Qualcomm Technologies, Inc.

Qualcomm Linux Bluetooth Guide

80-70018-13 AA

April 2, 2025

Contents

1 Bluetooth overview

The Bluetooth® wireless technology is a short-range communications system that facilitates wireless exchange of data between devices. The key advantages of Bluetooth technology are as follows:

- Replaces the cables connecting portable and fixed electronic devices.
- Provides robust, power efficient, and cost-effective solutions.
- Facilitates flexibility of solutions and their applications.

The Bluetooth technology provides the following radio options.

Radio	Description
Basic Rate/Enhanced Data Rate (BR/EDR)	• Offers optional EDR, alternate media access control (MAC), and physical layer (PHY) extensions. • Supports synchronous and asynchronous connections with a data rate of 721.2 Kbps for BR and 2.1 Mbps for EDR.
Low Energy	• Enables products that require lower current consumption, lower complexity, and lower cost than BR/EDR. • Facilitates use cases and applications with lower data rates and has lower duty cycles.

BR/EDR and Low Energy radio options enable device discovery, connection establishment, and connection mechanisms. The optimal choice of radio depends on the use case or application of a solution.

The Qualcomm® Linux® supports Bluetooth solutions for the following kits.

Kit	Hardware system-on-chip (SoC)	Bluetooth chipset	Bluetooth stack
• Qualcomm RB3 Gen 2 Vision Development Kit • Qualcomm RB3 Gen 2 Core Development Kit	• QCS6490	• WCN6750 • WCN6856	• BlueZ
• Qualcomm RB3 Gen 2 Lite Vision Development Kit • Qualcomm RB3 Gen 2 Lite Core Development Kit	• QCS5430	• WCN6750 • WCN6856	• BlueZ
• Qualcomm Dragonwing™ IQ-9075 Evaluation Kit (EVK)	• QCS9075	• QCA6698AQ	• BlueZ
• Qualcomm IQ-8 Beta EVK	• QCS8275	• QCA6698AQ	• BlueZ

The following figure shows the various components of the Qualcomm connectivity chipset, including the Bluetooth subsystem and its interaction with the application processor.

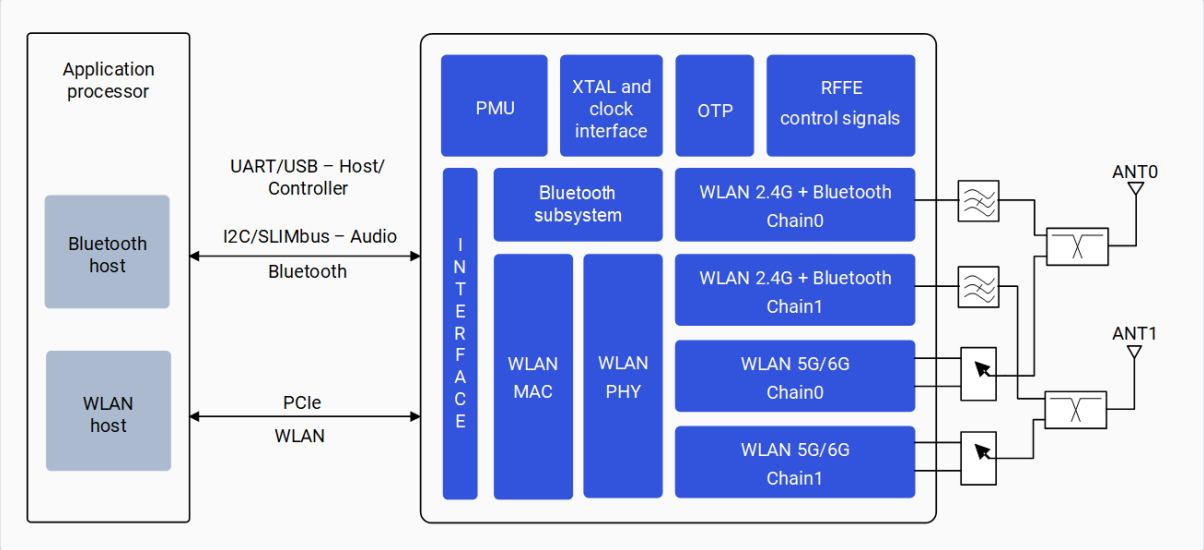


Figure : Qualcomm connectivity chipset block diagram

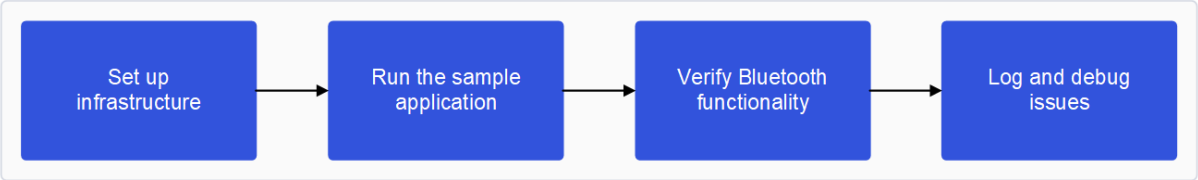
2 Get started with Bluetooth functionality

The Qualcomm Linux Bluetooth solution includes the BlueZ stack and sample test applications. These applications interact with the Bluetooth daemon of the BlueZ stack to run Bluetooth functions.

Qualcomm Linux supports the following sample applications for the BlueZ stack:

- bluetoothctl
- obexctl
- ofono
- evtool

The following workflow shows how to get started with verifying the Bluetooth functionality using a sample application.



The sequence to complete the workflow is as follows:

1. Set up your infrastructure as described in the *Qualcomm Linux Build Guide* and *Qualcomm Linux Yocto Guide*. The *Qualcomm Linux Build Guide* also provides information about the common build workflows.
2. Run the sample application and verify the Bluetooth functionality as described in Verify functionality of BlueZ stack.
3. If any issues occur, log and debug them as described in Debug.

2.1 Set the Bluetooth MAC address

By default, the factory sets the Bluetooth MAC address in the one-time programmable memory. If you want to set the Bluetooth MAC address manually, complete the following procedure.

Note: A manually set Bluetooth MAC address doesn't persist across device reboots.

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Switch off Bluetooth on the device.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth management tool and set the Bluetooth address by running the following command:

   ```
   btmgmt public-addr <bt_address>
   ```

   For example, to set the Bluetooth address of the device as `22:22:9B:2C:79:1E`, run the following command:

   ```
   btmgmt public-addr 22:22:9B:2C:79:1E
   ```

   Sample output

   ```
   sh-5.1# btmgmt public-addr 22:22:9B:2C:79:1E
   hci0 Set Public Address complete, options:
   sh-5.1# hciconfig
   hci0:     Type: Primary Bus: UART
             BD Address: 22:22:9B:2C:79:1E ACL MTU: 1024:7 SCO MTU:
   240:8
             DOWN
             RX bytes: 7763 acl:0 sco:0 events:364 errors:0
             TX bytes: 938685 acl:0 sco:0 commands:4004 errors:0
   ```

2.2 Source code location

The following table lists the source code location of Bluetooth components such as the stack, sample test applications, and drivers.

Component	Source code location
bluetoothctl	https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/tools
obexctl	https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/tools
ofono	https://git.kernel.org/pub/scm/network/ofono/ofono.git
BlueZ stack	• The git repository for BlueZ is at: https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/ • You can download the stack at: https://mirrors.edge.kernel.org/pub/linux/bluetooth/bluez-5.65.tar.gz • The git repository for the Bluetooth kernel subsystem is at: – https://git.kernel.org/pub/scm/linux/kernel/git/bluetooth/bluetooth.git/tree/ – https://git.kernel.org/pub/scm/linux/kernel/git/bluetooth/bluetooth-next.git/tree/
Bluetooth driver	You can download the Bluetooth driver from the following locations: • https://git.kernel.org/pub/scm/linux/kernel/git/bluetooth/bluetooth-next.git/tree/drivers/bluetooth/hci_qca.c • https://git.kernel.org/pub/scm/linux/kernel/git/bluetooth/bluetooth-next.git/tree/drivers/bluetooth/btqca.c

3 Bluetooth features

The WCN6750, WCN6856, and QCA6698AQ connectivity chipsets comply with the *Bluetooth Core specification v5.2*. These chipsets have the following controller features and capabilities:

- A Bluetooth Low Energy data rate of up to 2 Mbps
- A long-range mode with better sensitivity at two new lower bit rates, 500 Kbps and 125 Kbps
- An 8x improvement in broadcast capability with the use of advertising extensions
- An improved channel selection algorithm (CSA #2), which facilitates improved channel coordination and coexistence efficiency with other Bluetooth and non-Bluetooth traffic

The following table lists the Bluetooth specifications and features supported by the WCN6750, WCN6856, and QCA6698AQ connectivity chipsets.

Feature	**WCN6750**	**WCN6856**	**QCA6698AQ**
Bluetooth Low Energy secure connections	✓	✓	✓
Bluetooth Low Energy privacy 1.2	✓	✓	✓
Bluetooth 5.2 Core specification	✓	✓	✓
Data length extensions	✓	✓	✓
2 Mbps PHY	✓	✓	✓
Advertising extensions	✓	✓	✓

Feature	WCN6750	WCN6856	QCA6698AQ
Bluetooth Low Energy long range	✓	✓	✓
CSA #2	✓	✓	✓
USB 1.1 interface support	×	✓	✓

3.1 BlueZ stack

BlueZ is a Bluetooth Linux stack that supports the core Bluetooth layers and protocols. BlueZ features are as follows:

- Complies with *Bluetooth Core specification v5.2*
- Supports multiple Bluetooth devices
- Implements a socket interface for all layers
- Supports multithreaded data processing

3.2 Bluetooth profiles and roles

The Bluetooth profiles define the specifications, requirements, and roles of devices to establish a Bluetooth connection. A profile also determines the communication protocol used between devices.

The BlueZ stack supports the following Bluetooth profiles and roles for connectivity chipsets.

Profile	Role	Version	Support on chipset	
			WCN6750/ WCN6856	QCA6698AQ
General Access Profile (GAP)	Central and peripheral		✓	✓
Serial Port Profile (SPP)	Client and server	v1.2	✓	✓

Profile	Role	Version	Support on chipset	
			WCN6750/ WCN6856	**QCA6698AQ**
Human interface device (HID) over GATT Profile (HOGP)	Host	v1	✓	✓
General Attribute Profile (GATT)	Central and peripheral		✓	✓
Advanced Audio Distribution Profile (A2DP)	Source	Nonsplit v1.3	✓	×
	Sink	Nonsplit v1.3	✓	×
Audio/Video Remote Control Profile (AVRCP)	Target	v1.5	✓	×
	Controller	v1.5	✓	×
Hands-Free Profile (HFP)	Audio gateway	Nonsplit v1.7	✓	×
	Client	Nonsplit v1.7	✓	×
Object Push Profile (OPP)	Client and server	v1.2	✓	✓
File Transfer Protocol (FTP)	Client and server	v1.2	✓	✓
Phone Book Access Profile (PBAP)	Client and server	v1.1	✓	✓
Message Access Profile (MAP)	Client and server	v1.2	✓	✓

Note: Bluetooth profile concurrency is supported.

Profile role	Description

General Access Profile

Bluetooth Low Energy GAP is an extension of the existing BR/EDR GAP. It handles the following basic operations of a device:

- Discover a device
- Establish a connection
- Bond multiple devices
- Establish a private connection
- Resolve private addresses

All Bluetooth devices must implement a basic level of functionality that GAP defines. GAP ensures that all Bluetooth devices can establish baseband connections, regardless of the higher-level functionality they support. GAP is responsible for the following functions:

- Generic procedures for discovering Bluetooth devices
- Link-management aspects for connecting to Bluetooth devices
- Procedures related to security levels
- Common formats for user interface-level parameters, such as naming conventions

GAP defines and assigns the following profile roles to the devices.

Profile role	Description
Central	• A device with relatively greater processing power and memory. For example, a mobile phone or a tablet. • Always a primary device. • It does'nt advertise. • Supports active or passive scanning and all link layer control procedures.

Profile role	Description
Peripheral	• A small, low-power, and resource-contained device that can connect to a much more powerful central device. For example, an instrument like a heart rate monitor, or a Bluetooth Low Energy enabled proximity tag. • Always a secondary device. • Advertises over connections. • Supports all link layer control procedures.
Broadcaster	• Sends nonconnectable advertising events including characteristics and service data. • It doesn't require a receiver.
Observer	• Receives advertising events and listens for characteristics and service data. • It doesn't require a transmitter.

Serial Port Profile

SPP facilitates wireless communication between devices over a virtual serial port. It supports client and server roles.

SPP defines the following elements:

- Requirements of Bluetooth devices to set up emulated serial cable connections using Radio Frequency Communication (RFCOMM) between two peer devices.
- Terms of services provided to applications.
- Features and procedures for interoperability between Bluetooth devices.

General Attribute Profile

GATT is a service framework that uses the Attribute Protocol (ATT) to discover services, and to read and write characteristic values on a peer device. It supports client and server roles.

GATT performs the following functions:

- Interfaces with the application through application profiles.

  The application profile defines the collection of attributes and any permissions required for the attributes used in communication between devices.

- Specifies how two Bluetooth Low Energy devices exchange data using services or declarations, characteristics, and descriptors.

  It doesn't define rules for attribute use. The upper-layer applications derive functionality by using these concepts.

- Stores services, characteristics, and related data in a basic lookup table, with 16-bit IDs assigned to each entry.

Human Interface Device over GATT Profile

HOGP defines how a Bluetooth Low Energy wireless communications device can support HID services over the Bluetooth Low Energy protocol stack using GATT.

Profile role	Description

Advanced Audio Distribution Profile

A2DP defines the requirements to transmit or stream high-quality audio from one device to another over a Bluetooth connection. For example, streaming music from a mobile phone, laptop, or desktop to a wireless headset, and streaming audio to a hearing aid, cochlear implant, or car console.

A2DP supports the following roles.

Profile role	Description
Source	• An audio source that streams digitally to the sink of the piconet. • Facilitates streaming of stereo-quality audio from a multimedia player to a wireless headset or speakers.
Sink	• An audio receiver device in a wireless audio setup. • Compresses audio data using audio codecs and then decodes at the speaker with minimal loss. • Supports high-quality, wireless audio playback and offers a convenient way to enhance the audio experience.

Hands-Free Profile

HFP defines how an audio gateway device can connect to a hands-free device for functions like remote control and audio connection.

HFP defines and assigns the following profile roles to the devices.

Profile role	Description
Client	• A hands-free device. For example, a wireless headset or a vehicle console. • Establishes service level connections, creates an audio connection with a remote audio gateway, and transports dual-tone multifrequency (DTMF) codes. • Supports functionalities like answering or rejecting an incoming call, dialing a number, and controlling the volume.

Profile role	Description
Audio gateway	• A gateway for audio input and output. For example, a mobile phone. • Communicates with a hands-free device, which controls the audio mechanism and other functions of the audio gateway remotely.

Object Push Profile

OPP defines how two Bluetooth devices can exchange objects, such as business cards, images, wallpapers, ringtones, or videos. It allows a Bluetooth device to:

- Pull an object from another Bluetooth device.
- Push an object to another Bluetooth device.
- Exchange objects with another Bluetooth device.

OPP supports client and server roles.

File Transfer Protocol

FTP defines the requirements to exchange files between two Bluetooth devices. FTP supports client and server roles. It allows a Bluetooth device to:

- Browse the files and folders of another Bluetooth device.
- Exchange files and folders with another Bluetooth device.
- Create or delete files and folders on another Bluetooth device.

Phone Book Access Profile

PBAP facilitates the exchange of phone book objects between a remote and a local device. It's based on a client-server interaction model where the client device gets the phone book objects from the server device. In the BlueZ stack, PBAP supports the following roles:

- Phone book server equipment (PSE): Consists of the source phone book objects.
- Phone book client equipment (PCE): Retrieves phone book objects from the PSE.

The types of phone book objects are listed as follows.

Phone book object	Description
Main phone book	Phone book of the device, such as a mobile phone.
Incoming call history	List of most recently received calls.
Outgoing call history	List of most recently made calls.
Missed call history	List of most recently missed calls.
Combined call history	List of most recently received, made, or missed calls.
Speed-dial contacts	List of speed dial entries on the PSE.
Favorite contacts	List of favorite contacts on the PSE.

PBAP allows the following functions:

- Download contacts and call histories from a remote device.
- Browse contacts on a remote device.
- Select a phone book object.

Message Access Profile

MAP defines the features and procedures that devices use to exchange message objects. It's based on a client-server interaction model where the client initiates the transactions. The types of message objects include electronic messages (Email) and short message service (SMS).

MAP combines the messaging capabilities of a messaging server device and the user interface capabilities of a client device to notify, browse, read, delete, generate, and send messages. In the BlueZ stack, MAP supports the following device roles:

- Message server equipment (MSE): Provides the message repository engine.
- Message client equipment (MCE): Uses the message repository engine of the MSE for MAP functions.

4 Bluetooth software architecture

The Bluetooth software architecture consists of open-source and Qualcomm proprietary software.

The following figure shows how the user space and kernel space components interact through a socket interface. In the user space, the sample test applications interact with the BlueZ Bluetooth daemon to run Bluetooth functions.

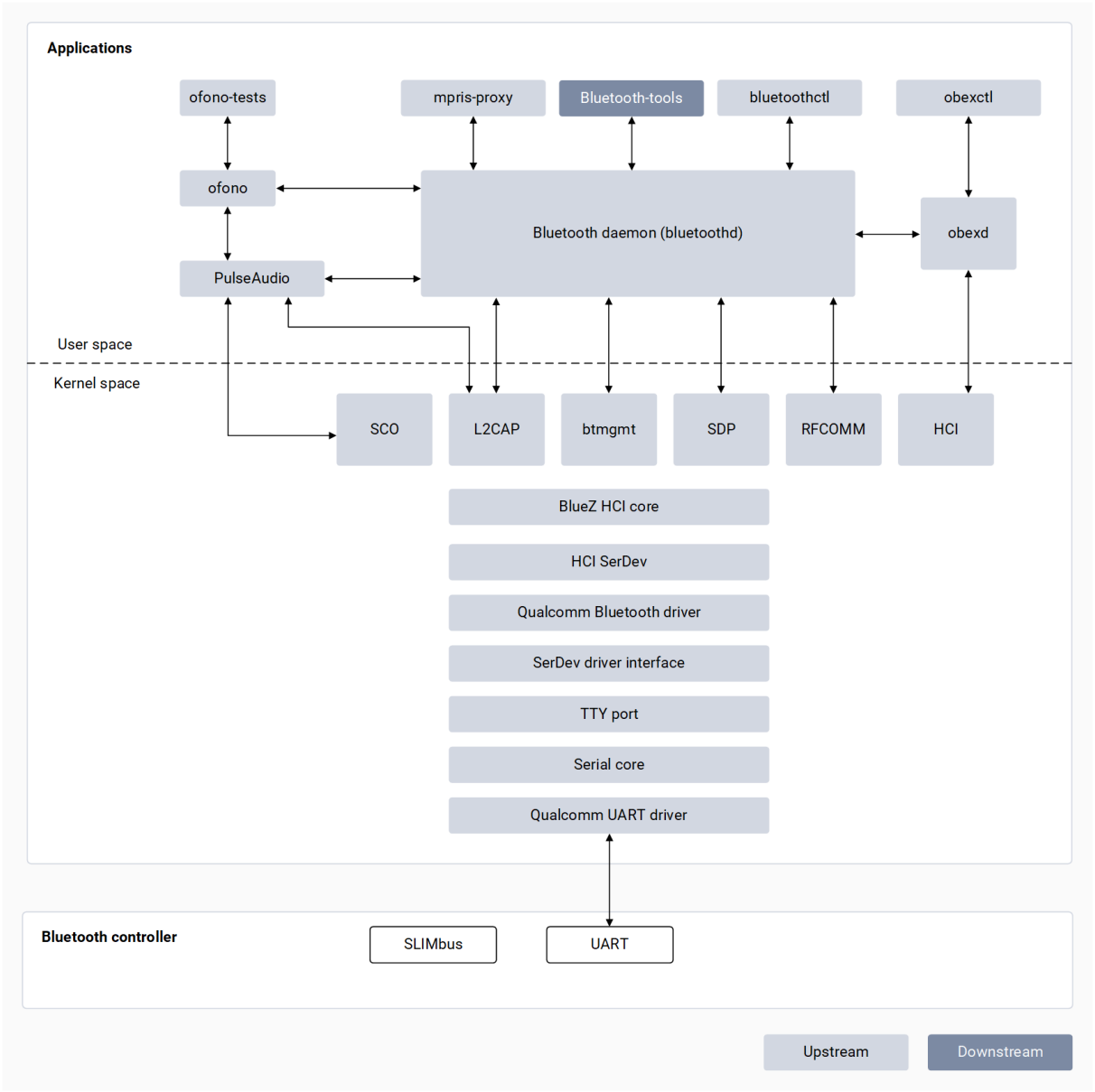


Figure : Bluetooth software architecture

The Bluetooth software architecture consists of the user space, the kernel space, and the Bluetooth controller.

4.1 User space components

The user space comprises the following Bluetooth applications, daemon, and interface layers.

Component	Description
Bluetooth daemon (bluetoothd)	• A central daemon. • Comprises data bus (DBUS) interfaces for the user interface and other subsystems. • Reduces exposure to low-level details. • Extends with plugins.
Bluetooth-tools	• Has the Bluetooth test applications.
bluetoothctl	• A test tool to verify Bluetooth functionality. • Provides options to verify using a command-line interface (CLI).
mpris-proxy	• A proxy DBUS service for Media Player Remote Interfacing Specification (MPRIS).
obexctl	• A test tool to verify Object Exchange Protocol (OBEX) functionality.
OBEX daemon (obexd)	• A daemon for the OBEX profile. • Comprises DBUS interfaces for the user interface.
ofono	• Provides a test application for HFP. • Searches for modem and then enables the HFP audio gateway role.
ofono-tests	Provides options to perform the following functions: • Dial • Answer • Create multiparty • Hold and answer • Swap • Release calls

Component	Description
PulseAudio	• A sound server for portable operating system interface (POSIX). • Facilitates advanced operations on sound data that passes between the applications and hardware.

4.2 Kernel space components

The kernel space comprises the following core components and drivers.

Component	Description
Bluetooth management (btmgmt)	• Enables user space to control kernel operations. • Supports GAP functionalities such as adapter settings, device discovery, and device pairing.
BlueZ Host Control Interface (HCI) core	• Sends and receives HCI commands/events from the user space HCI component to the Qualcomm Bluetooth driver through the raw socket.
HCI	• A layer in the Bluetooth protocol stack. • Defines how the host and the Bluetooth radio controller must interact.
Logical Link Control and Adaptation Protocol (L2CAP)	• Breaks large data packets into smaller frames for transmission. The destined component re-assembles the frames. • Handles quality-of-service (QoS) requirements.
Qualcomm Bluetooth driver	• Handles the Bluetooth general-purpose input/output (GPIO) configurations. • Enables or disables the regulators to turn the chip on or off. • Handles the download of Bluetooth patch and nonvolatile data during the Bluetooth on sequence.

Component	Description
Radio Frequency Communication (RFCOMM)	• Creates reliable and stream-based channels. • Has 30 ports. In contrast, the Transmission Control Protocol (TCP) has 65,535 ports. • An L2CAP connection encapsulates each RFCOMM connection.
Session Description Protocol (SDP)	• Provides the means for client applications to discover the existing services provided by server applications. • Provides the attributes of services, which include the type or class of service offered, and the mechanism or protocol information required to use the service.
SLIMbus Bluetooth driver	• Configures, opens, and closes the SLIMbus ports for Bluetooth audio use cases.
Synchronous Connection Oriented (SCO)	• A type of data link in Bluetooth. • Provides links that are point-to-point connections between a central device and a single peripheral device.
Universal asynchronous receiver/transmitter (UART) serial driver	• A teletype (TTY) serial driver facilitates communication with the Bluetooth controller.

4.3 Bluetooth controller components

The Bluetooth controller components include the Link Manager Protocol (LMP), link controller, baseband, and RF system of the connectivity chipset.

5 Verify functionality of BlueZ stack

In Qualcomm Linux, you can verify the Bluetooth functionality of the BlueZ stack using different sample test applications. The Bluetooth test procedures and use cases for profiles are demonstrated using the following applications.

Sample application	Profile
bluetoothctl	GAP, GATT, A2DP
evtool	HOGP
obexctl	OPP, FTP, PBAP, MAP
ofono	HFP

The sample applications provide the functions of the following Bluetooth profiles as menu options:

5.1 General Access Profile

Bluetooth Low Energy GAP is an extension of the existing BR/EDR GAP.

To perform Bluetooth GAP functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth GAP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the device under test (DUT) and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth test application by running the following command:

   ```
   bluetoothctl
   ```

   Sample output

   ```
   sh-5.1# bluetoothctl
   Agent registered        uetoothd...
   [CHG] Controller 22:22:F1:C1:99:C0 Pairable: yes
   ```

   - To view GAP functions, run the following command:

     ```
     help
     ```

     This command provides the main menu of **bluetoothctl**. The main menu includes the submenus and GAP functions. To perform GAP functions, see Perform Bluetooth GAP functions.

     Sample output

     ```
     [bluetooth]# help
     Menu main:
     Available commands:
     -------------------
     advertise                                       Advertise
     ```

```
Options Submenu
monitor
Advertisement Monitor Options Submenu
scan                                              Scan
Options Submenu
gatt                                              Generic
Attribute Submenu
admin                                             Admin
Policy Submenu
player                                            Media
Player Submenu
endpoint                                          Media
Endpoint Submenu
transport                                         Media
Transport Submenu
list                                              List
available controllers
show [ctrl]                                       Controller
information
select <ctrl>                                     Select
default controller
devices [Paired/Bonded/Trusted/Connected]         List
available devices, with an optional property as the filter
system-alias <name>                               Set
controller alias
reset-alias                                       Reset
controller alias
power <on/off>                                    Set
controller power
pairable <on/off>                                 Set
controller pairable mode
discoverable <on/off>                             Set
controller discoverable mode
discoverable-timeout [value]                      Set
discoverable timeout
agent <on/off/capability>                         Enable/
disable agent with given capability
default-agent                                     Set agent
as the default one
advertise <on/off/type>                           Enable/
disable advertising with given type
set-alias <alias>                                 Set device
alias
scan <on/off/bredr/le>                            Scan for
```

```
devices                                                  
info [dev]                                               Device
information
pair [dev]                                               Pair with
device
cancel-pairing [dev]                                     Cancel
pairing with device
trust [dev]                                              Trust
device
untrust [dev]                                            Untrust
device
block [dev]                                              Block
device
unblock [dev]                                            Unblock
device
remove <dev>                                             Remove
device
connect <dev>                                            Connect
device
disconnect [dev]                                         Disconnect
device
menu <name>                                              Select
submenu
version                                                  Display
version
quit                                                     Quit
program
exit                                                     Quit
program
help                                                     Display
help about this program
export                                                   Print
environment variables
```

- To verify the state of Bluetooth on the device, run the following command:

```
show
```

Sample output

```
[bluetooth]# show
Controller 8C:FD:F0:21:84:23 (public)
        Name: qcs9100-ride-sx
        Alias: qcs9100-ride-sx
```

```
        Class: 0x00000000
        Powered: no
        Discoverable: no
        DiscoverableTimeout: 0x000000b4
        Pairable: yes
        UUID: A/V Remote Control        (0000110e-0000-1000-
8000-00805f9b34fb)
        UUID: PnP Information           (00001200-0000-1000-
8000-00805f9b34fb)
        UUID: Message Access Server     (00001132-0000-1000-
8000-00805f9b34fb)
        UUID: Message Notification Se.. (00001133-0000-1000-
8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-
8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-
8000-00805f9b34fb)
        UUID: Phonebook Access Server   (0000112f-0000-1000-
8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-
8000-00805f9b34fb)
        UUID: OBEX File Transfer        (00001106-0000-1000-
8000-00805f9b34fb)
        UUID: Device Information        (0000180a-0000-1000-
8000-00805f9b34fb)
        UUID: IrMC Sync                 (00001104-0000-1000-
8000-00805f9b34fb)
        UUID: Handsfree                 (0000111e-0000-1000-
8000-00805f9b34fb)
        UUID: OBEX Object Push          (00001105-0000-1000-
8000-00805f9b34fb)
        Modalias: usb:v1D6Bp0246d0541
        Discovering: no
        Roles: central
        Roles: peripheral
Advertising Features:
        ActiveInstances: 0x00 (0)
        SupportedInstances: 0x10 (16)
        SupportedIncludes: tx-power
        SupportedIncludes: appearance
        SupportedIncludes: local-name
        SupportedSecondaryChannels: 1M
        SupportedSecondaryChannels: 2M
        SupportedSecondaryChannels: Coded
```

Next steps

Perform Bluetooth GAP functions

You can perform various Bluetooth GAP functions using the commands provided in the main menu of `bluetoothctl`.

Before you begin, set up the device as described in Set up device for Bluetooth GAP functions.

Enable Bluetooth

To enable Bluetooth on the device, run the following command:

```
power on
```

Sample output

```
[bluetooth]# power on
[CHG] Controller 22:22:F1:C1:99:C0 Class: 0x007c0000
Changing power on succeeded
[CHG] Controller 22:22:F1:C1:99:C0 Powered: yes
```

Run Bluetooth inquiry scan

To start an inquiry for nearby devices, run the following command:

```
scan on
```

Sample output

```
[bluetooth]# scan on
Discovery started
[CHG] Controller 22:22:9B:2C:79:1E Discovering: yes
[NEW] Device A4:30:7A:EE:AF:EF [TV] MyDeviceA 8 Series (43)
[NEW] Device 7E:08:AE:BC:66:58 7E-08-AE-BC-66-58
[CHG] Device A4:30:7A:EE:AF:EF RSSI: -91
[CHG] Device A4:30:7A:EE:AF:EF Modalias: bluetooth: v04E8p8080d0000
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110a-0000-1000-8000-
```

```
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 00001112-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000111f-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Key: 0xff19
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Value:
00 75 00 09 01 00 00 00 06 01 00 00 00 00 00 00 .u..............
00 00 00 00 00 00 00 00                         ........
[NEW] Device F8:7D:76:9D:9B:6B MyDeviceB
[CHG] Device 7E:08:AE:BC:66:58 RSSI: -73
```

Stop Bluetooth inquiry scan

To stop an inquiry that's in progress, run the following command:

```
scan off
```

Sample output

```
[bluetooth]# scan off
Discovery stopped
[CHG] Device 00:95:01:ED:A3:1C RSSI is nil
[DEL] Device 00:95:01:ED:A3:1C 00-95-01-ED-A3-1C
[CHG] Device 8C:FD:F0:0F:41:DB RSSI is nil
[DEL] Device 8C:FD:F0:0F:41:DB 8C-FD-F0-0F-41-DB
[CHG] Device 3F:5C:33:B2:F4:13 RSSI is nil
[DEL] Device 3F:5C:33:B2:F4:13 3F-5C-33-B2-F4-13
[CHG] Device C8:12:0B:50:F2:51 RSSI is nil
[DEL] Device C8:12:0B:50:F2:51 C8-12-0B-50-F2-51
[CHG] Device 8C:FD:F0:0F:1B:B8 RSSI is nil
[DEL] Device 8C:FD:F0:0F:1B:B8 8C-FD-F0-0F-1B-B8
[CHG] Device 0E:19:D7:85:53:8A RSSI is nil
[DEL] Device 0E:19:D7:85:53:8A 0E-19-D7-85-53-8A
[CHG] Device 75:AD:91:DD:40:2A TxPower is nil
```

```
[CHG] Device 75:AD:91:DD:40:2A RSSI is nil
[CHG] Device E4:24:2C:94:28:BC RSSI is nil
[CHG] Device 94:7C:00:B0:38:28 RSSI is nil
[CHG] Controller 8C:FD:F0:21:84:23 Discovering: no
```

Pair with a remote Bluetooth device

Before you pair a remote device, run a Bluetooth inquiry scan to ensure that the remote device is available.

