

VisualHMI - USB HID扫码枪(定制)

该 HMI 设备配备 USB Host 接口，并通过**定制固件**实现对 USB HID（Human Interface Device）类条码扫描器的支持。当扫描器以标准 USB HID 键盘协议上报按键事件时，HMI 的 USB Host 驱动层捕获原始 HID 报文中的 **Usage ID（即键盘扫描码）**，并将其传递至 Lua 应用层回调函数 `on_key(key)`。

在 HMI（人机界面）屏幕上集成 **USB HID 条码扫描器支持**，具有显著用户体验优势，尤其适用于工业自动化、仓储物流、生产管理等场景。

使用范围：VisualHMI - HMI&M&DX系列 USB接口/定制固件

应用下载：[VisualHMI - USB HID扫码枪\(定制\)\(点击下载\)](#)

1.API说明

1.1.on_key(key)

接收USB HID 事件 回调函数

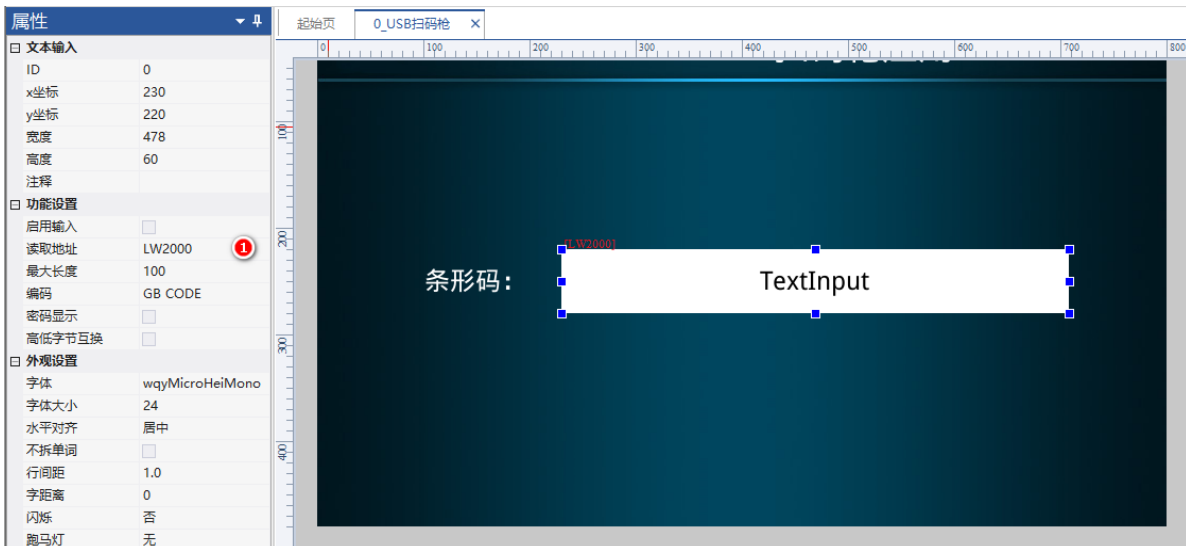
参数 `key` 为一个 16 位无符号整数，其编码格式定义如下：

- **低 8 位 (bit 7-0)**：表示 USB HID 键盘的 **Usage ID（扫描码）**；
- **高 8 位 (bit 15-8)**：表示 **Shift 修饰键状态**，非零值（通常为 `0x01`）表示 Shift 被按下，用于区分大小写或符号变体；
- **条码结束标志**：当低 8 位等于 `0x28`（对应 HID Usage ID 为 **Keyboard Return (Enter)**）时，表示条码输入结束，应用应在此时提交完整字符串并清空缓冲区。

