Realtek Linux Bluetooth Porting Guide

Date: 2018/08/01

Version: 5.0

Contents

1. Overview

This document describes how to support Realtek Bluetooth module in Linux, mainly the instructions for porting BlueZ and Bluetooth drivers in the system.

2. Platform Confirmation

Realtek has several different Bluetooth module solutions. First, clients should confirm the features of the module in their platforms. Basically, focus on the aspects below.

2.1. Module interface and eFuse

The Realtek Bluetooth module has three kinds of interfaces: USB, UART and PCM.
For UART, the protocol is divided into H4 or Three-wire (H5). At the same time, Realtek Bluetooth can support single/dual antenna. These settings are set in eFuse. It is necessary to confirm whether the module settings are correct. In addition, some settings can be modified by a config file in driver package, such as serial baud rate, PCM parameters and 32k clock setting, etc. For more details, please refer to **UART interface BT controller initial guide-H5/H4.pdf.**

2.2. Driver installation

If BlueZ has been installed into customer system and the kernel has net/bluetooth built with, the driver can be installed by the shell command line that is mentioned in Readme.txt file released in driver package. In that case, pass over chapter **3 Kernel setting**.

2.3. Porting BlueZ

If there is no BlueZ in customer system or BlueZ needs to update, please download the required version from the official website www.bluez.org and (cross) compile the source code.

2.4. Support for other stack than BlueZ

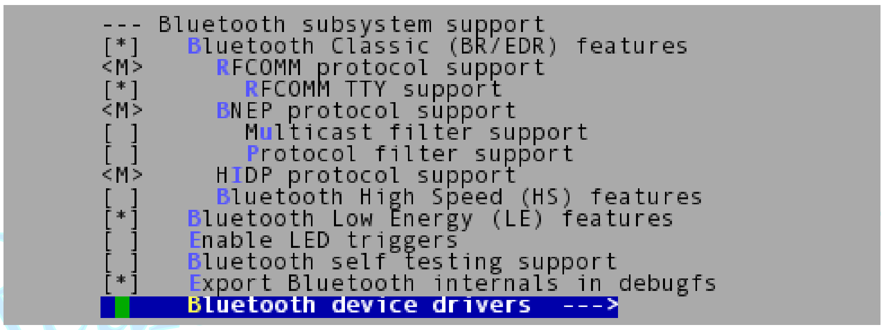
If customers wish to use a stack other than BlueZ, they will need to develop the driver themselves. In the case, please refer to the document **UART interface BT controller initial guide-H4/-H5.pdf** or **USB interface BT controller initial guide.pdf**.

3. Kernel setting

3.1. BlueZ kernel stack

<kernel>/net/bluetooth option can be set by using the shell command make menuconfig. You can also directly modify the platform's <kernel>/.config. Make sure the platform supports Bluetooth after configuration. The corresponding code is in the < kernel>/net/bluetooth:

make menuconfig (e.g. kernel 4.8) [Networking support > Bluetooth subsystem support]



```
--- Bluetooth subsystem support
[*]   Bluetooth Classic (BR/EDR) features
<M>     RFCOMM protocol support
[*]       RFCOMM TTY support
<M>     BNEP protocol support
[ ]       Multicast filter support
[ ]       Protocol filter support
<M>     HIDP protocol support
[ ]     Bluetooth High Speed (HS) features
[*]   Bluetooth Low Energy (LE) features
[ ]   Enable LED triggers
[ ]   Bluetooth self testing support
[*]   Export Bluetooth internals in debugfs
      Bluetooth device drivers  --->
```

Figure 3-1 BlueZ kernel stack set in make menuconfig

<kernel>/.config:

```
CONFIG_BT=m
CONFIG_BT_BREDR=y
CONFIG_BT_RFCOMM=m
CONFIG_BT_RFCOMM_TTY=y
CONFIG_BT_BNEP=m
CONFIG_BT_HIDP=m
CONFIG_BT_LE=y
CONFIG_BT_DEBUGFS=y
```

Please don't select RFCOMM, BNEP (for PAN), HIDP (Classic mouse/keyboard) if they are not required.

3.2. UART driver

Please copy all the C source or header files in bluetooth_uart_driver/ of Realtek Bluetooth UART driver to <kernel>/drivers/bluetooth/.

Modify the Kconfig and Makefile in the <kernel>/drivers/bluetooth to add support for

Realtek Three-wire (H5):

Kconfig:

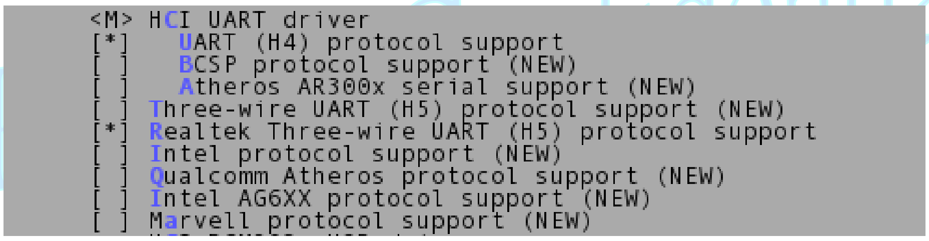
```
config BT_HCIUART_RTL3WIRE
        bool "Realtek Three-wire UART (H5) protocol support"
        depends on BT_HCIUART
        help
            Realtek Three-wire UART (H5) transport layer makes it possible
            to use Realtek Bluetooth controller with Three-wire UART.

            Say Y here to compile support for Realtek Three-wire UART.
```

Makefile:

```
hci_uart-y                          += rtk_coex.o
hci_uart-$(CONFIG_BT_HCIUART_RTL3WIRE)      += hci_rtk_h5.o
```

After changing above files, please run the shell command make menuconfig to enable Realtek Bluetooth UART driver. You can also change the <kernel>/.config directly.

make menuconfig [Networking support > Bluetooth subsystem support > Bluetooth device drivers]



```
<M> HCI UART driver
[*]   UART (H4) protocol support
[ ]   BCSP protocol support (NEW)
[ ]   Atheros AR300x serial support (NEW)
[ ] Three-wire UART (H5) protocol support (NEW)
[*] Realtek Three-wire UART (H5) protocol support
[ ] Intel protocol support (NEW)
[ ] Qualcomm Atheros protocol support (NEW)
[ ] Intel AG6XX protocol support (NEW)
[ ] Marvell protocol support (NEW)
```

Figure 3-2 Realtek Three-wire UART (H5) support in make menuconfig

<kernel>/.config

```
CONFIG_BT_HCIUART=m
CONFIG_BT_HCIUART_H4=y
CONFIG_BT_HCIUART_RTL3WIRE=y
```

3.3. USB driver setting

Please copy all the C source or header files in bluetooth_usb_driver/ of Realtek Bluetooth USB driver package to the <kernel>/drivers/bluetooth/ directory.

