

SPK Lua 脚本引擎 API 文档

1. 概述

SPK 内置 Lua 5.5 脚本引擎，支持两种工作模式（在 Lua 面板通过单选按钮切换）：

模式	入口	控制方式	用途
Frame Parser（帧解析器）	<code>spk_frame_parser(data)</code>	"Enable Frame Parser" 开关	被动解析串口数据、波形输出
Script Executor（脚本执行器）	脚本顶层代码（整个文件）	"Enable Script Executor"开关	自动化收发测试

- 自动化测试在后台线程运行，UI 不卡顿，可随时中断
- 脚本执行结束后自动输出汇总：`Automation finished: X passed, Y failed`
- 脚本加载后 `print(...)` 的输出会显示在软件 Log 区（`print` 已被重定向）

重要约定

1. **脚本执行器模式**：Run 时引擎在后台线程**从头执行整个脚本文件**，顶层代码即测试流程，阻塞函数直接按顺序编写即可。每次 Run 都会重建干净的 Lua 环境重新执行。
2. **帧解析器模式**：顶层代码只应做函数/变量定义，需要定义 `spk_frame_parser` 函数

2. 通用 API（两种模式均可使用）

2.1 print(...)

输出文本到软件 Log 显示区（等价于原 Lua 的 `print`，已重定向到日志）。多个参数以制表符分隔，末尾无换行。

```
print("测试开始")
print("测试名称: ", name, data)
print("收到数据长度: " .. #data)
```

2.2 println(...)

与 `print` 相同，但末尾自动追加换行 `\n`。

```
println("测试开始")           -- 输出：测试开始\n
println("结果:", "PASS")      -- 输出：结果:\tPASS\n
```

2.3 spk_get_version()

返回 SPK 版本字符串。

参数	返回值
无	<code>string</code> ：如 <code>"spk version 5.6"</code>

```
print(spk_get_version())
```

2.4 spk_send(data)

发送原始字节数据到串口。

参数	返回值
<code>data</code> ：string，原始字节（二进制安全）	<code>boolean</code> ：是否发送成功

```
spk_send("AT\r\n")           -- 发送文本命令
spk_send("\xAA\x55\x01\x02")  -- 发送二进制字节
if not spk_send("AT\r\n") then
    print("发送失败，请检查串口是否打开")
end
```

2.5 spk_send_hex(hexStr)

发送十六进制字符串表示的数据。

参数	返回值
<code>hexStr</code> ：string，hex 文本	<code>boolean</code> ：解析并发送成功；格式错误返回 <code>false</code> 并在日志输出警告

支持格式（可混用）：空格 / 逗号 / 分号 / 连字符 / 下划线 / 换行分隔，`0x` 前缀。

```
spk_send_hex("AA 55 01 02")
spk_send_hex("AA550102")      -- 紧凑格式
spk_send_hex("0xAA, 0x55, 0x01") -- 0x 前缀 + 逗号
```

2.6 spk_sleep_ms(ms)

延时（阻塞当前脚本执行），单位毫秒。测试运行期间可被 中断（中断后脚本以 "automation stopped" 错误中止）。

参数	返回值
<code>ms</code> ：integer，延时毫秒数	无

```
spk_send("AT\r\n")
spk_sleep_ms(200)      -- 等待设备处理
```

2.7 spk_clear_rx()

清空引擎接收缓存。用于测试前清除陈旧数据，避免上次通信残留干扰本次判定。

参数	返回值
无	无

```
spk_clear_rx() -- 清空旧数据
local ok = spk_expect("AT\r\n", "OK", 2000)
```

2.8 spk_port_is_open()

查询串口是否已打开。

参数	返回值
无	boolean

```
if not spk_port_is_open() then
    print("请先打开串口！")
    return
end
```

2.9 spk_report(name, passed, detail)

输出一条测试结果并计入统计。脚本执行结束后引擎自动输出 PASS/FAIL 汇总。

参数	返回值
name : string, 测试项名称	passed 原样返回 (boolean)
passed : boolean, 是否通过	
detail : string (可选), 失败详情	

```
local ok, resp = spk_expect("AT\r\n", "OK", 2000)
spk_report("AT 基础通信", ok, ok and "" or ("应答: " .. resp))
-- 日志输出: [PASS] AT 基础通信 或 [FAIL] AT 基础通信: 应答: ...
```

3. 阻塞接收 API（自动化测试核心）

3.1 spk_receive(timeout_ms)

阻塞等待接收数据。

- 收到数据后若 50ms 内无新数据（空闲间隔），立即返回累计收到的所有字节
- 否则等待直到 timeout_ms 超时返回
- timeout_ms <= 0 表示无限等待（仍可被中断）