To pair with a remote Bluetooth device, run the following command:

```
pair <bt_address>
```

To accept the outgoing/incoming pairing, enter `yes`. To reject the outgoing/incoming pairing, enter `no`.

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To pair with a remote device with `<bt_address>` `F8:7D:76:9D:9B:6B`, run the following command:

```
pair F8:7D:76:9D:9B:6B
```

Sample output

```
[bluetooth]# pair F8:7D:76:9D:9B:6B
Attempting to pair with F8:7D:76:9D:9B:6B
[CHG] Device F8:7D:76:9D:9B:6B Connected: yes
[CHG] Device F8:7D:76:9D:9B:6B Name: MyDeviceB
[CHG] Device F8:7D:76:9D:9B:6B Alias: MyDeviceB
Request /*-9
[agent] Confirm passkey 068560 (yes/no): yes
[CHG] Device F8:7D:76:9D:9B:6B Bonded: yes
[DEL] Device 28:DE:65:7B:59:54 28-DE-65-7B-59-54
[CHG] Device F8:7D:76:9D:9B:6B Modalias: bluetooth:v004Cp760Ad1160
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 00000000-deca-fade-deca-
deafdecacafe
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 00001000-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 0000110a-0000-1000-8000-
00805f9b34fb
```

```
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 00001116-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 0000111f-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 0000112f-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 00001132-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 00001801-0000-1000-8000-
00805f9b34fb
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 02030302-1d19-415f-86f2-
22a2106a0a77
[CHG] Device F8:7D:76:9D:9B:6B UUIDs: 1ff31936-572e-4b36-a2bf-
b2409b1aa6f4
[CHG] Device F8:7D:76:9D:9B:6B ServicesResolved: yes
[CHG] Device F8:7D:76:9D:9B:6B Paired: yes
Pairing successful
[DEL] Device 28:DE:65:7B:5B:71 28-DE-65-7B-5B-71
[CHG] Device F8:7D:76:9D:9B:6B ServicesResolved: no
[CHG] Device F8:7D:76:9D:9B:6B Connected: no
```

Get the bonded/paired device list

To get a verified list of paired devices, run the following command:

```
devices
```

Sample output

```
[bluetooth]# devices
Device F8:7D:76:9D:9B:6B MyDeviceB
```

Unpair a device

To unpair a device, run the following command:

Example

To unpair a device with the address `F8:7D:76:9D:9B:6B`, run the following command:

```
remove F8:7D:76:9D:9B:6B
```

Sample output

```
[bluetooth]# remove F8:7D:76:9D:9B:6B
[DEL] Device F8:7D:76:9D:9B:6B MyDeviceB
Device has been removed
```

Enable device discovery

To enable discovery mode in the DUT, run the following command:

```
discoverable on
```

Sample output

```
[bluetooth]# discoverable on
Changing discoverable on succeeded
[CHG] Controller 8C:FD:F0:21:84:23 Discoverable: yes
```

Disable Bluetooth

To disable Bluetooth on the device, run the following command:

```
power off
```

Sample output

```
[bluetooth]# power off
[CHG] Controller 8C:FD:F0:21:84:23 Discoverable: no
Changing power off succeeded
[CHG] Controller 8C:FD:F0:21:84:23 Powered: no
[CHG] Controller 8C:FD:F0:21:84:23 Discovering: no
[CHG] Controller 8C:FD:F0:21:84:23 Class: 0x00000000
```

5.2 General Attribute Profile

GATT is a service framework that uses ATT to discover services, and to read and write characteristic values on a peer device.

To perform Bluetooth Low Energy GATT server or client functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth Low Energy GATT functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth test application by running the following command:

```
bluetoothctl
```

   You can perform a few GATT functions like connecting and scanning through the main menu options of bluetoothctl.

   Sample output

```
sh-5.1# bluetoothctl
Agent registered        uetoothd...
[CHG] Controller 22:22:F1:C1:99:C0 Pairable: yes
```

4. Enable Bluetooth by running the following command:

```
power on
```

   Sample output

```
[bluetooth]# power on
[CHG] Controller 22:22:F1:C1:99:C0 Class: 0x007c0000
Changing power on succeeded
[CHG] Controller 22:22:F1:C1:99:C0 Powered: yes
```

5. Go to the **GATT submenu** by running the following command:

```
menu gatt
```

   Sample output

```
[bluetooth]# menu gatt
Menu gatt:
Available commands:
-------------------
list-attributes [dev/local]                       List
attributes
select-attribute <attribute/UUID/local> [attribute/UUID] Select
attribute
attribute-info [attribute/UUID]                   Select
attribute
read [offset]                                     Read attribute
value
write <data=xx xx ...> [offset] [type]            Write
attribute value
acquire-write                                     Acquire Write
file descriptor
```

```
release-write                                      Release Write
file descriptor
acquire-notify                                     Acquire Notify
file descriptor
release-notify                                     Release Notify
file descriptor
notify <on/off>                                    Notify
attribute value
clone [dev/attribute/UUID]                         Clone a device
or attribute
register-application [UUID ...]                    Register
profile to connect
unregister-application                             Unregister
profile
register-service <UUID> [handle]                   Register
application service.
unregister-service <UUID/object>                   Unregister
application service
register-includes <UUID> [handle]                  Register as
Included service in.
unregister-includes <Service-UUID> <Inc-UUID>      Unregister
Included service.
register-characteristic <UUID> <Flags=read,write,notify...>
[handle] Register application characteristic
unregister-characteristic <UUID/object>            Unregister
application characteristic
register-descriptor <UUID> <Flags=read,write...> [handle]
Register application descriptor
unregister-descriptor <UUID/object>                Unregister
application descriptor
back                                               Return to main
menu
version                                            Display
version
quit                                               Quit program
exit                                               Quit program
help                                               Display help
about this program
export                                             Print
environment variables
```

- For GATT server functions, see Perform Bluetooth Low Energy GATT server functions.
- For GATT client functions, see Perform Bluetooth Low Energy GATT client functions.

Next steps

Perform Bluetooth Low Energy GATT server functions

You can perform Bluetooth Low Energy GATT server functions using the **GATT submenu** options and `bluetoothctl` commands.

Before you begin, set up the device as described in Set up device for Bluetooth Low Energy GATT functions.

Connect to a remote device

To connect to a remote device, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Note: To get the Bluetooth address of the remote device, run a Bluetooth Low Energy GATT scan.

Example

To connect to a client with `<bt_address>` `6D:38:AF:C6:B5:62`, run the following command:

```
connect 6D:38:AF:C6:B5:62
```

Add a primary service

To add a primary service to the GATT server, run the following command from the `menu gatt` menu:

```
register-service <UUID> [handle]
```

Example

If the UUID of the service is `FF01` and the handle is `30`, run the following command:

```
register-service FF01 30
```

Sample output

```
[MyDeviceB:/service0001/char0008]# register-service FF01 30
[NEW] Primary Service (Handle 0x001e)
      /org/bluez/app/service0
      FF01
[/org/bluez/app/service0] Primary (yes/no): yes
```

Add a characteristic

To add a characteristic to a service of the server, run the following command from the `menu gatt` menu:

```
register-characteristic <UUID> <Flags=read,write,notify...> [handle]
```

Parameters

`<Flags>` are the flag values of the characteristic. For values, see Flag values.

Example

The UUID of a service is `FF02`, the flags are `read,write,notify`, and the handle is `31`. To add a characteristic, run the following command:

```
register-characteristic FF02 read,write,notify 31
```

Sample output

```
[MyDeviceB]# register-characteristic FF02 read,write,notify 31
[NEW] Characteristic (Handle 0x001f)
      /org/bluez/app/service0/chrc0
      FF02
<egister-characteristic FF02 read,write,notify 31[/org/bluez/app/
service0/chrc0] Enter value: 20
```

Add a descriptor

To add a descriptor to a characteristic in the server, run the following command from the `menu gatt` menu:

```
register-descriptor <UUID> <Flags=read,write...> [handle]
```

Parameters

`<Flags>` are the flag values of the descriptor. For values, see Flag values.

Example

The UUID of a service is `FF03`, the flags are `read,write`, and the handle is `33`. To add a descriptor, run the following command:

```
register-descriptor FF03 read,write 33
```

Sample output

```
[MyDeviceB]# register-descriptor FF03 read,write 33
[NEW] Descriptor (Handle 0x0021)
   /org/bluez/app/service0/chrc0/desc0
   FF03
<egister-descriptor FF03 read,write 33[/org/bluez/app/service0/chrc0/
desc0] Enter value: 21
```

Add an included service

Before you begin, add the intended services as primary services.

To add an included service to another service, run the following command from the `menu gatt` menu:

```
register-includes <UUID> <UUID>
```

Example

Consider two primary services with the UUID `FF01` and `1112`, respectively. To add the service `1112` as an included service to the service `FF01`, run the following command:

```
register-includes FF01 1112
```

Sample output

```
[MyDeviceB]# register-includes FF01 1112
[NEW] Primary Service (Handle 0x001e)
      /org/bluez/app/service0
      FF01
[NEW] Primary Included Service (Handle 0x0000)
      /org/bluez/app/service1
      1112
      Unknown
```

Register an application

To publish the services that are available or added to the server, run the following command from the `menu gatt` menu:

```
register-application [UUID]
```

Example

The UUID of the service is `FF01`. To publish the services that are available or added to the server, run the following command:

```
register-application FF01
```

Sample output

```
[MyDeviceB]# register-application FF01
[CHG] Secondary Service (Handle 0x0015)
   /org/bluez/app/service2
   1112
   Unknown
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000111f-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001108-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001800-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110a-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001801-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000180a-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000111e-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001112-0000-1000-8000-
00805f9b34fb
[CHG] Primary Service (Handle 0x0016)
```

```
   /org/bluez/app/service1
   FF01
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000111f-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001108-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001800-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110a-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001801-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000180a-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000ff01-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000111e-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001112-0000-1000-8000-
00805f9b34fb
Application registered
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000111f-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001108-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001800-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000110a-0000-1000-8000-
```

```
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 00001801-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000180a-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000ff01-0000-1000-8000-
00805f9b34fb
[CHG] Controller 22:22:9B:2C:79:1E UUIDs: 0000111e-0000-1000-8000-
00805f9b34fb
```

Start an advertisement

To start a GATT advertisement, run the following command from the `bluetoothctl` menu:

```
advertise on
```

Sample output

```
[MyDeviceB]# advertise on
[CHG] Controller 22:22:9B:2C:79:1E SupportedInstances: 0x0f (15)
[CHG] Controller 22:22:9B:2C:79:1E ActiveInstances: 0x01 (1)
Advertising object registered
Tx Power: off
Name: off
Appearance: off
Discoverable: on
```

Disconnect a remote device

To disconnect a remote device, run the following command from the `bluetoothctl` menu:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a client with `<bt_address>` `6D:38:AF:C6:B5:62`, run the following command:

```
disconnect 6D:38:AF:C6:B5:62
```

Flag values

The value of `<Flags>` can be:

broadcast	authenticated-signed-writes	encrypt-authenticated-read	encrypt-authenticated-notify
read	extended-properties	encrypt-authenticated-write	secure-notify
write	reliable-write	secure-read	encrypt-indicate
write-without-response	writable-auxiliaries	secure-write	encrypt-authenticated-indicate
notify	encrypt-read	authorize	secure-indicate
indicate	encrypt-write	encrypt-notify	

Perform Bluetooth Low Energy GATT client functions

You can perform Bluetooth Low Energy GATT client functions using the **GATT submenu** options and `bluetoothctl` commands.

Connect to a remote device

To connect to a remote device, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Note: To get the Bluetooth address of the remote device, run a Bluetooth Low Energy GATT scan.

Example

To connect to a server device with `<bt_address> 6D:38:AF:C6:B5:62`, run the following command:

```
connect 6D:38:AF:C6:B5:62
```

Sample output

```
[bluetooth]# connect 6D:38:AF:C6:B5:62
Attempting to connect to 6D:38:AF:C6:B5:62
[CHG] Device 6D:38:AF:C6:B5:62 Connected: yes
Connection successful
[NEW] Primary Service (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001
      00001801-0000-1000-8000-00805f9b34fb
      Generic Attribute Profile
[NEW] Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0002
      00002a05-0000-1000-8000-00805f9b34fb
      Service Changed
[NEW] Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0004
      00002b3a-0000-1000-8000-00805f9b34fb
      Server Supported Features
[NEW] Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0006
      00002b29-0000-1000-8000-00805f9b34fb
      Client Supported Features
[NEW] Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0008
      00002b2a-0000-1000-8000-00805f9b34fb
      Database Hash
[CHG] Device 6D:38:AF:C6:B5:62 UUIDs: 00001800-0000-1000-8000-
00805f9b34fb
[CHG] Device 6D:38:AF:C6:B5:62 UUIDs: 00001801-0000-1000-8000-
00805f9b34fb
[CHG] Device 6D:38:AF:C6:B5:62 ServicesResolved: yes
```

Start Bluetooth Low Energy GATT scan

To start a Bluetooth Low Energy GATT scan, run the following command from the `bluetoothctl` menu:

```
scan le
```

Note: To get the scan results, the remote device must advertise using any Bluetooth Low Energy application.

Sample output

```
[MyDeviceB]# scan le
Discovery started
[CHG] Controller 22:22:9B:2C:79:1E Discovering: yes
[CHG] Device F8:7D:76:9D:9B:6B RSSI: -70
[NEW] Device A4:30:7A:EE:AF:EF [TV] MyDeviceA (43)
[NEW] Device 52:2F:07:6F:AA:93 52-2F-07-6F-AA-93
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Key: 0x0075
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Value:
42 04 01 01 e7 a4 30 7a ee af ef a6 30 7a ee af B..... 0z....0z....
ee 01 3b 00 00 00 00 00                          ......
[CHG] Device A4:30:7A:EE:AF:EF Modalias: bluetooth:v04E8p8080d0000
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110a-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 00001112-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 0000111f-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Key: 0x0075
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Value:
42 04 01 01 e7 a4 30 7a ee af ef a6 30 7a ee af B..... 0z....0z....
ee 01 3b 00 00 00 00 00                          ......
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Key: 0xff19
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Value:
00 75 00 09 01 00 00 00 06 01 00 00 00 00 00 00 .u..................
00 00 00 00 00 00 00 00                          ................
[CHG] Device F8:7D:76:9D:9B:6B RSSI: -60
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Key: 0x0075
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Value:
42 04 01 20 e7 20 0d 00 02 01 2b 01 01 00 01 00 B .. ..... .......
00 00 00 00 00 00 00 00                          ........
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Key: 0xff19
[CHG] Device A4:30:7A:EE:AF:EF ManufacturerData Value:
00 75 00 09 01 00 00 00 06 01 00 00 00 00 00 00  .u..............
00 00 00 00 00 00 00 00                            .......
[NEW] Device 4A:04:87:DF:CB:35 4A-04-87-DF-CB-35
[DEL] Device 52:2F:07:6F:AA:93 52-2F-07-6F-AA-93
```

Stop Bluetooth Low Energy GATT scan

To stop a Bluetooth Low Energy GATT scan, run the following command from the `bluetoothctl` menu:

```
scan off
```

Sample output

```
[MyDeviceB]# scan off
Discovery stopped
[CHG] Device A4:30:7A:EE:AF:EF RSSI is nil
[CHG] Device F8:7D:76:9D:9B:6B RSSI is nil
[CHG] Controller 22:22:9B:2C:79:1E Discovering: no
```

Get the list of attributes

To get the list of attributes of the remote device, run the following command from the `menu gatt` menu:

```
list-attributes <bt_address>
```

Parameters

`<bt_address>` is the address of the remote device.

Example

To list the attributes of the remote device with the address `6D:38:AF:C6:B5:62`, run the following command:

```
list-attributes 6D:38:AF:C6:B5:62
```

Sample output

```
[MyDeviceB]# list-attributes 6D:38:AF:C6:B5:62
Primary Service (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001
      00001801-0000-1000-8000-00805f9b34fb
      Generic Attribute Profile
Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0002
      00002a05-0000-1000-8000-00805f9b34fb
```

```
      Service Changed
Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0004
      00002b3a-0000-1000-8000-00805f9b34fb
      Server Supported Features
Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0006
      00002b29-0000-1000-8000-00805f9b34fb
      Client Supported Features
Characteristic (Handle 0x0000)
      /org/bluez/hci0/dev_6D_38_AF_C6_B5_62/service0001/char0008
      00002b2a-0000-1000-8000-00805f9b34fb
      Database Hash
```

Get attribute information

To get the information of an attribute, run the following command from the `menu gatt` menu:

```
attribute-info <attribute/UUID>
```

Parameters

`<attribute/UUID>` is the UUID or the attribute path.

Example

To get the information of an attribute with the UUID `00002b29-0000-1000-8000-00805f9b34fb`, run the following command:

```
attribute-info 00002b29-0000-1000-8000-00805f9b34fb
```

Sample output

```
[MyDeviceB]# attribute-info 00002b29-0000-1000-8000-00805f9b34fb
Characteristic - Client Supported Features
      UUID: 00002b29-0000-1000-8000-00805f9b34fb
      Service: /org/bluez/hci0/dev_73_F4_3F_FA_0A_DF/service0001
      Flags: read
      Flags: write
      MTU: 0x0205
```

Select an attribute

To select the attribute using the UUID or path, run the following command from the `menu gatt` menu:

```
select-attribute <attribute/UUID>
```

Parameters

`<attribute/UUID>` is the UUID or path of the attribute.

Example

- To select an attribute using the UUID `00006677-0000-1000-8000-00805f9b34fb`, run the following command:

  ```
  select-attribute 00006677-0000-1000-8000-00805f9b34fb
  ```

  Sample output

  ```
  Characteristic User Description
  [MyDeviceB]# select-attribute 00006677-0000-1000-8000-
  00805f9b34fb
  <lect-attribute 00006677-0000-1000-8000-00805f9b34fb[MyDeviceB:/
  service0028/char0029/desc002c]#
  [MyDeviceB:/service0028/char0029/desc002c]#
  ```

- To select an attribute using the attribute path `/org/bluez/hci0/dev_6C_5F_B9_ED_5B_48/service0001/char0002`, run the following command:

  ```
  select-attribute /org/bluez/hci0/dev_6C_5F_B9_ED_5B_48/
  service0001/char0002
  ```

  Sample output

  ```
  [MyDeviceB:/service0001/char0004]# ED_5B_48/service0001/
  char0002
  [MyDeviceB:/service0001/char0002]#
  ```

Read a characteristic value

To read a characteristic value, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Read the selected attribute by running the following command:

```
read 0
```

Sample output

```
[MyDeviceB:/service0028/char0029]# read 0
Attempting to read /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/service0028/
char0029
[CHG] Attribute /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/service0028/
char0029 Value:
   11
   11
[MyDeviceB:/service0028/char0029]#
```

Write a characteristic value

To write a characteristic value, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Write a value to the selected attribute by running the following command:

```
write "<value>"
```

Parameters

`<value>` is the value you intend to write.

Example

To write the attribute value as `0x11`, run the following command:

```
write "0x11"
```

Sample output

```
[MyDeviceB:/service0028/char0029]# read 0b
Attempting to read /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/service0028/
char0029
[CHG] Attribute /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/service0028/
char0029 Value:
```

```
   22
   22
[MyDeviceB:/service0028/char0029]# write "0x11"
Attempting to write /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/
service0028/char0029
[MyDeviceB:/service0028/char0029]# read 0
Attempting to read /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/service0028/
char0029
[CHG] Attribute /org/bluez/hci0/dev_64_8C_C7_03_C4_B0/service0028/
char0029 Value:
   11
   11
[MyDeviceB:/service0028/char0029]#
```

Read a descriptor value

To read a descriptor value, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Read the selected attribute by running the following command:

```
read 0
```

Sample output

```
[MyDeviceB]# select-attribute 00006677-0000-1000-8000-00805f9b34fb
<lect-attribute 00006677-0000-1000-8000-00805f9b34fb[MyDeviceB:/
service0028/char0029/desc002c]#
[MyDeviceB:/service0028/char0029/desc002c]#
[MyDeviceB:/service0028/char0029/desc002c]# read 0
Attempting to read /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c
[CHG] Attribute /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c Value:
   00
   00
[MyDeviceB:/service0028/char0029/desc002c]# write "0x11"
Attempting to write /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/
service0028/char0029/desc002c
[MyDeviceB:/service0028/char0829/desc002c]# read 0
Attempting to read /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c
[CHG] Attribute /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
```

```
char0029/desc002c Value:
   11
   11
[MyDeviceB:/service0028/char0029/desc002c]#
```

Write a descriptor value

To write a descriptor value, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Write a value to the selected attribute by running the following command:

   ```
   write "<value>"
   ```

   Parameters

   `<value>` is the value you intend to write.

   Example

   To write the attribute value as `0x11`, run the following command:

   ```
   write "0x11"
   ```

Sample output

```
[MyDeviceB]# select-attribute 00006677-0000-1000-8000-00805f9b34fb
<lect-attribute 00006677-0000-1000-8000-00805f9b34fb[MyDeviceB:/
service0028/char0029/desc002c]#
[MyDeviceB:/service0028/char0029/desc002c]#
[MyDeviceB:/service0028/char0029/desc002c]# read 0
Attempting to read /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c
[CHG] Attribute /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c Value:
   00
   00
[MyDeviceB:/service0028/char0029/desc002c]# write "0x11"
Attempting to write /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/
service0028/char0029/desc002c
[MyDeviceB:/service0028/char0829/desc002c]# read 0
Attempting to read /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c
[CHG] Attribute /org/bluez/hci0/dev_55_B1_D2_67_7C_CD/service0028/
char0029/desc002c Value:
```

```
   11
   11
[MyDeviceB:/service0028/char0029/desc002c]#
```

Read an encrypted characteristic value

To read an encrypted characteristic value, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Read the selected attribute by running the following command:

   ```
   read 0
   ```

3. Enter `yes` when prompted to confirm the reading.

Sample output

```
[MyDeviceB:/service0028/char0029]# read 0
Attempting to read /org/bluez/hci0/dev_4F_A4_EC_0D_D2_F2/service0028/
char0029
[DEL] Device 60:A3:37:5A:35:5F 60-A3-37-5A-35-5F
[NEW] Device 60:A3:37:5A:35:5F 60-A3-37-5A-35-5F
Request confirmation
[agent] Confirm passkey 161326 (yes/no): yes
[CHG] Attribute /org/bluez/hci0/dev_4F_A4_EC_0D_D2_F2/service0028/
char0029 Value:
   00
[CHG] Device D4:8A:39:78:27:41 Address: D4:8A:39:78:27:41
[CHG] Device D4:8A:39:78:27:41 AddressType: public
   00
[CHG] Device D4:8A:39:78:27:41 Bonded: yes
[CHG] Device D4:8A:39:78:27:41 Paired: yes
[CHG] Device D4:8A:39:78:27:41 Class: 0x005a020c
[CHG] Device D4:8A:39:78:27:41 Icon: phone
```

Write an encrypted characteristic value

To write an encrypted characteristic value, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Write a value to the selected attribute by running the following command:

   ```
   write "<value>"
   ```

   Parameters

   `<value>` is the value you intend to write.

   Example

   To write the attribute value as `0x11`, run the following command:

   ```
   write "0x11"
   ```

3. Enter `yes` when prompted to confirm writing.

Turn on notifications for an attribute

To turn on notifications for an attribute, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Enable notifications for the attribute by running the following command:

   ```
   notify on
   ```

   You get notifications of any changes to the attribute.

Sample output

```
[MyDeviceB:/service0028/char0029]# notify on
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Notifying: yes
Notify started
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Value:
   55                                                U
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Value:
   88                                                .
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Value:
   55 55                                             UU
```

```
[CHG] Device D4:8A:39:78:27:41 RSSI: -74
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Value:
   55                                                U
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Value:
   11                                                U
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Value:
   66                                                f
```

Turn off notifications for an attribute

To turn off notifications for an attribute, do the following from the `menu gatt` menu:

1. Select the intended attribute.
2. Disable notifications for the attribute by running the following command:

```
notify off
```

Sample output

```
[MyDeviceB:/service0028/char0029]# notify off
[CHG] Attribute /org/bluez/hci0/dev_44_38_FB_67_B2_E2/service0028/
char0029 Notifying: no
Notify stopped
```

5.3 Human Interface Device over GATT Profile

HOGP defines how a Bluetooth Low Energy wireless communications device can support HID services over the Bluetooth Low Energy protocol stack using GATT.

Set up device for HOGP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

```
ssh root@<device_IP_address>
```

Example

If the IP address of the device is `10.92.160.222`, run the following command:

```
ssh root@10.92.160.222
```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Open the Bluetooth test application by running the following command:

```
bluetoothctl
```

You can perform a few GATT functions like connecting and scanning through the main menu options of **bluetoothctl**.

Sample output

```
sh-5.1# bluetoothctl
Agent registered        uetoothd...
[CHG] Controller 22:22:F1:C1:99:C0 Pairable: yes
```

4. Enable Bluetooth by running the following command:

```
power on
```

Sample output

```
[bluetooth]# power on
[CHG] Controller 22:22:F1:C1:99:C0 Class: 0x007c0000
Changing power on succeeded
[CHG] Controller 22:22:F1:C1:99:C0 Powered: yes
```

For HOGP functions, see Perform Bluetooth HOGP functions.

Next steps

Perform Bluetooth HOGP functions

You can perform HOGP functions using the `bluetoothctl` options and evtest tool.

Before you begin, set up your device as described in Set up device for HOGP functions.

Connect a remote device

To connect a remote device, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a paired remote device with `<bt_address>` `30:73:00:41:22:49`, run the following command:

```
connect 30:73:00:41:22:49
```

Sample output

```
[bluetooth]# connect 30:73:00:41:22:49
Attempting to connect to 30:73:00:41:22:49
[CHG] Device 30:73:00:41:22:49 Connected: yes
[CHG] Device 30:73:00:41:22:49 Modalias: usb:v04E8p7021d0001
[CHG] Device 30:73:00:41:22:49 UUIDs: 00001000-0000-1000-8000-
00805f9b34fb
[CHG] Device 30:73:00:41:22:49 UUIDs: 00001124-0000-1000-8000-
00805f9b34fb
[CHG] Device 30:73:00:41:22:49 UUIDs: 00001200-0000-1000-8000-
00805f9b34fb
[CHG] Device 30:73:00:41:22:49 ServicesResolved: yes
[CHG] Device 30:73:00:41:22:49 WakeAllowed: yes
Connection successful
```

Verify HOGP functionality

The evtest tool is an open-source tool used to verify HOGP functionality. It monitors and queries device events of an input device, such as a mouse or keyboard.

Note: The evtest tool isn't available in the user build. To verify HOGP functionality, compile the tool from open-source and push it to the DUT.

Example

To verify the functionality of a Bluetooth wireless mouse, do the following:

1. Switch on the Bluetooth wireless mouse.
2. Run a Low Energy scan on the DUT for the Bluetooth wireless mouse.
3. Connect the DUT to the Bluetooth wireless mouse.
4. Run the evtest tool in SSH.
5. Select the Bluetooth wireless mouse from the list of connected input devices that appears on the screen.
6. Use the mouse and perform different actions such as clicking, scrolling, dragging, and moving.
7. Verify the events printed on the screen for each action.

Disconnect a remote device

To disconnect a remote device, run the following command from the `bluetoothctl` menu:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a paired remote device with `<bt_address>` `30:73:00:41:22:49`, run the following command:

```
disconnect 30:73:00:41:22:49
```

Sample output

```
[MyBTkeyboard]# disconnect 30:73:00:41:22:49
Attempting to disconnect from 30:73:00:41:22:49
```

```
[CHG] Device 30:73:00:41:22:49 ServicesResolved: no
Successful disconnected
[CHG] Device 30:73:00:41:22:49 Connected: no
```

5.4 Advanced Audio Distribution Profile

A2DP defines how to stream multimedia audio from one device to another over a Bluetooth connection. This mechanism is also called as Bluetooth audio streaming.

To perform A2DP source or sink functions, you must first complete the steps in the following procedure.

Set up device for A2DP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth test application by running the following command:

   ```
   bluetoothctl
   ```

   Sample output

   ```
   sh-5.1# bluetoothctl
   Agent registered        uetoothd...
   ```

```
[CHG] Controller 22:22:F1:C1:99:C0 Pairable: yes
```

4. Go to the **Transport submenu** or **Player submenu**, as required.
   - To go to the **Transport submenu**, run the following command from `bluetoothctl`:

     ```
     menu transport
     ```

     Sample output

     ```
     [bluetooth]# menu transport
     Menu transport:
     Available commands:
     -------------------
     list                                          List
     available transports
     show <transport>                              Transport
     information
     acquire <transport>                           Acquire
     Transport
     release <transport>                           Release
     Transport
     send <transport> <filename>                   Send
     contents of a file
     receive <transport> [filename]                Get/Set
     file to receive
     volume <transport> [value]                    Get/Set
     transport volume
     back                                          Return to
     main menu
     version                                       Display
     version
     quit                                          Quit
     program
     exit                                          Quit
     program
     help                                          Display
     help about this program
     export                                        Print
     environment variables
     ```

     To perform **Transport submenu** functions in the source, see Perform Bluetooth A2DP source functions.

     To perform **Transport submenu** functions in the sink, see Perform Bluetooth A2DP sink

functions.

- To go to the **Player submenu**, run the following command from `bluetoothctl`:

```
menu player
```

Sample output

```
[bluetooth]# menu player
Menu player:
Available commands:
-------------------
list                                          List
available players
show [player]                                 Player
information
select <player>                               Select
default player
play [item]                                   Start
playback
pause                                         Pause
playback
stop                                          Stop
playback
next                                          Jump to
next item
previous                                      Jump to
previous item
fast-forward                                  Fast
forward playback
rewind                                        Rewind
playback
equalizer <on/off>                            Enable/
Disable equalizer
repeat <singletrack/alltrack/group/off>       Set repeat
mode
shuffle <alltracks/group/off>                 Set shuffle
mode
scan <alltracks/group/off>                    Set scan
mode
change-folder <item>                          Change
current folder
list-items [start] [end]                      List items
of current folder
search <string>                               Search
```

```
items containing string
queue <item>                                    Add item to
playlist queue
show-item <item>                                Show item
information
back                                            Return to
main menu
version                                         Display
version
quit                                            Quit
program
exit                                            Quit
program
help                                            Display
help about this program
export                                          Print
environment variables
```

To perform **Player submenu** functions in the sink, see Perform Bluetooth A2DP sink functions.

