CC2541-QQ 物联接入

规格书

SDK 版本：V1.0

深圳市昇润科技有限公司

2016 年 4 月 26 日

1. 概述

QQ 物联平台将 QQ 账号体系及关系链、QQ 消息通道能力等核心能力，提供给可穿戴设备、智能家居、智能车载、传统硬件等领域合作伙伴，实现用户与设备及设备与设备之间的互联互通互动，充分利用和发挥腾讯 QQ 的亿万手机客户端及云服务的优势，更大范围帮助传统行业实现互联网化。

CC2541 是一款针对低能耗以及私有 2.4GHz 应用的功率优化的真正片载系统(SoC)解决方案。 CC2541 非常适合应用于需要超低能耗的系统，非常适合用来开发可穿戴设备、智能家居、智能车载、传统硬件等。

因此，CC2541+QQ 物联的开发方案就显得很有价值了。

2. 应用

QQ 物联平台可以将所有符合协议的蓝牙设备接入到平台上，同时 QQ 平台还支持为设备定制控制界面，开发者只需要自己用 html5 制作好想要的控制界面，然后在 QQ 平台上将该网址绑定到我们的设备上就可以使用该网页对我们的蓝牙设备进行控制或者接收设备的信息了。

2.1 空气净化器

我司现有支持 QQ 物联的设备是一款空气净化器。用户通过手机 QQ 扫描设备对应的二维码后就可以绑定该设备，界面如图一所示。绑定之后进入设备，界面如图二所示。从图三中我们可以看出，通过这个网页 APP，我们可以轻松的控制净化器的开启和关闭，同时还可以为净化器设置定时开关，非常便捷、实用。图四是净化器工作时的效果图。

图一　　图二　　图三　　图四

3. QQ 物联接入步骤（参考自 QQ 物联平台上的相关文档）

说明：本节内容所述的方法适用于开发调试时使用，量产时的操作方法有些许不同，关于量产的介绍在第四节有详细说明，建议先看完本节的内容后再去看第四节。

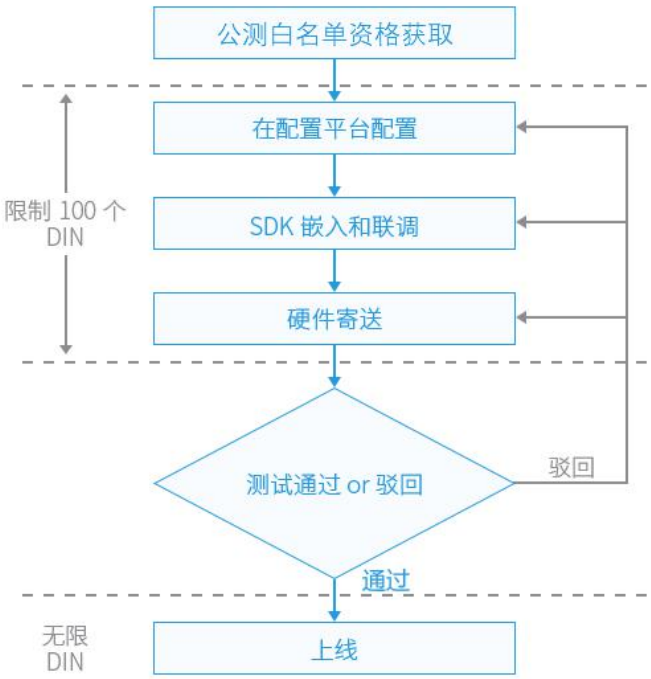


图 A　接入流程图

3.1 开发前的准备工作

3.1.1 申请开发者账号

进入腾讯开放平台官网 用一个 QQ 号，申请“公司”身份的开发者帐号。最好是用司公用的 QQ 号来申请，以防人事变动造成不必要的麻烦。如图 B 所示。

图 B

3.1.2 申请白名单

在 QQ 物联公测申请页中填写并提交相应的内容，来获得 QQ 物联的公测资格。如图 C 所示。

图 C

3.1.3 进入配置平台

白名单开启后，便可以登入 QQ 物联配置平台，进入平台后就可以添加、配置新的设备了。

3.1.4 添加新设备

进入 配置平台 - 设备 之后，点击添加设备按钮，填写“设备名称“及选择“设备类型”后即可开始快速注册一个新设备。

完成后，我们已经获得了该产品的重要信息：pid 和 server key，这两个信息非常重要，会在后面的 Step3.5 中用到。在设备导航栏中，选择进入相应的设备，即可在头部看到，如图 D。

建议：可以把与设备相关的一些文件或者信息(包括这里的 pid 和 server key 和后面的 ec_key.pem、public.pem 已及 SN 等)保存到一个文件夹里，方便后面的使用。

