

RockChip DeviceIo 蓝牙接口介绍

文件标识: RK-SM-YF-343

发布版本: V2.0.2

日期: 2020-12-02

文件密级: ☐绝密 ☐秘密 ☐内部资料 ☒公开

免责声明

本文档按“现状”提供, 瑞芯微电子股份有限公司(“本公司”, 下同)不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因, 本文档将可能在未经任何通知的情况下, 不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标, 归本公司所有。

本文档可能提及的其他所有注册商标或商标, 由其各自拥有者所有。

版权所有 © 2020 瑞芯微电子股份有限公司

超越合理使用范畴, 非经本公司书面许可, 任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部, 并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址: 福建省福州市铜盘路软件园A区18号

网址: www.rock-chips.com

客户服务电话: +86-4007-700-590

客户服务传真: +86-591-83951833

客户服务邮箱: fae@rock-chips.com

前言

概述

该文档旨在介绍RockChip DeviceIo库的蓝牙接口。不同的蓝牙芯片模组对应的DeviceIo库也不同，对应关系如下所示：

bluez/libDeviceIo.so：基于BlueZ协议栈，主要适用于Realtek的蓝牙模组，如：RTL8723DS.

broadcom/libDeviceIo.so：基于BSA协议栈，主要适用于正基的蓝牙模组，如：AP6255.

cypress/libDeviceIo.so：基于BSA协议栈，主要适用于海华的蓝牙模组，如：AW-CM256.

用户在配置好SDK的蓝牙芯片类型后，deviceio编译脚本会根据用户选择的芯片类型自动选择libDeviceIo库。SDK的蓝牙芯片配置方法请参考《Rockchip_Developer_Guide_Network_Config_CN》中“WIFI/BT配置”章节。基于不同协议栈实现的DeviceIo库的接口尽可能做到了统一，但仍有部分接口有些区别，这些区别会在具体接口介绍时进行详细描述。

名词说明

BLUEZ DEVICEIO：基于BlueZ协议栈实现的DeviceIo库，对应bluez/libDeviceIo.so

BSA DEVICEIO：基于BSA协议栈实现的DeviceIo库，对应broadcom/libDeviceIo.so和cypress/libDeviceIo.so

BLUEZ only：接口或文档仅支持BLUEZ DEVICEIO。

BSA only：接口或文档仅支持BSA DEVICEIO。

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 | V1.0.0 | V1.0.x / V1.1.x | francis.fan | 初始版本（BLUEZ only） |
| 2019-4-16 | V1.1.0 | 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 | V1.2.0 | 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 |
| 2019-5-27 | V1.3.0 | V1.2.2 | francis.fan | 增加A2DP SOURCE 反向控制事件通知 添加HFP HF接口支持 添加蓝牙类设置接口 添加蓝牙自动重连属性设置接口 添加A2DP SINK 音量反向控制（BSA only） |
| 2019-6-4 | V1.4.0 | V1.2.3 | francis.fan | Bluez: 实现A2DP SINK 音量正反向控制 Bluez: 取消SPP与A2DP SINK的关联 Bluez: rk_bt_enable_reconnec 保存属性到文件，设备重启后属性设置依旧生效 Bluez: 修复A2DP SOURCE 反向控制功能初始化概率性失败 Bluez: 修复 rk_bt_sink_set_visibilit BSA: 修复A2DP SOURCE自动重连失败 BSA: 修复 rk_bt_hfp_hangup api 删除rk_bt_sink_set_auto_reconnect接口 |
| 2019-6-24 | V1.5.0 | V1.2.4 | ctf | 增加HFP HF alsa控制 demo 添加hfp断开连接api: rk_bt_hfp_disconnect 修复手机上接听电话和拒接电话时，无法收到PICKUP、HANGUP事件BUG Bsa: 增加HFP HF 使能CVSD（8K采样）接口 Bsa: 修复cypress bsa 配对弹窗提示问题 Bsa: 更新broadcom bsa 版本 (rockchip_20190617) Bsa: 修复蓝牙扫描时，无法识别个别蓝牙音箱设备类型BUG Bsa: 修复电池电量上报BUG |

| 日期 | 文档版本 | 对应库版本 | 作者 | 修改说明 |
|------------|--------|--------|-----|--|
| 2019-10-30 | V1.6.0 | V1.3.0 | ctf | <p>Bluez: 实现蓝牙反初始化</p> <p>Bluez: 修正获取本机设备名、本机蓝牙Mac地址接口</p> <p>Bluez: 添加pbap profile 支持</p> <p>Bluez: 支持hfp 8K和16K采样率自适应</p> <p>Bluez: 添加sink 播放underrun上报</p> <p>Bsa: 添加设置sink 播放设备节点接口</p> <p>Bsa: 添加ble可见性设置接口</p> <p>Bsa: 添加ble主动断开连接接口</p> <p>Bsa: 支持在蓝牙初始化时, 设置蓝牙地址</p> <p>添加蓝牙启动状态上报</p> <p>添加蓝牙配对状态上报</p> <p>添加启动蓝牙扫描、停止蓝牙扫描接口</p> <p>添加获取蓝牙是否处于扫描状态接口</p> <p>添加打印当前扫描到的设备列表接口</p> <p>添加主动和指定设备配对、取消和指定设备配对接口</p> <p>添加获取当前已配对设备列表、释放获取的配对设备列表接口</p> <p>添加打印当前配对设备列表接口</p> <p>添加设置本机设备名接口</p> <p>添加歌曲信息上报</p> <p>添加歌曲播放进度上报</p> <p>添加avdtp(a2dp sink)状态上报</p> <p>sink添加主动和指定设备连接、主动断开指定设备连接接口</p> <p>添加获取当前播放状态接口</p> <p>添加获取当前连接的远程设备是否支持主动上报播放进度接口</p> <p>支持打印日志到syslog</p> |
| 2019-11-16 | V1.7.0 | V1.3.1 | ctf | <p>source 回调增加连接设备的address和name参数</p> |
| 2019-12-12 | V1.8.0 | V1.3.2 | ctf | <p>Bluez: 实现ble client 功能</p> <p>Bluez: 实现obex文件传输功能</p> |

| 日期 | 文档版本 | 对应库版本 | 作者 | 修改说明 |
|------------|--------|--------|------|--|
| 2020-03-17 | V1.9.0 | V1.3.4 | ctf | <p>Bluez : 扫描接口添加类型过滤 (LE or BR/EDR or both)</p> <p>Bluez : 添加获取扫描设备列表接口</p> <p>Bluez : 添加启动bt source 后, 第一次扫描自动回连最后一个连接的sink设备</p> <p>Bluez : 修正扫描期间连接设备失败BUG</p> <p>Bluez : 优化init、deinit执行时间</p> <p>Bluez : 修正在qt 非主mianloop线程启动蓝牙, 线程同步BUG</p> <p>Bluez : 添加source 断开连接失败、自动回连事件上报</p> <p>Bluez : 添加source 断开当前连接接口</p> <p>Bluez : 添加获取指定设备连接状态</p> <p>Bluez : 修正ble 初始化内存越界问题</p> <p>Bsa : 添加设置bsa_server.sh 路径接口</p> <p>Ble 状态回调带远程设备地址和名称</p> |
| 2020-07-08 | V2.0.0 | V1.3.5 | ctf | <p>修正一些bluez和bsa bug, 详情见Rk_system.h V1.3.5 说明</p> <p>添加设置ble 广播间隔接口</p> <p>添加hfp 拨打指定电话号码接口</p> <p>rk_ble_client_write 增加写数据长度参数</p> <p>支持ble MTU 上报</p> <p>ble client 添加获取ble 设备广播api</p> <p>Bluez : 添加obex 状态回调</p> <p>Bluez : 添加设置ble 地址接口</p> <p>Bluez : ble特征值添加 write-without-response 属性</p> <p>Bsa : 添加rk_bt_source_disconnect 接口</p> <p>Bsa : 支持LE BR/EDR 过滤扫描</p> <p>Bsa : 添加source 第一次扫描自动重连上一次连接的sink 设备</p> <p>Bsa : 支持ble client功能</p> <p>Bsa : 添加读取远程连接设备名接口</p> <p>Bsa : 添加获取当前扫描设备列表接口</p> |
| 2020-07-12 | V2.0.1 | V1.3.5 | Ruby | 修订格式 |

