Rockchip Blutooth DeviceIo Introduction

ID: RK-SM-YF-343

Release Version: V2.0.1

Release Date: 2020-07-12

Security Level: □Top-Secret □Secret □Internal ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document mainly introduce the Bluetooth interface in the Rockchip DeviceIo library. Different Bluetooth chip modules correspond to different DeviceIo libraries, and the Correspondence are as follows:

libDeviceIo_bluez.so: based on BlueZ protocol stack, it is mainly suitable for Realtek's Bluetooth modules, such as: RTL8723DS.

libDeviceIo_broadcom.so: based on BSA protocol stack, it is mainly suitable for AMPAK's Bluetooth modules such as AP6255.

libDeviceIo_cypress.so: based on BSA protocol stack, it is mainly suitable for AzureWave's Bluetooth modules, such as: AW-CM256.

After users configure the Bluetooth chip model of the SDK, deviceio compilation script will automatically select the libDeviceIo library according to the selected chip model. Please refer to the "WIFI/BT configuration" chapter in "Rockchip_Developer_Guide_Network_Config_CN" for the Bluetooth chip configuration of SDK. The interfaces of the DeviceIo library based on different protocol stacks have been integrated as much as possible, but there are still some differences in some interfaces. These differences will be described in details when a specific interface is introduced.

Terms Interpret

BLUEZ DEVICEIO: deviceIo library based on BlueZ protocol stack, corresponding to libDeviceIo_bluez.so.

BSA DEVICEIO: deviceIo library based on BSA protocol stack, corresponding to libDeviceIo_broadcom.so and libDeviceIo_cypress.so

BLUEZ only: the interface or document only supports BLUEZ DEVICEIO.

BSA only: The interface or document only supports BSA DEVICEIO.

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Date	Document Version	Library Version	Author	Revision History
2019-3-27	V1.0.0	V1.0.x / V1.1.x	Francis Fan	Initial version (BLUEZ only)
2019-4-16	V1.1.0	V1.2.0	Francis Fan	Add BLE network configuration Demo Update BtSource interface Add BSA library support Update the format of the document
2019-4-29	V1.2.0	V1.2.1	Francis Fan	Fixed the issue that BSA branch deviceio_test failed Fixed the BUG that BLUEZ fail to initialize and causing program stuck Update the method for A2DP SOURCE to get playrole
2019-5-27	V1.3.0	V1.2.2	Francis Fan	Add A2DP SOURCE reverse control event notice Add HFP HF interface support Add Bluetooth class setting interface Add Bluetooth automatic reconnection attribute setting interface Add A2DP SINK volume reverse control（BSA only）
2019-6-4	V1.4.0	V1.2.3	Francis Fan	Bluez: realize A2DP SINK volume forward and reverse control Bluez: cancel SPP and A2DP SINK relationship Bluez: rk_bt_enable_reconnec save attributes to the file, the attribute setting still takes effect after the device restarts Bluez: fix A2DP SOURCE reverse control function initialization probability failure issue Bluez: fix rk_bt_sink_set_visibilit BSA: fix A2DP SOURCE automatic reconnection failure BSA: fix rk_bt_hfp_hangup api Remove the rk_bt_sink_set_auto_reconnect interface

Date	Document Version	Library Version	Author	Revision History
2019-6-24	V1.5.0	V1.2.4	CTF	Add HFP HF alsa control demo Add hfp disconnect api: rk_bt_hfp_disconnect Fixed the bug that it cannot receive PICKUP, HANGUP events when answer and refuse calls on mobile phone Bsa: add HFP HF to enable CVSD (8K sampling) interface Bsa: fix cypress bsa corresponding pop up prompt problem Bsa: update broadcom bsa version (rockchip_20190617) Bsa: fix the bug that unable to recognize some Bluetooth speaker device types when Bluetooth scanning Bsa: fix battery power report BUG

Date	Document Version	Library Version	Author	Revision History
2019-10-30	V1.6.0	V1.3.0	CTF	Bluez: Bluetooth anti-initialization is implemented. Bluez: fix to obtain the name and Bluetooth Mac address interface of the local device Bluez: add pbap profile support Bluez: support hfp 8K and 16K sampling rate adaptation Bluez: add sink to play underrun report Bsa: add setting sink to play device node interface Bsa: add ble visibility setting interface Bsa: add ble disconnection interface actively Bsa: support setting Bluetooth address during Bluetooth initialization add Bluetooth start status report add Bluetooth pairing status report add start Bluetooth scanning, stop Bluetooth scanning interface Add an interface to get whether Bluetooth is in scanning status Add an interface to print the list of currently scanned devices Add an interface to actively pair with a specified device, cancel pairing with a specified device Add getting the current paired device list, and release the acquired paired device list interface Add printing the current paired device list interface Add setting the local device name interface Add songs information report Add songs playback progress report Add avdtp (a2dp sink) status report sink add actively connecting and disconnecting with a specified device interface Add getting the current playback status interface Add getting the currently connected remote device Whether to support reporting the playback progress interface actively Support to print the log to syslog
2019-11-16	V1.7.0	V1.3.1	CTF	The source callback adds the address and name parameters of the connected device
2019-12-12	V1.8.0	V1.3.2	CTF	bluez: implement ble client function bluez: implement obex file transfer function

Date	Document Version	Library Version	Author	Revision History
2020-03-17	V1.9.0	V1.3.4	CTF	bluez: add type filter for scanning interface (LE or BR/EDR or both) bluez: add interface for getting scanning device list bluez: add automatically connect back to the last connected sink device at first scanning after starting bt source bluez: fix the BUG that connection device failure during scanning bluez: optimize init and deinit execution time bluez: fix the BUG that thread synchronization in qt non-main mianloop thread start Bluetooth bluez: add source disconnect failure, automatic return event report bluez: add source disconnect current connection interface bluez: add getting the connection status of the specified device bluez: fix the problem of ble initial memory cross-border bsa: add setting bsa_server.sh path interface ble status callback with remote device address and name
2020-07-08	V2.0.0	V1.3.5	CTF	Fix some bluez and bsa bugs. Please see Rk_system.h V1.3.5 for details. Add setting ble broadcast interval interface. Add hfp calling the specified phone number interface. rk_ble_client_write adding write data length parameters support ble MTU reporting ble client add getting ble device broadcast api bluez: add obex status callback bluez: add setting ble address interface bluez: ble feature value adding write-without-response attribute bsa: add rk_bt_source_disconnect interface bsa: support LE BR/EDR filter scan bsa: add source reconnect the last connected sink device automatically at first scanning bsa: support ble client function bsa: add interface to read remote connection device name bsa: add interface to get list of current scanning devices
2020-07-12	V2.0.1	V1.3.5	Ruby Zhang	Update the format of the document

