

VisualHMI - 系统回调函数

本章节主要介绍常用的系统回调函数，常用函数如下所示：

- 1. 初始化函数：on_init()
- 2. 系统周期回调函数：on_run(screen)
- 3. 触摸回调函数：on_press(state,x,y)
- 4. 变量修改回调函数：on_update(slave,vtype,addr)
- 5. 切换画面回调函数：on_screen_change(screen)

系统函数一览表：

系统回调函数	描述	示例说明
on_init()	初始化	VisualHMI - 系统回调(点击跳转)
on_run(screen)	周期循环	VisualHMI - 系统回调(点击跳转)
on_press(state,x,y)	触摸回调	VisualHMI - 系统回调(点击跳转)
on_update(slave,vtype,addr)	变量修改	VisualHMI - 系统回调(点击跳转)
on_screen_change(screen)	切换画面	VisualHMI - 系统回调(点击跳转)
on_usb_inserted(driver)/on_usb_removed()	U盘回调	VisualHMI - 文件IO(点击跳转)
on_sd_inserted(dir)/on_sd_removed()	SD卡回调	VisualHMI - 文件IO(点击跳转)
on_copy_file_process(status,filesize,transfersize)	文件拷贝	VisualHMI - 文件IO(点击跳转)
on_draw(screen)	自绘回调	VisualHMI - 自绘(点击跳转)
on_timer(timer_id)	定时器	VisualHMI - 定时器(点击跳转)
on_parse_warning(id, text)	告警回调	VisualHMI - 告警说明(点击跳转)
on_uart_recv(ch,packet)	串口回调	VisualHMI - 自定义串口协议(点击跳转)

适用范围：VisualHMI - HMI&M系列&Dx系列

例程下载链接：[ViusalHMI - 系统回调函数\(点击下载\)](#)

1.on_init() 初始化

系统上电加载 LUA 脚本文件之后，立即调用此回调函数，通常用于执行初始化操作，仅执行一次

1.1. 加载其他lua文件

当项目的代码复杂，代码量也多，分多个Lua文件时，需要在初始化时候调用加载：

```
--初始化函数
function on_init()
    dofile('sysParam.lua')
end
```

1.2. 加载系统参数

系统参数，如SysCfg0触摸音、语言SysLang、音频音量大小SysSndVol、背光设置、用户密码、分期使用参数;

```
function on_init()
    set_uint16(VT_LW, 0x011B, (1<<1)) -- 选择需要加载的掩码，bit1,多语言
    --若有多多个系统参数掉电存储加载，则或运算多个位掩码
    --set_uint16(VT_LW, regAddr.sysParams1ct, (1 << 3)|(1 << 2)|(1 << 1)|(1 <<
0))
    set_uint16(VT_LW, regAddr.sysParamCtrl, 0x5502)
    update_system()
end

function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x119
        then
            set_uint16(VT_LW, 0x011B, (1<<1)) -- 选择需要加载的掩码，bit1,多语言
            set_uint16(VT_LW, 0x011A, 0x5501) -- 保存选中的系统参数
            update_system()
        end
    end
end
```

1.3. 初始化Flash参数

屏幕内部有64K用户掉电存储空间，RW0000~RW7FFFF（一个地址2个字节，共64K），屏幕flash新片，默认均为0xFF。实际应用中，通常设备第一次使用，需要初始化一些参数，可以通过判断Flash值，当不等于某个特定数值，执行相关初始化操作，如下所示：

```
--数据类型定义
VT_LW = 1    --变量地址
VT_RW = 2    --FLASH存储
VT_0x = 10   --线圈
VT_1x = 11   --输入点
VT_3x = 12   --输入寄存器
VT_4x = 13   --保持寄存器
```

```

function on_init()
    local fst = get_uint16(VT_RW, 0x7FFF)
    if fst ~= 0x55AA
    then
        set_uint16(VT_LW, regAddr.sysCfg0, 1)
        set_uint16(VT_LW, regAddr.sysLang, 0)
        set_uint16(VT_LW, regAddr.sysBLLevel, 100)
        set_uint16(VT_LW, regAddr.sysSndVol, 80)

        set_uint16(VT_LW, regAddr.sysParams1ct, (1 << 3)|(1 << 2)|(1 << 1)|(1 <<
0))
        set_uint16(VT_LW, regAddr.sysParamCtrl, 0x5501)
        update_system()
        set_uint16(VT_RW, 0x7FFF, 0x55AA)
    end
end

```

1.4. 初始化PLC/ 驱动板

屏幕上电，可以往PLC/ 驱动板发送指令，初始化参数，如下所示

```

--数据类型定义
VT_LW = 1    --变量地址
VT_RW = 2    --FLASH存储
VT_0x = 10   --线圈
VT_1x = 11   --输入点
VT_3x = 12   --输入寄存器
VT_4x = 13   --保持寄存器

function on_init()
    set_uint16(VT_4x,0x0000, 0x55AA)
    set_bit(VT_0x,0x1000, 0x0001)
end