参数	返回值
<code>timeout_ms</code> : integer	<code>string</code> : 累计接收的字节; 无数据时为空串 ""

```
spk_send_hex("AA 55 01 02")
local data = spk_receive(1000)      -- 最多等 1 秒
if #data > 0 then
    print("收到 " .. #data .. " 字节")
end
```

3.2 spk_receive_until(match, timeout_ms)

阻塞等待，直到累计接收数据**包含子串** `match`，或超时。

参数	返回值
<code>match</code> : string, 期望子串 (字节匹配)	<code>ok</code> : boolean, 是否匹配到
<code>timeout_ms</code> : integer, <=0 无限等待	<code>data</code> : string, 累计收到的所有字节

```
local ok, data = spk_receive_until("READY", 5000)
if ok then
    print("设备就绪")
else
    print("5 秒内未收到 READY")
end
```

3.3 spk_expect(cmd, expect, timeout_ms)

测试核心助手，等价于 `spk_clear_rx()` + `spk_send(cmd)` + `spk_receive_until(expect, timeout_ms)`。

参数	返回值
<code>cmd</code> : string, 发送的命令	<code>ok</code> : boolean, 应答是否包含 <code>expect</code>
<code>expect</code> : string, 期望应答子串	<code>data</code> : string, 累计收到的应答字节
<code>timeout_ms</code> : integer	

发送失败（如串口未打开）时返回 `false`, "" 并输出警告。

```
-- 发送 AT 并期望 2 秒内收到 OK
local ok, resp = spk_expect("AT\r\n", "OK", 2000)
spk_report("AT 指令", ok, resp)

-- 循环压测 5 次
local pass = 0
for i = 1, 5 do
    if spk_expect("AT\r\n", "OK", 1000) then pass = pass + 1 end
    spk_sleep_ms(50)
end
spk_report(string.format("压测通过 %d/5", pass), pass == 5)
```

4. 波形 API

4.1 spk_wave(...)

将数据帧输出到波形控件（仅波形录制开启时生效）。

两种调用方式：

1. **数字可变参数**：`spk_wave(v0, v1, ..., vN)` —— 组装成一帧 `"{v0,v1,...}"`，一帧 = 一个采样时刻，第 *i* 个参数绘制到通道 *i*
2. **单字符串透传**：直接输出 CSV 帧格式字符串；不含 `{` 时自动包一层 `{}`

参数	返回值
数字可变参数 或 1 个字符串	无

```
-- 方式1：每帧一个采样时刻，2 个通道
spk_wave(3.14, 2.71)
spk_wave(3.15, 2.70)

-- 方式2：透传 CSV 帧（可一次塞多帧）
spk_wave("{1.0,2.0}{1.1,2.1}")
spk_wave("5.5,6.6")           -- 自动包装为 {5.5,6.6}
```

参数必须是有限数字（NaN/Inf 会报错）；参数个数为 0 会报错。

5. 脚本入口函数

5.1 spk_frame_parser(data) — 仅帧解析器模式

引擎收到串口数据时反复调用该函数，直到其返回 `nil`（无完整帧）。

参数	返回值
<code>data</code> ：string，本批到达的原始字节	<code>nil</code> ：buffer 中没有完整帧
	<code>string</code> ：一个完整协议帧字节（用于 hex 显示）；空串表示已自行 print 解析结果

```
-- 简单定长帧解析：帧头 0xAA 0x55 + 长度字节 + 数据
local buffer = ""

function frame_parser(data)
    buffer = buffer .. data

    while true do
        local head = string.find(buffer, "\xAA\x55", 1, true)
        if not head then
            buffer = ""           -- 丢弃无帧头数据
            return nil
        end
    end
end
```

```

end
if #buffer < head + 2 then
    return nil      -- 长度字节尚未收全
end

local len = string.byte(buffer, head + 2)
if #buffer < head + 2 + len then
    return nil      -- 帧尚未收全
end

local frame = string.sub(buffer, head, head + 2 + len)
buffer = string.sub(buffer, head + 2 + len + 1)

-- 解析并打印帧内容
print("收到帧, 长度=" .. len)

return frame      -- 返回原始帧用于 hex 显示
end
end

```

5.2 脚本执行器模式 — 顶层代码即测试流程

点击 Run 时, 引擎在后台线程从头执行整个脚本文件 (每次运行重建干净的 Lua 环境)。无需任何入口函数, 按顺序编写测试用例即可。

- 中断后输出 Automation stopped
- 运行错误 (非停止) 会弹出 Lua Error 对话框并输出 Automation aborted on error
- 正常结束输出 Automation finished: X passed, Y failed

