Intel® Edison

Bluetooth* Guide

June 2015

Revision 007

Document Number: 331704-007

Contents

1 Introduction 7
1.1 BlueZ software stack 7
1.2 Software reference map 7
1.3 References 7
Terminology 8
2 Bluetooth* Integration in Linux* 9
2.1 The bluetoothd daemon 10
2.2 Configuration 10
2.3 Application interface 11
3 Basic Bluetooth* Operation 12
3.1 Enable and disable Bluetooth* on Intel® Edison 12
3.2 Bluetooth* status control via connman 13
3.3 The bluetoothctl utility 14
3.4 Device identification (DI) profile 14
4 Scanning and Connecting Devices 16
4.1 Connecting from a peer device 18
5 Changing a Bluetooth* MAC address 19
6 Bluetooth Profiles on Intel® Edison 20
6.1 Bluetooth* Low Energy (BLE) profile 21
6.1.1 Verifying BLE plugin compilation 21
6.1.2 Preparing to test Bluetooth* profiles 22
6.2 Scan and connect 24
6.2.1 bluetoothctl 24
6.2.2 hcitool 25
6.2.3 btmgmt 27
6.2.4 Python test scripts 30
6.2.5 GATTtool 31
6.3 Advanced audio distribution profile (A2DP) 32
6.4 Device identification (DI) profile 34
6.4.1 Reading and changing the local device identification 34
6.4.2 Retrieving the peer device's DI information 35
6.5 Human interface device (HID) profile 37
6.6 Personal area networking (PAN) profile 39
6.6.1 PAN test between Linux* host PC and Intel® Edison device 40
6.6.2 PAN test between two Intel® Edison devices 44
6.7 Serial port profile (SPP) 48
6.7.1 SPP verification using DBUS APIs 49
6.7.2 SPP verification using the RFCOMM tool 52
6.9 HID over GATT profile (HOGP) 56
6.10 Heart rate profile (HRP) 58
6.11 Proximity profile (PXP) 60
6.11.1 PXP services 60
6.11.2 PXP test 60
6.11.3 Proximity monitor 61
6.11.4 Proximity reporter 62
6.12 Time profile (TIP) 64

6.13 File transfer protocol (FTP) profile 65
6.13.1 FTP server 66
6.13.2 FTP client 69
Appendix A: SPP-loopback.py 71

Figures

Figure 1 Intel® Edison to Broadcom BCM43340 connections 7
Figure 2 The BlueZ package 9
Figure 3 Help view of available commands 14
Figure 4 Show command 15
Figure 5 Modalias change 15
Figure 6 BLE architecture 21
Figure 7 Bluetooth* plugins 22
Figure 8 The rfkill unblock bluetooth command 22
Figure 9 The hciconfig hci0 lestates command 23
Figure 10 HCI events 25
Figure 11 hcitool > hcidump traces 26
Figure 12 btmgmt > hcidump traces 28
Figure 13 btmgmt > hcidump traces (successful pairings) 29
Figure 14 The test-discovery Python script 30
Figure 15 Scan for the Bluetooth* headset 32
Figure 16 Pair/connect the Bluetooth* headset 32
Figure 17 Results from uncommented device ID line 33
Figure 18 Copy audio and playing using mplayer 33
Figure 19 Show command 34
Figure 20 Results from uncommented DeviceID line 35
Figure 21 sdptool tool results 35
Figure 22 bluetoothctl tool retrieval results 36
Figure 23 Raw data from the event file using the “more” command 38
Figure 24 PAN service networking models 39
Figure 25 Linux pairing successful 41
Figure 26 Editing the bluetooth.conf file 48
Figure 27 Serial port absent before running SPP-loopback.py 49
Figure 28 Serial port present after running SPP-loopback.py 49
Figure 29 Search for peer devices 50
Figure 30 Still searching 50
Figure 31 Android* screenshots 51
Figure 32 Connected devices 51
Figure 33 Sequence of screenshots showing the user inputs the text SPP application 52
Figure 34 BlueTerm app sending text via SPP 54
Figure 35 Minicom window on Linux* PC sending text 55
Figure 36 Mirrored text in Intel® Edison device’s cat shell window 55
Figure 37 Example event test results from Bluetooth mouse 57
Figure 38 Example heart rate monitor data 59
Figure 39 Current time service on Android* device 64
Figure 40 Checking obex profiles 65
Figure 41 Pairing Intel® Edison with Android* peer devices 66
Figure 42 Android* FTP screenshots 67
Figure 43 Send/browse files 68
Figure 44 Bluetooth* file transfer 68
Figure 45 Actions available after pairing 69
Figure 46 Actions available 69

Figure 47 Android* device screenshots 70

Tables

Table 1 Supported profiles 20

Revision History

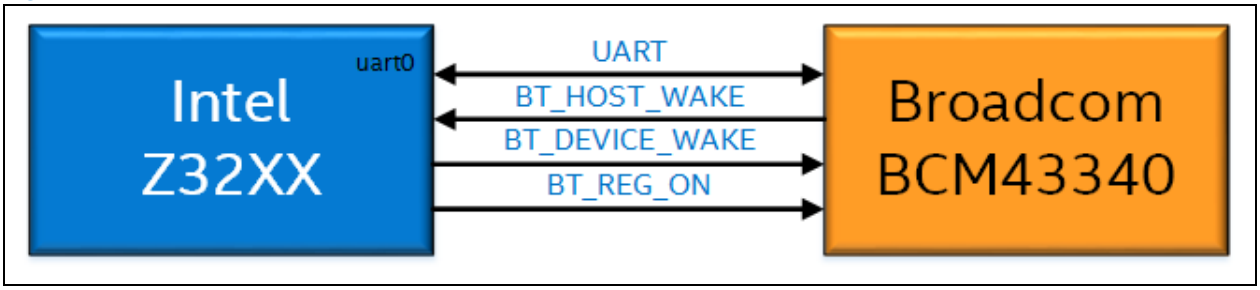
Revision	Description	Date
001	Initial release.	December 17, 2014
002	Added content on Bluetooth# profiles.	February 4, 2015
003	Added A2DP profile.	February 13, 2015
004	Added python script appendix.	February 20, 2015
005	Minor edits.	May 1, 2015
006	Fixed a hyperlink.	May 22, 2015
007	Fixed a hyperlink.	June 17, 2015

§

1 Introduction

The host processor on the Intel® Edison development board is connected to a Broadcom* BCM43340 combo chip via UART (uart0 mapped to */dev/MFD0*) as transport layer and uses additional GPIOs to handle power (on, reset, etc.), OOB (out-of-band) signaling for UART to support low power mode.

Figure 1 Intel® Edison to Broadcom BCM43340 connections



1.1 BlueZ software stack

BlueZ, an open source project, is the official Linux* Bluetooth* protocol stack. The BlueZ package has a doc folder that contains a DBUS API description text file with some other information related to supported features: settings, storage, etc. The BlueZ stack sources divide into components in both the kernel and user spaces, which should be compiled accordingly; the main component is the *bluetoothd* daemon, which exposes DBUS APIs to the application layer for development. (DBUS APIs are interfaces exposed to develop application; they do not explain internal working mechanisms.) We have modified the Yocto recipes to append the BlueZ 5.24 version, not the default.

Note: The Intel® Edison board currently runs with Linux* kernel 3.10 with a low-energy patch added to the kernel to handle Random Address. . For more information on BlueZ, refer to their website at *http://www.bluez.org*.

1.2 Software reference map

Release-1	https://communities.intel.com/community/makers/edison/documentation Software Downloads -> Rel-1-Maint-WW42 (is latest for Release-1)
Release-2	TBD (DEC2014)

1.3 References

Reference	Name	Number/location
331188	Intel® Edison Board Support Package User Guide	*http://www.intel.com/support/edison/sb/CS-035278.htm*
331189	Intel® Edison Compute Module Hardware Guide	*http://www.intel.com/support/edison/sb/CS-035274.htm*
331190	Intel® Edison Breakout Board Hardware Guide	*http://www.intel.com/support/edison/sb/CS-035252.htm*
331191	Intel® Edison Kit for Arduino* Hardware Guide	*http://www.intel.com/support/edison/sb/CS-035275.htm*
331192	Intel® Edison Native Application Guide	*http://www.intel.com/support/edison/sb/CS-035382.htm*
329686	Intel® Galileo and Intel® Edison Release Notes	*https://communities.intel.com/docs/DOC-23388*
332032	Intel® Edison Software Release Notes	
[GSG]	Intel® Edison Getting Started Guide	W: *http://www.intel.com/support/edison/sb/CS-035336.htm* M: *http://www.intel.com/support/edison/sb/CS-035344.htm* L: *http://www.intel.com/support/edison/sb/CS-035335.htm*
331438	Intel® Edison Wi-Fi Guide	*http://www.intel.com/support/edison/sb/CS-035380.htm*
331704	Intel® Edison Bluetooth* Guide	(This document)
332434	Intel® Edison Audio Setup Guide	

Terminology

Term	Definition
BNEP	Bluetooth Network Encapsulation Protocol. BNEP is an Ethernet interface created for each Bluetooth* connection.
BT	Bluetooth
BT-LE, BLE	Bluetooth low energy
DBUS	An interprocess communication protocol
DI	Device Identification
GPIO	General purpose input/output
HCI	Host controller interface
HID	Human interface device
MFD	Multifunction device
NAP	Network access point
OOB	Out-of-band
PAN	Personal area network
SDP	Service Discovery Profile
ssh	Secure shell
UART	Universal asynchronous receiver/transmitter
TIP	Time profile
PXP	Proximity Profile
SPP	Serial Port Profile
A2DP	Advanced Audio Distribution Profile
FTP	File Transfer Profile
HRP	Heart Rate Profile
HOGP	HID over GATT profile
GAP	Generic Access Profile

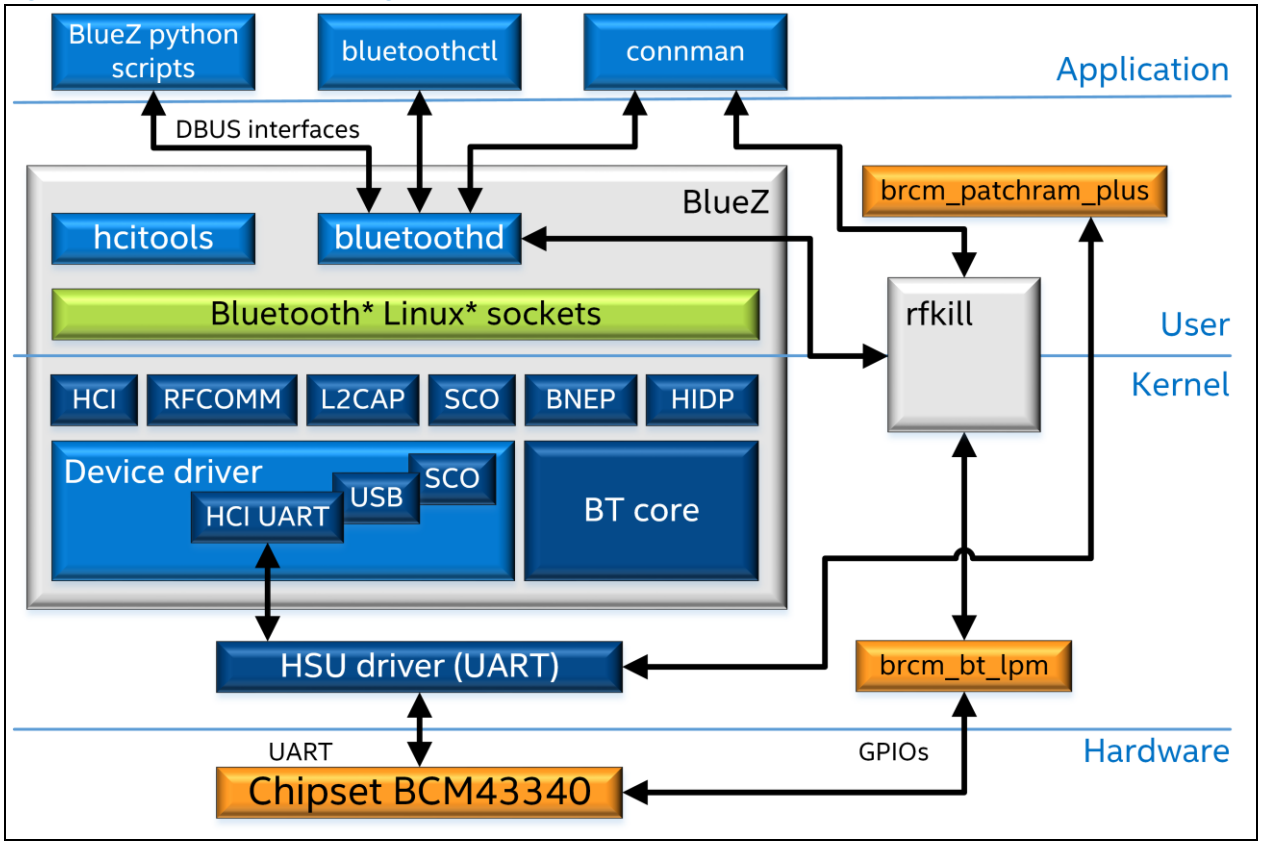
§

2 Bluetooth* Integration in Linux*

Bluetooth* controllers are handled in Linux* via interfaces accessible by the *rfkill* and *hci* utilities (*rfkill*, *hcidump*, *hciconfig*, *hcitools*, etc.). These utilities, which are provided in the BlueZ package (Figure 2), include the following:

- *rfkill*: Turns the chip on/off.
- *hcitools*: A series of utilities that manage controllers:
  - *hcidump*: Retrieves the trace of the HCI device.
  - *hciconfig*: Configures Bluetooth* devices.
  - *hcitool*: Configures Bluetooth* connections and sends commands to Bluetooth* devices.
  - *hciattach*: Attaches an HCI device to a dev interface, like USB or UART; usually it is used to download patchram to the Bluetooth* controller.

Figure 2 The BlueZ package



In general, *hciattach* launches automatically whenever a Bluetooth* controller connects over USB. This occurs when *rfkill* turns Bluetooth* on and the system calls *hciattach* with the proper patchram.

Since the Bluetooth* controller connects to the UART, *hciattach* does not launch automatically, even after starting Bluetooth* with *rfkill*. To support the functionalities of *hciattach*, the Intel® Edison image has a built-in service called *Bluetooth_rfkill_event* that starts at bootup and runs in the background, listening for Bluetooth* interface *rfkill* events. If *Bluetooth_rfkill_event* identifies an event intended for BCM43340, it calls the Broadcom download utility, which does the same job as *hciattach* (along with some Broadcom-specific functions). Whenever you are enabling or testing Bluetooth* functionality, make sure *Bluetooth_rfkill_event* is running in the background.

2.1 The bluetoothd daemon

The *bluetoothd* daemon can be started even when the Bluetooth controller is not enabled; at startup, it loads, initializes plugins, and listens to events from the kernel. As soon as the *MGMT_EV_INDEX_ADDED* management indication is received for an HCI device, the daemon registers an adapter entity for the BT controller and initializes.

The plugin is a piece of software that implements features/profile. BlueZ comes with set of built-in plugins (to support profiles like A2DP, AVRCP, networking/PAN, input/HID, GATT, and items like *wiimote* and *hostname*) that are loaded and enabled at boot time (if not differently specified).

BlueZ also has the support to load and initialize custom plugins developed by third-parties and that are looked for in init in the */usr/lib/bluetooth/plugins* folder. Basically, a plugin lets you run some actions when the *bluetoothd* daemon initializes (when the adapter is not already registered).

Typical actions performed at a plugin initialization include the following:

- Defining directly the DBUS interfaces for application layer (like the *hostname* plugin).
- Registering the adapter driver (*btd_adapter_driver* structure has a probe entry that is called when an adapter is registered).
- Register a profile (*btd_profile* structure has some "pointer-to-function" fields, between them an *adapter_probe* entry that is called when the adapter is registered. (There are similar entries for *device*, where *device* is the structure that handles a peer device when connected/paired etc.).

This mechanism allows plugin to be notified or do specific actions to be performed at init, when an adapter is registered or when a device is paired/connected.

Note: Registering a profile using the plugin mechanism doesn't mean the profile is advertised to a peer device since an application has to register the profile via DBUS interface so that SDP (Service Discovery Protocol) can discover the service provided by it.

All settings are stored under a storage directory (by default */var/lib/bluetooth*) that can be inspected for debugging purposes; this folder structure is documented in the *<bluez_package>/doc/settings-storage* file.

2.2 Configuration

By default the *bluetoothd* daemon will load and initialize all built-in plugins, but it is also possible to directly enable or disable a set of plugins with the `-plugin` and `--noplugin` command line options. When you disable a plugin, the corresponding profile won't be available, which means that no application will be able to register and advertise this service. You can do this on an Intel® Edison device by modifying the *systemd* file for BlueZ (*/etc/systemd/system/bluetooth.target.wants/bluetooth.service*) and adding command line arguments in *ExecStart*.

BlueZ also comes with *conf* files that let you specify some of the profile features (*input.conf*, *network.conf*, and *proximity.conf*) plus a more generic *conf* file (*main.conf*) that lets you specify name, discoverable and pairable timeouts, and other settings. These configuration files, which are located under */etc/bluetooth* on an Intel® Edison device, are loaded at the Bluetooth daemon's boot time (when the *bluetooth systemd* service is started). If you modify any of these configurations, you will need to stop and start the *systemd* Bluetooth service to activate the changes.

```
root@edison:~# ls
root@edison:~# systemctl stop bluetooth
root@edison:~# systemctl start bluetooth
```

2.3 Application interface

The application layer can use the Bluetooth service provided by the BlueZ stack using the DBUS API interface registered by each profile/component. The *<bluez_package>/doc* folder contains a text file that describes these DBUS API methods and properties.

The main components of this package include the following:

- **adapter**: Lets you start or stop discovery; remove a paired device; or set/get info about name, alias, pairable timeout, etc.
- **agent**: Lets you register or unregister agent; set the default one or all methods related to pairing or authorization, etc.
- **device**: Lets you connect or disconnect; pair a device; connect or disconnect a profile on a device, etc.; set or get info about trusted or blocked class of peer device.
- **profile**: Lets you register a profile implementation.

The folder also contains a file called *mgmt-api.txt*, which describes the format of data used for communicating with kernel using the so-called Bluetooth* management sockets. Profile-specific API documentation (like network, obex) is also available.

§

3 Basic Bluetooth* Operation

Before you can perform any Bluetooth operations, connect to the Intel® Edison device via *ssh* or *minicom* and call *rfkill* to unblock the Broadcom* BCM43340 chip. The *Bluetooth_rfkill_event* service, which should be running in background, will intercept the *rfkill* event, trigger a firmware patch download, configure the Broadcom* BCM43340 chip, and register the HCI device (hci0).

Note: Whenever you are enabling or testing Bluetooth* functionality, make sure *Bluetooth_rfkill_event* is running in the background. This utility downloads patches and registers HCI is *brcm_patchram_plus*. It operates like *hciattach* but has more Broadcom-specific options.

3.1 Enable and disable Bluetooth* on Intel® Edison

To enable or disable Bluetooth, using the following commands respectively:

```
root@edison:~# rfkill unblock bluetooth
root@edison:~# rfkill block bluetooth
```

Once Bluetooth is enabled, *rfkill* will usually list the available interfaces. You can also use the *rfkill list* command to show them. For instance, on the Intel® Edison board:

```
root@edison:~# rfkill block bluetooth
root@edison:~# rfkill list
0: phy0: wlan
        Soft blocked: no
        Hard blocked: no
1: brcmfmac-wifi: wlan
        Soft blocked: no
        Hard blocked: no
2: bcm43xx Bluetooth: bluetooth
        Soft blocked: yes
        Hard blocked: no
```

And:

```
root@edison:~# rfkill unblock bluetooth
root@edison:~# rfkill list
0: phy0: wlan
        Soft blocked: no
        Hard blocked: no
1: brcmfmac-wifi: wlan
        Soft blocked: no
        Hard blocked: no
2: bcm43xx Bluetooth: bluetooth
        Soft blocked: no
        Hard blocked: no
3: hci0: Bluetooth
        Soft blocked: no
        Hard blocked: no
root@edison:~#
```

The string "`bcm43xx Bluetooth: bluetooth`" is added by the power driver of the Bluetooth* controller, either already included in the kernel or loaded as module; blocking or unblocking it via the *rfkill block* or *rfkill unblock* command will power the chip off or on. The *rfkill unblock* command does both the tasks of *systemctl start connman* and *connmanctl enable bluetooth*.

3.2 Bluetooth* status control via connman

Connman is a connection manager with a Bluetooth plugin (relying on the BlueZ DBUS interfaces). *Connman* manages network connections over Bluetooth using PAN (with a PAN user role). *Connman* gets information about connected/paired devices from BlueZ DBUS interfaces (through the PAN NAP/GN service, available on the peer devices). You cannot use *connman* for all pairing and connection procedures. It does, however, let you enable/disable technology (keeping track of the previous status) via the *rfkill* component.

Because *connman* does not start automatically at boot time, start it manually.

```
root@edison:~# systemctl start connman
root@edison:~# connmanctl enable bluetooth
Enabled bluetooth
root@edison:~# rfkill list
0: phy0: wlan
        Soft blocked: no
        Hard blocked: no
1: brcmfmac-wifi: wlan
        Soft blocked: no
        Hard blocked: no
2: bcm43xx Bluetooth: bluetooth
        Soft blocked: no
        Hard blocked: no
3: hci0: bluetooth
        Soft blocked: no
        Hard blocked: no
root@edison:~#
```

The last item (`3: hci0: bluetooth`) is added when the serial device is attached to the BlueZ stack. If the kernel already configures some *_BT_ HCI_ driver* entries (like *CONFIG_BT_HCIBCM203X*, *CONFIG_BT_HCIBTSDIO*), it will already list an *hcix: bluetooth* interface.

