青风带你玩蓝牙 nRF52832 系列教程

----------作者：青风

出品论坛：　www.qfv8.com　青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：346518370
硬件平台：青云 QY-nRF52832 开发板

蓝牙协议中的通用访问规范 GAP 详解

对应蓝牙广播的初始化过程中，为了更好的探讨和认识基础协议栈，我们将深入主机层，讨论协议栈中的通用访问规范 GAP，GAP 是和用户应用层紧密相连的。

并且通过分析基本原理，在匹配的 SDK15 的蓝牙样例的例子基础上就行分析与讲解，使用的协议栈为：s132。

1： nRF52832 蓝牙 GAP 初识：

通用访问配置文件(Generic Access Profile, GAP)，该 Profile 保证不同的 Bluetooth 产品可以互相发现对方并建立连接。

（GAP）定义了蓝牙设备如何发现和建立与其他设备的安全（或不安全）连接。它处理一些一般模式的业务（如询问、命名和搜索）和一些安全性问题（如担保），同时还处理一些有关连接的业务（如链路建立、信道和连接建立）。 GAP 规定的是一些一般性的运行任务。因此，它具有强制性，并作为所有其它蓝牙应用规范的基础。 GAP 是所有其他配置文件的基础,它定义了在蓝牙设备间建立基带链路的通用方法.除此之外,GAP 还定义了下列内容:

①:必须在所有蓝牙设备中实施的功能

②:发现和链接设备的通用步骤

③:基本用户界面术语

GAP 确保了应用程序和设备间的高度互操作性，还允许开发人员利用现有的定义更加容易地定义新的配置文件。GAP 处理未连接的两个设备间的发现和建立连接过程。此配置文件定义了一些通用的操作，这些操作可供引用 GAP 的配置文件，以及实施多个配置文件的设备使用。GAP 确保了两个蓝牙设备可通过蓝牙技术交换信息，以发现彼此支持的应用程序。不符合任何其他蓝牙配置文件的蓝牙设备必须与 GAP 符合以确保基本的互操作性和共存。

2: GAP 软件设计分析：

如果光像开篇一样讲理论，大家可能会云里雾里不知所以，下面打开 NRF52832 蓝牙样例，找到 GAP 初始化函数 gap_params_init，配置 GAP 代码来进行对照分析，结合代码深入的理解蓝牙协议。这样非常直观。代码如下：

```
01. static void gap_params_init(void)
02. {
03.     uint32_t                err_code;
04.     ble_gap_conn_params_t   gap_conn_params;
05.     ble_gap_conn_sec_mode_t sec_mode;
06.   //安全模式，安全权限设置
07.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);//连接模式，主要是是否需要加密
08.     //设备名称设置
09.     err_code = sd_ble_gap_device_name_set(&sec_mode,
10.                                           (const uint8_t *) DEVICE_NAME,
11.                                           strlen(DEVICE_NAME));
12.     APP_ERROR_CHECK(err_code);
13.     //应用图标值的设置，如果没有图标，可以不配置
14.   /* YOUR_JOB: Use an appearance value matching the application's use case.
15.        err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_);
16.        APP_ERROR_CHECK(err_code); */
17.     memset(&gap_conn_params, 0, sizeof(gap_conn_params));
18.
19.     //连接参数配置
20.     gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
21.     gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
22.     gap_conn_params.slave_latency     = SLAVE_LATENCY;
23.     gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
24.     //连接参数设置，主要是时间间隔设置
25.     err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
26.     APP_ERROR_CHECK(err_code);
27. }
```

这个代码实际上主要完成了 3 个工作：

1：设置连接的安全模式，代码第 5 ~7 行。

2：配置设备外观和名称，代码第 9 ~16 行。

3：设置连接间隔，代码第 17 ~26 行。

下面就从这三个方面进行展开。

2.1 GAP 初始化安全模式配置：

首先来谈谈连接模式，也就是安全设置，GAP 通常还会负责启动 BLE 连接的安全功能。只有对通过身份验证的连接而言某些数据是可读或可写的。一旦形成一个连接，两个设备可以通过一个过程被称为配对。进行配对时，密钥建立加密和认证的链接。在

一个典型的案例，外围设备需要中央设备提供密钥以完成配对过程。这可能是一个固定值，如“000000”（看静态密钥实验），或可能是一个随机生成的值（看随机密钥实验）被提供给用户。中央设备发送正确的密钥后，两台设备交换安全密钥加密和验证的链接。

在许多情况下，相同的中央设备和外围设备将会经常建立连接和断开连接。BLE 具有一个安全功能允许两个设备在配对的时候给对方一个长久的安全密钥。此功能称为绑定，使得两个重连设备能够迅速重新确立加密和认证而不需要经过充分的配对过程，只要它们存储长期的密钥信息。

阅读蓝牙协议 5.0 核心规范，第 3 卷 C 部分 5 节了解详情。

Nrf52832 处理器中设置先查看 ble_gap_conn_sec_mode_t 结构体，下面是介绍下改结构体设置的安全等级：

Security Mode 0 Level 0: No access permissions at all (this level is not defined by the Bluetooth Core specification).

Security Mode 1 Level 1: No security is needed (aka open link).

Security Mode 1 Level 2: Encrypted link required, MITM protection not necessary.

Security Mode 1 Level 3: MITM protected encrypted link required.

Security Mode 1 Level 4: MITM protected LESC encrypted link required.

Security Mode 2 Level 1: Signing or encryption required, MITM protection not necessary.

Security Mode 2 Level 2: MITM protected signing required, unless link is MITM protected encrypted.

上面可以看出有 0,1,2 三种模式，0,1,2,3,4 五种水平，代码定义如下：

模式定义函数：解释安全模式有 (1 or 2), 0 模式是不允许做任何工作。

uint8_t ble_gap_conn_sec_mode_t::sm

水平定义函数：介绍水平有(1, 2 or 3,4), 0 水平是不允许做任何工作。

uint8_t ble_gap_conn_sec_mode_t::lv

Level (1, 2 or 3,4), 0 for no permissions at all.

结构体体定义：

```
28. typedef struct
29. {
30.     uint8_t sm : 4;                    /**< Security Mode (1 or 2), 0 for no permissions at all. */
31.     uint8_t lv : 4;                    /**< Level (1, 2 or 3,4), 0 for no permissions at all. */
32. } ble_gap_conn_sec_mode_t;
```

这里就清楚了：

安全模式 0 水平 0：不允许连接
安全模式 1 水平 1：无安全要求 （又称为直接打开链接）
安全模式 1 水平 2：需要加密链接，无 MITM 保护
安全模式 1 水平 3：加密链接和 MITM 保护
安全模式 1 水平 4：LESC 加密链接和 MITM 保护
安全模式 2 水平 1：签名或者加密，无 MITM 保护
安全模式 2 水平 2：MITM 保护签名或者加密链接

BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode)选择了上面

的哪种模式了？

查看 nrf52832 相关函数定义：

#define BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(ptr) do {(ptr)->sm = 0; (ptr)->lv = 0;} while(0)

设置模式没有访问权限，0 模式 0 水平

#define BLE_GAP_CONN_SEC_MODE_SET_OPEN(ptr) do {(ptr)->sm = 1; (ptr)->lv = 1;} while(0)

设置为无需密码保护，开放性链接，1 模式 1 水平

#define BLE_GAP_CONN_SEC_MODE_SET_ENC_NO_MITM(ptr) do {(ptr)->sm = 1; (ptr)->lv = 2;} while(0)

