BP15 系列 SDK
蓝牙应用开发说明文档
V1.4

版本记录

版本	作者	日期	修改日志
V1.4	KK	2024-10-16	1、增加外接 PA/LNA 芯片配置说明； 2、修改 BLE 的相关介绍；
V1.3	KK	2024-10-8	1、新增蓝牙异常情况问题排查手段；
V1.2	KK	2024-4-30	1、新增设备类型 COD 介绍； 2、完善 BLE 数据收发注意事项； 3、SPP、OBEX、自定义 UUID 服务等协议的说明 4、更新通过 AVRCP 获取歌词说明；
V1.1	Cole	2024-4-19	1、增加 BLE 数据发送说明；
V1.0	KK	2024-2-5	1、增加蓝牙相关版本信息说明；
V0.3	KK	2024-1-25	1、完善蓝牙参数说明；
V0.2	KK	2024-1-24	1、完善蓝牙通话说明； 2、修改描述信息；
V0.1	KK	2024-1-22	初版发布；

Note：此文档适用于 BP15 的 MVsB5_BT_Audio_SDK 版本指导说明；
其他版本 SDK 中有个别差异，如有疑问请联系 FAE；

目录

1. BP15 蓝牙协议版本说明

1. 蓝牙核心协议版本：V5.3
2. 蓝牙 Profile 版本支持如下：

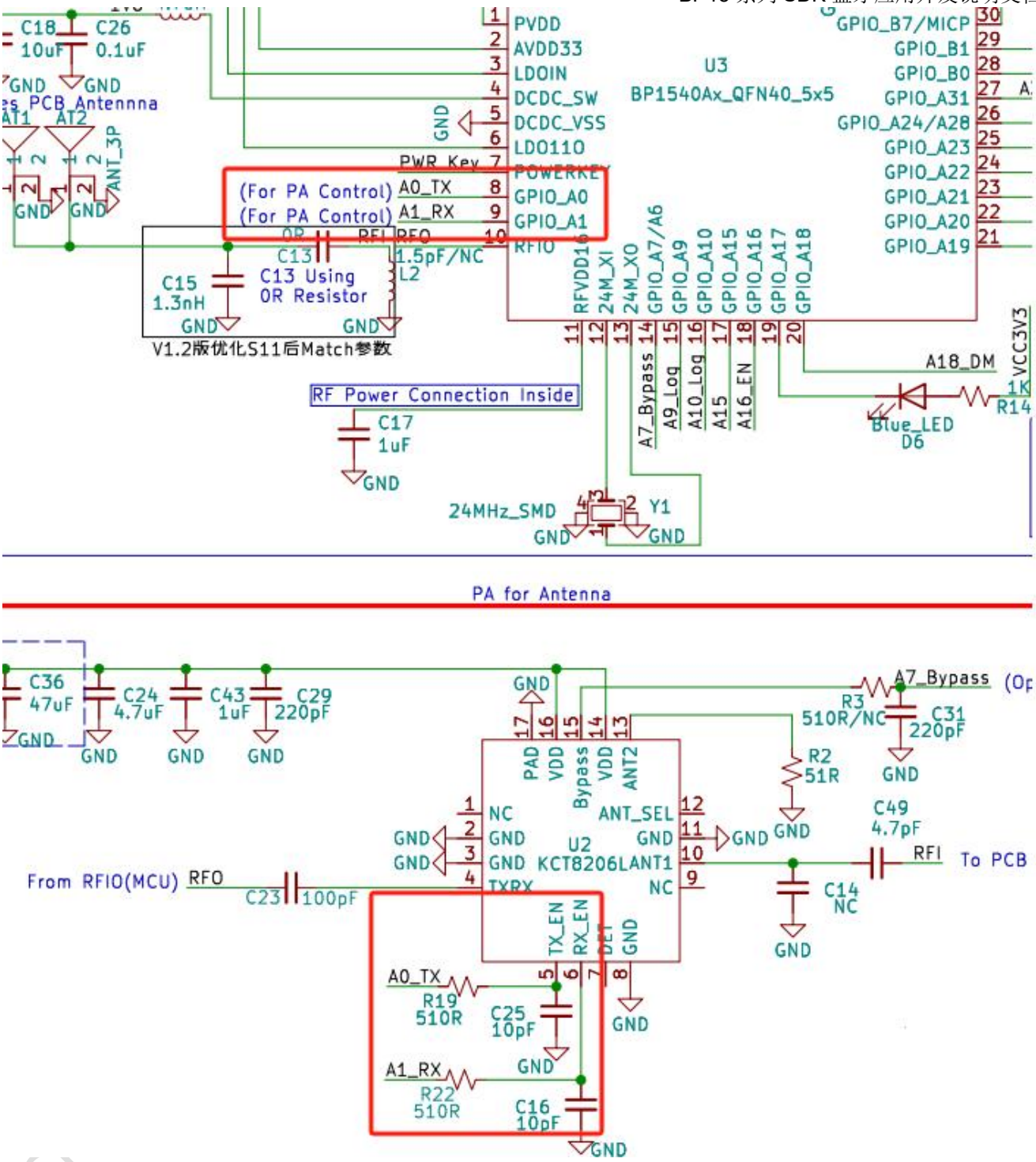
Profile	Role	Version
A2DP	source	V1.3.2
	sink	V1.3.2
AVCTP	controller	V1.4
	target	V1.4
AVDTP	source	V1.3
	sink	V1.3
AVRCP	controller	V1.6.2
	target	V1.6.2
GAVDP	initiator	V1.3
	acceptor	V1.3
HFP	AG	V1.8
	HF	V1.8
HID	N/A	V1.1.1
HSP	AG	not support
	HS	V1.2
OPP	client	V1.2.1
	server	V1.2.1
PBAP	PCE	V1.2.3
	PSE	not support
RFCOMM	N/A	V1.2
SPP	N/A	V1.2

2. 外接 PA/LNA 芯片配置

1. 在使用外挂 PA/LNA 芯片时，需要预留 GPIO-A0/A1 作为 TX/RX 的控制 IO；
2. 在初始化过程中，需要优先配置 IO，调用函数：`enable_pa_lna(1);`

申明函数定义：`extern void enable_pa_lna(uint8_t pa_lna_enable);`

3. 硬件接线方式，请参考如下原理图：



4. 使用外挂 PA/LNA 芯片时，需要调整 BP15 的蓝牙发射功率；具体请参考 PA/LNA 芯片规格书。

3. 蓝牙参数配置

1. 蓝牙功能开关 CFG_APP_BT_MODE_EN 在 app_config.h 文件中；
2. 基带 EM 空间的分配参数配置在 bt_em_config.h 文件中；
3. 蓝牙应用功能的配置，蓝牙相关参数配置数据在 bt_config.h 文件中；
4. 蓝牙常用功能参数，结构体定义在 sys_param.c 文件中，参数的定义在 bt_config.h 文件中；