| 日期 | 文档版本 | 对应库版本 | 作者 | 修改说明 |
|------------|--------|--------|-----|---|
| 2020-12-02 | V2.0.2 | V1.3.6 | ctf | 添加获取设备平台类型（ios/非ios）接口 修正一些Bluez和Bsa Bug，详情见Rk_system.h V1.3.6 说明 Bsa：添加蓝牙电话簿功能 Bsa：hfp添加来电号码上报 Bsa：hfp添加来电信息（来电号码、联系人）上报 Bsa：hfp添加去电事件、去电响铃事件上报 Broadcom Bsa：添加PAN 蓝牙共享网络功能 |

目录

RockChip DeviceIo 蓝牙接口介绍

1. 蓝牙基础接口 (RkBtBase.h)
2. BLE接口介绍 (RkBle.h)
3. BLE CLIENT接口介绍 (RkBleClient.h)
4. SPP接口介绍 (RkBtSpp.h)
5. A2DP SINK接口介绍 (RkBtSink.h)
6. A2DP SOURCE接口介绍 (RkBtSource.h)
7. HFP-HF接口介绍 (RkBtHfp.h)
8. OBEX 接口介绍 (RkBtObex.h)
9. PAN 接口介绍 (RkPan.h Broadcom BSA only)
 - 9.1 蓝牙共享网络配置
10. 示例程序说明
 - 10.1 编译说明
 - 10.2 基础接口演示程序
 - 10.2.1 接口说明
 - 10.2.1.1 蓝牙服务的基础接口测试说明
 - 10.2.1.2 BLE接口测试说明
 - 10.2.1.3 BLE CLIENT接口测试说明
 - 10.2.1.4 A2DP SINK接口测试说明
 - 10.2.1.5 A2DP SOURCE接口测试说明
 - 10.2.1.6 SPP接口测试说明
 - 10.2.1.7 HFP接口测试说明
 - 10.2.1.8 OBEX接口测试说明
 - 10.2.1.9 PAN接口测试说明
 - 10.2.2 测试步骤
 - 10.3 BLE配网演示程序

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 ble_addr[DEVICE_ADDR_LEN];     //BLE 地址, 默认使用随机地址 (BLUEZ Only)
    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
    const char *bt_addr;      //蓝牙地址 (Bsa only, 默认使用芯片内部固化的bt mac地址)
} RkBtContent;
```

- RkBtScanedDevice 结构

```
typedef struct scanned_dev {
    char *remote_address; //远程设备地址
    char *remote_name;    //远程设备名
    unsigned int cod;      //class of device
    bool is_connected;     //该远程设备当前是否处于连接状态(sink, source, hfp)
    struct paired_dev *next; //指向下一个设备
} RkBtScanedDevice;
```


- RK_BT_STATE 说明

```
typedef enum {  
    RK_BT_STATE_OFF,           //已关闭  
    RK_BT_STATE_ON,           //已开启  
    RK_BT_STATE_TURNING_ON,    //正在开启  
    RK_BT_STATE_TURNING_OFF,   //正在关闭  
} RK_BT_STATE;
```

- RK_BT_BOND_STATE 说明

```
typedef enum {  
    RK_BT_BOND_STATE_NONE,     //配对失败或取消配对  
    RK_BT_BOND_STATE_BONDING,   //正在配对  
    RK_BT_BOND_STATE_BONDED,    //配对成功  
} RK_BT_BOND_STATE;
```

- RK_BT_SCAN_TYPE 说明

```
typedef enum {  
    SCAN_TYPE_AUTO,           //LE and BR/EDR, 扫描所有类型设备  
    SCAN_TYPE_BREDR,          //只扫描BR/EDR类型设备  
    SCAN_TYPE_LE,             //只扫描LE类型设备  
    SCAN_TYPE_PAN              //过滤扫描带有PAN Service的设备 (Broadcom BSA only)  
} RK_BT_SCAN_TYPE;
```

- RK_BT_DISCOVERY_STATE 说明

```
typedef enum {  
    RK_BT_DISC_STARTED,       //开始扫描成功  
    RK_BT_DISC_STOPPED_AUTO,   //扫描完成, 自动停止扫描  
    RK_BT_DISC_START_FAILED,    //开始扫描失败  
    RK_BT_DISC_STOPPED_BY_USER, //通过rk_bt_cancel_discovery, 中断扫描  
} RK_BT_DISCOVERY_STATE;
```

- RK_BT_PLAYROLE_TYPE 说明

