

# RockChip Devicelo Bluetooth Interface Documentation

---

发布版本：1.4

作者：francis.fan

日期：2019.5.27

文件密级：公开资料

---

## 概述

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

libDevicelo\_bluez.so：基于BlueZ协议栈，主要适用于Realtek的蓝牙模组，如：RTL8723DS.

libDevicelo\_broadcom.so：基于BSA协议栈，主要适用于正基的蓝牙模组，如：AP6255.

libDevicelo\_cypress.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libDevicelo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Devicelo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Devicelo库，对应libDevicelo\_bluez.so。

BSA DEVICEIO：基于BSA协议栈实现的Devicelo库，对应libDevicelo\_broadcom.so和libDevicelo\_cypress.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	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
2019-5-27	V1.3	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	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

## RockChip Device Bluetooth Interface Documentation

- 1、蓝牙基础接口 ( RkBtBase.h )
- 2、BLE接口介绍 ( RkBle.h )
- 2、SPP接口介绍 ( RkBtSpp.h )
- 3、A2DP SINK接口介绍 ( RkBtSink.h )
- 4、A2DP SOURCE接口介绍 ( RkBtSource.h )
- 5、HFP-HF接口介绍 ( RkBtHfp.h )
- 6、示例程序说明
  - 6.1 编译说明
  - 6.2 基础接口演示程序
    - 6.2.1 接口说明
    - 6.2.2 测试步骤
  - 6.3 BLE配网演示程序

## 1、蓝牙基础接口 ( RkBtBase.h )

- `RkBtContent` 结构

```
typedef struct {
    Btle_Uuid_Type_t server_uuid; //BLE server uuid
    Btle_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_rcv_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。

- `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。

## 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 DEVICEIO两种方式均可用，而对BSA DEVICEIO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。

- `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。

### 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。

- `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。

- `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_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_set_visibility(const int visible, const int connectal)`

设置A2DP Sink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。

- `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。

## 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
    /* 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-
} 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。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_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 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、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_EVENT;
```

- `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_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。

## 6、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。

### 6.1 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。

### 6.2 基础接口演示程序

#### 6.2.1 接口说明

##### 6.2.1.1 蓝牙服务的基础接口测试说明

- `void bt_test_bluetooth_init(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。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\_rcv\_callback函数说明。*

- `bt_test_set_class(void *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(void *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

### 6.2.1.2 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(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。

### 6.2.1.3 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(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\_disable(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。

#### **6.2.1.4 A2DP SOURCE接口测试说明**

- 1、选择bt\_test\_source\_auto\_start函数。
- 2、设备会自动扫描身边的A2dp Sink类型设备，并连接信号最强的那个。
- 3、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\_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。

### 6.2.1.5 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(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 !”字符串。
- void bt\_test\_spp\_close(void \*data)  
关闭SPP。
- void bt\_test\_spp\_status(void \*data)  
查询SPP连接状态。

### 6.2.1.6 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 sucess ++++++”。
  - 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\_redail  
重播。
  - 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，主动加减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。

## 6.2.2 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:V1.2.3
#### Please Input Your Test Command Index ####
01.  bt_server_open
02.  bt_test_set_class
03.  bt_test_enable_reconnect
04.  bt_test_disable_reconnect
05.  bt_test_source_auto_start
06.  bt_test_source_connect_status
07.  bt_test_source_auto_stop
08.  bt_test_sink_open
09.  bt_test_sink_visibility00
10.  bt_test_sink_visibility01
11.  bt_test_sink_visibility10
12.  bt_test_sink_visibility11
13.  bt_test_sink_status
14.  bt_test_sink_music_play
15.  bt_test_sink_music_pause
16.  bt_test_sink_music_next
17.  bt_test_sink_music_previous
18.  bt_test_sink_music_stop
19.  bt_test_sink_disconnect
20.  bt_test_sink_set_volume
21.  bt_test_sink_close
22.  bt_test_ble_start
23.  bt_test_ble_write
24.  bt_test_ble_stop
25.  bt_test_ble_get_status
26.  bt_test_spp_open
27.  bt_test_spp_write
28.  bt_test_spp_close
29.  bt_test_spp_status
30.  bt_test_hfp_sink_open
31.  bt_test_hfp_hp_open
32.  bt_test_hfp_hp_accept
33.  bt_test_hfp_hp_hungup
34.  bt_test_hfp_hp_redial
35.  bt_test_hfp_hp_report_battery
```

```
36.  bt_test_hfp_hp_set_volume
37.  bt_test_hfp_hp_close
38.  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:05
#注：选择05前，要开启一个BT Sink设备，该设备处于可发现并可连接状态。05功能会自动扫描BT Sink设备并连接信号最强的那个设备。
```

## 6.3 BLE配网演示程序

请参见《Rockchip\_Developer\_Guide\_Network\_Config\_CN》文档。