rokid bluetooth service

Table of Contents

> 目前蓝牙service包含 bsa 和 bluez 两种协议栈控制方法。rokid 提供了 bluetooth_service 来兼容这两种底层蓝牙控制协议。

软件接口协议

bluetooth_service 为了和别的应用解耦，使用 pub/sub 的设计框架，通过 IPC (进程间通讯)控制蓝牙和解析蓝牙状态。

flora

flora 是 rokid 系统的进程间 pub/sub 设计框架，bluetooth_service 为每一种协议都提供了响应的控制接口和状态接口。具体 ipc 节点如下：
建议参考demo/bluetooth_test.c 配合阅读此文档

string 接口：采用Json封装解析

- common 控制 path : **bluetooth.common.command**
- common 状态 path : **bluetooth.common.event**
- ble 控制 path : **bluetooth.ble.command**
- ble 状态 path : **bluetooth.ble.event**
- ble client 控制 path : **bluetooth.ble.client.command**
- ble client 状态 path : **bluetooth.ble.client.event**
- a2dpsink 控制 path : **bluetooth.a2dpsink.command**
- a2dpsink 状态 path : **bluetooth.a2dpsink.event**
- a2dpsource 控制 path : **bluetooth.a2dpsource.command**
- a2dpsource 状态 path : **bluetooth.a2dpsource.event**
- hfp 控制 path : **bluetooth.hfp.command**
- hfp 状态 path : **bluetooth.hfp.event**
- HID HOST 控制 path : **bluetooth.hh.command**
- HID HOST 状态 path : **bluetooth.hh.event**

binary接口 采用私有rokid head封装，方便解析

- ble client 读 path : **bluetooth.ble.client.read**
- ble client 写 path : **bluetooth.ble.client.write**
- ble client notify path : **bluetooth.ble.client.notify**

COMMON

1. 获取本设备蓝牙MAC地址

```
{ "command": "MODULE_ADDRESS" }
```

> 状态返回

```
{ "module_address": "XX:XX:XX:XX:XX:XX" }
```

2.设置蓝牙可发现模式

```
{ "command": "VISIBILITY", "discoverable": true }
```

> 状态返回

```
{ "broadcast_state": "opened" }
```

3.获取绑定设备列表

```
{ "command": "PAIREDLIST" }
```

> 状态返回

```
{ "action": "pairedList", "results": { "deviceList": [ { "addre
ss": "38:f5:54:02:39:6a", "name": "Vmedia 8" }, { "address": "5
0:01:d9:38:2e:f2", "name": "Sweet" }, { "address": "6c:21:a2:2e
:60:6a", "name": "Rokid-Me-666666" }, { "address": "34:88:5d:7e
:ec:f9", "name": "Keyboard K480" } ] } }
```

4.从绑定列表删除某个设备

```
{ "command": "REMOVE_DEV", "address": "34:88:5d:7e:ec:f9" }
```

```
{ "command": "REMOVE_DEV", "address": "*" }
```

> "address": "*" 会删除所有配对设备
> 状态返回

```
{ "action": "remove_dev", "address": "34:88:5D:7E:EC:F9", "stat
us": 0 }
```

> status 非0则代表删除失败,如果有empty:true标记则表示绑定列表已空

```
{ "action": "remove_dev", "address": "34:88:5D:7E:EC:F9", "empt
y": true, "status": 0 }
```

5.搜索

```
{ "command": "DISCOVERY", "type": 2 }
```

> type 值详见: enum bt_profile_type 头文件：hardware/bt_type.h
>
> 状态返回

```
{ "action": "discovery", "results": { "deviceList": [ { "addres
s": "43:4b:a1:fe:ae:5b", "name": "XXX" }, { "address": "18:bc:5
a:17:0d:21", "name": "DINGTALK_DMC01" } ], "is_completed": fals
e } }
```

> is_completed : true 表示搜索完毕

6.取消搜索

```
{ "command": "CANCEL_DISCOVERY" }
```

BLE SERVICE

BLE控制命令

1. 打开 ble

```
{ "proto": "ROKID_BLE", "command": "ON", "name": "xxxxxx" }
```

> 如果需要自动关闭非 ble service的 profile，可增加 unique 字段

```
 { "proto": "ROKID_BLE", "command": "ON", "name": "xxxxxx", "un
```

```
ique": true }
```