```
typedef enum {  
    PLAYROLE_TYPE_UNKNOWN,    //未知设备  
    PLAYROLE_TYPE_SOURCE,     //a2dp Source 设备  
    PLAYROLE_TYPE_SINK,       //a2dp Sink 设备  
} RK_BT_PLAYROLE_TYPE;
```

- typedef void (*RK_BT_STATE_CALLBACK)(RK_BT_STATE state) ``typedef void (*RK_BT_STATE_CALLBACK)(RK_BT_STATE state) 蓝牙状态回调

- `typedef void (*RK_BT_BOND_CALLBACK)(const char *bd_addr, const char *name, RK_BT_BOND_STATE state)`

蓝牙配对状态回调，**bd_addr**: 当前配对设备的地址，**name**: 当前配对设备的名称

- `typedef void (*RK_BT_DISCOVERY_CALLBACK)(RK_BT_DISCOVERY_STATE state)`

蓝牙扫描状态回调，如果使用**rk_bt_start_discovery** 扫描周围蓝牙设备，则需要注册该回调

- `typedef void (*RK_BT_DEV_FOUND_CALLBACK)(const char *address, const char *name, unsigned int bt_class, int rssi)`

蓝牙设备扫描回调，如果使用**rk_bt_start_discovery** 扫描周围蓝牙设备，则需要注册该回调。**bluez**每扫描到一个设备，触发一次该回调；**bsa**扫描结束后，才会根据扫描到的设备数依次触发该回调。

- `typedef void (*RK_BT_NAME_CHANGE_CALLBACK)(const char *bd_addr, const char *name)`

远程设备名更新回调

- `typedef void (*RK_BT_MTU_CALLBACK)(const char *bd_addr, unsigned int mtu)`

ble MTU回调，**ble** 和 **ble client** 共用，MTU协商成功后，触发该回调

- `void rk_bt_register_state_callback(RK_BT_STATE_CALLBACK cb)`

注册获取蓝牙启动状态的回调函数

- `void rk_bt_register_bond_callback(RK_BT_BOND_CALLBACK cb)`

注册获取蓝牙配对状态的回调函数

- `void rk_bt_register_discovery_callback(RK_BT_DISCOVERY_CALLBACK cb)`

注册获取蓝牙扫描状态的回调函数

- `void rk_bt_register_dev_found_callback(RK_BT_DEV_FOUND_CALLBACK cb)`

注册发现设备的回调函数

- `void rk_bt_register_name_change_callback(RK_BT_NAME_CHANGE_CALLBACK cb)`

注册设备名更新回调函数

- `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/HFP** 任意一个服务处于连接状态，该函数都返回1；否则返回0。

- `int rk_bt_set_class(int value)`

设置蓝牙设备类型。**value**: 类型值。比如0x240404对应的含义为: Major Device Class: Audio/Video
Minor Device Class: Wearable headset device

Service Class: Audio (Speaker, Microphone, Headset service), Rendering (Printing, Speaker)

- `int rk_bt_enable_reconnect(int value)`

启动/关闭HFP/A2DP SINK的自动重连功能。**value**: 0表示关闭自动重连功能，1表示开启自动重连功能。

- `void rk_bt_start_discovery(unsigned int mseconds, RK_BT_SCAN_TYPE scan_type)`

启动蓝牙扫描，mseconds: 扫描时长，单位毫秒；scan_type: 扫描类型，详情参见RK_BT_SCAN_TYPE说明，仅bluez支持扫描类型过滤，bsa仅支持全类型扫描。

- `void rk_bt_cancel_discovery()`

停止蓝牙扫描，取消rk_bt_start_discovery 发起的扫描操作

- `bool rk_bt_is_discovering()`

蓝牙是否处于扫描周围设备的状态，正在扫描设备返回true，否则返回false

- `void rk_bt_display_devices()`

打印显示当前扫描到的设备列表

- `int rk_bt_pair_by_addr(char *addr)`

主动和addr指定的设备配对；addr: 设备地址，比如： 94:87:E0:B6:6D:AE

- `int rk_bt_unpair_by_addr(char *addr)`

取消和addr指定的设备的配对，取消配对后，会删除该设备的所有记录；addr: 设备地址

- `int rk_bt_set_device_name(char *name)`

设置本机设备名，name: 想要设置的设备名

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

获取本机设备名，name: 用于存储获取的设备名，len: 设备名长度

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

获取本机设备蓝牙mac地址，addr: 用于存储获取到的mac地址，len: mac地址长度

- `void rk_bt_display_paired_devices()`

打印当前已配对的设备列表

- `int rk_bt_get_paired_devices(RkBtScannedDevice**dev_list,int *count)`

获取当前已配对的设备列表，dev_list: 用于存储已配对的设备列表，count: 已配对的设备数

- `int rk_bt_free_paired_devices(RkBtScannedDevice*dev_list)`

释放rk_bt_get_paired_devices 分配的用于存储设备列表的内存

- `int rk_bt_get_scanned_devices(RkBtScannedDevice**dev_list,int *count)`

获取当前已扫描到的设备列表，dev_list: 用于存储已扫描的设备列表，count: 已扫描到的设备数

- `int rk_bt_free_scanned_devices(RkBtScannedDevice*dev_list)`

释放rk_bt_get_scanned_devices分配的用于存储设备列表的内存

- `void rk_bt_set_bsa_server_path(char *path)`

设置bsa_server.sh路径，默认为/usr/bin/bsa_server.sh（BSA only）

- `bool rk_bt_get_connected_properties(char *addr)`

获取addr指定设备的连接状态，addr: 设备地址，处于连接状态 则返回true，反之返回false（BLUEZ only）

- `int rk_bt_set_visibility(const int visible, const int connectal)`

设置可见/可连接特性。visible: 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。只适用于BR/EDR设备

- `RK_BT_PLAYROLE_TYPE rk_bt_get_playrole_by_addr(char *addr)`

获取addr指定设备的playrole，详见RK_BT_PLAYROLE_TYPE 说明。

- `int rk_bt_read_remote_device_name(char *addr, int transport)`

读取addr指定设备的设备名，transport 指定该设备类型，未知设备：RK_BT_TRANSPORT_UNKNOWN，BR/EDR设备：RK_BT_TRANSPORT_BR_EDR，LE 设备：RK_BT_TRANSPORT_LE。该接口需和rk_bt_register_name_change_callback配套使用，读取成功会触发RK_BT_NAME_CHANGE_CALLBACK 回调（BSA only）

- `RK_BT_DEV_PLATFORM_TYPE rk_bt_get_dev_platform(char *addr)`

获取addr指定设备的类型：返回RK_BT_DEV_PLATFORM_IOS代表ios平台，返回RK_BT_DEV_PLATFORM_UNKNOWN代表非ios平台

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)(const char *bd_addr, const char *name, RK_BLE_STATE state)`

BLE状态回调函数，bd_addr： 远程设备地址，name： 远程设备名称。

- `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 DEVICEIO只能通过rk_bt_init()接口指定，而对BSA DEVICEIO来说只能使用该接口注册接收回调函数。

- `void rk_ble_register_mtu_callback(RK_BT_MTU_CALLBACK cb)`

该接口用于注册mtu回调，mtu协商成功后触发RK_BT_MTU_CALLBACK 回调，上报协商后的mtu值

- `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

- `int rk_bt_ble_set_visibility(const int visible, const int connect)`

设置ble可见/可连接特性。visible: 0表示不可见, 1表示可见。connect: 0表示不可连接, 1表示可连接。该接口仅适用于bsa (BSA only)

- `int rk_ble_disconnect(void)`

主动断开当前ble连接

- `int rk_ble_set_address(char *address)`

设置ble地址, 也可在rk_bt_init时使用ble_addr 参数设置, 未设置默认随机地址 (BLUEZ Only)

- `int rk_ble_set_adv_interval(unsigned short adv_int_min, unsigned short adv_int_max)`

设置ble广播间隔, adv_int_min最小广播间隔, adv_int_max最大广播间隔, 最小值32(32 * 0.625ms = 20ms), 未设置时bsa 默认间隔30ms, bluez 默认100ms。

3. BLE CLIENT接口介绍 (RkBleClient.h)

- `RK_BLE_CLIENT_STATE` 说明

```
typedef enum {  
    RK_BLE_CLIENT_STATE_IDLE,           //空闲状态  
    RK_BLE_CLIENT_STATE_CONNECT,       //连接成功  
    RK_BLE_CLIENT_STATE_DISCONNECT     //断开连接  
} RK_BLE_CLIENT_STATE;
```

- `RK_BLE_CLIENT_SERVICE_INFO` 说明

```
typedef struct {  
    int service_cnt;                      //连接的远程设备包含的service数量  
    RK_BLE_CLIENT_SERVICE service[SERVICE_COUNT_MAX]; //每个service的具体信息  
} RK_BLE_CLIENT_SERVICE_INFO;
```

```
typedef struct {  
    char describe[DESCRIBE_BUF_LEN];      //uuid 对应的描述  
    char path[PATH_BUF_LEN];  
    char uuid[UUID_BUF_LEN];              //service uuid  
    int chrc_cnt;                          //该service 包含的characteristic数量  
    RK_BLE_CLIENT_CHRC chrc[CHRC_COUNT_MAX]; //每个characteristic的具体信息  
} RK_BLE_CLIENT_SERVICE;
```