Contents

1. Bluetooth Basic Interface (RkBtBase.h)

- `RkBtContent` structure

```
typedef struct {
    Ble_Uuid_Type_t server_uuid; //BLE server uuid
    Ble_Uuid_Type_t chr_uuid[12]; //BLE CHR uuid, 12 at most
    uint8_t chr_cnt; //the number of CHR
    const char *ble_name; //the name of BLE, which may be different from the name of bt_name
    uint8_t ble_addr[DEVICE_ADDR_LEN];  //BLE address, random address is used by default(BLUEZ Only)
    uint8_t advData[256]; //Broadcast data
    uint8_t advDataLen; //the length of broadcast data
    uint8_t respData[256]; //Broadcast response data
    uint8_t respDataLen; //the length of broadcast response data
    /* Ways to generate broadcast data with the value of BLE_ADVDATA_TYPE_USER/BLE_ADVDATA_TYPE_SYSTEM
     * BLE_ADVDATA_TYPE_USER: use data from advData and respData as BLE broadcast
     * BLE_ADVDATA_TYPE_SYSTEM: system's broadcast data by default.
     *     Broadcast packages: flag(0x1a), 128bit Server UUID;
     *     Broadcast response packages: bluetooth's name
     */
    uint8_t advDataType;
    //AdvDataKgContent adv_kg;
    char le_random_addr[6]; //random address, generated by system by default, users do not need to fill in.
    /* BLE data receiving callback function, uuid represents the current CHR UUID, data: data pointer, len: data's length */
    void (*cb_ble_recv_fun)(const char *uuid, unsigned char *data, int len);
    /* BLE data request callback function. When this function is used on the receiver side, it will trigger the function to fill data */
    void (*cb_ble_request_data)(const char *uuid);
} RkBleContent;
```

- `RkBtContent` structure

```
typedef struct {
    RkBleContent ble_content; //BLE parameter configuration
    const char *bt_name; //Bluetooth's name
    const char *bt_addr;      //Bluetooth address (Bsa only, use the bt mac address fixed inside the chip by default)
} RkBtContent;
```

- `RkBtScanedDevice` structure

```
typedef struct scaned_dev {
    char *remote_address;       //remote device address
    char *remote_name;          //remote device name
    unsigned int cod;           //class of device
    bool is_connected;          //whether the remote device is connected currently(sink, source, hfp)
    truct paired_dev *next; //point to next device
} RkBtScanedDevice;
```

- `RK_BT_STATE` introduction

```
typedef enum {
    RK_BT_STATE_OFF,         //closed
    RK_BT_STATE_ON,          //turned off
    RK_BT_STATE_TURNING_ON,  //is turnning on
    RK_BT_STATE_TURNING_OFF, //is trunning off
} RK_BT_STATE;
```

- `RK_BT_BOND_STATE` introduction

```
typedef enum {
    RK_BT_BOND_STATE_NONE,    //pairing failed or unpaired
    RK_BT_BOND_STATE_BONDING, //is pairing
    RK_BT_BOND_STATE_BONDED,  //paired successfully
} RK_BT_BOND_STATE;
```

- `RK_BT_SCAN_TYPE` introduction

```
typedef enum {
    SCAN_TYPE_AUTO,         //LE and BR/EDR, scan all types of devices
    SCAN_TYPE_BREDR,        //scan BR/EDR type devices only
    SCAN_TYPE_LE            //scan LE type devices only
} RK_BT_SCAN_TYPE;
```

- `RK_BT_DISCOVERY_STATE` introduction

```
typedef enum {
    RK_BT_DISC_STARTED,          //start scanning successfully
    RK_BT_DISC_STOPPED_AUTO,     //scan completed, automatically stop scanning
    RK_BT_DISC_START_FAILED,     //start scanning failed
    RK_BT_DISC_STOPPED_BY_USER, //interrupt scanning by rk_bt_cancel_discovery,
} RK_BT_DISCOVERY_STATE;
```

- `RK_BT_PLAYROLE_TYPE` introduction

```
typedef enum {
    PLAYROLE_TYPE_UNKNOWN,  //unknown device
    PLAYROLE_TYPE_SOURCE,   //a2dp Source device
    PLAYROLE_TYPE_SINK,     //a2dp Sink device
} RK_BT_PLAYROLE_TYPE;
```

- `typedef void (*RK_BT_STATE_CALLBACK)(RK_BT_STATE state)``typedef void (*RK_BT_STATE_CALLBACK)(RK_BT_STATE state)`

  Bluetooth status callback

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

  Bluetooth pairing status callback, bd_addr: address of current bound device, name: name of current paired device

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

  Bluetooth scanning status callback, if rk_bt_start_discovery is used to scan the surrounding Bluetooth devices, you need to register this callback

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

  Bluetooth device scan callback. If you use rk_bt_start_discovery to scan the surrounding Bluetooth devices, you need to register this callback. Bluez triggers this callback every time it scans a device; after bsa scan, it will trigger the callback in turn according to the number of devices scanned.

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

  Remote device name update callback

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

  ble MTU callback, shared with ble and ble client, after successful MTU negotiation, the callback is triggered

- `void rk_bt_register_state_callback(RK_BT_STATE_CALLBACK cb)`

  Register the callback function to get the Bluetooth start status

- `void rk_bt_register_bond_callback(RK_BT_BOND_CALLBACK cb)`

  Register callback function to get Bluetooth pairing status

- `void rk_bt_register_discovery_callback(RK_BT_DISCOVERY_CALLBACK cb)`

  Register the callback function to get the Bluetooth scanning status

- `void rk_bt_register_dev_found_callback(RK_BT_DEV_FOUND_CALLBACK cb)`

  Register the callback function of the discovered device

- `void rk_bt_register_name_change_callback(RK_BT_NAME_CHANGE_CALLBACK cb)`

  Registered device name update callback function

- `int rk_bt_init(RkBtContent *p_bt_content)`

  To initialize Bluetooth service, this interface should be called to initialize Bluetooth basic services before calling other Bluetooth interfaces.

- `int rk_bt_deinit(void)`

  To de-initialize Bluetooth service.

- `int rk_bt_is_connected(void)`

  To get whether there is a service connected to Bluetooth currently. Any one of SPP/BLE/SINK/SOURCE services is connected, it will return 1; otherwise return 0.

- `int rk_bt_set_class(int value)`

  Set the type of Bluetooth device. value: the type's value. For example, 0x240404 means:

  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)`

  To enables/disables the auto reconnect function of HFP/A2DP SINK. value: 0 means disable the auto reconnect function, 1 means enable the auto reconnect function.

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

  Start Bluetooth scanning, mseconds: scan duration, in milliseconds; scan_type: scan type, see the description of RK_BT_SCAN_TYPE for details, only bluez supports scan type filtering, bsa only supports full type scan.

- `void rk_bt_cancel_discovery()`

  Stop Bluetooth scanning and cancel the scanning operation initiated by rk_bt_start_discovery

- `bool rk_bt_is_discovering()`

  Whether Bluetooth is in the state of scanning the surrounding devices, returning true if the device is being scanned, otherwise false

- `void rk_bt_display_devices()`

  Print a list of currently scanned devices

- `int rk_bt_pair_by_addr(char *addr)`

  Pair with the device specified by addr actively; addr: device address, such as: 94:87:E0:B6:6D:AE

- `int rk_bt_unpair_by_addr(char *addr)`

  Cancel pairing with the device specified by addr. After canceling the pairing, all records of the device will be deleted; addr: device address

- `int rk_bt_set_device_name(char *name)`

  Set the local device name, name: the device name you want to set

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

  Get the local device name, name: used to store the obtained device name, len: the length of the device name

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

  Get the local device's Bluetooth mac address, addr: used to store the obtained mac address, len: the length of mac address

- `void rk_bt_display_paired_devices()`

  Print the currently paired device list

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

  Get the currently paired device list, dev_list: used to store the paired device list, count: the number of paired devices

- `int rk_bt_free_paired_devices(RkBtScanedDevice*dev_list)`

  Free the memory allocated by rk_bt_get_paired_devices to store the device list

- `int rk_bt_get_scaned_devices(RkBtScanedDevice**dev_list,int *count)`

  Get the currently scanned device list, dev_list: used to store the scanned device list, count: the number of scanned devices

- `int rk_bt_free_scaned_devices(RkBtScanedDevice*dev_list)`

  Free the memory allocated by rk_bt_get_scaned_devices to store the device list

- `void rk_bt_set_bsa_server_path(char *path)`

Set the bsa_server.sh path, is /usr/bin/bsa_server.sh (BSA only) by default

- `bool rk_bt_get_connected_properties(char *addr)`

  Get the connection status of the device specified by addr, addr: device address, return true if connected, otherwise true false (BLUEZ only)

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

  Set visible/connectable properties. visiable: 0 means invisible, 1 means visible. connectal: 0 means not connectable, 1 means connectable. Only applicable to BR/EDR devices

- `RK_BT_PLAYROLE_TYPE rk_bt_get_playrole_by_addr(char *addr)`

  Get the playrole of the device specified by addr, see the description of RK_BT_PLAYROLE_TYPE for details.