需要加密链接，无 MITM 保护，1 模式 2 水平

#define BLE_GAP_CONN_SEC_MODE_SET_ENC_WITH_MITM(ptr) do {(ptr)->sm = 1; (ptr)->lv = 3;} while(0)

加密链接和 MITM 保护

#define BLE_GAP_CONN_SEC_MODE_SET_SIGNED_NO_MITM(ptr) do {(ptr)->sm = 2; (ptr)->lv = 1;} while(0)

签名或者加密，无 MITM 保护

#define BLE_GAP_CONN_SEC_MODE_SET_SIGNED_WITH_MITM(ptr) do {(ptr)->sm = 2; (ptr)->lv = 2;} while(0)

MITM 保护签名或者加密链接

我们选择了开放式链接，这个就不需要加任何密码就可以直接连接，也就是 1 模式 1 水平。

注意,安全模式设置上，蓝牙 5.0 和蓝牙 4.2 没有任何变动，描述一致。

3： 设备名称修改与外观设置

3.1 蓝牙设备名称设置

sd_ble_gap_device_name_set 是一个 SoftDevice API 函数，SoftDevice API 函数是经过封装后的函数，无法查看源函数，大家只要通过帮助文档查找函数意义，所有带 sd 前缀的函数名就是 SoftDevice API 函数。

```
01. err_code = sd_ble_gap_device_name_set(&sec_mode, (const uint8_t *) DEVICE_NAME,
02.                                       strlen(DEVICE_NAME));
```

对应该函数解释如下：

```
uint32_t
sd_ble_gap_device_name_set (  ble_gap_conn_sec_mode_tconst *const p_write_perm,
                              uint8_t const *const p_dev_name,
                              uint16_t len
                           )
```

设置 GAP 设备名称。参数定义

[in] p_write_perm 指定名称的设备写入安全特性

[in] p_dev_name 设备名称

[in] len 设备名称长度

返回：

NRF_SUCCESSGAP 设备名称和权限设置成功.

NRF_ERROR_INVALID_ADDR 提供的是无效的设备指针.

NRF_ERROR_INVALID_PARAM 无效的参数.

NRF_ERROR_DATA_SIZE 无效的数据大小.

这里的 DEVICE_NAME 应用的名称在手机扫描接收的时候显示信号时可以见到，大家可以直接修改自己的名称，符号自己的要求，比如设置 DEVICE_NAME 为：

```
#define DEVICE_NAME "LedButtonDemo"
```

手机扫描设备的时候显示如下：

GAP 中设置的广播名称会通过广播发出，在广播初始化中定义了一个广播参数的结构体，里面有一个定义广播名称的数据，如下所示：

```
init.advdata.name_type= BLE_ADVDATA_FULL_NAME
```

广播具体内容我们会在广播初始化中进行展开，这里谈一下广播名称的配置。广播名称分为三种类型：

```
03. typedef enum
04. {
05.     BLE_ADVDATA_NO_NAME,        /*广播数据中不包含广播名称 */
06.     BLE_ADVDATA_SHORT_NAME,     /*广播数据中包含简称. */
07.     BLE_ADVDATA_FULL_NAME       /*广播名称中包含全名 */
08. } ble_advdata_name_type_t;
```

SHORT_NAME 简称：也可以称为可选名称，字面意思上可以理解为名称的简写，当一个名称比较长的时候，由于广播的信息容量有限，要留空间放其他信息，因此没有那么多空间显示全明，因此 APP 显示的时候为简称。如果客户需要全名，可以通过 GAP 的设备名称读取函数读取。

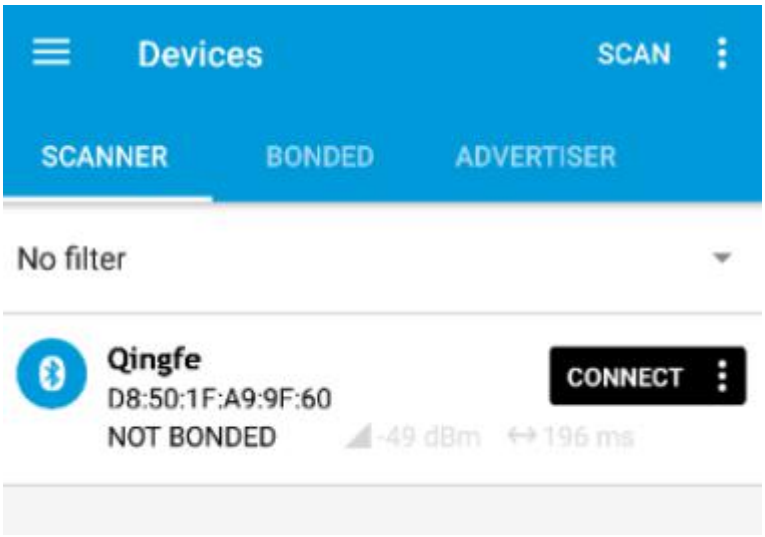
FULL_NAME 全名，如果名称不是很长，广播数据可以容纳，那你可以直接通过 GAP 名称设

置函数，设置全名在 APP 上显示，例子中的就是使用的全名显示。

下面我们来介绍下如何使用简称，打个比方，我把名称 Qingfengdianzhi_Nordic_Template 这个 31 字节长名称，如果使用全名只能显示一个确定的长度，如果使用简称可以自由设置显示的名称长度，但是不能超过全名的长度，测试了蓝牙样例里名称分配了 18 个字节长度空间给名称。如果超过这个长度，蓝牙就会出现广播超过长度的错误（这个在**错误调试码**那节教程讲过）。那么使用简称的代码如下：

```
static void advertising_init(void)
{
    ret_code_t             err_code;
    ble_advertising_init_t init;

    memset(&init, 0, sizeof(init));

    init.advdata.name_type               = BLE_ADVDATA_SHORT_NAME;
    init.advdata.short_name_len          = 6;
    init.advdata.include_appearance      = true;
    init.advdata.flags                   = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
    init.advdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    init.advdata.uuids_complete.p_uuids  = m_adv_uuids;

    init.config.ble_adv_fast_enabled  = true;
    init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;
    init.config.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS;

    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising, APP_BLE_CONN_CFG_TAG);
}
```

代码中名称类型改完：BLE_ADVDATA_SHORT_NAME，长度改为 6，那么我们显示这个 Qingfengdianzhi_Nordic_Template 长名称则在 APP 上只显示前 6 个名称字母，如下图所示：