```
typedef struct {
```

```

char describe[DESCRIBE_BUG_LEN];           //uuid 对应的描述
char path[PATH_BUF_LEN];
char uuid[UUID_BUF_LEN];                   //characteristic uuid
unsigned int props;                         //characteristic 属性
unsigned int ext_props;                     //characteristic 扩展属性
unsigned int perm;                          //characteristic 权限
bool notifying;                            //characteristic 是否开启notification(BLUEZ
only)
int desc_cnt;                              //该characteristic 包含的descriptor 数量
RK_BLE_CLIENT_DESC desc[DESC_COUNT_MAX]; //每个descriptor 的具体信息
} RK_BLE_CLIENT_CHRC;

typedef struct {
char describe[DESCRIBE_BUG_LEN];           //uuid 对应的描述
char path[PATH_BUF_LEN];
char uuid[UUID_BUF_LEN];                   //descriptor uuid
} RK_BLE_CLIENT_DESC;

```

注：path 用来指示service、characteristic、descriptor之间的关系，用于遍历查找，应用层不需要care这个参数，是bluez 独有的

- `typedef void (*RK_BLE_CLIENT_STATE_CALLBACK)(const char *bd_addr, const char *name, RK_BLE_CLIENT_STATE state)`

ble client 状态回调函数, bd_addr: 远程设备地址, name: 远程设备名称。

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

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

- `void rk_ble_client_register_state_callback(RK_BLE_CLIENT_STATE_CALLBACK cb)`

注册ble client 状态回调函数

- `int rk_ble_client_register_recv_callback(RK_BLE_CLIENT_RECV_CALLBACK cb)`

注册ble client 数据接收回调函数

- `int rk_ble_client_open(bool mtu_change)`

初始化ble client, mtu_change: 连接成功后是否自动协商mtu, 该参数只对Bsa协议有效。

- `void rk_ble_client_close(void)`

反初始胡ble client

- `RK_BLE_CLIENT_STATE rk_ble_client_get_state()`

主动获取ble client状态

- `int rk_ble_client_get_service_info(char *address, RK_BLE_CLIENT_SERVICE_INFO *info)`

获取address指定设备的信息, 包括service uuid, characteristic uuid、permission、Properties, descriptor uuid等, 具体可查看RK_BLE_CLIENT_SERVICE_INFO 结构体

- `int rk_ble_client_write(const char *uuid, char *data, int data_len)`

往对端指定uuid发送数据, data: 数据指针, len: 数据长度。

- `int rk_ble_client_read(const char *uuid)`

读取对端指定uuid的数据, 读取成功会触发RK_BLE_CLIENT_RECV_CALLBACK回调

- `int rk_ble_client_connect(char *address)`

连接指定address的设备

- `int rk_ble_client_disconnect(char *address)`

断开与address指定设备的连接

- `bool rk_ble_client_is_notifying(const char *uuid)`

查询指定uuid是否开启notification，开启则返回true（BLUEZ only）

- `int rk_ble_client_notify(const char *uuid, bool enable)`

设置指定uuid的notification，该uuid本身必须支持notifications或indications，enable = true时开启，enable = false时关闭。开启是远程设备（server）write 该uuid时，会触发RK_BLE_CLIENT_RECV_CALLBACK回调，自动上报修改的值。

- `int rk_ble_client_get_eir_data(char *address, char *eir_data, int len)`

获取address指定远程设备的广播数据，eir_data: 获取到的广播数据，len: 广播数据长度

- `int rk_ble_client_default_data_length()`

强制指定hci写数据长度为27byte，为特定客户定制的，一般情况下不使用该api（BSA only）

4. 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。

- `int rk_bt_spp_close(void)`

关闭SPP。

- `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。

5. A2DP SINK接口介绍（RkBtSink.h）

- BtTrackInfo 结构

```
typedef struct btmg_track_info_t {
    char title[256];           //标题
    char artist[256];          //艺术家
    char album[256];           //专辑
    char track_num[64];         //该歌曲处于专辑的第几首
    char num_tracks[64];        //该专辑总曲目数
    char genre[256];           //流派
    char playing_time[256];     //歌曲播放总长度
} btmg_track_info_t;

typedef struct btmg_track_info_t BtTrackInfo;
```

- 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_DISCONNECT          //断开连接
    RK_BT_SINK_STATE_PLAY ,              //avrcp播放状态
    RK_BT_SINK_STATE_PAUSE,              //avrcp暂停状态
    RK_BT_SINK_STATE_STOP,               //avrcp停止状态
    RK_BT_A2DP_SINK_STARTED,             //avdtp播放状态
    RK_BT_A2DP_SINK_SUSPENDED,           //avdtp暂停状态
    RK_BT_A2DP_SINK_STOPPED,             //avdtp停止状态
} RK_BT_SINK_STATE;
```

avdtp状态主要用于微信通话和微信语音时a2dp sink状态的上报，因为此时不会触发avrcp状态变更。

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

状态回调函数。

- `typedef void (*RK_BT_SINK_VOLUME_CALLBACK)(int volume)`

音量变化回调函数。当手机端音量变化时，调用该回调函数。volume: 新的音量值。注：因AVRCP版本以及不同手机厂商实现不同，因此有的手机并不兼容该功能。iPhone系列手机对该接口支持良好。

- `typedef void (*RK_BT_AVRCP_TRACK_CHANGE_CB)(const char *bd_addr, BtTrackInfo track_info)`

歌曲信息回调函数，当播放歌曲变化时，会触发该回调。bd_addr: 远程设备地址, track_info: 歌曲信息

- `typedef void (*RK_BT_AVRCP_PLAY_POSITION_CB)(const char *bd_addr, int song_len, int song_pos)`

歌曲播放进度回调，当远程设备支持position change时，会自动上报播放进度，触发该函数。bd_addr: 远程设备地址，song_len: 歌曲总长度，song_pos: 当前播放进度

- `typedef void (*RK_BT_SINK_UNDERRUN_CB)(void)`

播放underrun状态回调，当播放出现underrun时自动触发，该接口只适用于bluez（Bluez only）

- `int rk_bt_sink_register_callback(RK_BT_SINK_CALLBACK cb)`

注册状态回调函数。

- `int rk_bt_sink_register_volume_callback(RK_BT_SINK_VOLUME_CALLBACK cb)`

注册音量变化回调函数。

- `int rk_bt_sink_register_track_callback(RK_BT_AVRCP_TRACK_CHANGE_CB cb)`

注册歌曲信息回调函数

- `int rk_bt_sink_register_position_callback(RK_BT_AVRCP_PLAY_POSITION_CB cb)`

注册歌曲播放进度回调

- `void rk_bt_sink_register_underurn_callback(RK_BT_SINK_UNDERRUN_CB cb)`

注册underrun回调函数，该接口只适用于bluez（Bluez only）

- `int rk_bt_sink_open()`

打开A2DP SINK服务。如需要A2DP SINK与HFP并存，请参见<HFP-HF接口介绍>章节中rk_bt_hfp_sink_open接口。

- `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。音量范围[0, 127]，调用该接口，音量每次增加8。

注：因AVRCP版本以及不同手机厂商实现不同，因此有的手机并不兼容该功能。iPhone系列手机对该接口支持良好。

- `int rk_bt_sink_volume_down(void)`

反向控制：音量减小。音量范围[0, 127]，调用该接口，音量每次减小8。

注：因AVRCP版本以及不同手机厂商实现不同，因此有的手机并不兼容该功能。iPhone系列手机对该接口支持良好。

- `int rk_bt_sink_set_volume(int volume)`

反向控制：设置A2DP SOURCE端音量。volume取值范围[0, 127]。若超过取值范围，该接口内部自动修正。

注：因AVRCP版本以及不同手机厂商实现不同，因此有的手机并不兼容该功能。iPhone系列手机对该接口支持良好。

- `int rk_bt_sink_disconnect()`

断开A2DP Sink连接。

- `int rk_bt_sink_connect_by_addr(char *addr)`

主动连接addr指定的设备；addr: 设备地址，类似“94:87:E0:B6:6D:AE”

- `int rk_bt_sink_disconnect_by_addr(char *addr)`

主动断开addr指定设备的连接；addr: 设备地址，类似“94:87:E0:B6:6D:AE”

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

获取当前连接的远程设备的地址 (BLUEZ only)

- `int rk_bt_sink_get_play_status()`

获取当前连接的远程设备的播放状态，当远程设备不支持主动上报播放进度时，可以通过该接口获取播放进度，调用该接口会触发RK_BT_AVRCP_PLAY_POSITION_CB回调

