TCL WIFI SDK 使用说明文档

2016-04-28

文档修改记录

修改日期	版本/状态	作者	修改章节	修改描述
2016-5-30	1.0	陈艳春	3.4.协议消息及相关功能	1. Tcl_protocol_statusReport: 增加表示局域网和广域网的标志； 2. 完善 ota 升级接口函数 3. 4 章节例子完善。
2016-11-15	1.1	邝超群	3.6 空调参数列表 3.4.6 故障上报 API 完善	1、增加了原来遗漏的室内温度和 3D 送风选项，去掉不用的人感风向开关 2、故障上报 API 根据需求增加了 flag 参数

目录

1 概述

本文主要将 tcl 家电入网配置，通信协议等进行封装，屏蔽掉协议细节从而让使用者能快速的开发符合 tcl 云端规范的 wifi 应用。

本 Wifi sdk 包括：一个 lib 库和一个头文件 wifi_sdk_if.h.下面具体描述相关接口。

2 数据结构说明

2.1 T_cfgRetInfo

定义：

```
typedef struct
{
   char ssid[SSID_MAX_LEN]; /*  家庭路由器的 ssid */
   char pwd[PWD_MAX_LEN];/*  家庭路由器的 pwd */
}T_cfgRetInfo;
```

说明：

入网配置成功后返回路由器的 ssid 和密码给用户。SSID_MAX_LEN 和 PWD_MAX_LEN 定义见 wifi_sdk_if.h.

2.2 T_localInfo

定义：

```
typedef struct
{
    char ip[IP_MAX_LEN]; //表示从路由器获得的本机 ip，格式：x.x.x.x
    char mac[MAC_MAX_LEN]; //本机的 mac 地址，格式：X:X:X:X:X:X
    char version[ARG_NAME_LEN]; //固件版本号
    dataHandler handler; //局域网控制命令 handler,
    sysMsgHandler sysHandler;//需要嵌进局域网模块执行的特定流程；
                                如果无，填 NULL.
}T_localInfo;
```

说明：

局域网功能启动所需的参数结构。例子见 4 章节。

2.3 T_xmppInfo

定义：

```
typedef struct
{
    char userid[32];//首次时为 0x0，以后由服务器分配。
    char userPwd[64];//首次时为 0x0，以后由服务器分配。
    char barcode[32]; //首次时为 0x0，以后由服务器分配
    char mac[MAC_MAX_LEN]; //本机 mac 地址，格式同上。
    char routerMac[MAC_MAX_LEN];//所连路由器 bssid，格式同 mac
    char routerSsid[SSID_MAX_LEN];//所连路由器的 ssid。
    int localPort; //本地 socket 连接使用的端口。
    dataHandler handler; //同局域网描述
    resetAckHandler resetHandler;//复位设备关系结果处理回调函数
    sysMsgHandler sysHandler; //同局域网描述。
}T_xmppInfo;
```

说明：

广域网功能启动所需的参数结构。例子见 4 章节。

上述由服务器分配的参数 userid， userpwd，barcode 等在调用广域网接口时，首次填充 0x0，当从云端获取到参数值后，用户需自行保存，再次调用广域网功能时需由用户传入。

2.4 T_ParamItem

定义：

```
typedef struct
{
    char param[ARG_NAME_LEN]; //空调参数名
    char value[ARG_VALUE_LEN]; //空调参数值
}T_ParamItem;
```

说明：

无。

2.5 T_upgInfo

定义：

```
typedef struct
{
    char url[64]; //固件升级的 url
    char fileName[64];//固件名称
```

```
    int is_update;//是否需要升级标志；1：需要升级；0：不需要升级
}T_upgInfo;
```

说明：云端返回的升级信息。

2.6 T_SDK_info

定义：

```
typedef struct
{
    char did[32+1];          /* 设备唯一标识，如无可以不填 */
    char joinid[24];         /* 接入平台 ID:    compdevid */
    char joinkey[36];        /* 接入平台 key:   compdevkey */
    char brand[12];          /* 品牌: TCL */
    char company[16+1];      /* APP 或固件开发公司:   TCLYJY */
    char category[10+1];     /* 品类: AC */
    char childctp[10+1];     /* 小品类：titanium */
    char deviceType[32+1];/* 设备具体型号：DEVICE_ENCODE */
    char firmware[4];        /* 固件开发商编号（01: TCL 家电集团   02:古北）*/
    char platform[4];        /* 芯片型号（01：高通 4004，02：Realtek8711，
                                03：乐鑫 8266）*/
    char devResource[32]; /* 设备资源名:
                             空调：AC-linux-zx01-1
                             冰箱：FR-linux-zx01-1 */
}T_SDK_info;
```