图 D

按照自己的需求补全设备的基本信息，如图 E、图 F。

图 E

图 F

设备型号：可以自由填写，设备名称请尽量用中文并控制长度，因为这些字符会出现在手机 QQ 的界面里，是用户可以直接看到的。

设备类型： 必须填写准确，后期无法进行修改，这里我们选择设备类型为空气净化器。

操作系统：这里我们选择 其它，连接方式 我们选择 蓝牙。

初始化方式：决定了设备的入网方式以及跟手机 QQ 的绑定方式，所以这个选项很重要，这里选择蓝牙，这里只支持 4.0 以上，即低功耗蓝牙，不支持经典蓝牙。

公钥上传： 稍微复杂一点，因为需要下载一个工具来完成这个步骤，点击网页上的公钥&证书工具下载（Win 7 Only）链接，如图 F 所示的 “公钥上传” 处即可下载。然后运行密钥生成工具，点击下图 G 中的生成 KEY 按钮，会在指定的目录下生成一对非对称密钥文件：ec_key.pem 和 public.pem。之后点击上传按钮上传 public.pem 就可以了，如图 G 所示。

图 G

说明：以后的环节中会反复用到 ec_key.pem，所以一定要妥善保管这个文件。

接下来还需要上传一个设备激活登记表，如图 H 所示。

图 H

可以点击图 H 中箭头所指的位置查看说明。

这个登记表需要我们自己制作，制作方法如下：
1）、新建一个 excel 文件；
2）、在文件的第一行第一列位置输入 SN(serial number：序列号)，第一行第二列位置输入设备的 MAC 地址，如图 I 所示；
3）、添加完成之后，将文件另存为 csv 格式。

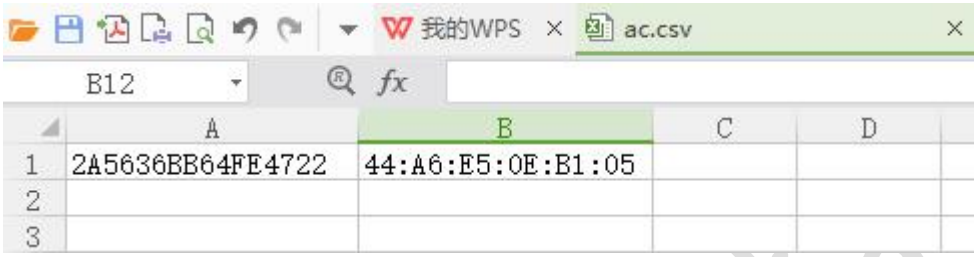


图 I

说明 1：上面第二步中用到的 SN 需要用工具软件生成，如图 J 所示，使用的工具是前面下载的公钥&证书工具（Win 7 Only）。

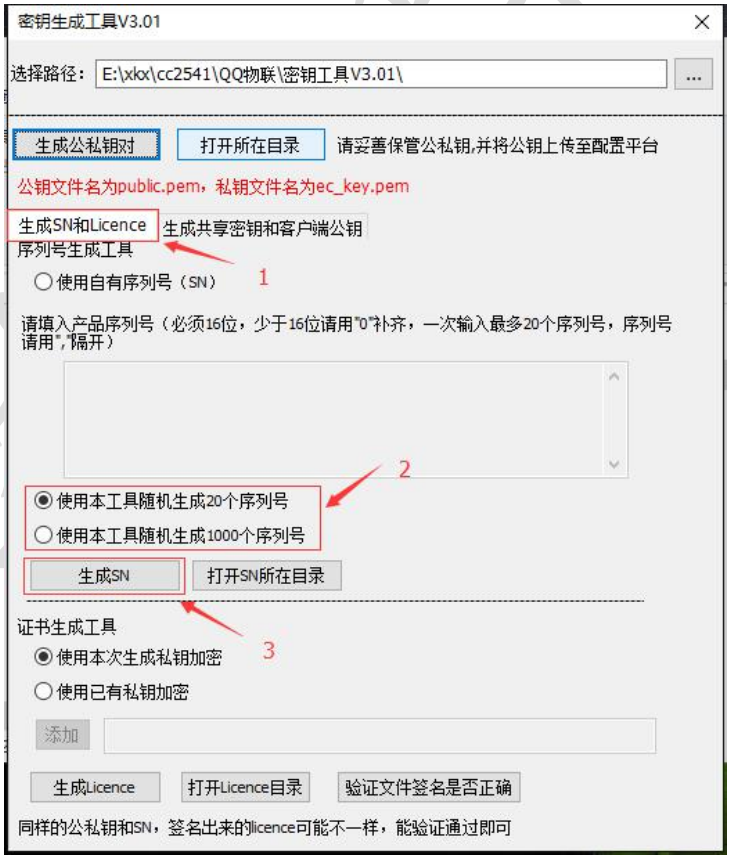


图 J

只需按照图中的 1、2、3 步分别点击操作即可获得 SN。

说明 2：不能直接将 excel 文件的后缀名改成 csv，而应该用另存为的方法。

制作好登记表后就可以上传了，后面如果需要增加设备的数量，也是在这里把新的 MAC 地址和对应的 SN 进行登记，只有在这里登记了，我们的设备才允许接入 QQ 物联平台。