- `bool rk_bt_sink_get_poschange()`

当前连接的远程设备是否支持主动上报播放进度，支持则返回true，否则返回false

- `void rk_bt_sink_set_alsa_device(char *alsa_dev)`

设置蓝牙播放设备节点，必须在rk_bt_sink_open后面调用。默认使用default，该接口仅适用于bsa (BSA only)
bluez播放设备节点位于external/bluez-alsa/utils/aplay.c，可自行修改

6. 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 or Audio Source or 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_DISCONNECT_FAILED,    //断开连接失败 (BLUEZ only)
    BT_SOURCE_EVENT_DISCONNECTED,        //断开连接
    /* Sink端反向控制事件 */
    BT_SOURCE_EVENT_RC_PLAY,             //播放
    BT_SOURCE_EVENT_RC_STOP,             //停止
    BT_SOURCE_EVENT_RC_PAUSE,            //暂停
    BT_SOURCE_EVENT_RC_FORWARD,          //上一首
    BT_SOURCE_EVENT_RC_BACKWARD,         //下一首
    BT_SOURCE_EVENT_RC_VOL_UP,           //音量+
    BT_SOURCE_EVENT_RC_VOL_DOWN,         //音量-
    BT_SOURCE_EVENT_AUTO_RECONNECTING,   //正在回连 (BLUEZ only)
} 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 char *bd_addr, const char *name, const RK_BT_SOURCE_EVENT event)`

状态回调函数。**userdata**: 用户指针, **bd_addr**: 连接的远程设备的地址, **name**: 连接的远程设备的名称, **event**: 连接事件。建议在`rk_bt_source_open`接口之前注册状态回调函数, 以免状态事件丢失。

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

注册状态回调函数。

- `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。该接口自动扫描时长为10秒，若10秒内每扫描到任何一个Audio Sink类型设备，该接口则不会做任何操作。若扫描到Audio Sink类型设备，则会打印出设备的基本信息，如果扫描不到Audio Sink设备则会打印“==== Cannot find audio Sink devices. ====”；若扫到的设备信号强度太低，则也会连接失败，并打印“==== BT SOURCE RSSI is is too weak !!! ====”。

- `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)`

获取本机设备名称。**name**: 存放名称的buffer，**len**: **name**空间大小。

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

获取本机设备地址。**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_by_addr(char *address)`

主动连接**address**指定的设备。

- `int rk_bt_source_disconnect_by_addr(char *address)`

断开与**address**指定设备的连接

- `int rk_bt_source_disconnect()`

断开当前连接。

- `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_resume(void)`

继续播放（BSA only）

- `int rk_bt_source_stop(void)`

停止播放（BSA only）

- `int rk_bt_source_pause(void)`

暂停播放（BSA only）

- `int rk_bt_source_vol_up(void)`

音量增（BSA only）

- `int rk_bt_source_vol_down(void)`

音量减（BSA only）

7. HFP-HF接口介绍（RkBtHfp.h）

- `RK_BT_HFP_EVENT` 介绍

```
typedef enum {
    RK_BT_HFP_CONNECT_EVT,           // HFP 连接成功
    RK_BT_HFP_DISCONNECT_EVT,        // HFP 断开连接
    RK_BT_HFP_RING_EVT,              // 收到AG(手机)的振铃信号
    RK_BT_HFP_AUDIO_OPEN_EVT,        // 接通电话
    RK_BT_HFP_AUDIO_CLOSE_EVT,       // 挂断电话
    RK_BT_HFP_PICKUP_EVT,            // 主动接通电话
    RK_BT_HFP_HANGUP_EVT,            // 主动挂断电话
    RK_BT_HFP_VOLUME_EVT,            // AG(手机)端音量改变
    RK_BT_HFP_BCS_EVT,               // AG(手机)Codec 类型: CVSD 或 mSBC
    RK_BT_HFP_CLIP_EVT,              // 来电信息: 电话号码、联系人(仅支持英文)
    RK_BT_HFP_OUTGOING_CALL_DIAL_EVT, // 去电
    RK_BT_HFP_OUTGOING_CALL_RING_EVT, // 去电响铃
    RK_BT_HFP_CLCC_EVT,              // 通话时信息: 电话号码、联系人(仅支持英文)
} RK_BT_HFP_EVENT;
```

- `RK_BT_SCO_CODEC_TYPE` 介绍

```
typedef enum {
    BT_SCO_CODEC_CVSD,           // CVSD (8K 采样), 蓝牙要求强制支持
    BT_SCO_CODEC_MSBC,           // mSBC (16K 采样), 可选支持
} RK_BT_SCO_CODEC_TYPE;
```

- `typedef int (*RK_BT_HFP_CALLBACK)(RK_BT_HFP_EVENT event, void *data)`

HFP状态回调函数。**event**: 参见上述 `RK_BT_HFP_EVENT` 介绍。**data**: 当**event**为 `RK_BT_HFP_VOLUME_EVT` 时, `*((int *)data)` 为当前AG（手机）端显示的音量值。*注: 实际通话音量仍需要在板端做相应处理。*

- `void rk_bt_hfp_register_callback(RK_BT_HFP_CALLBACK cb)`

注册HFP回调函数。该函数推荐在 `rk_bt_hfp_sink_open` 之前调用，这样避免状态事件丢失。

- `int rk_bt_hfp_sink_open(void)`

同时打开HFP-HF与A2DP SINK功能。BSA DEVICEIO可调用该接口，也可以单独调用A2DP Sink打开和HFP的打开接口，实现HFP-HF和A2DP SINK共存。而BLUEZ DEVICEIO则只能通过该接口实现HFP-HF与A2DP SINK并存。

调用该接口时，A2DP SINK 与 HFP的回调函数以及功能接口仍是独立分开的，

`rk_bt_hfp_register_callback` 与 `rk_bt_sink_register_callback` 最好在 `rk_bt_hfp_sink_open` 之前调用，以免丢失事件。对于BLUEZ DEVICEIO来说，在调用 `rk_bt_hfp_sink_open` 接口之前，不能调用 `rk_bt_hfp_open` 和 `rk_bt_sink_open` 函数，否则该接口返回错误（`return -1`）。参考代码如下：

```
/* 共存方式打开A2DP SINK 与 HFP HF功能 */
rk_bt_sink_register_callback(bt_sink_callback);
rk_bt_hfp_register_callback(bt_hfp_hp_callback);
rk_bt_hfp_sink_open();
```

```
/* 关闭操作 */
rk_bt_hfp_close(); //关闭HFP HF
rk_bt_sink_close(); //关闭A2DP SINK
```

- `int rk_bt_hfp_open(void)`

打开HFP服务。

BLUEZ DEVICEIO: 该接口与 `rk_bt_sink_open` 互斥, 调用该接口会自动退出A2DP协议相关服务, 然后启动HFP服务。若需要A2DP SINK 与 HFP 并存, 参见 `rk_bt_hfp_sink_open`。

BSA DEVICEIO: 该接口与 `rk_bt_sink_open` 不存在互斥情况。

- `int rk_bt_hfp_close(void)`

关闭HFP服务。

- `int rk_bt_hfp_pickup(void)`

主动接听电话。

- `int rk_bt_hfp_hangup(void)`

主动挂断电话。

- `int rk_bt_hfp_redial(void)`

重播通话记录中最近一次呼出的电话号码。注意是“呼出”的电话号码, 而不是通话记录中最近一次的电话号码。比如如下场景中, 调用 `rk_bt_hfp_redial` 接口, 则会回拨rockchip-003。

<1> rockchip-001 [呼入]

<2> rockchip-002 [呼入]

<3> rockchip-003 [呼出]