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

  Read the name of the device specified by addr, transport specifies the device type, unknown device: RK_BT_TRANSPORT_UNKNOWN, BR/EDR device: RK_BT_TRANSPORT_BR_EDR, LE device: RK_BT_TRANSPORT_LE. This interface needs to be used matched with rk_bt_register_name_change_callback. Reading successfully will trigger RK_BT_NAME_CHANGE_CALLBACK callback (BSA only)

2. BLE Interface Introduction (RkBle.h)

- `RK_BLE_STATE` introduction

```
typedef enum {
    RK_BLE_STATE_IDLE = 0, //idle state
    RK_BLE_STATE_CONNECT, //successful connection
    RK_BLE_STATE_DISCONNECT //disconnected
} RK_BLE_STATE;
```

- `typedef void (*RK_BLE_STATE_CALLBACK)(const char *bd_addr, const char *name, RK_BLE_STATE state)`

  BLE state callback function. bd_addr: remote device address, name: remote device name.

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

  BLE receiving callback function. uuid: CHR UUID, data: data pointer, len: data's length

- `int rk_ble_register_status_callback(RK_BLE_STATE_CALLBACK cb)`

  This interface is used to register a callback function to get BLE connection status.

- `int rk_ble_register_recv_callback(RK_BLE_RECV_CALLBACK cb)`

  This interface is used to register a callback function to receive BLE data. There are two ways to register the receiving callback function: one is specified by the RkBtContent parameter of the rk_bt_init () interface; the other is to call this interface for registration. For BLUEZ DEVICEIO, both of the two methods are available, but for BSA DEVICEIO, you can only use this interface to register the receiving callback function.

- `void rk_ble_register_mtu_callback(RK_BT_MTU_CALLBACK cb)`

  This interface is used to register mtu callback. After mtu negotiation is successful, RK_BT_MTU_CALLBACK callback is triggered to report the negotiated mtu value

- `int rk_ble_start(RkBleContent *ble_content)`

  To enable BLE broadcast. ble_content: should be consistent with the p_bt_content->ble_content in the rk_bt_init(RkBtContent *p_bt_content).

- `int rk_ble_stop(void)`

  Stop BLE broadcast. After this function is executed, BLE becomes invisible and disconnected.

- `int rk_ble_get_state(RK_BLE_STATE *p_state)`

  Get the current connection status of BLE actively.

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

  Send data to the other side.

  uuid: the CHR object of the written data

  data: the pointer of the written data

  len: the length of the written data. You should pay attention to that: the length is limited by the MTU connected to BLE, and it will be cut off when over the MTU.

  The current MTU's value is set to 134 Bytes by default to maintain a good compatibility

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

  Set ble visible/connectable characteristics. visible: 0 means invisible, 1 means visible. connect: 0 means not connectable, 1 means connectable. This interface is only applicable to bsa (BSA only)

- `int rk_ble_disconnect(void)`

  Disconnect the current ble connection actively

- `int rk_ble_set_address(char *address)`

  Set the ble address, you can also use ble_addr parameter setting in rk_bt_init, the default random address is not set (BLUEZ Only)

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

  Set the ble broadcast interval, adv_int_min minimum broadcast interval, adv_int_max maximum broadcast interval, minimum value is 32 (32 * 0.625ms = 20ms), is 30ms when not set bsa default interval, bluez is 100ms by default.

3. BLE CLIENT Interface Introduction (RkBtSpp.h)

- `RK_BT_SPP_STATE` introduction

```
typedef enum {
    RK_BT_SPP_STATE_IDLE = 0, //idle state
    RK_BT_SPP_STATE_CONNECT, //successful connection
    RK_BT_SPP_STATE_DISCONNECT //disconnected state
} RK_BT_SPP_STATE;
```

- `RK_BLE_CLIENT_SERVICE_INFO` introduction

```
typedef struct {
    int service_cnt;                                    //number of services
included in the connected remote device
```

```
    RK_BLE_CLIENT_SERVICE service[SERVICE_COUNT_MAX]; //detailed information
for each service
} RK_BLE_CLIENT_SERVICE_INFO;

typedef struct {
    char describe[DESCRIBE_BUG_LEN];         //uuid description
    char path[PATH_BUF_LEN];
    char uuid[UUID_BUF_LEN];                 //service uuid
    int chrc_cnt;                            //the number of characteristics
included in the service
    RK_BLE_CLIENT_CHRC chrc[CHRC_COUNT_MAX]; //detailed information for each
characteristic
} RK_BLE_CLIENT_SERVICE;

typedef struct {
    char describe[DESCRIBE_BUG_LEN];         //uuid description
    char path[PATH_BUF_LEN];
    char uuid[UUID_BUF_LEN];                 //characteristic uuid
    unsigned int props;                      //characteristic attributes
    unsigned int ext_props;                  //characteristic extended
attributes
    unsigned int perm;                       //characteristic permission
    bool notifying;                          //whether characteristic open
notification(BLUEZ only)
    int desc_cnt;                            //the number of descriptors
contained in this characteristic
    RK_BLE_CLIENT_DESC desc[DESC_COUNT_MAX]; //detailed information for each
descriptor
} RK_BLE_CLIENT_CHRC;

typedef struct {
    char describe[DESCRIBE_BUG_LEN];         //uuid description
    char path[PATH_BUF_LEN];
    char uuid[UUID_BUF_LEN];                 //descriptor uuid
} RK_BLE_CLIENT_DESC;
```

Note: the path indicates the relationship between service, characteristic, and descriptor. It is used to traversal search, application layer does not need to care the parameter, which is only used in bluez.

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

  ble client status callback function, bd_addr: remote device address, name: remote device name.

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

  ble client data reception callback function. uuid: CHR UUID, data: data pointer, len: data length.

- `void rk_ble_client_register_state_callback(RK_BLE_CLIENT_STATE_CALLBACK cb)`

  Register ble client status callback function

- `int rk_ble_client_register_recv_callback(RK_BLE_CLIENT_RECV_CALLBACK cb)`

  Register ble client data reception callback function

- `int rk_ble_client_open(void)`

  Initialize ble client

- `void rk_ble_client_close(void)`

Deinitialize ble client

- `RK_BLE_CLIENT_STATE rk_ble_client_get_state()`

  Get ble client status actively

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

  Get the specified information of the device by address, including service uuid, characteristic uuid, permission, Properties, descriptor uuid, etc. Please refer to the RK_BLE_CLIENT_SERVICE_INFO structure for details.

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

  Send data to the specify uuid of the opposite side, data: data pointer, len: data length.

- `int rk_ble_client_read(const char *uuid)`

  Read the specified uuid data from the opposite side, and it will trigger the RK_BLE_CLIENT_RECV_CALLBACK callback when read successfully.