上传之后记得点击页面下方的保存按钮。

3.1.5 设置控制器

功能添加完成之后，切换到控制器设置页签，填写控制器相关的信息。可以采用 QQ 物联开发的公用模版控制器或者自己设计的 Html 5 控制器。

控制器为用户实际用来操控设备的“轻 APP”。

如图 K 所示。

图 K

3.1.6 提交审核

配置平台允许在不提交审核之前以及审核中拥有 100 个 DIN 用于测试。也就是说在批量生产调试之前可以按需更改配置即时生效。

从右上角的提示可以看到当前设备所在的状态。同时获知您现在所使用的 DIN 数量，如图 L 所示。

图 L

当确认功能调试完毕时。请查看审核须知，并在再三确认调试完毕后点提交审核.请注意提交审核到审核通过期间将无法修改所有的功能及触发器信息,所以只有当确定调试完成后方可重新提交审核。

到 3.1.5 步为止，我们已经完成了一个新设备的新建，下面开始进行蓝牙设备的配置环节。

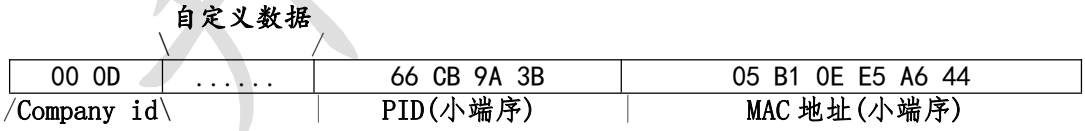
3.2 配置蓝牙广播和服务

蓝牙设备在未被 App 连接到之前，需要一直发广播，广播中至少包含 ServiceUUID 和 manufature specific data，内容里的字节序采用小端序(Little Endian)。

QQ 物联的蓝牙广播和 Service 标准如下：

名称	值	作用
manufature specific data	以 PID(4 字节) + MAC(6 字节)结尾	Device 发广播时标识设备
ServiceUUID	0xFEBA	标识 QQ 物联
Write Characteristics UUID	0000fec7-feba-f1f1-99c0-7e0ce07d0c03	App 向 Device 传输数据(需要 response)
Indicate Characteristics UUID	0000fec8-feba-f1f1-99c0-7e0ce07d0c03	Device 向 App 传输数据(需要 confirm)
Read Characteristics UUID	0000fec9-feba-f1f1-99c0-7e0ce07d0c03	Device 不发广播时标识设备

ServiceUUID 和 Characteristic 的值是固定的，按照格式填写即可。manufature specific data 的计算如下：比如某设备 PID=1000000358(0x3B9ACB66),MAC 地址=44:A6:E5:0E:B1:05，则 MSD 数据如下：

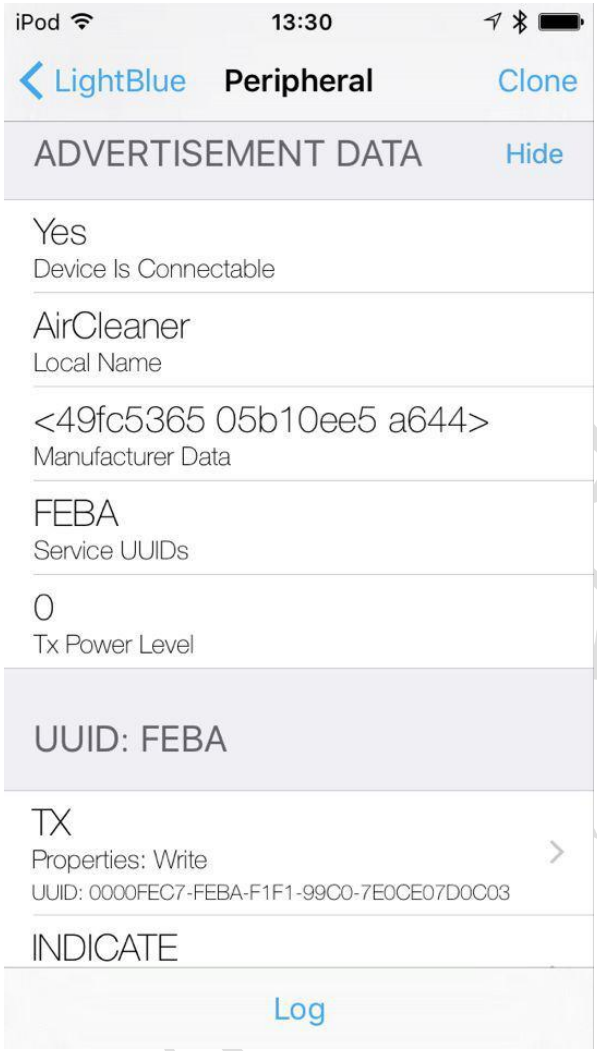


广播数据结构示意图

Read Characteristics 里的内容跟广播 manufature specific data 的数据要求一致。

注意：UUID 为 128bit 类型；数据传输为 Indicate 方式。

以上面的空气净化器为例，用 LigthBlue 查看设备信息如图一所示。



图一

3.3 下载硬件 SDK

硬件 SDK 的下载地址为：http://iot.open.qq.com/wiki/index.html#!SDK/BLE.md
（此文档是以 V1.0 的 SDK 为基础进行说明的，若后面 SDK 有更新，请以新的说明为准）