Next steps

Perform Bluetooth A2DP source functions

You can perform Bluetooth A2DP source functions using the `bluetoothctl` menu, `menu transport` submenu, and `paplay` commands.

Before you begin, set up your device and go to the required menu as described in Set up device for A2DP functions.

Connect a remote device

To connect a remote device in the A2DP source role, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a paired remote device with `<bt_address>` `17:1A:35:8D:A2:A4`, run the following command:

```
connect 17:1A:35:8D:A2:A4
```

Sample output

```
[bluetooth]# connect 17:1A:35:8D:A2:A4
Attempting to connect to 17:1A:35:8D:A2:A4
[CHG] Device 17:1A:35:8D:A2:A4 Connected: yes
[CHG] Device 17:1A:35:8D:A2:A4 UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Device 17:1A:35:8D:A2:A4 UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Device 17:1A:35:8D:A2:A4 UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Device 17:1A:35:8D:A2:A4 ServicesResolved: yes
[CHG] Device 17:1A:35:8D:A2:A4 Bonded: yes
[CHG] Device 17:1A:35:8D:A2:A4 Paired: yes
[NEW] Endpoint /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1
[NEW] Endpoint /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep2
[NEW] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
Connection successful
[CHG] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0 State:
active
[CHG] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
Volume: 0x0040 (64)
[CHG] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0 State:
idle
[CHG] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
Volume: 0x0044 (68)
[CHG] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
Volume: 0x0040 (64)
[MySoundbar]#
```

List available transport

To list the available transport, run the following command from the `menu transport` submenu:

```
list
```

Sample output

```
[MySoundbar]# list
Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
```

Get information about codec capabilities

To get the information about the codec capabilities of a transport, run the following command from the `menu transport` submenu:

```
show <transport>
```

Parameters

`<transport>` is the transport path.

Example

To get the information about the codec capabilities of `<transport> /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0`, run the following command:

```
show /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
```

Sample output

```
[MySoundbar]# show /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
      UUID: 0000110a-0000-1000-8000-00805f9b34fb
      Codec: 0x00 (0)
      Configuration: 0x21 (33)
      Configuration: 0x15 (21)
      Configuration: 0x02 (2)
      Configuration: 0x35 (53)
      Device: /org/bluez/hci0/dev_17_1A_35_8D_A2_A4
      State: idle
      Volume: 0x0040 (64)
```

Set absolute volume

Note: Ensure that the DUT and the remote device support the `setabsvolume` feature.

To set the absolute volume of a transport, run the following command from the `menu transport` submenu:

```
volume <transport> [value]
```

Parameters

- `<transport>` is the transport path.
- `[value]` is the volume level.

Example

To set the absolute volume of `<transport> /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0` as `50`, run the following command:

```
volume /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0 50
```

Sample output

```
[MySoundbar]# volume /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
50
Changing Volume succeeded
[CHG] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
Volume: 0x0032 (50)
[MySoundbar]# volume /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd0
10
Changing Volume succeeded
```

Play an audio file

Note:

- The DUT doesn't support an audio player. Use `paplay` commands to play music.
- Only PCM audio sample is supported.

To play songs from the DUT, do the following:

1. Run SSH.

2. Play a WAV file by running the following command:

```
paplay <audio_filepath> -v
```

Parameters

`<audio_filepath>` is the filepath of the WAV file.

Example

To play a WAV file at `/tmp/pcmtest.wav`, run the following command:

```
paplay /tmp/pcmtest.wav -v
```

Sample output

```
sh-5.1# paplay /tmp/pcmtest.wav -v
Opening a playback stream with sample specification 's16le 2ch
44100Hz' and channel map 'front-left,front-right'.
Connection established.
Strean successfully created.
Buffer metrics: maxlength=4194304, tlength=352800,
prebuf=349276, minreq=3528
Using sample spec 's16le 2ch 44100Hz', channel map 'front-left,
front-right'.
Connected to device bluez_sink.20_19_D8_36_90_40.a2dp_sink
(index: 4, suspended: no).
Stream started.
Time: 4.427 sec; Latency: 2042656 usec.
```

Get a list of sinks

To get the list of sinks when multiple devices are connected, run the following command in SSH:

```
pactl list sinks short
```

Sample output

```
sh-5.1# pactl list sinks short
0 low-latency0    module-pal-card.c    s16le 2ch 48000Hz    SUSPENDED
1 deep-buffer0    module-pal-card.c    s16le 2ch 48000Hz    SUSPENDED
2 offload0        module-pal-card.c    s16le 2ch 48000Hz    SUSPENDED
3 voip-rx0        module-pal-card.c    s16le 2ch 48000Hz    SUSPENDED
4 bluez_sink.20_19_D8_36_90_40.a2dp_sink    module-bluez5-device.c
s16le 2ch 44100Hz    SUSPENDED
```

Play audio on a specific sink

To play audio on a specific sink when multiple devices are connected, run the following command in SSH:

```
paplay <audio_filepath> --device=<device_name> -v
```

Parameters

- `<audio_filepath>` is the filepath of the WAV file.
- `<device_name>` is the sink name.

Example

To play `pcmtest.wav` on the `bluez_sink.20_19_D8_36_90_40.a2dp_sink` sink, run the following command:

```
paplay pcmtest.wav --device=bluez_sink.20_19_D8_36_90_40.a2dp_sink -v
```

Sample output

```
sh-5.1# paplay pcmtest.wav --device=bluez_sink.20_19_D8_36_90_40.
a2dp_sink -v
Opening a playback stream with sample specification 's16le 2ch
44100Hz' and channel map 'front-left,front-right'.
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, tlength=352800, prebuf=349276,
minreq=3528
Using sample spec 's16le 2ch 44100Hz', channel map 'front-left,front-
right'.
Connected to device bluez_sink.20_19_D8_36_90_40.a2dp_sink (index: 4,
suspended: no).
Stream started.
Time: 2.925 sec; Latency: 2062081 usec.
```

Disconnect a remote device

To disconnect a remote device, run the following command from the `bluetoothctl` menu:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a paired remote device with `<bt_address>` `17:1A:35:8D:A2:A4`, run the following command:

```
disconnect 17:1A:35:8D:A2:A4
```

Sample output

```
[MySoundbar]# disconnect 17:1A:35:8D:A2:A4
Attempting to disconnect from 17:1A:35:8D:A2:A4
[DEL] Transport /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1/fd2
[DEL] Endpoint /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep1
[DEL] Endpoint /org/bluez/hci0/dev_17_1A_35_8D_A2_A4/sep2
[CHG] Device 17:1A:35:8D:A2:A4 ServicesResolved: no
Successful disconnected
[CHG] Device 17:1A:35:8D:A2:A4 Connected: no
```

Perform Bluetooth A2DP sink functions

You can perform Bluetooth A2DP sink functions using the `bluetoothctl` menu, `menu transport` submenu, and `menu player` submenu.

Before you begin, set up your device and go to the required menu as described in Set up device for A2DP functions.

Connect a remote device

To connect a remote device, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a paired remote device with <bt_address> D4:8A:39:78:27:41, run the following command:

```
connect D4:8A:39:78:27:41
```

Sample output

```
[bluetooth]# connect D4:8A:39:78:27:41
Attempting to connect to D4:8A:39:78:27:41
[CHG] Device D4:8A:39:78:27:41 Connected: yes
[NEW] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1
[NEW] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep2
[NEW] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep3
[NEW] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep4
[NEW] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep5
[NEW] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
Connection successful
[NEW] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 [default]
[CHG] Device D4:8A:39:78:27:41 ServicesResolved: yes
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Repeat:
off
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Shuffle:
off
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
stopped
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Title: Not
Provided
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0
TrackNumber: 0x00000001 (1)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0
NumberOfTracks: 0x00000001 (1)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Duration:
0x00000000 (0)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Album:
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Artist:
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Genre:
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
Volume: 0x004d (77)
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
pending
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
```

```
active
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
playing
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
idle
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
stopped
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[MyDeviceB]#
```

List available transport

To list the available transport, run the following command from the `menu transport` submenu:

```
list
```

Sample output

```
[MyDeviceB]# list
Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
```

Get information about codec capabilities

To get information about the codec capabilities of a transport, run the following command from the `menu transport` submenu:

```
show <transport>
```

Parameters

`<transport>` is the transport path.

Example

To get information about the codec capabilities of `<transport> /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2`, run the following command:

```
show /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
```

Sample output

```
[MyDeviceB]# show /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
   UUID: 0000110b-0000-1000-8000-00805f9b34fb
   Codec: 0x00 (0)
   Configuration: 0x21 (33)
   Configuration: 0x15 (21)
   Configuration: 0x02 (2)
   Configuration: 0x35 (53)
   Device: /org/bluez/hci0/dev_D4_8A_39_78_27_41
   State: active
   Volume: 0x007f (127)
```

Set absolute volume

Note: Ensure that the DUT and the remote device support the `setabsvolume` feature.

To set the absolute volume of a transport, run the following command from the `menu transport` submenu:

```
volume <transport> [value]
```

Parameters

- `<transport>` is the transport path.
- `[value]` is the volume level.

Example

To set the absolute volume of `<transport> /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2` as `0`, run the following command:

```
volume /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 0
```

Sample output

```
<me /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 0
Changing Volume succeeded
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
Volume: 0x0000 (0)
<me /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 100
Changing Volume succeeded
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2
```

```
Volume: 0x0064 (100)
< /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 50
Changing Volume succeeded
```

Play an audio file

To play an audio file, run the following command from the `menu player` submenu:

```
play [item]
```

Sample output

```
[MyDeviceB]# play
Attempting to play
Play successful
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
pending
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
active
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x000cc8b9 (837817)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
playing
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x000cc932 (837938)
[MyDeviceB]#
```

Pause playback

To pause playback, run the following command from the `menu player` submenu:

```
pause
```

Sample output

```
[MyDeviceB]# pause
Attempting to pause
Pause successful
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x000cc888 (837768)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
paused
```

```
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x000cc888 (837768)
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
idle
[MyDeviceB]#
```

Stop playback

To stop playback, run the following command from the `menu player` submenu:

```
stop
```

Sample output

```
[MyDeviceB]# stop
Attempting to stop
Stop successful
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
stopped
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
[CHG] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd2 State:
idle
```

Play the next audio file

To play the next audio file, run the following command from the `menu player` submenu:

```
next
```

Sample output

```
[MyDeviceB]# next
Attempting to jump to next
Next successful
```

Play the previous audio file

To play the previous audio file, run the following command from the `menu player` submenu:

```
previous
```

Sample output

```
[MyDeviceB]# previous
Attempting to jump to previous
Previous successful
```

Fast forward playback

To fast forward playback, run the following command from the `menu player` submenu:

```
fast-forward
```

Sample output

```
[MyDeviceB]# fast-forward
Fast forward playback
FastForward successful
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Status:
forward-seek
[CHG] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 Position:
0x00000000 (0)
```

Disconnect a remote device

To disconnect a remote device, run the following command from the `bluetoothctl` menu:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a paired remote device with `<bt_address>` `D4:8A:39:78:27:41`, run the following command:

```
disconnect D4:8A:39:78:27:41
```

Sample output

```
[MyDeviceB]# disconnect D4:8A:39:78:27:41
Attempting to disconnect from D4:8A:39:78:27:41
[DEL] Player /org/bluez/hci0/dev_D4_8A_39_78_27_41/player0 [default]
[DEL] Transport /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1/fd1
[DEL] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep1
[DEL] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep2
[DEL] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep3
[DEL] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep4
[DEL] Endpoint /org/bluez/hci0/dev_D4_8A_39_78_27_41/sep5
[CHG] Device D4:8A:39:78:27:41 ServicesResolved: no
Successful disconnected
[CHG] Device D4:8A:39:78:27:41 Connected: no
```

Other player functions

The `menu player` submenu offers other functions that enable you to:

- Shuffle tracks
- Rewind an audio file
- Repeat a single track, all tracks, or a playlist
- Switch the equalizer on or off
- Search a track

5.5 Hands-Free Profile

HFP defines how an audio gateway device can connect to a hands-free device for functions like remote control and audio connection.

To perform HFP client or audio gateway functions, you must first complete the steps in the following procedure.

Set up the device for HFP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Configure the DUT for the HFP client or audio gateway functions, as required.

Configure the DUT for HFP client functions

After the initial device set up for HFP functions, do the following for an HFP client:

1. Verify the status of ofono service by running the following command in SSH:

   ```
   systemctl status ofono
   ```

2. Open the pulse configuration file `system.pa` at the `/etc/pulse/` directory in a text editor.
3. Add `headset=ofono` to the following lines in the file:

   ```
   Load module bluetooth discover
   load-module module-bluetooth-discover headset=ofono
   ```

   The file must appear as follows:

   ```
   ### Load module bluetooth discover
   load-module module-bluetooth-discover headset=ofono

   ### Load module bluetooth policy
   load-module module-bluetooth-policy
   sh-5.1#
   ```

4. Save the file.
5. Reload the daemon by running the following command:

```
systemctl daemon-reload
```

6. Restart PulseAudio by running the following command:

```
systemctl restart pulseaudio
```

   After you reload the daemon and restart PulseAudio, the changes in the `system.pa` configuration file reflect in the system. PulseAudio establishes a connection with ofono using DBUS.

7. Run `bluetoothctl` by running the following command in SSH:

```
bluetoothctl
```

8. Verify if the UUID of the hands-free-client device appears in the list by running the following command from the `bluetoothctl` menu:

```
show
```

   Sample output

```
[bluetooth]# show
Controller 22:22:9B:2C:79:1E (public)
   Name: qcm6490
   Alias: qcm6490
   Class: 0x002c0000
   Powered: yes
   Discoverable: no
   DiscoverableTimeout: 0x000000b4
   Pairable: yes
   UUID: A/V Remote Control        (e000110e-0000-1000-8000-
00805f9b34fb)
   UUID: PnP Information           (e0001200-0000-1000-8000-
00805f9b34fb)
   UUID: Audio Source              (e000110a-0000-1000-8000-
00805f9b34fb)
   UUID: Audio Sink                (e000110b-0000-1000-8000-
00805f9b34fb)
   UUID: A/V Remote Control Target (0000110c-0000-1000-8000-
00805f9b34fb)
   UUID: Generic Access Profile    (e0001800-0000-1000-8000-
00805f9b34fb)
   UUID: Generic Attribute Profile (00001801-0000-1000-8000-
```

```
00805f9b34fb)
   UUID: Device Information        (0000180a-0000-1000-8000-
00805f9b34fb)
   UUID: Handsfree                 (e000111e-0000-1000-8000-
00805f9b34fb)
   Modalias: usb:v1D6Bp0246d0541
   Discovering: no
   Roles: central
   Roles: peripheral
Advertising Features:
   ActiveInstances: 0x00 (0)
   SupportedInstances: 0x10 (16)
   SupportedIncludes: tx-power
   SupportedIncludes: appearance
   SupportedIncludes: local-name
   SupportedSecondaryChannels: 1M
   SupportedSecondaryChannels: 2M
   SupportedSecondaryChannels: Coded
[bluetooth]#
```

Note: The hands-free audio gateway and headset UUIDs aren't listed. Only the hands-free client UUID that's configured from PulseAudio appears in the list.

9. Open the `ofono/test` menu by running the following command in SSH:

```
ls -rt /usr/lib/ofono/test
```

Note: The `ofono/test` application is available only in `qcom-multimedia-test-image`. To run the `ofono/test` application on the user build, you must compile it.

Sample output

```
sh-5.1# ls -rt /usr/lib/ofono/test
unlock-pin                set-ddr                   hangup-
call
test-stk-menu             set-context-property      hangup-
all
test-ss-control-cs        set-cbs-topics            hangup-
active
test-ss-control-cf        set-call-forwarding       get-tech-
```

```
preference
test-ss-control-cb          send-vcard                  get-
serving-cell-info
test-ss                     send-vcal                   get-
operators
test-sms                    send-ussd                   get-icon
test-smart-messaging        send-sms                    enter-pin
test-serving-cell-info      scan-for-operators          enable-
throttling
test-push-notification      reset-pin                   enable-
modem
test-phonebook              remove-contexts             enable-
gprs
test-network-registration   release-and-swap           enable-
cbs
test-modem                  release-and-answer          display-
icon
test-message-waiting        reject-calls                disable-
throttling
test-gnss                   register-operator           disable-
modem
test-cbs                    register-auto               disable-
gprs
test-call-settings          receive-sms                 disable-
call-forwarding
test-call-forwarding        process-context-settings    dial-
number
test-call-barring           private-chat
deactivate-context
test-advice-of-charge       online-modem
deactivate-all
swap-calls                  offline-modem               create-
multiparty
set-use-sms-reports         monitor-ofono               create-
mms-context
set-umts-band               lockdown-modem              create-
internet-context
set-tty                     lock-pin                    change-
pin
set-tech-preference         list-operators              cdma-set-
credentials
set-speaker-volume          list-modems                 cdma-
list-call
set-sms-smsc                list-messages               cdma-
```

```
hangup
set-sms-bearer              list-contexts               cdma-
dial-number
set-sms-alphabet            list-calls                  cdma-
connman-enable
set-roaming-allowed         list-applications           cdma-
connman-disable
set-msisdn                  list-allowed-access-points  cancel-
ussd
set-mms-details             initiate-ussd               cancel-
sms
set-mic-volume              ims-unregister              backtrace
set-lte-property            ims-register                answer-
calls
set-gsm-band                hold-and-answer             activate-
context
set-fast-dormancy           hangup-multiparty
sh-5.1#
```

For HFP client functions, see Perform Bluetooth HFP client functions.

Configure the DUT for HFP audio gateway functions

After the initial device set up for HFP functions, do the following for an HFP audio gateway:

1. Run `bluetoothctl` by running the following command in SSH:

   ```
   bluetoothctl
   ```

2. Verify if the UUID of the hands-free audio gateway and the headset appear in the list by running the following command from the `bluetoothctl` menu:

   ```
   show
   ```

   Sample output

   ```
   [bluetooth]# show
   Controller 22:22:9B:2C:79:1E (public)
      Name: qcm6490
      Alias: qcm6490
      Class: 0x006c0000
      Powered: yes
      Discoverable: no
      DiscoverableTimeout: 0x000000b4
      Pairable: yes
   ```

```
   UUID: A/V Remote Control        (0000110a-0000-1000-8000-
00805f9b34fb)
   UUID: PnP Information           (00001200-0000-1000-8000-
00805f9b34fb)
   UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-
00805f9b34fb)
   UUID: Audio Sink                (0000110b-0000-1000-8000-
00805f9b34fb)
   UUID: Headset                   (00001108-0000-1000-8000-
00805f9b34fb)
   UUID: A/V Remote Control Target (0000110c-0000-1000-8000-
00805f9b34fb)
   UUID: Generic Access Profile    (00001800-0000-1000-8000-
00805f9b34fb)
   UUID: Audio Source              (0000110a-0000-1000-8000-
00805f9b34fb)
   UUID: Generic Attribute Profile (00001801-0000-1000-8000-
00805f9b34fb)
   UUID: Device Information        (0000180a-0000-1000-8000-
00805f9b34fb)
   UUID: Handsfree                 (0000111e-0000-1000-8000-
00805f9b34fb)
   Modalias: usb:v1D6Bp0246d0541
   Discovering: no
   Roles: central
   Roles: peripheral
Advertising Features:
   ActiveInstances: 0x00 (0)
   SupportedInstances: 0x10 (16)
   SupportedIncludes: tx-power
   SupportedIncludes: appearance
   SupportedIncludes: local-name
   SupportedSecondaryChannels: 1M
   SupportedSecondaryChannels: 2M
   SupportedSecondaryChannels: Coded
[bluetooth]# |
```

For HFP audio gateway functions, see Perform Bluetooth HFP audio gateway functions.

Next steps

Perform Bluetooth HFP client functions

You can perform HFP client functions using the `bluetoothctl` menu and `ofono/test` tools.

Before you begin, set up your device and configure the DUT for HFP client functions.

Connect a remote device

To connect a remote device, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a paired remote device with `<bt_address>` `F8:7D:76:9D:9B:6B`, run the following command:

```
connect F8:7D:76:9D:9B:6B
```

Sample output

```
[bluetooth]# connect F8:7D:76:9D:9B:6B
Attempting to connect to F8:7D:76:9D:9B:6B
(CHG) Device F8:7D:76:9D:9B:6B Connected: yes
[NEW] Endpoint /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep3
[NEW] Endpoint /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep4
[NEW] Endpoint /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep5
[NEW] Endpoint /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep6
(NEW) Endpoint /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep1
(NEW) Endpoint /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep2
[NEW] Transport /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/sep1/fd0
Connection successful
[NEW] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 [default]
[NEW] Item /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0/Filesystem /
Filesystem
[NEW) Item /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0/NowPlaying /
NowPlaying
[CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Type:
Audio
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Subtype:
None
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Status:
```

```
stopped
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Browsable:
yes
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0
Searchable: yes
[CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Playlist:
/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0/NowPlaying
(CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Name:
Music
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Repeat:
off
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Shuffle:
off
[NEW] Folder /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0
[CHG] Device F8:7D:76:9D:9B:6B ServicesResolved: yes
(CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Duration:
0x00000000 (0)
(CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Genre:
[CHG] Player /org/blwez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Title:
[CHG] Player /org/blwez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Item: /
org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player@/NowPlaying/
item18446749073709551615
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0
TrackNumber: 0x00000000 (0)
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Artist:
(CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0
NumberOfTracks: 0x00000000 (0)
[CHG) Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Album:
[NEW] Item /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0/NowPlaying/
item18446744073709551615 <unknown>
[CHG] Player /org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/player0 Position:
0x00000000 (0)
[MyDeviceA]#
```

Dial a phone number

To dial a phone number, run the following command from the `ofono/test` menu:

```
./dial-number <phone_number>
```

Parameters

`<phone_number>` is the phone number that you want to dial.

Sample output

```
sh-5.1# ./dial-number 7123456789
Using modem /hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B
/hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/voicecall01
sh-5.1#
```

Answer an incoming call

To answer an incoming call, run the following command from the `ofono/test` menu:

```
./answer-calls
```

Sample output

```
sh-5.1# ./answer-calls
[ /hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B ]
[ /hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/voicecall01 ] incoming
sh-5.1# |
```

Note: If you are on a call, an echo is audible. This is a known behavior in software with a BlueZ stack.

List active calls

To list the active calls, run the following command from the `ofono/test` menu:

```
./list-calls
```

Sample output

```
sh-5.1# ./list-calls
[ /hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B ]
```

```
[ /hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/voicecall01 ]
      State = incoming
      LineIdentification = +917123456789
      Name =
      Multiparty = 0
      RemoteHeld = 0
      RemoteMultiparty = 0
      Emergency = 0
```

Reject or disconnect a call

To reject or disconnect a call, run the following command from the `ofono/test` menu:

```
./hangup-call <call_path>
```

Parameters

`<call_path>` is the call path of the active call.

Example

To hang up a call with the path `/hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/voicecall01`, run the following command:

```
./hangup-call /hfp/org/bluez/hci0/dev_F8_7D_76_9D_9B_6B/voicecall01
```

Disconnect a remote device

To disconnect a remote device, run the following command from the `bluetoothctl` menu:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a paired remote device with `<bt_address>` `F8:7D:76:9D:9B:6B`, run the following command:

```
disconnect F8:7D:76:9D:9B:6B
```

Sample output

```
[MyDeviceA]# disconnect
Attempting to disconnect from F8:7D:76:9D:9B:6B
[CHG] Device F8:7D:76:9D:9B:6B ServicesResolved: no
Successful disconnected
[CHG] Device F8:7D:76:9D:9B:6B Connected: no
[bluetooth]#
```

Perform Bluetooth HFP audio gateway functions

You can perform HFP audio gateway functions using the `bluetoothctl` menu and `paplay` commands.

Before you begin, set up your device and configure the DUT for HFP audio gateway functions.

Connect a remote device

To connect a remote device, run the following command from the `bluetoothctl` menu:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a paired remote device with `<bt_address>` `20:19:D8:36:90:40`, run the following command:

```
connect 20:19:D8:36:90:40
```

Sample output

```
[Test]# connect 20:19:D8:36:90:40
Attempting to connect to 20:19:D8:36:90:40
[CHG] Device 20:19:D8:36:90:40 Connected: yes
[CHG] Device 20:19:D8:36:90:40 UUIDs: 00001108-0000-1000-8000-
00805f9b34fb
[CHG] Device 20:19:D8:36:90:40 UUIDs: 0000110b-0000-1000-8000-
00805f9b34fb
[CHG] Device 20:19:D8:36:90:40 UUIDs: 0000110c-0000-1000-8000-
00805f9b34fb
[CHG] Device 20:19:D8:36:90:40 UUIDs: 0000110e-0000-1000-8000-
00805f9b34fb
[CHG] Device 20:19:D8:36:90:40 UUIDs: 0000111e-0000-1000-8000-
```

```
00805f9b34fb
[CHG] Device 20:19:D8:36:90:40 ServicesResolved: yes
[CHG] Device 20:19:D8:36:90:40 Bonded: yes
[CHG] Device 20:19:D8:36:90:40 Paired: yes
[NEW] Endpoint /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1
[NEW] Transport /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1/fd1
Connection successful
[CHG] Transport /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1/fdl State:
active
[CHG] Transport /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1/fd1
Volume: 0x0068 (104)
[CHG] Transport /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1/fd1 State:
idle
```

Verify audio gateway functionality

To verify the audio gateway functionality, do the following:

1. Create a dummy SCO as follows:
    a. Run SSH.
    b. Play a WAV file by running the following command:

```
paplay <file.wav> -v
```

Example

To play the `AG_playback.wav` file, run the following command:

```
paplay AG_playback.wav -v
```

Sample output

```
sh-5.1# paplay AG_playback.wav -v
Opening a playback stream with sample specification 's16le
2ch 8000Hz' and channel map 'front-left,front-right'
Connection established.
Stream successfully created.
Buffer metrics: maxlength=4194304, tlength=64000,
prebuf=63364, minreq=640
Using sample spec 's16le 2ch 8000Hz', channel map 'front-
left,front-right'.
Connected to device bluez_sink.20_19_D8_36_90_40.handsfree_
head_unit (index: 7, suspended: no).
```

```
Stream started.
Time: 2.003 sec; Latency: 2042009 usec.
```

2. Receive microphone data from the remote device by running the following command:

```
parec -v --rate=<rate> --format=<format> --channels=<channel_
number> --file-format=<file_format audio_filepath> --device=
<device_name>
```

Parameters

Options	Parameter	Description	Example
`--rate`	`<rate>`	The specific sample rate to play the audio file.	`16000`
`--format`	`<format>`	The specific sample format to play the audio file.	`s16le`
`--channels`	`<channel_ number>`	The specific number of channels to play the audio file.	`1`
`--file-format`	`<file_format audio_filepath>`	The file format to play the audio file from a directory.	`wav /data/rec1.wav`
`--device`	`<device_name>`	The device name of the source or sink to play the audio file.	`bluez_source.20_19_D8_36_90_40.handsfree_head_unit`

Example

To receive microphone data from a remote device for the example values listed in the table, run the following command:

```
parec -v --rate=16000 --format=s16le --channels=1 --file-
format=wav /data/rec1.wav --device=bluez_source.20_19_D8_36_90_
40.handsfree_head_unit
```

3. Play the recorded microphone data and verify if the audio is clear.

Disconnect a remote device

To disconnect a remote device, run the following command from the `bluetoothctl` menu:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a paired remote device with `<bt_address>` `20:19:D8:36:90:40`, run the following command:

```
disconnect 20:19:D8:36:90:40
```

Sample output

```
[MyHeadset]# disconnect 20:19:D8:36:90:40
Attempting to disconnect from 20:19:D8:36:90:40
[DEL] Transport /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1/fd2
[DEL] Endpoint /org/bluez/hci0/dev_20_19_D8_36_90_40/sep1
[CHG] Device 20:19:D8:36:90:40 ServicesResolved: no
Successful disconnected
[CHG] Device 20:19:D8:36:90:40 Connected: no
```

5.6 Object Push Profile

OPP defines how two Bluetooth devices can exchange objects, such as business cards, images, wallpapers, ringtones, or videos.

To perform Bluetooth OPP functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth OPP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

```
ssh root@<device_IP_address>
```

Example

If the IP address of the device is `10.92.160.222`, run the following command:

```
ssh root@10.92.160.222
```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Open the Bluetooth test application by running the following command:

```
obexctl
```

To view OBEX functions, run the following command:

```
help
```

This command provides the main menu of **obexctl**. To perform OPP server and client functions, see Perform Bluetooth OPP server functions and Perform Bluetooth OPP client functions.

Sample output

```
sh-5.1# obexctl
[NEW] Client /org/bluez/obex
[obex]# help
Menu main:
Available commands:
-------------------
connect <dev> [uuid] [channel]                    Connect
session
disconnect [session]                              Disconnect
session
list                                              List available
sessions
show [session]                                    Session
information
select <session>                                  Select default
session
info <object>                                     Object
information
cancel <transfer>                                 Cancel
transfer
```

```
suspend <transfer>                                  Suspend
transfer
resume <transfer>                                   Resume
transfer
send <file>                                         Send file
pull <file>                                         Pull Vobject &
stores in file
cd <path>                                           Change current
folder
ls <options>                                        List current
folder
cp <source file> <destination file>                 Copy source
file to destination file
mv <source file> <destination file>                 Move source
file to destination file
rm <file>                                           Delete file
mkdir <folder>                                      Create folder
version                                             Display
version
quit                                                Quit program
exit                                                Quit program
help                                                Display help
about this program
export                                              Print
environment variables
[obex]#
```

Next steps

Perform Bluetooth OPP server functions

You can verify Bluetooth OPP server functionality using the commands provided in the main menu of `obexctl`.

Receive a file in server role

To receive a file in the server role, do the following:

1. Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
2. Initiate a connection from the remote device to the DUT.
3. Send any file from the remote device to the DUT through Bluetooth sharing.

   The transfer prints appear in the `obexctl` screen of the DUT. After the file transfer to the DUT, the file is stored at `/var/bluetooth`.

   Sample output

```
[NEW] Session /org/bluez/obex/server/session6
[NEW] Transfer /org/bluez/obex/server/session6/transfer5
[CHG] Transfer /org/bluez/obex/server/session6/transfer5 Size:
36
[CHG] Transfer /org/bluez/obex/server/session6/transfer5 Status:
active
[CHG] Transfer /org/bluez/obex/server/session6/transfer5
Transferred: 36 (@0KB/s 00:00)
[CHG] Transfer /org/bluez/obex/server/session6/transfer5 Status:
complete
[DEL] Transfer /org/bluez/obex/server/session6/transfer5
[DEL] Session /org/bluez/obex/server/session6
[22:22:75:C2:D2:72]#
```

Perform Bluetooth OPP client functions

You can verify Bluetooth OPP client functionality using the commands provided in the main menu of `obexctl`.

Send a file in the client role

Before you begin, set up the device as described in Set up device for Bluetooth OPP functions.

To send a file in the client role, do the following:

1. Connect to the remote device by running the following command from the `obexctl` menu:

```
connect <bt_address> <profile_name>
```

   Parameters

   - `<bt_address>` is the Bluetooth address of the remote device.

- <profile_name> is opp.

