RockChip DeviceIo Bluetooth Interface Documentation

发布版本：1.2

作者：francis.fan

日期：2019.4.29

文件密级：公开资料

概述

该文档旨在介绍RockChip DeviceIo库的蓝牙接口。

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	文档版本	对应库版本	作者	修改说明
2019-3-27	V1.0	V1.0.x / V1.1.x	francis.fan	初始版本（BLUEZ only）
2019-4-16	V1.1	V1.2.0	francis.fan	新增BLE配网Demo 修复BtSource接口 新增BSA库的支持 修复文档排版
2019-4-29	V1.2	V1.2.1	francis.fan	修复BSA分支deviceio_test测试失败 修复BLUEZ初始化失败程序卡住的BUG 修改A2DP SOURCE 获取playrole方法

1、蓝牙基础接口（RkBtBase.h）

- `RkBtContent`结构

```
typedef struct {
    Ble_Uuid_Type_t server_uuid; //BLE server uuid
    Ble_Uuid_Type_t chr_uuid[12]; //BLE CHR uuid，最多12个
    uint8_t chr_cnt; //CHR 个数
    const char *ble_name; //BLE名称，该名称可与bt_name不一致。
    uint8_t advData[256]; //广播数据
    uint8_t advDataLen; //广播数据长度
    uint8_t respData[256]; //广播回应数据
    uint8_t respDataLen; //广播回应数据长度
    /* 生成广播数据的方式，取值：BLE_ADVDATA_TYPE_USER/BLE_ADVDATA_TYPE_SYSTEM
     * BLE_ADVDATA_TYPE_USER：使用advData和respData中的数据作为BLE广播
     * BLE_ADVDATA_TYPE_SYSTEM：系统默认广播数据。
     *     广播数据包：flag(0x1a)，128bit Server UUID;
     *     广播回应包：蓝牙名称
     */
    uint8_t advDataType;
    //AdvDataKgContent adv_kg;
    char le_random_addr[6]; //随机地址，系统默认生成，用户无需填写
    /* BLE数据接收回调函数，uuid表示当前CHR UUID，data：数据指针，len：数据长度 */
    void (*cb_ble_recv_fun)(const char *uuid, unsigned char *data, int len);
    /* BLE数据请求回调函数。该函数用于对方读操作时，会触发该函数进行数据填充 */
    void (*cb_ble_request_data)(const char *uuid);
} RkBleContent;
```

- `RkBtContent`结构

```
typedef struct {
    RkBleContent ble_content; //BLE 参数配置
    const char *bt_name; //蓝牙名称
} RkBtContent;
```

- `int rk_bt_init(RkBtContent *p_bt_content)`

  蓝牙服务初始化。调用其他蓝牙接口前，需先调用该接口进行蓝牙基础服务初始化。

- `int rk_bt_deinit(void)`

  蓝牙服务反初始化。

- `int rk_bt_is_connected(void)`

  获取当前蓝牙是否有某个服务处于连接状态。SPP/BLE/SINK/SOURCE任意一个服务处于连接状态，该函数都返回1；否则返回0。

2、BLE接口介绍（RkBle.h）

- `RK_BLE_STATE`说明

```
typedef enum {
    RK_BLE_STATE_IDLE = 0, //空闲状态
    RK_BLE_STATE_CONNECT, //连接成功
    RK_BLE_STATE_DISCONNECT //断开连接
} RK_BLE_STATE;
```

- `typedef void (*RK_BLE_STATE_CALLBACK)(RK_BLE_STATE state)`

  BLE状态回调函数。

- `typedef void (*RK_BLE_RECV_CALLBACK)(const char *uuid, char *data, int len)`

  BLE接收回调函数。uuid：CHR UUID，data：数据指针，len：数据长度。

- `int rk_ble_register_status_callback(RK_BLE_STATE_CALLBACK cb)`

  该接口用于注册获取BLE连接状态的回调函数。

- `int rk_ble_register_recv_callback(RK_BLE_RECV_CALLBACK cb)`

  该接口用于注册接收BLE数据的回调函数。存在两种注册接收回调函数的方法：一种是通过rk_bt_init()接口的RkBtContent 参数进行指定；另一种是调用该接口进行注册。BLUEZ两种方式均可用，而对BSA来说只能使用该接口注册接收回调函数。