6. 完整示例

6.1 自动化收发测试 (脚本执行器模式)

```

-- =====
-- AT 设备自动化测试示例
-- 用法: 打开串口 → 加载本脚本 → 选择 Script Executor 模式 → 点 Run
-- =====

-- 将字节串转为可读 hex 文本, 便于在报告中显示二进制应答
local function to_hex(s)
    return (s:gsub('.', function(c) return string.format('%02X ', string.byte(c))
end))
end

println("=== 自动化测试开始 ===")

if not spk_port_is_open() then
    spk_report("串口已打开", false, "请先打开串口")
    return
end

-- 清理 Rx 缓存
spk_clear_rx()

```

```

-- 测试1: 基础通信
local ok, resp = spk_expect("AT\r\n", "OK", 2000)
spk_report("AT 基础通信", ok, ok and "" or ("应答: " .. to_hex(resp)))

-- 测试2: 延时 + hex 帧 + 接收任意应答
spk_sleep_ms(100)
if spk_send_hex("AA 55 01 02") then
    local data = spk_receive(1000)
    spk_report("HEX 帧收发", #data > 0, to_hex(data))
else
    spk_report("HEX 帧收发", false, "发送失败")
end

-- 测试3: 等待设备就绪标志
local ok2 = spk_receive_until("READY", 3000)
spk_report("等待 READY", ok2)

-- 测试4: 循环压测
local total, pass = 5, 0
for i = 1, total do
    if spk_expect("AT\r\n", "OK", 1000) then pass = pass + 1 end
    spk_sleep_ms(50)
end
spk_report(string.format("循环压测 %d/%d", pass, total), pass == total)

println("=== 测试结束 ===")

```

6.2 帧解析 + 波形输出（帧解析器模式）

```

-- 将帧内字段解析后输出到波形控件（需开启波形录制）
-- =====
-- 模式1: 二进制帧解析
-- 帧格式: F1 10 A5 XX XX XX XX CC F5（可以是自定义帧格式）
--          XX XX XX XX 部分表示数据
-- =====

local buffer = ""

function spk_frame_parser(data)
    buffer = buffer .. data

    while true do
        local s = string.find(buffer, "\xF1\x10\xA5", 1, true)
        if not s then
            return nil
        end

        local e = string.find(buffer, "\xCC\xF5", s + 3, true)
        if not e then
            return nil
        end

        local frame = string.sub(buffer, s, e + 1)
        buffer = string.sub(buffer, e + 2)

        local v0 = string.unpack("<f", string.sub(frame, 4, 7))
    end
end

```

```
    spk_wave(v0)  -- 输出数据到波形控件

    --println(to_hex(frame)) -- 输出原始数据帧到UI
end
end
```

7. 附录

7.1 错误处理

场景	行为
脚本语法错误 / 运行时错误	弹出 Lua Error 对话框，自动化测试输出 <code>Automation aborted on error</code>
运行中点击 Stop	脚本以 <code>automation stopped</code> 中止，输出 <code>Automation stopped</code>
加载脚本未定义 <code>spk_frame_parser</code>	加载失败报错: <code>spk_frame_parser not defined in script</code>

- 接收队列由 UI 线程直连喂入，与引擎使能状态无关，`spk_receive` 阻塞期间数据照常进队
- 接收队列上限 1MB，超出丢弃最旧数据
- 空闲间隔判定固定为 50ms：`spk_receive` 在数据停顿 50ms 后返回

7.3 线程与阻塞

- 执行器模式的脚本在引擎自建后台线程运行（每次 Run 重建干净的 Lua 环境），阻塞等待期间 UI 与数据接收不受影响
- 帧解析器模式的脚本加载同样在后台线程完成，**UI 线程不执行任何脚本代码**
- 自动化运行期间 Load / Save / Reload / 模式切换等按钮被禁用
- 帧解析器模式中 `spk_frame_parser` 在数据处理线程运行，阻塞调用会暂停数据解析显示，请谨慎使用

7.4 API 速查表

函数	参数	返回值
<code>print(...)</code>	任意	—
<code>println(...)</code>	任意	—
<code>spk_get_version()</code>	—	string
<code>spk_send(data)</code>	string	bool
<code>spk_send_hex(hexStr)</code>	string	bool
<code>spk_sleep_ms(ms)</code>	integer	—
<code>spk_clear_rx()</code>	—	—
<code>spk_receive(timeout_ms)</code>	integer	string
<code>spk_receive_until(match, timeout_ms)</code>	string, integer	bool, string
<code>spk_expect(cmd, expect, timeout_ms)</code>	string, string, integer	bool, string
<code>spk_report(name, passed, detail)</code>	string, bool, string?	bool
<code>spk_port_is_open()</code>	—	bool
<code>spk_wave(...)</code>	数字可变参数 / 1 个字符串	—