**开发指南**

# SDK说明

接入SDK主要包含两个部分，MachtalkLib(src/Machtalk2)和Demo(src/Demo)示例代码。MachtalkLib为对平台接入协议和配网协议的封装库，以源码形式提供。用户将MachtalkLib源码集成到自己的工程中，并按流程调用接口，即可完成接入。

## MachtalkLib说明

MachtalkLib将平台的接入协议和配网协议完全封装好，用户在使用时根据需要调用对应的接口即可，整体结构如下：

Adapter

Platform API

Link API

MachtalkLib

Adapter Implement

User Code

User

Adapter: 适配层，用于适配不同的硬件环境，要求用户根据各自的硬件环境实现Adapter中头文件所定义的接口。

Platform API: 平台协议相关接口，定义在MT2.h和MT2Ctx.h中

Link API: 配网相关接口，定义在machtalk\_link.h中。

User Code: 用户代码

# 开发

## 概念定义

### UUID

设备的唯一标识码，在申请设备ID时会发送给云平台，相同的UUID的设备申请的设备ID相同。通常情况下为模块的MAC地址。

### 设备ID（Deviceid）

平台分配的设备ID用于标识不同的设备。

### 产品型号（Model）

平台上分配的产品型号，每款产品不同，用于区分产品。同时平台也会分配一个型号PIN码，型号PIN码在申请设备ID时使用。

## Adapter实现

使用MachtalkLib需要用户首先实现Adapter中所定义的接口，主要包含以下部分：

1. MT2AdapterCrypto

AES加密接口。

1. MT2NetifIsOK

判断网络是否可用，如WiFi设备为判断是否连接路由器。

1. MT2GetDeviceUUID

获取设备唯一标识，如为WiFi设备则返回MAC地址如AABBCCDDEEFF。

1. MT2Socket

Socket相关接口，硬件环境上的socket如为posix风格接口请参考Demo/PC中的代码实现，lwip的接口请参考Demo/ESP8266中的代码实现。

## User Code

### 平台接入

平台接入的API定义在MT2.h和MT2Ctx.h中，用户需要引用MT2.h。

#### 初始化

初始化代码中的平台参数中的登陆服务器、ID分配服务器、平台ID需要在开发前进行确认。

//初始化

MT2Init();

//创建平台上下文

g\_ctx = MT2CtxCreate();

//设置平台参数

MT2CtxPlatformCfg\_t platformCfg;

//ServerHost如为域名则直接填写，如果是IP地址则需要以"IP:"开头，如"IP:192.168.0.1"

platformCfg.loginServerHost = "nls.machtalk.net"; //登陆服务器

platformCfg.loginServerPort = 6779;

platformCfg.applyidServerHost = "ds.machtalk.net"; //ID分配服务器

platformCfg.applyidServerPort = 10080;

platformCfg.platformid = 0; //平台ID

MT2CtxSetPlatformCfg(g\_ctx, &platformCfg);

//设置事件处理回调

MT2CtxSetEventHandler(g\_ctx, mt2EventHandler);

//设置设备型号ID，型号ID从平台分配，每款产品唯一

MT2CtxSetDeviceModel(g\_ctx, MODEL\_ID);

//开始设备添加发现

MT2CtxStartAddDiscover(g\_ctx);

#### 申请设备ID.

设备ID用于唯一标识一款设备，在开始连接平台之前先要连接ID分配服务器分配设备ID。

//使用型号ID和型号PIN码进行设备ID申请，型号ID和型号PIN由平台分配

MT2CtxApplyDeviceid(g\_ctx, MODEL\_ID, MODEL\_PIN);

申请到设备ID后，会通过事件回调给用户，需要将设备ID进行永久化存储避免每次连接平台都进行申请。

#### 添加设备

在APP绑定设备需要将设备切换到添加发现，一般逻辑是先进行配网(详见WiFi配网)，配网成功连上路由器后，调用接口MT2CtxStartAddDiscover，将设备切换到添加发现状态，此时在APP上搜索设备(发送UDP广播包)，SDK会向APP回复设备信息(要求设备同APP处于同一局域网并且已经有设备ID)，完成绑定。需要注意的是调用MT2CtxStartAddDiscover进入添加发现状态的超时时间为1分钟，1分钟之后SDK会退出添加发现状态。

//进入添加发现状态，超时1分钟

MT2CtxStartAddDiscover(g\_ctx);

//回复APP添加发现

else if(event == MT2EVENT\_RESP\_ADD\_DISCOVER)

{

SysLog("resp add discover");