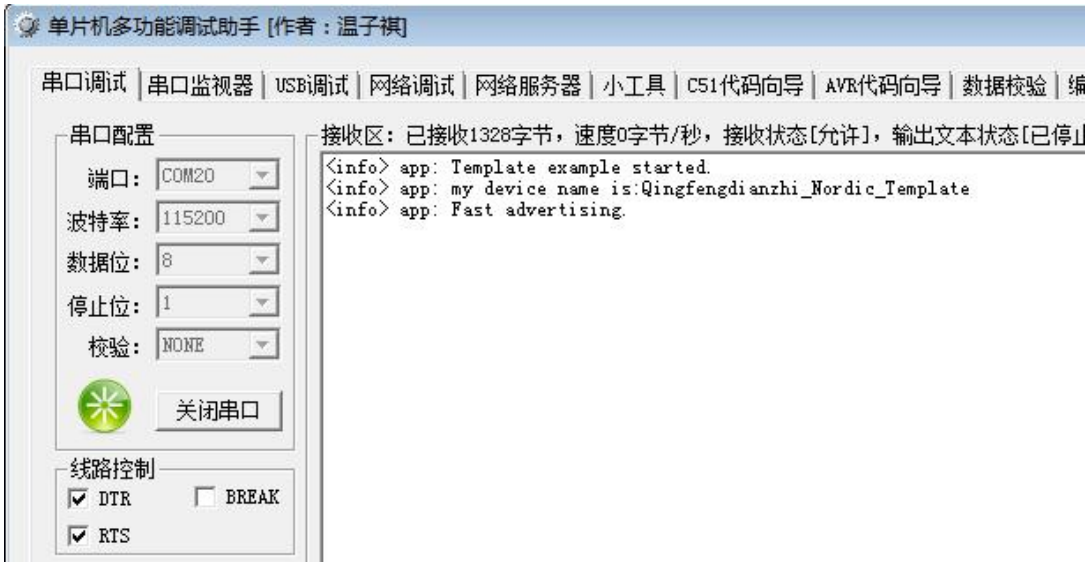


如果我们需要读取全名则可以在硬件中调用 GAP 名字获取函数：

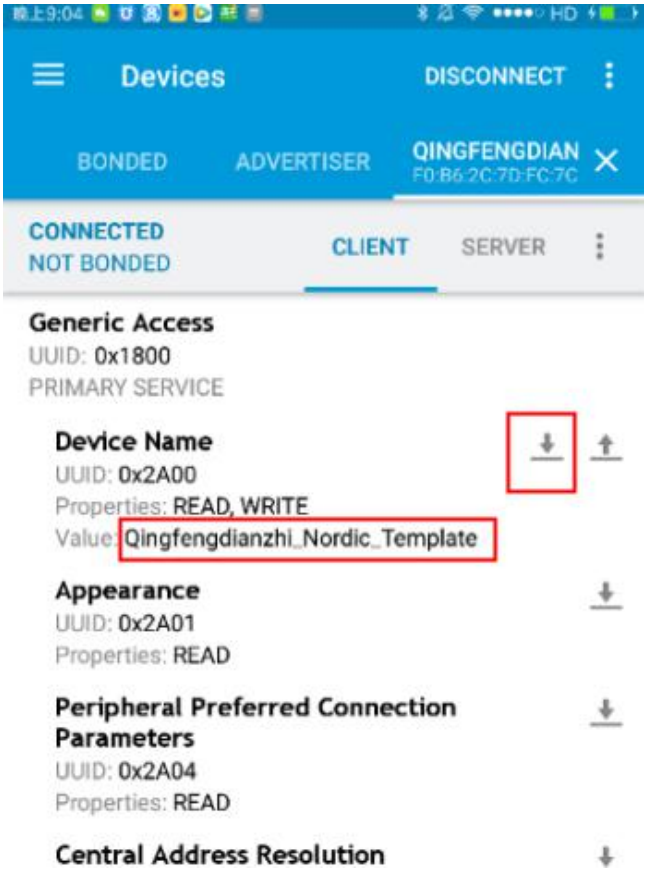
sd_ble_gap_device_name_get(uint8_t *p_dev_name, uint16_t *p_len)　该函数可以读取广播全名，然后通过任何方式展现，下面我们简单的通过串口输出来显示：

```
uint8_t shortmane[35];
    uint16_t mane_long=35;
    err_code=sd_ble_gap_device_name_get(shortmane,&mane_long);
```

```
if(err_code==NRF_SUCCESS)
{
    NRF_LOG_INFO("my device name is:%s",shortmane);
}
```



当然如果主机连接了从机，在 **Generic Access** 下也可以发动读属名称属性，读取全面，如下所示，关于 **Generic Access** 的读与写属性讲在后面远程修改名称讲义里讲到：



3.2 中文广播名称设置

很多设备需要使得它的名称为中文名称，如果直接做如下定义：

```
#define  DEVICE_NAME          "青风电子"
```

在主机上扫描的话会显示为乱码。这是出现乱码的原因是我们编程时候默认的编码格式和手机使用的编码格式不一样，手机 app 一般使用的编码格式为 UTF-8 格式，该格式支持中文编码，每个中文 3 个字节，而我们编译器中使用的编码格式中文为两个字节， 导致到手机 app 显示的时候乱码了，如果将我们的编码格式修改为 UTF-8 格式以广播形式广播出去，那么 app 就可以显示中文名字了。

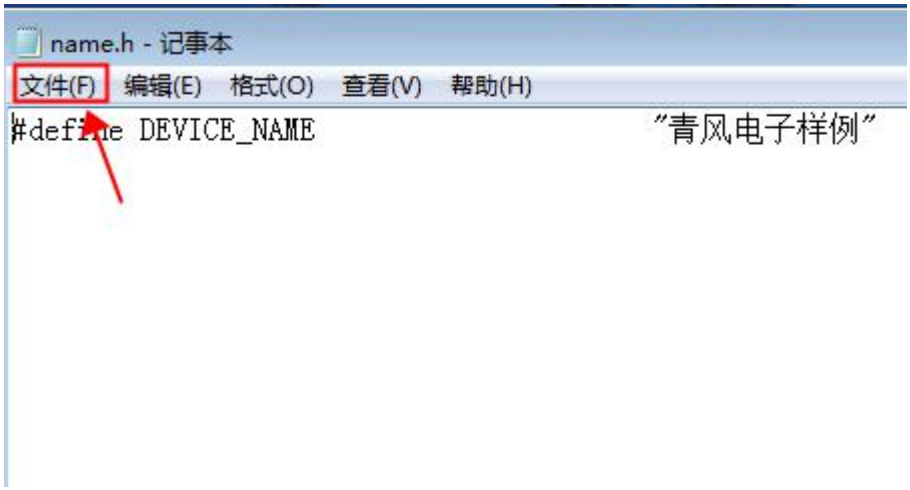
首先我们建立一个广播名字的文本，命名为 name.h，把改文件放到主函数文件同一个文件夹，如下图所示：

同时添加到主函数的头文件中，并且注释掉设备名称：

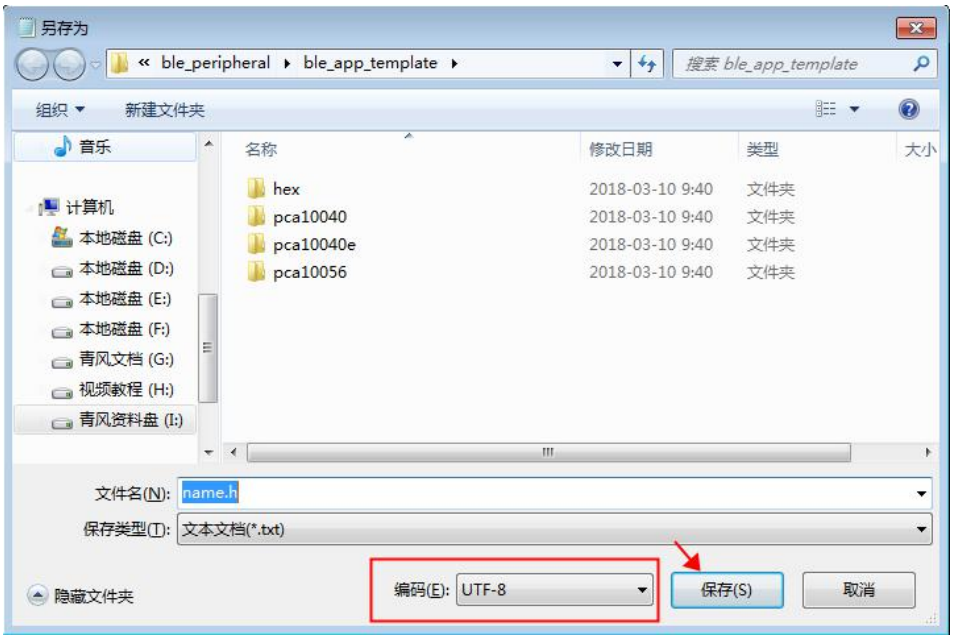
```
78  #include "nrf_log.h"
79  #include "nrf_log_ctrl.h"
80  #include "nrf_log_default_backends.h"
81  #include "name.h"
82
83  #define APP_FEATURE_NOT_SUPPORTED       BLE_GATT_STATUS_ATTERR_APP_BEGIN + 2
84
85  //#define DEVICE_NAME                     "Qingfengdianzhi_Nordic_Template"
86  #define MANUFACTURER_NAME               "NordicSemiconductor"
87  #define APP_ADV_INTERVAL                300
88  #define APP_ADV_TIMEOUT_IN_SECONDS      180
```