- `int rk_bt_hfp_dial_number(char *number)`

拨打number指定电话号码

- `int rk_bt_hfp_report_battery(int value)`

电池电量上报。value: 电池电量值, 取值范围[0, 9]。

- `int rk_bt_hfp_set_volume(int volume)`

设置AG (手机) 的Speaker音量。volume: 音量值, 取值范围[0, 15]。对于AG设备是手机来说, 调用该接口后, 手机端蓝牙通话的音量进度条会做相应改变。但实际通话音量仍需要在板端做相应处理。

- `void rk_bt_hfp_enable_cvds(void)`

hfp codec强制使用CVSD (8K 采样率), AG (手机) 和 HF (耳机) 不会再协商SCO codec类型, 此时SCO codec类型必须强制设为BT_SCO_CODEC_CVSD。该接口只适用于bsa (BSA only)。

bluez支持8K和16K采样率自适应, SCO codec 类型由AG (手机) 和 HF (耳机) 协商决定, 不支持强制使用CVSD。

- `void rk_bt_hfp_disable_cvds(void)`

禁止hfp codec强制使用CVSD（8K 采样率），SCO codec 类型由AG（手机）和 HF（耳机）协商决定，协商结果通过回调事件RK_BT_HFP_BCS_EVT告知应用层。该接口只适用于bsa (BSA only)。

- `int rk_bt_hfp_disconnect(void)`

断开当前连接

8. OBEX 接口介绍 (RkBtObex.h)

- RK_BT_OBEX_STATE 介绍

```
typedef enum {  
    RK_BT_OBEX_CONNECT_FAILED,           //连接失败  
    RK_BT_OBEX_CONNECTED,               //连接成功  
    RK_BT_OBEX_DISCONNECT_FAILED,       //断连失败  
    RK_BT_OBEX_DISCONNECTED,           //断连成功  
    RK_BT_OBEX_TRANSFER_ACTIVE,         //开始传输  
    RK_BT_OBEX_TRANSFER_COMPLETE,       //传输完成  
} RK_BT_OBEX_STATE;
```

- `typedef void (*RK_BT_OBEX_STATE_CALLBACK)(const char *bd_addr, RK_BT_OBEX_STATE state);`

obex 状态回调，bd_addr: 连接的远程设备的地址

- `void rk_bt_obex_register_status_cb(RK_BT_OBEX_STATE_CALLBACK cb)`

注册obex 状态回调

- `int rk_bt_obex_init(char *path)`

启动obexd进程，蓝牙文件传输功能只需要调用该接口即可，path: 文件存储路径。(BLUEZ only)

- `int rk_bt_obex_deinit()`

关闭obexd进程，和rk_bt_obex_init配套使用。(BLUEZ only)

- `int rk_bt_obex_pbap_init()`

初始化蓝牙电话簿。Bluez协议调用该接口之前必须先调用rk_bt_obex_init启动obexd。

- `int rk_bt_obex_pbap_deinit()`

反初始化蓝牙电话簿。Bluez协议调用该接口之后必须调用rk_bt_obex_deinit关闭obexd。

- `int rk_bt_obex_pbap_connect(char *btaddr)`

主动和btaddr指定的设备连接

- `int rk_bt_obex_pbap_get_vcf(char *dir_name, char *dir_file)`

获取dir_name指定的对象类型的信息，存储在dir_file指定的文件中

pbab定义了六种对象类型：

"pb": 联系人电话本

"ich": 来电历史记录

"och": 拨出历史记录

"mch": 未接电话历史记录

"cch": 组合历史记录, 即来电、拨出和未接全部记录

"spd": 快速拨号, 比如可以指定按键1为某联系人的快速拨号按键

"fav": 收藏夹

- `int rk_bt_obex_pbap_disconnect(char *btaddr)`

主动断开和btaddr指定设备的连接

9. PAN 接口介绍 (RkPan.h Broadcom BSA only)

- RK_BT_PAN_EVENT 介绍

```
typedef enum {  
    RK_BT_PAN_CONNECT_FAILED,           //连接失败  
    RK_BT_PAN_CONNECT,                  //连接成功  
    RK_BT_PAN_DISCONNECT                 //断连成功  
} RK_BT_PAN_EVENT;
```

- `typedef void (*RK_BT_PAN_EVENT_CALLBACK)(RK_BT_PAN_EVENT event, char *bd_addr)`

pan事件回调, bd_addr: 连接的远程设备的地址

- `void rk_bt_pan_register_event_cb(RK_BT_PAN_EVENT_CALLBACK cb)`

注册pan 事件回调

- `int rk_bt_pan_open()`

初始化蓝牙共享网络

- `int rk_bt_pan_close()`

反初始化蓝牙共享网络

- `int rk_bt_pan_connect(char *address)`

连接address指定的设备

- `int rk_bt_pan_disconnect(char *address)`

断开和address指定的设备的连接

9.1 蓝牙共享网络配置

使用Bsa 蓝牙共享网络功能, kernel需要开启CONFIG_NET_CORE 和 CONFIG_TUN 配置, 否则初始化PAN时会出现 ‘app_pan_tap_open: could not open /dev/net/tun, err:2’ 错误。开启配置后要确保配置选项有更新到对应的defconfig 中, 以64bit 3308 平台为例:


```
--- a/arch/arm64/configs/rk3308_linux_defconfig
+++ b/arch/arm64/configs/rk3308_linux_defconfig
@@ -91,7 +91,8 @@ CONFIG_DEVTMPFS_MOUNT=y
 # CONFIG_ALLOW_DEV_COREDUMP is not set
 # CONFIG_BLK_DEV is not set
 CONFIG_NETDEVICES=y
-# CONFIG_NET_CORE is not set
+CONFIG_NET_CORE=y
+CONFIG_TUN=y
```

10. 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。

10.1 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/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。

10.2 基础接口演示程序

10.2.1 接口说明

10.2.1.1 蓝牙服务的基础接口测试说明

- void bt_test_bluetooth_init(char *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。对应DeviceIOTest.cpp测试菜单中的“bt_server_open”。

注：BLE 读数据是通过注册回调函数实现。当BLE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_bluetooth_deinit(char *data)
蓝牙反初始化测试，反初始化所以蓝牙profile。
- bt_test_set_class(char *data)

设置蓝牙设备类型。当前测试值为0x240404.

- `bt_test_enable_reconnect(void *data)`

使能A2DP SINK 和 HFP 自动重连功能。推荐紧跟在`bt_test_bluetooth_init`后调用。

- `bt_test_disable_reconnect(char *data)`

禁用A2DP SINK 和 HFP 自动重连功能。推荐紧跟在`bt_test_bluetooth_init`后调用。

手机端

- `void bt_test_get_device_name(char *data)`

获取本机设备名

- `void bt_test_get_device_addr(char *data)`

获取本机设备地址

- `void bt_test_set_device_name(char *data)`

设置本机设备名

- `void bt_test_pair_by_addr(char *data)`

和指定地址的设备配对，data: " 94:87:E0:B6:6D:AE "

- `void bt_test_unpair_by_addr(char *data)`

取消和指定地址的设备配对，data: " 94:87:E0:B6:6D:AE "

- `void bt_test_get_paired_devices(char *data)`

获取当前已配对的设备列表

- `void bt_test_free_paired_devices(char *data)`

释放`bt_test_get_paired_devices` 中申请的，用于存放已配对设备信息的内存

- `void bt_test_get_scaned_devices(char *data)`

获取扫描设备列表

- `void bt_test_start_discovery(char *data)`

扫描周围设备，包括BR/EDR 和 LE 设备

- `void bt_test_start_discovery_bredr(char *data)`

扫描周围的BR/EDR 设备

- `void bt_test_start_discovery_le(char *data)`

扫描周围的LE 设备

- `void bt_test_cancel_discovery(char *data)`

取消扫描操作

- `void bt_test_is_discovering(char *data)`

是否正在扫描周围的设备

- `void bt_test_display_devices(char *data)`

打印扫描到的周围设备的信息

- `void bt_test_display_paired_devices(char *data)`

打印当前已配对的设备信息

10.2.1.2 BLE接口测试说明

1. 手机安装第三方ble测试apk，如nrfconnect。
2. 选择bt_test_ble_start函数。
3. 手机蓝牙扫描并连接“ROCKCHIP_AUDIO BLE”。
4. 连接成功后，设备端会回调bt_test.cpp中的ble_status_callback_test函数，打印“+++++ RK_BLE_STATE_CONNECT +++++”。
5. 执行如下函数，进行具体功能测试：