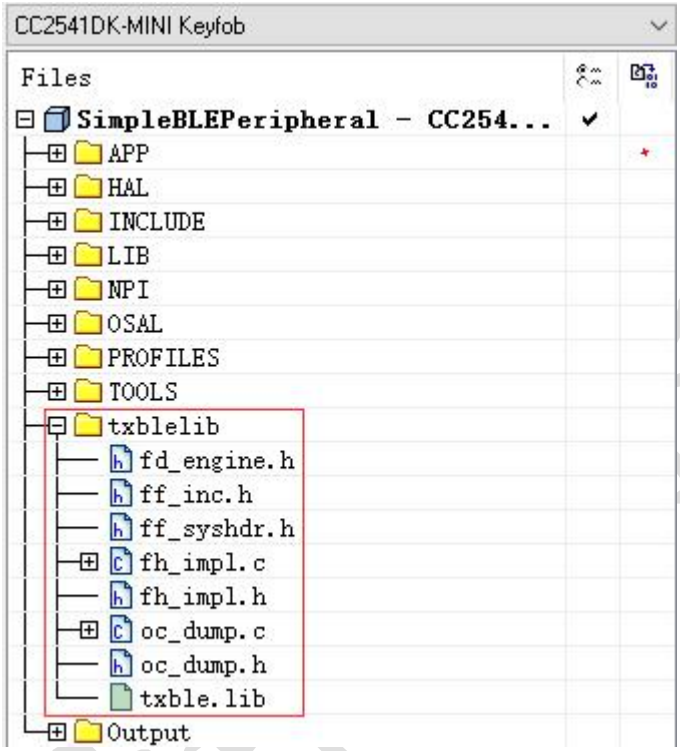
压缩包中包含静态库文件以及相应的头文件。

关于 SDK 压缩包的说明：

文件名	作用
ff_syshdr.h	系统头文件定义，它可以根据编译环境，配置是否使用某些基础库
ff_inc.h	通用数据结构及枚举值的定义
fd_engine.h	SDK 的接口定义
txble.lib/a	静态库文件，SDK 的具体实现

3.4 第一行代码

把刚下载的 SDK 加入到工程中，如图二所示。



图二

在工程的初始化函数里调用 SDK 初始化函数，如图三所示。

```
static uint8_t s_fd_zone[256];
/*********************************************************************************
【函    数】    SimpleBLEPeripheral_Init
【概    述】    任务初始化
【入口参数】    task_id 任务ID
【返回参数】    无
【说    明】    无
 *********************************************************************************/
void SimpleBLEPeripheral_Init( uint8 task_id )
{
    simpleBLEPeripheral_TaskID = task_id;

    impl_data.TaskID = simpleBLEPeripheral_TaskID;

    osal_setClock(osal_ConvertUTCSecs(&data.bell));                    //同步当前的时间

    fh_data_set_default();
    fh_data_load();                                                    //加载存储在flash里的数据

    g_fd_engine = fd_init(s_fd_zone, 256);

    if (g_fd_engine != NULL)
    {

    }
```

图三

关于 fh 和 fd 的说明：
在 fd_engine.h 里定义了两类函数
一类是以 fd 开头，由 SDK 实现，被上层调用；
一类是以 fh 开头，由开发者实现，供 SDK 调用。

3.5 配置初始化数据

初始数据每个设备都不相同，宏定义于 fh_impl.h 文件中，如图四所示。

```
#define FT_SET_DEFAULT          0

#define FT_LICENSE {                                                    \
                    0x30,0x45,0x02,0x21,0x00,0xE1,0xAC,0x3A,0x76,0x1E,0xD8,0xE5,0x53,\
                    0x55,0xCF,0x87,0x73,0x07,0xCA,0xC1,0xEE,0x04,0xA2,0x13,0xBE,0xAB,\
                    0x63,0x6F,0x0B,0x05,0xD1,0x8C,0xB3,0x5B,0x91,0xB0,0xF6,0x02,0x20,\
                    0x29,0x9F,0x33,0x74,0x86,0x1E,0x3C,0xC3,0xBD,0xFE,0xC2,0x48,0xAA,\
                    0xA2,0x89,0xB8,0xC7,0x9A,0xEF,0xBD,0x72,0xA8,0x9E,0x75,0x68,0x70,\
                    0xC8,0x0D,0x39,0x29,0xD3,0x24                       \
                  }


//设备PID
#define FT_DEFAULT_PID      1700002889                                  //0x6553FC49

//共享密钥
#define FT_DEFAULT_AUTH_KEY {0x2C,0xD2,0xD7,0x28,0x65,0xD5,0xB9,0x3E,0xAE,0x20,0xCB,0x98,0xF2,0x01,0xA5,0x62}

//客户端公钥
#define FT_DEFAULT_PUK      {0x03,0xBE,0x26,0xC7,0x7E,0x13,0x52,0x5D,0xC6,0x29,0x97,0x99,0x47,0xC3,0x26,0x25,\
                             0xD4,0x61,0xE0,0xF6,0x5A,0x9E,0x6C,0x6F,0x2B}
```