说明:

初始化配置 sdk 的信息结构。开发过程中，此结构信息需要研发单位与云端沟通确定。

3 API 接口说明

3.1 配置功能

3.1.1 tcl_smartcfg_init

原型：int tcl_smartcfg_init(func_softAP startSoftAp);
功能：配置模块初始化；
参数：startSoftAp 是 soft ap 切换函数指针；取值说明如下：
NULL：
表示外部已经完成 softap 切换，此函数直接启动配置服务，内部没有切换 wifi 热点操作。
非空时：
表示配置服务启动时，先执行此函数切换到 softap 成功，然后才执行配置服务；

返回值：0：成功；-1 失败；
说明：
1. 函数指针定义为：typedef int (*func_softAP)();
2. 设备做 softap 时，如果采用 TCL 超级 APP 进行配置，则约定配置如下：
SSID: 前缀 + 品类 + 匹配符 + MAC 地址后 3 字节；
例如：空调 MAC：01:02:03:04:05:06
则空调 ssid 为：**tcl_AC_t*ap_040506**
密码：“**12345678**”

3.1.2 tcl_smartcfg_process

原型：int tcl_smartcfg_process(T_cfgRetInfo *ret_info);
功能：执行配置流程，返回配置结果。
参数：ret_info,记录返回的配置数据，包括：ssid，密码。
返回值：0 成功，-1 失败；
说明：
调用此函数将会阻塞调用线程，当配置结束时返回配置结果，当返回成功时，ret_info 参数将包含配置的 ssid 和密码。

3.2 局域网功能

3.2.1 tcl_localSvr_init

原型：void tcl_localSvr_init(T_localInfo *localinfo);
功能：初始化局域网模块。
参数：localinfo,局域网模块启动需要的参数信息，见上面结构定义。
返回值：无；
说明：无。

3.2.2 tcl_localSvr_process

原型：int tcl_localSvr_process();
功能：执行局域网服务。
参数：无。
返回值：错误码 RET_ERR 等见 sdk 头文件定义。
说明：
调用此函数将会阻塞调用线程，当服务异常时会返回错误码。

3.3 广域网功能

3.3.1 tcl_xmppClient_init

原型：void tcl_xmppClient_init(T_xmppInfo *xmppInfo);
功能： 初始化广域网服务模块函数。
参数： 广域网模块启动需要的参数信息，见上面结构定义描述。
返回值：无。
说明：无。

3.3.2 tcl_xmppclient_connectSvr

原型：int tcl_xmppclient_connectSvr(T_xmppInfo *xmppInfo);
功能：执行广域网服务。
参数： xmppInfo 见上面结构描述。
返回值：错误码 RET_ERR 等见 sdk 头文件定义。

说明：无。

3.3.3 tcl_xmppclient_process

原型：int tcl_xmppclient_process();
功能：执行广域网服务。
参数：无。
返回值：错误码 RET_ERR 等见 sdk 头文件定义。
说明：
调用此函数将会阻塞调用线程，当服务异常时会返回错误码。

3.4 协议消息及相关功能

3.4.1 tcl_protocol_getItemFromList

原型：char* tcl_protocol_getItemFromList(char *paramList, T_ParamItem *item, int cmdtype)
功能： 从参数列表 paramList 中取出一个参数记录。
参数：
paramList： 接收到的参数列表指针。
Item： 从参数列表中取出的参数记录指针。
Cmdtype：0：表示当前是 set 命令；1：表示当前是 get 命令。
返回值：
返回下一个待处理的参数指针，如果为 NULL，表示本次是最后一个参数，参数列表已经处理完毕。
说明：
Tcl 空调操控命令分两类：
Set 类：表示修改空调属性值。
Get 类：表示查询空调属性值。

3.4.2 tcl_protocol_addItemToPacket

原型：int tcl_protocol_addItemToPacket(char *packet, int len, T_ParamItem *item);
功能：将参数记录加入协议报文中。
参数：
Packet： 发送数据缓冲区指针；
Len： 缓冲区大小。
Item： 待加入的记录。
返回值：返回缓冲区已使用的长度。
说明：无。