2.关闭 ble

```
{ "proto": "ROKID_BLE", "command": "OFF" }
```

ble 状态

```
{ "state": "opened" }
{ "state": "closed" }
{ "state": "connected" }
{ "state": "handshaked" }
{ "state": "disconected" }
{ "state": "openfailed" }
```

ble 数据

1. rokid ble 配网协议
   bluetooth_service 目前对 rokid 配网 sdk 的协议进行了封装，所以里面包含 rokid 的配网私有协议。目前不支持 ble 裸流的处理，所以 ble 配网必须使用 rokid 的配网sdk。具体配网协议详情请见附件（rokid 配网协议.pdf）。
2. 发送数据

```
{ "proto": "ROKID_BLE", "data": "xxxx" }
```

> 发送raw数据（没有rokid msgid封装）

```
{ "proto": "ROKID_BLE", "rawdata": "xxxx" }
```

3.接收数据

```
{ "data": "xxxx" }
```

A2DP_SINK

控制命令

1. 打开A2dpsink

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx"}
```

> 如果需要自动关闭非 a2dpsink 的 profile，可增加 unique 字段;（不会自动关闭HFP）

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx", "unique": true}
```

> 如果需要同时打开蓝牙电话，可以增加 "sec_pro": "HFP"

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx", "sec_pro": "HFP"}
```

> 如果需要自定义打开时，是否根据历史记录自动重连，蓝牙是否可发现，可设置auto_connect discoverable属性, 缺省默认均为true

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx", "unique": true, "auto_connect": false, "discoverable": false}
```

2.打开A2dpsink，连接后，做 sub 操作

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx", "subquent": "PLAY" }
```

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx", "subquent": "PAUSE" }
```

```
{ "proto": "A2DPSINK", "command": "ON", "name": "xxxxxx", "subquent": "STOP" }
```

3.关闭a2dpsink

```
{ "proto": "A2DPSINK", "command": "OFF"}
```

> 同时关闭蓝牙HFP

```
 { "proto": "A2DPSINK", "command": "OFF", "sec_pro": "HFP"}
```

4.播放音乐

```
{ "proto": "A2DPSINK", "command": "PLAY"}
```

5.暂停音乐

```
{ "proto": "A2DPSINK", "command": "PAUSE"}
```

6.停止音乐

```
{ "proto": "A2DPSINK", "command": "STOP"}
```

7.下一首

```
{ "proto": "A2DPSINK", "command": "NEXT"}
```

8.上一首

```
{ "proto": "A2DPSINK", "command": "PREV"}
```

9.断开设备

```
{ "proto": "A2DPSINK", "command": "DISCONNECT"}
DISCONNECT: 会断开a2dp sink和HFP的连接
```

```
{ "proto": "A2DPSINK", "command": "DISCONNECT_PEER"}
DISCONNECT_PEER ： 只会断开a2dp sink profile连接
```

10.静音

> 主要应用场景，是激活唤醒时，需要立即暂停蓝牙音乐的输出，但pause命

> 令实现延迟较高，mute则会立即生效。mute设备为若琪，并非手机

```
{ "proto": "A2DPSINK", "command": "MUTE"}
```

11.取消静音

```
{ "proto": "A2DPSINK", "command": "UNMUTE"}
```

12.暂停+静音

```
{ "proto": "A2DPSINK", "command": "PAUSE_MUTE"}
```

13.播放+取消静音

```
{ "proto": "A2DPSINK", "command": "PLAY_UNMUTE"}
```

14.获取播放歌曲信息

```
{ "proto": "A2DPSINK", "command": "GETSONG_ATTRS"}
```

15.连接

```
{ "proto": "A2DPSINK", "command": "CONNECT", "address": "xx:xx:xx:xx:xx:xx" }
```

16.主动获取当前状态信息

```
{ "proto": "A2DPSINK", "command": "GETSTATE" }
```

状态信息

> action: stateupdate
> a2dpstate： a2dp打开状态
> connect_state: 连接状态
> play_state：播放状态
> broadcast_state：蓝牙可发现状态

> linknum：打开蓝牙时，自动尝试连接设备个数，最大2。

1. 打开A2dpsink后，open 正常

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_stat
e": "invailed", "play_state": "invalid", "broadcast_state": "op
ened", "linknum": xx }
```

2.打开A2dpsink后，open 异常

```
{ "action": "stateupdate",  "a2dpstate": "open failed", "connec
t_state": "invailed", "play_state": "invalid", "broadcast_state
": "invalid"}
```

3.设备已连接

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_stat
e": "connected", "connect_name": "*******", "play_state": "inva
lid", "broadcast_state": "closed" }
```