Modify Kconfig and Makefile in <kernel>/drivers/bluetooth to add support for Realtek Bluetooth HCI USB driver:

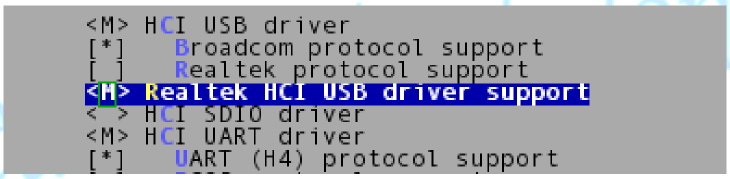
Kconfig:

```
config BT_HCIBTUSB_RTLBTUSB
        tristate "Realtek HCI USB driver support"
        depends on USB
        help
          Realtek Bluetooth HCI USB driver.
          This driver is required if you want to use Realtek Bluetooth
          device with USB interface.

          Say Y here to compile support for Bluetooth USB devices into the

          kernel or say M to compile it as module (rtk_btusb).
```

Makefile:

```
obj-$(CONFIG_BT_HCIBTUSB_RTLBTUSB)      := rtk_btusb.o
rtk_btusb-objs                          := rtk_bt.o rtk_misc.o rtk_coex.o
```

After changing above files, please run the shell command make menuconfig to enable Realtek Bluetooth USB driver. You can also change the <kernel>/.config directly.
make menuconfig [Networking support > Bluetooth subsystem support > Bluetooth device drivers]



```
<M> HCI USB driver
[*]   Broadcom protocol support
[ ]   Realtek protocol support
<M> Realtek HCI USB driver support
< > HCI SDIO driver
<M> HCI UART driver
[*]   UART (H4) protocol support
```

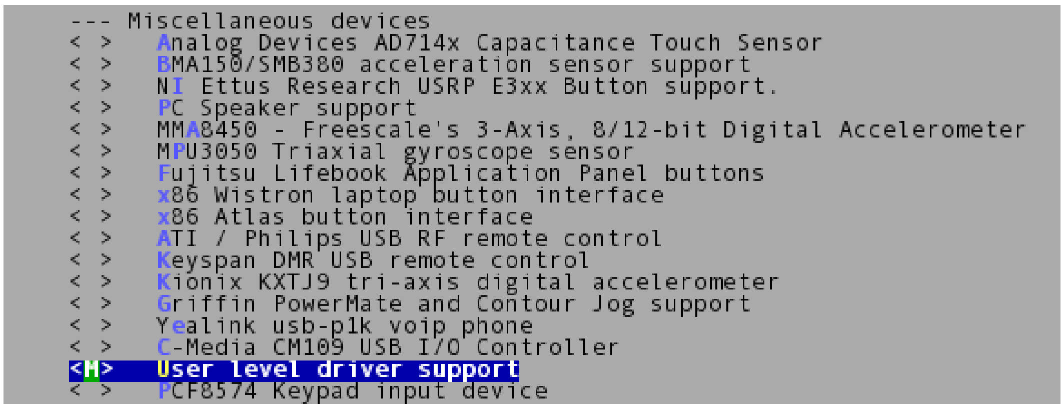
Figure 3-3 Realtek HCI USB driver support in make menuconfig

<kernel>/.config:

```
CONFIG_BT_HCIBTUSB_RTLBTUSB=m
```

3.4. UINPUT for AVRCP

If you want to send AVRCP Up, Down, Left and other button values and press/release information to the user-space program through the input device, you need to open the kernel user level driver support (UINPUT).
make menuconfig [Device Drivers > Input device support > Miscellaneous devices]



```
    --- Miscellaneous devices
    < >   Analog Devices AD714x Capacitance Touch Sensor
    < >   BMA150/SMB380 acceleration sensor support
    < >   NI Ettus Research USRP E3xx Button support.
    < >   PC Speaker support
    < >   MMA8450 - Freescale's 3-Axis, 8/12-bit Digital Accelerometer
    < >   MPU3050 Triaxial gyroscope sensor
    < >   Fujitsu Lifebook Application Panel buttons
    < >   x86 Wistron laptop button interface
    < >   x86 Atlas button interface
    < >   ATI / Philips USB RF remote control
    < >   Keyspan DMR USB remote control
    < >   Kionix KXTJ9 tri-axis digital accelerometer
    < >   Griffin PowerMate and Contour Jog support
    < >   Yealink usb-p1k voip phone
    < >   C-Media CM109 USB I/O Controller
    <M>   User level driver support
    < >   PCF8574 Keypad input device
```

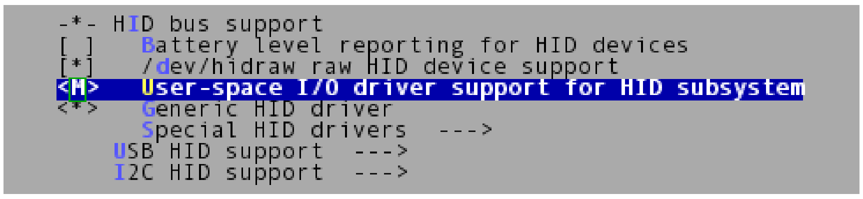
Figure 3-4 User lever input device support

<kernel>/.config

```
CONFIG_INPUT_MISC=y
CONFIG_INPUT_UINPUT=m
```

3.5. UHID for HOGP

If you want to send HoG KEY_1, KEY_2, KEY_ESC and other key values to the user-space program through the input device, you need to open the kernel user-space I/O driver support for HID subsystem(UHID).
make menuconfig [Device Drivers > HID support]



```
    -*- HID bus support
    [ ]   Battery level reporting for HID devices
    [*]   /dev/hidraw raw HID device support
    <M>   User-space I/O driver support for HID subsystem
    <*>   Generic HID driver
          Special HID drivers  --->
        USB HID support  --->
        I2C HID support  --->
```

Figure 3-5 UHID driver support

<kernel>/.config

```
CONFIG_UHID=m
```

4. Porting BlueZ

4.1. Compile bluez

Please download the BlueZ source code from the BlueZ official website **www.bluez.org** and compile it. In order to compile Bluetooth utilities, you need some dependent libraries, such as D-Bus, GLib, etc. BlueZ 5.xx is recommended. The newer the version, the more features are supported and the code is more mature. If you need to support the LE function, you need kernel version 3.5 and above, BlueZ 5.0 and above. Realtek Bluetooth driver supports kernel version 2.6.32 and above.

BlueZ can be compiled with reference to <bluez-5.xx>/README. First, you should run the script configure. You can add or subtract some functions through configure. For example, --enable-test can compile some test tools under <bluez-5.xx>/test. The BlueZ --enable-experimental option adds plugins such as heartrate. More details can be found in Makefile.plugins. Those plugins are compiled under the condition if EXPERIMENTAL.

There are two important configurations: CONFIGDIR and STORAGEDIR. Their corresponding configure parameters are --sysconfdir and --localstatedir. CONFIGDIR is where the BlueZ daemon bluetoothd configuration file is stored. STORAGEDIR is the location where BlueZ bluetoothd stores each adapter and its associated remote Bluetooth devices information.

<bluez-5.xx>/configure

```
if (test "$sysconfdir" = '${prefix}/etc'); then
        configdir="${prefix}/etc/bluetooth"
else
        configdir="${sysconfdir}/bluetooth"
fi

if (test "$localstatedir" = '${prefix}/var'); then
        storagedir="${prefix}/var/lib/bluetooth"
else
        storagedir="${localstatedir}/lib/bluetooth"

fi
```

If you follow the README settings: ./configure --prefix=/usr --mandir=/usr/share/man --sysconfdir=/etc --localstatedir=/var --libexecdir=/lib, CONFIGDIR is /etc/bluetooth and STORAGEDIR is /var/lib/bluetooth. bluetoothd will look up the main.conf in CONFIGDIR for configuration when it’s starting. You can set the Name, Class and other information in main.conf. After porting, main.conf should be placed in the corresponding directory.