3.3 The bluetoothctl utility

This command line utility can be used to perform basic Bluetooth* operation, such as:

- Register an agent,
- Start or stop discovery,
- Configure pairable or discoverable property of the adapter,
- Pair and connect a device.

The utility interacts with the *bluetoothd* daemon via DBUS interfaces. Enter *help* to display the full list of available commands (Figure 3).

Figure 3 Help view of available commands

```
root@edison:~# bluetoothctl
[NEW] Controller 98:4F:EE:01:FB:AC BlueZ 5.18 [default]
[bluetooth]# help
Available commands:
  list                       List available controllers
  show [ctrl]                Controller information
  select <ctrl>              Select default controller
  devices                    List available devices
  paired-devices             List paired devices
  power <on/off>             Set controller power
  pairable <on/off>          Set controller pairable mode
  discoverable <on/off>      Set controller discoverable mode
  agent <on/off/capability>  Enable/disable agent with given capability
  default-agent              Set agent as the default one
  scan <on/off>              Scan for devices
  info <dev>                 Device information
  pair <dev>                 Pair 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
  version                    Display version
  quit                       Quit program
[bluetooth]#
```

3.4 Device identification (DI) profile

The scope of the Device Identification (DI) profile is to provide additional information above and beyond the *Bluetooth* class of device and to incorporate the information into both the Service Discovery Profile (SDP) record and the Extended Inquiry Response (EIR).

A device can be identified by the following information:

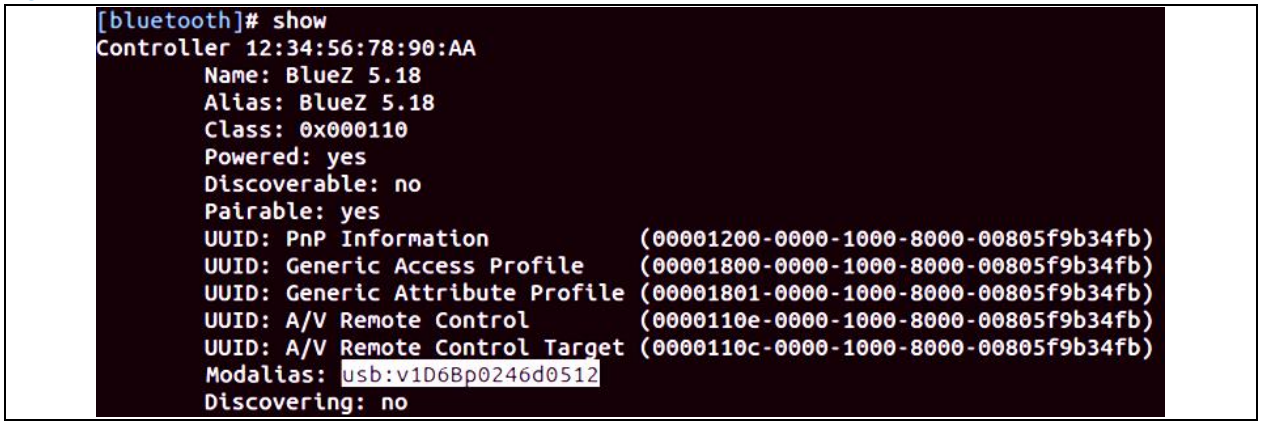
- VendorID Source: Indicates if the VendorID refers to Bluetooth or USB.
- The allowed values are:
  - **0x0001**, which means that the VendorID is assigned by the Bluetooth SIG (*https://www.bluetooth.org/en-us/specification/assigned-numbers/company-identifiers*)
  - **0x0002**, which means that the VendorID is assigned by the USB Group (*https://usb-ids.gowdy.us/read/UD/*)
- VendorID (16 bits)
- DeviceId (16 bits)
- Version (16 bits)

The default BlueZ Device Information is:

- VendorID Source = USB
- VendorID = 0x1D6B (Linux Foundation)
- ProductID = 0x0246 (BlueZ)
- Version = 0x0512 (5.18)

You can retrieve this information from the local device with the *bluetoothctl* program's *show* command (Figure 4).

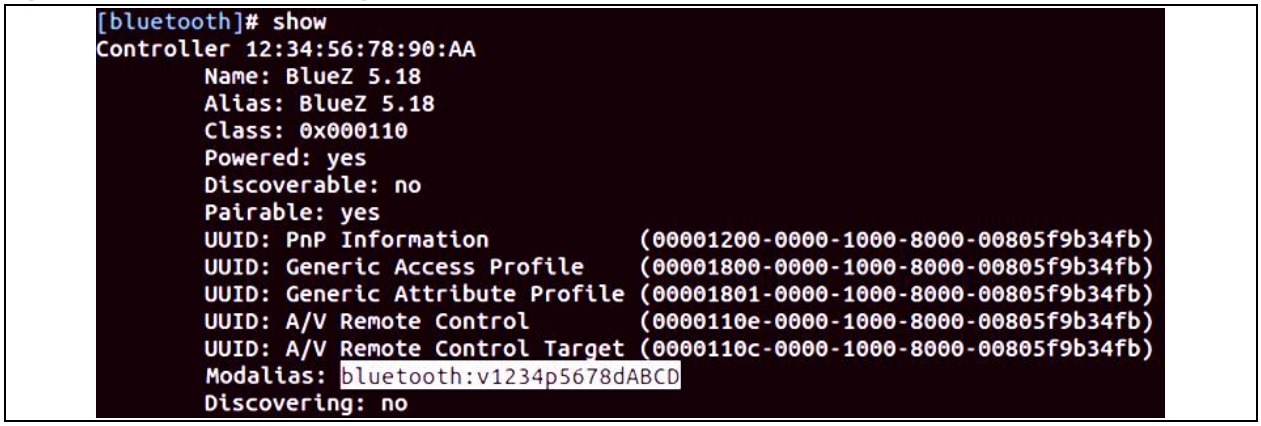
Figure 4 Show command



You can modify this information by changing the */etc/bluetooth/main.conf* file by uncommenting (and changing) the line containing the `DeviceID = ...` line. For example, the following line will change the *modalias*, as shown in Figure 5:

```
DeviceID = bluetooth:1234:5678:abcd
```

Figure 5 Modalias change



§

4 Scanning and Connecting Devices

To connect your Intel® Edison device to a Bluetooth network, do the following:

1. Enable Bluetooth:

```
root@edison:~# rfkill unblock bluetooth
```

2. Enter the BlueZ command line utility *bluetoothctl*, which will find the Bluetooth controller:

```
root@edison:~# bluetoothctl
[NEW] Controller 98:4F:EE:01:FD:D6 BlueZ 5.24 [default]
[bluetooth]#
```

3. Register an agent and set it as default. (An agent lets you handle actions such as pairing, when user interaction is needed.) Options include *KeyboardDisplay*, *DisplayOnly*, *DisplayYesNo*, *KeyboardOnly*, and *NoInputNoOutput*. These settings emulate different capabilities of the application developed by the end-user for Bluetooth using an Intel® Edison board.

```
[bluetooth]# agent KeyboardDisplay
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]#
```

4. Perform a scan. You can stop the scan as soon as it reports the device you are looking for.

```
[bluetooth]# scan on
Discovery started
[CHG] Controller 98:4F:EE:01:FD:D6 Discovering: yes
[NEW] Device F3:18:29:E8:DA:61 Flex
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device 40:2C:F4:86:72:54 TNGU25X-MOBL2
[bluetooth]# scan off
```

5. Pair the device. (You will need to confirm the pairing from the peer device, so be sure to have an agent set as described in step 3 above.)

```
[bluetooth]# pair 40:2C:F4:DB:EF:AA
Attempting to pair with 40:2C:F4:DB:EF:AA
[CHG] Device 40:2C:F4:DB:EF:AA Connected: yes
Request confirmation
[agent] Confirm passkey 788684 (yes/no): yes
[CHG] Device 40:2C:F4:DB:EF:AA UUIDs:
        00000002-0000-1000-8000-0002ee000002
        00001000-0000-1000-8000-00805f9b34fb
        00001104-0000-1000-8000-00805f9b34fb
[CHG] Device 40:2C:F4:DB:EF:AA Paired: yes
Pairing successful
[CHG] Device 40:2C:F4:DB:EF:AA Connected: no
[CHG] Device 40:2C:F4:DB:EF:AA Connected: yes
[CHG] Device 40:2C:F4:DB:EF:AA Connected: no
[bluetooth]
```

6. Trigger the connection step:

```
[bluetooth]# connect 10:68:3F:57:90:4F
Attempting to connect to 10:68:3F:57:90:4F
[CHG] Device 10:68:3F:57:90:4F Connected: yes  Connection successful
```

The Bluetooth connection is established at the profile level, so the involved devices have to support profiles (and roles, if applicable) that let them connect. For HID, there is no need to register the profile at the application layer. (The HID host is implemented at the kernel level.) So a basic *discovery + pair + connect* to an HID peripheral device will lead to a connection.

7. You can check supported services on a peer device using the *info* command:

```
[bluetooth]# info 40:2C:F4:DB:EF:AA
Device 40:2C:F4:DB:EF:AA
        Name: NAGESWAX-MOBL1
        Alias: NAGESWAX-MOBL1
        Class: 0x3e010c
        Icon: computer
        Paired: yes
        Trusted: no
        Blocked: no
        Connected: no
        LegacyPairing: no
        UUID: Vendor specific           (00000002-0000-1000-8000-0002ee000002)
        UUID: Service Discovery Serve.. (00001000-0000-1000-8000-00805f9b34fb)
        UUID: IrMC Sync                 (00001104-0000-1000-8000-00805f9b34fb)
        UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
        UUID: OBEX File Transfer        (00001106-0000-1000-8000-00805f9b34fb)
        UUID: IrMC Sync Command         (00001107-0000-1000-8000-00805f9b34fb)
        UUID: Headset                   (00001108-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
        UUID: PANU                      (00001115-0000-1000-8000-00805f9b34fb)
        UUID: Imaging Responder         (0000111b-0000-1000-8000-00805f9b34fb)
        UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
        UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
        UUID: Video Sink                (00001304-0000-1000-8000-00805f9b34fb)
[bluetooth]#
```

8. When you are done, exit the utility:

```
[bluetooth]# exit
Agent unregistered
[DEL] Controller 98:4F:EE;01;FD;D6 BlueZ 5.24 [default]
root@edison:~#
```

4.1 Connecting from a peer device

To connect your Intel® Edison device from a peer device, do the following:

1. Follow steps 1 through 3 above.
2. Set up the Intel® Edison device as “discoverable” in step 4:

```
[bluetooth]# discoverable on
Changing discoverable on succeeded
[CHG] Controller 98:4F:EE:01:FD:D6 Discoverable: yes
[bluetooth]#
```

3. Start the discovery from the peer device and pair the Intel® Edison device.

§

5 Changing a Bluetooth* MAC address

Each Intel® Edison device has its own unique Bluetooth* MAC address, which is in */factory/bluetooth_address*. The steps below describe the process to change the Bluetooth* MAC address, by mounting */factory*, editing the */factory/bluetooth_address* file, and rebooting the device.

```
root@edison:~# mount -v | grep factory
/dev/mmcblk0p5 on /factory type ext4 (ro,nosuid,nodev,noatime,discard,
noauto_da_alloc)
root@edison:~# mount -o remount,rw /dev/mmcblk0p5 /factory
root@edison:~# vi /factory/bluetooth_address
root@edison:~# reboot
         Unmounting /home...
[  OK  ]  Stopped target Sound Card.
[  OK  ]  Removed slice system-systemd\x2dfsck.slice.
[  OK  ]  Stopped target Multiuser System.
         Stopped the Edison status and configuration service...
```

Note: We do not advise changing an Intel® Edison board's Bluetooth* MAC address. If you do decide to change it, however, presumably for testing purposes, first make sure to back up the unique MAC address that was generated when the Intel® Edison board was first dispatched, and revert the MAC address back to what it originally was (its unique MAC address) as soon as you are finished with your testing.

§

6 Bluetooth Profiles on Intel® Edison

To use Bluetooth* wireless technology, a device must be able to interpret Bluetooth* profiles, which define possible applications and specify general behaviors that Bluetooth*-enabled devices use to communicate with each other. Each Bluetooth profile contains the following information:

- Dependencies on other profiles
- Suggested user interface formats
- Specific parts of the Bluetooth protocol stack used by the profile.

Intel® Edison supports all BlueZ profiles, but we have only validated a subset of these profiles and features so far (listed in Table 1).

Note: For details on all of the BlueZ profiles, visit the BlueZ website: *http://www.bluez.org*.

Table 1 Supported profiles

		Validated in Release 1	Validated in Release 2
Provided by BlueZ			
A2DP	Advanced audio distribution profile (A2DP)		Yes
AVRCP	Audio/video remote control profile		
DI	Device identification (DI) profile		Yes
HDP	Health device profile		
HID	Human interface device (HID) profile	Yes	Yes
PAN	Personal area networking (PAN) profile	Yes	Yes
SPP	Serial port profile (SPP)		Yes
GATT (LE) profiles			
CSCP	Cycling speed and cadence profile		
HOGP	HID over GATT profile (HOGP)		Yes
HRP	Heart rate profile (HRP)		Yes
HTP	Health thermometer profile		
PXP	Proximity profile (PXP)		Yes
TIP	Time profile (TIP)		Yes
OBEX-based profiles (by obexd)			
FTP	File transfer protocol (FTP) profile		Yes
MAP	Message access profile		
OPP	Object push profile		
PBAP	Phone book access profile		
Provided by the oFono project			
HFP (AG and HF)	Hands-free profile		[1]

Note: While testing Bluetooth* "classic" and low energy (BLE) profiles on the Intel® Edison platform, we used Linux* PCs, Android* phones, Logic tech HID devices, HTC-Fetch, Polar H7 heart rate monitors, and other BLE devices as peers. We did not perform any testing on Mac* OS X or Windows* devices.

The first two sections of this chapter explain the Bluetooth* LE (BLE) plugin and means to scan and connect; the remaining sections explain how to use the validated Bluetooth* profiles on the Intel® Edison platform.

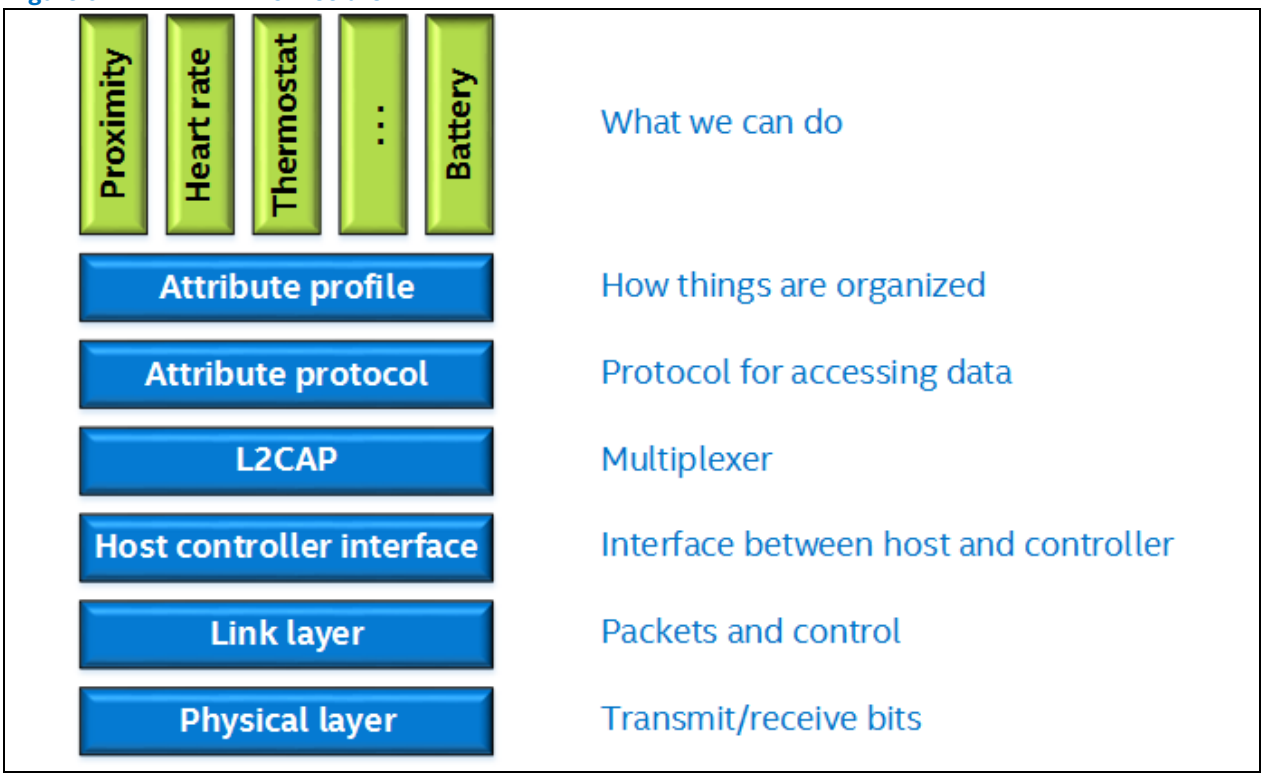
[1] *HFP functionality is not currently supported.*

6.1 Bluetooth* Low Energy (BLE) profile

Bluetooth* Low Energy (BLE), marketed by the Bluetooth* SIG as Bluetooth* Smart, is often used for applications related to healthcare, fitness, and security, or in any situation where low energy consumption is important. BLE is intended to provide the same functionalities as "classic" Bluetooth* technology, but with better energy and cost efficiencies. Figure 6 shows a diagram of the BLE architecture.

The BlueZ stack in Intel® Edison fully supports GATT client and server roles through internal native C APIs, but you will probably have to implement some of your own GATT profiles (custom or standard). For example, some of the default standard GATT profiles (health, alert, time, proximity, thermometer, heart rate, cycling speed, etc.) are already implemented in BlueZ as experimental, which means they are fully functional, but their DBUS interface APIs may change over time. Because the Intel® Edison software build includes the BlueZ stack configured in experimental mode, these profiles are available in the software by default.

Figure 6 BLE architecture



6.1.1 Verifying BLE plugin compilation

BlueZ on the Intel® Edison platform is compiled by default in experimental mode, to enable BLE profiles. To verify that BlueZ has been compiled with BLE plugins, do the following to enable all the logs:

1. Stop the bluetoothd daemon:
   ```
   root@edison:~# systemctl stop bluetooth
   ```
2. Change the Bluetooth* system service file (*/etc/systemd/system/bluetooth.target.wants/bluetooth.service*) by adding the `-d` option:
   ```
   ExecStart=/usr/lib/bluez5/bluetooth/bluetoothd -d
   ```
3. Restart the Bluetooth* service:
   ```
   root@edison:~# systemctl start bluetooth
   ```

4. Launch the Bluetooth* logs with the *journalctl --unit=bluetooth* command and verify that they are present when the Bluetooth* service starts (Figure 7). The Bluetooth* logs show the various plugins supported in the Intel® Edison image and show whether the *bluetoothd* daemon is stopped or started.

Figure 7 Bluetooth* plugins

```
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading health plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading gatt plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading scanparam plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading deviceinfo plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading alert plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading time plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading proximity plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading thermometer plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading heartrate plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:add_plugin() Loading cyclingspeed plugin
Dec 09 17:25:40 edison bluetoothd[244]: src/plugin.c:plugin_init() Loading plugins /usr/lib/b
luetooth/plugins /usr/lib/bluetooth/plugins
```

To test the Bluetooth* profiles using the BlueZ test scripts, you need to copy the BlueZ test scripts into the Intel® Edison device. To test the profiles with commonly available Python scripts, verify that the DBUS policy file */etc/dbus-1/system.d/bluetooth.conf* has the following lines:

```
<allow send_interface="org.bluez.ThermometerWatcher1"/>
<allow send_interface="org.bluez.AlertAgent1"/>
<allow send_interface="org.bluez.HeartRateWatcher1"/>
<allow send_interface="org.bluez.CyclingSpeedWatcher1"/>
```

If these lines are not in the file, you can add them dynamically at runtime.

6.1.2 Preparing to test Bluetooth* profiles

Before performing any profile test, you should run two commands: *rfkill unblock bluetooth* and *hciconfig hci0* (optional). After you run these commands, you can scan and connect to the BLE devices using different tools. (See section 6.2.)

To prepare for Bluetooth* profile testing, do the following:

1. Before testing any Bluetooth* functionality, run the *rfkill unblock bluetooth* command to make sure that Bluetooth* is on and that the HCI interface is up and running (Figure 8).

Figure 8 The rfkill unblock bluetooth command

```
root@edison:~# rfkill unblock bluetooth
root@edison:~# hciconfig hci0
hci0:   Type: BR/EDR  Bus: UART
        BD Address: 00:11:22:33:55:77  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:900 acl:0 sco:0 events:64 errors:0
        TX bytes:1650 acl:0 sco:0 commands:61 errors:0
root@edison:~#
```

2. If you want to check the Link Manager states supported by the controller, you can use the hciconfig hci0 lestates command (Figure 9). This command is not necessary to test the profile, but it does check the supported states of the Bluetooth* controller: Connectable, not advertising, scannable advertising, passive or active scanning, and all supported combinations. The supported combinations show that the device is BT4.0 (which means it does not support dual-mode topology).