```

2.on_run(screen) 周期回调函数

周期性回调函数，`on_run(screen)` 是 HMI 系统提供的**核心周期性任务调度入口**，由系统在每个主循环周期**自动调用**，用于执行需要定期更新的逻辑，如实时数据显示、状态轮询、后台监控等

参数名	类型	说明
<code>screen</code>	number	当前画面： 用于区分不同页面的逻辑分支

! 问题：

- 如HMI作为Modbus RTU Master 时，在 `on_run` 高频写入从机设备寄存器，造成通信负载过高，可能引发总线堵塞、设备响应延迟
- 不可在 `on_run` 中使用 `delay_ms()`、长循环、网络同步请求等
- **严禁在 `on_run` 内部调用 `set_run_cycle()`**（可能导致调度器死锁）

```

--数据类型定义
VT_LW = 1    --变量地址
VT_RW = 2    --FLASH存储
VT_0x = 10   --线圈
VT_1x = 11   --输入点
VT_3x = 12   --输入寄存器
VT_4x = 13   --保持寄存器

function on_run(screen)
    local start = get_uint16(VT_4x, 0x1000) --开机状态
    local dstVol = get_uint16(VT_4x, 0x1001) --设置电压
    local curVol = get_uint16(VT_4x, 0x1002) --当前电压
    local online = get_uint16(VT_4x, 0x01A3) --在线状态,位掩码
    if dstVol > curVol and start == 1 and (online&0x01 == 1)--设备开启后,且从机在线,当前电压不等于目标电压,设置当前电压值
    then
        set_uint16(VT_4x, 0x1003, 1) --设置频率
    end
end

function on_init()
    set_run_cycle(1000) --on_run(screen) 1s回调一次
end


```

3. on_press(state,x,y) 触摸回调函数

触摸回调函数常用于做自定义待机逻辑，搭配on_timer实现“屏幕空闲时间(无触摸)到达后，自动进入节能状态”

系统提供的**全局触摸事件回调函数**，用于捕获用户在屏幕上的**原始触摸操作**。该函数由系统在检测到触摸状态变化时自动调用，**每 100 毫秒最多触发一次**

参数名	类型	说明
state	number	触摸状态标识 <ul style="list-style-type: none"> 1：手指按下 2：长按触发（ 0：手指抬起 注意：不会重复发送 1，仅在按下瞬间触发一次
x	number	触摸点 X 坐标（像素）
y	number	触摸点 Y 坐标（像素）

 **注意：**此函数提供的是**底层坐标事件**，不依赖于具体控件，即使点击空白区域也会触发。

```

_ENCRYPT_=0    --LUA脚本加密

--数据类型定义

```

```
VT_LW = 1    --变量地址
VT_RW = 2    --FLASH存储
VT_0x = 10   --线圈
VT_1x = 11   --输入点
VT_3x = 12   --输入寄存器
VT_4x = 13   --保持寄存器
```

```
_EN_ON_UPDATA_API_ = 1
```

```
function on_init()
    _EN_ON_UPDATA_API_ = 0

    dofile('std.lua')
    std.init()

```

```
    _EN_ON_UPDATA_API_ = 1
```

```
end
```

```
--触摸回调
```

```
function on_press(state,x,y)
    _EN_ON_UPDATA_API_ = 0

```

```
    if state == 0
    then
        std.init()
    end

```

```
    _EN_ON_UPDATA_API_ = 1
```

```
end
```

```
--修改变量
```

```
function on_update(slave,vtype,addr)
    if _EN_ON_UPDATA_API_ == 0
    then
        return
    end

```

```
    std.set(slave,vtype,addr)

```

```
end
```

```
--定时器回调
```

```
function on_timer(timer_id)
    std.notify(timer_id)

```

```
end
```

```
--std.lua
```

```
std = {}
std.cnt = 0
```

```
--待机初始化
```

```
function std.init()
    local fst = get_uint16(VT_RW, 0x7FFF)
    if fst ~= 0x55AA

```

```

then
    set_uint16(VT_RW, 0x7FFF, 0x55AA)
    set_uint16(VT_RW, 0x1011, 3)--待机时间
    set_uint16(VT_RW, 0x1012, 3)--屏保时间
    set_uint16(VT_RW, 0x1013, 100)--激活亮度
end

local screenSaveTimer = get_uint16(VT_RW, 0x1011)*60
local standbyTimer = get_uint16(VT_RW, 0x1012)*60
local bk = get_uint16(VT_RW, 0x1013)

set_uint16(VT_LW, 0x0121, bk)

std.cnt = 0
stop_timer(_01_standby)
if lastAlarmStatus == 1
then
    return
end
start_timer(_01_standby, 1000 ,1 ,(screenSaveTimer + standbyTimer))
end

--待机通知，定时器里调用
function std.notify(timer_id)
    if _01_standby == timer_id
    then
        std.cnt = std.cnt + 1

        local screenSaveTimer = get_uint16(VT_RW, 0x1011)*60
        local standbyTimer = get_uint16(VT_RW, 0x1012)*60

        if std.cnt == screenSaveTimer --屏保时间
        then
            set_screen(__page.home)

        elseif std.cnt == (screenSaveTimer + standbyTimer)--待机时间
        then
            set_uint16(VT_LW, 0x0121, 0)
            set_screen(__page.std)
        end
    end
end

--待机时间设置
function std.set(slave,vtype,addr)
    if vtype == VT_RW
    then
        if addr >= 0x1011 and addr <= 0x1014
        then
            std.init()
        end
    end
end
end