In STORAGEDIR, bluetoothd will create a folder for each adapter. In adapter address folder, bluetoothd will create a folder for each remote device with address, and the info file will store information about the device, such as Name, Class, LinkKey, LongTermKey, etc.

```
[root@linux-pc /var/lib/bluetooth]# cat 00:05:44:33:22:11/33:E0:33:67:98:12/info
[LongTermKey]
Key=27708DFD10B8C6FA085C637D04AEB13E
Authenticated=0
EncSize=16
EDiv=13763
Rand=1129395808884020994

[ConnectionParameters]
MinInterval=12
MaxInterval=12
Latency=14
Timeout=100

[General]
Name=REALTEK_RCU
Appearance=0x03c0
AddressType=public
SupportedTechnologies=LE;
Trusted=false
Blocked=false
Services=00001800-0000-1000-8000-00805f9b34fb;0000180a-0000-1000-8000-00805f9b34fb;00
00180f-0000-1000-8000-00805f9b34fb;00001812-0000-1000-8000-00805f9b34fb;00001813-000
0-1000-8000-00805f9b34fb;0000ffd0-0000-1000-8000-00805f9b34fb;1800;180a;180f;1812;1813
;ffd0;

[DeviceID]
Source=1
Vendor=93
Product=1
Version=8485
```

4.2. UART hciattach

The hciattach tool is an initialization tool provided by BlueZ for the UART interface Bluetooth controller.

For Realtek Bluetooth UART, please use rtk_hciattach generated by the source code compiled in the driver package rtk_hciattach/ directory. Do not use the hciattach compiled by BlueZ. Initialization is performed by **rtk_hciattach -n -s 115200 ttyS1 rtk_h5** or

rtk_hciattach -n -s 115200 ttyS1 rtk_h4 commands, where the serial device name will be different on each platform.

4.3. Load firmware

The Realtek Bluetooth initialization needs to load the firmware file. For the Bluetooth UART interface, firmware position is defined in hciattach_rtk.c.

```
#define FIRMWARE_DIRECTORY "/system/etc/firmware/rtlbt/"
#define BT_CONFIG_DIRECTORY "/system/etc/firmware/rtlbt/"
```

For the Bluetooth module of the USB interface, driver obtains the firmware through the function request_firmware(). The path that the function looks for the firmware is related to the platform. In general, this directory is /lib/firmware. Sometimes it may be /user/firmware. Customers need to confirm the location and copy the firmware and config file to the directory.

4.4. BlueZ start-up

bluetoothd is the BlueZ central daemon that implements profiles such as A2DP, AVRCP, GATT, SDP and provides various D-Bus services for external applications.
The following steps are required to start bluetoothd:

1. Verify the bluetooth.conf has been placed in the /etc/dbus-1/system.d/.

If bluetoothd fails to start and the following log appears:

```
D-Bus setup failed: Connection ":1.12" is not allowed to own the service "org.bluez" due to security policies in the configuration file
```

It may be a D-Bus permission issue. Please add the code below in /etc/dbus-1/system.d/bluetooth.conf

```
<policy user="root">
        <allow own="org.bluez"/>
        <allow send_destination="org.bluez"/>
        <allow send_interface="org.bluez.Agent1"/>
        <allow send_interface="org.bluez.MediaEndpoint1"/>
        <allow send_interface="org.bluez.MediaPlayer1"/>
        <allow send_interface="org.bluez.ThermometerWatcher1"/>
        <allow send_interface="org.bluez.AlertAgent1"/>
        <allow send_interface="org.bluez.Profile1"/>
        <allow send_interface="org.bluez.HeartRateWatcher1"/>
        <allow send_interface="org.bluez.CyclingSpeedWatcher1"/>
        <allow send_interface="org.bluez.GattCharacteristic1"/>
        <allow send_interface="org.bluez.GattDescriptor1"/>
        <allow send_interface="org.freedesktop.DBus.ObjectManager"/>
        <allow send_interface="org.freedesktop.DBus.Properties"/>
```

```
    <!-- for bluez 4 -->
    <allow send_interface="org.bluez.Agent"/>
    <allow send_interface="org.bluez.HandsfreeAgent"/>
    <allow send_interface="org.bluez.MediaEndpoint"/>
    <allow send_interface="org.bluez.MediaPlayer"/>
    <allow send_interface="org.bluez.Watcher"/>
    <allow send_interface="org.bluez.ThermometerWatcher"/>
    <allow send_type="method_call"/>
  </policy>
```

NOTICE: Please run bluetoothd under root privilege.

2. If the interface is Bluetooth UART, please pull high #BT_DIS pin first, then complete initialization through rtk_hciattach

```
rtk_hciattach -n -s 115200 ttyUSB0 rtk_h5
```

3. Power on Bluetooth: Run hciconfig hci0 up. You can also run bluetoothctl and type **power on** command in its command-line context.

4. You need to run the shell command rfkill list to view the Bluetooth state, and unblock Bluetooth through rfkill unblock bluetooth if it is Blocked: yes.

5. Run the Bluetooth daemon

```
bluetoothd -n -d
```

You can add -C option to support deprecated command line interfaces, such as SDP interface for sdptool browse local.

5. BlueZ test and debug

After porting BlueZ, you can use some of the test tools and scripts to test and debug.

5.1. Check and change adapter settings

hciconfig -a command can display some basic information of the Controller. The **show** command in bluetoothctl can also display the Controller information. For more details about bluetoothctl, go to **5.2 Discovery, pairing and connection**.

You can set the Name, Class and other information in BlueZ's main.conf (/etc/bluetooth/main.conf). If you want to modify the appearance of the Controller on a remote device, such as a mobile phone, you can modify the Class. For example, to display as a headset, you can modify the Class to 0x210404. In bluetoothd, only the major and minor device class bits are considered. Major Service classes are determined by the SDP services that customer registers. Meanwhile, please comment the code in the functions update_name() and update_class() in plugins/hostname.c.

5.2. Discovery, pairing and connection

BlueZ provides a command line tool, bluetoothctl, which implements GAP operations. This program is located in the <bluez-5.xx>/client directory. Please run bluetoothctl under root privilege.

```
$ sudo bluetoothctl
[bluetooth]# show //Show the Power state: yes or no. If it is no, run power on
[bluetooth]# power on
[bluetooth]# agent NoInputNoOutput //Or set other IO capabilities, such as KeyboardDisplay
[bluetooth]# default-agent
[bluetooth]# scan on //After detecting the expected device, run scan off to stop scan。
[bluetooth]# pair 00:22:48:DC:89:0F //Initiate pairing with the remote device
[bluetooth]# connect 00:22:48:DC:89:0F //Connect to the remote device
```

After pairing, you can use **trust 00:22:48:DC:89:0F** in bluetoothctl to set the paired device trusted. Otherwise when requesting service authorization, the user is required to enter Yes/No. And an error will occur and connection will eventually be disconnected if there is no agent.

5.3. Discoverable and Connectable

If you need device to be discoverable to other devices, you need to open Inquiry Scan.
If you want to make device connectable, you need to open Page Scan.

- Inquiry and Page Scan: hciconfig hci0 piscan
- Inquiry Scan only: hciconfig hci0 iscan
- Page Scan only: hciconfig hci0 pscan
- No scan: hciconfig hci0 noscan

For BLE device discoverable, device should be advertising by hciconfig hci0 leadv <type>
The type is advertising type:

- 0x00: Connectable undirected advertising (ADV_IND)
- 0x01: Connectable high duty cycle directed advertising (ADV_DIRECT_IND)
- 0x02: Scannable undirected advertising (ADV_SCAN_IND)
- 0x03: Non connectable undirected advertising (ADV_NONCONN_IND)
- 0x04: Connectable low duty cycle directed advertising (ADV_DIRECT_IND)

The bluetoothctl can also be used to open the device discoverable and connectable mode. Executing the command **discoverable on** in bluetoothctl can enable discoverable and connectable mode simultaneously.
The default is limited discoverable mode and the period of time is 180s. If you want to keep device in general discoverable mode, set the DiscoverableTimeout in main.conf to 0. LE advertising is not open when device is in discoverable mode. If you want to turn on the advertising, you can use the above shell command (hciconfig hci0 leadv <type>) or you can use hcitool cmd to set advertising parameters, advertising data and start advertising.