4. 经典蓝牙地址获取方式

1. 上电后从 flash 中指定位置读取蓝牙地址信息（最后 4K）；
2. 如 flash 无数据，从芯片的 efuse 中读取 chip_id 生成蓝牙地址；
3. 如 efuse 数据为空，则通过内置 flash 芯片的 chip_id 生成蓝牙地址，然后保存到 flash；(该方式基本不会使用，只有在前期芯片没有烧录 efuse 信息的时候才有效，后续的芯片必然会烧录 efuse，存在 chip ID 信息)

注：用户可以不遵守如上的规则，自行定义蓝牙地址；

4. 蓝牙地址的顺序说明：
   (1) 客户烧录的蓝牙地址的顺序是： **NAP-UAP-LAP**；即保存到 flash 的顺序；
   (2) flash 内保存的蓝牙地址顺序： NAP-UAP-LAP，同时手机和其他外设上看到顺序都是这个顺序；
   (3) 系统内部蓝牙在使用时，地址的顺序为：LAP-UAP-NAP，包含蓝牙连接手机成功后，回调反馈的蓝牙顺序，都是此顺序；

注：此顺序和 flash 保存的顺序是反序的；

5. 蓝牙发射功率配置

在 bt_config.h 中，配置 BT_TX_POWER_LEVEL 参数来调整发射功率；

1. 发射功率默认参数为 level23(+7db -- max)；
2. 发射功率按照 1db 递减；
3. 发射功率过大，可能会导致蓝牙回连会有干扰声的产生；

6. 蓝牙相关数据保存位置

在 bt_app_ddb_info.h 文件中，数据保存位置信息，具体说明如下：

1. 蓝牙地址，BLE 地址，频偏参数蓝牙相关配置参数保存在 flash 的最后 4K 位置，即：（flash_table_info.flash_capacity - BT_CONFIG_OFFSET）；该区域受 download 保护；
2. 蓝牙的配对记录区域地址获取方式：get_bp_data_addr ()； (共 8K)
3. 蓝牙的参数配置：如蓝牙名称、BLE 名称、蓝牙回连次数等信息，保存区域地址获取方式：get_bt_config_addr()；（共 4K）
4. 清除配对记录：调用 BtDdb_Erase 函数可以清除配对记录，并不会再发起回连；

7. 系统参量化数据说明

1. 参量化数据来源于两个地方：SDK 默认值和 flash 中保存的值。
   - SDK 默认值来源： sys_param.c 文件 default_parameter 变量。

```
static const SYS_PARAMETER default_parameter =
{
    .bt_LocalDeviceName      = BT_NAME,
    .ble_LocalDeviceName     = BLE_NAME,
    .bt_TxPowerLevel         = BT_TX_POWER_LEVEL,
    .bt_PagePowerLevel       = BT_PAGE_TX_POWER_LEVEL,
    .BtTrim                  = BT_DEFAULT_TRIM,
    .TwsVolSyncEnable        = TRUE, //主从之间音量控制同步
    .bt_CallinRingType       = SYS_DEFAULT_RING_TYPE,
    .bt_BackgroundType       = SYS_BT_BACKGROUND_TYPE,
    .bt_SimplePairingEnable  = BT_SIMPLEPAIRING_FLAG,
    .bt_PinCode              = BT_PINCODE,
    .bt_ReconnectionEnable           = TRUE,
    .bt_ReconnectionTryCounts        = 5,
    .bt_ReconnectionInternalTime     = 3,
    .bt_BBLostReconnectionEnable     = TRUE,
    .bt_BBLostTryCounts              = 90,
    .bt_BBLostInternalTime           = 5,
    .bt_TwsReconnectionEnable        = TRUE,
    .bt_TwsReconnectionTryCounts     = 3,
    .bt_TwsReconnectionInternalTime  = 3,
    .bt_TwsBBLostReconnectionEnable  = TRUE,
    .bt_TwsBBLostTryCounts           = 3,
    .bt_TwsBBLostInternalTime        = 5,
    .bt_TwsConnectedWhenActiveDisconSupport = FALSE,
    .bt_TwsPairingWhenPhoneConnectedSupport = TRUE,
};
```

   - Flash 中保存的值：SDK 编译阶段由 flash_param.c 生成 flash_param.bin，并且打包到 MVA 文件里面。可以通过 PC 工具 OCPWorkBench 在线和离线修改。也可以在编译阶段直接在 flash_param.c 文件对 SysDefaultParm 进行修改。

```
const FLASH_PARAMETER SysDefaultParm =
{
    .SysVer = {SYS_PARA_VER_ID,sizeof(SysDefaultParm.SysVer.name),
            "Ver 1.0.0"},  //版本

    .BtName = {BT_PARA_BT_NAME_ID,sizeof(SysDefaultParm.BtName.name),
            BT_NAME},   //蓝牙名称
    .BleName = {BT_PARA_BLE_NAME_ID,sizeof(SysDefaultParm.BleName.name),
            BLE_NAME},  //BLE蓝牙名称
    .bt_TxPowerLevel = {BT_PARA_RF_TX_LEVEL_ID,sizeof(SysDefaultParm.bt_TxPowerLevel.para_val),
            BT_TX_POWER_LEVEL},//蓝牙正常工作时发射功率
    .bt_PagePowerLevel = {BT_PARA_RF_PAGE_LEVEL_ID,sizeof(SysDefaultParm.bt_PagePowerLevel.para_val),
            BT_PAGE_TX_POWER_LEVEL},//蓝牙回连发射功率
```