- `int rk_ble_client_connect(char *address)`

  Connect to the device with the specified address

- `int rk_ble_client_disconnect(char *address)`

  Disconnect from the device with the specified address

- `bool rk_ble_client_is_notifying(const char *uuid)`

  Search whether the specified uuid has enabled notification, and returns true (BLUEZ only) when it is enable.

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

  Set the notification with specified uuid. The uuid must support notifications or indications. It is turned on when enable = true and turned off when enable = false. When the remote device (server) writes the uuid, it will trigger the RK_BLE_CLIENT_RECV_CALLBACK callback to report the modified value automatically.

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

  Get the broadcast data of the remote device specified by address, eir_data: the obtained broadcast data, len: the length of the broadcast data

- `int rk_ble_client_default_data_length()`

  Force to specify the length of hci writing data to be 27 bytes, which is customized for specific customers. Generally, this API is not used (BSA only)

4. SPP Interface Introduction (RkBtSpp.h)

- `RK_BT_SPP_STATE` introduction

```
typedef enum {
    RK_BT_SPP_STATE_IDLE = 0,    //idle state
    RK_BT_SPP_STATE_CONNECT,    //successful connection
    RK_BT_SPP_STATE_DISCONNECT //disconnected state
} RK_BT_SPP_STATE;
```

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

State callback function.

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

  Reception callback function. data: data pointer, len: data length.

- `int rk_bt_spp_register_status_cb(RK_BT_SPP_STATUS_CALLBACK cb)`

  Registration status callback function.

- `int rk_bt_spp_register_recv_cb(RK_BT_SPP_RECV_CALLBACK cb)`

  Registration reception callback function.

- `int rk_bt_spp_open(void)`

  Turn on SPP, the device is in the connectable state.

- `int rk_bt_spp_close(void)`

  Close SPP。

- `int rk_bt_spp_get_state(RK_BT_SPP_STATE *pState)`

  Get the current SPP connection status actively

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

  Send data. data: data pointer, len: data length.

5. A2DP SINK Interface Introduction (RkBtSink.h)

- `BtTrackInfo` structure

```
typedef struct btmg_track_info_t {
    char title[256];              //title
    char artist[256];             //artist
    char album[256];              //album
    char track_num[64];           //track the number of the song in the
album
    char num_tracks[64];          //total number of the album
    char genre[256];              //genres
    char playing_time[256];       //total time of playing
} btmg_track_info_t;

typedef struct btmg_track_info_t BtTrackInfo;
```

- `RK_BT_SINK_STATE` introduction

```
typedef enum {
    RK_BT_SINK_STATE_IDLE = 0,      //idle sate
    RK_BT_SINK_STATE_CONNECT,       //connected state
    RK_BT_SINK_STATE_DISCONNECT     //disconnected
    RK_BT_SINK_STATE_PLAY ,         //avrcp playing state
    RK_BT_SINK_STATE_PAUSE,         //avrcp pause state
    RK_BT_SINK_STATE_STOP,          //avrcp stop state
    RK_BT_A2DP_SINK_STARTED,        //avdtp playing state
    RK_BT_A2DP_SINK_SUSPENDED,      //avdtp pause state
    RK_BT_A2DP_SINK_STOPPED,        //avdtp stop state
} RK_BT_SINK_STATE;
```

The avdtp state is mainly used for reporting a2dp sink state during WeChat calls and WeChat voices, because the avrcp status change will not be triggered at this time.

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

  Status callback function.

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

  Volume change callback function. Which is called when the volume of the mobile phone changes. volume: the new volume value. *Note: Due to the different implementations of AVRCP version and different mobile phone manufacturers, some mobile phones are not compatible with this function, iPhone series phones support this interface well.*

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

  Song information callback function, which will be triggered when the playing song changes. bd_addr: remote device address, track_info: song information

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

  Song playback progress callback, when the remote device supports position change, it will automatically report the playback progress and trigger this function. bd_addr: remote device address, song_len: total song length, song_pos: current playback progress

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

  Playback underrun status callback, which will be triggered automatically when playing underrun, this interface is only applicable to bluez (Bluez only).

- `int rk_bt_sink_register_callback(RK_BT_SINK_CALLBACK cb)`

  Register a status callback function.

- `int rk_bt_sink_register_volume_callback(RK_BT_SINK_VOLUME_CALLBACK cb)`

  Register the volume change callback function.

- `int rk_bt_sink_register_track_callback(RK_BT_AVRCP_TRACK_CHANGE_CB cb)`

  Register the song information callback function

- `int rk_bt_sink_register_position_callback(RK_BT_AVRCP_PLAY_POSITION_CB cb)`

  Register the song playback progress callback

- `void rk_bt_sink_register_underun_callback(RK_BT_SINK_UNDERRUN_CB cb)`

  Register the underrun callback function, which is only applicable to bluez (Bluez only)

- `int rk_bt_sink_open()`

  To enable A2DP SINK service. If A2DP SINK is required to coexist with HFP, please refer to `rk_bt_hfp_sink_open` interface in the chapter of "HFP-HF Interface Introduction"

- `int rk_bt_sink_close(void)`

  Close A2DP Sink function.

- `int rk_bt_sink_get_state(RK_BT_SINK_STATE *p_state)`

  To get A2DP Sink connection status actively.

- `int rk_bt_sink_play(void)`

  Reverse control: play.

- `int rk_bt_sink_pause(void)`

  Reverse control: pause

- `int rk_bt_sink_prev(void)`

  Reverse control: previous

- `int rk_bt_sink_next(void)`

  Reverse control: next

- `int rk_bt_sink_stop(void)`

  Reverse control: stop playing

- i`nt rk_bt_sink_volume_up(void)`

  Reverse control: increase the volume. Volume range [0, 127], each time the interface is called, the volume increases by 8.

  Note: Due to the different implementations of AVRCP version and different mobile phone manufacturers, some mobile phones are not compatible with this function. iPhone series phones support this interface well.

- i`nt rk_bt_sink_volume_down(void)`

  Reverse control: reduce the volume . Volume range [0, 127], each time the interface is called, the volume reduce by 8.

  Note: Due to the different implementations of AVRCP version and different mobile phone manufacturers, some mobile phones are not compatible with this function. iPhone series phones support this interface well.

- `int rk_bt_sink_set_volume(int volume)`

  Reverse control: Set the volume of A2DP SOURCE. The volume range [0, 127]. If it exceeds the value range, the interface will correct automatically .

Note: Due to the different implementations of AVRCP version and different mobile phone manufacturers, some mobile phones are not compatible with this function. iPhone series phones support this interface well.

- `int rk_bt_sink_disconnect()`

  Disconnect A2DP Sink.

- `int rk_bt_sink_connect_by_addr(char *addr)`

  Connect to the device specified by addr actively; addr: device address, like "94:87:E0:B6:6D:AE"

- `int rk_bt_sink_disconnect_by_addr(char *addr)`

  Disconnect the device specified by addr actively; addr: device address, like "94:87:E0:B6:6D:AE"

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

  Get the address of the currently connected remote device (BLUEZ only)

- `int rk_bt_sink_get_play_status()`

  Get the playback status of the currently connected remote device. When the remote device does not support reporting the playback progress actively, you can get the playback progress through this interface. Calling this interface will trigger the RK_BT_AVRCP_PLAY_POSITION_CB callback.

- `bool rk_bt_sink_get_poschange()`

  Whether the currently connected remote device supports reporting the progress of the playback actively; if it does, returns true, otherwise returns false.

- `void rk_bt_sink_set_alsa_device(char *alsa_dev)`

  To set the Bluetooth playback device node, it must be called after rk_bt_sink_open. Use "default" by default, this interface is only applicable to bsa (BSA only)

  The bluez playback device node is located in external/bluez-alsa/utils/aplay.c, which can be modified by yourselves.