Figure 9 The hciconfig hci0 lestates command

```
root@edison:~# hciconfig hci0 lestates
Supported link layer states:
    YES Non-connectable Advertising State
    YES Scannable Advertising State
    YES Connectable Advertising State
    YES Directed Advertising State
    YES Passive Scanning State
    YES Active Scanning State
    YES Initiating State/Connection State in Master Role
    YES Connection State in the Slave Role
    YES Non-connectable Advertising State and Passive Scanning State combination
    YES Scannable Advertising State and Passive Scanning State combination
    YES Connectable Advertising State and Passive Scanning State combination
    YES Directed Advertising State and Passive Scanning State combination
    YES Non-connectable Advertising State and Active Scanning State combination
    YES Scannable Advertising State and Active Scanning State combination
    YES Connectable Advertising State and Active Scanning State combination
    YES Directed Advertising State and Active Scanning State combination
    YES Non-connectable Advertising State and Initiating State combination
    YES Scannable Advertising State and Initiating State combination
    YES Non-connectable Advertising State and Master Role combination
    YES Scannable Advertising State and Master Role combination
    YES Non-connectable Advertising State and Slave Role combination
    YES Scannable Advertising State and Slave Role combination
    YES Passive Scanning State and Initiating State combination
    YES Active Scanning State and Initiating State combination
    YES Passive Scanning State and Master Role combination
    YES Active Scanning State and Master Role combination
    YES Passive Scanning State and Slave Role combination
    YES Active Scanning State and Slave Role combination
    YES Initiating State and Master Role combination/Master Role and Master Role combination
root@edison:~#
```

6.2 Scan and connect

From devices that support GAP initiator/observer roles, we can scan and connect to other devices as master of the connection (Central Role), using command line tools or test scripts that use the *bluetoothd* daemon. The command line tools include *bluetoothctl* (recommended) and a couple of alternative tools, *hcitool* and *btmgmt*. For ease of use, we recommend using *bluetoothctl* because it does not require checking traces with *hcidump* to determine if a peer device address is random or static.

6.2.1 bluetoothctl

To use the (recommended) BlueZ stack command line utility *bluetoothctl* to scan and connect, launch a console and do the following:

1. Launch *bluetoothctl* to start scanning for Bluetooth* classic and BLE devices:

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device D0:5F:B8:2A:0C:B9 Moto 360 0CB9
[NEW] Device 00:1F:20:42:27:12 Bluetooth Laser Travel Mouse
[NEW] Device 20:CD:39:A5:3B:62 HTC Fetch
```

2. Use the *scan on* command to scan for Bluetooth* devices to pair with. When you see the desired device, enter `scan off`.

```
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 00:22:D0:3B:2F:2A Polar H7 3B2F2A1C
[NEW] Device 88:0F:10:13:7D:CF MI
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[bluetooth]# scan off
```

Note: Discovery is “interleaved”, which means *bluetoothctl* finds and reports both classic and LE devices without distinction. The BlueZ daemon also hides the type of device from discovery, so there is no need to specify if an address is random or not using this tool.

3. Connect to the device:

```
[bluetooth]# connect 00:22:D0:3B:2F:2A
Attempting to connect to 00:22:D0:3B:2F:2A
[CHG] Device 00:22:D0:3B:2F:2A Connected: yes
Connection successful
[CHG] Device 00:22:D0:3B:2F:2A UUIDs:
        00001800-0000-1000-8000-00805f9b34fb
        00001801-0000-1000-8000-00805f9b34fb
        0000180a-0000-1000-8000-00805f9b34fb
        0000180d-0000-1000-8000-00805f9b34fb
        0000180f-0000-1000-8000-00805f9b34fb
        6217ff4b-fb31-1140-ad5a-a45545d7ecf3
[CHG] Device 00:22:D0:3B:2F:2A Appearance: 0x0341
[bluetooth]#
```

You can use the *hcidump -X* command to check exchanged HCI packets.

6.2.2 hcitool

This command line tool sends raw packets to the controller from the user space and initiates LE Scan.

On the Intel® Edison device, launch two consoles—one to execute the commands and the other to get *hcidump* log traces—then do the following:

1. Use the *hcitool lescan* command to scan for devices. In this example, notice that the Intel® Edison device has detected the reference device, *Polar H7*:

```
root@edison:~# hcitool lescan
LE Scan ...

88:0F:10:13:7D:CF (unknown)
88:0F:10:13:7D:CF MI

00:22:D0:3B:2F:2A (unknown)
00:22:D0:3B:2F:2A Polar H7 3B2F2A1C
^Croot@edison:~#
```

2. You can enter Ctrl+C to stop scanning as soon as you discover the device you are looking for. If the *hcitool lescan* command fails to discover the device, enter the *hciconfig hci0 down* and *hciconfig hci0 up* commands, then reenter the *lescan* command.
3. In the second console, launch the HCI traces using the *hcidump* command:

```
root@edison:~# hcidump -X
HCI sniffer - Bluetooth packet analyzer ver 5.24
device: hci0 snap_len: 1500 filter: 0xffffffff
```

This second console will continue to log traces (Figure 10) whenever there is an exchange of info between the controller and the BlueZ stack.

Figure 10 HCI events

```
> HCI Event: Command Complete (0x0e) plen 4
    LE Set Scan Parameters (0x08|0x000b) ncmd 1
    status 0x00
< HCI Command: LE Set Scan Enable (0x08|0x000c) plen 2
    value 0x01 (scanning enabled)
    filter duplicates 0x01 (enabled)
> HCI Event: Command Complete (0x0e) plen 4
    LE Set Scan Enable (0x08|0x000c) ncmd 1
    status 0x00
> HCI Event: LE Meta Event (0x3e) plen 42
    LE Advertising Report
      ADV_NONCONN_IND - Nonconnectable undirected advertising (3)
      bdaddr 00:22:D0:3B:2F:2A (Public)
      Flags: 0x04
      Unknown type 0xff with 6 bytes data
      Complete local name: 'Polar H7 3B2F2A1C'
      RSSI: -71
> HCI Event: LE Meta Event (0x3e) plen 23
    LE Advertising Report
      ADV_IND - Connectable undirected advertising (0)
      bdaddr 88:0F:10:13:7D:CF (Public)
      Flags: 0x06
      Complete service classes: 0xfee0 0xfee1 0xfee7
      RSSI: -90
```

4. Once a peer device is discovered, a connection can be initiated and a connection handler will be returned. You can identify whether a MAC address is random or public by inspecting the *hcidump*. For example, notice that the Polar H7 device MAC address in Figure 10 shows as "Public". For devices (like the MIO watch) that use random MAC addresses, you will need to use the --random flag with the *lecc* command.

```
root@edison:~# hcitool lecc 00:22:D0:3B:2F:2A
Connection handle 65
root@edison:~#
```

Note:

- You can interrupt the *lescan* command with Ctrl+C. (This will trigger *Set Scan Enable* with the value 0x0 to stop scanning advertising channels.)
- BlueZ is actively scanning, which means that after having scanned the advertised data it will send out a SCAN_REQ to get additional data). This is shown by received LE Meta Event with SCAN_RSP that shows no additional data.
- The Polar H7 advertises its complete local name (POLAR H7 3B2F2A1C) and that it does not support BD/EDR and General connectable (flags = 0x04).
- The Polar H7 is not using a Static Random access. (Random flag is detected in Tx/Rx field of PDU that is not shown. Since it is not random, there is no need to give the `--random` flag while establishing a connection.)

5. Check the *hcidump* traces, where you can see the data exchanges between controller and stack. Logs show that, after a connection was established, the slave initiated a connection update procedure.

Figure 11 hcitool > hcidump traces

```
< HCI Command: LE Create Connection (0x08|0x000d) plen 25
    bdaddr 20:CD:39:A5:3B:62 type 0
    interval 4 window 4 initiator_filter 0
    own_bdaddr_type 0 min_interval 15 max_interval 15
    latency 0 supervision_to 3200 min_ce 1 max_ce 1
> HCI Event: Command Status (0x0f) plen 4
    LE Create Connection (0x08|0x000d) status 0x0b ncmd 1
    Error: ACL Connection Already Exists
< HCI Command: LE Create Connection (0x08|0x000d) plen 25
    bdaddr 00:22:D0:3B:2F:2A type 0
    interval 4 window 4 initiator_filter 0
    own_bdaddr_type 0 min_interval 15 max_interval 15
    latency 0 supervision_to 3200 min_ce 1 max_ce 1
> HCI Event: Command Status (0x0f) plen 4
    LE Create Connection (0x08|0x000d) status 0x00 ncmd 1
> HCI Event: LE Meta Event (0x3e) plen 19
    LE Connection Complete
      status 0x00 handle 65, role master
      bdaddr 00:22:D0:3B:2F:2A (Public)
> ACL data: handle 65 flags 0x02 dlen 16
    L2CAP(d): cid 0x0005 len 12 [psm 0]
      0000: 12 05 08 00 fa 00 90 01  00 00 58 02              ..........X.
< ACL data: handle 65 flags 0x00 dlen 10
    L2CAP(d): cid 0x0005 len 6 [psm 0]
      0000: 13 05 02 00 00 00                                 ......
< HCI Command: LE Connection Update (0x08|0x0013) plen 14
  0000: 41 00 fa 00 90 01 00 00  58 02 01 00 01 00        A.......X.....
> HCI Event: Command Status (0x0f) plen 4
    LE Connection Update (0x08|0x0013) status 0x00 ncmd 1
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 65 packets 1
> HCI Event: LE Meta Event (0x3e) plen 10
    LE Connection Update Complete
      status 0x00 handle 65
      interval 495.00ms, latency 0.00ms, superv. timeout 6000.00ms
```

Note: This is just a connection at the link layer; no GATT procedures are exchanged to browse peer device services and eventually read/write or register to indications/notifications for exposed characteristics.

6.2.3 btmgmt

The *btmgmt* tool lets you discover peer Bluetooth* devices via the *find* command, with options (-l, -b) to specify Low Energy Scanning or Classic Inquiry. In this example, we are pairing an Intel® Edison device with a BLE-enabled Polar* H7 heart rate monitor. If the devices are already paired, disconnect/unpair them and follow the steps to pair and connect the BLE device with the Intel® Edison device.

Note: Use the *btmgmt* code in the BlueZ 5.24 package downloaded by the Yocto recipe. You can also find the *btmgmt* source from the BlueZ git repository at *http://git.kernel.org/cgit/bluetooth/bluez.git/tree/tools*.

1. Scan the BLE devices using the *find* command with the *btmgmt* tool. In the *find* command below, the `-l` option limits the scan to only BLE devices:

```
root@edison:~# ./btmgmt find -l
Discovery started
hci0 dev_found: 00:22:D0:3B:2F:2A type LE Public rssi -61 flags 0x0000
AD flags 0x04 name Polar H7 3B2F2A1C
hci0 dev_found: 88:0F:10:13:7D:CF type LE Public rssi -91 flags 0x0000
AD flags 0x06 eir_len 11
hci0 dev_found: 00:22:D0:3B:2F:2A type LE Public rssi -94 flags 0x0000
AD flags 0x06 name MI
root@edison:~#
```

Without the `-l` option, the tool will scan all Bluetooth* devices (classic and BLE devices):

```
root@edison:~# ./btmgmt find
Discovery started
hci0 dev_found: 00:22:D0:3B:2F:2A type LE Public rssi -60 flags 0x0000
AD flags 0x04 name Polar H7 3B2F2A1C
hci0 dev_found: 88:0F:10:13:7D:CF type LE Public rssi -90 flags 0x0000
AD flags 0x06 eir_len 11
hci0 dev_found: 00:22:D0:3B:2F:2A type LE Public rssi -90 flags 0x0000
AD flags 0x06 name MI
hci0 dev_found: 48:51:B7:15:D1:63 type BR/EDR rssi -35 flags 0x0000
name ubuntu-0
hci0 dev_found: 40:2C:F4:DB:EF:AA type BR/EDR rssi -47 flags 0x0000
name NAGESWAX-MOBL1
root@edison:~#
```

2. Observe the *hcidump* traces and the data exchange between the controller and the BlueZ stack (Figure 12).

Figure 12 btmgmt > hcidump traces

```
> HCI Event: LE Meta Event (0x3e) plen 42
    LE Advertising Report
      ADV_NONCONN_IND - Nonconnectable undirected advertising (3)
      bdaddr 00:22:D0:3B:2F:2A (Public)
      Flags: 0x04
      Unknown type 0xff with 6 bytes data
      Complete local name: 'Polar H7 3B2F2A1C'
      RSSI: -60
> HCI Event: LE Meta Event (0x3e) plen 23
    LE Advertising Report
      ADV_IND - Connectable undirected advertising (0)
      bdaddr 88:0F:10:13:7D:CF (Public)
      Flags: 0x06
      Complete service classes: 0xfee0 0xfee1 0xfee7
      RSSI: -90
> HCI Event: LE Meta Event (0x3e) plen 26
    LE Advertising Report
      SCAN_RSP - Scan Response (4)
      bdaddr 88:0F:10:13:7D:CF (Public)
      Complete local name: 'MI'
      Unknown type 0xff with 8 bytes data
      RSSI: -90
< HCI Command: LE Set Scan Enable (0x08|0x000c) plen 2
    value 0x00 (scanning disabled)
    filter duplicates 0x00 (disabled)
> HCI Event: Command Complete (0x0e) plen 4
    LE Set Scan Enable (0x08|0x000c) ncmd 1
    status 0x00
< HCI Command: Inquiry (0x01|0x0001) plen 5
    lap 0x9e8b33 len 4 num 0
```

3. The *btmgmt* tool doesn't have a command that lets you connect the link layer, but it does have a *pair* command that will send a request to pair using SMP. In this example, the Polar* H7 supports pairing so it will pair with an Intel® Edison device. Try to pair the Intel® Edison device with the Polar* H7. (Some other LE devices might not support pairing, in which case this method would fail.)

```
root@edison:~# ./btmgmt pair -t 1 00:22:D0:3B:2F:2A
Pairing with 00:22:D0:3B:2F:2A (LE Public)
Paired with 00:22:D0:3B:2F:2A (LE Public)
root@edison:~#
```

Note: In the command above, the `-t` option specifies the type of address: 0 for Classic devices, 1 for LE Public, and 2 for LE Random. For the public Polar* H7, we provide a "1"; and for the random MIO Watch, a "2".

4. Check the trace logs to see whether pairing is successful or not (Figure 13).

Figure 13 btmgmt > hcidump traces (successful pairings)

```
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 64 packets 1
> ACL data: handle 64 flags 0x02 dlen 11
    SMP: Pairing Response (0x02)
      capability 0x03 oob 0x00 auth req 0x01
      max key size 0x10 init key dist 0x00 resp key dist 0x01
      Capability: NoInputNoOutput (OOB data not present)
      Authentication: Bonding (No MITM Protection)
      Initiator Key Distribution:
      Responder Key Distribution:  LTK
< ACL data: handle 64 flags 0x00 dlen 21
    SMP: Pairing Confirm (0x03)
      key 73cdd5f503a65a357f085f39c92ae2a8
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 64 packets 1
> ACL data: handle 64 flags 0x02 dlen 21
    SMP: Pairing Confirm (0x03)
      key ca8abf091c803e5c7d098ee2b74f0848
< ACL data: handle 64 flags 0x00 dlen 21
    SMP: Pairing Random (0x04)
      random e110266a9db121a171814e7d5b98882f
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 64 packets 1
> ACL data: handle 64 flags 0x02 dlen 21
    SMP: Pairing Random (0x04)
      random 13c4450ca470c2fb1cd3d5c9b95ca771
< HCI Command: LE Start Encryption (0x08|0x0019) plen 28
  0000: 40 00 00 00 00 00 00 00  00 00 00 00 46 98 aa f7  @...........F...
  0010: 5c 49 2d 1b ee 87 4c d9  fc a4 d0 04              \I-...L.....
> HCI Event: Command Status (0x0f) plen 4
    LE Start Encryption (0x08|0x0019) status 0x00 ncmd 1
> HCI Event: Encrypt Change (0x08) plen 4
    status 0x00 handle 64 encrypt 0x01
> ACL data: handle 64 flags 0x02 dlen 21
    SMP: Encryption Information (0x06)
      LTK d9dc69999e2ddd2ad9da4919a47502fb
```

Note: Link layer connection is established here and then there is an SMP Pairing Request that is established between the Intel® Edison device and the Polar* H7 heart rate monitor.

6.2.4 Python test scripts

BlueZ provides a set a Python scripts (in the *test* folder) that interact with the *bluetoothd* daemon using the exposed D-Bus API, so it is possible to use these scripts, and get the same results as when using *bluetoothctl*.

To scan for both classic and LE devices (interleaved discovery) using python test scripts, do the following:

1. Copy the test package into the Intel® Edison device using the *scp* command and change the permissions of the files.
2. Go to the test folder and launch test-discovery to start the interleaved discovery. In this example, notice that the Intel® Edison device has detected the reference device, Polar H7 ().

Figure 14 The test-discovery Python script

```
root@edison:/usr/lib/bluez/test# ./test-discovery
[ 00:22:D0:3B:2F:2A ]
    Name = Polar H7 3B2F2A1C
    Alias = Polar H7 3B2F2A1C
    Adapter = /org/bluez/hci0
    Appearance = 833
    LegacyPairing = 0
    Paired = 1
    Connected = 1
    UUIDs = dbus.Array([dbus.String(u'00001800-0000-1000-8000-00805f9b34fb'), dbus.Strin
g(u'00001801-0000-1000-8000-00805f9b34fb'), dbus.String(u'0000180a-0000-1000-8000-00805f
9b34fb'), dbus.String(u'0000180d-0000-1000-8000-00805f9b34fb'), dbus.String(u'0000180f-0
000-1000-8000-00805f9b34fb'), dbus.String(u'6217ff4b-fb31-1140-ad5a-a45545d7ecf3')], sig
nature=dbus.Signature('s'), variant_level=1)
    Address = 00:22:D0:3B:2F:2A
    RSSI = -59
    Trusted = 0
    Blocked = 0

[ 88:0F:10:13:7D:CF ]
    Name = MI
    Paired = 0
    Adapter = /org/bluez/hci0
    LegacyPairing = 0
    Alias = MI
    Connected = 0
    UUIDs = dbus.Array([dbus.String(u'0000fee0-0000-1000-8000-00805f9b34fb'), dbus.Strin
g(u'0000fee1-0000-1000-8000-00805f9b34fb'), dbus.String(u'0000fee7-0000-1000-8000-00805f
9b34fb')], signature=dbus.Signature('s'), variant_level=1)
    Address = 88:0F:10:13:7D:CF
    RSSI = -95
    Trusted = 0
    Blocked = 0

^CTraceback (most recent call last):
```

3. Pair the Intel® Edison device with the Polar* H7 heart rate monitor using the *simple-agent* Python script:

```
root@edison:/usr/lib/bluez/test# ./simple-agent hci0 00:22:D0:3B:2F:2A
Agent registered
Device paired
root@edison:/usr/lib/bluez/test#
```

The Intel® Edison and Polar* H7 heart rate monitor devices are paired.

6.2.5 GATTtool

Once you establish the link layer using the *hcitool* tool, it is possible to test BlueZ's GATT client (but not server) functionality using *GATTtool*.

Note: *GATTtool* is not part of the standard Intel® Edison image, but the code is in the BlueZ 5.24 package. You can also find the *GATTtool* source at *http://git.kernel.org/cgit/bluetooth/bluez.git/tree/attrib*. This folder contains all the needed source code to compile *GATTtool*.

Copy the executable into the Intel® Edison device using the *scp* command, then do the following:

1. Launch *hcitool* to scan for BLE devices. In this example, notice that the Intel® Edison device has detected the reference device, *HTC Fetch*:

```
root@edison:~# hcitool lescan
LE Scan ...
00:22:D0:3B:2F:2A Polar H7 3B2F2A1C
20:CD:39:A5:3B:62 (unknown)
20:CD:39:A5:3B:62 HTC Fetch
88:0F:10:13:7D:CF MI
root@edison:~#
```

2. After you identify the device you are looking for, launch *gatttool* with the following command:

   `root@edison:~# ./gatttool -I –b <BT_MAC_address> -t random`

   …where :

   - `-I` specifies interactive mode.
   - `-b` specifies the peer device's Bluetooth* MAC address.
   - `-t random` declares that this is a random MAC address. (This option is required if the MAC address is random address. In this case, HTC-Fetch is public, so this option is not necessary.)

3. Once launched, *gatttool* will start the device shell, which allows you to connect to the end device.