2. 参量化数据使用规则：上电以后会优先读取 flash 中的参数，如果没有找到对应参数会使用 SDK 中提供的默认值。sys_param.c 文件中 sys_parameter_init 完成了该操作。sys_param.c 中定义了 FlashParamReadMap 数组，通过 ID 从 flash 中读取数据，如果读取成功会从 flash 中复制 len 个字节到指定的地方。如果不成功，会从指定默认值的地方复制 len 个字节。

```
//flash中参数读取
//从flash中读取参数成功，将参数拷贝到dest_para
//读取不成功，拷贝默认参数default_para到dest_para
static const struct
{
    SYS_PARAMETER_ID    id;             //参数ID
    void *              dest_para;      //参数地址
    void *              default_para;   //默认参数地址
    uint8_t             len;            //参数长度
}FlashParamReadMap[] =
{
    {BT_PARA_BT_NAME_ID,              (void *)sys_parameter.bt_LocalDeviceName,       (void *)default_parameter.bt_LocalDeviceName,       BT_NAME_SIZE},
    {BT_PARA_BLE_NAME_ID,             (void *)sys_parameter.ble_LocalDeviceName,      (void *)default_parameter.ble_LocalDeviceName,      BLE_NAME_SIZE},
    {BT_PARA_RF_TX_LEVEL_ID,          (void *)&sys_parameter.bt_TxPowerLevel,         (void *)&default_parameter.bt_TxPowerLevel,         1},
    {BT_PARA_RF_PAGE_LEVEL_ID,        (void *)&sys_parameter.bt_PagePowerLevel,       (void *)&default_parameter.bt_PagePowerLevel,       1},
    {BT_PARA_TRIM_VAL_ID,             (void *)&sys_parameter.BtTrim,                  (void *)&default_parameter.BtTrim,                  1},
    {BT_PARA_CallinRingType_ID,       (void *)&sys_parameter.bt_CallinRingType,       (void *)&default_parameter.bt_CallinRingType,       1},
    {BT_PARA_BackgroundType_ID,       (void *)&sys_parameter.bt_BackgroundType,       (void *)&default_parameter.bt_BackgroundType,       1},
    {BT_PARA_SimplePairingEnable_ID,  (void *)&sys_parameter.bt_SimplePairingEnable,  (void *)&default_parameter.bt_SimplePairingEnable,  1},
    {BT_PARA_PinCode_ID,              (void *)sys_parameter.bt_PinCode,               (void *)default_parameter.bt_PinCode,               BT_PIN_CODE_LEN},
    {BT_PARA_ReconnectionEnable_ID,   (void *)&sys_parameter.bt_ReconnectionEnable,   (void *)&default_parameter.bt_ReconnectionEnable,   6},

    {TWS_PARA_TWS_VOL_SYNC_ID,        (void *)&sys_parameter.TwsVolSyncEnable,        (void *)&default_parameter.TwsVolSyncEnable,        1},
    {TWS_PARA_ReconnectionEnable_ID,  (void *)&sys_parameter.bt_TwsReconnectionEnable,(void *)&default_parameter.bt_TwsReconnectionEnable,6},
    {TWS_PARA_TwsPairingWhenPhoneConnectedSupport_ID,(void *)&sys_parameter.bt_TwsPairingWhenPhoneConnectedSupport,(void *)&default_parameter.bt_TwsPairingWhenI
    {TWS_PARA_TwsConnectedWhenActiveDisconSupport_ID,(void *)&sys_parameter.bt_TwsConnectedWhenActiveDisconSupport,(void *)&default_parameter.bt_TwsConnectedWhe
};
```

- flash_param.h 文件中 SYS_PARAMETER_ID 增加一个自定义参数的参数 ID，在 FLASH_PARAMETER 中增加参数的结构体类型（包含 ID+长度+参数值）。
- flash_param.c 文件中 SysDefaultParm，对这个参数赋默认值。
- sys_param.c 读取参数值，并且对参数做合法判断，非法的时候设置一个默认值。

7.1. 参量化数据修改举例 1：蓝牙名修改

1. 离线修改：PC 运行 OCPWorkBench_V1.0.4(2023.10.28)软件。
   (1) 导入 MVA 文件。
   (2) 修改蓝牙名称。
   (3) 保存为新的 MVA 文件。
   (4) 用新的 MVA 文件下载/烧录到目标板上既可以生效。

2. 在线修改：PC 用 USB 线连接目标设备，然后运行 OCPWorkBench_V1.0.4(2023.10.28)软件。
   (1) 点击 Receive_data，读取设备中的参量化数据。

(2) 修改蓝牙名称。
(3) 点击 Send_data，写入数据到设备中。
(4) 重新断电上电设备既可以生效。

7.2. 参量化数据修改举例 2：蓝牙重连设置

1. 参考 6，蓝牙名称的修改步骤。同样支持在线/离线修改 2 种方式。
2. 参数说明（参考 SYS_PARAMETER 定义的结构体顺序）：

```
[1, 5, 3, 1, 90, 5]
参数 1 BT 自动重连(开机或者切换模式)----- 1 开启
参数 2 自动重连尝试次数 ----- 5 次
参数 3 自动重连每两次间隔时间(in seconds) ----- 间隔 3S
参数 4 BB Lost 之后自动重连 1-> 打开/0->关闭 ----- 1 打开
参数 5 BB Lost 尝试重连次数 ----- 90 次
参数 6 BB Lost 重连每两次间隔时间(in seconds) ----- 间隔 5S
```

3. 点击参数修改的时候，在工具下方也会有对应的参数说明。

7.3. 参量化数据修改举例 3：蓝牙后台设置

1. 参考 6，蓝牙名称的修改步骤。同样支持在线/离线修改 2 种方式。
2. 参数说明：

BT 后台设置 0 -> BT 后台不能连接手机 1 -> BT 后台可以连接手机 2 -> 无后台

7.4. 参量化数据支持例表（持续更新）

ID	长度	默认参数	范围	说明
0x00	40	BP15_BT		字符串，蓝牙名称
0x01	40	BP15_BLE		字符串，BLE 蓝牙名称
0x02	2	23	0-23	蓝牙正常工作时发射功率
0x03	2	16	0-23	蓝牙回连发射功率
0x04	2	20	0-31	频偏 trim 值
0x05	2	2	0-3	bt 铃声设置 0 -> 不支持来电铃声 1 -> 来电报号和铃声 2 -> 使用手机铃声，若没有则播本地铃声 3 -> 强制使用本地铃声

0x06	2	0	0-2	BT 后台设置 0 -> BT 后台不能连接手机 1 -> BT 后台可以连接手机 2 -> 无后台
0x07	2	1	0-1	简易配对开启/关闭 0 -> 关闭 1 -> 开启
0x08	8	0000		字符串，配对 pin code
0x09	6	1,5,3,1,90,5		蓝牙回连设置 参数 1 BT 自动重连(开机或者切换模式) --- 1 开启 参数 2 自动重连尝试次数 --- 5 次 参数 3 自动重连每两次间隔时间(in seconds) --- 间隔 3S 参数 4 BB Lost 之后自动重连 1-> 打开/0->关闭 --- 1 打开 参数 5 BB Lost 尝试重连次数 --- 90 次 参数 6 BB Lost 重连每两次间隔时间(in seconds) --- 间隔 5S
0x200	40	Ver 1.0.0		参量化数据版本
0x1000				用户自定义数据

8. 蓝牙频偏值相关说明

1. 上电时，系统会从 flash 的指定区域读取频偏值，并使用；如果 flash 中没有保存频偏参数，则会使用默认频偏值 BT_DEFAULT_TRIM (bt_config.h)；
2. 可以通过串口日志信息观察当前板级的频偏值，蓝牙测试盒(v0.2.7 之后的固件版本)上也会显示校准后的参数；