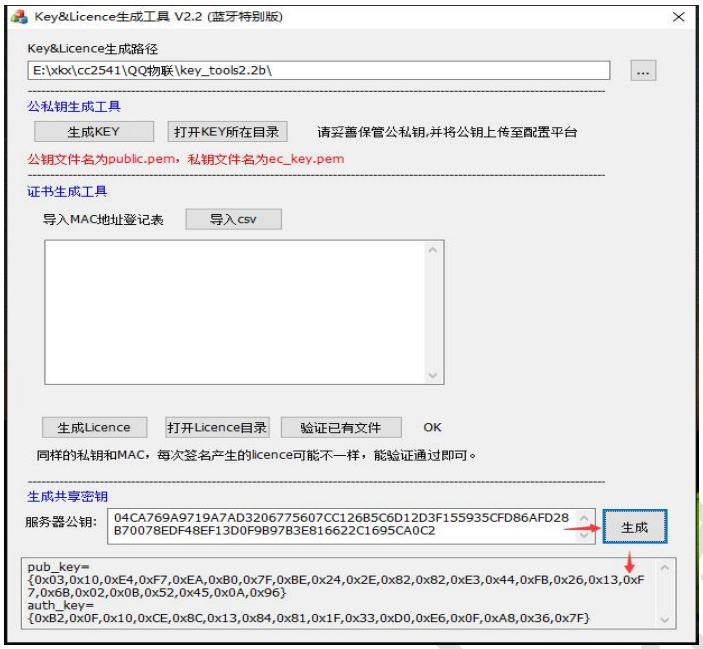
图四

1)、PID 与服务器公钥的获取
登录 QQ 物联平台，找到相应的设备即可得到 PID 与服务器公钥，如图五所示。

图五

其中，服务器公钥需要下载到本地，只需点击图中蓝色字体位置即可下载。

2)、PUK（设备公钥）和 AUTH KEY（设备共享密钥）
将上面下载的服务器公钥文件用文本的方式打开，复制其中的所有内容，打开我们前面下载的工具，将刚才复制的服务器公钥文件里的内容粘贴到“生成共享密钥”栏,然后点击“生成”按钮，如图六所示。



图六

根据服务器公钥，每一个设备都可以生成一对设备密钥。

注意：测试时同一类产品可以共用一对设备密钥，但量产时，需要为每一个设备生成一对唯一的密钥，将 QQ 物联提供的 C++源代码，集成到自己的烧录工具中，可为每一个设备自动生成一对密钥，并写入 flash。

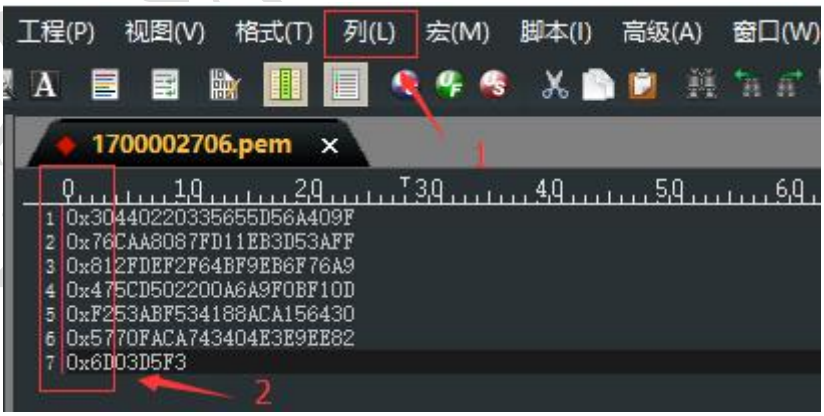
3)、licence(认证签名)

使用图七所示的工具，首先指定上面生成的 ec_key.pem 文件所在目录，然后点击 “导入 csv” 导入前面我们制作的 csv 文件，然后在其下的输入框中输入 CSV 文件中的 MAC 地址，如有多组 MAC 地址，需用逗号分隔开，如图七所示。