3.4.3 tcl_protocol_sendSetAck

原型： **int tcl_protocol_sendSetAck(int result, void *header);**
功能： 发送控制命令应答。
参数： **result**：1：执行成功。-1：执行失败
Header：由回调函数传入，用户透传即可
返回值： 失败返回-1，否则返回发送数据报文总长度。
说明：无。

3.4.4 tcl_protocol_sendGetAck

原型：int tcl_protocol_sendGetAck(char *data, void *header);
功能：发送 get 命令的应答。
参数：
data：要发送的参数表；
Header： 由回调函数传入，用户透传即可。
返回值： 失败返回-1，否则返回发送数据报文总长度。
说明：无。

3.4.5 tcl_protocol_statusReport

原型：int tcl_protocol_statusReport(char *pdata, int len, int flag);
功能：向云端上报设备状态参数。
参数：
Pdata： 要上报的参数列表。
Len：参数列表的长度字节数。
Flag: 0:表示向云端发送状态；1：表示向局域网广播发送状态；

返回值：成功返回 0,失败返回-1。
说明：
上述参数列表由 tcl_protocol_addItemToPacke 生成。

3.4.6 tcl_protocol_reportErrCode

原型：int tcl_protocol_reportErrCode(char *pdata, int len,int flag);
功能： 上报故障信息。
参数： pdata：要上报的故障内容。
Len： 故障内容长度。
Flag:0:表示向云端发送故障信息；1：表示向局域网广播发送故障信息；
返回值：失败返回-1，否则返回发送数据报文总长度。
说明：无。

3.4.7 tcl_protocol_sendRest

原型：int tcl_protocol_sendRest();
功能： 发送复位信息到云端。
参数： 无。
返回值：失败返回-1，否则返回发送数据报文总长度。
说明：当用户按下复位键时使用。

3.4.8 tcl_protocol_setResetEvtFlag

原型：void tcl_protocol_setResetEvtFlag(int resetFlag);
功能：设置复位结果。
参数：resetFlag：0：表示云端处理复位信息成功。1：云端处理失败，wifi 需继续尝试。
返回值：无。
说明：无。

3.4.9 tcl_protocol_disconnect

原型：void tcl_protocol_disconnect();
功能：通知云端设备主动断开网络。
参数：无。
返回值：无。
说明：无。

3.4.10 tcl_protocol_versionQuery

原型：int tcl_protocol_versionQuery(T_upgInfo *upgInfo);
功能：向云端查询版本信息。
参数：upgInfo: 表示云端返回的查询结果。
详细描述见 2 数据结构定义；
返回值：-1： 失败； 1: 成功。
说明：无。

3.4.11 tcl_protocol_upgradCmd

原型：int tcl_protocol_upgradCmd(char *url, char *file, upgradeCallBack upg_func);
功能：当需要升级固件时，调用此函数完成升级。
参数：

Url： 固件存放的 url；

File：文件名；

Upg_func: 回调函数，用于处理收到的每一个数据包,说明如下；

- **upgradeCallBack 定义如下：**

  typedef int (*upgradeCallBack)(char *data, int len,int fileSize);

- 参数说明：

  Data：表示接收到的数据报文首地址；

  Len： 数据报文的长度；

  fileSize：字节为单位，固件文件的大小。

- 函数返回值定义

```
typedef enum
{
    UPG_ERR = -1, //数据报文处理错误；
    UPG_FINISHED,//当固件升级完成时返回；
    UPG_CONTINUE//继续接收处理后续报文；
}E_UPG_RET;
```

返回值：-1： 失败； 1: 成功。

说明：升级例子见 4。

3.5 空调 SDK 初始化

3.5.1 tcl_sdk_init

原型：void tcl_sdk_init(T_SDK_info *sdk_info);