5.4. Test tools

BlueZ has provided some useful tools to test the basic functions. The source code is in the <bluez-5.xx>/tools/ directory. Before testing BlueZ, first make sure that the bluetoothd process is working properly and the Bluetooth module has been successfully brought up.

Verify the Bluetooth module with hciconfig -a

```
[alex_lu@linux-pc /tmp]$ sudo hciconfig -a
hci1:Type: BR/EDR   Bus: UART
        BD Address: 34:C3:D2:0E:AE:36   ACL MTU: 1021:8   SCO MTU: 255:16
        UP RUNNING PSCAN ISCAN
        RX bytes:11078 acl:62 sco:0 events:153 errors:0
        TX bytes:9582 acl:63 sco:0 commands:75 errors:0
        Features: 0xff 0xff 0xff 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH HOLD SNIFF PARK
        Link mode: SLAVE ACCEPT
        Name: 'luke-INP-131LT-MMTBKCN'
        Class: 0x0c010c
        Service Classes: Rendering, Capturing
        Device Class: Computer, Laptop
        HCI Version: 4.0 (0x6)   Revision: 0x1e6c
        LMP Version: 4.0 (0x6)   Subversion: 0xa747
        Manufacturer: Realtek Semiconductor Corporation (93)
```

(a) hciconfig: HCI Configuration tool.

hciconfig hci0 up   /* Open and initialize HCI device */

hciconfig hci0 iscan /* make device visible */

Use command hciconfig --help or view <bluez-5.xx>/tools/hciconfig,c to get more information.

```
[alex_lu@linux-pc /tmp]$ sudo hciconfig -h
Password:
hciconfig - HCI device configuration utility
Usage:
        hciconfig
        hciconfig [-a] hciX [command ...]
Commands:
        up                          Open and initialize HCI device
        down                        Close HCI device
        reset                       Reset HCI device
        rstat                       Reset statistic counters
        auth                        Enable Authentication
        noauth                      Disable Authentication
        encrypt                     Enable Encryption
        noencrypt                   Disable Encryption
        piscan                      Enable Page and Inquiry scan
        noscan                      Disable scan
        iscan                       Enable Inquiry scan
        pscan                       Enable Page scan
        ptype        [type]         Get/Set default packet type
        lm           [mode]         Get/Set default link mode
        lp           [policy]       Get/Set default link policy
        name         [name]         Get/Set local name
        class        [class]    Get/Set class of device
        voice        [voice]        Get/Set voice setting
        iac          [iac]          Get/Set inquiry access code
        inqtpl       [level]    Get/Set inquiry transmit power level
        inqmode      [mode]         Get/Set inquiry mode
        inqdata      [data]         Get/Set inquiry data
        inqtype      [type]         Get/Set inquiry scan type
        inqparms     [win:int]      Get/Set inquiry scan window and interval
        pageparms    [win:int]      Get/Set page scan window and interval
        ...
```

(b) hcitool: HCI tools

$ sudo hcitool cmd <ogf> <ocf> <parameter1> <parameter2> ... /* Send HCI command */

$ sudo hcitool scan /* Scan other bluetooth device */.

$ sudo hcitool lescan /* Scan other LE bluetooth device */.

Use command hcitool --help or view <bluez-5.xx>/tools/hcitool,c file to get more information.

(c) l2ping: L2CAP test command

l2ping <address>

```
[alex_lu@linux-pc /tmp]$ sudo l2ping 9c:fb:d5:87:03:a3
Ping: 9c:fb:d5:87:03:a3 from 34:C3:D2:0E:AE:36 (data size 44) ...
44 bytes from 9c:fb:d5:87:03:a3 id 0 time 6.02ms
44 bytes from 9c:fb:d5:87:03:a3 id 1 time 17.46ms
44 bytes from 9c:fb:d5:87:03:a3 id 2 time 26.25ms
```

5.5. Test scripts

BlueZ provides D-Bus and socket interfaces for upper application programming. There are some examples in <bluez-5.xx>/test that show how to utilize the interfaces.
Usually the python scripts are based on D-Bus interfaces and C programming applications uses sockets. You can refer to the tests to achieve upper application development.

5.6. Service discovery

If you want to view a profile supported locally or remotely, you can use sdptool to implement it.
View the locally supported profiles: sdptool browse local
NOTICE: You should run bluetoothd with -C option to support this command.

```
[alex_lu@linux-pc /tmp]$ sudo sdptool browse local
Browsing FF:FF:FF:00:00:00 ...
Service RecHandle: 0x10000
Service Class ID List:
  "PnP Information" (0x1200)
Profile Descriptor List:
  "PnP Information" (0x1200)
    Version: 0x0103

Browsing FF:FF:FF:00:00:00 ...
Service Search failed: Invalid argument
Service Name: Generic Access Profile
Service Provider: BlueZ
Service RecHandle: 0x10001
Service Class ID List:
  "Generic Access" (0x1800)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 31
  "ATT" (0x0007)
    uint16: 0x0001
    uint16: 0x0005

Service Name: Generic Attribute Profile
Service Provider: BlueZ
Service RecHandle: 0x10002
Service Class ID List:
  "Generic Attribute" (0x1801)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 31
  "ATT" (0x0007)
    uint16: 0x0006
    uint16: 0x0009

Service Name: AVRCP CT
Service RecHandle: 0x10003
Service Class ID List:
  "AV Remote" (0x110e)
  "AV Remote Controller" (0x110f)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 23
  "AVCTP" (0x0017)
    uint16: 0x0103
Profile Descriptor List:
  "AV Remote" (0x110e)
    Version: 0x0106

Service Name: AVRCP TG
Service RecHandle: 0x10004
Service Class ID List:
  "AV Remote Target" (0x110c)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 23
  "AVCTP" (0x0017)
    uint16: 0x0103
Profile Descriptor List:
    "AV Remote" (0x110e)
        Version: 0x0105

Service Name: Audio Source
Service RecHandle: 0x10005
Service Class ID List:
    "Audio Source" (0x110a)
Protocol Descriptor List:
    "L2CAP" (0x0100)
        PSM: 25
    "AVDTP" (0x0019)
        uint16: 0x0103
Profile Descriptor List:
    "Advanced Audio" (0x110d)
```

The command sdptool browse <address> can query supported profile on remote device.

6. Profile application

BlueZ provides D-Bus API, socket interface and static/shared library for upper-layer application calls. Users can develop applications based on this, or directly integrate with some open source libraries and programs.

Some Linux based systems, such as the Ubuntu/Debian distribution, already come with BlueZ and applications and UI, and you can do basic tests on these systems to understand how to use Bluetooth and application.

6.1. A2DP and HFP

BlueZ implements the a2dp protocol layer, but customers need to develop applications to encode/decode audio data and output/input. BlueALSA and PulseAudio support this part of the function and they are available at https://github.com/Arkq/bluez-Alsa and http://pulseaudio.org. Download the source code and port it to the Linux system.
The a2dp audio related D-Bus interface provided by BlueZ can be viewed in doc/media-api.txt.

After BlueZ 5.0, HFP acts as an external profile and needs to be implemented by an external application. Bluetoothd is only responsible for managing the rfcomm link. Open source applications that implement HFP are blueALSA and oFono. BlueALSA will be introduced later. Although oFono also implements HFP, the audio part is handled by PulseAudio, so it also needs to be used in conjunction with PulseAudio to implement HFP.