图七

说明 1：箭头 1 所指为 ec_key.pem 文件所在目录。

说明 2：生成的 licence.csv 文件的内容是一串十六进制的数据，我们需要将这些数据处理一下才能放到代码里。也就是在每个数据前加上“0x”，在每个数据之间加上 “,”。我们可以借助一些具有列编辑功能的文档编辑工具如 UltraEdit 来进行这些操作，如图八所示。



图八

首先复制 licence 文件里的内容，打开编辑工具，粘贴我们复制的内容，然后点击箭头 1 所指的列选项，在其下拉列表里选择“列模式”，用鼠标选中数据的一列，输入需要的内容即可。操作完了之后需要再次点击箭头 1 处，选择“列模式”，表示退出列模式。之后就可以复制我们的处理好的数据了。

得到这些数据之后对应替换图四中的内容。在 SDK 中有专门的一个数据类型用于存储这些数据，如下。

```
typedef struct _QQ_SEC_INFO
{
   uint8_t  license[FT_MAX_LICENSE_LEN];
   uint16_t licenselen;
   uint8_t  din[DIN_LEN];
   uint32_t pid;
   uint8_t  authkey[AUTH_KEY_LEN];
   uint8_t  pukstr[PUK_STR_BUFFER_LEN];
}QQSecInfo;
```

3.6 实现 fh_*中的函数

fd_engine.h 中有部分函数是交给开发者实现的，这部分函数都是以 fh 开头。

int8_t fh_send(uint8_t* datum,uint16_t length);

这个函数是 SDK 在发送数据时调用的，这个函数只是一个接口，功能就是发送数据，需要开发者用自己的方式实现这个发送数据的功能，如图九。

开发者在发送完一包数据后，需要调用 void fd_flush(struct fd_engine_s engine);来告知 SDK，否则 SDK 不会继续发送数据如图十一。另外，SDK 会传入需要发送的全部数据，由开发者负责分帧。

void fh_load_mac(uint8_t* zone);

返回设备的 mac 地址。比如我们这台设备(mac 地址 44:A6:E5:0E:B1:05)返回的是"44A6E50EB105"，注意顺序和大小写。

fd_save_din/fd_load_din

需要用到 NVRAM 或 Flash。这个 bin 与设备的连接有关，如果没有处理好则会导致设备在第一次绑定连接之后再次连接时无法连接，需重新扫描才能再次连接(正常的情况是在设备正常工作的情况下，再次连接时只需在手机 QQ 上直接点击该设备图标)。

fh_data_load/fh_data_store

这两个函数主要用于存储于 QQ 物联有关的数据，如图十二所示。

其它方法根据需要自行实现或留空。

```
/*********************************************************************************
【函    数】    fh_send
【概    述】    this should implement by hardware bussiness.
                这个函数是SDK在发送数据时调用的。
【入口参数】    datum - 数据地址  length - 数据长度
【返回参数】    无
【说    明】    无
 *********************************************************************************/
int8_t fh_send(uint8_t* datum, uint16_t length)
{
    data.InDecaFinish  = 0;                                          //指示数据是否全部发送完成
    data.InDecaDatum   = datum;
    data.InDecaLength  = length;

    osal_set_event(impl_data.TaskID, INDECA_EVT);                    //                        |

  return 0;
}
```

图九

图中的实现方法是将收到的数据存储到缓冲区中，然后置起专门的数据发送事件来发送这些数据，INDECA_EVT 事件里主要处理的就是数据的分包和发送，如图十所示。

```
/*********************************************************************************
【函    数】    SimpleBLEPeripheral_ProcessEvent
【概    述】    任务事件处理函数
【入口参数】    task_id - 任务ID    events - 事件
【返回参数】    返回未做处理的事件标号
【说    明】    无
*********************************************************************************/
uint16 SimpleBLEPeripheral_ProcessEvent( uint8 task_id, uint16 events )
{
    VOID task_id;

    if( events & SYS_EVENT_MSG )
    {
    if( events & INIT_EVT)                                           //设备初始化
    {
    if( events & MAIN_LOOP_EVT)                                      //主循环
    {
    if( events & INDECA_EVT)                                         //数据上传处理事件
    {

    return 0;
}
```

图十

上图是整个任务处理函数的结构，关于 INDECA_EVT 事件的具体处理如图十一所示。

```
if( events & INDECA_EVT)                                                    //数据上传处理事件
{
   if(data.InDecaLength)                                                    //剩余数据长度不为0
   {
      if(data.InDecaLength >= 20)                                           //剩余数据长度超过20  需要做分包处理
      {
         error_code = Periphral_Indicate(&data.InDecaDatum[data.InDecaIndex],20,0);//上传数据

         if(SUCCESS == error_code)                                          //数据上传处理成功
         {
            fd_flush(g_fd_engine);                                          //告知SDK 本次的数据已经发送完毕

            data.InDecaIndex   += 20;                                       //更新索引号
            data.InDecaLength  -= 20;                                       //更新剩余数据长度
         }
      }
      else
      {
         error_code = Periphral_Indicate(&data.InDecaDatum[data.InDecaIndex],data.InDecaLength,0);

         if(SUCCESS == error_code)
         {
            fd_flush(g_fd_engine);                                          //告知SDK 数据已经发送完毕

            data.InDecaLength = 0;
            data.InDecaIndex  = 0;
            data.InDecaFinish = 1;                                          //指示数据已经全部发送完毕
         }
      }
   }
   else if(0 == data.InDecaFinish)                                          //最后一包数据刚好是20byte
   {
      data.InDecaIndex  = 0;
      data.InDecaFinish = 1;
   }

   if(0 == data.InDecaFinish)                                               //若数据没发完
   {
      osal_start_timerEx(simpleBLEPeripheral_TaskID,INDECA_EVT,30);         //
```