```

4. on_update(slave,vtype,addr)

HMI 系统提供的**变量变更事件回调函数**，当**用户操作或脚本逻辑**修改了受监控的寄存器/变量值时，系统自动触发该回调。该机制实现了“**数据驱动**”的编程模型：**无需轮询，仅在数据变化时响应**，极大提升系统效率与实时性。

- slave：站号索引，0开始
- vtype：变量类型，生成main.lua，自动定义变量的数据类型
- addr：变量地址

参数名	类型	说明
slave	number	站号索引，0开始
vtype	number	变量类型，生成main.lua，根据协议自动生成数据类型，如 Modbus RTU 的 VT_0x（线圈）、FX3U的 VT_M（内部继电器）等
addr	number	寄存器地址

✅ 会触发 on_update 的场景

场景	说明
用户界面操作	点击按钮、滑动条、输入框等控件修改绑定寄存器
其他回调on_xx调用 set_xxx()	在 on_init、on_run、on_draw、on_screen_change 等回调中调用写入函数

❌ 不会触发 on_update 的场景

场景	说明
串口/Modbus 主站写入	外部设备通过通信协议直接写入从机寄存器（如 PLC 修改寄存器值）
on_update 内部调用 set_xxx()	防止无限递归，系统自动屏蔽嵌套触发

💡 示例

如下所示，已Modbus 协议为例，当其他回调函数设置寄存器，均不想触发on_update回调，在系统回调函数里面的第一行、最后一行限制，可以定义全局变量如下处理，：

```
ENCRYPT_=0      --LUA脚本加密

EN_ON_UPDATE_API_CB = 1 --全局变量，其他回调函数设置寄存器时，为1执行，为0直接退出

--数据类型定义
VT_LW = 1      --变量地址
VT_RW = 2      --FLASH存储
VT_0x = 10     --线圈
VT_1x = 11     --输入点
VT_3x = 12     --输入寄存器
```

```
VT_4x = 13    --保持寄存器
```

```
function on_init()
```

```
    EN_ON_UPDATE_API_CB = 0
```

```
    --user code
```

```
    set_uint16(VT_LW, 0x1000, 0x55AA)
```

```
    EN_ON_UPDATE_API_CB = 1
```

```
end
```

```
function on_run(screen)
```

```
    EN_ON_UPDATE_API_CB = 0
```

```
    --user code
```

```
    EN_ON_UPDATE_API_CB = 1
```

```
end
```

```
function on_update(slave, vtype, addr)
```

```
    if EN_ON_UPDATE_API_CB == 0
```

```
    then
```

```
        return
```

```
    end
```

```
    if slave == 0
```

```
    then
```

```
        if vtype == VT_4x --保持寄存器
```

```
        then
```

```
            if addr == 0x1000
```

```
            then
```

```
                local val = get_uint16(VT_4x, addr)
```

```
                if val == 1
```

```
                then
```

```
                    set_uint16(VT_4x, 0x0001, val)
```

```
                end
```

```
            end
```

```
        elseif vtype == VT_0x --线圈
```

```
        then
```

```
            if addr == 0x0000
```

```
            then
```

```
                local val = get_uint16(VT_0x, addr)
```

```
                if val == 1
```

```
                then
```

```
                    set_uint16(VT_0x, 0x0001, val)
```

```
                end
```

```
            end
```

```
        end
```

```
    end
```

```
    if vtype == VT_LW --内部寄存器
```

```
    then
```

```
        if addr == 0x1000
```

```
        then
```

```
            local val = get_uint16(VT_LW, addr)
```

```
            if 0x55AA == val
```

```
            then
```



```

        set_uint16(VT_4x, 0x0000, val)
    end
end
end
end

function on_draw(screen_id, control_id)
    EN_ON_UPDATE_API_CB = 0

    --user code

    EN_ON_UPDATE_API_CB = 1
end

function on_screen_change(screen)
    EN_ON_UPDATE_API_CB = 0

    --user code
    if screen == 1
    then
        set_bit(VT_0x, 0x0000, 1)
    end

    EN_ON_UPDATE_API_CB = 1
end

```

5. on_screen_change(screen)

画面切换事件回调函数，在每次画面切换完成之后由系统自动调用。该函数用于执行与画面切换相关的初始化、状态同步、资源加载或业务逻辑处理。

参数名	类型	描述
screen	number	当前切换到的目标画面ID