HFP's audio data can be passed to the Bluetooth controller via the HCI or PCM interface. Realtek provides a Config file to determine the path.

6.1.1.blueALSA

This project is a rebirth of a direct integration between Bluez and ALSA. Since Bluez >= 5, the build-in integration has been removed in favor of 3rd party audio applications. From now on, Bluez acts as a middleware between an audio application, which implements Bluetooth audio profile, and a Bluetooth audio device.
The current status quo is, that in order to stream audio from/to a Bluetooth device, one has to install PulseAudio, or use Bluez < 5. However, Bluez version 4 is considered to be deprecated, so the only reasonable way to achieve this goal is to install PulseAudio.
With this application (later named as BlueALSA), one can achieve the same goal as with

PulseAudio, but with less dependencies and more bare-metal-like. BlueALSA registers all known Bluetooth audio profiles in Bluez, so in theory every Bluetooth device (with audio capabilities) can be connected. In order to access the audio stream, one has to connect to the ALSA PCM device called bluealsa. The device is based on the ALSA software PCM plugin.

1) Dependencies

- Alsa-lib
- Bluez-5+
- Glib with GIO support
- sbc

2) Compilation and Installation

Download bluez-alsa source code：

```
$ git clone https://github.com/Arkq/bluez-alsa.git
$ autoreconf --install
$ mkdir build && cd build
$ ../configure --enable-aac --enable-debug
$ make && make install
```

The bluealsa daemon is similar to bluetoothd running in mainloop. When bluetoothd is running, it will provide some services through D-Bus: one of them is RegisterProfile(). When bluealsa is initialized, it registers three profiles a2dp-source, hsp-ag, hfp-ag by default.

If you want to change the profile, you can use the -p parameter. Such as sudo ./src/bluealsa -p a2dp-source -p a2dp-sink -p hfp-hf ... It can also be achieved by modifying the struct ba_config config of src/bluealsa.c.

```
[alex_lu@linux-pc ~/bluez-alsa]$ sudo ./src/bluealsa
Password:
./src/bluealsa: ctl.c:548: Starting controller loop
./src/bluealsa: bluez.c:680: Registering endpoint: /A2DP/SBC/Source/1
./src/bluealsa: bluez.c:914: Registering profile: /HSP/AudioGateway
./src/bluealsa: bluez.c:914: Registering profile: /HFP/AudioGateway
./src/bluealsa: main.c:281: Starting main dispatching loop
```

3) A2DP Usage

a) A2DP Sink

In terminal:

```
 $ sudo bluealsa-aplay 00:00:00:00:00
```

In bluetoothctl

```
 [bluetooth]# connect xx:xx:xx:xx:xx
```

b) A2DP Source

```
 $ sudo aplay –D bluealsa:HCI=hci0,DEV=xx:xx:xx:xx:xx:xx,PROFILE=a2dp xxx.wav
```

4) HFP

a) Hands-free

Connect to a mobile phone in bluetoothctl:

```
$ connect xx:xx:xx:xx:xx
```

After a call is answered on the phone, run in the terminal:

```
$ sudo bluealsa-aplay --profile-sco 00:00:00:00:00
```

At this point, you can hear the voice of the other party on the device.

At the same time, you can use the following command to play an 8k 16-bit wav file. The other party can hear the sound.

```
$ sudo aplay –D bluealsa:HCI=hci0,DEV=xx:xx:xx:xx:xx:xx,PROFILE=sco xxx.wav
```

b) audio gateway

Connect to headphones or HFP-enabled speakers, and no need to make a call. (Because the device may not have a phone feature)

First, connect headphones or speakers in bluetoothctl.

Then in terminal, run the following command:

```
$ sudo bluealsa-aplay --profile-sco 00:00:00:00:00
```

Speak into the headset and hear the voice on the local device.

At the same time, you can use the following command to play an 8k 16-bit wav file and the other party can hear the sound.

```
$ sudo aplay –D bluealsa:HCI=hci0,DEV=xx:xx:xx:xx:xx:xx,PROFILE=sco xxx.wav
```

Note: In platforms without a PCM interface, such as a Linux PC, please ensure that the configuration in the Config file is f4 00 01 00

6.1.2.PulseAudio

a. (Cross) compile

First you need to port the dependent libraries related with PulseAudio, such as intltool, libtool, json-c, libsndfile, sbc. If you test under PC Linux(Ubuntu or Gentoo or etc), first confirm whether these packages are installed.

Configure PulseAudio. The options (for example) are as follows:

```
./configure --with-gnu-ld \
        --enable-shared \
        --enable-dbus \
        --disable-static \
        --disable-x11 \
        --disable-oss-output \
        --disable-oss-wrapper \
        --disable-coreaudio-output \
        --disable-esound \
        --disable-solaris \
        --disable-waveout \
        --disable-gtk3 \
        --disable-gconf \
        --disable-jack \
        --disable-asyncns \
        --disable-tcpwrap \
        --disable-lirc \
        --disable-hal-compat \
        --disable-openssl \
        --disable-xen \
        --disable-systemd \
        --disable-systemd-journal \
        --disable-manpages \
        --disable-per-user-esound-socket \
        --disable-neon-opt \
        --disable-atomic-arm-linux-helpers \
        --disable-ipv6 \
        --disable-glib2 \
        --without-speex \
        --disable-systemd-daemon \
        --disable-systemd-login \
        --disable-systemd-journal \
        --with-system-user=root \
        --with-system-group=root \
        --with-access-group=root \
        --prefix=/usr \
        --sysconfdir=/etc \
        --localstatedir=/var \
        --libdir=/lib \
        --datarootdir=/usr/share \
        --disable-bluez4 \
        --enable-bluez5 \
        --enable-alsa \
        --enable-udev \
        --enable-bluez5-native-headset \
        --disable-bluez5-ofono-headset
```

If the platform is an embedded platform, you need to add **--host=xxx-linux** according to the CPU Type, such as **arm-linux** for ARM**, mips-linux** for MIPS and so on**.**
If the platform does not have udev, change **--enable-udev** to **--disable-udev**
If you don't need to support ALSA, change **--enable-alsa** to --**disable-alsa**
Please set the LIBJSON_CFLAGS, LIBJSON_LIBS, LIBSNDFILE_CFLAGS, LIBSNDFILE_LIBS correctly before compiling PulseAudio (For PC Linux, you don't need to care about these environments. In general, they are already set)

b. Run PulseAudio

Before running pulseaudio, you need to copy some configuration files to the system, such as daemon.conf, client.conf, default.pa, system.pa. Typically on Linux PC, these configuration files are copied to the /etc/pulse/ directory. This directory is specified by configure --sysconfdir. For embedded systems, the target directory may be changed.
The starting pulseaudio shell command:

```
$ pulseaudio --start
```

The pulseaudio runtime can also be loaded with --log-level=X, --dl-search-path=<modules where the path>, --file=<The configuration file with pa suffix>. For example, the path of the module is /lib/pulse-x.x/modules, and the configuration file with pa suffix is /etc/pulse/default.pa or /etc/pulse/system.pa

```
$ pulseaudio -n --log-level=3 --dl-search-path=/lib/pulse-7.1/modules --file=/etc/pulse/default.pa
```

or

```
$ pulseaudio -n --log-level=3 --dl-search-path=/lib/pulse-7.1/modules --file=/etc/pulse/system.pa
--system
```

-n means not to load the default file default.pa or system.pa.
The path to the pulseaudio configuration file is determined by the --sysconfdir parameter of configure. For example, --sysconfdir=/etc, the path of the pulseaudio configuration file is /etc/pulse/
The modules path is determined by the --libdir parameter of configure. For example, --libdir=/usr/lib, the path of modules is /usr/lib/pulse-x.y/modules/, where x.y is the main and minor version number of pulseaudio, such as 6.0, 7.1, etc.