功能： 初始化 sdk 配置参数。

参数： sdk 运行需要的参数信息，见上面结构定义描述。

返回值：无。

说明： 在使用局域网功能，广域网功能前必须先调用此函数完成 sdk 的配置。

3.6 空调操控参数列表

使用 tcl app 可以控制和获取如下空调参数信息：

参数名	说明	取值
turnOn	开关。	0： 关 1： 开
setTemp	设定温度	数字
windSpd	风速	0： 自动 1： 高速

		2： 中速 3： 低速
directH	水平摆风	0： 关 1： 开
directV	垂直摆风	0： 关 1： 开
baseMode	运行模式	0： 自感 1： 制冷 2： 除湿 3： 送风 4： 制热
optECO	经济	0： 关 1： 开
optHealthy	建康	0： 关 1： 开
optAntiM	防霉	0： 关 1： 开
optSuper	强力	0： 关 1： 开
optDisplay	数显	0： 关 1： 开
optHeat	电辅热	0： 关 1： 开
optSleepMd	睡眠模式	1： 睡眠模式 1 2： 睡眠模式 2 3： 睡眠模式 3 4： 睡眠模式 4 0： 禁用睡眠功能
sleeperCurve	睡眠曲线。十个温度值，用逗号隔开。表示：第一到第十小时的对应温度值，形如：x,x,x,x,x,x,x,x,x,x	x 范围：[16,31]
humidityEn	湿度开关	0： 关 1： 开
cleanEn	清洁度开关	0： 关 1： 开
infrDirect	人感风向	0： 关闭 1： 避人吹 2： 迎人吹
degreeH	设置温度加减 0.5°标志	1:+0.5°C 0: -0.5°C
beepEn	蜂鸣器开关	0： 关 1： 开
cntTmrOn	倒计时定时开，格式：hh:mm	//例如：10:30 表示 10 小时 30

		分后开 //当 hh:mm 为 00:00 时表示取消定时
cntTmrOff	倒计时定时关	同倒计时定时开。
inTemp	室内温度	数字
optSolidWd	3D 送风	0：关 1：开
其他可以查询的参数	说明	取值

4 sdk 使用举例

4.1 配置功能

```
static void smartCfgTask(void *param)
{
    T_MSG msg;
    T_cfgRetInfo info;

    memset(&msg, 0, sizeof(msg));
    memset(&info, 0, sizeof(info));

    tcl_smartcfg_init(smartcfg_startSoftAP);
    if(tcl_smartcfg_process(&info) == RET_OK)
    {
        tcl_printf("info.ssid = %s, info.pwd = %s\n", info.ssid, info.pwd);
    }
}
```

4.2 局域网功能

```
Void localTask()
{
   T_localInfo localinfo;

    tcl_wifi_get_ip(localinfo.ip, sizeof(localinfo.ip));
    tcl_wifi_get_mac(localinfo.mac, sizeof(localinfo.mac));
    localinfo.sysHandler = NULL; //如果无则填 NULL
    localinfo.handler = udpDataHandler;
```

```
    tcl_localSvr_init(&localinfo);
    for(;;)
    {
        tcl_printf("udp server start \n");
        if(tcl_localSvr_process() == RET_ERR)
        {
            tcl_printf("localSvr exit for error.\n");
        }
        os_taskDelay(5000);
    }
}
```

其中用户回调函数定义如下：

```
static int udpDataHandler(char *command, char *paramList, int listLen, void *header)
{
    T_ParamItem item;
    T_Item item1;
    char *pdata = NULL;
    int ret = RET_OK;
    int idx = 0;
    char buffer[256];

    tcl_printf("--command = %s, paramList = %s, listLen = %d\n", command, paramList, listLen);

    if(strcmp(command, "set") == 0)
    {
        //parser paramlist and call uart module
        pdata = paramList;
        while((pdata = tcl_protocol_getItemFromList(pdata, &item, 0)) != NULL)
        {
            tcl_printf("%s = %s\n", item.param, item.value);

        }
        tcl_printf("%s = %s\n", item.param, item.value);

        //send set message to uart.
        protocol_sendSetMsgToUart(buffer, vallen);
        //send ack.
        tcl_protocol_sendSetAck(ret, header);

    }
    else if(strcmp(command, "get") == 0)
    {
        len = 1500;
        pkt = (char *)malloc(len);
```

```
        if(pkt == NULL)
            return RET_ERR;
        memset(pkt, 0, len);
        pdata = paramList;

        if(strstr(pdata, "all") != 0)
        {
            //get all status from uart.
        }
        else
        {
            while((pdata = tcl_protocol_getItemFromList(pdata, &item, 1)) != NULL)
            {
                tcl_printf("%s = %s\n", item.param, item.value);
                //get    item value from uart.
                //add item to packet.
                offset += tcl_protocol_addItemToPacket(pkt+offset, len-offset, &item);

            }
            tcl_printf("%s = %s\n", item.param, item.value);
            //add item to packet.
            offset += tcl_protocol_addItemToPacket(pkt+offset, len-offset, &item);
        }
        //send packet
        tcl_protocol_sendGetAck(pkt,header);
        free(pkt);
    }
    else
    {
        tcl_printf("invalid command.(%s)\n", command);
    }

    return RET_OK;
}
```