6. A2DP SOURCE Interface Introduction (RkBtSource.h)

- `BtDeviceInfo` introduction

```
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;
```

The above structure is used to save the scanned device information. name: device's name. address: device's address. rssi_valid: indicates whether rssi is valid. rssi: signal strength. playrole: device role, values: "Audio Sink", "Audio Source", "Unknown".

- `BtScanParam` introduction

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

This structure is used to save the list of devices scanned in the rk_bt_source_scan (BtScanParam * data) interface. mseconds: scan time. item_cnt: the number of scanned devices. devices: device's information. BT_SOURCE_SCAN_DEVICES_CNT value is 30, which means that the interface scans up to 30 devices.

- `RK_BT_SOURCE_EVENT` introduction

```
typedef enum {
    BT_SOURCE_EVENT_CONNECT_FAILED, //fail to connect A2DP Sink device
    BT_SOURCE_EVENT_CONNECTED,      //connect to A2DP Sink device
successfully
    BT_SOURCE_EVENT_DISCONNECT_FAILED,  //fail to diconnect(BLUEZ only)
    BT_SOURCE_EVENT_DISCONNECTED,   //disconnect
    /* reverse control event on the Sink side*/
    BT_SOURCE_EVENT_RC_PLAY,        //play
    BT_SOURCE_EVENT_RC_STOP,        //stop
    BT_SOURCE_EVENT_RC_PAUSE,       //pause
    BT_SOURCE_EVENT_RC_FORWARD,     //Previous
    BT_SOURCE_EVENT_RC_BACKWARD,    //next
    BT_SOURCE_EVENT_RC_VOL_UP,      //volume+
    BT_SOURCE_EVENT_RC_VOL_DOWN,    //volume-
    BT_SOURCE_EVENT_AUTO_RECONNECTING,  //is reconnecting(BLUEZ only)
} RK_BT_SOURCE_EVENT;
```

- `RK_BT_SOURCE_STATUS` introduction

```
typedef enum {
    BT_SOURCE_STATUS_CONNECTED, //connected state
    BT_SOURCE_STATUS_DISCONNECTED, //disconnected state
} 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)`

  Status callback function. userdata: user pointer, bd_addr: address of the connected remote device, name: name of the connected remote device, event: connection event. It is recommended to register the status callback function before the `rk_bt_source_open` interface to avoid state events losing.

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

  Registration status callback function.

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

  Scans nearby Audio Sink devices, and connects to the device with strongest rssi automatically. userdata: user pointer, cb: status callback function. The time for the interface automatically scans is 10 seconds. If no Audio Sink device is scanned within 10 seconds, the interface will not do any operation. If an Audio Sink device is scanned, the basic information of the device will be printed. If the Audio Sink device cannot be scanned, it will print "=== Cannot find audio Sink devices. ==="; if the signal strength of the scanned device is too low, the connection will fail and print “=== BT SOURCE RSSI is too weak !!! ===”.

- `int rk_bt_source_auto_connect_stop(void)`

  Turn off automatic scan.

- `int rk_bt_source_open(void)`

  Open A2DP Source function。

- `int rk_bt_source_close(void)`

  Close A2DP Source function。

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

  Get local device name. name: the buffer to store the name, len: size of the name space

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

  Get the local device address. addr: the buffer to store the address, len: the size of the addr space.

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

  Get A2DP source connection status. pstatus: a pointer to store the current status value. If it is in the connected status, name stores the name of the device on the other side(A2DP Sink), name_len: is the name's length, addr: stores the address of the device on the other side(A2DP Sink), and addr_len is the length of addr. Both the name and addr parameters can be empty.

- `int rk_bt_source_scan(BtScanParam *data)`

  To scan device. The scanning parameters are specified by `data`, and the scanned results are also stored in `data`. For details, please see the introduction of BtScanParam.

- `int rk_bt_source_connect_by_addr(char *address)`

  Connect to the device specified by `address` automatically.

- `int rk_bt_source_disconnect_by_addr(char *address)`

  Disconnect to the device specified by `address` .

- `int rk_bt_source_disconnect()`

  Disconnect.

- `int rk_bt_source_remove(char *address)`

  Delete the connected device. It will not connect automatically after deletion.

- `int rk_bt_source_resume(void)`

Go on playing (BSA only)

- `int rk_bt_source_stop(void)`

  Stop playing (BSA only)

- `int rk_bt_source_pause(void)`

  Pause to play (BSA only)

- `int rk_bt_source_vol_up(void)`

  increase volume (BSA only)

- `int rk_bt_source_vol_down(void)`

  Decrease volume (BSA only)

7. HFP-HF Interface Introduction (RkBtHfp.h)

- `RK_BT_HFP_EVENT` Introduction

```
typedef enum {
    RK_BT_HFP_CONNECT_EVT,      // HFP connected successfully
    RK_BT_HFP_DISCONNECT_EVT,   // HFP disconnected
    RK_BT_HFP_RING_EVT,         // received ringing signal from AG (mobile
phone)
    RK_BT_HFP_AUDIO_OPEN_EVT,   // connected
    RK_BT_HFP_PICKUP_EVT,       // answer the phone actively
    RK_BT_HFP_HANGUP_EVT,       // hangup the phone actively
    RK_BT_HFP_VOLUME_EVT,       // AG (Mobile phone) Volume Change
} RK_BT_HFP_EVENT;
```

- `RK_BT_SCO_CODEC_TYPE` Introduction

```
typedef enum {
    BT_SCO_CODEC_CVSD,                  // CVSD(8K sampling), Bluetooth
required to support
    BT_SCO_CODEC_MSBC,                  // mSBC (16K sampling) , Optional
support
} RK_BT_SCO_CODEC_TYPE;
```

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

  HFP status callback function. event: refer to the introduction of `RK_BT_HFP_EVENT` above. data: when event is `RK_BT_HFP_VOLUME_EVT`, `*((int *)data)` is the volume value displayed on the current AG (mobile phone).*Note: the actual call volume still needs to be handled accordingly on the board.*

- `void rk_bt_hfp_register_callback(RK_BT_HFP_CALLBACK cb)`

  Register a HFP callback function, which is recommended to be called before `rk_bt_hfp_sink_open` to avoid losing state events.

- `int rk_bt_hfp_sink_open(void)`

  Turn on HFP-HF and A2DP SINK functions at the same time. BSA DEVICEIO can call this interface, or call the A2DP Sink open and HFP open interfaces separately to realize the coexistence of HFP-HF and A2DP SINK. But BLUEZ DEVICEIO can only realize the coexistence of HFP-HF and A2DP SINK through this interface.

For A2DP SINK and HFP-HF, the registration of callback functions and the functional interface are still separate. It is best to call `rk_bt_hfp_register_callback` and `rk_bt_sink_register_callback` before `rk_bt_hfp_sink_open` to avoid losing events. For BLUEZ DEVICEIO, before calling `rk_bt_hfp_sink_open` interface, you cannot call `rk_bt_hfp_open` and `rk_bt_sink_open` functions, otherwise the interface returns -1. The reference code is as follows:

```
/*opens A2DP SINK and HFP HF functions in coexistence mode */
rk_bt_sink_register_callback(bt_sink_callback);
rk_bt_hfp_register_callback(bt_hfp_hp_callback);
rk_bt_hfp_sink_open();
```