2.应用

2.1.工程配置

在 HMI 工程中，演示 USB HID 条码扫描枪功能，配置一个文本显示控件，其关联地址为 **LW2000**，用于实时呈现扫码枪所采集的条码内容。



2.2.编辑Lua

该脚本实现了 **HMI 系统对 USB HID 条码扫描枪的解析与条码内容捕获功能**。通过监听底层 `on_key(key)` 回调，将扫码枪以键盘模拟方式输入的原始 HID 扫描码转换为可读 ASCII 字符串，并在检测到回车 (Enter) 结束符后，将完整条码写入 HMI 的指定寄存器地址 (`LW2000`)，供画面控件显示或业务逻辑使用。

```
_barCode = ''

usbScanGun = {

    [0x04] = {'a', 'A'}, --abcdefghijklmnopqrstuvwxyz
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
    [0x05] = {'b', 'B'},
    [0x06] = {'c', 'C'},
    [0x07] = {'d', 'D'},
    [0x08] = {'e', 'E'},
    [0x09] = {'f', 'F'},
    [0x0A] = {'g', 'G'},
    [0x0B] = {'h', 'H'},
    [0x0C] = {'i', 'I'},
    [0x0D] = {'j', 'J'},
    [0x0E] = {'k', 'K'},
    [0x0F] = {'l', 'L'},
    [0x10] = {'m', 'M'},
    [0x11] = {'n', 'N'},
    [0x12] = {'o', 'O'},
    [0x13] = {'p', 'P'},
    [0x14] = {'q', 'Q'},
    [0x15] = {'r', 'R'},
    [0x16] = {'s', 'S'},
    [0x17] = {'t', 'T'},
    [0x18] = {'u', 'U'},
    [0x19] = {'v', 'V'},
    [0x1A] = {'w', 'W'},
    [0x1B] = {'x', 'X'},
    [0x1C] = {'y', 'Y'},
    [0x1D] = {'z', 'Z'},

    [0x1E] = {'1', '!'}, --1234567890 !@#$%^&*()
    [0x1F] = {'2', '@'},
    [0x20] = {'3', '#'},
    [0x21] = {'4', '$'},
    [0x22] = {'5', '%'},
    [0x23] = {'6', '^'},
    [0x24] = {'7', '&'},
    [0x25] = {'8', '*'},
    [0x26] = {'9', '('},
    [0x27] = {'0', ')'},

    [0x2C] = {' ', ' '}, -- -=[]\;'\` ,./ _+{}|:~<>?
    [0x2D] = {'-', '_'},
    [0x2E] = {'=', '+'},
```

```

[0x2F] = {'[', '{'},
[0x30] = {']', '}'},
[0x31] = {'\\', '|'},
[0x33] = {';', ':'},
[0x34] = {'\\', '"'},
[0x35] = {'`', '~'},
[0x36] = {'<', '<'},
[0x37] = {'>', '>'},
[0x38] = {'/', '?'},

}

function on_key(key)

    local mode = (key >> 8)&0xFF

    key = key & 0xFF
    if key ~= 0x28
    then
        if mode == 0
        then
            if usbScanGun[key][1] ~= nil
            then
                _barCode = _barCode..usbScanGun[key][1]
            end
        else
            if usbScanGun[key][2] ~= nil
            then
                _barCode = _barCode..usbScanGun[key][2]
            end
        end
    else
        set_string(VT_LW, 0x2000, _barCode)
        print('_barCode = '.._barCode)
        _barCode = ''
    end
end
end

```

3.扩展

本例程仅为 USB HID 条码扫描功能的基础演示，旨在实现扫码内容的实时显示。在实际应用场景中，为提升安全性与用户体验，建议结合具体业务需求引入以下工程化增强措施：

3.1.超时重置机制

当连续输入间隔超过预设阈值（如 3 秒）时，自动清空条码缓存，防止因扫描中断导致的残余数据污染后续识别结果。如扫码过程中断（如遮挡、设备移开），导致 `_barCode` 缓冲区残留不完整数据，影响下一次识别。

🔧 解决方案

使用 HMI 的 **周期性定时器**（如 100ms 宏）监控最后输入时间，超时自动清空。

```

local _barCode = ''

-- 在 on_key 中更新时间戳
function on_key(key)
    -- ... 原有逻辑 ...
    if key & 0xFF ~= 0x28 then start_timer(0, 3000, 1, 1) end
    if key & 0xFF == 0x28 then stop_timer(0) end
end

function on_timer(timer_id)
    if timer_id == 0 then _barCode = '' end
end

```

3.2.退格编辑支持

少数高端扫码枪支持“人工修正”模式（如扫描后按 ← 删除错误字符），扩展 HID 按键事件处理逻辑，识别标准 Backspace 扫描码（Usage ID 0x2A），实现字符级回退操作，以兼容支持人工修正的智能扫码设备。

✦ 解决方案

扩展 `on_key`，识别 Backspace Usage ID（0x2A）并回退字符串。

```

function on_key(key)

    local mode = (key >> 8) & 0xFF
    key = key & 0xFF

    if key ~= 0x28
    then
        if mode == 0
        then
            if usbScanGun[key][1] ~= nil
            then
                _barCode = _barCode..usbScanGun[key][1]
            end
        else
            if usbScanGun[key][2] ~= nil
            then
                _barCode = _barCode..usbScanGun[key][2]
            end
        end
    end

    elseif usage_id == 0x2A then -- Backspace (0x2A = Keyboard
Delete/Backspace)
        if #_barCode > 0 then
            _barCode = _barCode:sub(1, -2) -- 删除最后一个字符
        end
    else
        set_string(VT_LW, 0x2000, _barCode)
        print('_barCode = '.._barCode)
        _barCode = ''
    end
end
end

```

3.3.输入长度边界校验

对累积条码字符串实施最大长度限制（如 ≤ 100 字符），避免因超长条码引发内存溢出或 HMI 寄存器越界写入，确保系统资源安全与数据完整性。

🔧 解决方案

在拼接前检查长度，限制最大字符数（如 100）。

```
if mode == 0
then
  if usbScanGun[key][1] ~= nil and string.len(_barCode) < 100
  then
    _barCode = _barCode..usbScanGun[key][1]
  end
else
  if usbScanGun[key][2] ~= nil and string.len(_barCode) < 100
  then
    _barCode = _barCode..usbScanGun[key][2]
  end
end
end
```