
              */}]}]],
4          [[{"username":"demo1","message":null,"password":""}]],
5          [[{"challenge":"b64d-fnNQ6bRZ7CYiNIKwmdHoNgl9JR9MIYtz
6          jBhpQzYXCFrgARt9mNmgUuO7
       FoODGr1NieT9yTeB2SLztGkvIA4NXmN9Bi27hqx1ybJIQq6S2
7          L-AjQ3VTDClSmCsYFPOm9EMVZDZ0j hBX1fXw3o9VYj1j9KzSY5VCSAzGqYo-
       cBPY\n.b64","cert":"
8          b64MIIGyjCCBbKgAwIBAgIKFAC1ZgAAAAAUYzANBgkqh
9          kiG9w0BAQUFADBZMRUwEwYKCZImiZP
10         yLGQBGRYFbG9tp8uQuFjWGS_KxTHXz9v
11         kLNFjOoZY2bOwzsdEpshuYSdvX-9
       bRvHTQcoMNz8Q9nXG1aMl5x1nbV5byQNTCJlz4gzMJeNfe
12         KGci pdCj7B6e_VpF-n2P-dFZizUHjxMksCVZ3nTr51x3Uw\n.b64",
13         "key":"D79B30BA7954DF520B44897A 6FF58919"}]],
14         [[{"key":"D79B30BA7954DF520B44897A6FF58919"}]],
15         [[{"val":undefined}]],
16         [[{
17         "Image": {
18                 "Width": 800,
19                 "Height": 600,
20                 "Title": "View from 15th Floor",
21                  "Thumbnail": {
22                     "Url":
23                     "http://www.example.com/image/481989943",
24                      "Height": 125,
25                  "Width": "100"
26                                 },
27                   "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 = {
```

```
66          {[300] = {nil, true, 1,2,3, nil, 3}}
67  }
68  for i, v in ipairs(testValues) do
69          local ret = json.encode(v)
70          print(ret)
71          local dec = json.decode(ret)
72  json.util.printValue(dec, "Encoded value")
73  print("Re-encoded", json.encode(dec))
74  end
```

10.4 附录 4

```
1  sleep(0.5)
2  local xmlrpchttp=require("xmlrpc.http")
3  -- xmlrpchttp.call: 参数1: xmlrpc服务器地址, 参数2: 为请求的方法名, 其
       他参数: 为对应方法需要输入的参数 (有几个写几个)
4  local ok, res = xmlrpchttp.call("http://172.19.0.102:8080/RPC2", "
       dataSum", 1, 2)
5  elite_print(tostring(ok))
6  elite_print("Result: ",res)
```

ELITE ROBOT
艾利特机器人

明天比今天更简单一点

联系我们

商务合作：market@elibot.cn

技术咨询：tech@elibot.cn

苏州公司（生产研发基地）

苏州市工业园区长阳街 259 号中新钟园工业坊 4 号楼 1F（总部）

+86-400-189-9358

+86-0512-83951898

北京分公司

北京市北京经济技术开发区荣华南路 2 号院 6 号楼 1102 室

上海分公司

上海市浦东新区学林路 36 弄 18 号楼

深圳技术服务中心

深圳市宝安区航空路泰华梧桐岛科技创新园 1A 栋 202 室

关注公众号了解更多