我们打开 name.h 头文件，在头文件中添加我们的中文广播名称的宏定义：

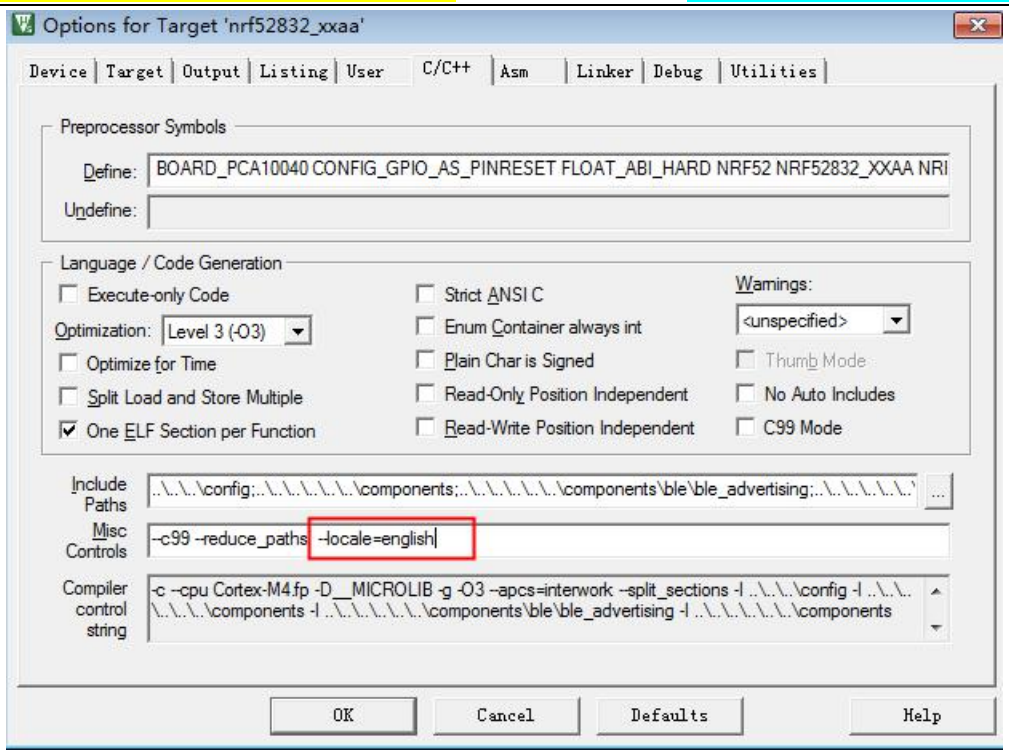
这样保存后，文件个编码格式任然会是 ANSI 模式，所以我们需要通过文本自带的格式存储方式改变文件的编码模式。一文本方式打开 name.h 文件，如下图所示：



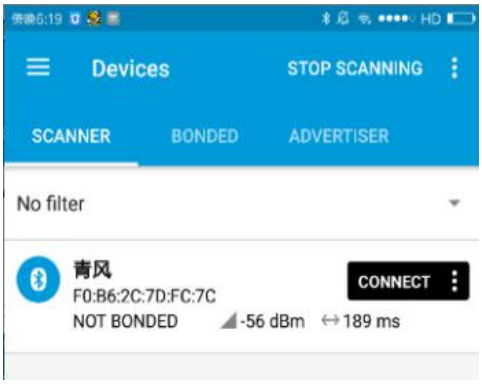
点击文件选项，选择另存为，目录不变，选择下面的编码方式为 UTF-8 点击保存，覆盖之前的文件：

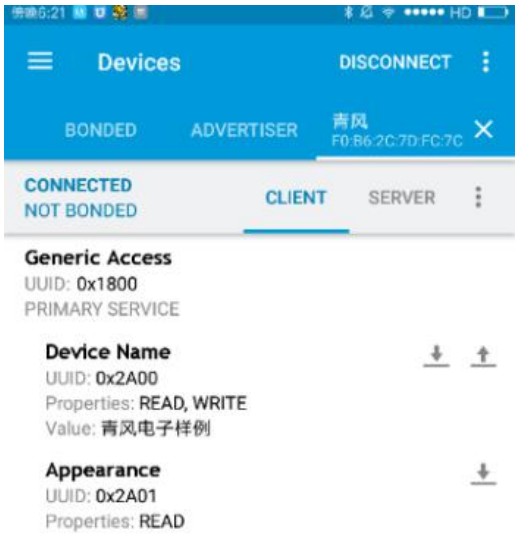


在编译设置框中的 C/C++位置设置的 Misc controls 内加入--locale=english,如下图所示



设置完所有的内容后，编译后下载，假设使用简称，则如下图所示：





或者使用全称，直接全部显示《青风电子样例》名称。

3.3 蓝牙应用图标的设置

2006 年，蓝牙 SIG 为了推广蓝牙应用，为了使蓝牙应用一目了然，对一些常用应用退出了应用图标功能。一般的智能系统可以直接识别图标符号，并且以图片的形式显示。具体的 SIG 到目前为止推出的图标应用定义如下链接所介绍：