The official recommendation is to run the pulseaudio daemon with normal user rights. However, the general embedded platform will run PulseAudio as root. Run pulseaudio as root, the running mode is system-wide, and the parameter --system is passed in runtime.

When debugging PulseAudio, you may need to set different log levels according to the debugging requirements:

- 0: error
- 1: error + warn
- 2: error + warn + info
- 3: error + warn + info + debug

It is recommended to open “error + warn + info” log at the beginning.

If a similar error occurs while starting pulseaudio daemon,

```
main.c: Failed to acquire org.pulseaudio.Server: org.freedesktop.DBus.Error.AccessDenied: Connection ":1.46" is not allowed to own the service "org.pulseaudio.Server" due to security policies in the configuration file
```

, add the following content in /etc/dbus-1/system.d/pulseaudio-system.conf (this file comes from <pulseaudio-x.y>/src/daemon/pulseaudio-system.conf).

If you run pulseaudio as a normal user, you need to add a policy for the normal user in this file. The content is similar to the root user.

```
<!DOCTYPE busconfig PUBLIC
  "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
  "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>
        <policy user="root">
                <allow own="org.pulseaudio.Server"/>
                <allow send_destination="org.bluez"/>
                <allow send_interface="org.bluez.Manager"/>
        </policy>
        <policy user="pulse">
                <allow own="org.pulseaudio.Server"/>
                <allow send_destination="org.bluez"/>
                <allow send_interface="org.bluez.Manager"/>
        </policy>
        <policy context="default">
                <deny own="org.pulseaudio.Server"/>
                <deny send_destination="org.bluez"/>
                <deny send_interface="org.bluez.Manager"/>
        </policy>
</busconfig>
```

In order to discover bluetooth devices, module-bluetooth-policy and module-bluetooth-discover should be added to /etc/pulse/system.pa or /etc/pulse/default.pa

```
load-module module-bluetooth-policy
load-module module-bluetooth-discover
```

c. Test A2DP

Refer to **5.2 Discovery, pairing and connection** to connect Bluetooth headset.

Ensure that module-bluez5-discover and module-bluetooth-policy are loaded

```
# pactl list modules
```

Check if BlueZ Card exists

```
# pactl list cards
```

```
Card #0
        Name: bluez_card.00_18_91_00_24_57
        Driver: module-bluez5-device.c
        Owner Module: 12
        Properties:
                device.description = "WOOWI HERO"
                device.string = "00:18:91:00:24:57"
                device.api = "bluez"
                device.class = "sound"
                device.bus = "bluetooth"
                device.form_factor = "headset"
                bluez.path = "/org/bluez/hci0/dev_00_18_91_00_24_57"
                bluez.class = "0x240404"
                bluez.alias = "WOOWI HERO"
                device.icon_name = "audio-headset-bluetooth"
                device.intended_roles = "phone"
        Profiles:
                headset_head_unit: Headset Head Unit (HSP/HFP) (sinks: 1, sources: 1, priority: 20,
available: yes)
                a2dp_sink: High Fidelity Playback (A2DP Sink) (sinks: 1, sources: 0, priority: 10,
available: no)
                off: Off (sinks: 0, sources: 0, priority: 0, available: yes)
        Active Profile: off
        Ports:
                headset-output: Headset (priority: 0, latency offset: 0 usec)
                        Part of profile(s): headset_head_unit, a2dp_sink
                headset-input: Headset (priority: 0, latency offset: 0 usec)
                        Part of profile(s): headset_head_unit
```

```
# pactl list sinks
```

```
Sink #1
        State: SUSPENDED
        Name: bluez_sink.00_18_91_00_24_57
        Description: WOOWI HERO
        Driver: module-bluez5-device.c
        Sample Specification: s16le 1ch 8000Hz
        Channel Map: mono
        Owner Module: 20
        Mute: no
        Volume: mono: 43691 /   67%
                        balance 0.00
        Base Volume: 65536 / 100%
        Monitor Source: bluez_sink.00_18_91_00_24_57.monitor
        Latency: 0 usec, configured 0 usec
        Flags: HARDWARE HW_VOLUME_CTRL LATENCY
```

```
	Properties:
		bluetooth.protocol = "headset_head_unit"
		device.intended_roles = "phone"
		device.description = "WOOWI HERO"
		device.string = "00:18:91:00:24:57"
		device.api = "bluez"
		device.class = "sound"
		device.bus = "bluetooth"
		device.form_factor = "headset"
		bluez.path = "/org/bluez/hci0/dev_00_18_91_00_24_57"
		bluez.class = "0x240404"
		bluez.alias = "WOOWI HERO"
		device.icon_name = "audio-headset-bluetooth"
	Ports:
		headset-output: Headset (priority: 0)
	Active Port: headset-output
	Formats:
		pcm
```

Set the profile to a2dp_sink. If "pactl list cards" shows that the BlueZ Card's Active Profile is a2dp_sink, then this operation is omitted.

```
$ pactl set-card-profile bluez_card.00_18_91_00_24_57 a2dp_sink
```

If "Failure: No such entity" appears, check if the connection is closed.

Load wav file

```
$ pactl upload-sample /usr/share/sounds/alsa/Front_Center.wav sp1
```

Play

```
$ pactl play-sample sp1 bluez_sink.00_18_91_00_24_57
```

6.2. AVRCP CT

The AVRCP CT-related D-Bus interface provided by BlueZ can be viewed in doc/media-api.txt with the interfaces org.bluez.MediaControl1 and org.bluez.MediaPlayer1. The implementation of org.bluez.MediaControl1 methods is in the profile/audio/control.c file. The control_play(), control_pause(), etc of this file is the concrete implementation of the D-Bus API such as Play, Pause. Users can add more operations as needed.

6.3. LE GATT Client

6.3.1. GATT Client API

BlueZ provides a number of useful GATT Client APIs for users to develop GATT Client applications.

```
struct bt_att *bt_att_new(int fd, bool ext_signed)
```

This function allocates a variable of type struct bt_att. This structure contains a socket descriptor indicating the connection of L2CAP. This connection channel is used to transmit GATT packets.

```
unsigned int bt_att_register_disconnect(struct bt_att *att,
                                bt_att_disconnect_func_t callback,
                                void *user_data,
                                bt_att_destroy_func_t destroy)
```

The function registers a callback that handles disconnection.

```
struct gatt_db *gatt_db_new(void)
```

This function allocates a GATT database to store the service information provided by the remote GATT server.

```
struct bt_gatt_client *bt_gatt_client_new(struct gatt_db *db,
                                        struct bt_att *att,
                                        uint16_t mtu)
```

The function allocates a GATT client instance.

```
unsigned int gatt_db_register(struct gatt_db *db,
                                gatt_db_attribute_cb_t service_added,
                                gatt_db_attribute_cb_t service_removed,
                                void *user_data,
                                gatt_db_destroy_func_t destroy)
```

This function registers three callback functions that capture server add event, server delete event, and database release event, respectively.

```
unsigned int bt_gatt_client_ready_register(struct bt_gatt_client *client,
                                bt_gatt_client_callback_t callback,
                                void *user_data,
                                bt_gatt_client_destroy_func_t destroy)
```

This function registers a callback function that captures the GATT Client Ready event. The registered callback function is called when the searching services is completed. We can view the information of all services in this callback function, refer to the print_services() function of **tools/btgatt-client.c**

```
unsigned int bt_gatt_client_read_value(struct bt_gatt_client *client,
                                       uint16_t value_handle,
                                       bt_gatt_client_read_callback_t callback,
                                       void *user_data,
                                       bt_gatt_client_destroy_func_t destroy)
```

The function is used to read the value of the attribute pointed to by the handle.

```
unsigned int bt_gatt_client_write_value(struct bt_gatt_client *client,
                                        uint16_t value_handle,
                                        const uint8_t *value, uint16_t length,
                                        bt_gatt_client_callback_t callback,
                                        void *user_data,
                                        bt_gatt_client_destroy_func_t destroy)
```

This function is used to change the value of the attribute pointed to by the handle.