4.设备断开连接

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_stat
e": "disconnected", "play_state": "invailed",  "broadcast_state
": "opened"}
```

5.设备播放音乐

> 播放状态 同步会设置prop属性 audio.bluetooth.playing : true
> 可以通过getprop audio.bluetooth.playing 命令查看
> C代码可以通过 property_get("audio.bluetooth.playing", value, "false");

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_stat
e": "connected", "connect_name": "******", "play_state": "playe
d", "broadcast_state": "closed"}
```

6.暂停停止播放音乐

```
{ "action" : "stateupdate", "a2dpstate": "opened", "connect_state": "connected", "connect_name": "******", "play_state": "stoped", "boardcast_state": "closed" }
```

7.播放歌曲信息

```
{ "action": "element_attrs", "title": "That Girl", "artist": "Olly Murs", "album": "24 HRS (Deluxe)", "track_num": "19", "track_nums": "25", "genre": "", "time": "176819" }
依次表示：titel:歌曲名；artist:艺术家；album:专辑；track_num:播放列表中第几首。track_nums：播放列表歌曲总数；genre：歌曲风格。time：歌曲总时间，单位ms
```

音量

蓝牙服务不涉及系统音量的更改，需要系统 core 程序对系统音量进行调整。部分蓝牙 source 设备直接通过修改自身的音频流进行控制，此时不需要 core 对应更改。另外一部分蓝牙设备会发送蓝牙音量，core 程序需要关注，进行匹配。

1. BT audio stream init OK 此时可通过pulseaudio来初始化本地蓝牙音量

```
{ "action" : "volumechange" }
```

2.BT audio volume change

```
{ "action" : "volumechange", "value" : 65 }
```

3.设置Absolutely volume同步给remote source设备

> 需要手机支持，不支持则命令设置无效，手机不会响应

```
{ "proto": "A2DPSINK", "command": "VOLUME", "value": 100 }
```

HFP

HFP建议和a2dp sink结合一起使用。
目前BT service基于bluez协议栈暂不支持HFP。
硬件必须使用PCM（蓝牙SCO）接口通讯蓝牙语音数据。

控制命令

1. 打开HFP

```
 { "proto": "HFP", "command": "ON", "name": "xxxxxx"}
```

2.关闭HFP

```
 { "proto": "HFP", "command": "OFF"}
```

3.断开设备

```
{ "proto": "HFP", "command": "DISCONNECT"}
```

```
{ "proto": "HFP", "command": "DISCONNECT_PEER"}
```

4.拨号

```
{ "proto": "HFP", "command": "DIALING", "NUMBER": "10086"}
```

5.接听

```
{ "proto": "HFP", "command": "ANSWERCALL"}
```

6.挂断

```
{ "proto": "HFP", "command": "HANGUP"}
```

状态信息

1. "action": "stateupdate"

```
{ "action": "stateupdate", "hfpstate": "opened", "connect_state
": "connected", "connect_name": "******", "service": "active",
"call": "inactive", "setup": "incoming", "held": "none","audio"
: "on" }
```

> hfpstate: HFP profile打开状态。可能值：opened, open failed, closing, closed, invalid
>
> connect_state: HFP连接状态。可能值：connecting, connected, connect failed, disconnecting, disconnected, invalid
>
> service: 手机网络状态。可能值：inactive, active
> active表示手机可拨打电话或接听电话
>
> call: 通话状态。可能值：inactive, active
> active表示手机已经处于通话接通状态。
>
> setup: 电话处理状态。可能值及意义：none：不需要任何处理（或处理已经完毕），incoming:手机来电状态。outgoing:手机正在拨号。alerting: 手机A拨打B的手机号，B的手机接收到被拨打。（即outgoing拨号时，呼叫的手机号码有效，且被呼叫号码手机响应了呼叫)。
>
> held: 通话保持状态。可能值：none: 没有被保持电话, hold_active：手机有一个active通话，并有一个保持电话, hold：手机通话被保持
>
> audio: HFP audio状态，表示当前是否有蓝牙电话音频流输入输出。可能值：on, off.
> 手机端可以主动选择使用蓝牙，听筒，外放(免提)来接听、拨打电话，包括通话中也能切换
>
> 上述例子表示：hfp打开成功，且已连接手机设备，手机网络正常，处于来电状态，没有接通，蓝牙音频流已开启，此时用户可以选择接通，挂断。

2.来电铃声

```
{ "action": "ring", "audio": "off" }
```

> 需要根据audio状态，确认是否需要播放本地来电铃声。
> audio: on 表示蓝牙音频已建立，手机已通过蓝牙发送了来电铃声
> audio: off 表示蓝牙音频并未建立，需要本地去播放来电铃声。
> 此消息可能会上报多次
> 如接通电话，请及时关闭本地播放的来电铃声