- void bt_test_ble_start(char *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(char *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(char *data)

测试BLE状态接口。

- void bt_test_ble_stop(char *data)

停止BLE。

- void bt_test_ble_disconnect(char *data)

断开连接

10.2.1.3 BLE CLIENT接口测试说明

1. 选择bt_test_ble_client_open函数，启动ble client
2. 选择bt_test_start_discovery 或 bt_test_start_discovery_le，开始扫描设备
3. 输入”60 input xx:xx:xx:xx:xx:xx“，调用bt_test_ble_client_connect连接指定地址的ble server设备
4. 连接成功后，会触发回调ble_client_test_state_callback，打印“+++++ RK_BLE_CLIENT_STATE_IDLE +++++”
5. 输入”61 input xx:xx:xx:xx:xx:xx“，调用bt_test_ble_client_disconnect断开和指定地址的ble server设备的连接，成功断开连接，打印“+++++ RK_BLE_CLIENT_STATE_DISCONNECT +++++”
6. 输入”63 input xx:xx:xx:xx:xx:xx“，调用bt_test_ble_client_get_service_info获取已连接设备的service uuid，characteristic uuid、permission、Properties，descriptor uuid等信息
7. 输入”64 input uuid“，比如“56 input 00009999-0000-1000-8000-00805F9B34FB”通过bt_test_ble_client_read 读取 9999 uuid的数据，读取成功会触发bt_test_ble_client_recv_data_callback 打印读取到的值
8. 输入”65 input uuid“，比如“57 input 00009999-0000-1000-8000-00805F9B34FB”通过bt_test_ble_client_write 写 9999 uuid
9. 选择59、68打开或关闭指定uuid 的notification

10.2.1.4 A2DP SINK接口测试说明

1. 选择bt_test_sink_open函数。
2. 手机蓝牙扫描并连接“ROCKCHIP_AUDIO”。
3. 连接成功后，设备端会回调bt_test.cpp中的bt_sink_callback函数，打印“+++++ BT SINK EVENT: connect sucess +++++”。

4. 打开手机的音乐播放器，准备播放歌曲。

5. 执行如下函数，进行具体功能测试:

- void bt_test_sink_open(char *data)
打开 A2DP Sink 模式。
- void bt_test_sink_visibility00(char *data)
设置 A2DP Sink 不可见、不可连接。
- void bt_test_sink_visibility01(char *data)
设置 A2DP Sink 不可见、可连接。
- void bt_test_sink_visibility10(char *data)
设置 A2DP Sink 可见、不可连接。
- void bt_test_sink_visibility11(char *data)
设置 A2DP Sink 可见、可连接。
- void bt_test_sink_music_play(char *data)
反向控制设备播放。
- void bt_test_sink_music_pause(char *data)
反向控制设备暂停。
- void bt_test_sink_music_next(char *data)
反向控制设备播放下一曲。
- void bt_test_sink_music_previous(char *data)
反向控制设备播放上一曲。
- void bt_test_sink_music_stop(char *data)
反向控制设备停止播放。
- void bt_test_sink_reconnect_enable(char *data)
使能 A2DP Sink 自动连接功能。
- void bt_test_sink_reconnect_disable(char *data)
禁用 A2DP Sink 自动连接功能。
- void bt_test_sink_disconnect(char *data)
A2DP Sink 断开链接。
- void bt_test_sink_close(char *data)
关闭 A2DP Sink 服务。
- void bt_test_sink_status(char *data)
查询 A2DP Sink 连接状态。
- void bt_test_sink_set_volume(char *data)
设置音量测试
- void bt_test_sink_connect_by_addr(char *data)
连接指定地址的设备，data: " 94:87:E0:B6:6D:AE "

- void bt_test_sink_disconnect_by_addr(char *data)
断开和指定地址的设备的连接，data: " 94:87:E0:B6:6D:AE "
- void bt_test_sink_get_play_status(char *data)
获取播放状态，会触发play position change 回调
- void bt_test_sink_get_poschange(char *data)
当前连接的设备，是否支持播放进度上报

10.2.1.5 A2DP SOURCE接口测试说明

1. 选择bt_test_source_open函数，启动source功能。
2. 选择bt_test_start_discovery 或 bt_test_start_discovery_bredr，开始扫描周围蓝牙设备
3. 选择bt_test_source_connect_by_addr连接addr 指定蓝牙设备（27 input xx:xx:xx:xx:xx:xx），连接成功后，设备端会回调bt_test.cpp中的bt_test_source_status_callback函数，打印“+++++++ BT SOURCE EVENT:connect sucess ++++++”。
4. 此时设备播放音乐，则音乐会从连接的A2dp Sink设备中播出。
5. 执行如下函数，进行具体功能测试:

- void bt_test_source_open(char *data)
打开source功能
- void bt_test_source_close(char *data)
关闭source功能
- void bt_test_source_connect_status(char *data)
获取 A2DP Source 连接状态。
- void bt_test_source_connect_by_addr(char *data)
连接addr指定设备
- bt_test_source_disconnect
断开当前连接
- bt_test_source_disconnect_by_addr
断开addr 指定设备连接

10.2.1.6 SPP接口测试说明

1. 手机安装第三方SPP测试apk，如“Serial Bluetooth Terminal”。
2. 选择bt_test_spp_open函数。
3. 手机蓝牙扫描并连接“ROCKCHIP_AUDIO”。
4. 打开第三方SPP测试apk，使用spp连接设备。设备连接成功后，设备端会回调bt_test.cpp中的_bt_spp_status_callback函数，打印“+++++ RK_BT_SPP_EVENT_CONNECT +++++”。
5. 执行如下函数，进行具体功能测试:

- void bt_test_spp_open(char *data)
打开SPP。

- void bt_test_spp_write(char *data)

测试SPP写功能。向对端发送“This is a message from rockchip board!”字符串。

- void bt_test_spp_close(char *data)

关闭SPP。

- void bt_test_spp_status(char *data)

查询SPP连接状态。

10.2.1.7 HFP接口测试说明

1. 选择bt_test_hfp_sink_open或bt_test_hfp_hp_open函数。
2. 手机蓝牙扫描并连接“ROCKCHIP_AUDIO”。注：如果之前测试SINK功能时已经连接过手机，此时应在手机端先忽略该设备，重新扫描并连接。
3. 设备连接成功后，设备端会回调bt_test.cpp中的bt_test_hfp_hp_cb函数，打印“+++++ BT HFP HP CONNECT +++++”。如果手机被呼叫，此时打印“+++++ BT HFP HP RING +++++”，接通电话时会打印“+++++ BT HFP AUDIO OPEN +++++”。其他状态打印请直接阅读bt_test.cpp中bt_test_hfp_hp_cb函数源码。注：若调用了bt_test_hfp_sink_open接口，当设备连接成功后，A2DP SINK的连接状态也会打印，比如“+++++ BT SINK EVENT: connect success +++++”。
4. 执行如下函数，进行具体功能测试：