```
/* close the operation */
rk_bt_hfp_close(); //close HFP HF
rk_bt_sink_close(); //close A2DP SINK
```

- `int rk_bt_hfp_open(void)`

  Turn on HFP service.

  BLUEZ DEVICEIO: this interface is mutually exclusive with `rk_bt_sink_open`. Calling this interface will automatically exit A2DP protocol related services, and then start HFP service. If A2DP SINK and HFP need to coexist, please refer to `rk_bt_hfp_sink_open`.

  BSA DEVICEIO: there is no mutual exclusion between this interface and `rk_bt_sink_open`

- `int rk_bt_hfp_close(void)`

  Turn off HFP service。

- `int rk_bt_hfp_pickup(void)`

  Answer the phone actively

- `int rk_bt_hfp_hangup(void)`

  Hang up actively.

- `int rk_bt_hfp_redial(void)`

  Recall the last dialed phone number in the call list. Note: it is "call out" phone number, not the most recent phone number in the call list. For example, in the following case, calling `rk_bt_hfp_redial` interface will call back rockchip-003.

  <1> rockchip-001 [Call in]

  <2> rockchip-002 [Call in]

  <3> rockchip-003 [Call out]

- `int rk_bt_hfp_dial_number(char *number)`

  Dial the phone number specified by "number"

- `int rk_bt_hfp_report_battery(int value)`

  Report the battery level. value: battery power value, the value range is [0, 9].

- `int rk_bt_hfp_set_volume(int volume)`

  Set the speaker volume of AG (mobile phone). volume: volume value, range is [0, 15]. When AG device is a mobile phone, after calling this interface, the volume progress bar of the Bluetooth call on the mobile phone will change accordingly. However, the actual call volume still needs to be set on the board.

- `void rk_bt_hfp_enable_cvsd(void)`

hfp codec is forced to use CVSD (8K sampling rate), AG (mobile phone) and HF (headphone) will no longer negotiate SCO codec type, at this time the SCO codec type must be forced to BT_SCO_CODEC_CVSD. This interface is only applicable to bsa (BSA only).

Bluez supports 8K and 16K sample rate adaptation. SCO codec type is negotiated and determined by AG (mobile phone) and HF (headphone). It does not support forcing to use of CVSD.

- `void rk_bt_hfp_disable_cvsd(void)`

  It is forbidden to force the use of CVSD (8K sampling rate) by hfp codec. The type of SCO codec is determined through negotiation between AG (mobile phone) and HF (headphone). The result of the negotiation is notified to the application layer through the callback event RK_BT_HFP_BCS_EVT. This interface is only applicable to bsa (BSA only).

- `int rk_bt_hfp_disconnect(void)`

  Disconnect current connection

8. OBEX Interface Introduction (RkBtObex.h BLUEZ only)

- `RK_BT_OBEX_STATE` introduction

```
typedef enum {
    RK_BT_OBEX_CONNECT_FAILED,          //connection failed
    RK_BT_OBEX_CONNECTED,               //connection succeeded
    RK_BT_OBEX_DISCONNECT_FAILED,       //disconnection failed
    RK_BT_OBEX_DISCONNECTED,            //disconnection succeeded
    RK_BT_OBEX_TRANSFER_ACTIVE,         //start transferring
    RK_BT_OBEX_TRANSFER_COMPLETE,       //complete transfer
} RK_BT_OBEX_STATE;
```

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

  obex status callback, bd_addr: address of the connected remote device

- `void rk_bt_obex_register_status_cb(RK_BT_OBEX_STATE_CALLBACK cb)`

  Register obex status callback

- `int rk_bt_obex_init(char *path)`

  Start obexd process, only needs to call this interface to realize Bluetooth file transfer function, path: file storage path

- `int rk_bt_obex_deinit()`

  Close the obexd process and use it with rk_bt_obex_init

- `int rk_bt_obex_pbap_init()`

  To initialize the Bluetooth phone book, you must call rk_bt_obex_init to start obexd before calling this interface

- `int rk_bt_obex_pbap_deinit()`

  To de-initialize the Bluetooth phone book, after calling this interface, you must call rk_bt_obex_deinit to close obexd

- `int rk_bt_obex_pbap_connect(char *btaddr)`

  Open the pbap service, and connect with the device specified by btaddr actively.

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

  Obtain information about the object type specified by dir_name and store it in the file specified by dir_file

  pbab defines six object types:

  "pb": contact phone book

  "ich": call history

  "och": dial history

  "mch": history of missed calls

  "cch": combined history records, that is, all calls, outgoing and missed records

  "spd": speed dial, for example, you can specify button 1 as a contact's speed dial button

  "fav": favorites

- `int rk_bt_obex_pbap_disconnect(char *btaddr)`

  Disconnect with device specified by btaddr actively

9. Demo Program Introduction

The sample program is stored in: external/deviceio /test. The bluetooth-related test cases are implemented in bt_test.cpp, which cover all the above interfaces. The function call is in DeviceIOTest.cpp.

9.1 Build

1. Execute `make deviceio-dirclean && make deviceio -j4` in the SDK root directory, and the following log will be displayed when building is successful (note: only part of log is showed below, rk-xxxx corresponds to the specific project root directory)

```
   -- 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. Run `./build.sh` to generate new firmware, and then flash the new firmware to device.

9.2 Basic Interface Demo Program

9.2.1 Interface Introduction

9.2.1.1 Basic Interface Test Introduction to Bluetooth Service

- void bt_test_bluetooth_init(void *data)

  To initialize Bluetooth test. This interface is called before execute Bluetooth test. To register BLE receiving and data request callback functions, please refer to `bt_server_open` in the DeviceIOTest.cpp test menu.

  Note: BLE reading data is achieved by registering callback functions. When BLE connection receives data, it will call the receiving callback function actively. For details, please refer to introduction of `RkBtContent` *structure and* `rk_ble_register_recv_callback` *function.*

- void bt_test_bluetooth_deinit(char *data)

  Bluetooth de-initialization test, de-initialize all Bluetooth profiles.

- bt_test_set_class(void *data)

  Set the type of Bluetooth device. The current test value is 0x240404.

- bt_test_enable_reconnect(void *data)

  Enable A2DP SINK and HFP auto reconnect function. It is recommended to call immediately after `bt_test_bluetooth_init`.

- bt_test_disable_reconnect(void *data)
- Disable the A2DP SINK and HFP auto-reconnect function. It is recommended to call immediately after `bt_test_bluetooth_init`.

On the phone side:

- void bt_test_get_device_name(char *data)

  Get local device name

- void bt_test_get_device_addr(char *data)

  Get local device address

- void bt_test_set_device_name(char *data)

  Set local device name

- void bt_test_pair_by_addr(char *data)

  Pair with the device at the specified address, data: " 94:87:E0:B6:6D:AE "

- void bt_test_unpair_by_addr(char *data)

  Unpair with the device at the specified address, data: " 94:87:E0:B6:6D:AE "

- void bt_test_get_paired_devices(char *data)

  Get a list of currently paired devices

- void bt_test_free_paired_devices(char *data)

  Release the memory requested in bt_test_get_paired_devices to store paired device information

- void bt_test_get_scaned_devices(char *data)

  Get a list of scanning devices

- void bt_test_start_discovery(char *data)

  Scan surrounding devices, including BR/EDR and LE devices

- void bt_test_start_discovery_bredr(char *data)

Scan the surrounding BR/EDR devices

- void bt_test_start_discovery_le(char *data)

  Scan the surrounding LE devices

- void bt_test_cancel_discovery(char *data)

  Cancel scan operation

- void bt_test_is_discovering(char *data)

  Whether is scanning the surrounding devices

- void bt_test_display_devices(char *data)

  Print the scanned information of surrounding devices

- void bt_test_display_paired_devices(char *data)

  Print the currently paired device information

9.2.1.2 BLE Interface Testing Introduction

1. Install a third-party BLE test APK on your phone, such as nrfconnnect.
2. Choose the `bt_test_ble_start` function.
3. Scans Bluetooth and connects to "ROCKCHIP_AUDIO BLE" on the phone.
4. After the connection is successful, the device will call back the `ble_status_callback_test` function in bt_test.cpp and print "+++++ RK_BLE_STATE_CONNECT +++++".
5. Execute the following functions to do specific functional tests.