串口日志表现如下：Freq trim:0x13

3. 在使用蓝牙测试盒的时候，可以根据实际情况选择操作：
   (1) 频偏校准：将测试盒档位打到校准频偏模式(000)，在测试盒连接成功被测设备后，会自动将频偏值校准，并保存到 flash 中；测试盒上显示校准后的频率偏差值；
   (2) 不需要频偏校准：将测试盒档位达到不校准频偏模式(001)，在测试盒连接成功被测设备后，不校准频偏，只是将当前被测设备的频率偏差值显示在屏幕上；
   (3) 在频偏校准完成后，蓝牙测试盒上会显示校准后的频偏值；如下图，0x13 是校准值；4Khz 为校准后的偏差频率；

a. 若生产环节不需要每台机器做频偏校准，可使用试产的产品多个使用测试盒进行频率的校准后，确认测试盒显示的校准值是否保持接近，然后将频偏值写到 SDK 中；
b. 如果蓝牙测试盒的频偏校正后面未显示参数，而是 “---”，则说明频偏值超过一定范围，测试盒无法校准；

解决方案：
a. 更换晶体；
b. 加大晶体谐振电流值；
例如：在 SystemClockInit() 函数内，系统默认是 Clock_HOSCCurrentSet(9); //晶体谐振电流配置为 9.

4. 目前蓝牙测试盒是支持 BT 和 BLE 进行频偏的校准；（固件版本：v0.2.11 之后）
(1) 基于经典蓝牙进行频偏校准，需要开启 A2DP 协议；同时兼顾经典蓝牙的相关功能测试。
(2) 基于 BLE 进行频偏校准，测试盒扫描特定的名称，需要按照测试盒的规范进行广播：（广播名称：MV_BLE）

9. 蓝牙名称说明

1) 在 bt_config.h 头文件中配置蓝牙名称 BT_NAME
2) 中文名称，可以直接在 bt_name.h 中，定义中文名称，但是需要在 bt_config.h 中，屏蔽 BT_NAME 定义，并打开 bt_name.h 的调用；

```
35
36 /*****************************************************************************
37  *
38  * 蓝牙名称注意事项:
39  * 1.蓝牙名称支持中文,需要使用URL编码
40  * 2.BLE的名称修改在ble广播数据中体现(ble_app_func.c)
41  *****************************************************************************/
42 #include "bt_name.h"
43 //#define BT_NAME                       "BP15_BT"
44 #define BLE_NAME                      "BP15_BLE"
```

注：bt_name.h 中中文名称的修改，需要保证修改的文件的格式为 UTF-8，避免由于文本格式错误，导致修改异常；

3) 如蓝牙名称为中文，或者其他语言，请将内容进行 URL 编码，再写入 bt_LocalDeviceName；

10.经典蓝牙单次连接超时时间

在 bt_config.h 中，BT_LSTO_DFT 可以配置蓝牙连接成功后，通讯异常，超时断开的时间；只有在 BP10 和手机连接时，角色为 Master 的情况下才有用；
注意：配置参数和实际参数的转换关系；

11.连接手机超时时间

在 bt_config.h 内，BT_PAGE_TIMEOUT 可以设置每次 page 的超时时间；默认时间：5s

12.BLE 地址、广播内容修改

1) BLE 的地址获取方式：先从 flash 的指定位置读取；若无保存的地址信息，则通过 BT 的地址来生成，并保存到 flash；
2) BLE 广播内容：
遵循 BLE 广播规范 LTV 数据格式：length + type + data；

具体修改如下：
1) SDK 默认名称修改可在 bt_config.h 文件夹如下：

```
#define BLE_NAME                    "BP15_BLE"
#define BLE_DFLT_DEVICE_NAME        (sys_parameter.ble_LocalDeviceName) //BLE 名称
```

2) 可直接修改 0x09 后的内容(注意修改后需要同时更改长度)

```
uint8_t ble_app_adv_data[30] = {
      //length + type + data
    // Flags general discoverable, BR/EDR not supported
    2, 0x01, 0x06,
    // Name
    9, 0x09, 'B','P','1','5','-','B','L','E',
};
```

3) 默认初始化配置接口代码如下：

```
le_user_config.adv_data.adv_data = (uint8_t *)ble_app_adv_data;
le_user_config.adv_data.adv_len = adv_len;

// 广播回复数据需要时填写,不需要时填NULL
le_user_config.rsp_data.adv_rsp_data = (uint8_t *)ble_app_rsp_adv_data;
le_user_config.rsp_data.adv_rsp_len = rsp_adv_len;
```

其中两个 BUFF 加起来总共可扩充 31+31 =62 个广播字节长度；
4) 动态广播数据更新 API，进行动态更新广播内容。

```
void app_set_adv_data(uint8_t *data, uint16_t len); //填充普通广播数据
void app_set_scan_rsp_data(uint8_t *data, uint16_t len); //塞入广播回复内容
void app_start_advertising(void); //使能广播
```

13.BLE GATT 服务修改

在 ble 文件夹内有 gatt_tools.exe 工具，可编辑 input.gatt 文件后，双击 gatt_tools.exe 生成 ble_app_gatt.h

1. 示例：

```
PRIMARY_SERVICE, GAP_SERVICE
CHARACTERISTIC, GAP_DEVICE_NAME, READ | WRITE | DYNAMIC,
CHARACTERISTIC, GAP_APPEARANCE, READ | WRITE | DYNAMIC,
PRIMARY_SERVICE, 49535343-FE7D-4AE5-8FA9-9FAFD205E455
CHARACTERISTIC, 49535343-aca3-481c-91ec-d85e28a60318, NOTIFY | WRITE | DYNAMIC,
CHARACTERISTIC, 49535343-6daa-4d02-abf6-19569aca69fe, NOTIFY | DYNAMIC,
CHARACTERISTIC, 49535343-8841-43F4-A8D4-ECBE34729BB3, WRITE | NOTIFY | DYNAMIC,
CHARACTERISTIC, 49535343-1E4D-4BD9-BA61-23C647249616, NOTIFY |
WRITE_WITHOUT_RESPONSE | DYNAMIC,
PRIMARY_SERVICE, AB00
CHARACTERISTIC, AB01, READ | WRITE | DYNAMIC,
CHARACTERISTIC, AB02, NOTIFY | DYNAMIC,
CHARACTERISTIC, AB03, NOTIFY | DYNAMIC,
PRIMARY_SERVICE, CB00
CHARACTERISTIC, CB01, READ | WRITE | WRITE_WITHOUT_RESPONSE | DYNAMIC,
CHARACTERISTIC, CB02, NOTIFY | DYNAMIC,
CHARACTERISTIC, CB03, NOTIFY | DYNAMIC,
```