```
root@edison:~# ./gatttool –I –b 20:CD:39:A5:3B:62
[20:CD:39:A5:3B:62][LE]> sec-level medium
[20:CD:39:A5:3B:62][LE]> connect
Attempting to connect to 20:CD:39:A5:3B:62
Connection successful
[20:CD:39:A5:3B:62][LE]>
attr handle: 0x0001, end grp handle: 0x000b uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x0010, end grp handle: 0x0022 uuid: 0000180a-0000-1000-8000-00805f9b34fb
attr handle: 0x0023, end grp handle: 0x0025 uuid: 00001803-0000-1000-8000-00805f9b34fb
attr handle: 0x002d, end grp handle: 0x0031 uuid: 0000180f-0000-1000-8000-00805f9b34fb
attr handle: 0x0032, end grp handle: 0x0036 uuid: 0000180f-0000-1000-8000-00805f9b34fb
attr handle: 0x0037, end grp handle: 0x0047 uuid: 00001804-0000-1000-8000-00805f9b34fb
attr handle: 0x0048, end grp handle: 0xffff uuid: 00001804-0000-1000-8000-00805f9b34fb
[20:CD:39:A5:3B:62][LE]>
```

4. Use the *help* command to list available commands, optional parameters, and explanations:

```
root@edison:~# ./gatttool –help
```

6.3 Advanced audio distribution profile (A2DP)

The Intel® Edison platform supports the A2DP profile, which defines how audio can stream from device A to device B over Bluetooth*. A2DP services are designed to transfer audio streams unidirectionally, in up to 2-channel stereo, from a Bluetooth* host (source) to another Bluetooth* device (a "sink"). An Intel® Edison device may serve as either an **A2DP source** (SRC) or an **A2DP sink** (SNK).

We used an Intel® Edison device as the A2DP source and an LG* Bluetooth* headset as the A2DP sink for this use case:

1. From the Intel® Edison device, scan for the LG* headset.

Figure 15 Scan for the Bluetooth* headset

```
root@edison:/usr/lib/bluez/test# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device D0:5F:B8:2A:0C:B9 Moto 360 0CB9
[NEW] Device D6:DD:17:61:EA:50 OMATE X
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 75:E4:C0:FF:18:ED 75-E4-C0-FF-18-ED
[NEW] Device 5C:2B:1E:09:C6:D9 5C-2B-1E-09-C6-D9
[CHG] Device D6:DD:17:61:EA:50 RSSI: -90
[NEW] Device 88:0F:10:13:8F:A7 MI
[NEW] Device E7:F0:15:7B:E0:96 E7-F0-15-7B-E0-96
[CHG] Device 40:2C:F4:DB:EF:AA RSSI: -59
[CHG] Device D0:5F:B8:2A:0C:B9 RSSI: -92
[NEW] Device E7:A6:61:0F:F3:7D CLARK GIZMO
[CHG] Device 40:2C:F4:DB:EF:AA RSSI: -50
[bluetooth]# scan on
Failed to start discovery: org.bluez.Error.InProgress
[NEW] Device 00:18:6B:4E:A4:B8 LG HBS730
[NEW] Device 88:0F:10:13:7D:CF 88-0F-10-13-7D-CF
[bluetooth]# trust 00:18:6B:4E:A4:B8
[CHG] Device 00:18:6B:4E:A4:B8 Trusted: yes
Changing 00:18:6B:4E:A4:B8 trust succeeded
```

2. From the Intel® Edison device, pair with and connect to the Bluetooth* headset.

Figure 16 Pair/connect the Bluetooth* headset

```
[bluetooth]# pair 00:18:6B:4E:A4:B8
Attempting to pair with 00:18:6B:4E:A4:B8
[CHG] Device 00:18:6B:4E:A4:B8 Connected: yes
[CHG] Device 00:18:6B:4E:A4:B8 UUIDs:
        00001108-0000-1000-8000-00805f9b34fb
        0000110b-0000-1000-8000-00805f9b34fb
        0000110c-0000-1000-8000-00805f9b34fb
        0000110e-0000-1000-8000-00805f9b34fb
        0000111e-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
[CHG] Device 00:18:6B:4E:A4:B8 Paired: yes
Pairing successful
[CHG] Device 00:18:6B:4E:A4:B8 Connected: no
[bluetooth]# connect 00:18:6B:4E:A4:B8
Attempting to connect to 00:18:6B:4E:A4:B8
[NEW] Device DE:81:66:97:CD:CC Zip
[CHG] Device 5C:2B:1E:09:C6:D9 RSSI: -87
[CHG] Device 00:18:6B:4E:A4:B8 Connected: yes
Connection successful
[CHG] Device 40:2C:F4:DB:EF:AA RSSI: -60
[bluetooth]# scan off
[CHG] Device DE:81:66:97:CD:CC RSSI is nil
[CHG] Device 88:0F:10:13:7D:CF RSSI is nil
[CHG] Device 00:18:6B:4E:A4:B8 RSSI is nil
```

3. Verify that your A2DP device (the LG* headset in this case) is recognized in pulse audio as a sink device and that its sink name starts with *bluez_sink*.

Figure 17 Results from uncommented device ID line

```
root@edison:/usr/lib/bluez/test# pactl list sinks
Sink #0
        State: SUSPENDED
        Name: alsa_output.0.analog-stereo
        Description: Loopback Analog Stereo
         ...

Sink #1
        State: SUSPENDED
        Name: bluez_sink.00_18_6B_4E_A4_B8
        Description: LG HBS730
        Driver: module-bluez5-device.c
        Sample Specification: s16le 2ch 44100Hz
        Channel Map: front-left,front-right
        Owner Module: 12
        Mute: no
        Volume: front-left: 65536 / 100% / 0.00 dB,   front-right: 65536 / 100%
/ 0.00 dB
                balance 0.00
        Base Volume: 65536 / 100% / 0.00 dB
        Monitor Source: bluez_sink.00_18_6B_4E_A4_B8.monitor
        Latency: 0 usec, configured 0 usec
        Flags: HARDWARE DECIBEL_VOLUME LATENCY
        Properties:
                bluetooth.protocol = "a2dp_sink"
                device.description = "LG HBS730"
                device.string = "00:18:6B:4E:A4:B8"
```

4. Configure the default sink to use pulse audio server with the following command:

```
root@edison:/usr/lib/bluez/test# pactl set-default-sink bluez_sink.00_
18_6B_4e_A4_B8
```

5. Copy an audio file (*.wav) to the Intel® Edison device using *scp*, and play the audio file using *mplayer*.

Figure 18 Copy audio and playing using mplayer

```
root@edison:/usr/lib/bluez/test# mplayer /home/root/
.ash_history
.config/
Media-Convert_test2_PCM_Mono_VBR_8SS_48000Hz.wav
bluez5-testtools_5.18+git0+cdfdc6b2b6-r0_i586.ipk
root@edison:/usr/lib/bluez/test# mplayer /home/root/Media-Convert_test2_PCM_Mono_VBR_8
SS_48000Hz.wav
Creating config file: /home/root/.mplayer/config
MPlayer2 2.0-379-ge3f5043 (C) 2000-2011 MPlayer Team
162 audio & 361 video codecs

Playing /home/root/Media-Convert_test2_PCM_Mono_VBR_8SS_48000Hz.wav.
Detected file format: WAV format (libavformat)
[wav @ 0x4eef03c0]max_analyze_duration reached
[lavf] stream 0: audio (pcm_u8), -aid 0
Clip info:
 title: short0__2f862hwon1iw6p37h7dtrzihbg163118.tmpsnd
Load subtitles in /home/root/
==========================================================================
Forced audio codec: mad
Opening audio decoder: [pcm] Uncompressed PCM audio decoder
AUDIO: 44100 Hz, 1 ch, u8, 352.8 kbit/100.00% (ratio: 44100->44100)
Selected audio codec: [pcm] afm: pcm (Uncompressed PCM)
==========================================================================
No such audio driver 'alsa'
AO: [pulse] 44100Hz 1ch u8 (1 bytes per sample)
```

You should be able to hear the audio file play on the LG* Bluetooth headset.

6.4 Device identification (DI) profile

The scope of the Device Identification (DI) profile is to provide additional information above and beyond the Bluetooth* Class of Device and also to incorporate the information into Service Discovery Profile (SDP) record and EIR response.

A device can be identified by the following information:

- VendorID source: Indicates if the VendorID refers to Bluetooth* or USB. The allowed values are:
  - 0x0001 means that the VendorID is assigned by the Bluetooth* SIG (*https://www.bluetooth.org/en-us/specification/assigned-numbers/company-identifiers*)
  - 0x0002 means that the VendorID is assigned by the USB Group (https://usb-ids.gowdy.us/read/UD/)
- VendorID (16 bits)
- DeviceId (16 bits)
- Version (16 bits)

6.4.1 Reading and changing the local device identification

The default BlueZ's device information is:

- VendorID Source = USB
- VendorID = 0x1D6B (Linux Foundation)
- ProductID = 0x0246 (BlueZ)
- Version = 0x0518 (5.18)

You can retrieve this information from the local device with the *bluetoothctl* program's *show* command (Figure 4).

Figure 19 Show command

```
[bluetooth]# show
Controller 12:34:56:78:90:AA
        Name: BlueZ 5.18
        Alias: BlueZ 5.18
        Class: 0x000110
        Powered: yes
        Discoverable: no
        Pairable: yes
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        Modalias: usb:v1D6Bp0246d0512
        Discovering: no
```

You can modify this information by editing the */etc/bluetooth/main.conf* file and uncommenting (and changing) the line containing the `DeviceID =`. For example, uncommenting `DeviceID = bluetooth:1234:5678:abcd` gives the result in Figure 20.

Figure 20 Results from uncommented DeviceID line

```
[CHG] Controller 00:11:22:33:55:77 Powered: yes
[bluetooth]# show
Controller 00:11:22:33:55:77
        Name: BlueZ 5.24
        Alias: BlueZ 5.24
        Class: 0x0c0110
        Powered: yes
        Discoverable: no
        Pairable: yes
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
        Modalias: bluetooth:v1234p5678dABCD
        Discovering: no
[bluetooth]# 
```

Note: You must restart the Bluetooth* service and the *bluetoothctl* utility after modifying the */etc/bluetooth/main.conf* file.

6.4.2 Retrieving the peer device's DI information

You can retrieve the DI information of a peer device with the following tools:

- *sdptool:* Available on both versions 4.x and 5.x of BlueZ. (Can be executed on both Intel® Edison device or Ubuntu computer.)
- *bluetoothctl*: Available only in version 5.x of BlueZ. (Can be executed on both Intel® Edison but doesn't exist in Ubuntu 12.04 computer.)

The *sdptool* tool retrieves the information by connecting the SDP server of the peer device (ACL connection):

Figure 21 *sdptool* tool results

```
[DEL] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
root@edison:~# sdptool browse --tree --uuid 0x1200 98:0D:2E:C8:BD:2C
Browsing 98:0D:2E:C8:BD:2C ...
Attribute Identifier : 0x0 - ServiceRecordHandle
  Integer : 0x10002
Attribute Identifier : 0x1 - ServiceClassIDList
  Data Sequence
    UUID16 : 0x1200 - PnPInformation
Attribute Identifier : 0x5 - BrowseGroupList
  Data Sequence
    UUID16 : 0x1002 - PublicBrowseGroup
Attribute Identifier : 0x200 - SpecificationID
  Integer : 0x103
Attribute Identifier : 0x201 - VendorID
  Integer : 0xf
Attribute Identifier : 0x202 - ProductID
  Integer : 0x0
Attribute Identifier : 0x203 - Version
  Integer : 0x0
Attribute Identifier : 0x204 - PrimaryRecord
  Integer : 0x1
Attribute Identifier : 0x205 - VendorIDSource
  Integer : 0x1
Attribute Identifier : 0x8001
```

The *bluetoothctl* tool retrieves the information from the EIR packet received from the peer device (no ACL connection created/needed).

Figure 22 *bluetoothctl* tool retrieval results

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[bluetooth]# info 98:0D:2E:C8:BD:2C
Device 98:0D:2E:C8:BD:2C
        Name: HTC One nag
        Alias: HTC One nag
        Class: 0x5a020c
        Icon: phone
        Paired: yes
        Trusted: no
        Blocked: no
        Connected: no
        LegacyPairing: no
        UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
        UUID: NAP                       (00001116-0000-1000-8000-00805f9b34fb)
        UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
        UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
        UUID: Message Access Server     (00001132-0000-1000-8000-00805f9b34fb)
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Vendor specific           (00006675-7475-7265-6469-616c62756d70)
        Modalias: bluetooth:v000Fp0000d0000
[bluetooth]#
```

6.5 Human interface device (HID) profile

With the HID profile, you can connect any human interface device (mouse, keyboard, etc.) directly without needing to register any service on the Intel® Edison device. To connect an HID, do the following:

1. Unblock the Bluetooth* device to make sure Bluetooth* is enabled, then launch the *bluetoothctl* utility and register an agent, set the default agent, and scan for HID and other Bluetooth* devices.

```
root@edison:~# rfkill unblock bluetooth
root@edison:~# bluetoothctl
[NEW] Controller 98:4F:EE:01:FD:E4 BlueZ 5.18 [default]
[bluetooth]# agent DisplayYesNo
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]# scan on
Discovery started
[CHG] Controller 98:4F:EE:01:FD:E4 Discovering: yes
[NEW] Device 40:2C:F4:DB:EF:AA 40:2C:F4:DB:EF:AA
[CHG] Device 40:2C:F4:DB:EF:AA Name: NAGESWAX-MOBL1
[CHG] Device 40:2C:F4:DB:EF:AA Alias: NAGESWAX-MOBL1
[NEW] Device 5C:51:4F:9E:49:AD DSGAO-MOBL1
[NEW] Device 00:1F:20:42:27:12 00:1F:20:42:27:12
[NEW] Device FC:F8:AE:1E:ED:98 XSDONGX-MOBL2
[NEW] Device 00:1B:DC:06:59:9C RJGUARIN-MOBL1
[CHG] Device 00:1F:20:42:27:12 LegacyPairing: no
[CHG] Device 00:1F:20:42:27:12 Name: Bluetooth Laser Travel Mouse
[CHG] Device 00:1F:20:42:27:12 Alias: Bluetooth Laser Travel Mouse
[CHG] Device 00:1F:20:42:27:12 LegacyPairing: yes
[NEW] Device 40:2C:F4:86:72:54 40:2C:F4:86:72:54
[NEW] Device B8:76:3F:AB:7E:D1 B8:76:3F:AB:7E:D1
[bluetooth]# pair 00:1F:20:42:27:12
```

2. Pair the devices and verify that the pairing is successful:

```
[bluetooth]# pair 00:1F:20:42:27:12
Attempting to pair with 00:1F:20:42:27:12
[CHG] Device 00:1F:20:42:27:12 Connected: yes
[CHG] Device 00:1F:20:42:27:12 Modalias: usb:v046DpB008d0318
[CHG] Device 00:1F:20:42:27:12 UUIDs:
        00001124-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
[CHG] Device 00:1F:20:42:27:12 Paired: yes
Pairing successful
[CHG] Device 00:1F:20:42:27:12 Connected: no
[bluetooth]# scan off
[CHG] Device B8:76:3F:AB:7E:D1 RSSI is nil
[CHG] Device 40:2C:F4:86:72:54 RSSI is nil
[CHG] Device 00:1B:DC:06:59:9C RSSI is nil
Discovery stopped
[CHG] Controller 98:4F:EE:01:FD:E4 Discovery: no
[bluetooth]# pair 00:1F:20:42:27:12
Attempting to pair with 00:1F:20:42:27:12
Failed to pair: org.bluez.Error.AlreadyExists
```

3. Use the *connect* command to connect the mouse as an input device to the Intel® Edison device:

```
[bluetooth]# connect 00:1F:20:42:27:12
Attempting to connect to 00:1F:20:42:27:12
[CHG] Device 00:1F:20:42:27:12 Connected: yes
[bluetooth]# [ 3144.632878] hid-generic 0005:046D:B008.0001: unknown...
Connection successful
[bluetooth]# info 00:1F:20:42:27:12
Device 00:1F:20:42:27:12
        Name: Bluetooth Laser Travel Mouse
        Alias: Bluetooth Laser Travel Mouse
        Class: 0x002580
        Icon: input-mouse
        Paired: yes
        Trusted: no
        Blocked: no
        Connected: yes
        LegacyPairing: yes
        UUID: Human Interface Device... (00001124-0000-1000-8000-
00805f9b34fb)
        UUID: PnP Information
        Modalias: usb:v046DpB008d0318   (00001200-0000-1000-8000-
00805f9b34fb)
[bluetooth]# more /dev/input/event1
Invlaid command
[bluetooth]# quit
[DEL] Controller 98:4F:EE:01:FD:E4 BlueZ 5.18 [default]
```

4. Check the kernel logs to verify that the device is correctly connected. The example below is for a Bluetooth mouse:

```
[15337.082135] hid-generic 0005:0A5C:2004.0001: unknown main item tag 0x0
[15337.083809] input: MoGo Mouse BT as
/devices/pci0000:00/0000:00:04.1/tty/ttyMFD0/hci0/hci0:12/input1
[15337.086105] hid-generic 0005:0A5C:2004.0001: input,hidraw0:
BLUETOOTH HID v3.00 Mouse [MoGo Mouse BT] on 43:34:1b:00:1f:ac
```

5. When you make a connection, a */dev/input/eventX* file is created. Use the *more* command to check this event file (Figure 23) and verify that events are correctly received.

Figure 23 Raw data from the event file using the "more" command

```
root@edison:/# more /dev/input/event2
s@eTÿ´s@eTÿóÿÿÿs@eTÿs@eTñÿÿÿÿs@eTñÿÿÿÿs@eTñs@eTÜò÷ÿÿÿs@eTÜòøÿÿÿs@eTÜòs@eTúÿÿÿs@e
Tüÿÿÿs@eTs@eTêÿÿÿs@eTçÿÿÿs@eTç@eTOóÿÿÿs@eTOûÿÿÿs@eTOs@eTpÊäÿÿÿs@eTpÊøÿÿÿs@eTpÊs@
eTp÷òÿÿÿs@eTp÷þÿÿÿs@eTp÷s@eT‡åÿÿÿs@eT‡s@eTOóÿÿÿs@eTOs@eTyÓçÿÿÿs@eTyÓs@eTyÓs@eTÿþ
ôÿÿÿs@eTÿþs@eTÿþs@eTZ*ëÿÿÿs@eTZs@eTZ*s@eTFv÷ÿÿÿs@eTFvs@eTFvs@eT8ñÿÿÿs@eT8
s@eT+Ùs@eTýÿÿÿs@eTs@eTs@eTØs@eTØs@eT9s@eT9s@eT9s@eT
s@eT
 s@eT; s@eT; s@eT; s@eTÛe .s@eTÛe Os@eTù
                                  s@eTÛe s@eTÏ s@eTÏ s@eTÏ s@eT;¾ 3s@eT;¾ s@eT;¾
 s@eT·é s@eT·é ÿÿÿÿs@eT·é s@eTS
2s@eTS
--More--
root@edison:/#
```

6. To decode these incoming events, use this *freedesktop* utility: *http://cgit.freedesktop.org/~whot/evtest*. Either compile the code for Intel® Edison, copy the binary to the Intel® Edison device, and then launch *freedesktop*; or copy the *freedesktop* utility into the Intel® Edison device, and then launch it.

Note: For compilation instructions, visit *http://cgit.freedesktop.org/~whot/evtest/tree/INSTALL.*

```
root@edison:~# ./evtest /dev/input/event1
Input driver version is 1.0.1
Input device ID: bus 0x5 vendor 0x1131 product 0x1616 version 0x410
Input device name: "Bluetooth Keyboard"
Supported events:
Event type 0 (Sync)
 ...
Event type 20 (Repeat)
Testing ... (interrupt to exit)
Event: time 1404754634.580274, type 4 (Misc), code 4 (ScanCode), value 70014
Event: time 1404754634.580274, type 1 (Key), code 16 (Q), value 1
Event: time 1404754634.580274, -------------- Report Sync ------------
Event: time 1404754634.736606, type 4 (Misc), code 4 (ScanCode), value 70014
Event: time 1404754634.736606, type 1 (Key), code 16 (Q), value 0
Event: time 1404754634.736606, -------------- Report Sync ------------
Event: time 1404754645.460014, type 4 (Misc), code 4 (ScanCode), value 7001a
Event: time 1404754645.460014, type 1 (Key), code 17 (W), value 1
```

After listing the mapping between events and character, *evtest* will listen for incoming events and trace them.

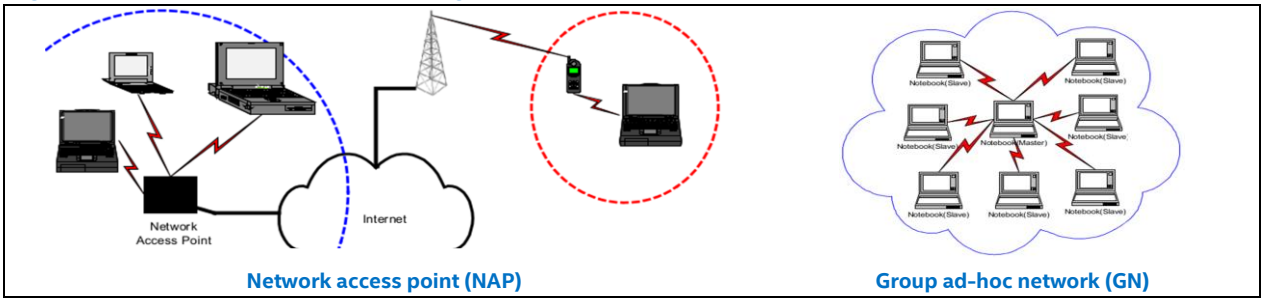
6.6 Personal area networking (PAN) profile

The personal area networking (PAN) profile describes how two or more Bluetooth-enabled devices can form a network and access other networks through a network access point (NAP). The PAN profile defines how to use the Bluetooth Network Encapsulation Protocol (BNEP) to provide networking capabilities for Bluetooth devices. PAN profile roles include the following:

- NAP: Network access point.
- GN: Group ad-hoc network.
- PANU: Personal area network user.

NAP and GN offer services for different networking requirements. NAP provides network services to each Bluetooth device connected, while GN allows two or more devices to become part of an ad-hoc network (Figure 24).

Figure 24 PAN service networking models



Network access point (NAP) Group ad-hoc network (GN)

For Intel® Edison software, we have validated the PAN profile in NAP and GN. However, to perform a PAN test, you will need to download test scripts that are part of the BlueZ package, but which are not included in the Intel® Edison image. (You can also find these at *http://git.kernel.org/cgit/bluetooth/bluez.git/tree/test* in the *test* folder.)

- Select the role with the `-s` option.
- Compress and copy the BlueZ test package into the Intel® Edison board via *scp*.
- Unzip and copy BlueZ test package into the Intel® Edison board.
- Enable Bluetooth* as described in chapter 4 Scanning and Connecting Device.

After you have performed the above steps, you can perform the PAN test between a Linux* host PC and an Intel® Edison device, or between two Intel® Edison devices.