4.3 广域网功能

用法类似局域网操控，回调函数定义参见局域网功能。

```
Void xmppTask()
{
    T_xmppInfo xmppInfo;
    xmppInfo.sysHandler = sysHandler;
    xmppInfo.handler = xmppDataHandler;
```

```
xmppInfo.resetHandler = resetHandler;
……
tcl_xmppClient_init(&xmppInfo);
While(1)
{
        T_xmppInfo Info;
        memset(&Info, 0, sizeof(Info));
        if(tcl_xmppclient_connectSvr(&Info) == ERR_CONNECT_OK)
        {
            //save user & pwd to flash.
            Printf("userid = %s, userPwd = %s\n", Info.userid, Info.userPwd);
            //判断是否需要发送复位消息
            //从 flash 读取 resetFlag
            If（resetFlag ==1）
                tcl_protocol_sendRest();


            //进入 xmpp message loop
            ret = tcl_xmppclient_process();
            if((ret == ERR_TCP_SOCK) || (ret == ERR_XMPP_CLOSE))
            {
                msg.code = MSG_CENT_XMPP_DISCONNECT;
                sendMsgToCent(&msg);
            }
      }
}
```

4.4 复位流程

用伪代码描述如下：

```
{
    If(复位事件发生)
    {
        If( tcl_protocol_sendRest() != -1 ) //发送 reset 消息到云端成功
        {
            启动复位超时定时器；
            设置按键复位事件标志；
        }
        Else
        {
            ResetEvtFlag = 1；//此标志由用户定义，用于记录复位消息是否成功。
            将 ResetEvtFlag 写入 flash；
            tcl_protocol_setResetEvtFlag(ResetEvFlag);//更新 udp 广播中的 reset 标
    志；
```

```
                清空网络配置信息；
                切换到配置模式；
            }
        }
}
```

说明：

1. 当发送复位消息成功后，用户需要定义复位消息回调函数和定时器超时处理函数；

   A 定时器超时处理逻辑：

```
{
    ResetEvtFlag = 1；//此标志由用户定义，用于记录复位消息是否成功。
    将 ResetEvtFlag 写入 flash；
    tcl_protocol_setResetEvtFlag(ResetEvFlag);//更新 udp 广播中的 reset 标志；
    清空网络配置信息；
    切换到配置模式；
}
```

   B 复位消息回调函数：

```
{
    If(云端处理成功)
    {
        ResetEvtFlag = 0；//此标志由用户定义，用于记录复位消息是否成功。
    }
    Else
    {
        ResetEvtFlag = 1；//此标志由用户定义，用于记录复位消息是否成功。
    }
    tcl_protocol_setResetEvtFlag(ResetEvFlag);//更新 udp 广播中的 reset 标志；
    If(设置了按键复位标志)
    {
        将 ResetEvtFlag 写入 flash；
        清空网络配置信息；
        切换到配置模式；
    }
}
```

2. 当发送复位消息是失败后，用户必须在网络配置成功后再次尝试发送复位消息。代码参考 4.3 中蓝色部分。

4.5 升级功能

流程如下：

其中，tcl_protocol_upgradCmd 中已经有重试机制。

```
{
    if(tcl_protocol_versionQuery(&upginfo) != -1)
    {
        printf("url=%s,filename=%s, is_update = %d\n", upginfo.url,
                                                upginfo.fileName,upginfo.is_update);
        If(upginfo.is_update == 1)
        {
            Ret =tcl_protocol_upgradCmd(upginfo.url, upginfo.fileName,upgcb);
            If(ret ==RET_OK)
            {
                //升级成功；
            }
        }
   }
}
```

其中，upgcb 函数定义如下：

```
int upgcb(char *data, int len, int fileSize)
{
    Static int recvSize =0;
    recvSize += len；
    //wirte data to flash.

    If（处理出错）
        Return UPG_ERR；
    If(recvSize == fileSize)//file 接收完毕
    {
        Return UPG_FINISHED；
    }
    Else
        return UPG_CONTINUE;
}
```

说明：

在编写上层逻辑时要求：

1. 每次登录到云端后，需要查询一次版本信息；
2. 以后每隔 24 小时查询一次版本信息；