A2DP_SOURCE

目前BT service基于bluez协议栈对souce支持未经量产验证。

控制命令

1. 打开A2dpsource

```
 { "proto": "A2DPSOURCE", "command": "ON", "name": "xxxxxx"}
```

> 如果需要自动关闭非 a2dpsource 的 profile，需要增加 unique 字段;

```
 { "proto": "A2DPSOURCE", "command": "ON", "name": "xxxxxx", "u
nique": true}
```

2.关闭A2dpsource

```
 { "proto": "A2DPSOURCE", "command": "OFF"}
```

3.连接设备

```
{ "proto": "A2DPSOURCE", "command": "CONNECT", "address": "xx:x
x:xx:xx:xx:xx" }
```

4.断开设备

```
{ "proto": "A2DPSOURCE", "command": "DISCONNECT"}
```

```
{ "proto": "A2DPSOURCE", "command": "DISCONNECT_PEER"}
```

状态命令

1. 打开A2dpsource后，open 正常

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_stat
e": "invailed", "broadcast_state": "opened", "linknum": xx }
```

2.打开A2dpsource后，open 异常

```
{ "action": "stateupdate",  "a2dpstate": "open failed", "connect_state": "invailed", "broadcast_state": "invalid"}
```

3.设备连接后

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_state": "connected", "connect_name": "*******", "broadcast_state": "closed" }
```

4.断开连接后

```
{ "action": "stateupdate", "a2dpstate": "opened", "connect_state": "disconnected", "broadcast_state": "opened"}
```

BLE CLIENT

目前BT service基于bluez协议栈对BLE CLIENT功能暂不支持

控制命令

1.打开

```
{ "proto": "BLE_CLIENT", "command": "ON", "name": "XXX" }
```

2.关闭

```
{ "proto": "BLE_CLIENT", "command": "OFF" }
```

3.连接

```
{ "proto": "BLE_CLIENT", "command": "CONNECT", "address": "xx:xx:xx:xx:xx:xx" }
```

4.断开

```
{ "proto": "BLE_CLIENT", "command": "DISCONNECT" }
```

5.使能监听某个service uuid 中的character uuid notify消息

```
{ "proto": "BLE_CLIENT", "command": "REGISTER_NOTIFY", "address
": "38:f5:54:02:39:6a", "service": "AB5E0001-5A21-4F05-BC7D-AF0
1F617B664", "character": "AB5E0003-5A21-4F05-BC7D-AF01F617B664"
 }
```

6.READ某个service 的character uuid

```
{ "proto": "BLE_CLIENT", "command": "READ", "address": "38:f5:5
4:02:39:6a", "service": "AB5E0001-5A21-4F05-BC7D-AF01F617B664",
 "character": "AB5E0003-5A21-4F05-BC7D-AF01F617B664" }
```

状态信息

"action": "stateupdate"

```
{ "action": "stateupdate", "state": "opened", "connect_state":
"disconnected", "name": "Vmedia 8", "address": "38:f5:54:02:39:
6a" }
```

> state: 包含opened、 openfailed、 closed、 invalid;
> connect_state: 包含connected、 connectfailed、 disconnected、 invalid;

数据

BLE CLIENT数据通道均为**二进制数据，不能通过JSON解析**
为方便解析数据，数据会增加rokid定义的私有head数据，定义在 bluetooth_service/ble_client.h

```
#pragma pack(1)
struct rokid_ble_client_header {
    uint32_t len; // header length
    uint8_t address[18];
    uint8_t service[MAX_LEN_UUID_STR];
    uint8_t character[MAX_LEN_UUID_STR];
    uint8_t description[MAX_LEN_UUID_STR];
};
#pragma pack()
```

> address: remote设备地址即 BLE SERVICE设备的地址
> 格式 XX:XX:XX:XX:XX:XX strlen 为17
>
> sevice、character、description：为UUID, strlen为36, 格式为"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", 代表128bit UUID

1. 发送REGISTER_NOTIFY命令后，如果此character uuid有notify消息则会通过 **bluetooth.ble.client.notify** 发送数据过来

> 简单的解析例子

```
flora_agent_subscribe(agent, "bluetooth.ble.client.notify", ble
_client_notify_callback, NULL);

static void ble_client_notify_callback(const char* name, caps_t
 msg, uint32_t type, void* arg) {
    const void *buf = NULL;
    uint32_t length;
    struct rokid_ble_client_header *head;
    char *data;
    int i;

    caps_read_binary(msg, &buf, &length);
    head = (struct rokid_ble_client_header *)buf;
    data = (char *)buf + head->len;
    printf("ble client notify: length(%d), data(%d)\n", length,
 length - head->len);
    printf("ble client notify: address %s, service %s, characte
r %s\n", head->address, head->service, head->character);
    printf("ble client notify data:\n");
    for (i = 0; i < (length - head->len); i++) {
        printf("0x%02X ", data[i]);
    }
    printf("\n");
}
```