6.6.1 PAN test between Linux* host PC and Intel® Edison device

To perform the PAN test between a Linux* host PC and an Intel® Edison device, do the following:

1. Start *connman* and enable Bluetooth* on both the Intel® Edison device and on the Linux* PC.

   On the Intel® Edison device:

```
root@edison:~# systemctl start connman
root@edison:~# connmanctl enable Bluetooth
root@edison:~# hciconfig
hci0:   Type: BR/EDR Bus: UART
        BD Address: 00:11:22:33:55:77 ACL MTU: 1021:8 SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:41408 acl:308 sco:0 events:300 errors:0
        TX bytes:31530 acl:270 sco:0 commands:65 errors:0
root@edison:~#
```

2. On the Linux* PC, the “RSSI” line provides the Bluetooth* address of the PC.

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[bluetooth]# agent DisplayYesNo
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]# discoverable on
Changing discoverable on succeeded
[CHG] Controller 00:11:22:33:55:77 Discoverable: yes
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device E8:BE:82:BE:75:19 E8-BE-82-BE-75-19
[NEW] Device D5:B3:ED:7E:A5:83 D5-B3-ED-7E-A5-83
[CHG] Device D5:B3:ED:7E:A5:83 Name: Halitoshi
[CHG] Device D5:B3:ED:7E:A5:83 Alias: Halitoshi
[NEW] Device C8:F7:33:2C:A8:93 JSWALKEN-MOBL1
[NEW] Device 3C:15:C2:DC:E9:41 adaniele-mac01
[CHG] Device 48:51:B7:15:D1:63 RSSI:  -35
[CHG] Device 48:51:B7:15:D1:63 UUIDs:
        0000112d-0000-1000-8000-00805f9b34fb
        00001112-0000-1000-8000-00805f9b34fb
        00001234-0000-1000-8000-00805f9b34fb
        00001700-0000-1000-8000-00805f9b34fb
        00001701-0000-1000-8000-00805f9b34fb
        00001708-0000-1000-8000-00805f9b34fb
[CHG] Device 48:51:B7:15:D1:63 Paired: yes
Pairing successful
[CHG] Device 40:2C:F4:DB:EF:AA Connected: no
[CHG] Device 40:2C:F4:DB:EF:AA Connected: yes
[CHG] Device 40:2C:F4:DB:EF:AA Connected: no
[bluetooth]
```

3. Pair the devices.

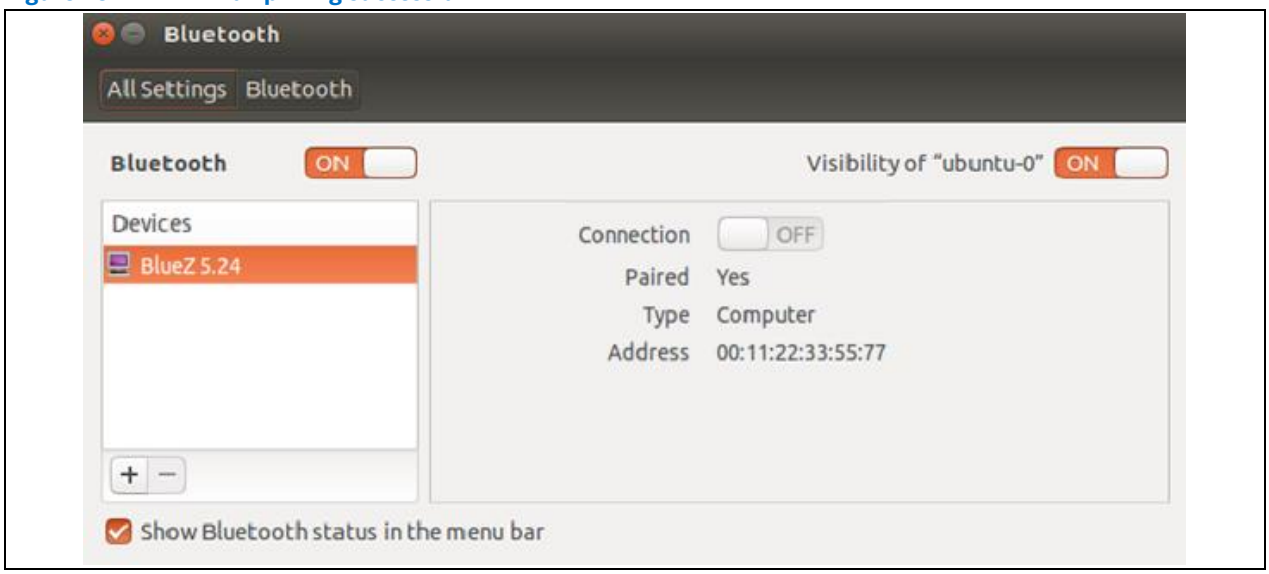
   On the Intel® Edison device:

```
[bluetooth]# pair 00:11:22:33:55:77
Attempting to pair with 00:11:22:33:55:77
[CHG] Device 00:11:22:33:55:77 Connected: yes
[CHG] Device 00:11:22:33:55:77 UUIDs:
        0000110c-0000-1000-8000-00805f9b34fb
        0000110e-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
        00001800-0000-1000-8000-00805f9b34fb
        00001801-0000-1000-8000-00805f9b34fb
        0000a004-0000-1000-8000-00805f9b34fb
        feee74dc-a8de-3196-1149-d43596c00a4f
[CHG] Device 00:11:22:33:55:77 Paired: yes
Pairing successful
[CHG] Device 00:11:22:33:55:77 Connected: no
[CHG] Device E4:F5:9F:82:56:94 RSSI: -89
[bluetooth]# scan off
[CHG] Device E4:F5:9F:82:56:94 RSSI is nil
[CHG] Device 7C:7A:91:F2:6E:84 RSSI is nil
[CHG] Controller 00:11:22:33:55:66 Discovering: no
Discovery stopped
[bluetooth]#
```

   On the Linux* PC:

   You will see that pairing is successful between the Intel® Edison device and the Linux PC when the right pane of the Bluetooth window indicates that *Paired* equals *Yes* (Figure 25).

Figure 25 Linux pairing successful



4. Create a bridge and configure its address on the Intel® Edison device:

```
root@edison:~# brctl addbr br0
root@edison:~# ip addr add 192.168.10.1 dev br0
root@edison:~# ip link set br0 up
root@edison:~# ifconfig
br0:    Link encap:Ethernet  HWaddr e2:68:df:c3:6f:1f
        inet addr:192.168.10.1 Bcast:0.0.0.0 Mask: 255.255.255.255
        inet6 addr: fe80::e086:dfff:fec3:6f1f/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (o.0 B)  TX bytes:4284 (4.1 KiB)
```

   Once the bridge has been created on the Intel® Edison device, you can check with the *ifconfig* command.

5. Launch the PAN test script for NAP service; this will register the NAP service so a peer device will see this service available. Use one of the following commands:

```
root@edison:~/test-bluez# ./test-pan -s nap br0
root@edison:~/test-bluez# ./test-nap br0
```

   The bnep0 interface will be added to the br0 bridge (the same one created at step 0).

   Note: This test script will only keep NAP registered for a few minutes then disconnect. If you need more time, you will have to modify the script.

6. Connect to the peer device as a PAN user. Before you can do this from a Linux* PC, you will need to install the BlueZ package (if you haven't already) and use the *pand* command.

   Note: The *pand* service interface is available in BlueZ 4 but not in BlueZ 5.

   a. To install the BlueZ package, enter the following: `sudo apt-get install bluez-compat`.
   b. Use the *pand* command to connect. In this example, 00:11:22:33:55:77 is the Intel® Edison device's Bluetooth* MAC address.

```
user1@ndg05:~/$ sudo pand -n --connect 00:11:22:33:55:77 --service NAP
pand[2990]: Bluetooth PAN daemon version 4.101
pand[2990]: Connecting to 00:11:22:33:55:77
pand[2990]: bnep0 connected
```

7. If everything succeeds, the *bnep* interface will be added to the bridge in Intel® Edison; the *bnep* interface will be listed on the Linux* PC as well. Enter the *ifconfig* command on each device to verify.

   On an Intel® Edison device:

```
root@edison:~# ifconfig -a

bnep0   Link encap:Ethernet  HWaddr 00:43:34:b1:de:ad
        inet6 addr: fe80::243f:34ff:feb1:dead/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:16 (16.0 B)  TX bytes:64 (64.0 B)
```

8. On a Linux* PC:

```
user1@ndg05:~/$ ifconfig -a
bnep0    Link encap:Ethernet  HWaddr 00:43:34:b1:de:ad
         inet6 addr: fe80::243f:34ff:feb1:dead/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:100 (100.0 B)  TX bytes:2443 (2.4 KB)

eth0     Link encap:Ethernet  HWaddr 00:43:34:b1:de:ad
         inet addr:10.3.83.69  Bcast:10.3.83.255  Mask 255.255.255.0
         inet6 addr: fe80::243f:34ff:feb1:dead/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:1307577 (1.3 MB)  TX bytes:60367 (60.3 KB)
         Interrupt:20 Memory:f7d00000- f7d20000

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask 255.0.0.0
         inet6 addr:  ::1/128 Scope:Host
         UP LOOPBACK RUNNING MULTICAST  MTU:65536  Metric:1
```

Note: Sometimes bnep0 will not be visible in *ifconfig* until you execute the following command:
`sudo ip link set bnep0 up`

9. Configure both *bnep* interfaces with an IP address and try to ping them.
On a Linux* PC:

```
user1@ndg05:~/$ sudo ip link set bnep0 up
user1@ndg05:~/$ sudo ip addr add 192.168.10.10 dev bnep0
user1@ndg05:~/$ sudo ip route add 192.168.10.0/24 via 192.168.10.10
```

On a Edison device :

```
root@edison:~# ip addr add 192.168.10.2 dev bnep0
root@edison:~# ip route add 192.168.10.0/24 via 192.168.10.1
```

10. With a connection established between the Intel® Edison device and the Linux* PC, you can ping the Intel® Edison device from the Linux* PC (and vice versa).

    Note: Pinging the Access Point from the Linux* PC over Bluetooth to the Intel® Edison device provides basic verification. Successfully accessing the web from the PC provides functional verification.

    From the Intel® Edison device, pinging the Linux* PC (IP address: 192.168.10.10):

```
root@edison:~# ping 192.168.10.10
PING 192.168.10.10 (192.168.10.10): 56 data bytes
64 bytes from 192.168.10.10: seq=0 ttl=64 time=19.563 ms
64 bytes from 192.168.10.10: seq=1 ttl=64 time=11.526 ms
64 bytes from 192.168.10.10: seq=2 ttl=64 time=13.279 ms
...
```

    From the Linux* PC, pinging the Intel® Edison device (IP address: 192.168.10.2):

```
user1@ndg05:~/$ ping 192.168.10.2

PING 192.168.10.2 (192.168.10.102) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=20.1 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=22.4 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=7.08 ms
...
```

6.6.2 PAN test between two Intel® Edison devices

This section explains how to test PAN, using one Intel® Edison device as PANU and the second as PAN-NAP.

Note: While you can use *connman* to connect a NAP service on a peer device, you cannot use *connman* to discover, scan, or pair devices; you must use normal BlueZ tools beforehand for these basic operations.

To perform the PAN test between two Intel® Edison devices, do the following:

1. Enable Bluetooth* on both Intel® Edison devices. (See Chapter 4 Scanning and Connecting Device.)
2. Start *connman* and enable Bluetooth* on both devices using *connmanctl*.

```
root@edison:~# systemctl start connman
root@edison:~# connmanctl enable bluetooth
Enabled bluetooth
root@edison:~#
```

3. Use *hciconfig* to verify that Bluetooth* is enabled on both devices:

```
root@edison:~# hciconfig
hcio:   Type: BR/EDR Bus: UART
        BD Address: 00:11:22:33:55:77 ACL MTU: 1021:8 SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:41408 acl:308 sco:0 events:300 errors:0
        TX bytes:31530 acl:270 sco:0 commands:65 errors:0
root@edison:~#
```

4. Prepare the second device (PAN-NAP) for pairing:

   a. If the second device does not have the *bluez-test* packages, download *test-bluez-5.18.tar.gz* to the device using the *scp* command and untar it to create the test folder containing the test code.
   b. In a console, enter the following to register the NAP service:

```
root@edison:~# ./test/test-nap br0
Server for nap registered to br0
Press CTRL-C to disconnect
```

   c. Add a bridge using the *brctl* command and configure the bridge to use a static IP address:

```
root@edison:~# brctl addbr br0
root@edison:~# ifconfig br0 192.168.1.1
root@edison:~#
```

   d. Launch the Bluetooth* controller utility, then set discoverable on and register an agent.

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 00:11:22:33:55:66 BlueZ 5.24
[bluetooth]# discoverable on
Changing discoverable on succeeded
[CHG] Controller 00:11:22:33:44:77 Discoverable: yes
[bluetooth]# agent DisplayOnly
Agent registered
[bluetooth]# default-agent
Default agent request successful
[CHG] Controller 00:11:22:33:55:77 Discoverable: no
[bluetooth]#
```

5. To pair the first device (PANU) with the second device (PAN-NAP), use *bluetoothctl* to register an agent:

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:66 BlueZ 5.24 [default]
[NEW] Device 40:2C:F4:60:C1:02 MKODANDX-MOBL
[NEW] Device B4:B6:76:4F:60:F4 Edison-temp-2-0
[NEW] Device E4:F5:9F:82:56:94 Force
[NEW] Device 30:76:6F:50:DB:FC LGA340
[NEW] Device 7C:7A:91:F2:6E:84 GPHATAK-MOBL1
[NEW] Device 00:02:72:C9:5C:A4 ndg-leb-sys-0
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device C6:22:DD:95:29:E1 tkr
[NEW] Device 00:1F:20:8E:7C:45 Dell Travel Mouse WM524
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[bluetooth]# agent DisplayOnly
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]#
```

6. Scan for the second (PAN-NAP) device:

```
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:44:66 Discovering: yes
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device 5C:51:4F:9E:49:AD DSGAO-MOBL1
[NEW] Device 00:11:22:33:44:77 BlueZ 5.24
[NEW] Device FC:F8:AE:1E:ED:98 XSDONGX-MOBL2
[NEW] Device 3C:5A:37:4C:3A:11 3C:5A:37:4C:3A:11
[CHG] Device 3C:5A:37:4C:3A:11 LegacyPairing: no
[CHG] Device 3C:5A:37:4C:3A:11 Name: SHG-A777
[CHG] Device 3C:5A:37:4C:3A:11 Alias: SHG-A777
[CHG] Device FC:F8:AE:1E:ED:98 RSSI: -85
[CHG] Device FC:F8:AE:1E:ED:98 RSSI: -77
[CHG] Device 3C:5A:37:4C:3A:11 LegacyPairing: yes
[bluetooth]#
```

7. Pair the devices:

```
[bluetooth]# pair 00:11:22:33:55:77
Attempting to pair with 00:11:22:33:55:77
[CHG] Device 00:11:22:33:55:77 Connected: yes
Request confirmation
[agent] Confirm passkey 804573 (yes/no): yes
[CHG] Device 00:11:22:33:55:77 UUIDs:
        0000110c-0000-1000-8000-00805f9b34fb
        0000110e-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
        00001800-0000-1000-8000-00805f9b34fb
        00001801-0000-1000-8000-00805f9b34fb
        0000a004-0000-1000-8000-00805f9b34fb
        feee74dc-a8de-3196-1149-d43596c00a4f
[CHG] Device 00:11:22:33:55:77 Paired: yes
Pairing successful
[CHG] Device 00:11:22:33:55:77 Connected: no
[CHG] Device E4:F5:9F:82:56:94 RSSI: -89
[bluetooth]# scan off
[CHG] Device E4:F5:9F:82:56:94 RSSI is nil
[CHG] Device 7C:7A:91:F2:6E:84 RSSI is nil
[CHG] Device B4:B6:76:4F:60:F4 RSSI is nil
[CHG] Device 00:02:72:C9:5C:A4 RSSI is nil
[CHG] Device 00:11:22:33:55:77 RSSI is nil
[CHG] Device 40:2C:F4:D8:EF:AA RSSI is nil
[CHG] Device DA:0D:F3:BA:56:CB RSSI is nil
[CHG] Controller 00:11:22:33:55:66 Discovering: no
Discovery stopped
[bluetooth]# exit
Agent unregistered
```

8. From the second (PAN-NAP) device, use the *trust* command to trust the first (PANU) device:

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:44:66 BlueZ 5.18 [default]
[NEW] Device 00:11:22:33:44:77 BlueZ 5.24
[bluetooth]# trust 00:11:22:33:44:77
[CHG] Device 00:11:22:33:44:77 Trusted: yes
Changing 00:11:22:33:44:77 trust succeeded
[bluetooth]#
```

9. From the first (PANU) device, connect to the second (PAN-NAP) device using the *connmanctl* utility:

```
root@edison:~# connmanctl
connmanctl> services
    BlueZ 5.24      bluetooth_001122335577_001122335566
connmanctl> connect bluetooth_001122335577_001122335566
    /net/connman/service/bluetooth_001122335577_001122335566: connected
```

10. If the second (PAN-NAP) device does not trust the first device, you will have to authenticate the first device:

```
connmanctl> config bluetooth_001122335577_001122335566 --ipv4 manual 192.168.1.10
```

11. At this point, each Intel® Edison device should have its own *bnep* interface, in a place where you can configure and test it with *ping* or *iperf*. For example, on the first device (PANU):

```
root@edison:~# ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10): 56 data bytes
64 bytes from 192.168.1.10: seq=0 ttl=64 time=0.463 ms
64 bytes from 192.168.1.10: seq=1 ttl=64 time=0.295 ms
64 bytes from 192.168.1.10: seq=2 ttl=64 time=0.295 ms
64 bytes from 192.168.1.10: seq=3 ttl=64 time=0.296 ms
64 bytes from 192.168.1.10: seq=4 ttl=64 time=0.301 ms
64 bytes from 192.168.1.10: seq=5 ttl=64 time=0.294 ms
64 bytes from 192.168.1.10: seq=6 ttl=64 time=0.296 ms
64 bytes from 192.168.1.10: seq=7 ttl=64 time=0.294 ms
64 bytes from 192.168.1.10: seq=8 ttl=64 time=0.294 ms
64 bytes from 192.168.1.10: seq=9 ttl=64 time=0.295 ms
64 bytes from 192.168.1.10: seq=10 ttl=64 time=0.293 ms
64 bytes from 192.168.1.10: seq=11 ttl=64 time=0.292 ms
64 bytes from 192.168.1.10: seq=12 ttl=64 time=0.294 ms
64 bytes from 192.168.1.10: seq=13 ttl=64 time=0.369 ms
64 bytes from 192.168.1.10: seq=14 ttl=64 time=0.296 ms
```

Note: You can also use the *ping* command on the second Intel® Edison device for verification.

6.7 Serial port profile (SPP)

SPP (serial port profile), which is based on ETSI 07.10 and RFCOMM protocol, defines how two Bluetooth*-enabled devices create a virtual/emulated serial port connection and communicate with each other.

- SDP is the Bluetooth* Service Discovery Protocol, which allows devices to provide browsing services to each other.
- Devices accepting an incoming connection over RFCOMM expose a record in SDP for SPP indicating that the RFCOMM channel is listening.
- Devices initiating a connection will first search for SPP records on the peer device database and in turn may initiate a connection to the RFCOMM server channel on a peer device.

We can test this by creating a virtual serial port between two devices via Bluetooth* and using SPP to send info from one Bluetooth* device to another. We can verify using the following methods:

- SPP verification using DBUS APIs (recommended)
- SPP verification using the RFCOMM tool

Note: SPP verification using DBUS interface APIs is the preferred way to test and use SPP because it exposes the file descriptor (fd) of the connection in the user space, and it can be directly used to send and receive data over SPP. The RFCOMM tool is deprecated if the fd is available in the user space. RFCOMM is used to set up, maintain, and inspect the RFCOMM configuration of the Bluetooth* subsystem in the Linux kernel. (an emulated TTY device file is created and it has to be opened to read/write data over SPP, whereas when using DBUS interfaces opening of the device file is not needed as fd of connection is already available with DBUS interfaces in userspace).

To have the connection method called in the test-profile script (or in the modified version), modify the Intel® Edison device's DBUS BlueZ policy file */etc/dbus-1/system.d/bluetooth.conf* using the *vi* editor:

```
root@edison:~# vi /etc/dbus-1/system.d/bluetooth.conf
```

If the *bluetooth.conf* file (Figure 26) doesn't have the line *<allow send_interface="org.bluez.Profile1"/>*, add this line and save the file.

Figure 26 Editing the bluetooth.conf file

```
ram@ram-desktop: ~
<!-- This configuration file specifies the required security policies
     for Bluetooth core daemon to work. -->

<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>

  <!-- ../system.conf have denied everything, so we just punch some holes -->

  <policy context="default">
    <allow own="org.bluez"/>
    <allow own="org.bluez.obex"/>
    <allow send_destination="org.bluez"/>
    <allow send_destination="org.bluez.obex"/>
    <allow send_interface="org.bluez.Agent1"/>
    <allow send_interface="org.bluez.MediaEndpoint1"/>
    <allow send_interface="org.bluez.Profile1"/>
    <allow send_interface="org.bluez.ThermometerWatcher1"/>
    <allow send_interface="org.bluez.AlertAgent1"/>
    <allow send_interface="org.bluez.HeartRateWatcher1"/>
    <allow send_interface="org.bluez.CyclingSpeedWatcher1"/>
  </policy>

</busconfig>
~
```

6.7.1 SPP verification using DBUS APIs

It is possible to get at the application layer of the RFCOMM socket file using the *test-profile* python script in the BlueZ test folder (*http://git.kernel.org/cgit/bluetooth/bluez.git/tree/test*). We have modified the original file slightly to loopback received data on the other side to verify SPP, and renamed the modified file *SPP-loopback.py*. This file is included in *Appendix A: SPP-loopback.py* and is also available for download at:

- *http://downloadmirror.intel.com/24909/eng/SPP-loopback.pyhttp://downloadmirror.intel.com/24909/eng/SPP-loopback.py*

Copy this script into your Intel® Edison device. Find the changes in the *test-profile.py* file, make the necessary changes, and push the *SPP_loopback.py* file into your Intel® Edison device using *scp*.