Please refer to **<bluez-5.xx>/tools/btgatt-client.c** for more information on the use of the GATT Client API.

6.3.2.btgatt-client

You can connect to remote LE device with btgatt-client and search service, read and write characteristics.

```
[alex_lu@linux-pc ~/bluez/bluez-5.41]$ sudo ./tools/btgatt-client -d 00:E0:00:34:56:78
Connecting to device... Done
[GATT client]# Service Added - UUID: 00001800-0000-1000-8000-00805f9b34fb start: 0x0001
end: 0x0007
[GATT client]# Service Added - UUID: 00001812-0000-1000-8000-00805f9b34fb start: 0x0008
end: 0x0026
[GATT client]# Service Added - UUID: 00001801-0000-1000-8000-00805f9b34fb start: 0x0027
end: 0x002a
…
[GATT client]# GATT discovery procedures complete
[GATT client]#
service - start: 0x0001, end: 0x0007, type: primary, uuid:
00001800-0000-1000-8000-00805f9b34fb
        charac - start: 0x0002, value: 0x0003, props: 0x02, ext_props: 0x0000, uuid:
00002a00-0000-1000-8000-00805f9b34fb
        charac - start: 0x0004, value: 0x0005, props: 0x02, ext_props: 0x0000, uuid:
00002a01-0000-1000-8000-00805f9b34fb
        charac - start: 0x0006, value: 0x0007, props: 0x02, ext_props: 0x0000, uuid:
00002a04-0000-1000-8000-00805f9b34fb

service - start: 0x0008, end: 0x0026, type: primary, uuid:
```

```
00001812-0000-1000-8000-00805f9b34fb
        charac - start: 0x0009, value: 0x000a, props: 0x02, ext_props: 0x0000, uuid:
00002a4a-0000-1000-8000-00805f9b34fb
        charac - start: 0x000b, value: 0x000c, props: 0x04, ext_props: 0x0000, uuid:
00002a4c-0000-1000-8000-00805f9b34fb
        charac - start: 0x000d, value: 0x000e, props: 0x06, ext_props: 0x0000, uuid:
00002a4e-0000-1000-8000-00805f9b34fb
        charac - start: 0x000f, value: 0x0010, props: 0x02, ext_props: 0x0000, uuid:
00002a4b-0000-1000-8000-00805f9b34fb
        charac - start: 0x0011, value: 0x0012, props: 0x1a, ext_props: 0x0000, uuid:
00002a4d-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0013, uuid: 00002902-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0014, uuid: 00002908-0000-1000-8000-00805f9b34fb
        charac - start: 0x0015, value: 0x0016, props: 0x1a, ext_props: 0x0000, uuid:
00002a4d-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0017, uuid: 00002902-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0018, uuid: 00002908-0000-1000-8000-00805f9b34fb
        charac - start: 0x0019, value: 0x001a, props: 0x0e, ext_props: 0x0000, uuid:
00002a4d-0000-1000-8000-00805f9b34fb
                descr - handle: 0x001b, uuid: 00002908-0000-1000-8000-00805f9b34fb
        charac - start: 0x001c, value: 0x001d, props: 0x1a, ext_props: 0x0000, uuid:
00002a4d-0000-1000-8000-00805f9b34fb
                descr - handle: 0x001e, uuid: 00002902-0000-1000-8000-00805f9b34fb
                descr - handle: 0x001f, uuid: 00002908-0000-1000-8000-00805f9b34fb
        charac - start: 0x0020, value: 0x0021, props: 0x1a, ext_props: 0x0000, uuid:
00002a4d-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0022, uuid: 00002902-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0023, uuid: 00002908-0000-1000-8000-00805f9b34fb
        charac - start: 0x0024, value: 0x0025, props: 0x0e, ext_props: 0x0000, uuid:
00002a4d-0000-1000-8000-00805f9b34fb
                descr - handle: 0x0026, uuid: 00002908-0000-1000-8000-00805f9b34fb

service - start: 0x0027, end: 0x002a, type: primary, uuid:
00001801-0000-1000-8000-00805f9b34fb
        charac - start: 0x0028, value: 0x0029, props: 0x20, ext_props: 0x0000, uuid:
00002a05-0000-1000-8000-00805f9b34fb
                descr - handle: 0x002a, uuid: 00002902-0000-1000-8000-00805f9b34fb


…

[GATT client]# help
Commands:
    help                    Display help message
    services                Show discovered services
    read-value              Read a characteristic or descriptor value
    read-long-value         Read a long characteristic or desctriptor value
    read-multiple           Read Multiple
    write-value             Write a characteristic or descriptor value
    write-long-value    Write long characteristic or descriptor value
    write-prepare           Write prepare characteristic or descriptor value
    write-execute           Execute already prepared write
    register-notify     Subscribe to not/ind from a characteristic
    unregister-notify   Unregister a not/ind session
    set-security            Set security level on le connection
    get-security            Get security level on le connection
    set-sign-key            Set signing key for signed write command

[GATT client]# read-value 0x0003
Read value (11 bytes): 4d 65 64 69 61 51 20 52 43 55 00
```

6.3.3.gatttool

In addition to btgatt-client, BlueZ also provides another useful tool, gatttool, which is located in the <bluez-5.xx>/attrib/ directory and also provides the functions of searching services, reading and writing characteristics, and more.

```
[alex_lu@linux-pc ~/bluez/bluez-5.47]$ sudo ./attrib/gatttool --help-all
Password:
Usage:
  gatttool [OPTION...]

Help Options:
  -h, --help                              Show help options
  --help-all                              Show all help options
  --help-gatt                             Show all GATT commands
  --help-params                           Show all Primary Services/Characteristics
arguments
  --help-char-read-write                  Show all Characteristics Value/Descriptor
Read/Write arguments

GATT commands
  --primary                               Primary Service Discovery
  --characteristics                       Characteristics Discovery
  --char-read                             Characteristics Value/Descriptor Read
  --char-write                            Characteristics Value Write Without Response
(Write Command)
  --char-write-req                        Characteristics Value Write (Write Request)
```

```
    --char-desc                                       Characteristics Descriptor Discovery
    --listen                                          Listen for notifications and indications

Primary Services/Characteristics arguments
    -s, --start=0x0001                                Starting handle(optional)
    -e, --end=0xffff                                  Ending handle(optional)
    -u, --uuid=0x1801                                 UUID16 or UUID128(optional)

Characteristics Value/Descriptor Read/Write arguments
    -a, --handle=0x0001                               Read/Write characteristic by handle(required)
    -n, --value=0x0001                                Write characteristic value (required for write
operation)

Application Options:
    -i, --adapter=hciX                                Specify local adapter interface
    -b, --device=MAC                                  Specify remote Bluetooth address
    -t, --addr-type=[public | random]                 Set LE address type. Default: public
    -m, --mtu=MTU                                     Specify the MTU size
    -p, --psm=PSM                                     Specify the PSM for GATT/ATT over BR/EDR
    -l, --sec-level=[low | medium | high]             Set security level. Default: low
    -I, --interactive                                 Use interactive mode
```