}

#### 连接平台

//设置设备ID和PIN码

MT2CtxSetDeviceid(g\_ctx, g\_deviceid, g\_devicepin);

//开始连接

MT2CtxStartConnect(g\_ctx);

#### 状态上报

平台同设备之间的控制、状态上报都通过属性来进行，不同的属性的ID不同，如将开关定义为属性1，温度定义为属性2，则将设备自身的开关状态通过属性1来上传，温度状态通过属性2来上传。属性类型包括两种，数值属性与文本属性。

//注册属性值

MT2CtxRegistProperty(g\_ctx, 1, MT2\_PROPERTY\_TYPE\_NUM);

MT2CtxRegistProperty(g\_ctx, 2, MT2\_PROPERTY\_TYPE\_NUM);

//同步属性值

MT2PropertyValue\_t value;

value.num = 0;

MT2CtxSyncPropertyValue(g\_ctx, 1, value);

MT2CtxSyncPropertyValue(g\_ctx, 2, value);

#### 控制

APP来的控制命令会通过事件回调通知用户，具体处理详见**事件处理**部分。

#### 事件处理

static void mt2EventHandler(MT2Ctx\_t \*ctx, muint8\_t event, void \*arg)

{

//控制

if(event == MT2EVENT\_OPT)

{

MT2EventOptArg\_t \*opt = arg;

uint8\_t i;

for(i = 0; i < opt->propertyNum; i++)

{

MT2Property\_t \*property = &opt->propertys[i];

//收到控制后需要将被控制后的属性值进行同步

MT2CtxSyncPropertyValue(g\_ctx, property->pid, property->value);

SysLog("control propertyid:%d value:%d", property->pid, property->value.num);

}

}

//申请设备ID

else if(event == MT2EVENT\_APPLYID)

{

//申请到设备ID和PIN码，应该写入flash保存

MT2EventApplyidArg\_t \*applyidArg = arg;

strcpy(g\_deviceid, applyidArg->deviceid);

strcpy(g\_devicepin, applyidArg->devicepin);

SysLog("deviceid:%s pin:%s", applyidArg->deviceid, applyidArg->devicepin);

}

//查询

else if(event == MT2EVENT\_QUERY)

{

//如果设备状态已经实时通过MT2CtxSyncPropertyValue同步则不需要进行任何处理，否则需要将属性同步一遍。

SysLog("Query");

}

//更新服务器时间

else if(event == MT2EVENT\_UPDATE\_SERVER\_TIME)

{

MT2EventUpdaeServerTimeArg\_t \*timearg = arg;

SysLog("timezone:%d seconds:%d", timearg->timezone, timearg->seconds);

}

//解绑成功

else if(event == MT2EVENT\_UNBIND\_RESULT)

{

SysLog("unbind success.");

}

//固件升级

else if(event == MT2EVENT\_UPGRADE)

{

}

}

### WiFi配网

#### 设备要求

**无法满足以下以下能力要求的设备或模块将无法进行WiFi配网。**

**●能够切换信道**

**●能够设置为混杂模式，接收802.11 无线帧**

**●提供一种进入配网模式的控制方式，例如一个按键**

#### 配网流程

* 开始配网

//初始化

mtlink\_init();

//切换到混杂模式并接收802.11无线帧

WifiSnifferStart(wifiSnifferCallback);

* 100ms循环切换信道

static uint8\_t chn = 1;

//未锁定信道时，循环100ms切换信道

if(!g\_lockChn && g\_running - g\_lastSwitchChnTime > 100)

{

g\_lastSwitchChnTime = g\_running;

WifiSnifferSetChannel(chn);

chn++;

if(chn > 14)

{

chn = 1;

}

}

* 解析802.11帧数据

static WifiSnifferStatus\_t wifiSnifferCallback(const uint8\_t \*data, uint16\_t len)

{

mtlink\_status\_t status = mtlink\_recv(data, len);

//锁定信道

if(status == MTLINK\_STATUS\_LOCK)

{

g\_lockChn = 1;

}

//配网完成

else if(status == MTLINK\_STATUS\_COMPLETE)

{

mtlink\_res\_t res;

mtlink\_get\_res(&res);

SysLog("got ssid:%s pwd:%s", res.ssid, res.pwd);

g\_startNetCfg = 0;

WifiSnifferStop();

//连接路由器

WifiJoinAp(res.ssid, res.pwd, WIFI\_AUTH\_WPA\_WPA2\_PSK);

WifiReconnect();

//开始设备添加发现，开始设备添加发现之后才能被APP发现并添加

MT2CtxStartAddDiscover(g\_ctx);

}

}