Example

To connect to a remote device with <bt_address> 22:22:5A:A5:56:6B, run the following command:

```
connect 22:22:5A:A5:56:6B opp
```

Sample output

```
[obex]# connect 22:22:5A:A5:56:6B opp
Attempting to connect to 22:22:5A:A5:56:6B
[NEW] Session /org/bluez/obex/client/session1 [default]
[NEW] ObjectPush /org/bluez/obex/client/session1
Connection successful
[22:22:5A:A5:56:6B]#
```

2. Create a text file at a local directory of the DUT.
3. Send this text file to the remote device by running the following command from the obexctl menu:

```
send <filepath>
```

Parameters

<filepath> is the filepath of the text file.

Example

To send a text file temp.txt at /local/mnt/workspace/, run the following command:

```
send /local/mnt/workspace/temp.txt
```

Sample output

```
[22:22:5A:A5:56:6B]# send /local/mnt/workspace/temp.txt
Attempting to send /local/mnt/workspace/temp.txt to /org/bluez/
obex/client/session1
[NEW] Transfer /org/bluez/obex/client/session1/transfer1
Transfer /org/bluez/obex/client/session1/transfer1
   Status: queued
   Name: temp.txt
   Size: 36
   Filename: /local/mnt/workspace/temp.txt
   Session: /org/bluez/obex/client/session1
[CHG] Transfer /org/bluez/obex/client/session1/transfer1 Status:
```

```
complete
[DEL] Transfer /org/bluez/obex/client/session1/transfer1
[22:22:5A:A5:56:6B]#
```

4. On the remote device, accept the prompt to receive the file.

   The file is successfully sent to the remote device.

5.7 File Transfer Protocol

FTP defines the requirements to exchange files between two Bluetooth devices.

To perform Bluetooth FTP functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth FTP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Ensure that the DUT and the remote device support the BlueZ stack. One device acts as a server and another device acts as a client.

  Note: To test or verify FTP server functions, a test application isn't required. Perform FTP functions through the client connection.

- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Open the Bluetooth test application by running the following command:

```
obexctl
```

To view OBEX functions, run the following command:

```
help
```

This command provides the main menu of **obexctl**.

Sample output

```
sh-5.1# obexctl
[NEW] Client /org/bluez/obex
[obex]# help
Menu main:
Available commands:
-------------------
connect <dev> [uuid] [channel]                    Connect
session
disconnect [session]                              Disconnect
session
list                                              List available
sessions
show [session]                                    Session
information
select <session>                                  Select default
session
info <object>                                     Object
information
cancel <transfer>                                 Cancel
transfer
suspend <transfer>                                Suspend
transfer
resume <transfer>                                 Resume
transfer
send <file>                                       Send file
pull <file>                                       Pull Vobject &
stores in file
cd <path>                                         Change current
folder
ls <options>                                      List current
folder
```

```
cp <source file> <destination file>                Copy source
file to destination file
mv <source file> <destination file>                Move source
file to destination file
rm <file>                                          Delete file
mkdir <folder>                                     Create folder
version                                            Display
version
quit                                               Quit program
exit                                               Quit program
help                                               Display help
about this program
export                                             Print
environment variables
[obex]#
```

4. Connect to the remote device by running the following command from the `obexctl` menu:

```
connect <bt_address> <profile_name>
```

Parameters

- `<bt_address>` is the Bluetooth address of the remote device.
- `<profile_name>` is `ftp`.

Example

To connect to a remote device with `<bt_address>` `E8:48:B8:C8:20:00`, run the following command:

```
connect E8:48:B8:C8:20:00 ftp
```

Sample output

```
[E8:48:B8:C8:20:00]# connect E8:48:B8:C8:20:00 ftp
Attempting to connect to E8:48:B8:C8:20:00
[NEW] Session /org/bluez/obex/client/session7
[NEW] FileTransfer /org/bluez/obex/client/session7
Connection successful
```

To perform FTP client functions, see Perform Bluetooth FTP client functions.

Next steps

Perform Bluetooth FTP client functions

You can verify Bluetooth FTP client functionality using the commands provided in the main menu of `obexctl`.

Note: To test or verify FTP server functions, a test application isn't required. Perform FTP functions through the client connection.

Before you begin, set up the device as described in Set up device for Bluetooth FTP functions.

Create a folder

To create a folder on the remote device, run the following command from the `obexctl` menu:

```
mkdir <folder_name>
```

You can verify folder creation on the server at the `/var/bluetooth` directory.

Parameters

`<folder_name>` is the new folder name.

Example

To create a folder called `new_dir`, run the following command:

```
mkdir new_dir
```

Sample output

```
#mkdir new_dir
Attempting to CreateFolder
CreateFolder successful
```

Change the current folder

To change the current folder of the remote device, run the following command from the `obexctl` menu:

```
cd <folder_path>
```

Parameters

`<folder_path>` is the folder that you intend to switch to.

Example

To change the current folder to `new_dir`, run the following command:

```
cd new_dir
```

Sample output

```
#cd new_dir
Attempting to ChangeFolder to new_dir
ChangeFolder successful
```

Get current folder information

To get information about the current folder, run the following command from the `obexctl` menu:

```
ls .
```

Sample output

```
# ls .
Attempting to ListFolder
[NEW] Transfer /org/bluez/obex/client/session7/transfer4
[CHG] Transfer /org/bluez/obex/client/session7/transfer4 Size: 611
[CHG] Transfer /org/bluez/obex/client/session7/transfer4 Status:
complete
      Type: folder
      Name: My_folder
      User-perm: RWD
      Group-perm: R
      Other-perm:
      Accessed: 20231214T123503Z
      Modified: 20231214T123502Z
      Created: 20231214T123502Z
      Type: folder
```

```
      Name: new_dir
      User-perm: RWD
      Group-perm: R
      Other-perm:
      Accessed: 20231215T093311Z
      Modified: 20231215T093245Z
      Created: 20231215T093245Z
```

Copy a file to a remote device

To copy a local file to the remote device, run the following command from the `obexctl` menu:

```
cp :<source_file> <destination_file>
```

Parameters

- `<source_file>` is the local file.
- `<destination_file>` is the target file on the remote device.

Example

To copy a local file at `Documents/ftp_client.txt` to the `ftp_client.txt` on the remote device, run the following command:

```
cp :Documents/ftp_client.txt ftp_client.txt
```

Sample output

```
# cp :Documents/ftp_client.txt ftp_client.txt
Attempting to PutFile
[NEW] Transfer /org/bluez/obex/client/session7/transfer5
Transfer /org/bluez/obex/client/session7/transfer5
      Status: queued
      Name: ftp_client.txt
      Size: 10
      Filename: Documents/ftp_client.txt
      Session: /org/bluez/obex/client/session7
[CHG] Transfer /org/bluez/obex/client/session7/transfer5 Status:
complete
[DEL] Transfer /org/bluez/obex/client/session7/transfer5
```

Copy a file from a remote device

Before you begin, change the current folder to the folder of the intended file.

To copy a file from the remote device to a local directory, run the following command from the `obexctl` menu:

```
cp <destination_file> :<source_file>
```

Parameters

- `<source_file>` is the file on the remote device.
- `<destination_file>` is the target file on the local device.

Example

To copy `ftp_client_2.txt` file from the remote device to the local device, run the following command:

```
cp ftp_client_2.txt :ftp_client_2.txt
```

Sample output

```
# cp ftp_client_2.txt :ftp_client_2.txt
Attempting to GetFile
[NEW] Transfer /org/bluez/obex/client/session7/transfer21
Transfer /org/bluez/obex/client/session7/transfer21
      Status: queued
      Name: ftp_client_2.txt
      Size: 0
      Filename: ftp_client_2.txt
      Session: /org/bluez/obex/client/session7
[CHG] Transfer /org/bluez/obex/client/session7/transfer21 Size: 13
[CHG] Transfer /org/bluez/obex/client/session7/transfer21 Status:
complete
[DEL] Transfer /org/bluez/obex/client/session7/transfer21
```

Copy a file in a remote device

To copy a file from a directory to another directory on the remote device, run the following command from the `obexctl` menu:

```
cp <source_file> <destination_file>
```

Parameters

- `<source_file>` is the file you intend to copy.
- `<destination_file>` is the target location.

Example

To copy `ftp_client.txt` to `/new_dir_2/`, run the following command:

```
cp ftp_client.txt ../new_dir_2/ftp_client.txt
```

Sample output

```
# cp ftp_client.txt ../new_dir_2/ftp_client.txt
Attempting to CopyFile
CopyFile successful
```

Move a file

To move a file from one folder to another on the remote device, run the following command from the `obexctl` menu:

```
mv <source_file> <destination_file>
```

Parameters

- `<source_file>` is the file you intend to move.
- `<destination_file>` is the target location.

Example

To move `ftp_client_2.txt` file to `/new_dir_2/`, run the following command:

```
mv ftp_client_2.txt ../new_dir_2/ftp_client_2.txt
```

Sample output

```
#mv ftp_client_2.txt ../new_dir_2/ftp_client_2.txt
Attempting to MoveFile
MoveFile successful
```

Delete a file

To delete a file, run the following command from the `obexctl` menu:

```
rm <filename>
```

Parameters

`<filename>` is the file you intend to delete.

Example

To delete `ftp_client.txt` file, run the following command:

```
rm ftp_client.txt
```

Sample output

```
# rm ftp_client.txt
Attempting to Delete
Delete successful
```

5.8 Phone Book Access Profile

PBAP is a Bluetooth profile for the exchange of phone book objects between a remote and a local device.

To perform Bluetooth PBAP functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth PBAP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth test application by running the following command:

   ```
   obexctl
   ```

   To view OBEX functions, run the following command:

   ```
   help
   ```

   This command provides the main menu of **obexctl**. To perform PBAP server and client functions, see Perform Bluetooth PBAP server functions and Perform Bluetooth PBAP client functions.

   Sample output

   ```
   sh-5.1# obexctl
   [NEW] Client /org/bluez/obex
   [obex]# help
   Menu main:
   Available commands:
   -------------------
   connect <dev> [uuid] [channel]                    Connect
   session
   ```

```
disconnect [session]                                    Disconnect
session
list                                                    List available
sessions
show [session]                                          Session
information
select <session>                                        Select default
session
info <object>                                           Object
information
cancel <transfer>                                       Cancel
transfer
suspend <transfer>                                      Suspend
transfer
resume <transfer>                                       Resume
transfer
send <file>                                             Send file
pull <file>                                             Pull Vobject &
stores in file
cd <path>                                               Change current
folder
ls <options>                                            List current
folder
cp <source file> <destination file>                     Copy source
file to destination file
mv <source file> <destination file>                     Move source
file to destination file
rm <file>                                               Delete file
mkdir <folder>                                          Create folder
version                                                 Display
version
quit                                                    Quit program
exit                                                    Quit program
help                                                    Display help
about this program
export                                                  Print
environment variables
[obex]#
```

Next steps

Perform Bluetooth PBAP server functions

You can verify Bluetooth PBAP server functionality using the commands provided in the main menu of `obexctl`.

Pull a contact from the server

Before you begin, do the following:

- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Create a PBAP directory on the server.

Note: In the BlueZ stack, PBAP server functionality can't be tested or verified directly as you can't create contacts on the server. Hence, you must create a PBAP directory on the DUT. For more information about creating a PBAP directory, see Sample PBAP directory.

To pull a contact from the server, do the following:

1. Initiate a connection from the remote device to the DUT.
2. Accept the connection request on the DUT as follows:
   a. Run SSH on the DUT.
   b. Open the `bluetoothctl` application by running the following command:

      ```
      bluetoothctl
      ```

   c. Authenticate the connection request.
3. Pull the intended contact from the server to the client.
4. Open and verify the retrieved contact on the client.

Sample PBAP directory

You can create a sample PBAP directory to verify PBAP server functionality as follows:

1. Run SSH on the DUT.
2. Create a `telecom` folder for each phone book repository at the `root` by running the following command:

   ```
   mkdir telecom
   ```

3. Change the current directory to `telecom` by running the following command:

   ```
   cd telecom
   ```

4. In the `telecom` directory, create subfolders for different phone book objects.

   To create a subfolder at `telecom`, run the following command:

   ```
   mkdir <subfolder>
   ```

   Parameters

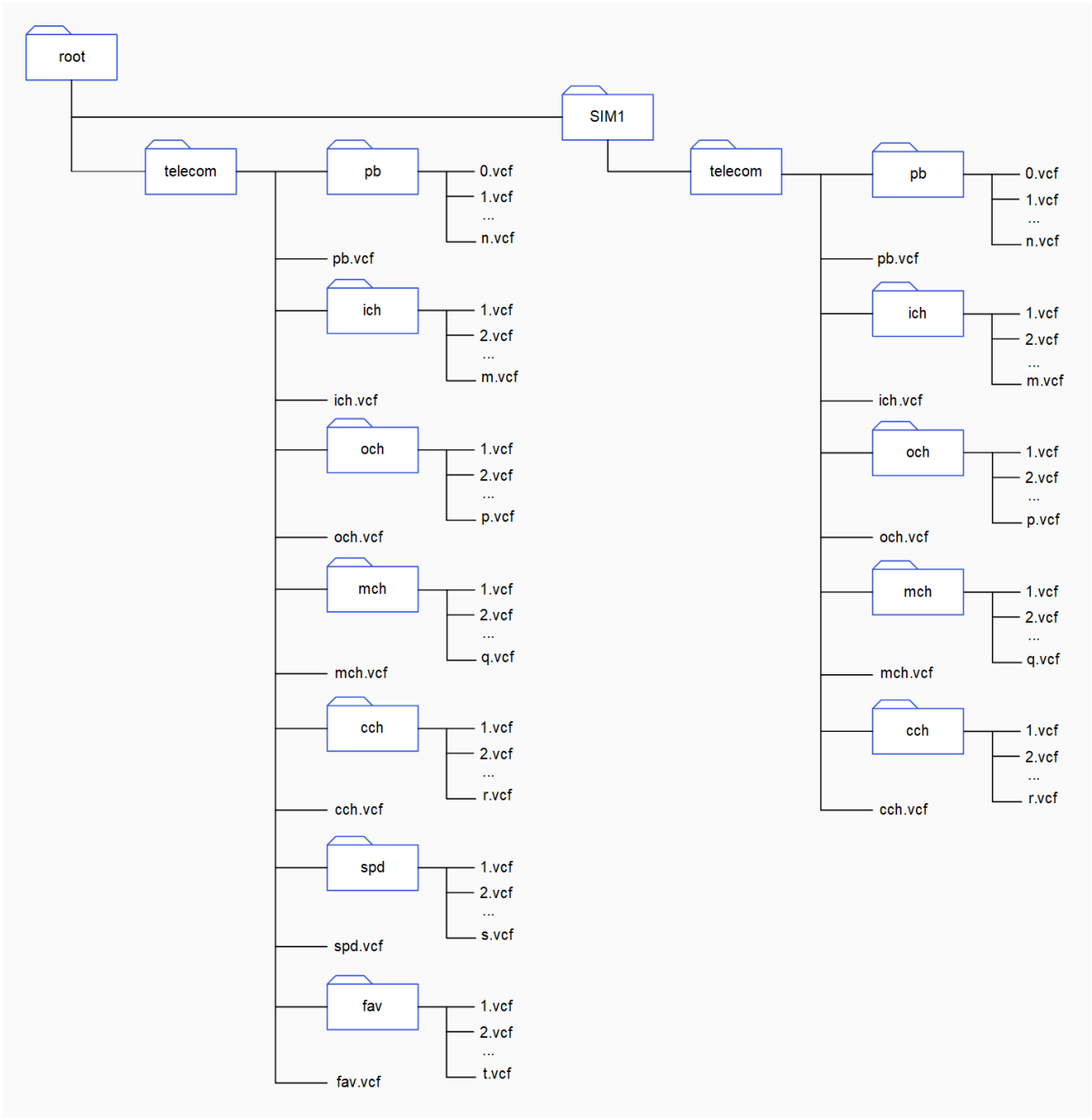
   `<subfolder>` is the name of the subfolder. For example, `pb`.

   Use the following folder names for phone book objects:

Phone book object	Folder name
Main phone book	`pb`
Incoming call history	`ich`
Outgoing call history	`och`
Missed call history	`mch`
Combined call history	`cch`
Speed-dial contacts	`spd`
Favorite contacts	`fav`

5. In each subfolder, create sample VCF files.

The following figure shows a sample PBAP directory. It contains `telecom` directories for two phone book repositories. In the `telecom` directory, there are subfolders for different phone book objects. These subfolders contain VCF files.



Perform Bluetooth PBAP client functions

You can verify Bluetooth PBAP client functionality using the commands provided in the main menu of `obexctl`.

Before you begin, set up the device as described in Set up device for Bluetooth PBAP functions.

Connect the remote device

To connect a remote device in PBAP, run the following command from the `obexctl` menu:

```
connect <bt_address> <profile_name>
```

Parameters

- `<bt_address>` is the Bluetooth address of the remote device.
- `<profile_name>` is `pbap`.

Example

To connect to a remote device with `<bt_address>` `F8:7D:76:9D:9B:6B`, run the following command:

```
connect F8:7D:76:9D:9B:6B pbap
```

Sample output

```
[obex]# connect F8:7D:76:9D:9B:6B pbap
Attempting to connect to F8:7D:76:9D:9B:6B
[NEW] Session /org/bluez/obex/client/session1 [default]
[NEW] PhonebookAccess /org/bluez/obex/client/session1
Connection successful
[F8:7D:76:9D:9B:6B]#
```

Select a phone book object

To select a phone book object, run the following command from the `obexctl` menu:

```
cd <phonebook_object>
```

Parameters

`<phonebook_object>` can be:

- `pb` for the phone book.
- `ich` for the incoming call history.
- `och` for the outgoing call history.
- `mch` for the missed call history.
- `cch` for the history of incoming, outgoing, and missed calls.
- `spd` for the speed dial contacts.

- `fav` for the favorite contacts.

Example

To select the entire phone book, run the following command from the `obexctl` menu:

```
cd pb
```

Sample output

```
[F8:7D:76:9D:9B:6B]# cd pb
Attempting to Select to pb
Select successful
[F8:7D:76:9D:9B:6B]#
```

Pull a phone book

To pull a phone book, do the following:

1. Connect the DUT and the remote device.
2. Select the intended phone book.
3. Pull the intended phone book by running the following command:

   ```
   cp <source file> <destination file>
   ```

   The file is stored at the root directory or at the `/var/bluetooth/` directory.

   Parameters

   - `<source file>` is the phone book you intend to pull.
   - `<destination file>` is the file into which you must pull the phone book.

   Example

   To pull `*.vcf` to `contact.vcf`, run the following command:

   ```
   cp *.vcf contact.vcf
   ```

   Sample output

   ```
   [F8:7D:76:9D:9B:6B]# cp *. vcf contact.vcf
   Attempting to PullAll
   [NEW] Transfer /org/bluez/obex/client/session2/transfer22
   PullAll successful
   [CHG] Transfer /org/bluez/obex/client/session2/transfer22
   Status: active
   ```

```
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 23976 (@23KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 79920 (@55KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 130536 (@50KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 175824 (@45KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 229184 (@53KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 281052 (@51KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 336330 (@55KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 382284 (@45KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 478188 (095KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 615384 (@137KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 691974 (@76KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 740592 (@48KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 786546 (@45KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 829836 (@43KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 877122 (047KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 925074 (047KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 972360 (@47KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 1082250 (@109KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 1206126 (@123KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 1337328 (@131KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Transferred: 1347984 (@10KB/s)
[CHG] Transfer /org/bluez/obex/client/session2/transfer22
Status: complete
```

```
[DEL] Transfer /org/bluez/obex/client/session2/transfer22
[F8:7D:76:9D:9B:6B]#
```

4. Open and verify the phone book.

Get phone book size

To get the size of a phone book, do the following:

1. Connect the DUT and the remote device.
2. Select the intended phone book.
3. Get the size of the intended phone book by running the following command:

```
ls -l
```

Sample output

```
Attempting to GetSize
         [NEW] Transfer /org/bluez/obex/client/session1/
transfer4
         [CHG] Transfer /org/bluez/obex/client/session1/
transfer4 Status: complete
               Size: 0x0006
        Attempting to List
         [DEL] Transfer /org/bluez/obex/client/session1/
transfer4
         [NEW] Transfer /org/bluez/obex/client/session1/
transfer5
         [CHG] Transfer /org/bluez/obex/client/session1/
transfer5 Status: complete
        0.vcf: MyContact1
        1.vcf: MyContact2
        2.vcf: MyContact3
        3.vcf: MyContact4
        4.vcf: MyContact5
        5.vcf: MyContact6
      [DEL] Transfer /org/bluez/obex/client/session1/transfer5
```

Search for a contact

To search for a contact by name or number, do the following:

1. Connect the DUT and the remote device.
2. Select the intended phone book.
3. Search for the contact by running the following command:

```
ls <name_or_number>
```

The contact file appears.

Parameters

`<name_or_number>` is the contact name or number.

Example

To search for a contact called `BT` in a phone book, run the following command:

```
ls BT
```

Sample output

```
# ls BT
      Attempting to Search
   [NEW] Transfer /org/bluez/obex/client/session1/transfer7
   [CHG] Transfer /org/bluez/obex/client/session1/transfer7
Status: complete
      4.vcf: MyContact5
   [DEL] Transfer /org/bluez/obex/client/session1/transfer7
```

5.9 Message Access Profile

MAP defines the features and procedures for devices to exchange message objects.

To perform Bluetooth MAP functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth MAP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

```
ssh root@<device_IP_address>
```

Example

If the IP address of the device is `10.92.160.222`, run the following command:

```
ssh root@10.92.160.222
```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Open the Bluetooth test application by running the following command:

```
obexctl
```

To view OBEX functions, run the following command:

```
help
```

This command provides the main menu of **obexctl**. To perform MAP server and client functions, see Perform Bluetooth MAP server functions and Perform Bluetooth MAP client functions.

Sample output

```
sh-5.1# obexctl
[NEW] Client /org/bluez/obex
[obex]# help
Menu main:
Available commands:
-------------------
connect <dev> [uuid] [channel]                    Connect
session
```

```
disconnect [session]                                    Disconnect
session
list                                                    List available
sessions
show [session]                                          Session
information
select <session>                                        Select default
session
info <object>                                           Object
information
cancel <transfer>                                       Cancel
transfer
suspend <transfer>                                      Suspend
transfer
resume <transfer>                                       Resume
transfer
send <file>                                             Send file
pull <file>                                             Pull Vobject &
stores in file
cd <path>                                               Change current
folder
ls <options>                                            List current
folder
cp <source file> <destination file>                     Copy source
file to destination file
mv <source file> <destination file>                     Move source
file to destination file
rm <file>                                               Delete file
mkdir <folder>                                          Create folder
version                                                 Display
version
quit                                                    Quit program
exit                                                    Quit program
help                                                    Display help
about this program
export                                                  Print
environment variables
[obex]#
```

Next steps

Perform Bluetooth MAP server functions

You can verify Bluetooth MAP server functionality using the commands provided in the main menu of `obexctl`.

Pull and read a message

Before you begin, do the following:

- Pair the DUT and the remote device. For instructions, see Pair with a remote Bluetooth device.
- Create a MAP directory on the server.

Note: In the BlueZ stack, MAP server functionality can't be tested or verified directly as you can't send or receive cellular messages on the device. Hence, you must create a MAP directory on the DUT. For more information about creating a MAP directory, see Sample MAP directory.

To pull and read a message from the server, do the following:

1. Initiate a connection from the remote device to the DUT.
2. Accept the connection request on the DUT as follows:
   a. Run SSH on the DUT.
   b. Open the `bluetoothctl` application by running the following command:

      ```
      bluetoothctl
      ```

   c. Authenticate the connection request.
3. Pull the intended message from the server to the client.
4. Open and verify the retrieved message on the client.

Sample MAP directory

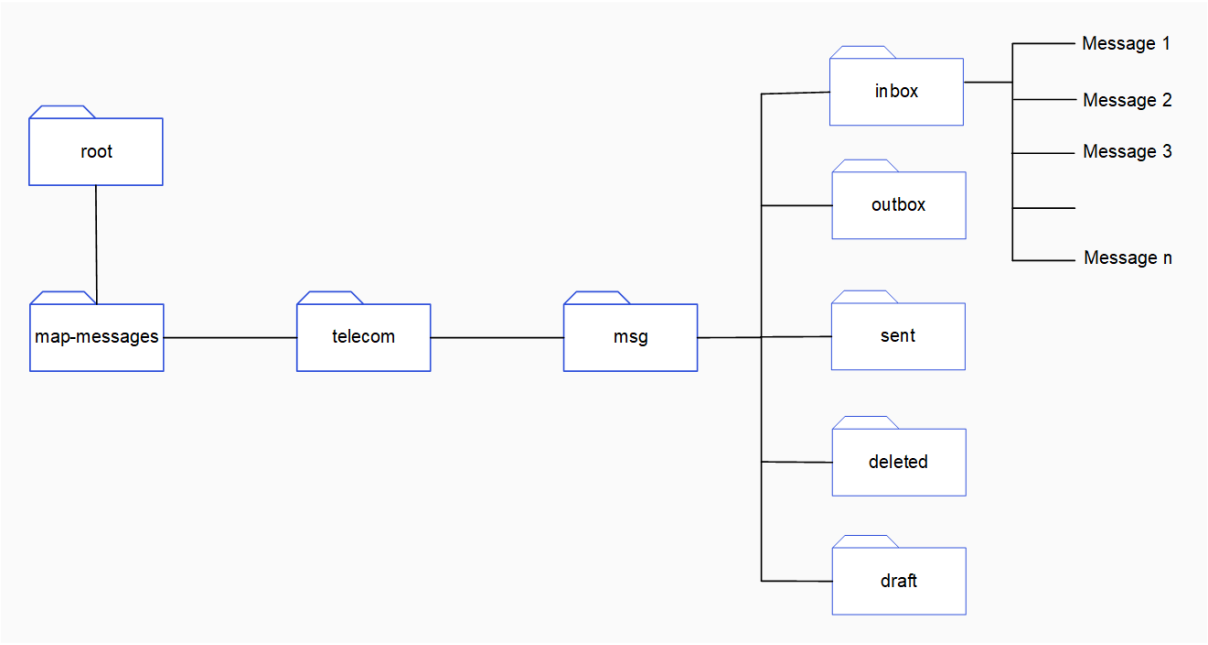
You can create a sample MAP directory to verify MAP server functionality as follows:

1. Run SSH on the DUT.
2. Create a message directory `map-messages/telecom/msg` by running the following commands in sequence:
   a. `mkdir map-messages`
   b. `cd map-messages`
   c. `mkdir telecom`
   d. `cd map-messages/telecom`
   e. `mkdir msg`
   f. `cd telecom/msg`
3. In the `msg` directory, create the following subfolders:
   - `inbox`
   - `outbox`
   - `sent`
   - `deleted`
   - `draft` To create the subfolder at `msg`, run the following command:

```
mkdir <subfolder>
```

   `<subfolder>` is the name of the subfolder. For example, `inbox`.
4. In each subfolder, create message files.

The following figure shows a sample MAP directory. In the `/map-messages/telecom/msg/` directory, there are subfolders for different types of messages. These subfolders contain messages.



Perform Bluetooth MAP client functions

You can verify Bluetooth MAP client functionality using the commands provided in the main menu of `obexctl`.

Before you begin, set up the device as described in Set up device for Bluetooth MAP functions.

Connect the remote device

To connect a remote device in MAP, run the following command from the `obexctl` menu:

```
connect <bt_address> <profile_name>
```

Parameters

- `<bt_address>` is the Bluetooth address of the remote device.
- `<profile_name>` is `map`.

Example

To connect to a remote device with `<bt_address>` `22:22:23:DB:F2:4A`, run the following command:

```
connect 22:22:23:DB:F2:4A map
```

Sample output

```
#connect 22:22:23:DB:F2:4A map
Attempting to connect to 22:22:23:DB:F2:4A
[NEW] Session /org/bluez/obex/client/session23 [default]
[NEW] MessageAccess /org/bluez/obex/client/session23
[NEW] Transfer /org/bluez/obex/client/session23/transfer149
Connection successful
```

List messages in a folder

To list messages in a folder, do the following:

1. Connect the DUT and the remote device.
2. Change the current directory to the intended directory.

   Example

   If the message folder is at `telecom/msg`, run the following command from the `obexctl` menu:

   ```
   cd telecom/msg
   ```

3. List the messages in the intended folder by running the following command:

   ```
   ls <folder_name>
   ```

   Example

   To list the `inbox` messages, run the following command:

   ```
   ls inbox
   ```

   Sample output

   ```
   [22:22:23:DB:F2:4A]# ls inbox
   Attempting to ListMessages
   [NEW] Transfer /org/bluez/obex/client/session22/transfer141
   [CHG] Transfer /org/bluez/obex/client/session22/transfer141
   Status: complete
   [NEW] Message /org/bluez/obex/client/session22/
   message288230376151711846
   [NEW] Message /org/bluez/obex/client/session22/
   message288230376151711844
   [NEW] Message /org/bluez/obex/client/session22/
   message288230376151711842
   ```

Send a message

To send a message, do the following:

1. Connect the DUT and the remote device.
2. Change the current directory to the outbox directory.

   Example

   To change the directory to `outbox`, run the following command from the `obexctl` menu:

   ```
   cd outbox
   ```

3. Ensure that the message is present in the `outbox` folder. If the message isn't present, create one.

   Example

   You can create a BMSG file called `map_file.msg` with the following content:

   ```
   BEGIN:BMSG
   VERSION:1.0
   STATUS:UNREAD
   TYPE:SMS_GSM
   FOLDER:outbox
   NOTIFICATION:1
   BEGIN:VCARD
   VERSION:2.1
   N:QCOM-BTD
   END:VCARD
   BEGIN:BENV
   BEGIN:VCARD
   VERSION:2.1
   N:null;;;;
   TEL:123-456-7890
   END:VCARD
   BEGIN:BBODY
   CHARSET:UTF-8
   LENGTH:50
   BEGIN:MSG
   Hello from client side
   END:MSG
   END:BBODY
   END:BENV
   END:BMSG
   ```

4. Send the message by running the following command from the `obexctl` menu:

```
send <message_filename>
```

Parameters

<message_filename> is the filename of the message you intend to send.

Example

To send map_file.msg, run the following command:

```
send map_file.msg
```

Sample output

```
[22:22:23:DB:F2:4A]# send map_file.msg
Attempting to send map_file.msg to /org/bluez/obex/client/
session22
[NEW] Transfer /org/bluez/obex/client/session22/transfer147
Transfer /org/bluez/obex/client/session22/transfer147
   Status: queued
   Name :
   Size: 322
   Filename: map_file.msg
   Session: /org/bluez/obex/client/session22
[CHG] Transfer /org/bluez/obex/client/session22/transfer147
Status: complete
[DEL] Transfer /org/bluez/obex/client/session22/transfer147
```

6 Verify functionality of Fluoride stack

Bluetooth Fluoride stack supports customization, and provides advanced features for better performance and user experience.