Before running the *SPP-loopback.py* script, notice that the *bluetoothctl* utility does not display the serial profile (Figure 27).

Figure 27 Serial port absent before running SPP-loopback.py

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[bluetooth]# show
Controller 00:11:22:33:55:77
        Name: BlueZ 5.24
        Alias: BlueZ 5.24
        Class: 0x0c0110
        Powered: yes
        Discoverable: no
        Pairable: yes
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
        Modalias: usb:v1D6Bp0246d0518
        Discovering: no
[bluetooth]# 
```

After you run the *SPP-loopback.py* script on your Intel® Edison device, the serial port does display (Figure 28).

Figure 28 Serial port present after running SPP-loopback.py

```
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[bluetooth]# show
Controller 00:11:22:33:55:77
        Name: BlueZ 5.24
        Alias: BlueZ 5.24
        Class: 0x0c0110
        Powered: yes
        Discoverable: no
        Pairable: yes
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
        UUID: Serial Port               (00001101-0000-1000-8000-00805f9b34fb)
        Modalias: usb:v1D6Bp0246d0518
        Discovering: no
[bluetooth]# 
```

Search for the peer devices (we have taken Android* device) with the *discoverable on* and *scan on* commands (Figure 29).

Figure 29 Search for peer devices

```
[CHG] Controller 00:11:22:33:55:77 Discovering: no
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[CHG] Device 74:A8:E3:18:C9:B6 RSSI: -88
[CHG] Device 55:5E:CA:03:CD:4D RSSI: -89
[CHG] Device D9:AB:B4:0F:3D:A7 RSSI: -85
[CHG] Device 48:51:B7:15:D1:63 RSSI: -33
[CHG] Device 00:1B:DC:06:59:9C RSSI: -71
[CHG] Device 40:2C:F4:DB:EF:AA RSSI: -59
[CHG] Device FC:F8:AE:1E:ED:98 RSSI: -79
[CHG] Device 80:86:F2:54:7C:24 RSSI: -68
[CHG] Device 40:2C:F4:86:72:54 RSSI: -75
[CHG] Device 5C:51:4F:9E:49:AD RSSI: -76
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[CHG] Device B8:76:3F:AB:7E:D1 RSSI: -78
[CHG] Device C8:F7:33:8B:48:08 RSSI: -79
[bluetooth]# 
```

Figure 30 Still searching

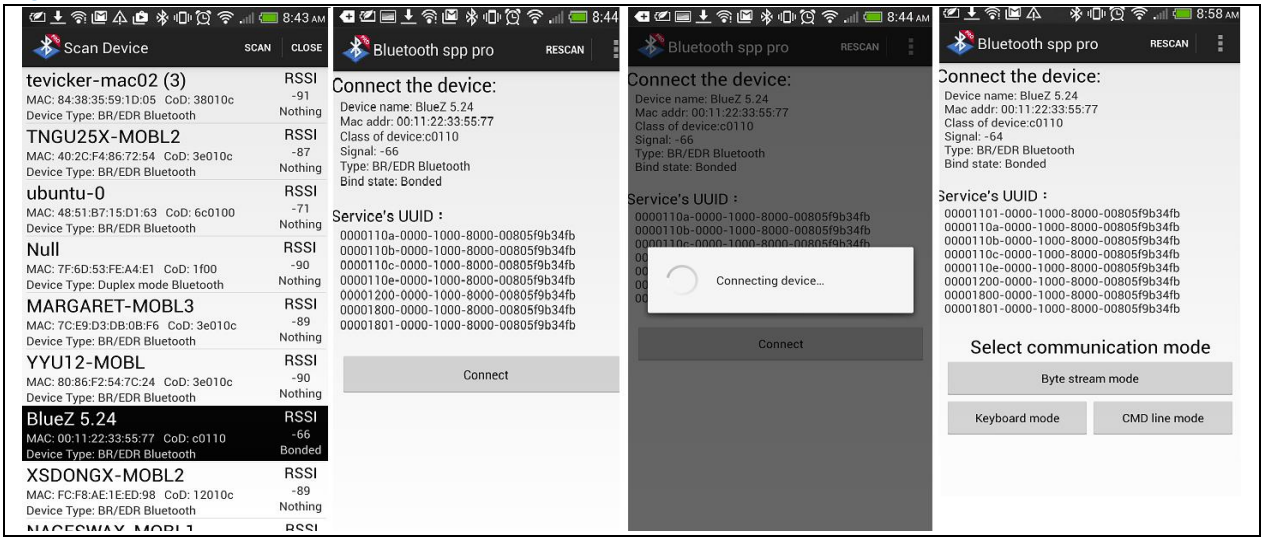
```
[NEW] Device FC:F8:AE:1E:ED:98 XSDONGX-MOBL2
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device 54:2D:54:61:13:7D 54-2D-54-61-13-7D
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
Request confirmation
[agent] Confirm passkey 454749 (yes/no): yes
[CHG] Device 98:0D:2E:C8:BD:2C Modalias: bluetooth:v000Fp0000d0000
[CHG] Device 98:0D:2E:C8:BD:2C UUIDs:
        00001105-0000-1000-8000-00805f9b34fb
        0000110a-0000-1000-8000-00805f9b34fb
        0000110c-0000-1000-8000-00805f9b34fb
        00001112-0000-1000-8000-00805f9b34fb
        00001116-0000-1000-8000-00805f9b34fb
        0000111f-0000-1000-8000-00805f9b34fb
        0000112f-0000-1000-8000-00805f9b34fb
        00001132-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
        00001800-0000-1000-8000-00805f9b34fb
        00001801-0000-1000-8000-00805f9b34fb
[CHG] Device 98:0D:2E:C8:BD:2C Paired: yes
```

Pair with the Android* device. Request Android* device to pair with the Intel® Edison board. Confirm in the Intel® Edison for pairing or Android* device can be added as trusted device.

Download the *Bluetooth spp pro* (a free app) from the Google playstore. Make sure Bluetooth* is enabled. After you install this application, launch it and give connect request from the application to the Intel ® Edison device.

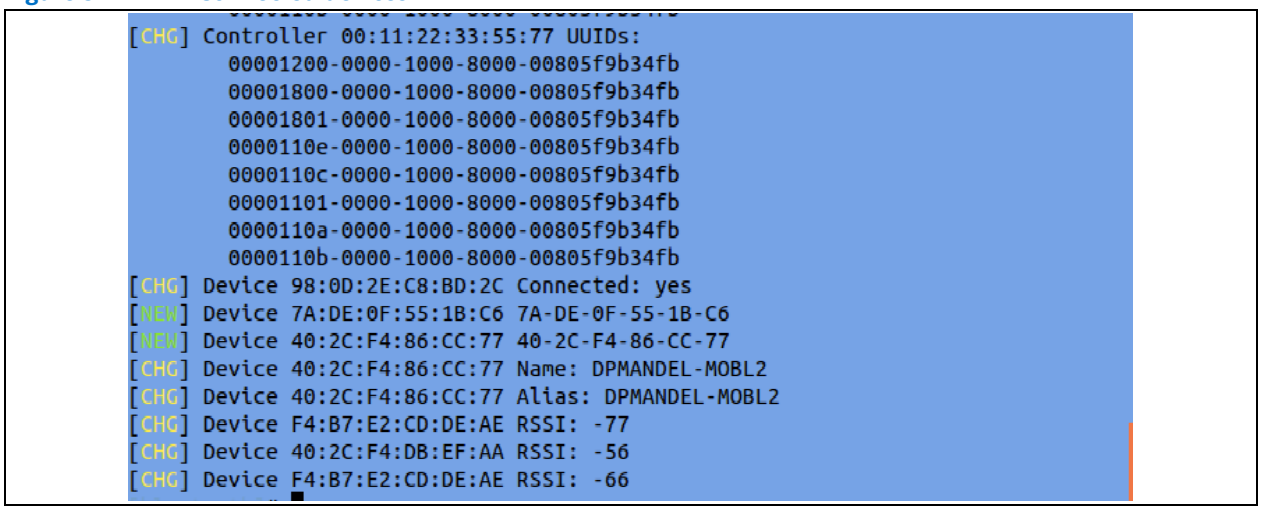
Figure 31 shows a series of screenshots from the launch of the application to scanning for Bluetooth* devices, connecting to the Intel® Edison Bluetooth* device (BlueZ 5.24), and the communication mode screen once the SPP connection has been established.

Figure 31 Android* screenshots



The Intel® Edison device is now connected to the Android* peer device.

Figure 32 Connected devices

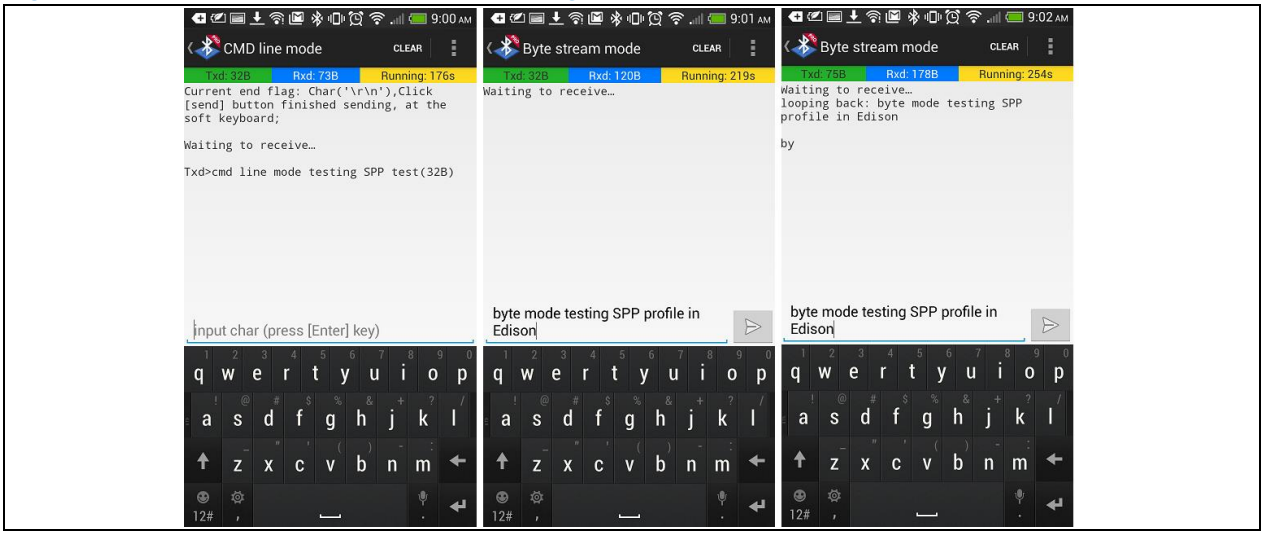


On the Intel® Edison *SPP-loopback.py* terminal, you can see that the SPP connection has been established with the Android* device:

```
root@edison:~# python ./SPP-loopback.py -C 22
NewConnection(/org/bluez/hci0/dev_98_0D_2E_C8_BD_2C, 10)
```

Once the Intel® Edison and Android* devices are connected with SPP, the devices can exchange profile data. Figure 33 shows a series of user-entered text in the device.

Figure 33 Sequence of screenshots showing the user inputs the text SPP application



And on the Intel® Edison side, you can see the data received and retransmitted back:

```
root@edison:~# python ./SPP-loopback.py -C 22
NewConnection(/org/bluez/hci0/dev_98_0D_2E_C8_BD_2C, 10)
received: cmd line mode testing SPP test

received: byte mode testing SPP profile in Edison
```

This completes testing SPP verification over DBUS APIs.

6.7.2 SPP verification using the RFCOMM tool

The Intel® Edison device must listen for incoming connections. You can do this with the RFCOMM tool, which is used to set up, maintain, and inspect the RFCOMM configuration of the Bluetooth* subsystem in the Linux* kernel.

After you successfully pair an Intel® Edison device with both a Linux* PC and an Android* device, you should also be able to pair with other Bluetooth*-enabled devices.

6.7.2.1 Intel® Edison configuration

Use the RFCOMM tool to set up, maintain, and inspect configuration of the Bluetooth* subsystem in the Linux* kernel:

1. Add the RFCOMM channel SDP entry:

```
root@edison:/usr/lib/bluez/test# ./test-profile -u 1101 -n edisonSpp -s
-P 3 -C 22 serial22
```

2. Start RFCOMM to listen to the incoming connection from a peer device:

```
root@edison:~# rfcomm listen 0 22
Waiting for connection on channel 22
```

...where 0 is the */dev/rfcommX* device that will be created, and 22 is the RFCOMM channel.

Note: Because Android* apps connect to RFCOMM channel 1, we need to pass channel 1 instead of 22 as an inline parameter for *test-profile*.

3. Using the *bluetoothctl scan on* command, discover a peer device, such as a Linux* PC or Android * phone:

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[NEW] Device D0:5F:B8:2A:0C:B9 Moto 360 0CB9
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device FC:F8:AE:1E:ED:98 XSDONGX-MOBL2
[bluetooth]#
```

4. Pair the Intel® Edison device with the discovered peer device:

```
[bluetooth]# pair 48:51:B7:15:D1:63
Attempting to pair with 48:51:B7:15:D1:63
[CHG] Device 48:51:B7:15:D1:63 Connected: yes
[CHG] Device 48:51:B7:15:D1:63 UUIDs:
        0000110a-0000-1000-8000-00805f9b34fb
        0000110b-0000-1000-8000-00805f9b34fb
        0000110c-0000-1000-8000-00805f9b34fb
        0000110e-0000-1000-8000-00805f9b34fb
        00001112-0000-1000-8000-00805f9b34fb
        0000111e-0000-1000-8000-00805f9b34fb
        0000111f-0000-1000-8000-00805f9b34fb
        0000112d-0000-1000-8000-00805f9b34fb
[CHG] Device 48:51:B7:15:D1:63 Paired: yes
Pairing successful
```

6.7.2.2 Android* devices

To test SPP using RFCOMM between an Android* device and an Intel® Edison device, you will need to download and install an SPP app (such as *Bluetooth SPP Pro* or *BlueTerm*/*BlueTerm+*) into an Android* device and pair the Android* device with the Intel® Edison device, as described in the Linux* setup. After you have successfully paired the devices, do the following:

1. As explained in the Intel® Edison configuration section, start RFCOMM on the Intel® Edison device to listen to the incoming connection from peer devices.

   Note: If you are using BlueTerm/BlueTerm+, use channel 1 for Android* devices instead of channel 22.

2. Launch the BlueTerm/BlueTerm+/Bluetooth SPP Pro app on your Android* device and, in the App menu, tap on *Connect devices*. Select the Intel® Edison Bluetooth* device (BlueZ 5.24) and select *Connected*. Once they are connected, you should be able to see the status listed as “connected” on the right side of the Android* screen. You can see the same status on the Intel® Edison device.

   On an Intel® Edison device:

```
root@edison:~# rfcomm listen 0 1
Waiting for connection on channel 1
Connection from D0:C1:B1:BD:17:97 to /dev/rfcomm0
Press CTRL-C for hangup
```

3. Text that you enter in the Android* app (Figure 34) will transmit to and display on the Intel® Edison device via SPP.

Figure 34 BlueTerm app sending text via SPP

4. Use the `cat /dev/rfcommX` command to see the text transmitted from the Android* device:

```
root@edison:~# cat /dev/rfcomm0
hi

vejbshs
  this is spp testing with android phone
```

Note: Because the *BlueTerm/BlueTerm+* apps use the RFCOMM channel exposed by the SDP entry and they don't work with Android* 4.1/4.2+ devices, we tested this process on Samsung* S and Nexus* 4 devices. Also note that because BlueTerm does not take care of the RFCOMM channel exposed in the SDP entry, we recommend using RFCOMM channel 1.

6.7.2.3 Linux* PC

To test SPP between a Linux* PC and an Intel® Edison device, discover the Intel® Edison RFCOMM channel exposed, then do the following:

1. Connect to both the Intel® Edison device and the Linux* PC using the RFCOMM tool that is also available on the Linux* PC. (It comes with the BlueZ package.)

   On a Linux* PC:

```
ram@ram-desktop:~$ sudo rfcomm connect 0 00:11:22:33:55:77 22
Connected /dev/rfcomm0 to 00:11:22:33:55:77 on channel 22
Press CTRL-C for hangup
```

   On an Intel® Edison device:

```
root@edison:~# rfcomm listen 0 22
Waiting for connection on channel 22
Connection from 48:51:B7:15:D1:63 to /dev/rfcomm0
Press CTRL-C for hangup
```

2. Once the connection is established, a */dev/rfcommX* device node is created on both devices. If you see a "NewConnection" description (in the shell where *test-profile* is running) that looks like this:

```
NewConnection(/org/bluz/hci0/dev_48_51_B7_15_D1_63, 10)
```

...it means that the script and not the RFCOMM tool is handling the file description. In this case, stop the *test-profile* script and run the previous steps on both the Intel® Edison device and the Linux* PC to establish a connection.

3. On the Linux* PC, verify */dev/rfcommX* with `sudo cat /dev/rfcommX` on the command line or launch Minicom and enter `cat /dev/rfcommX`. On the Intel® Edison device, `echo '<Text>' > /dev/rfcommX`.

   On the Linux* PC

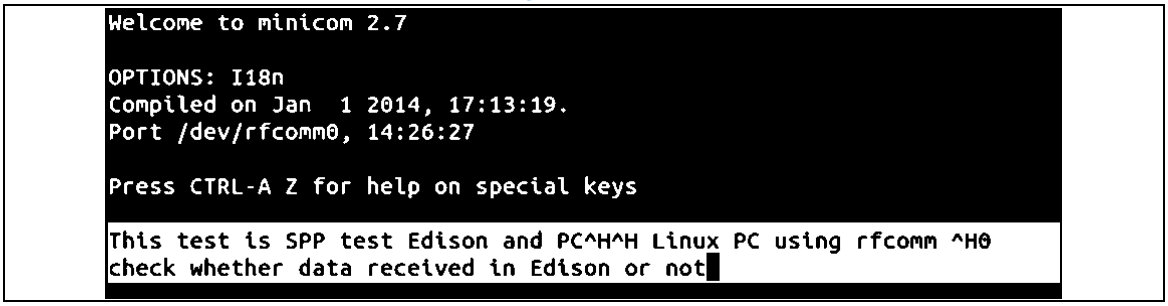
```
sudo minicom -D /dev/rfcomm0
```

   On the Intel® Edison device:

```
root@edison:~# cat /dev/rfcomm0
```

4. On the Linux* PC, enter text in the minicom window (Figure 35).

Figure 35 Minicom window on Linux* PC sending text



5. On the Intel® Edison device, verify that the text displays in the cat shell window (Figure 36).

Figure 36 Mirrored text in Intel® Edison device's cat shell window

6. On the Intel® Edison device, run the *echo* command:

```
root@edison:~# echo "This is an SPP test from Edison." > /dev/rfcomm0
```

7. On the Linux* PC, use the *cat* command to view the text string:

```
ram@ram-desktop:~$ sudo cat /dev/rfcomm0
[sudo] password for ram:
```

6.9 HID over GATT profile (HOGP)

The HID over GATT profile (HOGP) defines how a Bluetooth* Low Energy (BLE) device can support HID services over the BLE protocol stack, which is itself using the generic attribute profile (GAP). BlueZ supports HOGP as host. Regardless of what role (boot/report host or HID device) the device plays, the mandatory services *HID service*, *Device Information service*, and *Battery service* are always available.

For more information on HOGP, visit *https://developer.bluetooth.org/TechnologyOverview/Pages/HOGP.aspx*.

We used a Logitech* mouse for this use case:

1. Enable Bluetooth* and set default agents with the *bluetoothctl* utility:

```
root@edison:~# rfkill unblock bluetooth
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device D0:5F:B8:2A:0C:B9 Moto 360 0CB9
[bluetooth]# agent DisplayOnly
Agent registered
[bluetooth]# default-agent
Default agent request successful
```

2. Turn on the Bluetooth* mouse and enable it into pairing mode before scanning in the Intel® Edison device.
3. Run the *scan on* command on the Intel® Edison device to discover the MAC address of the mouse, pair the Intel® Edison device with the mouse, and connect the Intel® Edison device to the mouse:

```
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 00:1F:20:42:27:12 Bluetooth Laser Travel Mouse
[NEW] Device 5C:51:4F:9E:49:AD DSGAO-MOBL1
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device F0:79:59:03:69:FC Nexus Player
[CHG] Device 40:2C:F4:DB:EF:AA RSSI: -53
[bluetooth]# pair 00:1F:20:42:27:12
Attempting to pair with 00:1F:20:42:27:12
[CHG] Device 00:1F:20:42:27:12 Connected: yes
[CHG] Device 00:1F:20:42:27:12 Modalias: usb:v046DpB008d0318
[CHG] Device 00:1F:20:42:27:12 UUIDs:
        00001124-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
[CHG] Device 00:1F:20:42:27:12 Paired: yes
Pairing successful
[CHG] Device 00:1F:20:42:27:12 Connected: yes
[CHG] Device 40:2C:F4:DB:EF:AA RSSI: -76
[bluetooth]# connect 00:1F:20:42:27:12
Attempting to connect with 00:1F:20:42:27:12
[CHG] Device 00:1F:20:42:27:12 Connected: yes
Connection  successful
```

4. Verify the services supported by the Bluetooth* mouse with the *info* command:

```
[bluetooth]# info 00:1F:20:42:27:12
Device 00:1F:20:42:27:12
        Name: Bluetooth Laser Travel Mouse
        Alias: Bluetooth Laser Travel Mouse
        Class: 0x002580
        Icon: input-mouse
        Paired: yes
        Trusted: no
        Blocked: no
        Connected: yes
        LegacyPairing: yes
        UUID: Human Interface Device    (00001124-0000-1000-8000-00805f9b34fb)
        UUID: PnP Information
        Modalias: usb:v046DpB008d0318   (00001200-0000-1000-8000-00805f9b34fb)
```

5. Verify that the mouse has connected successfully to the Intel® Edison device. When you make a connection, the system creates a */dev/input/eventX* file. Viewing the file with the *cat* command yields unreadable code, but you can decode these incoming events with the free desktop utility *evtest*, available at this website: *http://cgit.freedesktop.org/~whot/evtest*.