图十一

上图中 Periphral_Indicate()函数是调用库函数实现的以 indicate 方式上传数据的函数。整个过程的处理就是先判断需要发送的数据包的长度,若长度超过 20 字节(蓝牙每次最多发送 20 字节)则做分包处理，即本次数据以最大长度(20 字节)发送，发送之后将数据包的长度减少 20，同时索引号增加 20，之后就给这个事件设置一个定时，也就是等待一段时间后再次来这个事件里接着上次的位置继续发送数据，直至全部发送为止。

```
/*********************************************************************************
【函    数】    fh_data_load
【概    述】    数据加载.
【入口参数】
【返回参数】    无
【说    明】    无
 *********************************************************************************/
void fh_data_load(void)
{
    osal_snv_read(BLE_USER_ID,(sizeof(qqSecurityInfo) / sizeof(uint8_t)),&qqSecurityInfo);//加载数据
}
/*********************************************************************************
【函    数】    fh_data_store
【概    述】    数据存储.
【入口参数】
【返回参数】    无
【说    明】    无
 *********************************************************************************/
void fh_data_store(void)
{
    osal_snv_write(BLE_USER_ID,(sizeof(qqSecurityInfo) / sizeof(uint8_t)),&qqSecurityInfo);//存储数据
}
```

图十二

上图的操作是直接把 qqSecurityInfo 存储到指定 flash 区域。

3.7 调用 fd_engine.h 中的函数

实现了 fd_engine.h 中的函数后，工程就可以编译通过。但是要让 SDK 正常运行，还需要按顺序调用下面方法。

设备连接/断开时，调用 fd_connect/fd_disconnet，如图十二所示。

在接收数据的地方，调用 int8_t fd_received(struct fd_engine_s* engine, uint8_t* datum,uint16_t length);将收到的数据传给 SDK。这里上层业务从 QQ 物联定义的 Write 特征值里收到数据，立即调用此函数，不需要累积数据包，而由 SDK 组包，如图十三所示。

```
/*********************************************************************************
【函    数】    peripheralStateNotificationCB
【概    述】    设备状态回调处理函数
【入口参数】    newState - 设备状态
【返回参数】    无
【说    明】    无
*********************************************************************************/
static void peripheralStateNotificationCB( gaprole_States_t newState )
{
  int ret = 0;

  switch ( newState )
  {
    gapProfileState = newState;

    if(gapProfileState == GAPROLE_CONNECTED)                    //连接状态
    {
      ret = fd_connect(g_fd_engine);

      if(ret == 0)
      {
      }
    }
    else                                                        //未连接状态
    {
      ret = fd_disconnet(g_fd_engine);

      if(ret == 0)
      {
      }
    }
}
```

图十三

```
/*********************************************************************************
【函    数】    simpleProfile_WriteAttrCB
【概    述】    特征值变更回调函数
【入口参数】    无
【返回参数】    无
【说    明】    无
*********************************************************************************/
static bStatus_t simpleProfile_WriteAttrCB( uint16 connHandle, gattAttribute_t *pAttr,uint8 *pValue, 
{
    bStatus_t status = SUCCESS;
    uint8     notifyApp = 0xFF;
    uint8     notifyLen = 0;

    // If attribute permissions require authorization to write, return error
    if ( gattPermitAuthorWrite( pAttr->permissions ) )
    {
    if ( pAttr->type.len == ATT_BT_UUID_SIZE )                                  //16-bit UUID
    {
    else if( pAttr->type.len == ATT_UUID_SIZE )                                 //128-bit UUID
    {
        const uint8 uuid[ATT_UUID_SIZE] =
        {
        if ( osal_memcmp(uuid, simpleProfilechar1UUID, ATT_UUID_SIZE) == 0 )
        {
            if ( offset == 0 )
            {
            else
            {
            if ( status == SUCCESS )
            {
                uint8 * pCurValue = (uint8 *)pAttr->pValue;
                VOID osal_memcpy(pCurValue, pValue, len);
                fd_received(g_fd_engine, pValue, len);
```