https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.gap.appearance.xml

在 nrf52832 的工程文件 ble_types.h 文件中，对这些应用图标符号进行了宏定义，定义如下图所示：

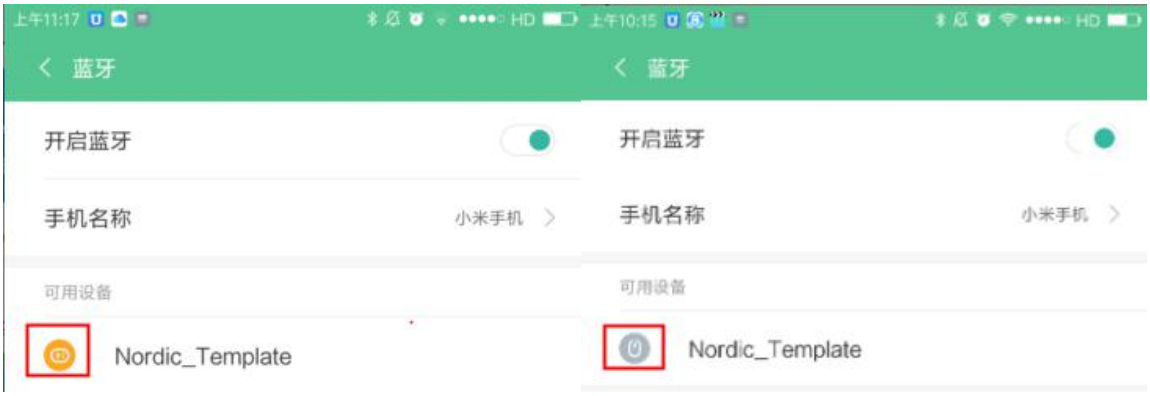
main.c　ble_types.h

```
102
103 /** @defgroup BLE_APPEARANCES Bluetooth Appearance values
104  *  @note Retrieved from http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u
105  * @{ */
106 #define BLE_APPEARANCE_UNKNOWN                                0 /**< Unknown. */
107 #define BLE_APPEARANCE_GENERIC_PHONE                         64 /**< Generic Phone. */
108 #define BLE_APPEARANCE_GENERIC_COMPUTER                     128 /**< Generic Computer. */
109 #define BLE_APPEARANCE_GENERIC_WATCH                        192 /**< Generic Watch. */
110 #define BLE_APPEARANCE_WATCH_SPORTS_WATCH                   193 /**< Watch: Sports Watch. */
111 #define BLE_APPEARANCE_GENERIC_CLOCK                        256 /**< Generic Clock. */
112 #define BLE_APPEARANCE_GENERIC_DISPLAY                      320 /**< Generic Display. */
113 #define BLE_APPEARANCE_GENERIC_REMOTE_CONTROL               384 /**< Generic Remote Control. */
114 #define BLE_APPEARANCE_GENERIC_EYE_GLASSES                  448 /**< Generic Eye-glasses. */
115 #define BLE_APPEARANCE_GENERIC_TAG                          512 /**< Generic Tag. */
116 #define BLE_APPEARANCE_GENERIC_KEYRING                      576 /**< Generic Keyring. */
117 #define BLE_APPEARANCE_GENERIC_MEDIA_PLAYER                 640 /**< Generic Media Player. */
118 #define BLE_APPEARANCE_GENERIC_BARCODE_SCANNER              704 /**< Generic Barcode Scanner. */
119 #define BLE_APPEARANCE_GENERIC_THERMOMETER                  768 /**< Generic Thermometer. */
120 #define BLE_APPEARANCE_THERMOMETER_EAR                      769 /**< Thermometer: Ear. */
121 #define BLE_APPEARANCE_GENERIC_HEART_RATE_SENSOR            832 /**< Generic Heart rate Sensor. */
122 #define BLE_APPEARANCE_HEART_RATE_SENSOR_HEART_RATE_BELT    833 /**< Heart Rate Sensor: Heart Rate Belt. */
123 #define BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE               896 /**< Generic Blood Pressure. */
124 #define BLE_APPEARANCE_BLOOD_PRESSURE_ARM                   897 /**< Blood Pressure: Arm. */
125 #define BLE_APPEARANCE_BLOOD_PRESSURE_WRIST                 898 /**< Blood Pressure: Wrist. */
126 #define BLE_APPEARANCE_GENERIC_HID                          960 /**< Human Interface Device (HID). */
127 #define BLE_APPEARANCE_HID_KEYBOARD                         961 /**< Keyboard (HID Subtype). */
128 #define BLE_APPEARANCE_HID_MOUSE                            962 /**< Mouse (HID Subtype). */
129 #define BLE_APPEARANCE_HID_JOYSTICK                         963 /**< Joystick (HID Subtype). */
130 #define BLE_APPEARANCE_HID_GAMEPAD                          964 /**< Gamepad (HID Subtype). */
131 #define BLE_APPEARANCE_HID_DIGITIZERSUBTYPE                 965 /**< Digitizer Tablet (HID Subtype). */
132 #define BLE_APPEARANCE_HID_CARD_READER                      966 /**< Card Reader (HID Subtype). */
133 #define BLE_APPEARANCE_HID_DIGITAL_PEN                      967 /**< Digital Pen (HID Subtype). */
134 #define BLE_APPEARANCE_HID_BARCODE                          968 /**< Barcode Scanner (HID Subtype). */
135 #define BLE_APPEARANCE_GENERIC_GLUCOSE_METER               1024 /**< Generic Glucose Meter. */
136 #define BLE_APPEARANCE_GENERIC_RUNNING_WALKING_SENSOR      1088 /**< Generic Running Walking Sensor. */
```

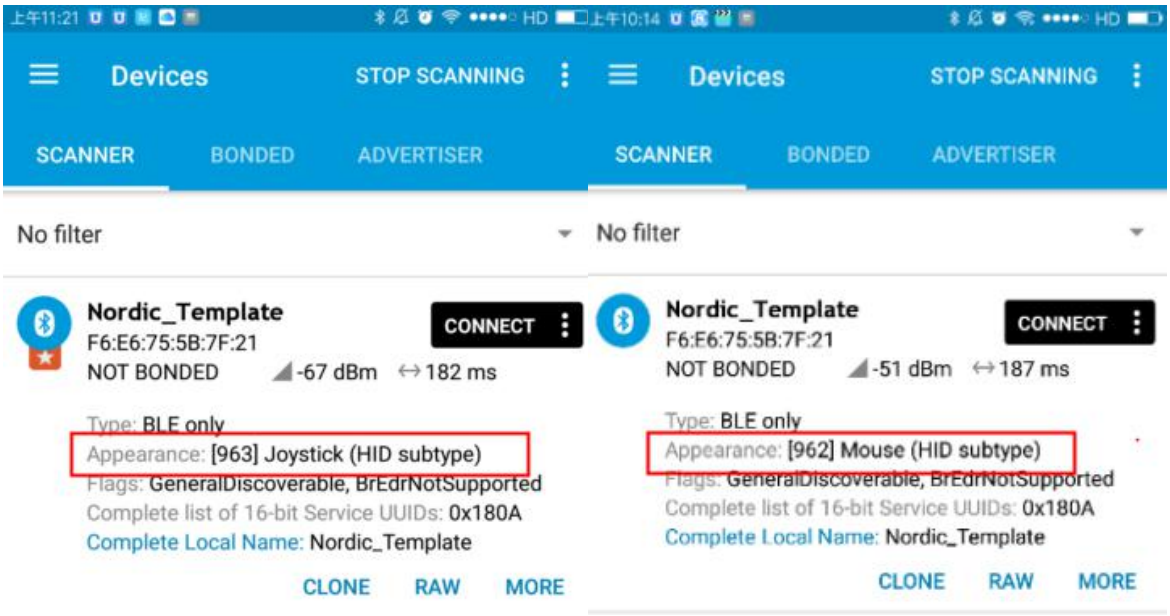
在蓝牙样例中，GAP 初始化的时候没有使用定义图标，如下所示，但是代码中以标注的形式给我们提示了一个应用函数 sd_ble_gap_appearance_set，这个函数就是来设置图标的函数，我们可以简单定义一个游戏手柄的图标，如下所示：

```
275 static void gap_params_init(void)
276 {
277     ret_code_t              err_code;
278     ble_gap_conn_params_t   gap_conn_params;
279     ble_gap_conn_sec_mode_t sec_mode;
280
281     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
282
283     err_code = sd_ble_gap_device_name_set(&sec_mode,
284                                           (const uint8_t *)DEVICE_NAME,
285                                           strlen(DEVICE_NAME));
286     APP_ERROR_CHECK(err_code);
287
288     /* YOUR_JOB: Use an appearance value matching the application's use case.
289        err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_);
290        APP_ERROR_CHECK(err_code); */
291
292     err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_HID_JOYSTICK);
293     APP_ERROR_CHECK(err_code);
294
295     memset(&gap_conn_params, 0, sizeof(gap_conn_params));
296
297     gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
298     gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
299     gap_conn_params.slave_latency     = SLAVE_LATENCY;
300     gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
301
302     err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
303     APP_ERROR_CHECK(err_code);
304 }
```