- bt_test_hfp_sink_open

并存方式打开HFP HF与A2DP SINK。

- bt_test_hfp_hp_open

仅打开HFP HF功能。

- bt_test_hfp_hp_accept

主动接听电话。

- bt_test_hfp_hp_hungup

主动挂断电话。

- bt_test_hfp_hp_redial

重播。

- void bt_test_hfp_hp_dial_number(char *data)

拨打指定电话号码

- bt_test_hfp_hp_report_battery

从0到9，每隔一秒上报一次电池电量状态，此时手机端可看到电量从空到满的图标变化过程。注：有些手机不支持蓝牙电量图标显示。

- bt_test_hfp_hp_set_volume

从1到15，每隔一秒设置一次蓝牙通话的音量，此时手机端可看到蓝牙通话音量进度条变化过程。注：有些手机并不动态显示进度条变化，主动加減音量触发进度条显示，此时可看到设备成功设置了手机端的音量。比如本身音量为0，该接口运行结束后，主动按手机音量‘+’按钮，发现音量已经满格。

- bt_test_hfp_hp_close

关闭HFP 服务。

- `bt_test_hfp_open_audio_diplex`

打开hfp音频通路，在回调事件RK_BT_HFP_AUDIO_OPEN_EVT中调用。

- `bt_test_hfp_close_audio_diplex`

关闭hfp音频通路，在回调事件RK_BT_HFP_AUDIO_CLOSE_EVT中调用。

10.2.1.8 OBEX接口测试说明

执行如下函数，进行具体功能测试:

- `bt_test_obex_init`

打开obexd进程，执行该函数即可进行文件传输测试

- `bt_test_obex_deinit`

关闭obexd进程

- `bt_test_obex_pbap_init`

蓝牙电话簿测试，需要先执行`bt_test_obex_init`

- `bt_test_obex_pbap_deinit`

反初始化蓝牙电话簿，之后需要执行`bt_test_obex_deinit`

- `bt_test_obex_pbap_connect`

打开pbap服务，并连接指定设备

- `bt_test_obex_pbap_get_pb_vcf`

获取联系人电话簿，结果存储在/data/pb.vcf

- `bt_test_obex_pbap_get_ich_vcf`

获取来电历史记录，结果存储在/data/ich.vcf

- `bt_test_obex_pbap_get_och_vcf`

获取拨出历史记录，结果存储在/data/och.vcf

- `bt_test_obex_pbap_get_mch_vcf`

获取未接来电历史记录，结果存储在/data/mch.vcf

- `bt_test_obex_pbap_disconnect`

关闭pbap服务，并断开连接

- `bt_test_obex_close`

关闭obex服务

10.2.1.9 PAN接口测试说明

执行如下函数，进行具体功能测试:

- `bt_test_pan_init`

打开pan服务，开启蓝牙共享网络

- `bt_test_pan_deinit`

关闭pan服务，关闭蓝牙共享网络

- `bt_test_pan_connect`

连接扫描到的手机蓝牙热点，连接成功后，可以使用ping www.baidu.com 测试网络

- `bt_test_pan_disconnect`

断开连接

10.2.2 测试步骤

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

```
# deviceio_test bluetooth
version:V1.3.6
#### Please Input Your Test Command Index ####
01.  bt_server_open
02.  bt_test_set_class
03.  bt_test_get_device_name
04.  bt_test_get_device_addr
05.  bt_test_set_device_name
06.  bt_test_enable_reconnect
07.  bt_test_disable_reconnect
08.  bt_test_start_discovery
09.  bt_test_start_discovery_le
10.  bt_test_start_discovery_bredr
11.  bt_test_start_discovery_pan
12.  bt_test_cancel_discovery
13.  bt_test_is_discovering
14.  bt_test_display_devices
15.  bt_test_read_remote_device_name
16.  bt_test_get_scanned_devices
17.  bt_test_display_paired_devices
18.  bt_test_get_paired_devices
19.  bt_test_free_paired_devices
20.  bt_test_pair_by_addr
21.  bt_test_unpair_by_addr
22.  bt_test_get_connected_properties
23.  bt_test_source_auto_start
24.  bt_test_source_connect_status
25.  bt_test_source_auto_stop
26.  bt_test_source_open
27.  bt_test_source_close
28.  bt_test_source_connect_by_addr
29.  bt_test_source_disconnect
30.  bt_test_source_disconnect_by_addr
31.  bt_test_source_remove_by_addr
32.  bt_test_sink_open
33.  bt_test_sink_visibility00
34.  bt_test_sink_visibility01
35.  bt_test_sink_visibility10
36.  bt_test_sink_visibility11
```


37. bt_test_ble_visibility00
38. bt_test_ble_visibility11
39. bt_test_sink_status
40. bt_test_sink_music_play
41. bt_test_sink_music_pause
42. bt_test_sink_music_next
43. bt_test_sink_music_previous
44. bt_test_sink_music_stop
45. bt_test_sink_set_volume
46. bt_test_sink_connect_by_addr
47. bt_test_sink_disconnect_by_addr
48. bt_test_sink_get_play_status
49. bt_test_sink_get_poschange
50. bt_test_sink_disconnect
51. bt_test_sink_close
52. bt_test_ble_start
53. bt_test_ble_set_address
54. bt_test_ble_set_adv_interval
55. bt_test_ble_write
56. bt_test_ble_disconnect
57. bt_test_ble_stop
58. bt_test_ble_get_status
59. bt_test_ble_client_open
60. bt_test_ble_client_close
61. bt_test_ble_client_connect
62. bt_test_ble_client_disconnect
63. bt_test_ble_client_get_status
64. bt_test_ble_client_get_service_info
65. bt_test_ble_client_read
66. bt_test_ble_client_write
67. bt_test_ble_client_is_notify
68. bt_test_ble_client_notify_on
69. bt_test_ble_client_notify_off
70. bt_test_ble_client_get_eir_data
71. bt_test_spp_open
72. bt_test_spp_write
73. bt_test_spp_close
74. bt_test_spp_status
75. bt_test_hfp_sink_open
76. bt_test_hfp_hp_open
77. bt_test_hfp_hp_accept
78. bt_test_hfp_hp_hungup
79. bt_test_hfp_hp_redail
80. bt_test_hfp_hp_dial_number
81. bt_test_hfp_hp_report_battery
82. bt_test_hfp_hp_set_volume
83. bt_test_hfp_hp_close
84. bt_test_hfp_hp_disconnect
85. bt_test_obex_init
86. bt_test_obex_pbap_init
87. bt_test_obex_pbap_connect
88. bt_test_obex_pbap_get_pb_vcf
89. bt_test_obex_pbap_get_ich_vcf

```
90.  bt_test_obex_pbap_get_och_vcf
91.  bt_test_obex_pbap_get_mch_vcf
92.  bt_test_obex_pbap_get_spd_vcf
93.  bt_test_obex_pbap_get_fav_vcf
94.  bt_test_obex_pbap_disconnect
95.  bt_test_obex_pbap_deinit
96.  bt_test_obex_deinit
97.  bt_test_pan_init
98.  bt_test_pan_deinit
99.  bt_test_pan_connect
100. bt_test_pan_disconnect
101. bt_server_close
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:26
#注：打开source功能
Which would you like:8 input 15000
#注：开始扫描周围的蓝牙设备，扫描时间15s
Which would you like:28 input xx:xx:xx:xx:xx:xx
#注： 开始和地址为xx:xx:xx:xx:xx:xx 的设备进行连接
```

3. 需要传输地址或其他参数的测试程序，输入： 编号（空格）input（空格）参数，比如要和指定地址的设备进行配对

```
Which would you like:20 input 94:87:E0:B6:6D:AE
#注：开始和地址为 94:87:E0:B6:6D:AE 的设备进行配对
```

10.3 BLE配网演示程序

请参见《Rockchip_Developer_Guide_Network_Config_CN》文档。