- `int rk_ble_start(RkBleContent *ble_content)`

  开启BLE广播。ble_content：需与rk_bt_init(RkBtContent *p_bt_content)中p_bt_content->ble_content保持一致

- `int rk_ble_stop(void)`

  停止BLE广播。该函数执行后，BLE变为不可见并且不可连接。

- `int rk_ble_get_state(RK_BLE_STATE *p_state)`

  主动获取BLE当前的连接状态。

- `rk_ble_write(const char *uuid, char *data, int len)`

  往对端发送数据。

  uuid：写入数据的CHR对象

  data：写入数据的指针

  len：写入数据的长度。特别说明：该长度受到BLE连接的MTU限制，超过MTU将被截断。

  为保持良好的兼容性，当前MTU值默认为：134 Bytes

2、SPP接口介绍（RkBtSpp.h）

- `RK_BT_SPP_STATE`介绍

```
typedef enum {
    RK_BT_SPP_STATE_IDLE = 0, //空闲状态
    RK_BT_SPP_STATE_CONNECT, //连接成功状态
    RK_BT_SPP_STATE_DISCONNECT //断开连接
} RK_BT_SPP_STATE;
```

- `typedef void (*RK_BT_SPP_STATUS_CALLBACK)(RK_BT_SPP_STATE status)`

  状态回调函数。

- `typedef void (*RK_BT_SPP_RECV_CALLBACK)(char *data, int len)`

  接收回调函数。data：数据指针，len：数据长度。

- `int rk_bt_spp_register_status_cb(RK_BT_SPP_STATUS_CALLBACK cb)`

  注册状态回调函数。

- `int rk_bt_spp_register_recv_cb(RK_BT_SPP_RECV_CALLBACK cb)`

  注册接收回调函数。

- `int rk_bt_spp_open(void)`

  打开SPP，设备处于可连接状态。由于连接管理对A2DP Sink有依赖，因此该接口内部会检测A2DP Sink是否开启，若没开启则会先打开A2DP Sink。

- `int rk_bt_spp_close(void)`

  关闭SPP，打开时会触发A2DP Sink打开，但关闭仅关闭SPP的服务。

- `int rk_bt_spp_get_state(RK_BT_SPP_STATE *pState)`

  主动获取当前SPP连接状态。

- `int rk_bt_spp_write(char *data, int len)`

  发送数据。data：数据指针，len：数据长度。

3、A2DP SINK接口介绍（RkBtSink.h）

- `RK_BT_SINK_STATE`介绍

```
typedef enum {
    RK_BT_SINK_STATE_IDLE = 0, //空状态
    RK_BT_SINK_STATE_CONNECT,  //连接状态
    RK_BT_SINK_STATE_PLAY , //播放状态
    RK_BT_SINK_STATE_PAUSE, //暂停状态
    RK_BT_SINK_STATE_STOP, //停止状态
    RK_BT_SINK_STATE_DISCONNECT //断开连接
} RK_BT_SINK_STATE;
```