添加 PRIMARY_SERVICE 首要服务， CHARACTERISTIC 特性，UUID，属性，按照如上格式添加，

2. 特殊字符：

特殊 UUID 字符替代解释：
GAP_SERVICE：1800
GATT_SERVICE：1801
GAP_DEVICE_NAME：2A00
GAP_APPEARANCE: 2A01
GAP_PERIPHERAL_PRIVACY_FLAG：2A02
GAP_RECONNECTION_ADDRESS: 2A03
GAP_PERIPHERAL_PREFERRED_CONNECTION_PARAMETERS: 2A04
GATT_SERVICE_CHANGED：0x2A05
以上 UUID 可以用上述字符代替，也可以直接输入 UUID

3. 可输入的属性

NOTIFY，WRITE_WITHOUT_RESPONSE，READ，WRITE，INDICATE。

DYNAMIC 为结束符！！！

注意事项：以英文输入法输入！！

input.gatt 和 gatt_tools.exe 需在同级目录

4. 输出文件：ble_app_gatt.h

编好 input.gatt 文件后执行 gatt_tools.exe 会输出 ble_app_gatt.h 文件，

该文件即为 BLE 堆栈所需的 profile 数据格式！！！

14.BLE MTU 配置

在 ble_app_func.c 中，BLE 初始化函数：LeInitConfigParams ()内，
le_user_config.att_default_mtu = DEFAULT_MTU_SIZE;（默认大小：250 字节）

15.BLE 读写发送

15.1. 接收数据(write)

当有数据发送至设备时（WRITE COMMAND）会产生 LE_RCV_DATA_EVENT 事件,如下:

```
case LE_RCV_DATA_EVENT:
 {
    uint8_t i;
    printf("\n*****LE_RCV_DATA_EVENT*****\n");
    BLE_INFO("connect handle: 0x%04x,att_handle: 0x%04x\n", param-
>rcv_data.conhdl, param->rcv_data.handle);
    BLE_INFO("RCV DATA: ");
    for (i = 0; i < param->rcv_data.len; i++) {
        printf("0x%02x,", param->rcv_data.data[i]); }
    printf("\n");
    //发送示例
    ble_send_data(param->rcv_data.conhdl,0,0,0x0e,test_buff,sizeof(test_buff),0);
     break;
 }
```

其中 param->rcv_data.conhdl 指示是哪一个设备发送，做多连接时会被变化， 一对一连接均为 0，标准 SDK 均为 1 对 1 连接， param->rcv_data.handle 则为属性句柄，其值根据 GATT PROFILE 配置决定的， 一 般则是根据起始值累加上来，可用手机调试助手发送数据观察其打印值也可看出；

15.2. 发送数据(Notify Or Indicate)

```
/**
```

```
 * @brief          发送数据接口 *
 * @param conidx    0x00
 * @param user_lid  0x00
 * @param metainfo  0x00
 * @param hdl       CCC--CFG handle -1
 * @param p         数据
 * @param len       长度
 * @param nf_or_nd  0x00 GATT_NOTIFY, 0x01 GATT_INDICATE 方式
 * @return uint32_t 0x00:发送成功,0x4B:资源不足
 */
   uint32_t ble_send_data(uint8_t conidx, uint8_t user_lid, uint16_t metainfo,
uint16_t hdl, uint8_t *p, uint32_t len, uint8_t nf_or_nd);
```

5. 参数 uint8_t conidx 决定发往哪个设备，多连接会变化，可根据连接完成回调上来的值设定;
6. user_lid API 一致性预留接口， 一对一连接均为 0;
7. metainfo API 一致性预留接口， 一对一连接均为 0;
8. uint16_t hdl 为 value_handle,其值也是根据 GATT PROFILE 配置决定的， 一般可以用手机调试助手 测试打开需要发送的通知功能，会打印 handle 值，此时为特性下 GATT_DESC_CLIENT_CHAR_CFG 的 value_handle 值，**在此基础上减一则为将代入 hdl 的值;**
9. nf_or_nd 为发送的类型 0x00 为 GATT_NOTIFY 发送方式，此方式无回复， 0x01 为 GATT_INDICATE 方 式带回复发送;
10. 发送的类型 0x00 为 GATT_NOTIFY 发送方式，此方式无回复， 0x01 为 GATT_INDICATE 方 式带回复发送;
11. **值得注意的是发送频率是当前连接参数息息相关的，连接间隔越小，则发送的频率可以提高，其发 送速率就越快，可通过如下 API 进行连接参数协商更新;**

```
/**
*@brief 更新连接参数 *
*/
void app_update_param(struct le_connection_param *p_conn_param);
```

一般安卓能接受的最小连接间隔为 7.5mS,苹果的为 15mS,连接参数更新成功会产生 LE_CONNECT_PARAMS_UPDATE 事件，打印出协商成功后的当前连接参数值;

15.3. 读数据(Read)

当有数据想要读取设备相关特性内容数据时（READ REQUEST），会产生 LE_APP_READ_DATA_EVENT 事件回调如下：

```
case LE_APP_READ_DATA_EVENT: {
    printf("\n*****LE_APP_READ_DATA_EVENT*****\n"); switch
    (param->read_data.handle)
    {
    case 0:
        break;
```

```
        default:
            BLE_INFO("\n*****test_buff*****\n");
            BLE_INFO("param->read_data.connect_handle: 0x%04x\n", param-
    >read_data.connect_handle);
            BLE_INFO("param->read_data.handle: 0x%04x\n", param-
    >read_data.handle);
            BLE_INFO("param->read_data.offset: 0x%04x\n", param-
    >read_data.offset);
            param->read_data.len = sizeof(test_buff);
            param->read_data.data = test_buff;//有生命周期，需为全局 BUFF break;
        }

        break; }
    default:
break; }
```

1. param->read_data.handle 为想要读取哪一特性的内容；
2. param->read_data.connect_handle 为想要读取哪个设备的， 一对一连接时均为 0；
3. param->read_data.offset 为读取 BUFF 内容的偏移大小；
4. param->read_data.len 则为读物 BUFF 大小的长度；
5. param->read_data.data 则为需要填入 BUFF 的内容，注意此为回调取内容 BUFF 需为全局 BUFF

15.4. BLE 数据收发注意事项

1. 在 ble_app_callback，收到 BLE 数据后，对数据进行缓存处理；
   (1) 不能在此回调中，直接进行数据的发送；
   (2) 不能在此回调中，做任务阻塞处理的任何操作；
2. 在 bt_stack_service 任务的 runloop 中，进行数据发送流程的处理；
3. 在处理 BLE OTA 流程时，对于接收到的数据，进行 flash 擦写，建议是在非蓝牙任务的其他任务下处理；
   (1) 其他任务的优先级不能高于 bt_stack_service 任务的优先级；
   (2) flash 擦写的函数调用接口，需要使用如下接口：

```
void SpiFlashErase(ERASE_TYPE_ENUM erase_type, uint32_t index, bool IsSuspend);
SPI_FLASH_ERR_CODE SpiFlashWrite( uint32_tAddr,  uint8_t *Buffer,  uint32_t  Length,
 uint32_t IsSuspend);
```