To enable Fluoride stack on the kit, do the following:

1. Apply the following patches:
   - Patch 1:

```
diff --git a/recipes-products/packagegroups/packagegroup-
qcom-multimedia.bb b/recipes-products/packagegroups/
packagegroup-qcom-multimedia.bb
index ac0e838..281fc8a 100644
--- a/recipes-products/packagegroups/packagegroup-qcom-
multimedia.bb
+++ b/recipes-products/packagegroups/packagegroup-qcom-
multimedia.bb
@@ -27,6 +27,7 @@ RDEPENDS:${PN} = "\

RDEPENDS:${PN}:append:qcom-custom-distro = "\
     packagegroup-qcom-audio \
+    packagegroup-qcom-bluetooth \
     packagegroup-qcom-fastcv \
     packagegroup-qcom-graphics \
     packagegroup-qcom-iot-base-utils \
```

   - Patch 2:

```
diff --git a/conf/machine/qcm6490-idp.conf b/conf/machine/
qcm6490-idp.conf
index a108cc07..e105f11f 100644
--- a/conf/machine/qcm6490-idp.conf
+++ b/conf/machine/qcm6490-idp.conf
@@ -19,20 +19,21 @@ KERNEL_DEVICETREE:pn-linux-qcom-custom =
" \
# Additional dtbo to overlay on top of kernel devicetree
files
```

```
KERNEL_TECH_DTBOS[qcm6490-addons-idp] = " \
     qcm6490-graphics.dtbo \
-    qcm6490-camera-idp.dtbo \
+    qcm6490-camera-idp.dtbo qcm6490-bt.dtbo \
     qcm6490-wlan-idp.dtbo qcm6490-video.dtbo \
     qcm6490-wlan-upstream.dtbo \
     "

KERNEL_TECH_DTBOS[qcm6490-addons-idp-amoled] = " \
     qcm6490-graphics.dtbo \
-    qcm6490-camera-idp.dtbo \
+    qcm6490-camera-idp.dtbo qcm6490-bt.dtbo \
     qcm6490-wlan-idp.dtbo qcm6490-video.dtbo \
     qcm6490-wlan-upstream.dtbo \
     "

# Recipe providers of above dtbo files.
KERNEL_TECH_DTBO_PROVIDERS = "\
+    btdevicetree \
     cameradtb \
     qcom-graphicsdevicetree \
     qcom-videodtb \
diff --git a/conf/machine/qcs6490-rb3gen2-core-kit.conf b/
conf/machine/qcs6490-rb3gen2-core-kit.conf
index 13a842a2..63d512df 100644
--- a/conf/machine/qcs6490-rb3gen2-core-kit.conf
+++ b/conf/machine/qcs6490-rb3gen2-core-kit.conf
@@ -27,14 +27,16 @@ KERNEL_DEVICETREE:pn-linux-qcom-custom =
" \
# Additional dtbo to overlay on top of kernel devicetree
files
KERNEL_TECH_DTBOS[qcs6490-addons-rb3gen2] = " \
     qcm6490-graphics.dtbo qcm6490-wlan-rb3.dtbo \
+    qcm6490-bt.dtbo \
     qcm6490-video.dtbo qcm6490-wlan-upstream.dtbo \
     "
KERNEL_TECH_DTBOS[qcs6490-addons-rb3gen2-hsp] = " \
     qcm6490-wlan-rb3-hsp.dtbo \
+    qcm6490-bt-rb3-hsp.dtbo \
     "
KERNEL_TECH_DTBOS[qcs5430-fp1-addons-rb3gen2] = " \
     qcs5430-graphics.dtbo qcm5430-camera-rb3.dtbo \
-    qcs5430-wlan-rb3.dtbo \
+    qcs5430-wlan-rb3.dtbo qcm6490-bt.dtbo \
```

```
     qcm6490-video.dtbo \
     qcs5430-wlan-upstream.dtbo \
     "
@@ -42,7 +44,7 @@ KERNEL_TECH_DTBOS[qcs5430-fp1-addons-
rb3gen2-hsp] = " \
     "
KERNEL_TECH_DTBOS[qcs5430-fp2-addons-rb3gen2] = " \
     qcs5430-graphics.dtbo qcm5430-camera-rb3.dtbo \
-    qcs5430-wlan-rb3.dtbo \
+    qcs5430-wlan-rb3.dtbo qcm6490-bt.dtbo \
     qcm6490-video.dtbo \
     qcs5430-wlan-upstream.dtbo \
     "
@@ -65,6 +67,7 @@ KERNEL_TECH_DTBOS[qcs5430-fp3-addons-
rb3gen2-hsp] = " \

# Recipe providers of above dtbo files.
KERNEL_TECH_DTBO_PROVIDERS = "\
+    btdevicetree \
     cameradtb \
     qcom-graphicsdevicetree \
     qcom-videodtb \
diff --git a/conf/machine/qcs6490-rb3gen2-industrial-kit.conf
b/conf/machine/qcs6490-rb3gen2-industrial-kit.conf
index ed71c55f..7380079f 100644
--- a/conf/machine/qcs6490-rb3gen2-industrial-kit.conf
+++ b/conf/machine/qcs6490-rb3gen2-industrial-kit.conf
@@ -17,6 +17,7 @@ KERNEL_DEVICETREE:pn-linux-qcom-custom = "
\

# Recipe providers of above dtbo files.
KERNEL_TECH_DTBO_PROVIDERS = "\
+    btdevicetree \
     cameradtb \
     qcom-graphicsdevicetree \
     qcom-videodtb \
diff --git a/conf/machine/qcs6490-rb3gen2-vision-kit.conf b/
conf/machine/qcs6490-rb3gen2-vision-kit.conf
index 61237603..b5967a60 100644
--- a/conf/machine/qcs6490-rb3gen2-vision-kit.conf
+++ b/conf/machine/qcs6490-rb3gen2-vision-kit.conf
@@ -31,21 +31,22 @@ KERNEL_DEVICETREE:pn-linux-qcom-custom =
" \
KERNEL_TECH_DTBOS[qcs6490-addons-rb3gen2-video-mezz] = " \
```

```
     qcm6490-graphics.dtbo qcm6490-camera-rb3.dtbo \
     qcm6490-wlan-rb3.dtbo \
-    qcm6490-video.dtbo \
+    qcm6490-bt.dtbo qcm6490-video.dtbo \
     qcm6490-wlan-upstream.dtbo \
     "
KERNEL_TECH_DTBOS[qcs6490-addons-rb3gen2-vision-mezz] = " \
     qcm6490-graphics.dtbo qcm6490-camera-rb3.dtbo \
     qcm6490-wlan-rb3.dtbo \
-    qcm6490-video.dtbo \
+    qcm6490-bt.dtbo qcm6490-video.dtbo \
     qcm6490-wlan-upstream.dtbo \
     "
KERNEL_TECH_DTBOS[qcs6490-addons-rb3gen2-vision-mezz-hsp] = "
\
     qcm6490-wlan-rb3-hsp.dtbo \
+    qcm6490-bt-rb3-hsp.dtbo \
     "
KERNEL_TECH_DTBOS[qcs5430-fp1-addons-rb3gen2-vision-mezz] = "
\
     qcs5430-graphics.dtbo qcm5430-camera-rb3.dtbo \
-    qcs5430-wlan-rb3.dtbo \
+    qcs5430-wlan-rb3.dtbo qcm6490-bt.dtbo \
     qcm6490-video.dtbo \
     qcs5430-wlan-upstream.dtbo \
     "
@@ -53,7 +54,7 @@ KERNEL_TECH_DTBOS[qcs5430-fp1-addons-
rb3gen2-vision-mezz-hsp] = " \
     "
KERNEL_TECH_DTBOS[qcs5430-fp2-addons-rb3gen2-vision-mezz] = "
\
     qcs5430-graphics.dtbo qcm5430-camera-rb3.dtbo \
-    qcs5430-wlan-rb3.dtbo \
+    qcs5430-wlan-rb3.dtbo qcm6490-bt.dtbo \
     qcm6490-video.dtbo \
     qcs5430-wlan-upstream.dtbo \
     "
@@ -75,6 +76,7 @@ KERNEL_TECH_DTBOS[qcs5430-fp3-addons-
rb3gen2-vision-mezz-hsp] = " \
     "
# Recipe providers of above dtbo files.
KERNEL_TECH_DTBO_PROVIDERS = "\
+    btdevicetree \
     cameradtb \
```

```
    qcom-graphicsdevicetree \
    qcom-videodtb \
```

Note: If the patches are not applied correctly, apply them manually.

2. Recompile the entire image and flash it as described in the *Qualcomm Linux Build Guide*.

In Qualcomm Linux, use the **btapp** sample test application to verify the Bluetooth functionality of the Fluoride stack. The Bluetooth test procedures and use cases for each profile are demonstrated using this application.

Note: As the **btapp** is a command-line executable, enter your input in the form of commands in the SSH.

The **btapp** provides the functions of the following Bluetooth profiles as menu options:

6.1 General Access Profile

Bluetooth Low Energy GAP is an extension of the existing BR/EDR GAP.

To perform Bluetooth GAP functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth GAP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

```
ssh root@<device_IP_address>
```

Example

If the IP address of the device is `10.92.160.222`, run the following command:

```
ssh root@10.92.160.222
```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Set a unique Bluetooth MAC address.
   - By default, the Bluetooth MAC address of all devices is `00:00:00:00:5a:ad`.
   - If your device has the default Bluetooth MAC address, set a unique address.
   - If your device doesn't have the default Bluetooth MAC address, then it's a one-time program (OTP) setting. You don't need to change it.

   Example

   To set the Bluetooth MAC address as `11:22:33:44:55:66`, run the following commands:

```
mount -o remount rw /
```

```
cd /var/bluetooth
```

```
python3 BtNvmUtility.py --BDA 11:22:33:44:55:66
```

   Sample output

```
sh-5.1#
sh-5.1# mount -o remount rw /
sh-5.1# cd /var/bluetooth
sh-5.1# python3 BtNvmUtility.py --BDA 11:22:33:44:55:66
------------------------------------------------------------------
------------
**********************BD_Address updated successfully !!
**********************
------------------------------------------------------------------
------------
```

4. Open the Bluetooth test application by running the following command:

```
btapp
```

   Sample output

```
sh-5.1# btapp
get_ap_interface
:: get_ap_interface

***************** Menu ********************
         gap_menu
         test_menu
```

```
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
****************************************
```

5. Go to the **GAP Menu** by running the following command:

```
gap_menu
```

The `gap_menu` consists of generic Bluetooth functions. To perform these functions, see Perform Bluetooth GAP functions.

Sample output

```
sh-5.1# btapp
get_ap_interface
:: get_ap_interface

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
****************************************
gap_menu

***************** Menu ******************
         enable
```

```
         disable
         inquiry
         cancel_inquiry
         get_role_req<space><bt_address>     eg. get_role_req
00:11:22:33:44:55
         pair<space><bt_address><space><transport>     eg. pair
00:11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
         unpair<space><bt_address>     eg. unpair 00:11:22:33:
44:55
         inquiry_list
         bonded_list
         get_state
         get_bt_name
         get_bt_address
         set_bt_name<space><bt name>     eg. set_bt_name MDM_
Fluoride
         set_scan_mode<space><scan mode value (range 0-2)>
eg. set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_
CONNECTABLE,2-BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
         set_afh<space><AFH_Host_Channel_Classification>     eg.
set_afh 112233445566778899f0
         send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd 01,
04,05,33,8b,9e,0a,00 - For Inquiry
         read_clock<space><which_clock range(0-1)><space><bt_
address>     eg. read_clock 0(local)/1(acl connection) 00:11:22:
33:44:55
         main_menu
         switch_role_req<bt_address><space><new_role>     eg.
switch_role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:33:
44:55 1
****************************************
```

Next steps

Perform Bluetooth GAP functions

You can perform various Bluetooth GAP functions using the generic `gap_menu` options.

Before you begin, set up the device and go to the `gap_menu` as described in Set up device for Bluetooth GAP functions.

Enable Bluetooth

To enable Bluetooth on the device, run the following command:

```
enable
```

Sample output

```
gap_menu

****************** Menu ********************
         enable
         disable
         inquiry
         cancel_inquiry
         get_role_req<space><bt_address>     eg. get_role_req 00:11:
22:33:44:55
         pair<space><bt_address><space><transport>     eg. pair 00:
11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
         unpair<space><bt_address>     eg. unpair 00:11:22:33:44:55
         inquiry_list
         bonded_list
         get_state
         get_bt_name
         get_bt_address
         set_bt_name<space><bt name>     eg. set_bt_name MDM_Fluoride
         set_scan_mode<space><scan mode value (range 0-2)>     eg.
set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_CONNECTABLE,2-
BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
         set_afh<space><AFH_Host_Channel_Classification>     eg. set_
afh 112233445566778899f0
         send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd 01,04,05,
33,8b,9e,0a,00 - For Inquiry
         read_clock<space><which_clock range(0-1)><space><bt_address>
    eg. read_clock 0(local)/1(acl connection) 00:11:22:33:44:55
         main_menu
         switch_role_req<bt_address><space><new_role>     eg. switch_
role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:33:44:55 1
```

```
****************************************
enable
current State = 0, new state = 1
BT State is ON
```

Run Bluetooth inquiry scan

To start an inquiry for nearby devices, run the following command:

```
inquiry
```

Sample output

```
inquiry
Inquiry Started
Device Found details:
Found device Addr: d8:b0:53:e5:6a:32
Found device Name: MyDeviceB
Device Type is: 1
Device Found details:
Found device Addr: f8:7d:76:9d:9b:6b
Found device Name: MyDeviceA
Device Type is: 1
Inquiry Stopped automatically
```

Cancel Bluetooth inquiry scan

To cancel an inquiry that's in progress, run the following command:

```
cancel_inquiry
```

Sample output

```
cancel_inquiry

**************************** Inquiry List
********************************
1          MyDeviceB                                  d8:b0:53:e5:6a:
32
2          MyDeviceA                                  f8:7d:76:9d:9b:
6b
**************************** End of List
```

```
*******************************
Inquiry Stopped due to user input
```

Get Bluetooth/Bluetooth Low Energy device list

To get the list of discovered devices, run the following command:

```
inquiry_list
```

Sample output

```
inquiry_list

**************************** Inquiry List
*******************************
1          MyDeviceB                                  d8:b0:53:e5:6a:
32
2          MyDeviceA                                  f8:7d:76:9d:9b:
6b
**************************** End of List
*******************************
```

Pair with remote Bluetooth device

To initiate outgoing Secure Simple Pairing (SSP), run the following command:

```
pair <bt_address> <transport>
```

To accept the outgoing/incoming pairing, enter `yes`. To reject the outgoing/incoming pairing, enter `no`.

Note: Ensure that you provide the correct pin for the incoming legacy pairing.

Parameters

- `<bt_address>` is the Bluetooth address of the remote device.
- `<transport>` is the radio for pairing. The value of the `<transport>` parameter can be:
  - `0`: Auto selection
  - `1`: BR/EDR
  - `2`: Bluetooth Low Energy

Example

To pair a remote device with <bt_address> f8:7d:76:9d:9b:6b, run the following command:

```
pair f8:7d:76:9d:9b:6b 0
```

Sample output

```
pair f8:7d:76:9d:9b:6b 0
Auto select in the stack
ACL state:0 change with reason 00 for device: f8:7d:76:9d:9b:6b

BT pairing_variant 0
***********************************************
BT pairing request::Device MyDeviceB::Pairing Code:: 999712
***********************************************
** Please enter yes / no **
yes

*****************DidInfoCb******************
MAC address:f8:7d:76:9d:9b:6b
spec_id:0x102 :
vendor:0x4c :
vendor_id_source:0x1 :
product:0x760a :
vendor:0x4c :
primary_record:0x1 :
client_executable_url: :
service_description:PnP Information :
documentation_url: :
DID info result :0 :

*****************FINISH******************

***********************************************
Pairing state for MyDeviceB is BONDED
***********************************************
```

Get the bonded/paired device list

To get a verified list of paired devices, run the following command:

```
bonded_list
```

Sample output

```
bonded_list

**************************** Bonded Device List
****************************
MyDeviceB                                 f8:7d:76:9d:9b:6b
****************************  End of List
*********************************
```

Set the Bluetooth device name

To change or set the local Bluetooth device name, run the following command:

```
set_bt_name <bt name>
```

Parameters

`<bt name>` is the device name you intend to set or assign.

Example

To set the device name as `K2L`, run the following command:

```
set_bt_name K2L
```

Sample output

```
get_bt_name
BT Name : Pstack_test
set_bt_name K2L
BT name is set to K2L
get_bt_name
BT Name : K2L
```

Get the Bluetooth name

To get the Bluetooth device name, run the following command:

```
get_bt_name
```

Sample output

In the following sample output of this command, the device name appears as `K2L`:

```
get_bt_name
BT Name : K2L
```

Get the Bluetooth address

To set the Bluetooth MAC address, see General Access Profile.

To get the Bluetooth device address, run the following command:

```
get_bt_address
```

Sample output

In the following sample output of this command, the device address appears as `11:22:33:44:55:66`:

```
get_bt_address
BT Address : 11:22:33:44:55:66
```

Disable Bluetooth

To disable Bluetooth, run the following command:

```
disable
```

Sample output

```
gap_menu

****************** Menu ********************
         enable
         disable
         inquiry
         cancel_inquiry
         get_role_req<space><bt_address>     eg. get_role_req 00:11:
```

```
22:33:44:55
         pair<space><bt_address><space><transport>     eg. pair 00:
11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
         unpair<space><bt_address>     eg. unpair 00:11:22:33:44:55
         inquiry_list
         bonded_list
         get_state
         get_bt_name
         get_bt_address
         set_bt_name<space><bt name>     eg. set_bt_name MDM_Fluoride
         set_scan_mode<space><scan mode value (range 0-2)>     eg.
set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_CONNECTABLE,2-
BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
         set_afh<space><AFH_Host_Channel_Classification>     eg. set_
afh 112233445566778899f0
         send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd 01,04,05,
33,8b,9e,0a,00 - For Inquiry
         read_clock<space><which_clock range(0-1)><space><bt_address>
    eg. read_clock 0(local)/1(acl connection) 00:11:22:33:44:55
         main_menu
         switch_role_req<bt_address><space><new_role>     eg. switch_
role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:33:44:55 1
****************************************
disable
current State = 1, new state = 0
BT_AUDIO_HAL_INTEGRATION needs to be defined
killall: qcbtdaemon: no process killed
killall: wcnssfilter: no process killed
BT State is OFF
```

6.2 Serial Port Profile

SPP is a Bluetooth profile that facilitates wireless communication between devices over a virtual serial port.

To perform Bluetooth SPP server or client functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth SPP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

```
ssh root@<device_IP_address>
```

Example

If the IP address of the device is `10.92.160.222`, run the following command:

```
ssh root@10.92.160.222
```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Open the Bluetooth test application by running the following command:

```
btapp
```

Sample output

```
sh-5.1# btapp
get_ap_interface
:: get_ap_interface

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
```

```
****************************************
```

4. Enable Bluetooth and pair the device as follows:

   a. Go to the **Gap Menu** by running the following command:

   ```
   gap_menu
   ```

   b. Enable Bluetooth by running the following command:

   ```
   enable
   ```

   Sample output

   ```
   gap_menu

   ****************** Menu ********************
            enable
            disable
            inquiry
            cancel_inquiry
            get_role_req<space><bt_address>     eg. get_role_req
   00:11:22:33:44:55
            pair<space><bt_address><space><transport>     eg.
   pair 00:11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
            unpair<space><bt_address>     eg. unpair 00:11:22:
   33:44:55
            inquiry_list
            bonded_list
            get_state
            get_bt_name
            get_bt_address
            set_bt_name<space><bt name>     eg. set_bt_name MDM_
   Fluoride
            set_scan_mode<space><scan mode value (range 0-2)>
    eg. set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_
   CONNECTABLE,2-BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
            set_afh<space><AFH_Host_Channel_Classification>
   eg. set_afh 112233445566778899f0
            send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd
   01,04,05,33,8b,9e,0a,00 - For Inquiry
            read_clock<space><which_clock range(0-1)><space><bt_
   address>     eg. read_clock 0(local)/1(acl connection) 00:11:
   22:33:44:55
            main_menu
   ```

```
         switch_role_req<bt_address><space><new_role>     eg.
switch_role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:
33:44:55 1
****************************************
enable
current State = 0, new state = 1
BT State is ON
```

c. Pair the device for SPP outgoing by running the following command:

```
pair <bt_address> <transport>
```

To accept the outgoing/incoming pairing, enter `yes`.

Parameters

- Run Bluetooth inquiry scan to get the `<bt_address>` of the remote device.
- The values of the `<transport>` parameter can be:
  - `0`: Auto selection
  - `1`: BR/EDR
  - `2`: Bluetooth Low Energy

Example

To initiate a BR/EDR pairing for `<bt_address>` `98:09:cf:a9:82:23`, run the following command:

```
pair 98:09:cf:a9:82:23 1
```

Sample output

```
pair 98:09:cf:a9:82:23 1
BR/EDR Bonding
ACL state:0 change with reason 00 for device: f8:7d:76:9d:9b:
6b
**********************************************
Pairing state for MyDeviceA is BOND NONE
**********************************************
BT pairing request :: MyDeviceB :: Pairing Code :: 776996
**********************************************
**Please enter yes / no **
yes
**********************************************
Pairing state for MyDeviceB is BONDED
```

```
***********************************************
```

d. Return to the **Main Menu** by running the following command:

```
main_menu
```

5. Go to the SPP server or client menu, as required.

   - To go to the **SPP Server Menu**, run the following command from the `main_menu`:

```
spp_server_menu
```

Sample output

```
main_menu

****************** Menu *******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
   ****************************************
spp_server_menu

****************** Menu *******************
         start_server
         stop_server
         send_file<space><directory_with_file_name>eg: send_
file /var/fileName.txt
         recv_file<space><directory_with_file_name>eg: recv_
file /var/fileName.txt
         send_data<space><with_size>eg: send_data 1000[Note:
1000 means 1MB]
         recv_data
         main_menu
****************************************
```

For SPP server functions, see Perform Bluetooth SPP server functions.

- To go to the **SPP Client Menu**, run the following command from the `main_menu`:

```
spp_client_menu
```

Sample output

```
main_menu

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
   ****************************************
spp_client_menu

***************** Menu ******************
         connect <bt_addr>
         disconnect
         send_file<space><directory_with_file_name>eg: send_
file /var/fileName.txt
         recv_file<space><directory_with_file_name>eg: recv_
file /var/fileName.txt
         send_data<space><with_size>eg: send_data 1000[Note:
1000 means 1MB]
         recv_data
         main_menu
   ****************************************
```

For SPP client functions, see Perform Bluetooth SPP client functions.

6. Install any third-party Bluetooth host application, which supports SPP client and server roles, on a remote device. For example, a Bluetooth SPP server/client application from the Google Play Store.

- DUT: Test device
- Remote device: Any smartphone

Next steps

Perform Bluetooth SPP server functions

You can perform SPP server functions using the `spp_server_menu` options.

Before you begin, set up the device and go to the SPP server menu as described in Set up device for Bluetooth SPP functions.

Configure DUT in SPP server role

To configure the DUT in an SPP server role, do the following:

1. Start the server connection by running the following command:

```
start_server
```

2. Initiate a connection from the remote device SPP client application.

   Sample output

   The following sample output shows that the SPP server connection has been established successfully:

```
start server
ACL state:0 change with reason 00 for device: 22:22:85:3d:2e:50
Device is Connected
----------------------
Device Address       : 22:22:85:3d:2e:50
Connection Direction : Server
```

Send a file in the server role

To send a file in the server role, do the following:

1. Set the remote device in Receive mode in the SPP application, if applicable.
2. Create a test file with some sample data and place the file in the `/etc/bluetooth` directory of the DUT.

   For example, create a test file `bt_stack.conf`.
3. Send the test file to the remote device by running the following command:

```
send_file <directory_with_file_name>
```

Parameters

`<directory_with_file_name>` is the complete file path with the filename.

Example

To send the test file `bt_stack.conf` at `/etc/bluetooth`, run the following command:

```
send_file /etc/bluetooth/bt_stack.conf
```

Sample output

```
send_file /etc/bluetooth/bt_stack.conf
File Transfer Complete
----------------------
Device Address : 22:22:85:3d:2e:50
File Name      : /etc/bluetooth/bt_stack.conf
```

Note: If an issue occurs, disconnect the operation from the SPP client application on the remote device, and stop the SPP server in the DUT by running the following command:

```
stop_server
```

4. Verify the status of file reception on the remote application.

Receive a file in server role

To receive a file in the server role, do the following:

1. Set the DUT in Receive mode.
2. Receive a file from the remote device by running the following command:

```
recv_file <directory_with_file_name>
```

Parameters

`<directory_with_file_name>` is the complete file path with the filename.

Note: To receive a file, use the `/etc/bluetooth` directory only.

Example

To receive `fileName.txt` at `/etc/bluetooth` from the remote device, run the following command:

```
recv_file /etc/bluetooth/fileName.txt
```

3. Send the file from the remote device SPP server application.
4. Verify the received file as follows:
   a. Open the command prompt window.
   b. Pull the received file by running the following command:

```
scp -r root@<IP_address>:<source_file_path> <destination_
file_path>
```

To pull a file to the current file path, enter the `<destination_file_path>` as `.` in the command.

Note: When prompted for a password, enter `oelinux123` to authenticate the file transfer through the Secure Copy Protocol (SCP).

Example

The IP address of the device is `10.92.160.222`. To pull `fileName.txt` from `/etc/bluetooth`, run the following command:

```
scp -r root@10.92.160.222:/etc/bluetooth/fileName.txt .
```

   c. Open the received file and verify the data.

5. Disconnect operation from the remote device SPP client application, and stop the SPP server in DUT by running the following command:

```
stop_server
```

Note: All devices of a particular development kit have the same default MAC address.

Perform Bluetooth SPP client functions

You can perform SPP client functions using the `spp_client_menu` options.

Before you begin, set up the device and go to the SPP client menu as described in Set up device for Bluetooth SPP functions.

Configure DUT in SPP client role

To configure the DUT in an SPP client role, do the following:

1. Run the SPP server in the remote device SPP application.
2. Initiate an SPP profile connection from DUT by running the following command:

```
connect <bt_addr>
```

Parameters

`<bt_addr>` is the Bluetooth address of the remote device.

Run Bluetooth inquiry scan to get the `<bt_addr>`.

3. Verify the output on the DUT for successful SPP connection.

Sample output

The following sample output shows that the SPP connection has been established successfully:

```
spp_client_menu

****************** Menu ********************
         connect <bt_addr>
         disconnect
         send_file<space><directory_with_file_name>eg: send_file
/var/fileName.txt
         recv_file<space><directory_with_file_name>eg: recv_file
/var/fileName.txt
```

```
        send_data<space><with_size>eg: send_data 1000[Note:1000
means 1MB]
        recv_data
        main_menu
****************************************
connect 22:22:85:3d:2e:50
Connecting Device ... Please wait ...!!!
ACL state:0 change with reason 00 for device: 22:22:85:3d:2e:50
Device is Connected
-----------------------
Device Address       : 22:22:85:3d:2e:50
Connection Direction : Client
```

Note: If you face any permission issues in the `/var` directory, use the `/etc/bluetooth` directory to send or receive files.

Send a file in the client role

To send a file in the client role, do the following:

1. Set the remote device in Receive mode in the SPP application, if applicable.
2. Create a test file with some sample data and place the file in the `/etc/bluetooth` directory of the DUT.

   For example, create a test file `bt_stack.conf`.
3. Send the test file to the remote device by running the following command:

   ```
   send_file <directory_with_file_name>
   ```

   Parameters

   `<directory_with_file_name>` is the complete file path along with the filename.

   Example

   To send the test file `bt_stack.conf` at `/etc/bluetooth`, run the following command:

```
send_file /etc/bluetooth/bt_stack.conf
```

Sample output

```
send_file /etc/bluetooth/bt_stack.conf
File Transfer Complete
----------------------
Device Address : 22:22:85:3d:2e:50
File Name      : /etc/bluetooth/bt_stack.conf
```

Note: If an issue occurs, disconnect from the device and resume the process.

4. Verify the status of the file reception on the remote application.

Receive a file in client role

To receive a file in the client role, do the following:

1. Set the DUT in Receive mode.
2. Receive a file from the remote device by running the following command:

```
recv_file <directory_with_file_name>
```

Parameters

`<directory_with_file_name>` is the complete file path along with the filename.

Note: To receive a file, use the `/etc/bluetooth` directory only.

Example

To receive `fileName.txt` at `/etc/bluetoot`` from the remote device, run the following command:

```
recv_file /etc/bluetooth/fileName.txt
```

3. Send the file from the remote device SPP server application.
4. Verify the received file as follows:
   a. Open the command prompt window.
   b. Pull the received file by running the following command:

```
scp -r root@<IP_address>:<source_file_path> <destination_
file_path>
```

To pull a file to the current file path, enter the `<destination_file_path>` as `.` in the command.

Note: When prompted for a password, enter `oelinux123` to authenticate the file transfer through the Secure Copy Protocol (SCP).

Example

The IP address of the device is `10.92.160.222`. To pull `fileName.txt` from `/etc/bluetooth`, run the following command:

```
scp -r root@10.92.160.222:/etc/bluetooth/fileName.txt .
```

c. Open the received file and verify the data.

5. Disconnect the SPP connection from the DUT by running the following command:

```
disconnect
```

6.3 General Attribute Profile

GATT is a service framework that uses the ATT to discover services, and to read and write characteristic values on a peer device.

To perform Bluetooth Low Energy GATT server or client functions, you must first complete the steps in the following procedure.

Set up device for Bluetooth Low Energy GATT functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

```
ssh root@<device_IP_address>
```

Example

If the IP address of the device is `10.92.160.222`, run the following command:

```
ssh root@10.92.160.222
```

2. Connect to the SSH by entering the following password:

```
oelinux123
```

3. Open the Bluetooth test application by running the following command:

```
btapp
```

Sample output

```
sh-5.1# btapp
get_ap_interface
:: get_ap_interface

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
*****************************************
```

4. Enable Bluetooth as follows:

   a. Go to the **Gap Menu** by running the following command:

   ```
   gap_menu
   ```

   b. Enable Bluetooth by running the following command:

   ```
   enable
   ```

   Sample output

```
gap_menu

****************** Menu ********************
         enable
         disable
         inquiry
         cancel_inquiry
         get_role_req<space><bt_address>     eg. get_role_req
00:11:22:33:44:55
         pair<space><bt_address><space><transport>     eg.
pair 00:11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
         unpair<space><bt_address>     eg. unpair 00:11:22:
33:44:55
         inquiry_list
         bonded_list
         get_state
         get_bt_name
         get_bt_address
         set_bt_name<space><bt name>     eg. set_bt_name MDM_
Fluoride
         set_scan_mode<space><scan mode value (range 0-2)>
 eg. set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_
CONNECTABLE,2-BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
         set_afh<space><AFH_Host_Channel_Classification>
eg. set_afh 112233445566778899f0
         send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd
01,04,05,33,8b,9e,0a,00 - For Inquiry
         read_clock<space><which_clock range(0-1)><space><bt_
address>     eg. read_clock 0(local)/1(acl connection) 00:11:
22:33:44:55
         main_menu
         switch_role_req<bt_address><space><new_role>     eg.
switch_role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:
33:44:55 1
****************************************
enable
current State = 0, new state = 1
BT State is ON
```

c. Return to the **Main Menu** by running the following command:

```
main_menu
```

5. Go to the GATT server or client menu, as required.

- To go to the GATT server menu, run the following command:

```
gattstest_menu
```

Sample output

```
main_menu
****************** Menu *******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
   ****************************************
gattstest_menu

****************** Menu *******************
         gattstest_init_server (only for Init time)
         gattstest_addservers
         gattstest_addservices<space><server instance><space>
<service instance>
         gattstest_init_advertiser initialzes advertiser
         gattstest_start_advertiser<space><server instance>
         gattstest_readphy<space><remote address><server
instance>
         gattstest_set_preferred_phy<space><remote address>
<space><server instance><space><tx phy><space><rx phy><space>
<phy opt>
         gattstest_stop<space><server_instance>
         gattstest_disable
         gattstest_cancel_connection<space><remote address>
         gattstest_unregister_server<space><server instance>
         main_menu
   ****************************************
```

For GATT server functions, see Perform Bluetooth Low Energy GATT server functions.