- `typedef int (*RK_BT_SINK_CALLBACK)(RK_BT_SINK_STATE state)`

  状态回调函数。

- `int rk_bt_sink_register_callback(RK_BT_SINK_CALLBACK cb)`

  注册状态回调函数。

- `int rk_bt_sink_open()`

打开A2DP Sink功能。

- `int rk_bt_sink_set_visibility(const int visiable, const int connectal)`

  设置A2DP Sink可见/可连接特性。visiable：0表示不可见，1表示可见。connectal：0表示不可连接，1表示可连接。

- `int rk_bt_sink_close(void)`

  关闭A2DP Sink功能。

- `int rk_bt_sink_get_state(RK_BT_SINK_STATE *p_state)`

  主动获取A2DP Sink连接状态。

- `int rk_bt_sink_play(void)`

  反向控制：播放。

- `int rk_bt_sink_pause(void)`

  反向控制：暂停。

- `int rk_bt_sink_prev(void)`

  反向控制：上一曲。

- `int rk_bt_sink_next(void)`

  反向控制：下一曲。

- `int rk_bt_sink_stop(void)`

  反向控制：停止播放。

- `int rk_bt_sink_volume_up(void)`

  反向控制：音量增大。

- `int rk_bt_sink_volume_down(void)`

  反向控制：银两减小。

- `int rk_bt_sink_set_auto_reconnect(int enable)`

  设置A2DP Sink自动连接属性。enable：1表示可自动连接，0表示不可自动连接。

- `int rk_bt_sink_disconnect()`

  断开A2DP Sink连接。

4、A2DP SOURCE接口介绍（RkBtSource.h）

- `BtDeviceInfo`介绍

```
typedef struct _bt_device_info {
    char name[128]; // bt name
    char address[17]; // bt address
    bool rssi_valid;
    int rssi;
    char playrole[12]; // Audio Sink? Audio Source? Unknown?
} BtDeviceInfo;
```

上述结构用于保存扫描到的设备信息。name：设备名称。address：设备地址。rssi_valid：表示rssi是否有效值。rssi：信号强度。playrole：设备角色，取值为“Audio Sink”、“Audio Source”、“Unknown”

- `BtScanParam`介绍

```
typedef struct _bt_scan_parameter {
    unsigned short mseconds;
    unsigned char item_cnt;
    BtDeviceInfo devices[BT_SOURCE_SCAN_DEVICES_CNT];
} BtScanParam;
```

该结构用于保存rk_bt_source_scan(BtScanParam *data)接口中扫描到的设备列表。mseconds：扫描时长。item_cnt：扫描到的设备个数。devices：设备信息。BT_SOURCE_SCAN_DEVICES_CNT值为30个，表示该接口扫描到的设备最多为30个。

- `RK_BT_SOURCE_EVENT`介绍

```
typedef enum {
    BT_SOURCE_EVENT_CONNECT_FAILED, //连接A2DP Sink设备失败
    BT_SOURCE_EVENT_CONNECTED, //连接A2DP Sink设备成功
    BT_SOURCE_EVENT_DISCONNECTED, //断开连接
} RK_BT_SOURCE_EVENT;
```

- `RK_BT_SOURCE_STATUS`介绍

```
typedef enum {
    BT_SOURCE_STATUS_CONNECTED, //连接状态
    BT_SOURCE_STATUS_DISCONNECTED, //断开状态
} RK_BT_SOURCE_STATUS;
```

- `typedef void (*RK_BT_SOURCE_CALLBACK)(void *userdata, const RK_BT_SOURCE_EVENT event)`

状态回调函数。userdata：用户指针，event：连接事件。

- `int rk_bt_source_auto_connect_start(void *userdata, RK_BT_SOURCE_CALLBACK cb)`

自动扫描周围Audio Sink类型设备，并主动连接rssi最强的设备。userdata：用户指针，cb：状态回调函数。

- `int rk_bt_source_auto_connect_stop(void)`

关闭自动扫描。

- `int rk_bt_source_open(void)`

打开A2DP Source功能。

- `int rk_bt_source_close(void)`

关闭A2DP Source功能。

- `int rk_bt_source_get_device_name(char *name, int len)`

获取本端设备名称。name：存放名称的buffer，len：name空间大小。

- `int rk_bt_source_get_device_addr(char *addr, int len)`

获取本端设备地址。addr：存放地址的buffer，len：addr空间大小。

- `int rk_bt_source_get_status(RK_BT_SOURCE_STATUS *pstatus, char *name, int name_len, char *addr, int addr_len)`