2.发送READ命令后，会通过**bluetooth.ble.client.read** 发送数据过来，解析方式同notify
3.发送数据给remote设备某个UUID可以通过**bluetooth.ble.client.write**

```
>比如发送字节0x0A 0x0B 0X0C 三个字节例子如下
```

```
static flora_agent_t agent;
static void bt_flora_send_binary_msg(char* name, uint8_t *buf,
int len) {
    caps_t msg = caps_create();
    caps_write_binary(msg, (const char *)buf, len);
    flora_agent_post(agent, name, msg, FLORA_MSGTYPE_INSTANT);
    caps_destroy(msg);
}
static void ble_client_write(void) {
    struct rokid_ble_client_header *head;
    unsigned char buff[1024] = {0};
    unsigned int data_len = 3;
    unsigned char *data;

    head = (struct rokid_ble_client_header *)buff;
    head->len = sizeof(struct rokid_ble_client_header);
    snprintf(head->address, sizeof(head->address),"11:22:33:44:
55:66");
    snprintf(head->service, sizeof(head->service),"0000ffe1-000
0-1000-8000-00805f9b34fb");
    snprintf(head->character, sizeof(head->character),"00002a07
-0000-1000-8000-00805f9b34fb");
    data = buff + head->len;
    data[0] = 0x0A;
    data[1] = 0x0B;
    data[2] = 0x0C;
    bt_flora_send_binary_msg("bluetooth.ble.client.write", buff
, head->len + data_len);
}
```

HID HOST

目前BT service基于bluez协议栈对HID HOST功能暂不支持

控制命令

1.打开

```
{ "proto": "HH", "command": "ON", "name": "XXX" }
```

2.关闭

```
{ "proto": "HH", "command": "OFF" }
```

3.连接

```
{ "proto": "HH", "command": "CONNECT", "address": "xx:xx:xx:xx:
xx:xx" }
```

4.断开

```
{ "proto": "HH", "command": "DISCONNECT" }
```

状态信息

```
{ "action": "stateupdate", "state": "opened", "connect_state":
"connected", "name": "Vmedia 8", "address": "38:f5:54:02:39:6a"
 }
```

> "state" : opened、openfailed、closed、invalid
> "connect_state": connected、 connectfailed、 disconnected、 invalid

调试方法

> bluetooth 代码仓库位于*frameworks/native/services/bluetooth_service*。在仓库里面的 demo 目录下面有两个工具。一个是 bt_monitor(可以抓取蓝牙的所有通讯节点的数据) 和一个 bt_service_test(可以模拟第三方应用通过命令去操作蓝牙单项功能)。运行效果如下：

bt_monitor

```
/ # bt_monitor
[2019-07-04 20:52:22 255]  name :: bluetooth.ble.client.event
[2019-07-04 20:52:22 255]  data :: { "action": "stateupdate", "sta
[2019-07-04 20:52:22 255]  name :: bluetooth.hh.event
[2019-07-04 20:52:22 256]  data :: { "action": "stateupdate", "sta
[2019-07-04 21:27:40 709]  name :: bluetooth.a2dpsink.command
[2019-07-04 21:27:40 709]  data :: { "name": "Rokid-Me-9999zz", "
[2019-07-04 21:27:40 715]  name :: bluetooth.a2dpsink.event
[2019-07-04 21:27:40 715]  data :: { "linknum": 1, "a2dpstate": "
```

bt_service_test

```
/ # bt_service_test
Bluetooth test CMD menu:
    0 common
    1 ble
    2 a2dp_sink
    3 a2dp_source
    4 hfp
    5 hid host
    6 ble client
    99 exit
```

选2 进入 a2dp sink菜单如下 可以按照菜单简单操作蓝牙

```
Select cmd => 2
Bluetooth a2dp sink CMD menu:
    1 a2dp sink on
    2 a2dp sink off
    3 play
    4 pause
    5 stop
    6 next
    7 prev
    8 disconnect
    9 on play
    10 mute
    11 unmute
    12 disconnect_peer
    13 set volume
    14 getstate
    15 call getstate
    16 getsong attrs
    20 play & unmute
    21 pause & mute
    50 a2dp sink on &&  hfp on
    51 a2dp sink off &&  hfp off
    99 exit
```