16.蓝牙设备类型 COD 介绍

1. 蓝牙 COD（Class Of Device）是一个蓝牙设备分类类型值，由 24bit 组成，分别表示主要服务类别，主要设备类别和次要设备类别。

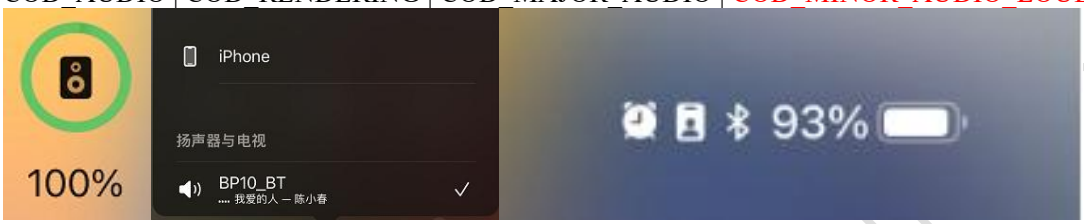
23 ... 13	12 ... 8	7 ... 2	1	0
Major Service Classes	Major Device Class	Minor Device Class	0b00	

2. 设置方式：在函数 ConfigBtStackParams 函数内，使用 pCod 进行设备类型的配置；

16.1. COD 示例

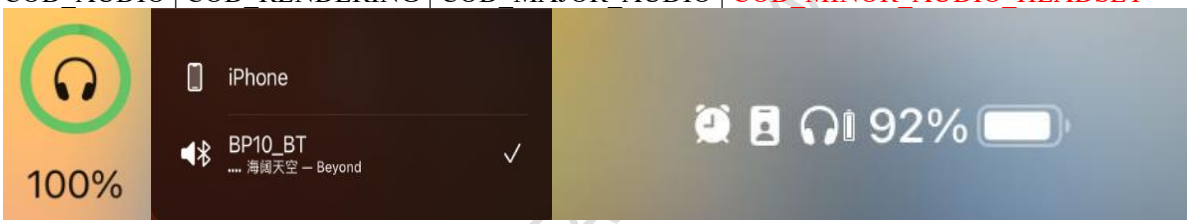
1. 音响设备：

COD_AUDIO | COD_RENDERING | COD_MAJOR_AUDIO | COD_MINOR_AUDIO_LOUDSPEAKER



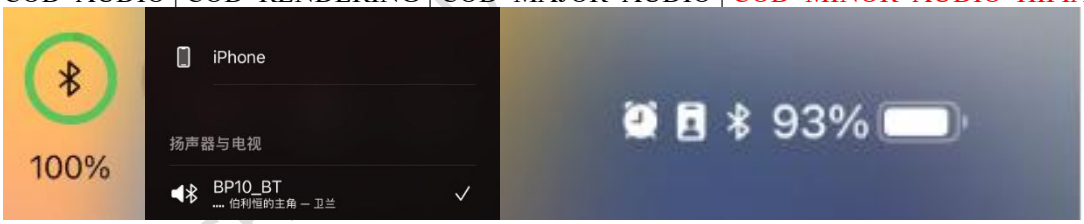
2. 头戴式耳机：

COD_AUDIO | COD_RENDERING | COD_MAJOR_AUDIO | COD_MINOR_AUDIO_HEADSET



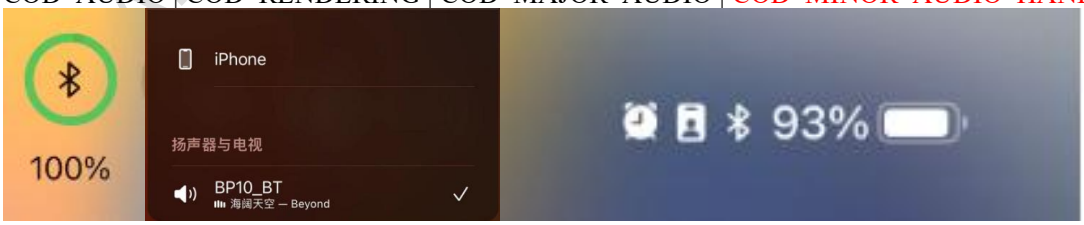
3. 高保真音频设备：

COD_AUDIO | COD_RENDERING | COD_MAJOR_AUDIO | COD_MINOR_AUDIO_HIFIAUDIO



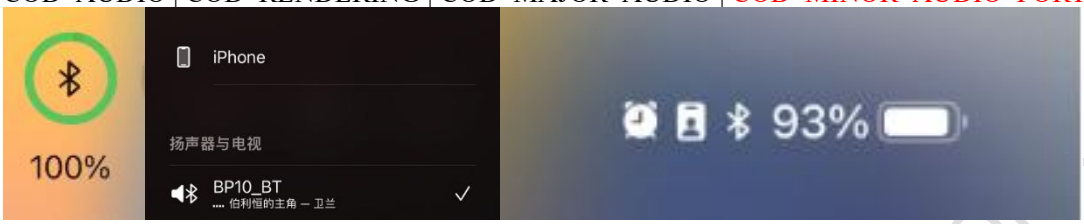
4. 免提设备：

COD_AUDIO | COD_RENDERING | COD_MAJOR_AUDIO | COD_MINOR_AUDIO_HANDSFREE



5．便携式音频：

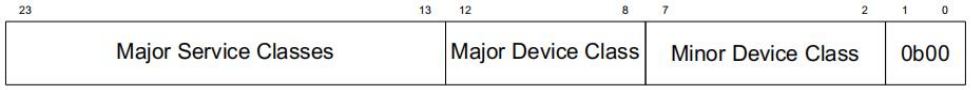
COD_AUDIO | COD_RENDERING | COD_MAJOR_AUDIO | COD_MINOR_AUDIO_PORTABLEAUDIO



6．键盘外设：

COD_MAJOR_PERIPHERAL | COD_MINOR_PERIPH_KEYBOARD

16.2. COD 定义



Bit	Class of Device Major Service Class
13	Limited Discoverable Mode
14	LE audio
15	Reserved for future use
16	Positioning (Location identification)
17	Networking (LAN, Ad hoc, ...)
18	Rendering (Printing, Speakers, ...)
19	Capturing (Scanner, Microphone, ...)
20	Object Transfer (v-Inbox, v-Folder, ...)
21	Audio (Speaker, Microphone, Headset service, ...)
22	Telephony (Cordless telephony, Modem, Headset service, ...)
23	Information (WEB-server, WAP-server, ...)