修改编译下载后，下载到开发板。开发板开始广播，然后打开手机设置中--蓝牙--搜索应用，会搜索显示出一个手柄图标，如下图左边所示。如果修改成鼠标应用图标，修改后编译下载，同样打开手机设置中--蓝牙--搜索应用，会搜索显示出一个鼠标图标，如下图右边所示。对比如下图：



打开 nrf connect app，观察两种情况下广播中 Appearance 中的内容，可以发现广播数据信息里是包含了对应图标的符号码的，如下图所示，也就是说 gap Appearance 的值是以广播形式发出的，和蓝牙名称同样的包含在广播数据中。



4： GAP 初始化设置连接间隔

在一个典型的蓝牙系统中，外围设备发送具体的广告数据让任何中央设备知道他是一个可连接的设备。广告内容包含设备地址，还可以包含一些额外的额数据，比如设备名称。中央设备接收到广告后发送一个搜索请求给外围设备，外围设备答复一个搜索答复。这就是设备发现的过程。

这样中央设备就知道外围设备是一个可连接的设备。中央设备可以发送一个建立连接的请求给外围设备，发生主机和从机设备的互联。那么一个连接情况包含下面的连接参数：

连接间隔	在一个 BLE 连接中跳频机制需要被使用，这样两个设备之间可以在一个特定的通道上进行数据收发，在一个特定的时间之后会跳到一个新的通道上，LL 层负责通道切换。

	这个遇见设备收发数据被称作是连接事件。尽管没有应用程序数据需要收发，两个设备之间仍然会交换链路层数据来保持连接。连接间隔是两个连接事件之间的时间，使用一个单元值为 1.25ms 的步进。连接间隔从最小值 6（7.5ms）到最大值 3200（4.0s）。 不同的应用也许需要不同的连接间隔，一个长时间的连接间隔将会节约更多的能量，因为设备可以在两个连接事件之间睡眠更长的时间。但是他会导致数据发送不及时，如果有数据要发送那么他只能够在下一次连接事件到来时才能被发送。
从机潜伏周期	这个参数描述了从机跳过连接事件的次数。这使外围设备具有一定的灵活性，如果它不具有任何数据传送，它可以选择跳过连接事件，并保持睡眠，从而提供了一些积蓄力量。这一决定取决于外围设备。 也可以成为从设备延迟，从设备能够忽略主设备的链接事件的最大值。比如：从设备延迟为 6 的话，那么从设备每隔 6 个锚点可以监听到主设备发送来的数据包，如果从设备延迟为 0，从设备在每个锚点都可以监听主设备的数据包。
监督超时	这是两个成功的连接事件之间间隔的最大值。如果超过这个时间还未出现成功的连接事件，那么设备将会考虑失去连接，返回一个未连接状态。这个参数值使用 10ms 的步进。监督超时时间从最小 10（100ms）到最大 3200（32.0s）。同时超时时间必须大于有效连接事件。 有效连接事件时间 = 连接间隔 X （1 + 从机延迟值）

短连接间隔	高功耗，高数据吞吐量，发送等待时间短
长连接间隔	低功耗，低数据吞吐量，发送等待时间长
低或者 0 潜伏值	从机在没有数据发送的情况下高功耗，从机可以快速的收到主机的数据
高潜伏值	从机在没有数据发送的情况下可以低功耗；从机无法及时收到主机的数据，但主机能及时收到从机的数据

BLE 协议栈的 GAP 层负责处理设备的接入方式和过程，包括设备发现，链路建立，链路终止，启动安全功能，设备配置。 GAP 层通常扮演以下四种角色中的一种：

广播者	**广告发送者，不是可连接的设备**

观察者	扫描广告，不能够启动连接
外围设备	广告发送者，是可连接的设备，在单一链路层连接时作为一个从机
中央设备	扫描广告启动连接，在单一或者多链路层连接时作为主机，支持三个同时连接

在某些情况下，中央设备请求与外围设备建立连接包含连接参数对外围设备而言是不利的。在其他情况下，外围设备可以在连接过程中改变连接参数这个取决于外围设备的应用程序。外围设备可以请求中央设备改变连接参数通过设置连接参数更新请求。这个请求是被协议栈的 L2CAP 处理的。

这个请求包含刚才讲的四个参数：**最小连接间隔、最大连接间隔、从机潜伏周期、连接超时时间**。这些值代表了外围设备针对连接的期望参数，连接间隔是以范围的形式提供的。当中央设备接收到这个请求,他有权利决定是接受还是拒绝这些参数。

在 nrf52832 的代码中提供了如下结构体表述上面四个参数：

```
/*
 * 接收事件时，下面的参数会在被接收时起作用
 */
typedef struct
{
    uint16_t min_conn_interval;          /**< 最小的连接间隔 in 1.25 ms units,参数 @ref BLE_GAP_CP_LIMITS.*/
    uint16_t max_conn_interval;          /**< 最大的连接间隔 in 1.25 ms units, see @ref BLE_GAP_CP_LIMITS.*/
    uint16_t slave_latency;              /**< 在连接事件中从设备潜伏周期, see @ref BLE_GAP_CP_LIMITS.*/
    uint16_t conn_sup_timeout;           /**< 连接超时监督 in 10 ms units, see @ref BLE_GAP_CP_LIMITS.*/
} ble_gap_conn_params_t;
```

改参数在 GAP 初始化的 SoftDevice API 函数使用了如下：

```
err_code = sd_ble_gap_ppcp_set(&gap_conn_params);//连接参数设置，主要是时间间隔设置
```

uint32_t sd_ble_gap_ppcp_set (ble_gap_conn_params_t const *const *p_conn_params*)

设置 GAP 外围首选的连接参数。

参数 [in] p_conn_params Pointer to a ble_gap_conn_params_t structure with the desired parameters.

Returns

NRF_SUCCESS 外围首选连接参数设置成功.

NRF_ERROR_INVALID_ADDR 无效的指针提供.

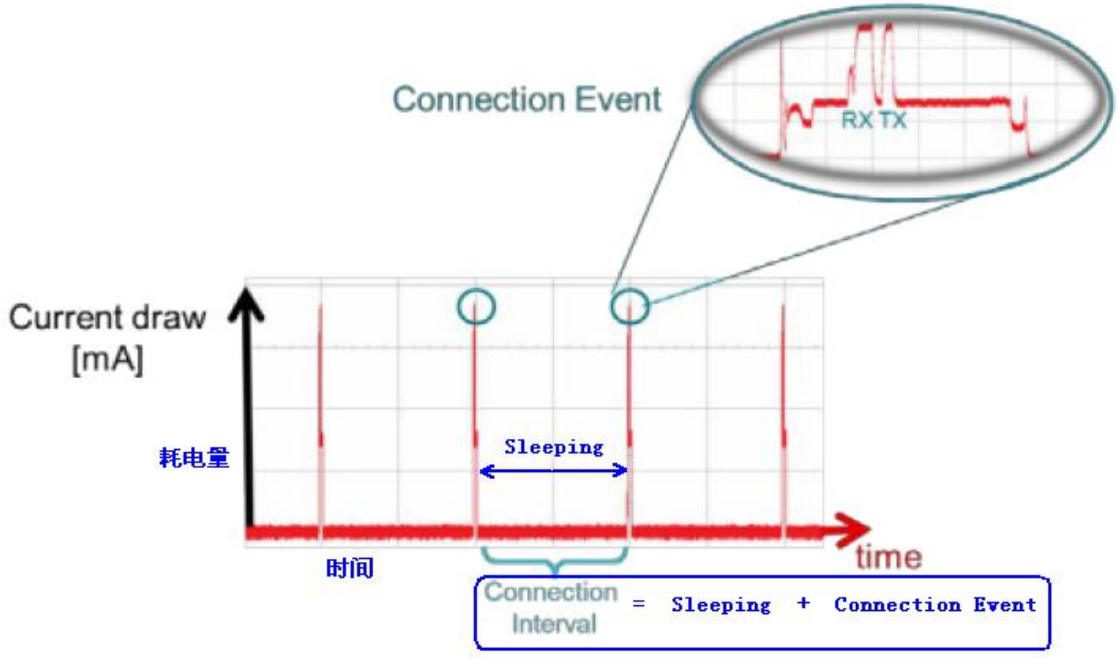
NRF_ERROR_INVALID_PARAM 无效的参数提供.