- void bt_test_ble_start(void *data)

  To enable BLE. After the device is connected passively, it will receive "Hello RockChip" and responds with "My name is rockchip".

- void bt_test_ble_write(void *data)

  Test BLE write function and send 134 strings with '0'-'9'.

- void bt_test_ble_get_status(void *data)

  Test BLE status interface.

- void bt_test_ble_stop(void *data)

  Disabled BLE.

- void bt_test_ble_disconnect(char *data)

  Disconnect.

9.2.1.3 BLE CLIENT Interface Test Introduction

1. Select bt_test_sink_open function, start ble client
2. Select bt_test_start_discovery or bt_test_start_discovery_le to start scanning the device
3. Enter "60 input xx:xx:xx:xx:xx:xx" and call bt_test_ble_client_connect to connect to the ble server device at the specified address
4. After the connection is successful, the callback ble_client_test_state_callback will be triggered, printing "+++++ RK_BLE_CLIENT_STATE_IDLE +++++"
5. Enter "61 input xx:xx:xx:xx:xx:xx", call bt_test_ble_client_disconnect to disconnect the ble server device at the specified address, and successfully disconnect, will print "+++++ RK_BLE_CLIENT_STATE_DISCONNECT ++++ +"

6. Enter "63 input xx:xx:xx:xx:xx:xx" and call bt_test_ble_client_get_service_info to get the service uuid, characteristic uuid, permission, properties, descriptor uuid and other information of the connected device
7. Enter "64 input uuid", such as "56 input 00009999-0000-1000-8000-00805F9B34FB" to read the data of 9999 uuid through bt_test_ble_client_read. Successful reading will trigger bt_test_ble_client_recv_data_callback to print the read value
8. Enter "65 input uuid", such as "57 input 00009999-0000-1000-8000-00805F9B34FB" and write 9999 uuid via bt_test_ble_client_write
9. Select 59, 68 to turn on or off the notification of the specified uuid

9.2.1.4 A2DP SINK Interface Test Introduction

1. Select the bt_test_sink_open function.
2. Use the mobile phone Bluetooth to scan and connect to "ROCKCHIP_AUDIO".
3. After the connection is successful, the device will call back the bt_sink_callback function in bt_test.cpp and print "++++++++++++ BT SINK EVENT: connect success ++++++++++".
4. Turn on music player of the phone, and make sure it is ready to play songs.
5. Execute the following functions to test specific functions:

- void bt_test_sink_open(void *data)

  Turn on A2DP Sink mode.

- void bt_test_sink_visibility00(void *data)

  Set A2DP Sink to be invisible and unreachable.

- void bt_test_sink_visibility01(void *data)

  Set the A2DP Sink to be invisible and connectable.

- void bt_test_sink_visibility10(void *data)

  Set the A2DP Sink to be visible and disconnectable.

- void bt_test_sink_visibility11(void *data)

  Set A2DP Sink visible and connectable.

- void bt_test_sink_music_play(void *data)

  Control the device to play in reverse.

- void bt_test_sink_music_pause(void *data)

  Control the device to pause in reverse.

- void bt_test_sink_music_next(void *data)

  Control the device to play the next song in reverse

- void bt_test_sink_music_previous(void *data)

  Control the device to play the previous song in reverse.

- void bt_test_sink_music_stop(void *data)

  Control the device to stop playing in reverse.

- void bt_test_sink_reconnect_enable(void *data)

  Enable A2DP Sink auto-connect function.

- void bt_test_sink_reconnect_disenable(void *data)

  Disable the A2DP Sink auto-connect function.

- void bt_test_sink_disconnect(void *data)

Disconnected A2DP Sink。

- void bt_test_sink_close(void *data)

  Close A2DP Sink service。

- void bt_test_sink_status(void *data)

  Query A2DP Sink connection status.

- void bt_test_sink_set_volume(char *data)

  Set volume test

- void bt_test_sink_connect_by_addr(char *data)

  Connect to the device with the specified address, data: " 94:87:E0:B6:6D:AE "

- void bt_test_sink_disconnect_by_addr(char *data)

  Disconnect the device with the specified address, data: " 94:87:E0:B6:6D:AE "

- void bt_test_sink_get_play_status(char *data)

  Get the playback status, it will trigger the "play position change" callback

- void bt_test_sink_get_poschange(char *data)

  Whether the currently connected device supports reporting of playback progress

9.2.1.5 A2DP SOURCE Interface Test Introduction

1. Select `bt_test_source_open` function, start source function
2. Select `bt_test_start_discovery` or `bt_test_start_discovery_bredr` to start scanning the surrounding Bluetooth devices
3. Select `bt_test_source_connect_by_addr` to connect to the addr specified Bluetooth device (27 input xx:xx:xx:xx:xx:xx). After the connection is successful, the device will call back the `bt_test_source_status_callback` function in bt_test.cpp and print "+++++++++" +++ BT SOURCE EVENT: connect sucess ++++++++++".
4. At this time, music will be broadcast from the connected A2DP Sink device.
5. Execute the following functions to do detailed functional tests.

- void bt_test_source_open(char *data)

  Open source function

- void bt_test_source_close(char *data)

  Close source function

- void bt_test_source_connect_status(char *data)

  Get A2DP Source connection status.

- void bt_test_source_connect_by_addr(char *data)

  Connect to the device with specified addr.

- bt_test_source_disconnect

  Disconnect.

- bt_test_source_disconnect_by_addr

  Disconnect the device specified by addr

9.2.1.6 SPP Interface Testing Introduction

1. Install a third-party SPP test APK on the phone, such as "Serial Bluetooth Terminal".
2. Select the `bt_test_spp_open` function.
3. Scan Bluetooth and connects to "ROCKCHIP_AUDIO" on the phone.
4. Open the third-party SPP test APK and connect the device by SPP. After the device is connected successfully, the device will call back the `_btspp_status_callback` function in bt_test.cpp and print "+++++++ RK_BT_SPP_EVENT_CONNECT +++++".
5. Execute the following functions for detailed functional tests.

- void bt_test_spp_open(void *data)

  Open SPP

- void bt_test_spp_write(void *data)

  Test SPP writing function, send “This is a message from rockchip board!” string to the other side

- void bt_test_spp_close(void *data)

  Close SPP

- void bt_test_spp_status(void *data)

  Query SPP connection status

9.2.1.7 HFP Interface Test Introduction

1. Select `bt_test_hfp_sink_open` or `bt_test_hfp_hp_open` function.
2. Scans Bluetooth and connects to "ROCKCHIP_AUDIO" on the mobile phone. Note: If you have already connected the mobile phone before testing SINK function, you should ignore the device at the mobile phone, then scan and connect again.
3. After the device is successfully connected, the device will call back the `bt_test_hfp_hp_cb` function in bt_test.cpp and print "+++++ BT HFP HP CONNECT +++++". If the phone is called, it will print "+++++ BT HFP HP RING +++++" , and "+++++ BT HFP AUDIO OPEN +++++" when the phone is connected. For other status printing, please read the source code of `bt_test_hfp_hp_cb` function in bt_test.cpp directly. Note: If the `bt_test_hfp_sink_open` interface is called, when the device is successfully connected, the connection status of A2DP SINK will also be printed, such as "++++++++++++ BT SINK EVENT: connect success +++++++ +++ ".
4. Execute the following functions for detailed functional tests.