Search for primary services.

```
gatttool sudo ./attrib/gatttool -b 00:E0:00:34:56:78 --primary
```

```
[alex_lu@linux-pc ~/bluez/bluez-5.47]$ sudo ./attrib/gatttool -b 00:E0:00:34:56:78 --primary
Password:
attr handle = 0x0001, end grp handle = 0x0007 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle = 0x0008, end grp handle = 0x0026 uuid: 00001812-0000-1000-8000-00805f9b34fb
attr handle = 0x0027, end grp handle = 0x002a uuid: 00001801-0000-1000-8000-00805f9b34fb
…
```

Search for characteristics.

```
gatttool -b <address> --characteristics
```

or

```
gatttool -b < address> --characteristics -s <start handle> -e <end handle>
```

```
[alex_lu@linux-pc ~/bluez/bluez-5.47]$ sudo ./attrib/gatttool -b 00:E0:00:34:56:78 --char-desc
handle = 0x0001, uuid = 00002800-0000-1000-8000-00805f9b34fb
handle = 0x0002, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0003, uuid = 00002a00-0000-1000-8000-00805f9b34fb
handle = 0x0004, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0005, uuid = 00002a01-0000-1000-8000-00805f9b34fb
handle = 0x0006, uuid = 00002803-0000-1000-8000-00805f9b34fb
handle = 0x0007, uuid = 00002a04-0000-1000-8000-00805f9b34fb
handle = 0x0008, uuid = 00002800-0000-1000-8000-00805f9b34fb
handle = 0x0009, uuid = 00002803-0000-1000-8000-00805f9b34fb
…

[alex_lu@linux-pc ~/bluez/bluez-5.47]$ sudo ./attrib/gatttool -b 00:E0:00:34:56:78
--characteristics
handle = 0x0002, char properties = 0x02, char value handle = 0x0003, uuid =
00002a00-0000-1000-8000-00805f9b34fb
handle = 0x0004, char properties = 0x02, char value handle = 0x0005, uuid =
00002a01-0000-1000-8000-00805f9b34fb
handle = 0x0006, char properties = 0x02, char value handle = 0x0007, uuid =
00002a04-0000-1000-8000-00805f9b34fb
handle = 0x0009, char properties = 0x02, char value handle = 0x000a, uuid =
00002a4a-0000-1000-8000-00805f9b34fb
handle = 0x000b, char properties = 0x04, char value handle = 0x000c, uuid =
00002a4c-0000-1000-8000-00805f9b34fb
handle = 0x000d, char properties = 0x06, char value handle = 0x000e, uuid =
00002a4e-0000-1000-8000-00805f9b34fb
handle = 0x000f, char properties = 0x02, char value handle = 0x0010, uuid =
00002a4b-0000-1000-8000-00805f9b34fb
handle = 0x0011, char properties = 0x1a, char value handle = 0x0012, uuid =
00002a4d-0000-1000-8000-00805f9b34fb
handle = 0x0015, char properties = 0x1a, char value handle = 0x0016, uuid =
00002a4d-0000-1000-8000-00805f9b34fb
…

[alex_lu@linux-pc ~/bluez/bluez-5.47]$ sudo ./attrib/gatttool -b 00:E0:00:34:56:78
--characteristics -s 0x0001 -e 0x0010
handle = 0x0002, char properties = 0x02, char value handle = 0x0003, uuid =
00002a00-0000-1000-8000-00805f9b34fb
handle = 0x0004, char properties = 0x02, char value handle = 0x0005, uuid =
00002a01-0000-1000-8000-00805f9b34fb
handle = 0x0006, char properties = 0x02, char value handle = 0x0007, uuid =
00002a04-0000-1000-8000-00805f9b34fb
handle = 0x0009, char properties = 0x02, char value handle = 0x000a, uuid =
00002a4a-0000-1000-8000-00805f9b34fb
handle = 0x000b, char properties = 0x04, char value handle = 0x000c, uuid =
00002a4c-0000-1000-8000-00805f9b34fb
handle = 0x000d, char properties = 0x06, char value handle = 0x000e, uuid =
00002a4e-0000-1000-8000-00805f9b34fb
handle = 0x000f, char properties = 0x02, char value handle = 0x0010, uuid =
00002a4b-0000-1000-8000-00805f9b34fb
```

6.4. LE GATT Server

6.4.1. GATT Server API

BlueZ provides some GATT Server APIs for users to develop GATT Server applications.

```
struct bt_gatt_server *bt_gatt_server_new(struct gatt_db *db,
                            struct bt_att *att, uint16_t mtu)
```

This function allocates a structure of type struct bt_gatt_server. This function also registers the processing functions of ATT request such as exchange mtu, read by group type, read by type and so on.

```
struct gatt_db_attribute *gatt_db_add_service(struct gatt_db *db,
                            const bt_uuid_t *uuid,
                            bool primary,
                            uint16_t num_handles)
```

Add a service, after which you need to call other functions to add characteristics and descriptors.

```
struct gatt_db_attribute *
gatt_db_service_add_characteristic(struct gatt_db_attribute *attrib,
                            const bt_uuid_t *uuid,
                            uint32_t permissions,
                            uint8_t properties,
                            gatt_db_read_t read_func,
                            gatt_db_write_t write_func,
                            void *user_data)
```

Add a characteristic to the service pointed to by attrib.

```
struct gatt_db_attribute *
gatt_db_service_add_descriptor(struct gatt_db_attribute *attrib,
                            const bt_uuid_t *uuid,
                            uint32_t permissions,
                            gatt_db_read_t read_func,
                            gatt_db_write_t write_func,
                            void *user_data)
```

Add a descriptor to the service pointed to by attrib.

```
bool gatt_db_attribute_write(struct gatt_db_attribute *attrib, uint16_t offset,
                            const uint8_t *value, size_t len,
                            uint8_t opcode, struct bt_att *att,
                            gatt_db_attribute_write_t func,
                            void *user_data)
```

Set the value of the attribute. This function is generally called without setting attribute write callback.

bool gatt_db_service_set_active(struct gatt_db_attribute *attrib, bool active)

Activate the service.

Please refer to **<bluez-5.xx>/tools/btgatt-server.c** for more information on the use of the GATT Server API.

6.4.2.btgatt-server

The development of GATT server can refer to <bluez-5.xx>/tools/btgatt-server.c.

6.4.3.Built-in GATT server

The builtin GATT server can refer to rtk_gatt_server.c.

7. Debug tool

7.1. hcidump

BlueZ has a tool hcidump that can record the Bluetooth data exchanged between the host stack and the Bluetooth controller. This tool is located at <bluez-5.xx>/tools/.
The hcidump tool can print the log to the terminal or save it to the file as raw data. The saved file can be parsed with PC tools.

Print directly to terminal:

```
$ sudo hcidump -t
```

Save the text log to a txt file:

```
$ sudo hcidump -t > xxx.txt
```

Save raw data:

```
$ sudo hcidump -w xxx.cfa
```

7.2. BlueZ log

When running bluetoothd, add the **-d -n** parameter to open the BlueZ debug log (the log printed in the source code using the DBG() function will be output).
BlueZ's log is saved to syslog at the same time. In general, the Ubuntu system log will be stored in /var/log/syslog file.

```
void btd_debug(const char *format, ...)
{
        va_list ap;

        va_start(ap, format);
        vsyslog(LOG_DEBUG, format, ap);
        va_end(ap);

        …
}
```

7.3. Kernel log

Modify the printk level with echo 7 > /proc/sys/kernel/printk to print the log of net/bluetooth and Realtek Bluetooth driver.

8. List of Figures