  获取A2DP Source连接状态。pstatus：保存当前状态值的指针。若当前处于连接状态，name保存对端设备（A2DP Sink）的名称，name_len为name长度，addr保存对端设备（A2DP Sink）的地址，addr_len为addr长度。参数name和addr均可置空。

- `int rk_bt_source_scan(BtScanParam *data)`

  扫描设备。扫描参数通过data指定，扫描到的结果也保存在data中。具体参见BtScanParam说明。

- `int rk_bt_source_connect(char *address)`

  主动连接address指定的设备。

- `int rk_bt_source_disconnect(char *address)`

  断开连接。

- `int rk_bt_source_remove(char *address)`

  删除已连接成功的设备。删除后无法自动连接。

- `int rk_bt_source_register_status_cb(void *userdata, RK_BT_SOURCE_CALLBACK cb)`

  注册状态回调函数。

5、示例程序说明

示例程序的路径为：external/deviceio/test。其中bluetooth相关的测试用例都实现在bt_test.cpp中，该测试用例涵盖了上述所有接口。函数调用在DeviceIOTest.cpp中。

5.1 编译说明###

1、在SDK根目录下执行`make deviceio-dirclean && make deviceio -j4`，编译成功会提示如下log（注：仅截取部分，rk-xxxx对应具体的工程根目录） -- Installing: /home/rk-xxxx/buildroot/output/target/usr/lib/librkmediaplayer.so -- Installing: /home/rk-xxxx/buildroot/output/target/usr/lib/libDeviceIo.so -- Installing: /home/rk-xxxx/buildroot/output/target/usr/include/DeviceIo/Rk_battery.h -- Installing: /home/rk-xxxx/buildroot/output/target/usr/include/DeviceIo/RK_timer.h -- Installing: /home/rk-xxxx/buildroot/output/target/usr/include/DeviceIo/Rk_wake_lock.h -- Installing: /home/rk-xxxx/buildroot/output/target/usr/bin/deviceio_test

2、执行./build.sh生成新固件，然后将新固件烧写到设备中。

5.2 基础接口演示程序

5.2.1 接口说明

- void bt_test_init_open(void *data)

  蓝牙测试初始化，执行蓝牙测试前，先调用该接口。BLE的接收和数据请求回调函数的注册。

  注：BLE 读数据是通过注册回调函数实现。当BLE连接收到数据主动调用接收回调函数。具体请参见 RkBtContent 结构说明和rk_ble_register_recv_callback函数说明。

- void bt_test_ble_start(void *data)

  启动BLE。设备被动连接后，收到“Hello RockChip”，回应“My name is rockchip”。

- void bt_test_ble_write(void *data)

  测试BLE写功能，发送134个‘0’-‘9’组成的字符串。
- void bt_test_ble_get_status(void *data)

  测试BLE状态接口。
- void bt_test_ble_stop(void *data)

  停止BLE。
- void bt_test_sink_open(void *data)

  打开 A2DP Sink 模式。
- void bt_test_sink_visibility00(void *data)

  设置 A2DP Sink 不可见、不可连接。
- void bt_test_sink_visibility01(void *data)

  设置 A2DP Sink 可见、不可连接。
- void bt_test_sink_visibility10(void *data)

  设置 A2DP Sink 不可见、可连接。
- void bt_test_sink_visibility11(void *data)

  设置 A2DP Sink 可见、可连接。
- void bt_test_sink_music_play(void *data)

  反向控制设备播放。
- void bt_test_sink_music_pause(void *data)

  反向控制设备暂停。
- void bt_test_sink_music_next(void *data)

  反向控制设备播放下一曲。
- void bt_test_sink_music_previous(void *data)