- To go to the GATT client menu, run the following command:

```
gattctest_menu
```

Sample output

```
main_menu

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
   ****************************************
gattctest_menu

***************** Menu ******************
         gattctest_init (only for Init time)
         gattctest_scanset<space><scan_type><space><value>
     eg: scanType: 0-NO_SET,1-SCAN_MODE,2-CB_Type,3-RESULT_
TYPE,4-PHY,5-LEGACY,6-REPORT_DELAY,7-NUM_RESPONSE
         gattctest_scanFilter<space><filter_type><space>
<filter_Value>         eg: filterType: 0-NO_FILT,1-FILT_BD_
ADDR,2-FILT_DEV_NAME,3-FILT_SRVC_UUID
         gattctest_scanFilter_manData<space><manuId><space>
<ManuData><space><ManuMask>
         gattctest_start_scan
         gattctest_stop_scan
         gattctest_batch_scan  0-FULL MODE 1- TRUNCATED MODE
         main_menu
         gattctest_conn_params<space><isAuto><space><phy>
<space><isOppur>         eg: isAuto(0/1);phy (0-255 (0 bit:
1M(1); 1bit:2M(2); 2bit:Coded(4); or any combination);
isOppur(0/1))
         gattctest_connect<space><bt_address><space>
<transport>         eg. gattctest_connect 00:11:22:33:44:55
0(Auto)/1(BREDR)/2(LE)
```

```
         gattctest_disconnect<space><bt_address>
eg.gattctest_connect 00:11:22:33:44:55
         gattctest_discsrvc<space><bdaddr>  discovering
services
         gattctest_rdchar_uuid<space><bdaddr><space><uuid>
     eg: reading char by uuid
         gattctest_readPhy<space><bt_address>
         gattctest_readrssi<space><bt_address>
         gattctest_reqMtu<space><bt_address><space><value>
         gattctest_refresh<space><bt_address>
         gattctest_setphy<space>          <TxValue(0-255,(0
bit:1M(1); 1bit:2M(2); 2bit:Coded(4); or any combination))>
<space><RxValue(0-255)><space><PhyOpt(0:no pref,1:s2,2:s8)>
<space><bt_address>
         gattctest_getservices<space><bt_address>
         gattctest_reqconn_pri<space><bt_address><space>
<priority 0/1/2>
         gattctest_getcharid<space><bt_address><space>
<instanceid>
         gattctest_reliablewrite<space><bt_address><space>
<instanceid>
         gattctest_getdescid<space><bt_address><space>
<instanceid>
         gattctest_getsrvc<space><bt_address><space><UUID>
<space><INSTANCEID>
         gattctest_RdWrDesc<space><bt_address><space><R-2/W-
1><space><value><space><INSTANCEID><space><value length>
         gattctest_RdWrchar<space><bt_address><space><R-2/W-
1><space><value><space><INSTANCEID><space><value length>
         gattctest_conn_dev
         gattctest_register_notifications<space><bt_address>
<space><CHARINSTANCEID><space><DESCINSTANCEID><space><E-1/D-
0>
   ****************************************
```

For GATT client functions, see Perform Bluetooth Low Energy GATT client functions.

Next steps

Perform Bluetooth Low Energy GATT server functions

You can perform Bluetooth Low Energy GATT server functions using the `gattstest_menu` options.

Before you begin:

1. Configure the Bluetooth Low Energy GATT server in the command prompt as follows:
   a. Pull the server configuration file, `ServerConfigFile.txt`, by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/ServerConfigFile.txt
   <destination_file_path>
   ```

   To pull a file to the current file path, enter the `<destination_file_path>` as `.` in the command.

   Note: When prompted for a password, enter `oelinux123` to authenticate a file transfer through the Secure Copy Protocol (SCP).

   Example

   The IP address of the device is `10.92.160.222`. To pull the `ServerConfigFile.txt` file from the `/etc/bluetooth/` directory of the device, run the following command:

   ```
   scp -r root@10.92.160.222:/etc/bluetooth/ServerConfigFile.txt
   .
   ```

   For more information about the server configuration file, see GATT server configuration parameters.

   b. Configure or add services, characteristics, and descriptors for the servers in the file, as required.

   c. Push the modified server configuration file to the device by running the following command:

   ```
   scp -r ServerConfigFile.txt root@<IP_address>:/etc/bluetooth/
   ServerConfigFile.txt
   ```

   Example

   The IP address of the device is `10.92.160.222`. To push the `ServerConfigFile.txt` file back to the device, run the following command:

```
scp -r ServerConfigFile.txt root@10.92.160.222:/etc/
bluetooth/ServerConfigFile.txt
```

This command ensures that the newly added or modified parameters reflect in the device.

2. Configure the Bluetooth Low Energy GATT advertisement in the command prompt as follows:

   a. Pull the advertiser configuration file, `AdvertiserConfigFile.txt`, by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/AdvertiserConfigFile.
   txt <destination_file_path>
   ```

   To pull a file to the current file path, enter the `<destination_file_path>` as `.` in the command.

   Example

   The IP address of the device is `10.92.160.222`. To pull the `AdvertiserConfigFile.txt` file from the `/etc/bluetooth/` directory of the device, run the following command:

   ```
   scp -r root@10.92.160.222:/etc/bluetooth/
   AdvertiserConfigFile.txt .
   ```

   For more information about the advertiser configuration file, see GATT server configuration parameters.

   b. Configure the advertising parameters in the file, as required.

   c. Push the modified advertiser configuration file to the device by running the following command:

   ```
   scp -r AdvertiserConfigFile.txt root@<IP_address>:/etc/
   bluetooth/AdvertiserConfigFile.txt
   ```

   Example

   The IP address of the device is `10.92.160.222`. To push the `AdvertiserConfigFile.txt` file back to the device, run the following command:

   ```
   scp -r AdvertiserConfigFile.txt root@10.92.160.222:/etc/
   bluetooth/AdvertiserConfigFile.txt
   ```

   This command ensures that the modified parameters reflect in the device.

3. Set up the device and go to the GATT server menu as described in Set up device for Bluetooth Low Energy GATT functions.

Start Bluetooth Low Energy GATT server

To start the Bluetooth Low Energy GATT server, do the following:

Note: Ensure that you configure the server configuration file and advertiser configuration file.

1. Read the `ServerConfigFile.txt` file by running the following command:

```
gattstest_init_server
```

Sample output

```
gattstest_menu

***************** Menu ******************
         gattstest_init_server (only for Init time)
         gattstest_addservers
         gattstest_addservices<space><server instance><space>
<service instance>
         gattstest_init_advertiser initialzes advertiser
         gattstest_start_advertiser<space><server instance>
         gattstest_readphy<space><remote address><server
instance>
         gattstest_set_preferred_phy<space><remote address>
<space><server instance><space><tx phy><space><rx phy><space>
<phy opt>
         gattstest_stop<space><server_instance>
         gattstest_disable
         gattstest_cancel_connection<space><remote address>
         gattstest_unregister_server<space><server instance>
         main_menu
*****************************************
gattstest_init_server
ENABLE GATTSTEST
Initializing Gattstest
Reading Server Configuration File ....
File reading Done
```

2. Add a server instance by running the following command:

```
gattstest_addservers
```

3. Add a service to this server instance by running the following command:

```
gattstest_addservices <server instance> <service instance>
```

In this command, the server instance is followed by the service instance that must be added to it. The service number is mentioned in the `ServerConfigFile.txt`. For more information, see GATT server configuration parameters.

Example

To add the service `1` recorded in the `ServerConfigFile.txt` file to the newly created server `1`, run the following command:

```
gattstest_addservices 1 1
```

Sample output

```
gattstest_addservers
Adding Server 1
gattstest_addservices 1 1
AddServices
```

4. Initialize the GATT advertiser by running the following command:

```
gattstest_init_advertiser
```

5. Advertise the server that's added and configured by running the following command:

```
gattstest_start_advertiser <server instance>
```

Example

To advertise a server instance `1` that's added and configured, run the following command:

```
gattstest_start_advertiser 1
```

6. After the device starts advertising, connect it from the Bluetooth Low Energy application on another device.

Sample output

```
gattstest_start_advertiser 1
StartAdvertisement
onAdvertisingSetStarted - Success
The device 22:22:85:3d:2e:50 got connected
ACL state:0 change with reason 00 for device: 22:22:85:3d:2e:50
```

Read and set PHY parameters

To read and set the PHY parameters, do the following:

1. Read the PHY parameters of the current connection by running the following command:

```
gattstest_readphy <remote address> <server instance>
```

Example

The server instance `1` is advertised for a remote device with the address `22:22:85:3d:2e:50`. To read the PHY parameters of this connection, run the following command:

```
gattstest_readphy 22:22:85:3d:2e:50 1
```

Sample output

```
gattstest_readphy 22:22:85:3d:2e:50 1
Read Phy
User input is 22:22:85:3d:2e:50

onPhyRead deviceAddress: 22:22:85:3d:2e:50, txPhy: 1 rxPhy: 1
status: 0
```

2. Set the preferred PHY of the current connection by running the following command:

```
gattstest_set_preferred_phy <remote address> <server instance>
<tx Phy> <rx Phy> <Phy option>
```

For more information about the PHY parameters, see GATT server configuration parameters.

Example

A remote device with the address `22:22:85:3d:2e:50` is connected to the server instance `1`. To set Tx PHY as `1 M`, Rx PHY as `1 M`, and PHY option as `0` for this connection, run the following command:

```
gattstest_set_preferred_phy 22:22:85:3d:2e:50 1 1 1 0
```

Sample output

```
gattstest_set_preferred_phy 22:22:85:3d:2e:50 1 1 1 0
Set Preferred Phy
the user options are address: 22:22:85:3d:2e:50 server_instance:
1 txoption: 1 rxoption: 1 phyoption: 0
```

Disconnect a remote device

To disconnect a remote device, run the following command:

```
gattstest_cancel_connection <remote address>
```

Example

To disconnect a remote device with the address `22:22:85:3d:2e:50`, run the following command:

```
gattstest_cancel_connection 22:22:85:3d:2e:50
```

Sample output

```
gattstest_cancel_connection 22:22:85:3d:2e:50
Cancel Connection
The device 22:22:85:3d:2e:50 got disconnected
ACL state:1 change with reason 16 for device: 22:22:85:3d:2e:50
```

Stop Bluetooth Low Energy GATT server

To stop advertising a Bluetooth Low Energy GATT server instance, run the following command:

```
gattstest_stop <server_instance>
```

Example

To stop the server instance `1`, run the following command:

```
gattstest_stop 1
```

Perform Bluetooth Low Energy GATT client functions

You can perform Bluetooth Low Energy GATT client functions using the `gattctest_menu` options.

Before you begin, set up the device and go to the GATT client menu as described in Set up device for Bluetooth Low Energy GATT functions.

Initialize Bluetooth Low Energy GATT client

To initialize the GATT client test, run the following command:

```
gattctest_init
```

Sample output

```
gattctest_menu

***************** Menu ******************
         gattctest_init (only for Init time)
         gattctest_scanset<space><scan_type><space><value>
eg: scanType: 0-NO_SET,1-SCAN_MODE,2-CB_Type,3-RESULT_TYPE,4-PHY,5-
LEGACY,6-REPORT_DELAY,7-NUM_RESPONSE
         gattctest_scanFilter<space><filter_type><space><filter_
Value>         eg: filterType: 0-NO_FILT,1-FILT_BD_ADDR,2-FILT_DEV_
NAME,3-FILT_SRVC_UUID
         gattctest_scanFilter_manData<space><manuId><space><ManuData>
<space><ManuMask>
         gattctest_start_scan
         gattctest_stop_scan
         gattctest_batch_scan  0-FULL MODE 1- TRUNCATED MODE
         main_menu
         gattctest_conn_params<space><isAuto><space><phy><space>
<isOppur>         eg: isAuto(0/1);phy (0-255 (0 bit:1M(1); 1bit:
2M(2); 2bit:Coded(4); or any combination); isOppur(0/1))
         gattctest_connect<space><bt_address><space><transport>
  eg. gattctest_connect 00:11:22:33:44:55 0(Auto)/1(BREDR)/2(LE)
         gattctest_disconnect<space><bt_address>           eg.
gattctest_connect 00:11:22:33:44:55
         gattctest_discsrvc<space><bdaddr>  discovering services
         gattctest_rdchar_uuid<space><bdaddr><space><uuid>
eg: reading char by uuid
         gattctest_readPhy<space><bt_address>
         gattctest_readrssi<space><bt_address>
         gattctest_reqMtu<space><bt_address><space><value>
         gattctest_refresh<space><bt_address>
         gattctest_setphy<space>          <TxValue(0-255,(0 bit:
1M(1); 1bit:2M(2); 2bit:Coded(4); or any combination))><space>
<RxValue(0-255)><space><PhyOpt(0:no pref,1:s2,2:s8)><space><bt_
address>
         gattctest_getservices<space><bt_address>
         gattctest_reqconn_pri<space><bt_address><space><priority 0/
1/2>
```

```
        gattctest_getcharid<space><bt_address><space><instanceid>
        gattctest_reliablewrite<space><bt_address><space>
<instanceid>
        gattctest_getdescid<space><bt_address><space><instanceid>
        gattctest_getsrvc<space><bt_address><space><UUID><space>
<INSTANCEID>
        gattctest_RdWrDesc<space><bt_address><space><R-2/W-1><space>
<value><space><INSTANCEID><space><value length>
        gattctest_RdWrchar<space><bt_address><space><R-2/W-1><space>
<value><space><INSTANCEID><space><value length>
        gattctest_conn_dev
        gattctest_register_notifications<space><bt_address><space>
<CHARINSTANCEID><space><DESCINSTANCEID><space><E-1/D-0>
****************************************
gattctest_init
BtCmdHandler GATTC_TEST_MENUENABLE GATTCTEST
gattctest not initialized
EnableGATTCTEST done
```

Configure Bluetooth Low Energy GATT client scan settings

To apply Bluetooth Low Energy GATT scan settings such as scan mode, PHY, and legacy options before scanning, run the following command:

```
gattctest_scanset <scan_type> <value>
```

Parameters

- `<scan_type>` is the type of scan.
- `<value>` is the mode of scan.

> **Note:** For more information about scan types and their values, see Scan settings.

If scan settings aren't configured, then the scan is initiated with default settings.

Example

To set `<scan_type>` and `<value>` as `<1>` and `<1>`, respectively, run the following command:

```
gattctest_scanset 1 1
```

Sample output

```
gattctest_scanset 1 1
BtCmdHandler GATTC_TEST_MENUScan settings
Do scan settings
scanSettings Type : 1
SCAN_MODE value : 1
```

Configure Bluetooth Low Energy GATT client scan filter

To apply specific Bluetooth Low Energy GATT scan filters before scanning, run the following command:

```
gattctest_scanFilter <filter_type> <filter_Value>
```

Parameters

- `<filter_type>` is the type of filter.
- `<filter_Value>` is the device name.

If scan settings aren't configured, then the scan is initiated with default settings.

Example

To set `<filer_type>` as `<2>` for a Bluetooth device named `QIPL_BT`, run the following command:

```
gattctest_scanFilter 2 QIPL_BT
```

Sample output

```
gattctest_scanFilter 2 QIPL_BT
BtCmdHandler GATTC_TEST_MENUScan filter
Do scan filtering
FilterType 2
FILTER_DEVICE_NAME
Dev_Name is QIPL_BT
```

Start Bluetooth Low Energy GATT client scan

To start the Bluetooth Low Energy GATT scan, run the following command:

```
gattctest_start_scan
```

If the scan settings are configured, the scan is initiated based on the specified settings and filters. If the scan settings aren't configured, the scan is initiated with default settings.

Note: To get the scan results, the remote device must advertise using any Bluetooth Low Energy application.

Sample output

```
gattctest_start_scan
BtCmdHandler GATTC_TEST_MENUtrying to start scan
starting scan
[INFO:gatt_api.cc(949)] GATT_Register37d79a07-e02f-f98a-21e9-
af96b7f9e01f
INFO:gatt_api.cc(969)] allocated gatt_if=5
VERBOSE1:gatt_api.cc(1077)]_GATT_StartIf gatt_if=+
VERBOSE1:gatt_utils.cc(264)] gatt_find_the_connected_bda start_idx=0
[VERBOSE1:gatt_utils.cc(276)] found=0 found_idx=7
VERBOSE1:btm_ble_gap. cc(1952)] btm_ble_process_adv_addr: bda=1e:44:
9c:50:13:5a
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 1e:44:9c:50:13:5a
The scanned device is 1e:44:9c:50:13:5a
The scanned device is
[VERBOSE1:btm_ble_gap.cc(1952)] btm_ble_process_adv_addr: bda=3d: f0:
ba:09:2b:35
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 3d: f0:ba:09:2b:35
The scanned device is 3d: f0:ba: 09:2b:35
The scanned device is
[VERBOSE1:btm_ble_gap.cc(1952)] btm_ble_process_adv_addr: bda=22:fe:
78:cc:32:f9
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 22:fe:78:cc:32:f9
The scanned device is 22:fe:78:cc:32:f9
The scanned device is
[VERBOSE1:btm_ble_gap.cc(1952)] btm_ble_process_adv_addr: bda=8c: fd:
f0:Of:29:de
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 8c:fd: f0:Of:29:de
The scanned device is 8c:fd:f0:Of:29:de
The scanned device is
[VERBOSE1:btm_ble_gap.cc(1952)] btm_ble_process_adv_addr: bda=3c: 12:
```

```
87: 58:0c:97
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 3c:12:87:58:0c:97
The scanned device is 3c:12:87:58:0c:97
The scanned device is
[VERBOSE1:btm_ble_gap.cc(1952)] btm_ble_process_adv_addr: bda=23:32:
b0:bd: a6:28
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 23:32:b0:bd: a6:28
[VERBOSE1:btm_ble_gap.cc(1952)] btm_ble_process_adv_addr: bda=3b: a0:
94:d1:16:2a
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 3b: a0:94:d1:16:2a
The scanned device is 23:32:b0:bd:a6:28
The scanned device is
The scanned device is 3b:a0:94:d1:16:2a
```

Stop Bluetooth Low Energy GATT client scan

To stop the Bluetooth Low Energy GATT scan, run the following command:

```
gattctest_stop_scan
```

Sample output

```
gattctest_stop_scan
BtCmdHandler GATTC_TEST_MENUstopping scan
stopping scan results
[VERBOSE1:gatt_api.cc(992)] GATT_Deregister gatt_if=5
```

Connect Bluetooth Low Energy GATT client

To start a GATT connection to the remote device, run the following command:

```
gattctest_connect <bt_address> <transport>
```

Parameters

- `<bt_address>` is the address of the remote device.
- `<transport>` is the radio for pairing. The value of the `<transport>` parameter can be:
  - `0`: Auto selection
  - `1`: BR/EDR
  - `2`: Bluetooth Low Energy

Example

The address of the remote device is `44:c8:f4:b6:af:00` and the `<transport>` type is `2`. To connect to the remote device, run the following command:

```
gattctest_connect 44:c8:f4:b6:af:00 2
```

Sample output

```
gattctest_connect 44:c8:f4:b6:af:00 2
BtCmdHandler GATTC_TEST_MENUconnecting
[INFO:gatt_api.cc(949)] GATT_Register82462309-bb44-a4a7-db81-
b204201f49dd
[INFO:gatt_api.cc(969)] allocated gatt_if=5
VERBOSE1:gatt_api.cc(1077)] GATT_StartIf gatt_if=+
VERBOSE1:gatt_utils.cc(264)] gatt_find_the_connected_bda start_idx=0
VERBOSE1:gatt_utils.cc(276)] found=0 found_idx=7
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 44:c8:f4:b6:af:00
INFO:gatt_api.cc(1122)] GATT_Connectgatt_if=5 44:c8:f4:b6:af:00
[VERBOSE1:gatt_main.cc(1217)] gatt_get_ch_state: ch_state=0
[VERBOSE1:gatt_main.cc(1200)] gatt_set_ch_state: old=0 new=o
[VERBOSE1: 12c_api.cc(1673)] L2CA_ConnectFixedChnl BDA: 44:c8:f4:b6:
af:00CID: 0x0004
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 44:c8:f4:b6:af:00
[VERBOSE1:gatt_main.cc(334)] gatt_update_app_use_link_flag: is_add=1
chk_link=1
[VERBOSE1:gatt_main.cc(300)] gatt_update_app_hold_link_status
[VERBOSE1:gatt_main.cc(304)] added gatt_if=5
```

Read Bluetooth Low Energy GATT client current transmitter PHY

To read the Tx and Rx PHY parameters of the GATT connection, run the following command:

```
gattctest_readPhy <bt_address>
```

Parameters

`<bt_address>` is the address of the connected remote device.

Example

The address of the remote device is `66:6b:26:88:df:83`. To read the PHY parameters of the connection, run the following command:

```
gattctest_readPhy 66:6b:26:88:df:83
```

Sample output

```
gattctest_readphy 66:6b:26:88:df:83
BtCmdHandler GATTC_TEST_MENUReading PHY
Readphy Setting initiated
[VERBOSE1:btm_ble.cc(889)] read_phy_cb Received read_phy_cb
Txphy is 1 and RxPhy is 1
```

Set Bluetooth Low Energy GATT client current transmitter PHY

To set the Tx and Rx PHY parameters of the GATT connection, run the following command:

```
gattctest_setphy <tx value> <rx value> <Phy option> <bt_address>
```

Parameters

`<bt_address>` is the address of the connected remote device.

Note: For more information about PHY parameters, see GATT configuration parameters.

Example

The address of the remote device is `4e:0f:49:d9:85:0c`. To set the Tx PHY, Rx PHY, and PHY option of the connection as `1`, `1`, and `0`, respectively, run the following command:

```
gattctest_setphy 1 1 0 4e:0f:49:d9:85:0c
```

Sample output

```
gattctest_setphy 1 1 0 4e:0f:49:d9:85:0c
BtCmdHandler GATTC_TEST_MENUSetting PHY
Txphy is 1 and RxPhy is 1
```

Note: The Tx and Rx PHY values are updated according to the negotiated remote values. These values can vary from the PHY values provided by you while running the `gattctest_setphy` command.

Read the remote device RSSI

To read the RSSI of a remote device, run the following command:

```
gattctest_readrssi <bt_address>
```

Parameters

`<bt_address>` is the address of the connected remote device.

Example

The address of the remote device is `66:6b:26:88:df:83`. To read the RSSI of the device, run the following command:

```
gattctest_readrssi 66:6b:26:88:df:83
```

Sample output

```
gattctest_readrssi 66:6b:26:88:df:83
BtCmdHandler GATTC_TEST_MENUReading RSSI
gattReadRemoteRssi Initiated Success
[VERBOSE1:btm_inq.cc(1019)] BTM_InqDbRead: bd addr 66:6b:26:88:df:83
onReadRemoteRssi RSSI: -51
```

Discover remote device services

To discover the services of the remote device, run the following command:

```
gattctest_discsrvc <bdaddr>
```

Parameters

`<bdaddr>` is the address of the connected remote device.

Example

The address of the remote device is `66:6b:26:88:df:83`. To discover the services of the device, run the following command:

```
gattctest_discsrvc 66:6b:26:88:df:83
```

Sample output

```
gattctest_discsrvc 66:6b:26:88:df:83
BtCmdHandler GATTC_TEST_MENUDiscovering services
================================
=== The service type is 0 InstaceId 1
```

```
== SERVICE uuid is 00001801-0000-1000-8000-00805f9b34fb
The no of Characteristics for thisservice: 1
== == CHAR uuid is 00002a05-0000-1000-8000-00805f9b34fb InstanceID 3
== == Properties 32 ; permissions 0; writeType 2
The no of descriptors for thischar 0
==============================
=== The service type is 0 InstaceId 20
== SERVICE uuid is 00001800-0000-1000-8000-00805f9b34fb
The no of Characteristics for thisservice: 3
== == CHAR uuid is 00002a00-0000-1000-8000-00805f9b34fb InstanceID 22
== == Properties 2 ; permissions 0; writeType 2
The no of descriptors for thischar 0
== == CHAR uuid is 00002a01-0000-1000-8000-00805f9b34fb InstanceID 24
== == Properties 2 ; permissions 0; writeType 2
The no of descriptors for thischar 0
== == CHAR uuid is 00002aa6-0000-1000-8000-00805f9b34fb InstanceID 26
== == Properties 2 ; permissions 0; writeType 2
The no of descriptors for thischar 0
==============================
=== The service type is 0 InstaceId 40
== SERVICE uuid is 0000180d-0000-1000-8000-00805f9b34fb
The no of Characteristics for thisservice: 3
== == CHAR uuid is 00002a37-0000-1000-8000-00805f9b34fb InstanceID 42
== == Properties 16 ; permissions 0; writeType 2
The no of descriptors for thischar 1
== == CHAR uuid is 00002a38-0000-1000-8000-00805f9b34fb InstanceID 45
== == Properties 2 ; permissions 0; writeType 2
The no of descriptors for thischar 0
== == CHAR uuid is 00002a39-0000-1000-8000-00805f9b34fb InstanceID 47
== == Properties 8 ; permissions 0; writeType 2
The no of descriptors for thischar 0
```

Get remote device services

To get the list of services of a remote device, run the following command:

```
gattctest_getservices <bt_address>
```

Parameters

`<bt_address>` is the address of the connected remote device.

Example

The address of the remote device is `66:6b:26:88:df:83`. To get the list of services of the device, run the following command:

```
gattctest_getservices 66:6b:26:88:df:83
```

Sample output

```
gattctest_getservices 66:6b:26:88:df:83
BtCmdHandler GATTC_TEST_MENUgetting services
==============================================
=== The service type is 0 Instance ID is1
== SERVICE uuid is 00001801-0000-1000-8000-00805f9b34fb
The no of Characteristics for this service:1
== == == CHAR uuid is 00002a05-0000-1000-8000-00805f9b34fb InstanceId
: 3
== == == Properties 32 ; permissions 0;writeType 2
==============================================
=== The service type is 0 Instance ID is20
== SERVICE uuid is 00001800-0000-1000-8000-00805f9b34fb
The no of Characteristics for this service:3
== == == CHAR uuid is 00002a00-0000-1000-8000-00805f9b34fb InstanceId
: 22
== == == Properties 2 ; permissions 0;writeType 2
== == == CHAR uuid is 00002a01-0000-1000-8000-00805f9b34fb InstanceId
: 24
== == == Properties 2 ; permissions 0;writeType 2
== == == CHAR uuid is 00002aa6-0000-1000-8000-00805f9b34fb InstanceId
: 26
== == == Properties 2 ; permissions 0;writeType 2
==============================================
=== The service type is 0 Instance ID is40
== SERVICE uuid is 0000180d-0000-1000-8000-00805f9b34fb
The no of Characteristics for this service:3
== == == CHAR uuid is 00002a37-0000-1000-8000-00805f9b34fb InstanceId
: 42
```

```
== == == Properties 16 ; permissions 0;writeType 2
== == == CHAR uuid is 00002a38-0000-1000-8000-00805f9b34fb InstanceId
: 45
== == == Properties 2 ; permissions 0;writeType 2
== == == CHAR uuid is 00002a39-0000-1000-8000-00805f9b34fb InstanceId
: 47
== == == Properties 8 ; permissions 0;writeType 2
==========================================
=== The service type is 0 Instance ID is48
```

Read a characteristic value

To read the characteristic value, run the following command:

```
gattctest_RdWrChar <bt_address> <Read> <Value> <InstanceID> <Value
length>
```

Parameters

- `<bt_address>` is the address of the connected remote device.
- `<Read>` command value is `2`.
- `<Value>` is `0`. As the characteristic value is read from the remote device, the value is `0`.
- `<InstanceID>` is the instance value of the particular characteristic.

  During service discovery, all the services and characteristics are updated with the instance IDs in the command line.
- `<Value Length>` is the length of the characteristic value to be written. For reading the characteristic value, the value length can be `0`.

Example

The address of the remote device is `48:33:48:d9:2c:b0` and the `<InstanceID>` is `9`. To read the characteristic value, run the following command:

```
gattctest_RdWrchar 48:33:48:d9:2c:b0 2 0 9 0
```

Sample output

```
gattctest RdWrchar 48:33:48:d9:2c:b0 2 0 9 0
BtCmdHandler GATTC_TEST_MENUReading writing char
instanceid 9
readcharacteristic Initiated
onCharacteristicRead UUID 00002b2a-0000-1000-8000-00805f9b34fb, value
```

```
is 16~0
```

Write a characteristic value

To write a characteristic value, run the following command:

```
gattctest_RdWrChar <bt_address> <Write> <Value> <InstanceID> <Value
length>
```

Parameters

- `<bt_address>` is the address of the connected remote device.
- `<Write>` command value is `1`.
- `<Value>` is the characteristic value to be written.
- `<InstanceID>` is the instance value of the particular characteristic.

  During service discovery, all the services and characteristics are updated with the instance IDs in the command line.
- `<Value Length>` is the length of the characteristic value to be written.

Example

The address of the remote device is `48:33:48:d9:2c:b0` and the `<InstanceID>` is `56`. To write the characteristic value `78` with value length of `2`, run the following command:

```
gattctest_RdWrchar 48:33:48:d9:2c:b0 1 78 56 2
```

Sample output

```
gattctest_RdWrchar 48:33:48:d9:2c:b0 1 78 56 2
BtCmdHandler GATTC_TEST_MENUReading writing char
instanceid 56
writeCharacteristic value 78
writeCharacteristic success
write characteristic uid 0000aaa2-0000-1000-8000-aabbccddeeff, value:
78 == success
write characteristic: 0 success
```

Read a descriptor value

To read a descriptor value, run the following command:

```
gattctest_RdWrDesc <bt_address> <Read> <Value> <InstanceID> <Value
length>
```

Parameters

- `<bt_address>` is the address of the connected remote device.
- `<Read>` command value is `2`.
- `<Value>` is `0`. As the descriptor value is read from the remote device, the value is `0`.
- `<InstanceID>` is the instance value of the particular descriptor.

  For the instance IDs of the descriptors of a characteristic, see Get characteristic ID.

  During service discovery, all the services and descriptors are updated with the instance IDs in the command line.
- `<Value Length>` is the length of the descriptor value to be written. For reading the descriptor value, the value length can be `0`.