图十四

至此，代码部分的修改已经完成。

接下来还有一个很重要的内容，那就是生成设备的二维码，蓝牙设备目前只支持通过扫描设备二维码的方式绑定、连接设备。二维码的生成步骤如下；

1)、去网上任意搜索一个二维码生成工具，比如：
http://cli.im/text?5bfc12ac983fae4719e32ad7f440b58c

2)、在输入栏输入如下内容(是一个网址,直接复制，而不是打开这个网址)：
http://iot.qq.com/add?pid=<>&sn=<>

3)、将设备对应的 PID 和 SN 填写到相应的位置，如图十四。

图十五

注意：替换的时候需要连同尖括号一起替换掉。

生成二维码之后就可以下载到本地保存了。

将前面修改好的程序烧写到我们的设备之后，用手机 QQ 扫描二维码即可绑定、连接我们的设备，如图十五、图十六。

图十六

图十七

注意：QQ 物联对手机 QQ 的版本也有要求。Android QQ 要求 5.5 版本以上，iPhone QQ 要求 5.7 版本以上，在扫描二维码之前请先检查手机 QQ 的版本是否符合要求。

至此，一个可用于调试的 QQ 物联设备已经完成。接下来介绍关于 QQ 物联设备量产的内容。

注意：以上的所有操作方法都是针对只用于调试的设备的。量产时，这些步骤又有所不同，具体的内容请参照下面的内容。

4. 关于 QQ 物联设备量产的操作方法

4.1 用于调试的操作方法的局限性介绍

从 Step3 中我们了解到，每一个蓝牙设备，我们都需要为其生成对应的 FT_DEFAULT_PUK、FT_DEFAULT_AUTH_KEY、licence、二维码等，操作十分繁琐，效率非常低，不适用于量产。

4.2 量产时的操作方法介绍

首先我们来列举一下我们需要得到的数据以及其来源。

服务器公钥　：同一类设备(比如空气净化器)共用一个服务器公钥，从 QQ 平台获取；
PID　：同一类设备(比如空气净化器)共用一个 PID，从 QQ 平台获取；
SN　：通过工具批量生成；
设备公钥(PUK)　：服务器公钥 + 工具，量产时要求每一个设备的 PUK 不同；
设备共享密钥(AUTH KEY)：服务器公钥 + 工具，量产时要求每一个设备的 AUTH KEY 不同；
(同一个　服务器公钥，每次生成出来的 PUK 和 AUTH KEY 都不同)
设备登记表　：SN + MAC；
Licence　：MAC + 设备登记表 + 工具；
二维码　：PID + SN + 工具；

从上面的总结可以看出对于同一类设备来说，有些信息是固定的，而有些信息各不相同。

可行的做法如下：

1)、 在代码中把已知的数据填写进去，比如 PID。其他的留空，如 licence 等；
2)、 在代码中制作一个数据接口，用于接收在第一步中留空的数据；
3)、 使用工具批量生成 SN；
4)、 使用工具批量生成二维码(PID + SN + 工具)；

5)、 将生成的二维码和我们的设备随机组合(物理上的组合，也就是放到一起)；
6)、 将第五步中组合好的设备一个一个通过我们的量产工具；
7)、 量产工具扫描设备得到 MAC 地址，扫描和设备组合的二维码，得到对应的 SN；
8)、 量产工具在得到 SN 和 MAC 后生成一个 csv 文件；
9)、 量产工具在生成 csv 文件后再生成该设备对应的 licence；
10)、量产工具通过输入的服务器公钥生成 PUK 和 AUTH KEY；
11)、量产工具将生成的 licence 通过第二步中预留的数据接口把 licence 写入到设备中；
12)、量产工具将 PUK 和 AUTH KEY 通过第二步中预留的数据接口把数据写入到设备中；
13)、量产工具处理完所有设备后，输出一个总的 csv 文件，用于 QQ 物联平台登记注册。

也就是说，在编写底层代码的时候，我们可以先把那些确定的信息写入到代码里，那些不确定的信息到后面的量产处理时再生成、写入，这样就避免了为每一个设备编写一版程序的操作。也就是说如果用上面的方法，我们就可以编写出一套针对一类设备的通用代码了。

上面流程中的一个关键点就是这个量产工具，这个工具需要自己开发，开发方法可以参照 3.5.2 点的注意事项，QQ 物联平台有提供相关的开源包，开发者需要将该源码包集成到量产工具里，源码包中包含了 licence 等数据的生成方法。

说明：上面步骤中，设备在收到数据接口发过来的数据后就会将数据保存到非易失性存储器(flash)里,成功收到数据后会有相应的回应，以供量产工具进行判断。

5.版本说明与修订记录

Software Version: BLE-STACK V4.0
Hardware Version: HY-254101 V1 模组
(深圳昇润科技 www.tuner168.com)
IDE: IAR 8.30.3 for 8051
Diver Version: V1.0

修订记录：

日期	修订	版本	修订说明
2016/4/28	0	V1.0	首次发布