12	11	10	9	8	Major Device Class
0	0	0	0	0	Miscellaneous
0	0	0	0	1	Computer (desktop, notebook, PDA, organizer, ...)
0	0	0	1	0	Phone (cellular, cordless, pay phone, modem, ...)
0	0	0	1	1	LAN/Network Access point
0	0	1	0	0	Audio/Video (headset, speaker, stereo, video display, VCR, ...)
0	0	1	0	1	Peripheral (mouse, joystick, keyboard, ...)
0	0	1	1	0	Imaging (printer, scanner, camera, display, ...)
0	0	1	1	1	Wearable
0	1	0	0	0	Toy
0	1	0	0	1	Health
1	1	1	1	1	Uncategorized: device code not specified

2.8.2.4 Minor Device Class field – Audio/Video Major Class

Last Modified: 2022-05-25

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	0	Uncategorized, code not assigned
0	0	0	0	0	1	Wearable Headset Device
0	0	0	0	1	0	Hands-free Device
0	0	0	0	1	1	(Reserved)
0	0	0	1	0	0	Microphone
0	0	0	1	0	1	Loudspeaker
0	0	0	1	1	0	Headphones
0	0	0	1	1	1	Portable Audio
0	0	1	0	0	0	Car Audio
0	0	1	0	0	1	Set-top box
0	0	1	0	1	0	HiFi Audio Device
0	0	1	0	1	1	VCR
0	0	1	1	0	0	Video Camera
0	0	1	1	0	1	Camcorder
0	0	1	1	1	0	Video Monitor
0	0	1	1	1	1	Video Display and Loudspeaker
0	1	0	0	0	0	Video Conferencing
0	1	0	0	0	1	(Reserved)
0	1	0	0	1	0	Gaming/Toy

2.8.2.1 Minor Device Class field – Computer Major Class

Last Modified: 2022-05-25

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	0	Uncategorized, code for device not assigned
0	0	0	0	0	1	Desktop workstation
0	0	0	0	1	0	Server-class computer
0	0	0	0	1	1	Laptop
0	0	0	1	0	0	Handheld PC/PDA (clamshell)
0	0	0	1	0	1	Palm-size PC/PDA
0	0	0	1	1	0	Wearable computer (watch size)
0	0	0	1	1	1	Tablet

2.8.2.2 Minor Device Class field – Phone Major Class

Last Modified: 2022-05-25

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	0	Uncategorized, code for device not assigned
0	0	0	0	0	1	Cellular
0	0	0	0	1	0	Cordless
0	0	0	0	1	1	Smartphone
0	0	0	1	0	0	Wired modem or voice gateway
0	0	0	1	0	1	Common ISDN access

2.8.2.3 Minor Device Class field – LAN/Network Access point Major Class

Last Modified: 2022-05-25

7	6	5	Minor Device Class
0	0	0	Fully available
0	0	1	1% to 17% utilized
0	1	0	17% to 33% utilized
0	1	1	33% to 50% utilized
1	0	0	50% to 67% utilized
1	0	1	67% to 83% utilized
1	1	0	83% to 99% utilized
1	1	1	No service available

Last Modified: 2022-05-25

4	3	2	Minor Device Class
0	0	0	Uncategorized (use this value if no others apply)

2.8.2.5 Minor Device Class field – Peripheral Major Class

Last Modified: 2022-05-25

7	6	Minor Device Class
0	0	Uncategorized, code not assigned
0	1	Keyboard
1	0	Pointing device
1	1	Combo Keyboard/Pointing device

Last Modified: 2022-05-25

5	4	3	2	Minor Device Class
0	0	0	0	Uncategorized, code not assigned
0	0	0	1	Joystick
0	0	1	0	Gamepad
0	0	1	1	Remote control
0	1	0	0	Sensing device
0	1	0	1	Digitizer tablet
0	1	1	0	Card Reader (e.g. SIM Card Reader)
0	1	1	1	Digital Pen
1	0	0	0	Handheld scanner for barcodes, RFID, etc.
1	0	0	1	Handheld gestural input device (e.g., “wand” form factor)

2.8.2.6 Minor Device Class field – Imaging Major Class

Last Modified: 2022-05-25

7	6	5	4	Minor Device Class
X	X	X	1	Display
X	X	1	X	Camera
X	1	X	X	Scanner
1	X	X	X	Printer

Last Modified: 2022-05-25

3	2	Minor Device Class
0	0	Uncategorized, default

2.8.2.7 Minor Device Class field – Wearable Major Class

Last Modified: 2022-05-25

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	1	Wristwatch
0	0	0	0	1	0	Pager
0	0	0	0	1	1	Jacket
0	0	0	1	0	0	Helmet
0	0	0	1	0	1	Glasses

2.8.2.8 Minor Device Class field – Toy Major Class

Last Modified: 2022-05-25

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	1	Robot
0	0	0	0	1	0	Vehicle
0	0	0	0	1	1	Doll/Action figure
0	0	0	1	0	0	Controller
0	0	0	1	0	1	Game

2.8.2.9 Minor Device Class field – Health Major Class

Last Modified: 2022-05-25

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	0	Undefined
0	0	0	0	0	1	Blood Pressure Monitor
0	0	0	0	1	0	Thermometer
0	0	0	0	1	1	Weighing Scale
0	0	0	1	0	0	Glucose Meter
0	0	0	1	0	1	Pulse Oximeter
0	0	0	1	1	0	Heart/Pulse Rate Monitor
0	0	0	1	1	1	Health Data Display
0	0	1	0	0	0	Step Counter
0	0	1	0	0	1	Body Composition Analyzer
0	0	1	0	1	0	Peak Flow Monitor
0	0	1	0	1	1	Medication Monitor
0	0	1	1	0	0	Knee Prosthesis
0	0	1	1	0	1	Ankle Prosthesis
0	0	1	1	1	0	Generic Health Manager
0	0	1	1	1	1	Personal Mobility Device

17.蓝牙 Profile 定义说明

1. BT_A2DP_SUPPORT 宏定义开关，包含了 A2DP 和 AVRCP 2 个协议的开关，由于考虑到 SDK 的使用情况，默认都是支持蓝牙音乐播放，所以必须要开启 A2DP；
2. BLE、HFP、SPP、OBEX、PBAP 根据实际的需要进行开启和关闭；
3. 支持不同的 profile，使用的 ram 资源是有区别的，请查看 bt_stack_memory.h 文件中 BT_STACK_MEM_SIZE 的相关说明；
   (1) 当出现 `BT_STACK_EVENT_COMMON_STACK_FREE_MEM_SIZE` 时，说明 BT_STACK_MEM_SIZE 大小设置小了，需要将 BT_STACK_MEM_SIZE 加大；
   (2) 可以在事件 `BT_STACK_EVENT_COMMON_STACK_FREE_MEM_SIZE` 打开日志信息，对需要的内存进行监控；

```
case BT_STACK_EVENT_COMMON_STACK_FREE_MEM_SIZE:
{
    //APP_DBG("stack mem:%d, %d\n", param->params.stackMemParams.stackMallocMemSize, param->params.stackMemParams.stackFreeMemSize);
}
break;
```