比如在蓝牙串口例子中，我们在主函数中设置连接参数如下所示：

```
01. #define MIN_CONN_INTERVAL                MSEC_TO_UNITS(20, UNIT_1_25_MS)
    /**< 最小的连接间隔 (20m seconds). */
02. #define MAX_CONN_INTERVAL                MSEC_TO_UNITS(75, UNIT_1_25_MS)
    /**<最大的连接间隔 (75m second). */
03. #define SLAVE_LATENCY                    0
    /**< Slave latency. */
04. #define CONN_SUP_TIMEOUT                 MSEC_TO_UNITS(4000, UNIT_10_MS)
    /**< Connection supervisory timeout (4 seconds). */
```

下面结合具体情况进行分析：

最小连接间隔、最大连接间隔：



尖刺的波就是连接事件（Connection events），剩下的 Sleeping 是睡眠时间，设备在建立连接之后的大多数时间都是处于 Sleeping，这种情况下耗电量比较低，而在连接事件（Connection events）中，耗电量就相对高很多，这也是 BLE 为什么省电的原因之一。

Connection Interval 连接间隔，在 BLE 的两个设备的连接中使用跳频机制。两个设备使用特定的信道发送和接收数据，然后过一段时间后再使用新的信道（BLE 协议栈的链路层处理信道的切换）。两个设备在切换信道后发送和接收数据称为一个连接事件。尽管没有应用数据被发送和接收，两个设备仍旧会交换链路层数据（空包 Empty PDU）来维持连接每个连接事件（Connection events）中，都需要由 Master 发起包，再由 Slave 回复。

Master 即主机，简称 M；Slave 即从机，简称 S。抓包过程中看到的 M->S 或者 S->M 即主机到从机或者从机到主机。

比如：大数据传递时：通讯数据包是连续传递的，主机会选择 min 值来进行通讯。当然 Min 和 max 的值都设置大小。对比蓝牙样例和蓝牙串口例子例 max 和 min 值的设置：

	串口蓝牙例子	蓝牙样例
Max 时间	75ms	200ms
Min 时间	20ms	100ms

注意，刚开始连接的时候的连接参数是主机默认给的连接时间，这个时间设置的时间是从机参数更新的时间，参数更新里我们要详细讲的。

无数据传递时：通讯是空闲状态，主机会选择 max 值来定期询问从机状态，以保持连接不中断。我们通过抓捕器抓包官方蓝牙串口例子例连接事件发生过程：

抓包如下所示，在没有数据的情况下，抓包过程中看到的 M->S 或者 S->M 即主机到从机或者从机到主机，连接间隔，在 BLE 的两个设备的连接中使用跳频机制，通过 0x08 通道调到 0x22 通道，主机给的默认连接事件间隔大概是 44ms，数据包为 Empty PDU。

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					CRC	RSSI (dBm)	FCS
							LLID	NESN	SN	MD	PDU-Length			
694	+44770 =27408608	0x08	0x66A1D3A3	M->S	OK	Empty PDU	1	0	0	0	0	0x6395F3	-44	OK
695	+230 =27408838	0x08	0x66A1D3A3	S->M	OK	Empty PDU	1	1	0	0	0	0x639320	-46	OK
696	+44771 =27453609	0x22	0x66A1D3A3	M->S	OK	Empty PDU	1	1	1	0	0	0x639E86	-54	OK
697	+230 =27453839	0x22	0x66A1D3A3	S->M	OK	Empty PDU	1	0	1	0	0	0x639855	-46	OK

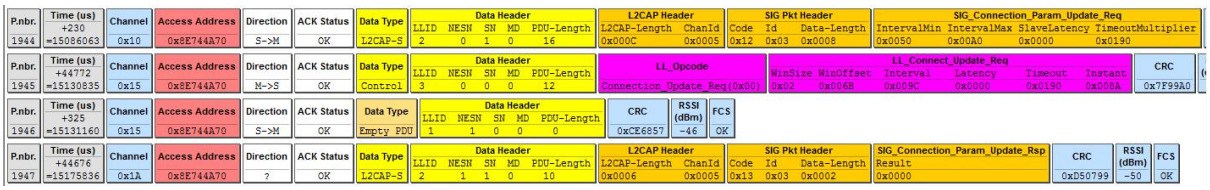
当到了申请更新的时间后，当从机向主机发送一个连接参数更新请求后，连接间隔会对应我们之前的设置，主机按照请求改变事件间隔。如图所示变成了 75ms。这个连接间隔时间主要取决于主机分配的时间，在设置的 min 和 max 时间范围内：

蓝牙串口的参数更新后状态：

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		SIG Pkt Header			SIG_Connection_Param_Update_Req				CRC	RSSI (dBm)	FCS
							LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Code	Id	Data-Length	IntervalMin	IntervalMax	SlaveLatency	TimeoutMultiplier			
863	+229 =23777346	0x1E	0x3396BB26	S->M	OK	L2CAP-S	2	1	0	0	16	0x000C	0x0005	0x12	0x03	0x0008	0x0010	0x003C	0x0000	0x0190			
864	+44773 =23822119	0x0F	0x3396BB26	M->S	OK	Control	3	1	1	0	12	LL_Opcode		WinSize	WinOffset	Interval	Latency	Timeout	Instant		CRC		
												Connection_Update_Req(0x00)		0x02	0x000B	0x003C	0x0000	0x0190	0x009C		0x5EDDC0		
865	+326 =23822445	0x0F	0x3396BB26	S->M	OK	Empty PDU	1	0	1	0	0										0x078F56	-40	OK
866	+44675 =23867120	0x16	0x3396BB26	?	OK	L2CAP-S	2	0	0	0	10	L2CAP-Length	ChanId	Code	Id	Data-Length	Result				CRC	RSSI (dBm)	FCS
												0x0006	0x0005	0x13	0x03	0x0002	0x0000				0x80B8E2	-42	OK

P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					CRC	RSSI (dBm)	FCS
							LLID	NESN	SN	MD	PDU-Length			
1432	+44772 =15338898	0x0B	0xBD4840D2	?	OK	Empty PDU	1	1	1	0	0	0x36208D	-36	OK
1433	+229 =15339127	0x0B	0xBD4840D2	?	OK	Empty PDU	1	0	1	0	0	0x36265E	-39	OK
1434	+44773 =15383900	0x11	0xBD4840D2	?	OK	Empty PDU	1	0	0	0	0	0x362BF8	-35	OK
1435	+229 =15384129	0x11	0xBD4840D2	?	OK	Empty PDU	1	1	0	0	0	0x362D2B	-39	OK
1436	+59773 =15443902	0x17	0xBD4840D2	?	OK	Empty PDU	1	1	1	0	0	0x36208D	-36	OK
1437	+230 =15444132	0x17	0xBD4840D2	?	OK	Empty PDU	1	0	1	0	0	0x36265E	-38	OK
1438	+74773 =15518905	0x04	0xBD4840D2	?	OK	Empty PDU	1	0	0	0	0	0x362BF8	-36	OK
1439	+230 =15519135	0x04	0xBD4840D2	?	OK	Empty PDU	1	1	0	0	0	0x362D2B	-38	OK