  反向控制设备播放上一曲。
- void bt_test_sink_music_stop(void *data)

  反向控制设备停止播放。
- void bt_test_sink_reconnect_enable(void *data)

  使能 A2DP Sink 自动连接功能。
- void bt_test_sink_reconnect_disenable(void *data)

  禁用 A2DP Sink 自动连接功能。
- void bt_test_sink_disconnect(void *data)

  A2DP Sink 断开链接。
- void bt_test_sink_close(void *data)

  关闭 A2DP Sink 服务。
- void bt_test_sink_status(void *data)

  查询 A2DP Sink 连接状态。

- void bt_test_source_auto_start(void *data)

  A2DP Source 自动扫描开始。
- void bt_test_source_auto_stop(void *data)

  A2DP Source 自动扫描接口停止。
- void bt_test_source_connect_status(void *data) 获取 A2DP Source 连接状态。
- void bt_test_spp_open(void *data)

  打开SPP。
- void bt_test_spp_write(void *data)

  测试SPP写功能。向对端发送“This is a message from rockchip board！”字串。
- void bt_test_spp_close(void *data)

  关闭SPP。
- void bt_test_spp_status(void *data)

  查询SPP连接状态。

5.2.2 测试步骤

1、执行测试程序命令：`DeviceIOTest bluetooth`显示如下界面：

```
# deviceio_test bluethood
version:V1.2.1
#### Please Input Your Test Command Index ####
00.
01.  bt_server_open
02.  bt_test_source_auto_start
03.  bt_test_source_connect_status
04.  bt_test_source_auto_stop
05.  bt_test_sink_open
06.  bt_test_sink_visibility00
07.  bt_test_sink_visibility01
08.  bt_test_sink_visibility10
09.  bt_test_sink_visibility11
10.  bt_test_sink_status
11.  bt_test_sink_music_play
12.  bt_test_sink_music_pause
13.  bt_test_sink_music_next
14.  bt_test_sink_music_previous
15.  bt_test_sink_music_stop
16.  bt_test_sink_reconnect_disenable
17.  bt_test_sink_reconnect_enable
18.  bt_test_sink_disconnect
19.  bt_test_sink_close
20.  bt_test_ble_start
21.  bt_test_ble_write
22.  bt_test_ble_stop
23.  bt_test_ble_get_status
24.  bt_test_spp_open
25.  bt_test_spp_write
26.  bt_test_spp_close
```

```
27.  bt_test_spp_status
Which would you like:
```

2、选择对应测试程序编号。首先要选择01进行初始化蓝牙基础服务。比如测试BT Source功能

```
Which would you like:01
#注：等待执行结束，进入下一轮选择界面。
Which would you like:02
#注：选择02前，要开启一个BT Sink设备，该设备处于可发现并可连接状态。02功能会自动扫描BT Sink设备并连接信号最强的那个设备。
```

5.3 BLE配网演示程序

1、手机端安装external/deviceio/test/apk/Rkble.apk。

2、设备端执行`wpa_supplicant -B -i wlan0 -c /data/cfg/wpa_supplicant.conf &`

3、ps命令查看第2步进程在后台运行。

4、设备端执行`DeviceIOTest blewifi`

5、打开手机端Rkble.apk，直接点击“CONTINUE”按钮（默认为BLE配网）。

6、点击“START SCAN”按钮，扫描ble设备。扫描到名为RockChipBle的设备，点击名称进行连接。

7、BLE连接成功后，会进入密码提示窗口。当前APK默认选中手机已连接的WIFI名称，若要主动选择则需点击“>>”按钮，弹窗显示设备端扫描到的wifi列表。选择你想要设置的网络名称。

8、输入密码，点击“Confirm”按钮，配网成功后APK界面下端会有弹窗提示配网成功或失败消息。

注：external/deviceio/test/apk/Rkble.zip为Rkble.apk源码