Example

The address of the remote device is `48:33:48:d9:2c:b0` and the `<InstanceID>` is `52`. To read the descriptor value, run the following command:

```
gattctest_RdWrDesc 48:33:48:d9:2c:b0 2 0 52 0
```

Sample output

```
gattctest_RdWrDesc 48:33:48:d9:2c:b0 2 0 52 0
BtCmdHandler GATTC_TEST_MENUReading writing DESC
instanceid 52
readDescriptor Initiated
DESCRIPTOR VALUE is #4Vx
```

Write a descriptor value

To write a descriptor value, run the following command:

```
gattctest_RdWrDesc <bt_address> <Write> <Value> <InstanceID> <Value
length>
```

Parameters

- `<bt_address>` is the address of the connected remote device.
- `<Write>` command value is `1`.
- `<Value>` is the descriptor value to be written.
- `<InstanceID>` is the instance value of the particular descriptor.

  For the instance IDs of the descriptors of a characteristic, see Get characteristic ID.

  During service discovery, all the services and descriptors are updated with the instance IDs in the command line.
- `<Value Length>` is the length of the descriptor value to be written.

Example

The address of the remote device is `48:33:48:d9:2c:b0` and the `<InstanceID>` is `52`. To read the descriptor value `7` with value length of `1`, run the following command:

```
gattctest_RdWrDesc 48:33:48:d9:2c:b0 1 7 52 1
```

Sample output

```
gattctest_RdWrDesc 48:33:48:d9:2c:b0 1 7 52 1
BtCmdHandler GATTC_TEST_MENUReading writing DESC
instanceid 52
onDescriptorWrite Completed 0
onDescriptorWrite Success
```

Set connection priority

To set the connection priority for a particular remote device, run the following command:

```
gattctest_reqconn_pri <bt_address> <priority>
```

Parameters

- `<bt address>` is the address of the connected remote device.
- `<priority>` can be `0`, `1`, or `2`.

Example

To set the connection priority of the remote device with the address `66:6b:26:88:df:83` as `1`, run the following command:

```
gattctest_reqconn_pri 66:6b:26:88:df:83 1
```

Sample output

```
gattctest_reqconn_pri 66:6b:26:88:df:83 1
BtCmdHandler GATTC_TEST_MENURequesting Connection Priority
Requested Connection priority
onConnectionUpdated interval (12), latency (0), timeout (2000),
status (0)
```

Get characteristic ID

To get the details of a characteristic, run the following command:

```
gattctest_getcharid <bt_address> <instanceID>
```

This command displays the details of the characteristic such as the characteristic UUID, properties, permissions, the number of descriptors, and the UUID and instance IDs of the descriptors.

Parameters

- `<bt address>` is the address of the connected remote device.
- `<instanceID>` is the instance ID of the characteristic.

Example

The address of the remote device is `54:8b:83:4a:1f:4e`. To get the characteristic with instance ID `74`, run the following command:

```
gattctest_getcharid 54:8b:83:4a:1f:4e 74
```

Sample output

```
gattctest_getcharid 54:8b:83:4a:1f:4e 74
BtCmdHandler GATTC_TEST_MENUgetting Characteristic
== == == CHAR uuid is ffffeeee-0000-1000-8000-00805f9b34fb InstanceId
: 74
== == == Properties 10 ; permissions 0 ; writeType 2
The no of descriptors for this char: 1
@@@@@@ desc uuid is bbbb1111-0000-1000-8000-00705f9b34fb InstanceId :
75
```

```
****** permissions 0 ; value (null)
```

Get descriptor ID

To get the details of a descriptor, run the following command:

```
gattctest_getdescid <bt address> <instanceID>
```

Parameters

- `<bt address>` is the address of the connected remote device.
- `<instanceID>` is the instance ID of the descriptor.

Example

The address of the remote device is `54:8b:83:4a:1f:4e`. To get the descriptor with instance ID `75`, run the following command:

```
gattctest_getdescid 54:8b:83:4a:1f:4e 75
```

Sample output

```
gattctest_getdescid 54:8b:83:4a:1f:4e 75
BtCmdHandler GATTC_TEST_MENUgetting Descriptor
@@@@@@ desc uuid is bbbb1111-0000-1000-8000-00705f9b34fb InstanceId :
75
```

Disconnect Bluetooth Low Energy GATT client

To disconnect from the remote device, run the following command:

```
gattctest_disconnect <bt_address>
```

Parameters

`<bt address>` is the address of the remote device that you intend to disconnect.

Example

The address of the remote device is `69:d0:d6:99:80:75`. To disconnect the device, run the following command:

```
gattctest_disconnect 69:d0:d6:99:80:75
```

Sample output

```
gattctest_disconnect 69:d0:d6:99:80:75
BtCmdHandler GATTC_TEST_MENUdisconnecting
[INFO:gatt_api.cc(1225)] GATT_Disconnect conn_id=5
[VERBOSE1:gatt_main.cc(334)] gatt_update_app_use_link_flag: is_add=0
chk_link=1
[VERBOSE1:gatt_main.cc(300)] gatt_update_app_hold_link_status
[VERBOSE1:gatt_main.cc(317)] removed_gatt_if=5
```

GATT configuration parameters

To configure the GATT server and client functions, use the server, client, and scan parameters.

Parameters	Values

GATT server configuration parameters

The two types of server configuration parameters used to perform Bluetooth Low Energy GATT server functions are: advertiser and server parameters. These configuration parameters are managed through `AdvertiserConfigFile.txt` and `ServerConfigFile.txt` files, respectively.

`AdvertiserConfigFile.txt`

The following table lists the possible values of each advertisement parameter in the `AdvertiserConfigFile.txt` file.

Parameters	Values
`TxPower`	**Legacy:** • `0`: ADVERTISE_TX_POWER_ULTRA_LOW(-21) • `1`: ADVERTISE_TX_POWER_LOW(-15) • `2`: ADVERTISE_TX_POWER_MEDIUM(-7) • `3`: ADVERTISE_TX_POWER_HIGH(1) **Extended:** • `0`: TX_POWER_ULTRA_LOW =-21 • `1`: TX_POWER_LOW =-15 • `2`: TX_POWER_MEDIUM =-7 • `3`: TX_POWER_HIGH = 1
`LegacyFlag`	• `0`: Use extended advertiser • `1`: Use legacy advertiser
`PeriodicFlag`	• `0`: Disable periodic advertiser • `1`: Enable periodic advertiser
`ConnectableFlag`	• `0`: Enable non-connectable advertiser • `1`: Enable connectable advertiser

Parameters	Values
`ScannableFlag`	• `0`: Enable non-scannable advertiser • `1`: Enable scannable advertiser This value is applicable only to the extended advertisers. For the legacy advertisers, always set it to `true`.
`AnonymousFlag`	• `0`: Use the advertiser address in all PDUs • `1`: Don't use the advertiser address in PDUs
`IncludePower`	• `0`: Don't include Tx power in the extended header • `1`: Include Tx power in the extended header
`PrimaryPhy`	• `1`: 1 M-PHY • `3`: Low Energy (LE) coded PHY
`SecondaryPhy`	• `1`: 1 M-PHY • `2`: 2 M-PHY • `3`: LE coded PHY
`Interval`	• `160`: Minimum advertisement interval (100 ms) • `16777215`: Maximum advertisement interval (10,485 s)
`TimeOutLegacy`	• `0`: Minimum possible value • `180000`: Maximum possible timeout value in ms
`AdvertiseMode`	• `0`: Low-power mode • `1`: Balanced mode • `2`: Low latency mode (used for legacy) The advertisement interval is selected based on the advertiser mode as follows: • Low-power mode: 1 s advertisement interval • Balanced mode: 250 ms advertisement interval • Low latency mode: 100 ms advertisement interval

Parameters	Values
BLEBtName	The name to be displayed for that particular advertiser set.

The following snippet shows an example of an advertisement set:

```
******************AdvertisingSet1***************
TxPower:1
LegacyFlag:1
PeriodicFlag:0
ConnectableFlag:1
ScannableFlag:1
AnonymousFlag:0
IncludePower:1
PrimaryPhy:1
SecondaryPhy:1
Interval: 320
TimeOutLegacy:0
AdvertiseMode:1
BLEBtName:Adv1
```

`ServerConfigFile.txt`

The following table lists the possible values of each service parameter in the `ServerConfigFile.txt` file.

Parameter	Description
service uuid	The service ID. A server can have multiple services.
characteristic uuid	The ID of the characteristic. A service can have multiple characteristics. A characteristic can have properties and permissions.

Parameter	Description

`characteristic properties`	The properties of the characteristic.
	Combine the following characteristic properties with a logical OR argument, as required:
	`PROPERTY_BROADCAST = 0x01;`
	`PROPERTY_READ = 0x02;`
	`PROPERTY_WRITE_NO_RESPONSE = 0x04;`
	`PROPERTY_WRITE = 0x08;`
	`PROPERTY_NOTIFY = 0x10;`
	`PROPERTY_INDICATE = 0x20;`
	`PROPERTY_SIGNED_WRITE = 0x40;`
	`PROPERTY_EXTENDED_PROPS = 0x80;`

`characteristic permissions`	The permissions of the characteristic.
	Combine the following characteristic permissions with a logical OR argument, as required:
	`PERMISSION_READ = 0x01;`
	`PERMISSION_READ_ENCRYPTED = 0x02;`
	`PERMISSION_READ_ENCRYPTED_MITM = 0x04;`
	`PERMISSION_WRITE = 0x10;`
	`PERMISSION_WRITE_ENCRYPTED = 0x20;`
	`PERMISSION_WRITE_ENCRYPTED_ MITM = 0x40;`
	`PERMISSION_WRITE_SIGNED = 0x80;`
	`PERMISSION_WRITE_SIGNED_MITM = 0x100;`

`descriptor uuid`	The ID of the descriptor. A characteristic can have multiple descriptors. A descriptor can have permissions.
`descriptor permissions`	The permissions of the descriptor.

The following snippet shows an example of a service in the GATT server.

```
**************Server1*************
Service1:[service uuid],[characteristic uuid],[characteristic
properties],[characteristic permissions],[descriptor uuid],
[descriptor permissions]
Service1:00001200-0000-1000-8000-00805f9b34fb, 00001101-0000-1000-
8000-00805f9b34fb,26,31,00002a06-0000-1000-8000-00805f9b34fb,17
ManufacturerId:0
ManufacturerData:AAAA
**************End of service*************
```

Parameter	Values	Notes

GATT client configuration parameters

The following table lists the connection parameters used to configure the GATT client.

Parameter	Values	Notes
`isAuto`	• `0`: Connects directly • `1`: Connects automatically as soon as a remote device is available	If this parameter is set to `1`, ensure that the devices are bonded for auto connection.
`isOpportunistic`	• `0`: Connects directly • `1`: Provides the handle if it's already connected based on the address and transport	If this parameter is set to `1`, it doesn't start the connection if it's not already present.
`Tx_Phy`	• `1`: 1 M-PHY • `2`: 2 M-PHY • `4`: LE coded PHY • `7`: All PHYs	
`Rx_Phy`	• `1`: 1 M-PHY • `2`: 2 M-PHY • `4`: LE coded PHY • `7`: All PHYs	
`Phy_Opt:00`	• `0`: Host preferred coding scheme • `1`: Use S = 2 (500 Kbps) coding scheme • `2`: Use S = 8 (125 Kbps) coding scheme. Use it only when the PHY is LE coded.	

Parameter	Values	Notes
`Mtu_Size`	• The minimum value is `23`. • The maximum value is `512`.	

Scan settings

The following table lists the scan settings used to configure the GATT client.

Scan types	Values	Notes
`1: Scan_mode`	• `-1`: Opportunistic mode • `0`: Low-power mode • `1`: Balanced mode • `2`: Low latency mode **Regular scan:** • `0`: Scan window (512 ms); scan interval (5120 ms) • `1`: Scan window (1024 ms); scan interval (4096 ms) • `2`: Scan window (4096 ms); scan interval (4096 ms) **Batch scan:** • `0`: Scan window (1500 ms); scan interval (150,000 ms) • `1`: Scan window (1500 ms); scan interval (15,000 ms) • `2`: Scan window (1500 ms); scan interval (5000 ms)	
`2: CallbackType`	• `1`: Initiates a callback to all the matched advertisements • `2`: Initiates a callback to the first matched advertisement • `4`: Initiates a callback when a device detected earlier stops sending advertisements.	

Scan types	Values	Notes
`3: ResultType`	• `0`: Provides full scan results that have the device, RSSI, advertising data, scan response and, scan timestamps. • `1`: Provides scan results that have the device, RSSI, and scan timestamps.	
`4: ScanPhy`	• `1`: 1 M-PHY • `3`: LE coded PHY. Use it for extended scanning only. • `255`: All supported PHYs	Currently, it's hard coded to 255 in the application code.
`5: Legacy`	• `True`: Legacy scanning • `False`: Extended scanning	
`6: ReportDelay`	• `0`: Deliver reports immediately • `>0`: Accumulate scan results and deliver them after the delay mentioned (in ms) or if buffers are full (batch scan).	
`7: MatchMode`	• `1`: Aggressive match mode. The hardware finds a match even with feeble signal strength. `2`: Sticky match mode. It requires a higher threshold of signal strength.	

Scan types	Values	Notes
`8: NumOfAdvMatches`	• `1`: Matches one advertisement per filter • `2`: Matches few advertisements according to availability of hardware resources • `3`: Matches as many advertisements as the hardware allows	

6.4 Advanced Audio Distribution Profile

A2DP defines how to stream multimedia audio from one device to another over a Bluetooth connection. This mechanism is also called as Bluetooth audio streaming.

To perform A2DP source or sink functions, you must first complete the steps in the following procedure.

Set up device for A2DP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Enable A2DP source or sink role on the device, as required.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth test application by running the following command:

```
btapp
```

Sample output

```
sh-5.1# btapp
get_ap_interface
:: get_ap_interface

****************** Menu ********************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
****************************************
```

4. Enable Bluetooth and pair the device as follows:

   a. Go to the **Gap Menu** by running the following command:

```
gap_menu
```

   b. Enable Bluetooth by running the following command:

```
enable
```

Sample output

```
gap_menu

****************** Menu ********************
         enable
         disable
         inquiry
         cancel_inquiry
         get_role_req<space><bt_address>     eg. get_role_req
```

```
00:11:22:33:44:55
         pair<space><bt_address><space><transport>     eg.
pair 00:11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
         unpair<space><bt_address>     eg. unpair 00:11:22:
33:44:55
         inquiry_list
         bonded_list
         get_state
         get_bt_name
         get_bt_address
         set_bt_name<space><bt name>     eg. set_bt_name MDM_
Fluoride
         set_scan_mode<space><scan mode value (range 0-2)>
 eg. set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_
CONNECTABLE,2-BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
         set_afh<space><AFH_Host_Channel_Classification>
eg. set_afh 1122334455667788990f0
         send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd
01,04,05,33,8b,9e,0a,00 - For Inquiry
         read_clock<space><which_clock range(0-1)><space><bt_
address>     eg. read_clock 0(local)/1(acl connection) 00:11:
22:33:44:55
         main_menu
         switch_role_req<bt_address><space><new_role>     eg.
switch_role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:
33:44:55 1
****************************************
enable
current State = 0, new state = 1
BT State is ON
```

c. Pair the device by running the following command:

```
pair <bt_address> <transport>
```

To accept the outgoing/incoming pairing, enter `yes`.

Parameters

- Run Bluetooth inquiry scan to get the `<bt_address>` of the remote device.
- The values of the `<transport>` parameter can be:
  - `0`: Auto selection
  - `1`: BR/EDR

– `2`: Bluetooth Low Energy

Example

To initiate a BR/EDR pairing for `<bt_address>` `98:09:cf:a9:82:23`, run the following command:

```
pair 98:09:cf:a9:82:23 1
```

Sample output

```
pair 98:09:cf:a9:82:23 1
BR/EDR Bonding
ACL state:0 change with reason 00 for device: f8:7d:76:9d:9b:
6b
**********************************************
Pairing state for MyDeviceA is BOND NONE
**********************************************
BT pairing request :: MyDeviceB :: Pairing Code :: 776996
**********************************************
**Please enter yes / no **
yes
**********************************************
Pairing state for MyDeviceB is BONDED
**********************************************
```

d. Return to the **Main Menu** by running the following command:

```
main_menu
```

5. Go to the A2DP source or sink menu, as required.

   • To go to the **A2DP Source Menu**, run the following command from the `main_menu`:

```
a2dp_source_menu
```

Sample output

```
main_menu

****************** Menu ********************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
```

```
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
   ****************************************
a2dp_source_menu

***************** Menu ******************
         connect<space><bt_address>
         disconnect<space><bt_address>
         codec_list<space><codec1,param1,param2....,codec2,
param1,param2....>
         avrcp_trackchange
         avrcp_now_playing_content_changed
         avrcp_setabsolutevol<space><volstep>  eg:
setabsolutevol 10 (range 0-15)
         avrcp_sendvolupdown<space><1/0>  eg: sendvolupdown 1
(1-up, 0-down)
         avrcp_availplayerchange
         avrcp_biggermetadata info:enables bigger avrcp
metadata
         avrcp_setequalizerval<space><val> (1/2)
         avrcp_setrepeatval<space><val> (1 to 4)
         avrcp_setshuffleval<space><val>(1 to 3)
         avrcp_setscanval<space><val> (1 to 3)
         set_scmst_cp_flag<space><bd_addr><space><0-2> (0-
Copyrighted 1-Only Once 2-Content not protected)
         main_menu
   ****************************************
```

For A2DP source menu functions, see Perform Bluetooth A2DP source functions.

- To go to the **A2DP Sink Menu**, run the following command from the `main_menu`:

```
a2dp_sink_menu
```

Sample output

```
main_menu

***************** Menu ******************
         gap_menu
```

```
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
   ****************************************
a2dp_sink_menu

***************** Menu ******************
         connect<space><bt_address>
         disconnect<space><bt_address>
         play<space><bt_address>
         pause<space><bt_address>
         stop<space><bt_address>
         rewind<space><bt_address>
         fastforward<space><bt_address>
         forward<space><bt_address>
         backward<space><bt_address>
         power<space><bt_address>
         volup<space><bt_address>
         voldown<space><bt_address>
         volchangednoti<space><vol level(0-15)>
         mute<space><bt_address>
         codec_list<space><codec1,param1,param2,codec2,
param1,param2,....>
         getcap<space><bt_address><space><cap_ID>
         listplayersettingattr<space><bt_address>
         listplayersettingvalue<space><bt_address><space>
<attri_ID>
         getplayersetting<space><bt_address><space><attri_
IDs>
         setplayersetting<space><bt_address><space><attri_
IDs><space><attri_Values>
         getelementattr<space><bt_address><space><num_attrb>
<space><attribute_IDs>
         getplayerstatus<space><bt_address>
         regnotification<space><bt_address><space><event_ID>
```

```
         setaddressedplayer<space><bt_address><space><player_
ID>
         setbrowsedplayer<space><bt_address><space><player_
ID>
         changepath<space><bt_address><space><direction>
<space><folder_uID>
         getfolderitems<space><bt_address><space><scopeID>
<space><startItem><space><endItem><space><num_attrb><space>
<attrib_IDs>
         getitemattributes<space><bt_address><space><scopeID>
<space><uID><space><uID_Counter><space><num_attrb><space>
<attrib_IDs>
         playitem<space><bt_address><space><scopeID><space>
<uID><space><uID_Counter>
         addtonowplaying<space><bt_address><space><scopeID>
<space><uID><space><uID_Counter>
         search<space><bt_address><space><length><space>
<string>
         main_menu
   ****************************************
```

For A2DP sink menu functions, see Perform Bluetooth A2DP sink functions.

Enable A2DP source role

To enable the A2DP source role on the device, do the following:

1. Open the command prompt.
2. Pull the `bt_app.conf` file from the device by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/bt_app.conf .
   ```

   `<IP_address>` is the IP address of the device.
3. Configure the following values in the `bt_app.conf` file:
   - `BtA2dpSinkEnable=false`
   - `BtA2dpSinkSplitEnable=false`
   - `BtHfClientEnable=false`
   - `BtA2dpSourceEnable=true`
4. Save the `bt_app.conf` file.
5. Push the `bt_app.conf` file to the device by running the following command:

```
scp -r bt_app.conf root@<IP_address>:/etc/bluetooth/bt_app.conf
```

`<IP_address>` is the IP address of the device.

Enable A2DP sink role

To enable the A2DP sink role on the device, do the following:

1. Open the command prompt.
2. Pull the `bt_app.conf` file from the device by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/bt_app.conf .
   ```

   `<IP_address>` is the IP address of the device.
3. Configure the following values in the `bt_app.conf` file:
   - `BtA2dpSinkEnable=true`
   - `BtA2dpSinkSplitEnable=true`
   - `BtHfClientEnable=true`
   - `BtA2dpSourceEnable=false`
4. Save the `bt_app.conf` file.
5. Push the `bt_app.conf` file to the device by running the following command:

   ```
   scp -r bt_app.conf root@<IP_address>:/etc/bluetooth/bt_app.conf
   ```

   `<IP_address>` is the IP address of the device.

Next steps

Perform Bluetooth A2DP source functions

You can perform A2DP source functions using the `a2dp_source_menu` options.

Before you begin, set up the device and go to the A2DP source menu as described in Set up device for A2DP functions.

Connect to an A2DP source device

To connect to an A2DP source device, run the following command:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a remote device with the Bluetooth address `20:19:d8:36:90:40`, run the following command:

```
connect 20:19:d8:36:90:40
```

Sample output

```
connect 20:19:d8:36:90:40
A2DP Source Connecting to 20:19:d8:36:90:40
ACL state:0 change with reason 00 for device: 20:19:d8:36:90:40
Codec Configuration for device 20:19:d8:36:90:40
Codec type = sbc
Sample Rate = 48000
Bits per sample = 16
Channel Mode = joint
Block Len = 16
Num of Subbands = 8
Allocation Method = loudness
Max Bitpool = 53
Min Bitpool = 2
A2DP Source Connected to 20:19:d8:36:90:40
param is device_connection=trueBT a2dp source connect success
percent volume changed: 81
```

Set the sampling rate of codecs

To set the sampling rate of codecs, run the following command:

```
codec_list <codec1,param1,param2....,codec2,param1,param2....>
```

Parameters

`<codec1,param1,param2....,codec2,param1,param2....>` is the list of codecs with respective parameters.

Example

- To set the sampling rate of SBC and AAC codecs to 48 kHz, run the following command:

```
[codec_list sbc,48,16,joint,16,8,loud,250,2 ] [codec_list aac,
48,16,stereo,128,195000 ]
```

- To set the sampling rate of SBC and AAC codecs to 44.1 kHz, run the following command:

```
[codec_list sbc,44.1,16,joint,16,8,loud,250,2 ] [codec_list aac,
44.1,16,stereo,128,195000 ]
```

Play songs from the DUT

To play songs from the DUT, do the following:

1. Open a command prompt window.
2. Push the WAV or MP3 files to the DUT.
3. Play the audio as follows:

Note: The DUT doesn't support an audio player. Use `paplay` commands to play music.

- To play PCM samples, run the following command:

```
paplay <wav_filepath> --device=<device_name>
```

Parameters

– `<wav_filepath>` is the filepath of the WAV file to play.

– `<device_name>` is the name of the playback device.

- To play compressed audio files like MP3, run the following command:

```
paplay -d offload0 --encoding=<format> --raw <audio_filepath>
```

Parameters

– `<audio_filepath>` is the filepath of the MP3 file to play.

– `<format>` is the audio file format.

Example

To play an MP3 audio file at `/tmp/Chogada.mp3`, run the following command:

```
paplay -d offload0 --encoding=mpeg --raw /tmp/Chogada.mp3
```

Sample output

```
sh-5.1# paplay -d offload0 --encoding=mpeg --raw /tmp/
Chogada.mp3
sh-5.1# paplay -d offload0 --encoding=mpeg --raw /tmp/
Chogada.mp3
```

Set the absolute volume

To set the absolute volume, run the following command:

```
avrcp_setabsolutevol <volstep>
```

Parameters

`<volstep>` is the level of volume. It ranges from 0 to 15.

Example

To set the absolute volume to level 15, run the following command:

```
avrcp_setabsolutevol 15
```

Sample output

```
avrcp_setabsolutevol 0
percent volume changed: 0
avrcp_setabsolutevol 10
percent volume changed: 66
avrcp_setabsolutevol 15
percent volume changed: 100
avrcp_setabsolutevol 10
percent volume changed: 66
avrcp_setabsolutevol 7
percent volume changed: 47
```

Note: To set the absolute volume, the DUT and the remote device must support the absolute

volume functionality.

Increase the volume

To increase the volume by one, run the following command:

```
avrcp_sendvolupdown 1
```

Sample output

```
avrcp_sendvolupdown 1
percent volume changed: 53
avrcp_sendvolupdown 1
percent volume changed: 59
```

Note: To increase the volume, the DUT and the remote device must support the volume increase functionality.

Decrease the volume

To decrease the volume by one, run the following command:

```
avrcp_sendvolupdown 0
```

Sample output

```
avrcp_sendvolupdown 0
percent volume changed: 53
```

Note: To decrease the volume, the DUT and the remote device must support the volume

decrease functionality.

Disconnect an A2DP source device

To disconnect an A2DP source device, run the following command:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a remote device with the Bluetooth address `20:19:d8:36:90:40`, run the following command:

```
disconnect 20:19:d8:36:90:40
```

Sample output

```
disconnect 20:19:d8:36:90:40
A2DP Source DisConnecting: 20:19:d8:36:90:40
A2DP Source DisConnected
param is device_connection=falseBT a2dp source disconnect success
ACL state:1 change with reason 13 for device: 20:19:d8:36:90:40
```

Perform Bluetooth A2DP sink functions

You can perform A2DP sink functions using the `a2dp_sink_menu` options.

Before you begin, set up the device and go to the A2DP sink menu as described in Set up device for A2DP functions.

Connect to an A2DP sink device

To connect to an A2DP sink device, run the following command:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
connect f8:7d:76:9d:9b:6b
```

Sample output

```
connect f8:7d:76:9d:9b:6b
No session exists for the usecase
The server answered: obj path: '/org/pulseaudio/ext/pal/loopback/ses_
1'
BT a2dp connect success
ACL state:0 change with reason 00 for device: f8:7d:76:9d:9b:6b
Codec Configuration for device f8:7d:76:9d:9b:6b
Codec type = SBC
Sample Rate = 44100
Channel Mode = 2
A2DP Sink Connected to f8:7d:76:9d:9b:6b
   AVRCP_CTRL_CONNECTED_CB
<-- getcap rsp message received!
   num_supported:1, rsp_type: 12
   CompanyIDO: 0x6488
AVRCP_CTRL_BR_CONNECTED_CB
<-- getcap rsp message received!
   num_supported:7, rsp_type: 12
   SupportedEvent0: 0x1
   SupportedEvent1: 0x2
   SupportedEvent2: 0x8
   SupportedEvent3: 0x9
   SupportedEvent4: 0x10
   SupportedEvent5: 0x11
   SupportedEvent6: 0x12
<-- notification message received!
   event_id:1, rsp_type:0
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x0
<-- notification message received!
   event_id:2, rsp_type:0
   EVENT_TRACK_CHANGED track: 0xff
<-- notification message received!
   event_id:8, rsp_type:0
   EVENT_APP_SETTING_CHANGED attr_id0: 0x2 attr_value0: 0x1
   EVENT_APP_SETTING_CHANGED attr_id1: 0x3 attr_valuel: 0x1
<-- notification message received!
   event_id:9, rsp_type:0
   EVENT_NOW_PLAYING_CHANGED
```

```
<-- notification message received!
   event_id:10, rsp_type:0
   EVENT_AVAL_PLAYERS_CHANGED
<-- notification message received!
   event_id:11, rsp_type:0
   EVENT_ADDR_PLAYER_CHANGED player_id: 0x1 uid_counter: 0x0
^[S Processing event 441 in state 1
current State = 1, new state = 2
```

Play an audio file

To play an audio file on the remote device, run the following command:

```
play <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To play an audio file on a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
play f8:7d:76:9d:9b:6b
```

Sample output

```
*********************************************
play f8:7d:76:9d:9b:6b
*********************************************
Recieved Start from Src device
*********************************************
<-- notification message received!
   event_id:1, rsp_type:1
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x1
<-- notification message received!
   event_id:2, rsp_type:1
   EVENT_TRACK_CHANGED track: 0x0
<-- notification message received!
   event_id:1, rsp_type:0
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x1
<-- notification message received!
   event_id:2, rsp_type:0
   EVENT_TRACK_CHANGED track: 0x0
```

```
btsink_enable=1 success
```

Pause audio playback

To pause audio playback, run the following command:

```
pause <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To pause audio playback on a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
pause f8:7d:76:9d:9b:6b
```

Sample output

```
pause f8:7d:76:9d:9b:6b
<-- notification message received!
   event_id:1, rsp_type:1
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x0
<-- notification message received!
   event_id:2, rsp_type:1
   EVENT_TRACK_CHANGED track: 0xff
<-- notification message received!
   event_id:1, rsp_type:0
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x0
<-- notification message received!
   event_id:2, rsp_type:0
   EVENT_TRACK_CHANGED track: 0xff
************************************************
Recieved Suspend from Src device
************************************************
btsink enable=0 success
```

Stop audio playback

To stop audio playback, run the following command:

```
stop <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To stop audio playback on a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
stop f8:7d:76:9d:9b:6b
```

Sample output

```
stop f8:7d:76:9d:9b:6b
<-- notification message received!
   event_id:1, rsp_type:1
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x0
<-- notification message received!
   event_id:2, rsp_type:1
   EVENT_TRACK_CHANGED track: 0xff
<-- notification message received!
   event_id:1, rsp_type:0
   EVENT_PLAYBACK_STATUS_CHANGED play_status: 0x0
<-- notification message received!
   event_id:2, rsp_type:0
   EVENT_TRACK_CHANGED track: 0xff
```

Rewind an audio playback

To rewind an audio playback, run the following command:

```
rewind <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To rewind audio playback on a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
rewind f8:7d:76:9d:9b:6b
```

Increase the volume

To increase the volume by one, run the following command:

```
volup <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To increase the volume of a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
volup f8:7d:76:9d:9b:6b
```

Decrease the volume

To decrease the volume by one, run the following command:

```
voldown <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To decrease the volume of a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
voldown f8:7d:76:9d:9b:6b
```

Set volume level

To set the volume level, run the following command:

```
volchangednoti <vol level(0-15)>
```

Parameters

`<vol level(0-15)>` is the level of volume. It ranges from 0 to 15.

Example

To set the absolute volume to level 10, run the following command:

```
volchangednoti 10
```

Sample output

```
volchangednoti 10
btsink_volume=10 set successfully
volchangednoti 15
btsink_volume=15 set successfully
```

Mute the volume

To mute the volume, run the following command:

```
mute <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To mute the volume of a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
mute f8:7d:76:9d:9b:6b
```

Set codec priority

To set the codec priority of the sink, run the following command:

```
codec_list <codec1,param1,param2,codec2,param1,param2,....>
```

Parameters

`<codec1,param1,param2,codec2,param1,param2,....>` is the order of priority.

Example

To set the first priority as SBC 44.1 kHz and the second priority as SBC 48 kHz, run the following command:

```
codec_list sbc,44.1,sbc,48
```

Sample output

```
codec_list sbc,44.1,sbc,48
Codec List: sbc,44.1,sbc,48
num_codec_configs = 2
```

Note: The priority of codecs is applied to new connections. Hence, set the codec priority before establishing a connection.

Other functions

The following player-related settings depend on the capabilities of the remote device player. You can configure or explore these settings based on your remote device player.

- Get and configure player settings
- Know the current player status
- Add a track to a playlist
- Get details about the current track
- Search for an audio file

6.5 Hands-Free Profile

HFP defines how an audio gateway device can connect to a hands-free device for functions like remote control and audio connection.

To perform HFP client or audio gateway functions, you must first complete the steps in the following procedure.

Set up the device for HFP functions

Prerequisites

- Enable SSH to access your host device. For instructions, see Sign in using SSH.
- Enable HFP client or audio gateway role on the device, as required.
- Place the DUT and the remote device in the Bluetooth vicinity.

Procedure

1. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

2. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

3. Open the Bluetooth test application by running the following command:

```
btapp
```

Sample output

```
sh-5.1# btapp
get_ap_interface
:: get_ap_interface

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
*****************************************
```

4. Enable Bluetooth and pair the device as follows:
   a. Go to the **Gap Menu** by running the following command:

   ```
   gap_menu
   ```

   b. Enable Bluetooth by running the following command:

   ```
   enable
   ```

   Sample output

   ```
   gap_menu

   ***************** Menu ******************
            enable
            disable
            inquiry
            cancel_inquiry
            get_role_req<space><bt_address>     eg. get_role_req
   00:11:22:33:44:55
   ```

```
         pair<space><bt_address><space><transport>     eg.
pair 00:11:22:33:44:55 0(auto)/1(BREDR)/2(BLE)
         unpair<space><bt_address>     eg. unpair 00:11:22:
33:44:55
         inquiry_list
         bonded_list
         get_state
         get_bt_name
         get_bt_address
         set_bt_name<space><bt name>     eg. set_bt_name MDM_
Fluoride
         set_scan_mode<space><scan mode value (range 0-2)>
 eg. set_scan_mode 0 --0-BT_SCAN_MODE_NONE,1- BT_SCAN_MODE_
CONNECTABLE,2-BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE
         set_afh<space><AFH_Host_Channel_Classification>
eg. set_afh 11223344556677889f0
         send_hci_cmd<space><hci_cmd>     eg. send_hci_cmd
01,04,05,33,8b,9e,0a,00 - For Inquiry
         read_clock<space><which_clock range(0-1)><space><bt_
address>     eg. read_clock 0(local)/1(acl connection) 00:11:
22:33:44:55
         main_menu
         switch_role_req<bt_address><space><new_role>     eg.
switch_role_req 00:11:22:33:44:55 0 or get_role_req 00:11:22:
33:44:55 1
****************************************
enable
current State = 0, new state = 1
BT State is ON
```

c. Pair the device by running the following command:

```
pair <bt_address> <transport>
```

To accept the outgoing/incoming pairing, enter `yes`.

Parameters

- Run Bluetooth inquiry scan to get the `<bt_address>` of the remote device.
- The values of the `<transport>` parameter can be:
  - `0`: Auto selection
  - `1`: BR/EDR
  - `2`: Bluetooth Low Energy

Example

To initiate a BR/EDR pairing for `<bt_address> 98:09:cf:a9:82:23`, run the following command:

```
pair 98:09:cf:a9:82:23 1
```

Sample output

```
pair 98:09:cf:a9:82:23 1
BR/EDR Bonding
ACL state:0 change with reason 00 for device: f8:7d:76:9d:9b:
6b
**********************************************
Pairing state for MyDeviceA is BOND NONE
**********************************************
BT pairing request :: MyDeviceB :: Pairing Code :: 776996
**********************************************
**Please enter yes / no **
yes
**********************************************
Pairing state for MyDeviceB is BONDED
**********************************************
```

d. Return to the **Main Menu** by running the following command:

```
main_menu
```

5. Go to the HFP client or audio gateway menu, as required.

   - To go to the **HFP Client Menu**, run the following command from the `main_menu`:

```
hfp_client_menu
```

Sample output

```
main_menu

***************** Menu ******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
```

```
        a2dp_source_menu
        spp_client_menu
        spp_server_menu
        eslap_menu
        exit
   ****************************************
hfp_client_menu
***************** Menu ******************
        connect<space><bt_address>
        disconnect<space><bt_address>
        create_sco<space><bt_address>
        destroy_sco<space><bt_address>
        accept_call
        reject_incoming_call
        end_active_call
        hold_active_call
        unhold_held_call
        reject_incoming_call_when_active_call_present
        end_held_call
        end_active_call_and_accept_waiting_or_held_call
        swap_calls
        add_held_call_to_conference
        end_specified_active_call<space><index of the call>
        private_consultation_mode<space><index of the call>
        put_incoming_call_on_hold_when_no_other_call_present
        accept_held_incoming_call_when_no_other_call_present
        reject_held_incoming_call_when_no_other_call_present
        dial<space><phone_number>
        redial
        dial_memory<space><memory_location>
        start_voice_recognition
        stop_voice_recognition
        query_current_calls
        query_operator_name
        query_subscriber_info
        mic_volume_control<space><value>
        speaker_volume_control<space><value>
        send_dtmf<space><code>
        disable_nrec_on_AG
        main_menu
   ****************************************
```

For HFP client menu functions, see Perform Bluetooth HFP client functions.

- To go to the **HFP Audio Gateway Menu**, run the following command from the `main_menu`:

```
hfp_ag_menu
```

Sample output

```
main_menu

****************** Menu *******************
         gap_menu
         test_menu
         a2dp_sink_menu
         hfp_client_menu
         gattctest_menu
         gattstest_menu
         hogp_menu
         hfp_ag_menu
         a2dp_source_menu
         spp_client_menu
         spp_server_menu
         eslap_menu
         exit
*******************************************
hfp_ag_menu
****************** Menu ******************
         connect<space><bt_address>
         disconnect<space><bt_address>
         create_sco<space><bt_address>
         destroy_sco<space><bt_address>
         voip_call_ind<space><bt_address>
         end_voip_call<space><bt_address>
         main_menu
*******************************************
```

For HFP audio gateway menu functions, see Perform Bluetooth HFP audio gateway functions.

Enable HFP client role

To enable the HFP client on the device, do the following:

1. Open the command prompt.
2. Pull the `bt_app.conf` file from the device by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/bt_app.conf .
   ```

   `<IP_address>` is the IP address of the device.
3. Configure the following values in the `bt_app.conf` file:
   - `BtA2dpSinkEnable=true`
   - `BtA2dpSinkSplitEnable=true`
   - `BtHfClientEnable=true`
   - `BtA2dpSourceEnable=false`
4. Save the `bt_app.conf` file.
5. Push the `bt_app.conf` file to the device by running the following command:

   ```
   scp -r bt_app.conf root@<IP_address>:/etc/bluetooth/bt_app.conf
   ```

   `<IP_address>` is the IP address of the device.

Enable HFP audio gateway role

To enable the HFP audio gateway role on the device, do the following:

1. Open the command prompt.
2. Pull the `bt_app.conf` file from the device by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/bt_app.conf .
   ```

   `<IP_address>` is the IP address of the device.
3. Configure the following values in the `bt_app.conf` file:
   - `BtHfpAGEnable=true`
   - `BtA2dpSinkEnable=false`
   - `BtA2dpSinkSplitEnable=false`
   - `BtHfClientEnable=false`
4. Save the `bt_app.conf` file.

5. Push the `bt_app.conf` file to the device by running the following command:

```
scp -r bt_app.conf root@<IP_address>:/etc/bluetooth/bt_app.conf
```

`<IP_address>` is the IP address of the device.

Next steps

Perform Bluetooth HFP client functions

You can perform HFP client functions using the `hfp_client_menu` options.

Before you begin, set up the device and go to the HFP client menu as described in Set up the device for HFP functions.

Connect to an HFP client device

To connect to an HFP client device, run the following command:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
connect f8:7d:76:9d:9b:6b
```

Sample output

```
connect f8:7d:76:9d:9b:6b
Processing event 405 in state 1
connecting with device f8:7d:76:9d:9b:6b
current State = 1, new state = 2
Processing event 441 in state 2
network state is available
no call is in progress
no call in setup
```

```
battery level is 3
signal level is 2
AG is in home network
no held call
Processing event 439 in state 2
in-band ringtone provided
Processing event 442 in state 2
SLC connected with device f8:7d:76:9d:9b:6b
current State = 2, new state = 3
```

Create an SCO link

To create an SCO link, run the following command:

```
create_sco <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To create an SCO link for a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
create_sco f8:7d:76:9d:9b:6b
```

Accept an incoming call

To accept an incoming call, run the following command:

```
accept_call
```

Sample output

```
accept_call
Processing event 409 in state 4
a call is in progress
no call in setup
Processing event 439 in state 4
```

Reject an incoming call

To reject an incoming call, run the following command:

```
reject_incoming_call
```

Sample output

```
reject_incoming_call
Processing event 410 in state 4
no call in setup
Processing event 439 in state 4
Processing event 444 in state 4
current mode = 2, new mode = 0
current State = 4, new state = 3
set param hfp_enable=false success
BT disconnect is success for SCO usecase
Disconnected SCO connection with device f8:7d:76:9d:9b:6b
```

End an active call

To end an active call, run the following command:

```
end_active_call
```

Sample output

```
end_active_call
Processing event 411 in state 4
Processing event 444 in state 4
current mode = 2, new mode = 0
current State = 4, new state = 3
no call is in progress
set param hfp_enable=false success
BT disconnect is success for SCO usecase
Disconnected SCO connection with device f8:7d:76:9d:9b:6b
```

Dial a phone number

To dial a phone number, run the following command:

```
dial <phone_number>
```

Parameters

<phone_number> is the phone number that you intend to dial.

Sample output

```
dial 7123456789
Processing event 422 in state 3
Outgoing call is in setup
Processing event 447 in state 3
SCO/eSCO connected with device f8:7d:76:9d:9b:6b
Connected state: Requesting for focus
current mode = 0, new mode = 2
current State = 3, new state = 4
Processing event 252 in state 4
earlier status = 1 new status = 2
Configure audio for SCO
No session exists for the usecase
The server answered: obj path: '/org/pulseaudio/ext/pal/loopback/ses_
2'
BT connect is success for SCO usecase
set param- hfp_enable=true success
Outgoing call in alerting state
```

Redial a phone number

To redial a phone number, run the following command:

```
redial
```

Sample output

```
redial
Processing event 423 in state 3
Outgoing call is in setup
Processing event 447 in state 3
SCO/eSCO connected with device f8:7d:76:9d:9b:6b
Connected state: Requesting for focus
current mode = 0, new mode = 2
```

```
current State = 3, new state = 4
Processing event 252 in state 4
earlier status = 1 new status = 2
Configure audio for SCO
No session exists for the usecase
The server answered: obj path: '/org/pulseaudio/ext/pal/loopback/ses_
2'
BT connect is success for SCO usecase
set param- hfp_enable=true success
Outgoing call in alerting state
no call in setup
Processing event 439 in state 4
Processing event 444 in state 4
current mode = 2, new mode = 0
current State = 4, new state = 3
set param hfp_enable=false success
BT disconnect is success for SCO usecase
Disconnected SCO connection with device f8:7d:76:9d:9b:6b
```

Hold a call

To hold a call, run the following command:

```
hold_active_call
```

Sample output

```
hold_active_call
Processing event 412 in state 4
a call is on hold, no active calls
```

Unhold a call

To unhold a call, run the following command:

```
unhold_held_call
```

Sample output

```
unhold_held_call
Processing event 412 in state 4
no held call
```

End a call on hold

To end a call on hold, run the following command:

```
end_held_call
```

Sample output

```
end_held_call
Processing event 413 in state 4
Processing event 444 in state 4
current mode = 2, new mode = 0
current State = 4, new state = 3
set param hfp_enable=false success
BT disconnect is success for SCO usecase
Disconnected SCO connection with device f8:7d:76:9d:9b:6b
no call is in progress
```

Reject an incoming call during an active call

To reject an incoming call during an active call, run the following command:

```
reject_incoming_call_when_active_call_present
```

Start voice recognition

To start voice recognition, run the following command:

```
start_voice_recognition
```

Sample output

```
start_voice_recognition
Processing event 425 in state 3
volume_change_cb : speaker volume is 4
Processing event 433 in state 3
Processing event 447 in state 3
SCO/eSCO connected with device f8:7d:76:9d:9b:6b
Connected state: Requesting for focus
current mode = 0, new mode = 2
current State = 3, new state = 4
Processing event 252 in state 4
earlier status = 1 new status = 2
```

```
Configure audio for SCO
No session exists for the usecase
The server answered: obj path: '/org/pulseaudio/ext/pal/loopback/ses_
2'
BT connect is success for SCO usecase
set param- hfp_enable=true success
signal level is 2
```

Stop voice recognition

To stop voice recognition, run the following command:

```
stop_voice_recognition
```

Sample output

```
stop_voice_recognition
Processing event 426 in state 4
Processing event 444 in state 4
current mode = 2, new mode = 0
current State = 4, new state = 3
set param hfp_enable=false success
BT disconnect is success for SCO usecase
Disconnected SCO connection with device f8:7d:76:9d:9b:6b
```

Get the operator name

To get the operator name, run the following command:

```
query_operator_name
```

Sample output

```
query_operator_name
Processing event 429 in state 3
operator name is IND airtel
```

Get subscriber information

To get subscriber information, run the following command:

```
query_subscriber_info
```

Sample output

```
query_subscriber_info
Processing event 430 in state 3
subscriber name is 917123456789 type is voice
```

Disconnect an HFP client device

To disconnect an HFP client device, run the following command:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a remote device with the Bluetooth address `f8:7d:76:9d:9b:6b`, run the following command:

```
disconnect f8:7d:76:9d:9b:6b
```

Sample output

```
disconnect f8:7d:76:9d:9b:6b
Processing event 406 in state 3
Disconnecting with device f8:7d:76:9d:9b:6b
current State = 3, new state = 2
Processing event 437 in state 2
Disconnected from or Unable to connect with device f8:7d:76:9d:9b:6b
current State = 2, new state = 1
```

Other functions

The HFP client menu offers other functions that enable you to:

- Swap calls
- Verify conference calls
- Access recently dialed phone numbers
- Control the volume of the speaker and microphone
- Send DTMF code

Perform Bluetooth HFP audio gateway functions

You can perform HFP audio gateway functions using the `hfp_ag_menu` options.

Before you begin, set up the device and go to the HFP audio gateway menu as described in Set up the device for HFP functions.

Connect to an audio gateway

To connect to an audio gateway, run the following command:

```
connect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To connect to a remote device with the Bluetooth address `20:19:d8:36:90:40`, run the following command:

```
connect 20:19:d8:36:90:40
```

Sample output

```
connect 20:19:d8:36:90:40
HandleHfpAGConnand cnd_id = 22
BtHfpAgMsgHandler event = 687
AG: Processing event = 687
state_disconnected_handler Processing event 687
connecting with device 28:19:d8:36:90:40 BtHfpAgMsgHandler event =
629
AG: Processing event = 629
state_pending_handler Processing event 629
```

```
connected with device 20:19:d8:36:90:48 wbs_callback
BtHfpAgMsgHandler event = 654
AG: Processing event = 654
state_connected_handler Processing event = 65Wat_cind_callback
BtHfpAgMsgHandler event = 657
AG: Processing event = 657
state_connected_handler Processing event = 657Sending CIND resp to
device 20:19:d8:36:90:40
BtHfpAgMsgHandler event = 630
AG: Processing event = 630
state_connected_handler Processing event = 630SLC connected with
device 28:19:d8:36:90:40 noice_reduction_callback
BtHfpAgMsgHandler event = 653
AG: Processing event = 653
state_connected_handler Processing event = 653volune_control_callback
: speaker volume is 12
BtHfpAgMsgHandler event = 650
AG: Processing event = 650
state_connected_handler Processing event = 650unknown_at_callback
BtHfpAgMsgHandler event = 660
AG: Processing event = 660
state_connected_handler Processing event = 660unknown_at_callback
BtHfpAgMsgHandler event = 668
AG: Processing event = 668
state_connected_handler Processing event = 668at_clcc_callback
BtHfpAgMsgHandler event = 659
AG: Processing event = 659
state_connected_handler Processing event = 659Sending CLCC resp to
device 20:19:d8: 36:90:40unknown_at_callback
BtHfpAgMsgHandler event = 660
AG: Processing event = 660
```

Disconnect an audio gateway

To disconnect an audio gateway, run the following command:

```
disconnect <bt_address>
```

Parameters

`<bt_address>` is the Bluetooth address of the remote device.

Example

To disconnect a remote device with the Bluetooth address `20:19:d8:36:90:40`, run the following command:

```
disconnect 20:19:d8:36:90:40
```

Sample output

```
disconnect 20:19:d8:36:90:40
HandleHfpAGComnand cnd_id = 23
BtHfpAgMsgHandler event = 688
AG: Processing event = 608
state_connected_handler Processing event = 688Disconnecting with
device 20:19:d8:36:90:40 BtHfpAgMsgHandler event = 632
AG: Processing event = 632
state_pending_handler Processing event 632
Disconnected from or Unable to connect with device 20:19:d8:36:90:40
Wrong option selected
```

7 Debug Bluetooth issues

To debug Bluetooth issues, use host logs or over-the-air (OTA) logs. Host logs include Logcat and BTSnoop (stack)/HCI logs. You can use these logs for initial debugging.

The following table lists the logs required to debug different Bluetooth issues.

Issue type	Logcat	BTSnoop/HCI	Kernel logs	OTA	PulseAudio	Subsystem restart (SSR) logs
General	Required	Required	Not required	Recommended	Not required	Not required
Bluetooth controller	Not required	Recommended	Not required	Required	Not required	Required
Bluetooth on/off	Required	Not required	Not required	Not required	Not required	Required
Crash	Required	Not required	Required	Not required	Not required	Required
Audio	Not required	Required	Not required	Required	Required	Not required

7.1 Debug BlueZ stack

Before you enable or collect logs, do the following:

1. Enable SSH to access your host device. For instructions, see Sign in using SSH.
2. Place the DUT and the remote device in the Bluetooth vicinity.
3. Run the SSH in command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

4. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

Enable BlueZ logs

To enable BlueZ logs, do the following in SSH:

1. Enable read and write permissions by running the following command:

```
mount -o remount rw /
```

2. Open the Bluetooth service file in a text editor.

   Example

   To open the `bluetooth.service` file in a VI editor, run the following command:

```
vi /lib/systemd/system/bluetooth.service
```

3. Append `-d` option to the following line in the file:

```
ExecStart=/usr/libexec/bluetooth/bluetoothd -d
```

4. Save the file.
5. Reload the Bluetooth daemon by running the following command:

```
systemctl daemon-reload
```

6. Restart Bluetooth by running the following command:

```
systemctl restart bluetooth
```

Collect BlueZ logs

- To collect BlueZ user space logs, run the following command in SSH:

```
journalctl -f -u bluetooth.service > /var/log/bluetooth.log
```

- To collect all logs, run the following command in SSH:

```
journalctl -f > /var/log/full.log
```

Collect snoop logs

To collect snoop logs, do the following in SSH:

1. Run the `hcidump` tool.
2. Save the file by running the following command:

```
hcidump --save-dump=<filename.log>
```

Example

To save the snoop logs to `snoop.log` file, run the following command:

```
hcidump --save-dump=/var/log/snoop.log
```

Enable PulseAudio logs

To enable PulseAudio logs, do the following in SSH:

1. Verify the status of PulseAudio by running the following command:

```
systemctl status pulseaudio
```

Sample output

```
sh-5.1# systemctl status pulseaudio
* pulseaudio.service - PulseAudio Sound Service
   Loaded: loaded (/lib/systemd/system/pulseaudio.service;
enabled; vendor preset: enabled)
   Active: active (running) since Sun 1980-01-06 00:00:10 UTC;
27min ago
   Main PID: 1607 (pulseaudio)
   Tasks: 14 (limit: 6234)
   Memory: 19.8M
   CGroup: /system.slice/pulseaudio.service
           '- 1607 /usr/bin/pulseaudio -- system --
daemonize=no -v

Jan 06 00:00:11 qcm6490 pulseaudio[1607]: session_close: 1376
exit, ret e
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: freeFrontEndIds: 7243:
stream ty .. .110
Jan 06 00:80:11 qcm6490 pulseaudio[1607]: close: 514: Enter.
deviceCount 1 ... ER)
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: Reset path: speaker-
vbat
```

```
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: close: 528: Exit.
deviceCount e ... s 0
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: GetConcurrencyInfo:
5471: stream ... wed
Jan 06 00:80:11 qcm6490 pulseaudio[1607]: GetConcurrencyInfo:
5471: stream ... wed
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: GetConcurrencyInfo:
5471: stream ... wed
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: deregisterStream:
3690: stream type 1
Jan 06 00:00:11 qcm6490 pulseaudio[1607]: pal_stream_close: 306:
Exit. status 0
Hint: Some lines were ellipsized, use -l to show in full.
sh-5.1#
```

2. Open the PulseAudio service file in a text editor.

   Example

   To open the `pulseaudio.service` file in a VI editor, run the following command:

   ```
   vi /lib/systemd/system/pulseaudio.service
   ```

3. Append `-vvvv` option to the following line in the file:

   ```
   /usr/bin/pulseaudio --system --daemonize=no -vvvv
   ```

4. Save the file.
5. Reload the Bluetooth daemon by running the following command:

   ```
   systemctl daemon-reload
   ```

6. Restart PulseAudio by running the following command:

   ```
   systemctl restart pulseaudio
   ```

Enable logs of ofono service

To enable logs of ofono service, do the following in SSH:

1. Verify the status of the ofono service by running the following command:

```
systemctl status ofono
```

Sample output

```
sh-5.1# systemctl status ofono
* ofono.service - Telephony service
   Loaded: loaded (/lib/systemd/system/ofono.service; enabled;
vendor preset: enabled)
   Active: active (running) since Sun 1980-01-06 00:00:00 UTC;
30min ago
   Main PID: 990 (ofonod)
   Tasks: 1 (limit: 6234)
   Memory: 3.8M
   CGroup: /system.slice/ofono.service
           `- 990 /usr/sbin/ofonod -n

Jan 06 00:00:00 qcm6490 systemd[1]: Started Telephony service.
sh-5.1# |
```

2. Open the ofono service file in a text editor.

Example

To open the `ofono.service` file in a VI editor, run the following command:

```
vi /lib/systemd/system/ofono.service
```

3. Append `-d` option to the following line in the file:

```
ExecStart=/usr/sbin/ofonod -n -d
```

4. Save the file.
5. Reload the Bluetooth daemon by running the following command:

```
systemctl daemon-reload
```

6. Restart ofono service by running the following command:

```
systemctl restart ofono
```

7. Validate the status of the service to verify if it's running with the specified flags.

Collect SSR dump logs

In Qualcomm Linux software, if any Bluetooth subsystem module crashes, the SSR functionality collects the required firmware crash dumps in the Bluetooth driver. It then restarts the Bluetooth subsystem. The crash dump file is at the `/var/spool/crash/` directory.

Consider the following points about SSR logging:

- To verify if SSR logs are enabled or disabled, run the following command:

```
cat /sys/class/devcoredump/disabled
```

  Return:

  – `1`: SSR logging is disabled.

  – `0`: SSR logging is enabled.

- To enable SSR logs, run the following command:

```
echo 0 > /sys/class/devcoredump/disabled
```

- To collect SSR logs manually, do the following:

Note: To collect the SSR crash dump manually, ensure that the device Bluetooth is on.

1. Run the `hcitool` tool.
2. Send the crash command to the controller by running the following command:

```
hcitool cmd 0x3f 0c 26
```

Sample output

```
hcitool cmd 0x3f 0c 26
< HCI Command: ogf 0x3f, ocf 0x000c, plen 1
   26
> HCI Event: 0xff plen 246
   01 09 A5 7A 01 02 84 BD 4C 00 00 00 00 7C 0C 01 10 0C 8B
00
   00 00 02 00 61 0C 8B 00 00 0C FC 00 00 AC 08 02 00 B8 04
02
   00 08 08 08 08 09 09 09 09 10 10 10 10 11 11 11 11 04 00
00
   00 80 FF 06 00 CC DE 06 00 67 0C 01 10 4A 0F 01 10 00 00
00
   00 02 00 00 00 A0 E8 DE 06 00 0C 8B 00 00 0C FC 00 00 AC
```

```
08
   02 00 84 16 02 00 3F 42 01 10 04 00 00 00 06 00 00 00 0C
8B
   00 00 0C FC 00 00 3F 00 00 00 BD AE 02 00 04 00 00 00 FF
FF
   00 00 0C 8B 00 00 0C FC 00 00 3F 00 00 00 B1 BB 0D 10 90
FD
   01 00 90 2B 01 00 10 36 01 00 4C 09 02 00 90 FD 01 00 15
C6
   0D 10 90 2B 01 00 49 E2 02 10 00 00 00 00 00 00 00 10 35
D5
   A5 2D 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02
   02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02
   02 02 02 02 02 02
```

The crash triggered generates a BIN file at the `/var/spool/crash/` directory.

Sample output

```
ls -l /var/spool/crash/
total 904
-r--------. 1 root root 458888 Jan  1 00:53 hci0_1970-01-01_
00-53-36.bin
-r--------. 1 root root 458888 Jan  1 00:58 hci0_1970-01-01_
00-58-18.bin
```

3. Ensure that the BIN file is generated at the `/var/spool/crash/` directory.

Note:

- Only the advanced build version supports the SSR feature.
- If you observe any timeout or Bluetooth hardware error, contact Qualcomm and share the BIN file.

7.2 Debug Fluoride stack

To debug Bluetooth issues in software with the Fluoride stack, use host logs or OTA logs.

Host logs

Prerequisites

Ensure that you enable SSH to access your host device. For instructions, see Sign in using SSH.

Procedure

To get the Bluetooth logs from the DUT, do the following:

1. Turn off Bluetooth on the device.
2. Run SSH in the command prompt using the following command:

   ```
   ssh root@<device_IP_address>
   ```

   Example

   If the IP address of the device is `10.92.160.222`, run the following command:

   ```
   ssh root@10.92.160.222
   ```

3. Connect to the SSH by entering the following password:

   ```
   oelinux123
   ```

4. Get the Logcat logs as follows:

   a. Pull the Bluetooth stack configuration file, `bt_stack.conf`, from the device by running the following command:

   ```
   scp -r root@<IP_address>:/etc/bluetooth/bt_stack.conf
   <destination_file_path>
   ```

   To pull a file to the current file path of the PC, enter the `<destination_file_path>` as `.` in the command.

   Note: When prompted for a password, enter `oelinux123` to authenticate a file transfer through the Secure Copy Protocol (SCP).

   Example

   The IP address of the device is `10.92.160.222`. To pull the `bt_stack.conf` file from the device, run the following command:

   ```
   scp -r root@10.92.160.222:/etc/bluetooth/bt_stack.conf .
   ```

   b. Change the log level to verbose by configuring the following parameters in the Bluetooth stack configuration file, `bt_stack.conf`:

```
TRC_HCI=6
TRC_L2CAP=6
TRC_RFCOMM=6
TRC_OBEX=6
TRC_AVCT=6
TRC_AVDT=6
TRC_AVRC=6
TRC_AVDT_SCB=6
TRC_AVDT_CCB=6
TRC_A2D=6
TRC_SDP=6
TRC_SMP=6
TRC_BTAPP=6
TRC_BTIF=6
TRC_BNEP=6
TRC_PAN=6
TRC_HID_HOST=6
TRC_HID_DEV=6
```

c. Push the Bluetooth stack configuration file to the device by running the following command:

```
scp -r bt_stack.conf root@<IP_address>:/etc/bluetooth/bt_
stack.conf
```

Example

The IP address of the device is `10.92.160.222`. To push the `bt_stack.conf` file to the device, run the following command:

```
scp -r bt_stack.conf root@10.92.160.222:/etc/bluetooth/bt_
stack.conf
```

d. Collect the Logcat logs by running the following command:

```
journalctl -f > all_logs.txt
```

e. Pull the Logcat file by running the following command:

```
scp -r root@<IP_address>:<source_file_path_of_all_logs.txt>
<destination_file_path>
```

To pull a file to the current file path of the PC, enter the `<destination_file_path>` as `.` in the command.

Example

The IP address of the device is `10.92.160.222`. To pull the Logcat file `all_logs.txt` to the current file path, run the following command:

```
scp -r root@10.92.180.250:/home/root/all_logs.txt .
```

5. Pull the BTSnoop logs by running the following command:

```
scp -r root@<IP_address>:/etc/bluetooth/btsnoop_hci.log
<destination_file_path>
```

The HCIDUMP/BTSnoop logs are useful to determine whether the issue relates to the application, Bluetooth host, Bluetooth chipset, or a remote device.

To pull a file to the current file path of the PC, enter the `<destination_file_path>` as `.` in the command.

Example

The IP address of the device is `10.92.160.222`. To pull the `btsnoop_hci.log` file, run the following command:

```
scp -r root@10.92.160.222:/etc/bluetooth/btsnoop_hci.log .
```

If there is a composite USB, Q6 handles the waking host, connects the USB, and then sends the crash dump to the host.

In the standalone mode, Bluetooth wakes up the host and sends the crash dump.

OTA logs

OTA logs are also known as sniffer logs. Bluetooth OTA logs are captured using the FTS/Ellisys sniffer tool to debug certain chipset or remote device issues, and environment issues.

8 References

8.1 Reference documents

Document title	DCN
Qualcomm Linux Build Guide	80-70018-254
Qualcomm Linux Yocto Guide	80-70018-27

8.2 Acronyms and terms

Acronyms/terms	Definition
A2DP	Advanced Audio Distribution Profile
AG	Audio gateway
ATT	Attribute Protocol
AVRCP	Audio/Video Remote Control Profile
BR/EDR	Basic Rate/Enhanced Data Rate
CLI	Command-line interface
CSA	Channel selection algorithm
CT	Controller
DBUS	Data bus
DTMF	Dual-tone multifrequency
EDR	Enhanced Data Rate
FTP	File Transfer Protocol
GAP	General Access Profile
GATT	General Attribute Profile
GPIO	General-purpose input/output
HAL	Hardware Abstraction Layer
HCI	Host Control Interface
HF	Hands-free
HFP	Hands-Free Profile
HID	Human interface device
HOGP	HID over GATT Profile

Acronyms/terms	Definition
IM	Instant messages
JNI	Java Native Interface
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
LL	Link Layer
LMP	Link Manager Protocol
MAC	Media access control
MAP	Message Access Profile
MAS	Message access service
MMS	Multimedia messaging service
MNS	Message notification service
MPRIS	Media Player Remote Interfacing Specification
NV	Nonvolatile
OBEX	Object Exchange Protocol
OPP	Object Push Profile
OTA	Over-the-air
OTP	One-time programmable
PAL	Platform Abstraction Layer
PBAP	Phone Book Access Profile
PHY	Physical layer
POSIX	Portable operating system interface
PSE	Phone book server equipment
RFCOMM	Radio Frequency Communication
SCO	Synchronous Connection Oriented
SCP	Secure Copy Protocol
SDP	Session Description Protocol
SMS	Short message service
SoC	System-on-chip
SPP	Serial Port Profile
SSH	Secure shell
SSP	Secure Simple Pairing
TCP	Transmission Control Protocol
TG	Target
TTY	Teletype
UART	Universal asynchronous receiver/transmitter
UI	User interface
UUID	Universal Unique ID

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.