   Note: Compile the *evtest* code for Intel® Edison, copy the binary to the Intel® Edison device, and then launch the app.

6. As you move the mouse or press the mouse buttons, you should see the *evtest* app decode mouse events into human readable events (Figure 37).

Figure 37 Example event test results from Bluetooth mouse

```
root@edison:~# ./evtest /dev/input/event2
Input driver version is 1.0.1
Input device ID: bus 0x5 vendor 0x46d product 0xb008 version 0x318
Input device name: "Bluetooth Laser Travel Mouse"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
    Event code 272 (LeftBtn)
    Event code 273 (RightBtn)
    Event code 274 (MiddleBtn)
    Event code 275 (SideBtn)
    Event code 276 (ExtraBtn)
    Event code 277 (ForwardBtn)
    Event code 278 (BackBtn)
    Event code 279 (TaskBtn)
  Event type 2 (Relative)
    Event code 0 (X)
    Event code 1 (Y)
    Event code 6 (HWheel)
    Event code 8 (Wheel)
  Event type 4 (Misc)
    Event code 4 (ScanCode)
Testing ... (interrupt to exit)
Event: time 1418146812.719418, type 2 (Relative), code 0 (X), value 22
Event: time 1418146812.719418, -------------- Report Sync ------------
Event: time 1418146813.975514, type 2 (Relative), code 1 (Y), value 1
Event: time 1418146813.975514, -------------- Report Sync ------------
Event: time 1418146813.987044, type 2 (Relative), code 1 (Y), value 2
Event: time 1418146813.987044, -------------- Report Sync ------------
Event: time 1418146813.998241, type 2 (Relative), code 0 (X), value -1
Event: time 1418146813.998241, type 2 (Relative), code 1 (Y), value 1
Event: time 1418146813.998241, -------------- Report Sync ------------
Event: time 1418146814.009678, type 2 (Relative), code 1 (Y), value 1
Event: time 1418146814.009678, -------------- Report Sync ------------
```

6.10 Heart rate profile (HRP)

An Intel® Edison device may act as a heart rate collector, receiving heart rate information from a wearable heart rate sensor like the Mio* ALPHA, Polar* H7, or Intel Basis Peak. The BlueZ test package contains Python scripts to test the BLE profiles. One of these, *test-heartrate*, decodes notifications sent by heart rate sensors. We tested this using a Polar H7 as a peer device.

Copy the test scripts into the Intel® Edison device using *scp* and do the following:

1. Unblock *bluetoothctl*:

```
root@edison:/usr/lib/bluez/test# rfkill unblock bluetooth
```

2. Launch *bluetoothctl* and scan for the heart rate sensor device:

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 00:18:6B:4E:A4:B8 LG HBS730
[NEW] Device D0:5F:B8:2A:0C:B9 Moto 360 0CB9
[bluetooth]# agent DisplayYesNo
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 00:22:D0:3B:2F:2A Polar H7 3B2F2A1C
[NEW] Device D9:A8:B4:0F:3D:A7 D9-A8-B4-0F-3D-A7
[CHG] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[bluetooth]#
```

3. Add the heart rate sensor (Polar* H7 heart rate monitor) as a trusted device and pair it with the Intel® Edison device:

```
[bluetooth]# trust 00:22:D0:3B:2F:2A
[CHG] Device 00:22:D0:3B:2F:2A Trusted: yes
Changing 00:22:D0:3B:2F:2A trust succeeded
[CHG] Device 48:51:B7:15:D1:63 RSSI: -50
[bluetooth]# pair 00:22:D0:3B:2F:2A
Attempting to pair with 00:22:D0:3B:2F:2A
[CHG] Device 00:22:D0:3B:2F:2A Connected: yes
[CHG] Device 00:22:D0:3B:2F:2A UUIDs:
        00001800-0000-1000-8000-00805f9b34fb
        00001801-0000-1000-8000-00805f9b34fb
        0000180a-0000-1000-8000-00805f9b34fb
        0000180d-0000-1000-8000-00805f9b34fb
        0000180f-0000-1000-8000-00805f9b34fb
        6217ff4b-fb31-1140-ad5a-a45545d7ecf3
[CHG] Device 00:22:D0:3B:2F:2A Paired: yes
Pairing successful
[CHG] Device 00:22:D0:3B:2F:2A Appearance: 0x0341
[CHG] Device 50:4C:EF:64:8B:BB RSSI: -71
```

4. Connect to the heart rate sensor device:

```
[bluetooth]# connect 00:22:D0:3B:2F:2A
Attempting to connect to 00:22:D0:3B:2F:2A
Connection successful
```

5. Inspect the supported services on the peer device (heart rate monitor). You should see heart rate in the UUID list:

```
[bluetooth]# info 00:22:D0:3B:2F:2A
Device 00:22:D0:3B:2F:2A
        Name: Polar H7 3B2F2A1C
        Alias: Polar H7 3B2F2A1C
        Appearance: 0x0341
        Paired: yes
        Trusted: yes
        Blocked: no
        Connected: yes
        LegacyPairing: no
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Device Information        (0000180a-0000-1000-8000-00805f9b34fb)
        UUID: Heart Rate                (0000180d-0000-1000-8000-00805f9b34fb)
        UUID: Battery Service           (0000180f-0000-1000-8000-00805f9b34fb)
        UUID: Vendor Specific           (6217ff4b-fb31-1140-ad5a-a45545d7ecf3)
```

6. Run the *test-heartrate* script on the Intel® Edison device. This script registers notification, reads the sensor location, then decodes the received notification packets transmitted by the heart rate monitor. The example below shows the data retrieved from the notifications that the heart rate monitor published.

Figure 38 Example heart rate monitor data

```
root@edison:/usr/lib/bluez/test# ./test-heartrate -b 00:22:D0:3B:2F:2A
Sensor location: chest
Measurement received from /org/bluez/hci0/dev_00_22_D0_3B_2F_2A
Value:  86
Contact:  1
Measurement received from /org/bluez/hci0/dev_00_22_D0_3B_2F_2A
Value:  87
Contact:  1
Interval:  698
Measurement received from /org/bluez/hci0/dev_00_22_D0_3B_2F_2A
Value:  86
Contact:  1
Interval:  715
Measurement received from /org/bluez/hci0/dev_00_22_D0_3B_2F_2A
Value:  86
Contact:  1
Interval:  717
Interval:  731
Measurement received from /org/bluez/hci0/dev_00_22_D0_3B_2F_2A
Value:  86
Contact:  1
Interval:  708
```

6.11 Proximity profile (PXP)

Commonly used in security-related appliances, the proximity profile (PXP) defines behavior when a device moves away from a peer and results in a dropped connection or a path loss. This produces an alert, which notifies the user that the device is moving away.

PXP supports the following roles:

- Monitor, which acts as a GATT client that makes use of services on the peer device. The Proximity Monitor may alert when the path loss exceeds the threshold.
- Reporter (with the following services):
  - Mandatory services: *Link Loss Service*. This is instantiated as a primary service.
  - Optional services: *Immediate Alert Service* and *Tx Power Service.*

Note: PXP devices can support both of the optional services or neither; they cannot support only one.

DBUS APIs for PXP are documented at *http://git.kernel.org/cgit/bluetooth/bluez.git/tree/doc/proximity-api.txt*, and you can find sources at *http://git.kernel.org/cgit/bluetooth/bluez.git/tree/profiles/proximity*. There is also a test script to test the monitor role by setting a link loss alert level on a peer device and allowing it to trigger an immediate alert. Visit *https://developer.bluetooth.org/TechnologyOverview/Pages/PXP.aspx* for details on PXP.

6.11.1 PXP services

PXP supports the following services:

- **Link Loss:** This service can be initiated only as a primary service, and only one instance may run on a device. The service will have only one alert status in the Link Loss service. The service has three alert levels (None, Mild, High), which are used to notify how the device alerts the user when the connection/link is lost. For example, the proximity monitor writes the alert characteristics into the proximity reporter, and the reporter will alert at this level when a link with the peer device is lost.
- **Immediate Alert Service:** This service is used to alert the user when there is a path loss. This service uses alert level characteristics and causes an alert whether a value other than “No Alert” is written to it.
- **Tx Power Service:** This service is also used to alert the user when there is a path loss, and only one instance may run on a device. This service enables the GATT client to retrieve the device’s current transmit power level when there is a connection.

Note: As mentioned in the official documentation (*http://www.bluez.org/proximity-link-loss-and-find-me*), currently only link loss is functional; path loss needs some tweaking to test.

6.11.2 PXP test

To perform the PXP test between an HTC-Fetch device and an Intel® Edison device, do the following:

1. Check whether Bluetooth* is active. If it is not, turn it on using the *rfkill* command.
2. Run the *bluetoothctl* utility to scan and pair with the HTC-Fetch device.

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device D0:5F:B8:2A:0C:B9 Moto 360 0CB9
[NEW] Device 00:1F:20:42:27:12 Bluetooth Laser Travel Mouse
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 20:CD:39:A5:3B:62 20-CD-39-A5-3B-62
[CHG] Device 20:CD:39:A5:3B:62 Name: HTC Fetch
[CHG] Device 20:CD:39:A5:3B:62 Alias: HTC Fetch
```

3. Connect to the HTC-Fetch device.

```
[bluetooth]# connect 20:CD:39:A5:3B:62
Attempting to connect to 20:CD:39:A5:3B:62
[CHG] Device 20:CD:39:A5:3B:62 Connected: yes
Connection successful
```

4. Use the *info* command to verify whether the peer-device (HTC-Fetch) has the desired services. As the highlights below show, the HTC Fetch device supports the mandatory (*Link Loss*) and optional (*Tx Power* and *Immediate Alert*) services. It also supports custom services (*Device Information* and *Battery Service*).

```
[bluetooth]# info 20:CD:39:A5:3B:62
Device 20:CD:39:A5:3B:62
        Name: HTC Fetch
        Alias: HTC Fetch
        Paired: no
        Trusted: no
        Blocked: no
        Connected: yes
        LegacyPairing: no
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Immediate Alert           (00001802-0000-1000-8000-00805f9b34fb)
        UUID: Link Loss                 (00001803-0000-1000-8000-00805f9b34fb)
        UUID: Tx Power                  (00001804-0000-1000-8000-00805f9b34fb)
        UUID: Device Information        (0000180a-0000-1000-8000-00805f9b34fb)
        UUID: Battery Service           (0000180f-0000-1000-8000-00805f9b34fb)
        UUID: Unknown                   (0000ffe0-0000-1000-8000-00805f9b34fb)
        UUID: Vendor Specific           (f000ffc0-)
        UUID: Vendor Specific           (f000ffc0-)
        Modalias: bluetooth:v000Dp0000d0110
[NEW] Device C8:F7:33:8B:48:08 C8-F7-33-8B-48-08
[bluetooth]#
```

6.11.3 Proximity monitor

The python test script *test-proximity* monitors the proximity profile. This script lets you use arguments, such as the level/value of an alert (None, Mild, High) and whether the alert is *ImmediateAlertLevel* or *LinkLossAlertLevel*, to configure the alert levels of an HTC-Fetch device.

To use the *test-proximity* script, do the following:

1. Configure the HTC-Fetch device with an *ImmediateAlertLevel* set to *mild*. The script will write the immediate alert characteristics (values) into the peer device, and you will see the HTC-Fetch device start alerting. After some time, it will set itself to *none* and become idle.

```
root@edison:/usr/lib/bluez/test# ./test-proximity –b 20:CD:39:A5:3B:62
ImmediateAlertLevel mild
Proximity SetProperty('ImmediateAlertLevel', 'mild')
Property ImmediateAlertLevel changed: mild
Property ImmediateAlertLevel changed: none
```

2. Enter the same command with parameters *LinkLossAlertLevel* and a value of *high*:

```
root@edison:/usr/lib/bluez/test# ./test-proximity -b 20:CD:39:A5:3B:62
LinkLossAlertLevel high
Proximity SetProperty('LinkLossAlertLevel', 'high')
Property LinkLossAlertLevel changed: high
Property LinkLossAlertLevel changed: high
```

3. As soon as you notice the command is effective, move the HTC-Fetch device some distance away, until the link breaks—typically 50 ft. (15 m) or more. The HTC-Fetch will emit an alert sound until you stop it.

Note: This test script works for the proximity monitor role only.

6.11.4 Proximity reporter

BlueZ registers a list of GATT servers—among them Link Loss, Immediate Alert, and Tx Power—that support the proximity profile in reporter mode.

To manually set one Intel® Edison device in advertising mode and use a second Intel® Edison device as the connecting device, do the following:

1. On the first Intel® Edison device (the proximity reporter), start LE advertising data, set advertising data, and eventually disable scan in BT classic.

```
root@edison:~# hciconfig hci0 noscan
root@edison:~# hciconfig hci0 leadv
root@edison:~#
root@edison:~# hciconfig -i hci0 cmd 0x08 0x0008 16 02 01 06 07 02 03
18 02 18 04 18 0a 09 45 64 69 73 6f 6e 2d 4c 45
< HCI Command: ogf 0x08, ocf 0x0008, plen 23
  16 02 01 06 07 02 03 18 02 18 04 18 0A 09 45 64 69 73 6F 6E 2D 4C 45
> HCI Event: 0x0e plen 4
  01 08 20 00
root@edison:~#
```

2. On the second Intel® Edison device (the proximity monitor), do a normal scan on and connect:

```
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 98:4F:EE:02:E8:4B Edison-LE
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[CHG] Device 00:1F:20:42:27:12 Connected: yes
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[CHG] Device 98:4F:EE:02:E8:4B RSSI: -60
[CHG] Device 00:1F:20:42:27:12 Connected: no
```

3. Connect the second Intel® Edison device (proximity monitor) with the first device (proximity reporter).

```
[bluetooth]# connect 98:4F:EE:02:E8:4B
Attempting to connect to 98:4F:EE:02:E8:4B
[CHG] Device 98:4F:EE:02:E8:4B Connected: yes
Connection successful
[CHG] Device 98:4F:EE:02:E8:4B UUIDs:
        00001800-0000-1000-8000-00805f9b34fb
        00001801-0000-1000-8000-00805f9b34fb
        00001802-0000-1000-8000-00805f9b34fb
        00001803-0000-1000-8000-00805f9b34fb
        00001804-0000-1000-8000-00805f9b34fb
```

```
        00001805-0000-1000-8000-00805f9b34fb
        00001806-0000-1000-8000-00805f9b34fb
        0000180e-0000-1000-8000-00805f9b34fb
        00001811-0000-1000-8000-00805f9b34fb
[CHG] Device 98:4F:EE:02:E8:4B Appearance: 0x0110
[bluetooth]#
```

4. Use the *info <BT_MAC_address>* command to verify that the first BLE device supports the services from the second device.

```
[bluetooth]# info 98:4F:EE:02:E8:4B
Device 98:4F:EE:02:E8:4B
        Name: Edison-LE
        Alias: Edison-LE
        Appearance: 0x0110
        Paired: no
        Trusted: no
        Blocked: no
        Connected: yes
        LegacyPairing: no
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Immediate Alert           (00001802-0000-1000-8000-00805f9b34fb)
        UUID: Link Loss                 (00001803-0000-1000-8000-00805f9b34fb)
        UUID: Tx Power                  (00001804-0000-1000-8000-00805f9b34fb)
        UUID: Current Time Service      (00001805-0000-1000-8000-00805f9b34fb)
        UUID: Reference Time Update S.. (00001806-0000-1000-8000-00805f9b34fb)
        UUID: Phone Alert Status Serv.. (0000180e-0000-1000-8000-00805f9b34fb)
        UUID: Alert Notification Serv.. (00001811-0000-1000-8000-00805f9b34fb)
```

5. The two Intel® Edison devices are connected. On the proximity monitor (second device), we can start the *test-proximity* script (acting as monitor) that sets immediate alerts and link loss alerts on the proximity reporter board. Notice the status of the second device as soon as the link is lost:

```
root@edison:/usr/lib/bluez/test# ./test-proximity –b 98:4F:EE:02:E8:4B
ImmediateAlertLevel high
Proximity SetProperty('ImmediateAlertLevel', 'high')
Property ImmediateAlertLevel changed: high
Property ImmediateAlertLevel changed: none


^CTraceback (most recent call last):
  File "./test-proximity", line 70, in <module>
    mainloop.run()
KeyboardInterrupt
root@edison:/usr/lib/bluez/test# ./test-proximity –b 98:4F:EE:02:E8:4B
LinkLossAlertLevel high
Proximity SetProperty('LinkLossAlertLevel', 'high')
```

Note: There is no script to validate the proximity reporter role in BlueZ.

6.12 Time profile (TIP)

The time profile (TIP) controls the functionalities related to time and allows devices to retrieve various information parameters, such as date, time, time zone, and daylight saving time (DST), as exposed by peer devices. Using this profile, a device can request the time from a peer device using the time update service. BlueZ can act as a Time Server because it implements mandatory *Current Time Service* and optional *Reference Time Update Service*. (It does not implement other optional services, such as *Next DST Change Service*.)

Note: Typically, a Time Server acts as a central role in a connection provided to a peripheral Time Service. (In most cases, peripherals won't have time information available.)

For testing purposes, we used the Android* *Nordic* app to retrieve the information. To do so, we had to assign the Intel® Edison device the peripheral role, and the Android* device the central role. (We cannot make the phone behave as a peripheral device.) We tested with this feature with an Android* Moto G device and an Intel® Edison device connected as peers. To test TIP functionality on an Intel® Edison device, do the following:

1. Download and install the *https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp* app into the Android* device.
2. Set up the Intel® Edison device as a peripheral device and start the Intel® Edison device in advertise mode by executing these commands:

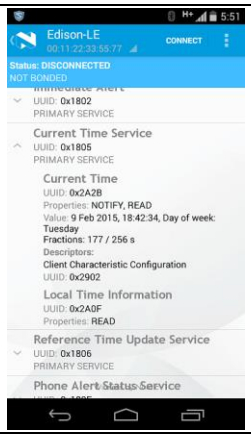
```
root@edison:~# hciconfig hci0 noscan
root@edison:~# hciconfig hci0 leadv
root@edison:~#
```

3. Set the advertising data by publishing the data in the peripheral role.

```
root@edison:~# hciconfig -i hci0 cmd 0x08 0x0008 16 02 01 06 07 02 03
18 02 18 04 18 0a 09 45 64 69 73 6f 6e 2d 4c 45
< HCI Command: ogf 0x08, ocf 0x0008, plen 23
  16 02 01 06 07 FE FD 18 02 18 04 18 0A 09 45 64 69 73 6F 6E 2D 4C 45
> HCI Event: 0x0e plen 4
  01 08 20 00
root@edison:~#
```

4. The Intel® Edison device will publish the information above. Launch the Nordic app and scan for Bluetooth* devices. When you identify the Intel® Edison device, connect to it. It should display the service supported by the Intel® Edison device. Tap on the current time service where BlueZ mandatory services (current time, local time information) appear (Figure 39).

Figure 39 Current time service on Android* device



For more information on TIP, visit: *https://developer.bluetooth.org/TechnologyOverview/Pages/TIP.aspx.*

6.13 File transfer protocol (FTP) profile

FTP (File Transfer Protocol) allows two or more devices in a network to share folders/files. The devices can be Windows* PCs, laptops, mobile devices, Intel® Edison boards, Mac* or Linux* computers, or devices like Android* phones. Once an FTP client identifies and connects with a valid FTP server, it can "put" files/folders into the location or "get" files/folders from it. Any FTP device can act as client or server.

- **FTP client:** Initiates put/get of objects (files/folders) to and from the server.
- **FTP server:** Provides an object exchange server and folder browsing (using the OBEX Folder Listing format).

To complete profile registration, do the following:

1. Enable Bluetooth*.
2. Start the *obex* service and verify that it has stated correctly:

```
root@edison:~# systemctl start obex
root@edison:~# systemctl status obex
● obex.service - Bluetooth OBEX service
   Loaded: loaded (/lib/systemd/system/obex.service; disabled)
   Active: active (running) since Fri 2015-01-02 19:08:27 UTC, 2s ago
MAIN PID: 817 (obexd)
   CGroup: /system.slice/obex.service
           └ 817 /usr/lib/bluez5/bluetooth/obexd

Jan 02 19:08:27 edison obexd[817]: OBEX daemon 5.24
Jan 02 19:08:27 edison systemd[1]: Started Bluetooth OBEX service.
root@edison:~#
```

3. In the *bluetoothctl* utility console, check whether the obex profiles are correctly registered BlueZ 5.24.

Figure 40 Checking obex profiles

```
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[bluetooth]# show
Controller 00:11:22:33:55:77
        Name: BlueZ 5.24
        Alias: BlueZ 5.24
        Class: 0x1c0110
        Powered: yes
        Discoverable: no
        Pairable: yes
        UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
        UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
        UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
        UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
        UUID: Message Notification Se.. (00001133-0000-1000-8000-00805f9b34fb)
        UUID: Serial Port               (00001101-0000-1000-8000-00805f9b34fb)
        UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
        UUID: IrMC Sync                 (00001104-0000-1000-8000-00805f9b34fb)
        UUID: OBEX File Transfer        (00001106-0000-1000-8000-00805f9b34fb)
        UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
        UUID: Vendor specific           (00005005-0000-1000-8000-0002ee000001)
        Modalias: usb:v1D6Bp0246d0518
        Discovering: no
[bluetooth]# 
```

An Intel® Edison device can act as FTP client and server. The above profile registration is common for both FTP server as well as FTP client use case.