蓝牙样例的参数更新后状态，变成 180ms：



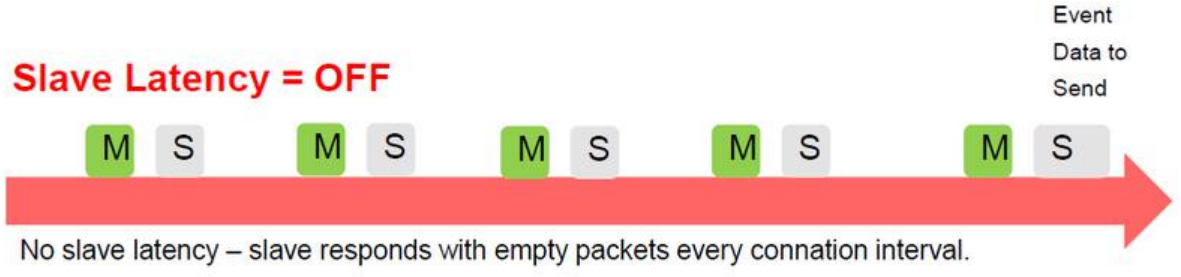
P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					L2CAP Header		SIG Pkt Header			SIG_Connection_Param_Update_Req					CRC	RSSI (dBm)	FCS
							LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Code	Id	Data-Length	IntervalMin	IntervalMax	SlaveLatency	TimeoutMultiplier				
1944	+230 =15086063	0x10	0x8E744A70	S->M	OK	L2CAP-S	2	0	1	0	16	0x000C	0x0005	0x12	0x03	0x0008	0x0050	0x00A0	0x0000	0x0190				
1945	+44772 =15130835	0x15	0x8E744A70	M->S	OK	Control	LLID	NESN	SN	MD	PDU-Length	LL_Opcode		LL_Connect_Update_Req								CRC		
							3	0	0	0	12	Connection_Update_Req(0x00)		WinSize	WinOffset	Interval	Latency	Timeout	Instant		0x7F99A0		
1946	+325 =15131160	0x15	0x8E744A70	S->M	OK	Empty PDU	LLID	NESN	SN	MD	PDU-Length											CRC	RSSI (dBm)	FCS
							1	1	0	0	0											0xCE6857	-46	OK
1947	+44676 =15175836	0x1A	0x8E744A70	?	OK	L2CAP-S	LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Code	Id	Data-Length	SIG_Connection_Param_Update_Rsp					CRC	RSSI (dBm)	FCS
							2	1	1	0	10	0x0006	0x0005	0x13	0x03	0x0002	Result					0xD50799	-50	OK

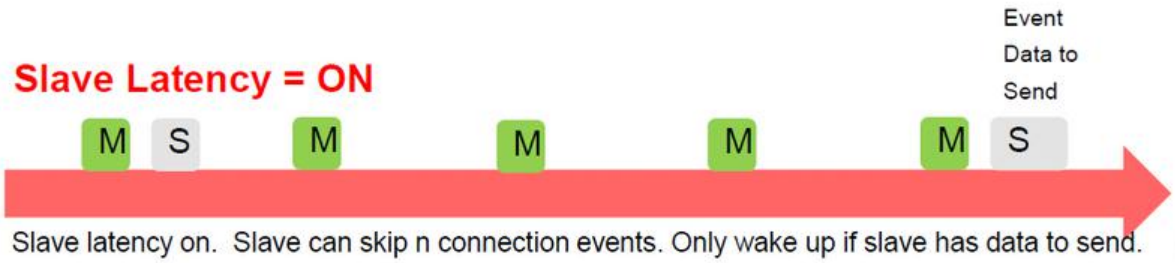
P.nbr.	Time (us)	Channel	Access Address	Direction	ACK Status	Data Type	Data Header					CRC	RSSI (dBm)	FCS
							LLID	NESN	SN	MD	PDU-Length			
1961	+44771 =15490847	0x18	0x8E744A70	?	OK	Empty PDU	1	0	0	0	0	0xCE6E84	-48	OK
1962	+230 =15491077	0x18	0x8E744A70	?	OK	Empty PDU	1	1	0	0	0	0xCE6857	-47	OK
1963	+179777 =15670854	0x1D	0x8E744A70	?	OK	Empty PDU	1	1	1	0	0	0xCE65F1	-48	OK
1964	+230 =15671084	0x1D	0x8E744A70	?	OK	Empty PDU	1	0	1	0	0	0xCE6322	-48	OK

从机潜伏周期:

Slaver 设备没有数据要发时，跳过一定数目的 ConnectionEvent 的值，Rang：0-499. 跳过的 interval 个数，设置为 0 则每次 interval 都连接。数字约小通信速度越快，但是功耗越高。注意：这个参数更重要的应用 ：比如在距离远或者干扰大的时候无数据通信的时候。把这个值设大可以减少掉线概率。

比如防丢器：放在口袋里面假如这个参数设置为 0 表示规定时间内必须响应从机，不然就以为是蓝牙断开了。假如设置为 3。假如信号不好的时候，即使中间丢了 3 个，只接受了 1 个就表示连接了。因为他会跳过其中 3 个，保证了不掉线的概率。如下所示，对比下 0 和 3 的设置：





允许 Slave（从设备）在没有数据要发的情况下，跳过一定数目的连接事件（Connection events），在这些连接事件（Connection events）中不必回复 Master（主设备）的包，这样就能更加省电。范围可以是 0～499。

连接超时：

超时时间，就是两个设备在连接的这段时间没有发生通讯而导致连接自动断开的值。Range（10ms-----32s）连接超时时间，用在信号不太好的情况下, 给对方一点时间。超过这个时间通信就建立失败。

connTimeout: (2 octets)

Range	Parameter Description
10 - 3200	Supervision timeout for the LE Link. Range: 0x000A to 0x0C80 Time = *connTimeout* * 10 msec Time Range: 100 msec to 32 seconds

连接间隔、从机时延以及超时时间这三者必须满足如下公式：

Supervision Timeout ＞（1 +slaveLatency）*（connectionInterval）

上述公式必须满足，否则连接就会不正常断开。

这三个连接参数不同情况下对通信速率和功耗的影响：

1.Connection Interval 缩短，Master 和 Slave 通信更加频繁，提高数据吞吐速度，缩短了数据发送的时间，当然也增加了功耗。

2.Connection Interval 增长，通信频率降低，数据吞吐速度降低，增加了数据发送的时间，当然，这种设置降低了功耗。

3.Slave Latency 减少或者设置为 0，每次 Connection Events 中都需要回复 Master 的包，功耗会上升，数据发送速度会提高。

4.Slave Latency 加长，功耗下降，数据发送速度降低。

蓝牙协议栈决定就是主机决定连接参数的值(connection interval, slave latency, timeout)，从机可以请求更新这些参数，主机决定是不是接受，接受的值是多少。所以是会出现手机接受参数后和从机请求的参数有偏差，或者甚至是拒绝（ios）。android 和 ios 都是在手机和设备建立连接时就会默认设置这些参数，app 开发是无法修改这些参数的，这些默认参数由手机厂商决定。

上面的设置完成了 GAP 初始化，注意 GAP 实际上定义了蓝牙设备的基本需求，包括广播功能,广播名称设置，连接加密等级等功能。