- bt_test_hfp_sink_open

  Open HFP HF and A2DP SINK in coexist mode.

- bt_test_hfp_hp_open

  Open HFP HF function only.

- bt_test_hfp_hp_accept

  Answer the phone actively.

- bt_test_hfp_hp_hungup

  Hang up actively。

- bt_test_hfp_hp_redail

  To re-dial.

- void bt_test_hfp_hp_dial_number(char *data)

  Dial the specified phone number

- bt_test_hfp_hp_report_battery

  Battery power status is reported per second from 0 to 9, At this time, you will see the icon change from empty to full on the phone. Note: Some phones do not support Bluetooth power icon display.

- bt_test_hfp_hp_set_volume

  Set the Bluetooth call volume per second from 1 to 15, . At this time, you will see the Bluetooth call volume progress bar change process on the mobile phone.

  Note: Some mobile phones do not support display the progress bar change dynamically. Actively increasing or decreasing volume to trigger progress bar display. At this time, you will see that the device has set the volume of mobile phone successfully . For example, if the original volume is 0. After running the interface, press the mobile phone volume '+' button and you will find that the volume is full.

- bt_test_hfp_hp_close

  Close HFP service.

- bt_test_hfp_open_audio_diplex

  Open the hfp audio channel, which is called in the callback event RK_BT_HFP_AUDIO_OPEN_EVT.

- bt_test_hfp_close_audio_diplex

  Close the hfp audio channel and which is called in the callback event RK_BT_HFP_AUDIO_CLOSE_EVT.

9.2.1.8 OBEX Interface Test Introduction

Execute the following functions for detailed functional tests:

- bt_test_obex_init

  Open obexd process and execute this function to test the file transfer

- bt_test_obex_deinit

  Close the obexd process

- bt_test_obex_pbap_init

  Execute bt_test_obex_init fore test the Bluetooth phone book

- bt_test_obex_pbap_deinit

  Deinitialize the Bluetooth phone book, and then execute bt_test_obex_deinit

- bt_test_obex_pbap_connect

  Open the pbap service and connect to the specified device

- bt_test_obex_pbap_get_pb_vcf

  Get the contact phone book, the result is stored in /data/pb.vcf

- bt_test_obex_pbap_get_ich_vcf

  Get call history, the results are stored in /data/ich.vcf

- bt_test_obex_pbap_get_och_vcf

  Get outgoing history record, the result is stored in /data/och.vcf

- bt_test_obex_pbap_get_mch_vcf

  Get the history of missed calls, the results are stored in /data/mch.vcf

- bt_test_obex_pbap_disconnect

  Turn off the pbap service, and disconnect

- bt_test_obex_close

  Close obex service

9.2.2 Test Steps

1. Execute the test program command: `DeviceIOTest bluetooth` to display the following interface:

```
# deviceio_test bluetooth
version:V1.3.5
#### 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_cancel_discovery
12.  bt_test_is_discovering
13.  bt_test_display_devices
14.  bt_test_read_remote_device_name
15.  bt_test_get_scaned_devices
16.  bt_test_display_paired_devices
17.  bt_test_get_paired_devices
18.  bt_test_free_paired_devices
19.  bt_test_pair_by_addr
20.  bt_test_unpair_by_addr
21.  bt_test_get_connected_properties
22.  bt_test_source_auto_start
23.  bt_test_source_connect_status
24.  bt_test_source_auto_stop
25.  bt_test_source_open
26.  bt_test_source_close
27.  bt_test_source_connect_by_addr
28.  bt_test_source_disconnect
29.  bt_test_source_disconnect_by_addr
30.  bt_test_source_remove_by_addr
31.  bt_test_sink_open
32.  bt_test_sink_visibility00
33.  bt_test_sink_visibility01
34.  bt_test_sink_visibility10
35.  bt_test_sink_visibility11
36.  bt_test_ble_visibility00
37.  bt_test_ble_visibility11
38.  bt_test_sink_status
39.  bt_test_sink_music_play
40.  bt_test_sink_music_pause
41.  bt_test_sink_music_next
42.  bt_test_sink_music_previous
43.  bt_test_sink_music_stop
44.  bt_test_sink_set_volume
45.  bt_test_sink_connect_by_addr
```

```
46.  bt_test_sink_disconnect_by_addr
47.  bt_test_sink_get_play_status
48.  bt_test_sink_get_poschange
49.  bt_test_sink_disconnect
50.  bt_test_sink_close
51.  bt_test_ble_start
52.  bt_test_ble_set_address
53.  bt_test_ble_set_adv_interval
54.  bt_test_ble_write
55.  bt_test_ble_disconnect
56.  bt_test_ble_stop
57.  bt_test_ble_get_status
58.  bt_test_ble_client_open
59.  bt_test_ble_client_close
60.  bt_test_ble_client_connect
61.  bt_test_ble_client_disconnect
62.  bt_test_ble_client_get_status
63.  bt_test_ble_client_get_service_info
64.  bt_test_ble_client_read
65.  bt_test_ble_client_write
66.  bt_test_ble_client_is_notify
67.  bt_test_ble_client_notify_on
68.  bt_test_ble_client_notify_off
69.  bt_test_ble_client_get_eir_data
70.  bt_test_spp_open
71.  bt_test_spp_write
72.  bt_test_spp_close
73.  bt_test_spp_status
74.  bt_test_hfp_sink_open
75.  bt_test_hfp_hp_open
76.  bt_test_hfp_hp_accept
77.  bt_test_hfp_hp_hungup
78.  bt_test_hfp_hp_redail
79.  bt_test_hfp_hp_dial_number
80.  bt_test_hfp_hp_report_battery
81.  bt_test_hfp_hp_set_volume
82.  bt_test_hfp_hp_close
83.  bt_test_hfp_hp_disconnect
84.  bt_test_obex_init
85.  bt_test_obex_pbap_init
86.  bt_test_obex_pbap_connect
87.  bt_test_obex_pbap_get_pb_vcf
88.  bt_test_obex_pbap_get_ich_vcf
89.  bt_test_obex_pbap_get_och_vcf
90.  bt_test_obex_pbap_get_mch_vcf
91.  bt_test_obex_pbap_disconnect
92.  bt_test_obex_pbap_deinit
93.  bt_test_obex_deinit
94.  bt_server_close
Which would you like:
```

2. Select the corresponding test program number. Firstly, select 01 to initialize the Bluetooth basic service. Such as testing BT Source function。

```
Which would you like:01
#Note: enter the next round of selection interface until finish execution
Which would you like:25
#Note: open source function
Which would you like:8 input 15000
#Note: Start to scan the surrounding Bluetooth devices, scan time is 15s
Which would you like:27 input xx:xx:xx:xx:xx:xx
#Note: Start to connect with the device wtih the address xx:xx:xx:xx:xx:xx
```

3. The test program needed to transfer the address or other parameters, input: number (space) input (space) parameters, such as pairing with the specified address device

```
Which would you like:19 input 94:87:E0:B6:6D:AE
#Note: start pairing with the device with the address of 94:87:E0:B6:6D:AE
```

9.3 BLE Network Configuration Demo Program

Please refer to "Rockchip_Developer_Guide_Network_Config_CN".