6.13.1 FTP server

When the *obexd* daemon starts, it will by default support FTP server functionality. So Intel® Edison will become an FTP server, and its peer device can be used as FTP client. You can use either a Linux* PC or Android* device as the FTP client device.

6.13.1.1 Android*

Because the Android* phone doesn't support FTP, you will probably need to download and install an app to your Android* device from the playstore, such as the freeware app used in this example, *Bluetooth File Transfer*.

To pair an Intel® Edison device with Android* peer devices, do the following:

1. Set discoverable on, scan on, and agent registration.

Figure 41 Pairing Intel® Edison with Android* peer devices

```
root@edison:~# bluetoothctl
[NEW] Controller 00:11:22:33:55:77 BlueZ 5.24 [default]
[NEW] Device 48:51:B7:15:D1:63 ubuntu-0
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]# agent DisplayYesNo
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:11:22:33:55:77 Discovering: yes
[NEW] Device 5F:35:3A:59:B8:F7 5F-35-3A-59-B8-F7
[NEW] Device 5D:C6:FB:1F:38:3C 5D-C6-FB-1F-38-3C
[NEW] Device ED:9C:0E:D1:6A:2B MTKBTDEVICE
[NEW] Device 40:2C:F4:DB:EF:AA NAGESWAX-MOBL1
[NEW] Device 20:C9:D0:79:59:3A jlabreo-mac04
[NEW] Device A4:17:31:B2:DA:E9 TWFRANK-MOBL
[NEW] Device 98:0D:2E:C8:BD:2C HTC One nag
[NEW] Device 7C:E9:D3:DB:0B:F6 MARGARET-MOBL3
[NEW] Device C8:F7:33:2D:2B:F1 DLARMSTR-MOBL4
[NEW] Device 60:D8:19:BD:62:57 ELIZAARI-MOBL
[CHG] Device 98:0D:2E:C8:BD:2C Connected: yes
[NEW] Device FC:F8:AE:1E:ED:98 XSDONGX-MOBL2
[NEW] Device 40:2C:F4:86:72:54 TNGU25X-MOBL2
[CHG] Device 48:51:B7:15:D1:63 RSSI: -34
Request authorization
```

2. Pair with a peer Android* device. If you are pairing from an Android* phone, you may pair Intel® Edison from *Settings* > *Bluetooth*. You can also set a peer device as trusted to avoid confirmation when connecting to the FTP service.

```
[CHG] Device 98:0D:2E:C8:BD:2C Paired: yes
Authorize service
o): yes Authorize service 0000110d-0000-1000-8000-00805f9b34fb (yes/no)
Request confirmation
[NEW] Device D5:B3:ED:7E:A5:83 D5-B3-ED-7E-A5-83
[agent] Confirm passkey 144410 (yes/no): yes
[NEW] Device E8:BE:82:BE:75:19 E8-BE-82-BE-75-19
[CHG] Device 98:0D:2E:C8:BD:2C UUIDs:
        00001105-0000-1000-8000-00805f9b34fb
        0000110a-0000-1000-8000-00805f9b34fb
        0000110c-0000-1000-8000-00805f9b34fb
        0000110d-0000-1000-8000-00805f9b34fb
        0000110e-0000-1000-8000-00805f9b34fb
        00001112-0000-1000-8000-00805f9b34fb
[NEW] Device 80:86:B7:15:D1:66 BAT-MOBL
```

```
o): yes Authorize service 0000110e-0000-1000-8000-00805f9b34fb (yes/no
[CHG] Device 5D:C6:FB:1F:38:3C RSSI: -72
[CHG] Device EC:55:F9:F0:14:EA Name: YTORRES-MOBL3
[CHG] Device EC:55:F9:F0:14:EA Alias: YTORRES-MOBL3
[bluetooth]
```

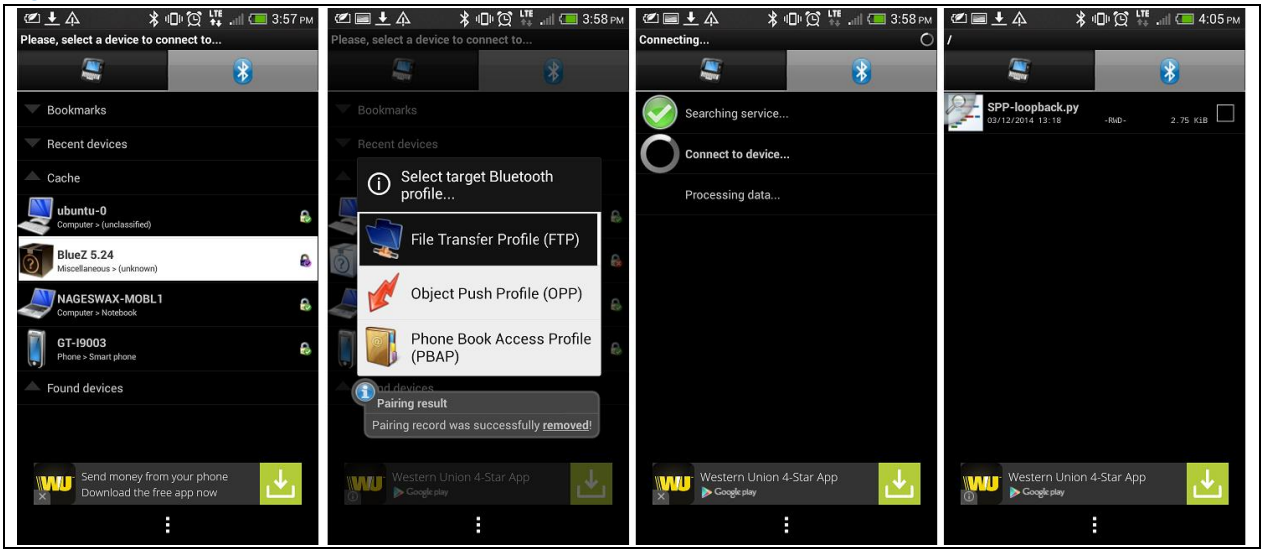
3. Verify that the Intel® Edison and Android* devices are paired:

```
        # paired-devices
Device 48:51:B7:15:D1:63 ubuntu-0
Device 98:0D:2E:C8:BD:2C HTC One nag
```

4. After you launch the FTP client app on your Android* phone, do the following:

   a. Connect to FTP on Intel® Edison by clicking on the Bluetooth* icon in the app.
   b. Select the Intel® Edison device in the listed peripherals.
   c. Select FTP option in the target Bluetooth* screen.

Figure 42 Android* FTP screenshots



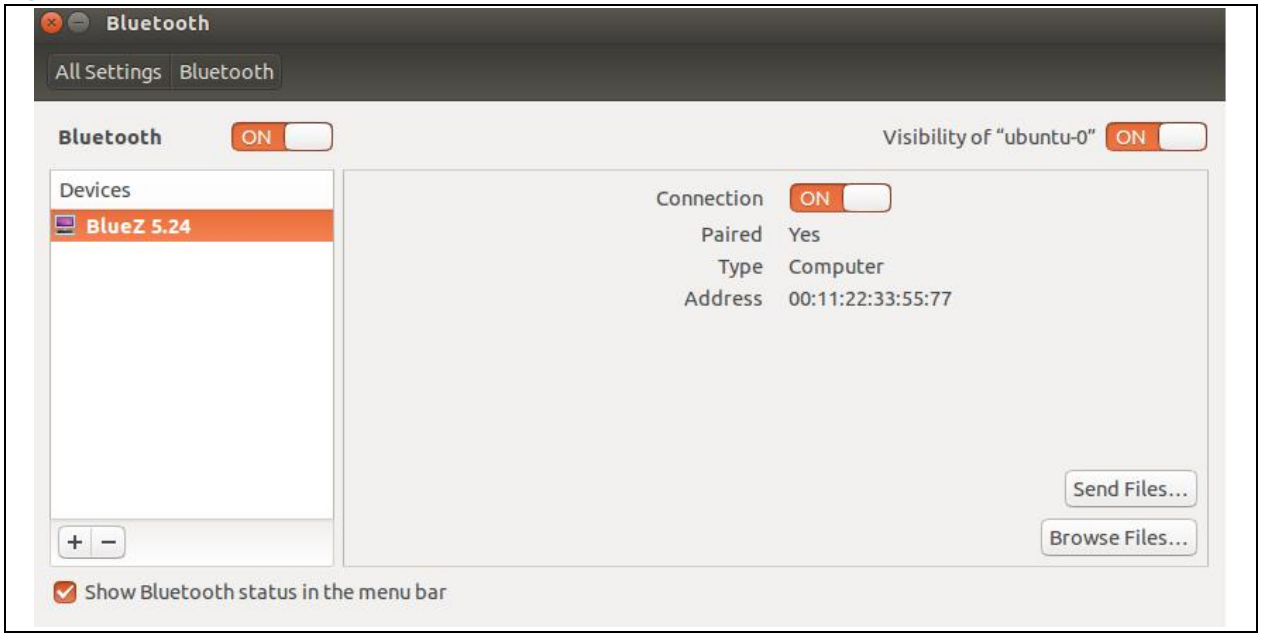
The screenshots in Figure 42 are for reference. This would establish the FTP connection to Intel® Edison device and you could see the files in the *obex* folder (by default, in *~/.cache/obexd* under the user's home folder).

From the app, you can download the files to your Android* device or push files from the local Android* device folder to the Intel® Edison device. Pull the file from the Intel® Edison device as mentioned above.

6.13.1.2 Linux* PC

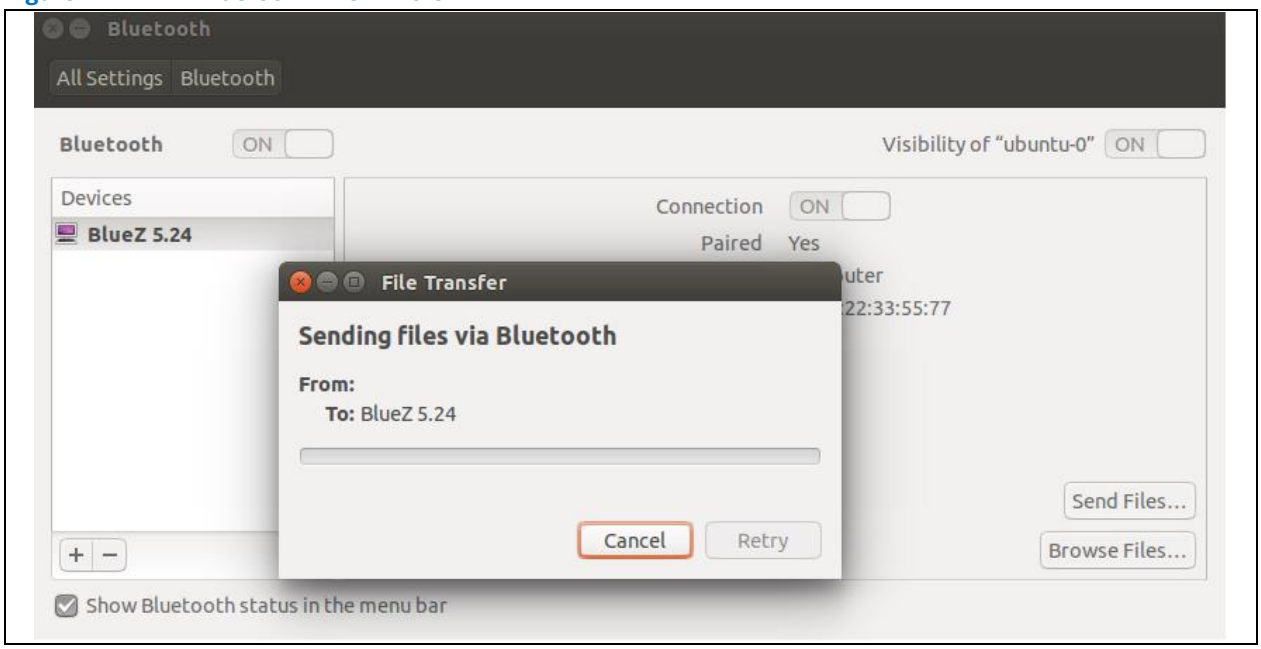
Pair the Intel® Edison device and Linux* PC as described above. As soon as you pair and connect the Intel® Edison device from a Linux* PC, you will see the *Send Files* and *Browse Files* buttons (Figure 43).

Figure 43 Send/browse files



You can send or browse files to/from the Intel® Edison device (Figure 44).

Figure 44 Bluetooth* file transfer



6.13.2 FTP client

BlueZ provides *obexctl*, a command line utility that you can use as an FTP client.

To connect as an FTP client, do the following:

1. Unblock Bluetooth* on the device:

```
root@edison:~# rfkill unblock Bluetooth
```

2. Add the *DBUS_SESSION_BUS_ADDRESS* variable to the environment path:

```
root@edison:~# export DBUS_SESSION_BUS_ADDRESS=unix:path=/var/run/dbus/
system_bus_socket
```

3. Start the *obexctl* utility:

```
root@edison:~# obexctl
[NEW] Client /org/bluez/obex
```

Once you start the command line utility, a previously paired device can be connected over FTP, and once it is connected, you can browse its file system, create or delete folders, delete or copy files, etc. (Figure 45).

Figure 45 Actions available after pairing

```
[98:0D:2E:C8:BD:2C]# connect 98:0D:2E:C8:BD:2C FTP
Attempting to connect to 98:0D:2E:C8:BD:2C
[NEW] Session /org/bluez/obex/client/session1 [default]
[NEW] FileTransfer /org/bluez/obex/client/session1
Connection successful
[98:0D:2E:C8:BD:2C]# ls
Attempting to ListFolder
[NEW] Transfer /org/bluez/obex/client/session1/transfer0
[CHG] Transfer /org/bluez/obex/client/session1/transfer0 Size: 2860
[CHG] Transfer /org/bluez/obex/client/session1/transfer0 Status: complete
        Type: folder
        Name: Android
        Size: 4096
        Modified: 19700104T144208
        User-perm: RWD
        Type: folder
        Name: Music
        Size: 4096
        Modified: 20141020T151722
```

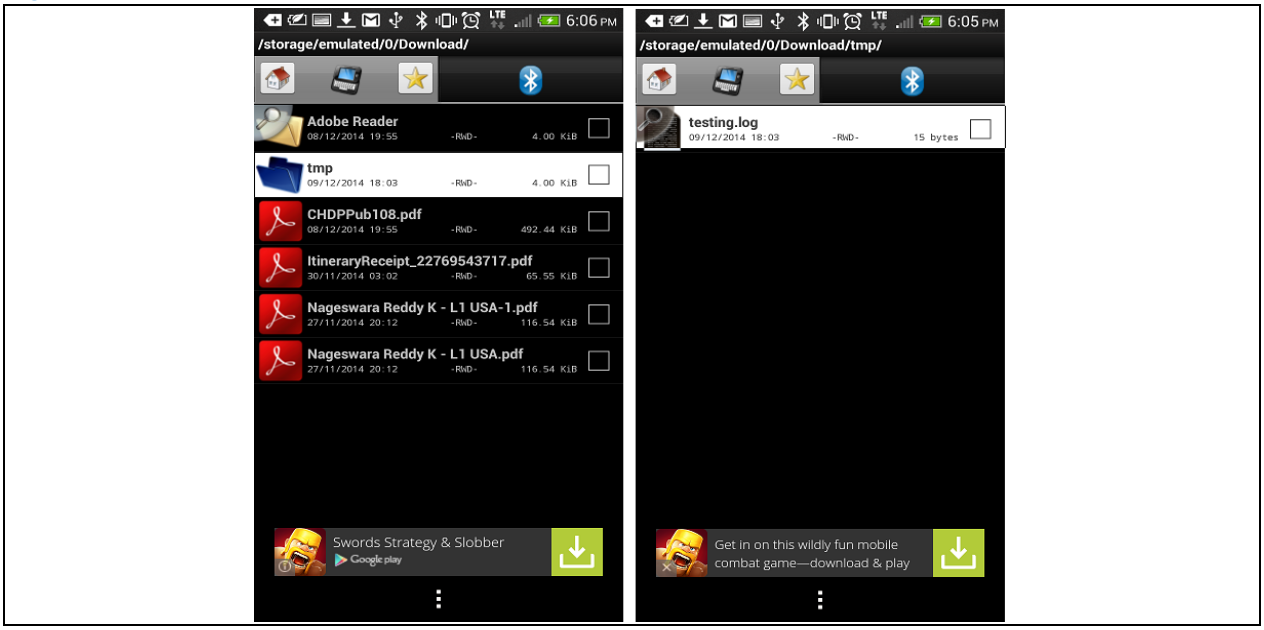
Figure 46 shows actions done interacting with an Android* phone with the FTP app.

Figure 46 Actions available

```
[DEL] Transfer /org/bluez/obex/client/session1/transfer1
[98:0D:2E:C8:BD:2C]# mkdir tmp
Attempting to CreateFolder
CreateFolder successful
[98:0D:2E:C8:BD:2C]# cp :/home/root/testing.log testing.log
Attempting to PutFile
[NEW] Transfer /org/bluez/obex/client/session1/transfer2
Transfer /org/bluez/obex/client/session1/transfer2
        Status: queued
        Name: testing.log
        Size: 15
        Filename: /home/root/testing.log
        Session: /org/bluez/obex/client/session1
[CHG] Transfer /org/bluez/obex/client/session1/transfer2 Status: complete
[DEL] Transfer /org/bluez/obex/client/session1/transfer2
[DEL] Session /org/bluez/obex/client/session1 [default]
[DEL] FileTransfer /org/bluez/obex/client/session1
[obex]# 
```

On the Intel® Edison device: Execute the FTP operations to change the remote folder into *Download*, then create a folder named *tmp* inside the *Download* folder, and copy a file "testing.log" in there from the Intel® Edison device. (Figure 47 shows an FTP *put* function.)

Figure 47 Android* device screenshots



Here are some tips that may be helpful when using FTP:

- When *obexctl* starts, a blue `[obex]` prompt displays with `[NEW] Client` at the start of the line.
- When triggering a connection, specify FTP after the peer BD address; otherwise, other *obex* profiles may connect.
- When a remote device connection succeeds, a *[remote BD addr]* prompt will display.
- Enter *help* at the prompt to list available commands.
- Remote files are located in the current browsed remote folder. You can change directories with the *cd* command, and list a folder's contents with the *ls* command.
- With file transfer commands like *cp* and *mv*, the first argument is the source file and the second is the destination.
- For local files, include a colon character before the file path; for remote files, use just the path/filename.

§

Appendix A: SPP-loopback.py

```
#!/usr/bin/python

from __future__ import absolute_import, print_function, unicode_literals

from optparse import OptionParser, make_option
import os
import sys
import socket
import uuid
import dbus
import dbus.service
import dbus.mainloop.glib
try:
  from gi.repository import GObject
except ImportError:
  import gobject as GObject

class Profile(dbus.service.Object):
     fd = -1

     @dbus.service.method("org.bluez.Profile1",
                          in_signature="", out_signature="")
     def Release(self):
          print("Release")
          mainloop.quit()

     @dbus.service.method("org.bluez.Profile1",
                          in_signature="", out_signature="")
     def Cancel(self):
          print("Cancel")

     @dbus.service.method("org.bluez.Profile1",
                    in_signature="oha{sv}", out_signature="")
     def NewConnection(self, path, fd, properties):
          self.fd = fd.take()
          print("NewConnection(%s, %d)" % (path, self.fd))

          server_sock = socket.fromfd(self.fd, socket.AF_UNIX,
socket.SOCK_STREAM)
          server_sock.setblocking(1)
          server_sock.send("This is Edison SPP loopback test\nAll data will
be loopback\nPlease start:\n")

          try:
              while True:
                  data = server_sock.recv(1024)
                  print("received: %s" % data)
               server_sock.send("looping back: %s\n" % data)
          except IOError:
              pass
```

```
            server_sock.close()
            print("all done")

        @dbus.service.method("org.bluez.Profile1",
                             in_signature="o", out_signature="")
        def RequestDisconnection(self, path):
            print("RequestDisconnection(%s)" % (path))

            if (self.fd > 0):
                os.close(self.fd)
                self.fd = -1

if __name__ == '__main__':
        dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

        bus = dbus.SystemBus()

        manager = dbus.Interface(bus.get_object("org.bluez",
                             "/org/bluez"), "org.bluez.ProfileManager1")

        option_list = [
                  make_option("-C", "--channel", action="store",
                              type="int", dest="channel",
                              default=None),
                  ]

        parser = OptionParser(option_list=option_list)

        (options, args) = parser.parse_args()

        options.uuid = "1101"
        options.psm = "3"
        options.role = "server"
        options.name = "Edison SPP Loopback"
        options.service = "spp char loopback"
        options.path = "/foo/bar/profile"
        options.auto_connect = False
        options.record = ""

        profile = Profile(bus, options.path)

        mainloop = GObject.MainLoop()

        opts = {
                  "AutoConnect" :     options.auto_connect,
            }

        if (options.name):
            opts["Name"] = options.name

        if (options.role):
            opts["Role"] = options.role

        if (options.psm is not None):
            opts["PSM"] = dbus.UInt16(options.psm)
```

```
if (options.channel is not None):
     opts["Channel"] = dbus.UInt16(options.channel)

if (options.record):
     opts["ServiceRecord"] = options.record

if (options.service):
     opts["Service"] = options.service

if not options.uuid:
     options.uuid = str(uuid.uuid4())

manager.RegisterProfile(options.path, options.uuid, opts)

mainloop.run()
```

§