18.高级 AVRCP 功能说明

默认开启高级 AVRCP 功能：BT_AVRCP_ADVANCED

1. 通过高级 AVRCP 功能能及时更新当前的播放状态；
2. 开启 BT_AVRCP_VOLUME_SYNC，支持蓝牙音量同步功能，主要适用于苹果手机的播放音乐音量同步；
3. 开启 BT_AVRCP_SONG_PLAY_STATE，实时的刷新歌曲的播放时间；
4. 开启 BT_AVRCP_SONG_TRACK_INFOR，在歌曲开始播放、歌曲切换，获取歌曲的 ID3 信息；

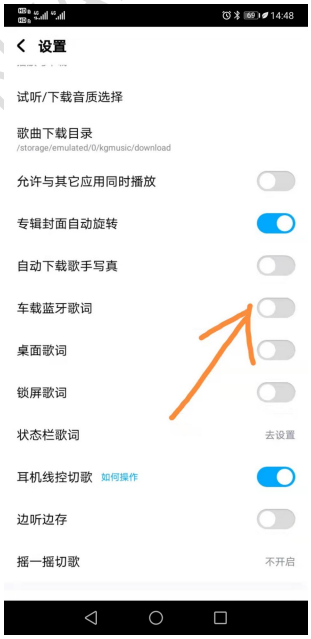
19.获取歌曲歌词功能说明

手机部分音乐播放 APP 支持歌曲歌词的获取，代码层面需要修改如下：

1. 在 bt_config.h 文件中，需要开启如下宏定义：

BT_AVRCP_SONG_PLAY_STATE ENABLE （播放状态和播放时间）
BT_AVRCP_SONG_TRACK_INFOR ENABLE （歌曲和歌词信息）

2. 通过回调事件 BT_STACK_EVENT_AVRCP_ADV_MEDIA_INFO 反馈歌曲的信息；
3. 歌词信息会在 Title of the media 中显示；
4. 歌曲信息根据手机 APP 上，有歌词的变化后，会自动进行推送；(不再需要定时获取)
5. 每次只能获取到 1 行歌词，此功能是由手机 APP 来决定；
6. 手机端 APP 需要开启“车载蓝牙歌词”功能，可以参考下图：



7. 在使用时，需要将歌词信息保存下来，不要直接打印，转到其他任务中再次处理，否则可能会导致蓝牙任务阻塞而出现异常；

20.通话相关参数配置

1. AEC 音效的参数配置，都集成在了音效框图内
2. SDK 默认关闭手机的 AEC 功能，即开启宏定义：BT_REMOTE_AEC_DISABLE；开启后会在连上手机的时候，发送 CMD 关闭手机端的 AEC；

21.HFP 电池电量同步功能

在 bt_config.h 文件内，BT_HFP_BATTERY_SYNC 宏定义开关用于开启电池电量同步功能。

1. 在 HFP 协议连接成功后，自动同步当前设备的电量到手机端显示；
2. 需要配合 CFG_FUNC_POWER_MONITOR_EN 功能一起使用；

22.蓝牙连接成功、断开连接，提示音导入

蓝牙协议栈会在蓝牙连接成功和断开连接的时候，将 MSG_BT_STATE_CONNECTED 和 MSG_BT_STATE_DISCONNECT 2 个消息发送到 main_task.c 中，可在消息处理中加入相关的提示音；

23.SPP 协议

1. SPP 数据接收的回调事件：BT_STACK_EVENT_SPP_DATA_RECEIVED
   (1) 在此回调事件中，建议只进行数据的接收和缓存；
   (2) 不要在此回调事件处理中，增加 SPP 数据发送的流程；
   (3) 不要在此回调事件处理中，有任何阻塞行为；可能会导致蓝牙异常；
2. 可以在 BtSppHookFunc() 函数内，统一进行 SPP 数据的处理和发送；
3. 使用 SPP 协议进行 OTA 升级，升级流程可以参考 Obex 升级流程：MVA_BT_OBEX_UPDATE_FUNC_SUPPORT 宏定义关联的函数和流程；

24.OBEX 协议

1. 通常开启 OBEX 协议，会用于文件推送，进行 OTA 双 BANK 升级；
2. 需要开启 MVA_BT_OBEX_UPDATE_FUNC_SUPPORT 才能进行升级；
3. OTA 升级会临时创建一个任务，专门用于升级流程的处理，包含数据的接收，保存到 flash，最后进行双 BANK 升级；
4. 注意：是否有足够的 RAM，用于 OTA 升级任务的创建；

25.经典蓝牙自定义 UUID 服务

可以根据客户项目需求，增加自定义的 UUID 服务，这部分需要定制蓝牙库；联系 FAE 进行需求申请；

26.蓝牙异常现象说明

26.1. 上电后蓝牙搜索不到

此问题需要通过串口信息来定位异常情况。

1. 出现错误：**!!!ERR: BT_STACK_EVENT_COMMON_STACK_FREE_MEM_SIZE**

1） 此问题是由于蓝牙协议栈内存分配不足，导致协议栈初始化不成功

2） bt_stack_memory.h 中，在结构体 BT_STACK_MEM_ALLOC 中，增加 BtBaseMem[] 的大小，如以 1KB 为单位。

3） 同时把出现异常情况下的蓝牙配置信息，反馈给山景 FAE。

2. 排查可见性状态是否存在异常

在上电后，查看可见性的状态：

1） $$$ access mode 0->3：此状态符合预期，说明蓝牙可见性从 不可被搜索不可被连接 变成了 可被搜索可被连接。

2） $$$ access mode 3->0：说明蓝牙可见性从 可被搜索可被连接 变成了 不可被搜索不可被连接。此时则需要在 BtAccessModeSetting()函数内，根据实际的应用场景，配置可见性参数；

3） 可见性参数如下：

0=不可被搜索不可被连接

1=可被搜索不可被连接

2=不可被搜索可被连接

3=可被搜索可被连接

```
/**
 * BtAccessibleMode type
 */
typedef U8      BtAccessMode;

#define BtAccessModeNotAccessible       0x00 /*!< Non-discoverable or connectable      */
#define BtAccessModeDiscoverbleOnly     0x01 /*!< Discoverable but not connectable     */
#define BtAccessModeConnectableOnly     0x02 /*!< Connectable but not discoverable     */
#define BtAccessModeGeneralAccessible   0x03 /*!< General discoverable and connectable */
```

在使用过程中，如有疑问请联系 FAE。