CHAPTER 16

Developing LE Applications

16.1 Introduction

In Chapter 5, we looked at some practical exercises that can be done on an Ubuntu system using the BlueZ stack to understand the various Bluetooth operations.

In this chapter, we will look at how to build an LE application based on the BlueZ stack. We will start with basic operations like setting up the development environment, enabling the Bluetooth adapter, advertising, and scanning. After that various GATT operations like discovering services, characteristics and reading and writing characteristics will be explained. Finally this chapter will explain the steps needed to make a real world application—an application to find lost keys.

Broadly this chapter will show examples of the various concepts that were explained in previous chapters and how various components come together to make a complete real world application. Besides the various procedures, this chapter will also explain what happens inside the protocol stack when those procedures are executed. In particular, this chapter will go into the transactions happening at the ATT level as well as the HCI level.

The BlueZ stack running in an Ubuntu environment has been selected for the examples due to several reasons:

1. This environment is easy to setup. The Ubuntu operating system can be downloaded and is pretty straightforward to install.
2. This environment is quite inexpensive. Only a couple of PCs are needed along with Bluetooth dongles and application development can be started.
3. The latest versions of BlueZ provide in-built support for LE. This means that the developer can directly focus on writing the application instead of first looking at how to bring up the various protocol stack layers.
4. BlueZ is open source. This means that the source code is available for reference at any time to see how exactly the different components are implemented. It can, for example, be used to see how the *gatttool* interfaces with the BlueZ stack to carry out various GATT operations.
5. Once the developer has made some enhancements to the BlueZ stack it can also be contributed back to the community so that others can also benefit from those.

6. BlueZ provides a powerful *hcidump* tool which can be used for analysis of the various procedures and what happens internally when those procedures are invoked. BlueZ also provides several utilities and sample examples which can be used for reference.
7. Even though the end environment may not be Linux- or BlueZ-based, these examples and sample source code will be very useful in understanding how a typical LE implementation works. This can help in quick ramp-up and result in speedier application development in the target environment.
8. BlueZ is a powerful environment to use as a peer device for testing since it supports both the LE only and dual mode roles. For example, if the developer is developing an LE device (Bluetooth Smart), then the BlueZ environment can be used as a Bluetooth Smart Ready device for testing that LE device. On the other hand if the developer is developing a Bluetooth Smart Ready device or application, the BlueZ environment can be used as a Bluetooth Smart device.

This chapter will also explain some of the tips and tricks that can be useful while developing and debugging an LE application. Some of the tools that can be used for assistance are also explained.

16.2 Ingredients

You will need the following:

1. Two PCs running any flavor of Linux (Preferably a current one to ensure it includes support for LE). For the purpose of examples, Ubuntu 12.10 is used here though any other Linux system will serve the purpose provided it's not too old. These two PCs will be referred to by the following names:
   a. *lepc*: This is the PC that will run the LE side of the sample applications.
   b. *dualpc*: This is the PC that will run the Dual mode side of the sample applications.
2. A couple of LE devices. If you search the internet you may find development kits from some vendors which can be used for developing LE applications. Some LE devices have also started appearing in the market and these are expected to grow rapidly. For the purpose of examples in this chapter, we will use a couple of PTS dongles and attach them via the USB interface to each of the Ubuntu PCs. The PTS dongle is available for purchase from the Bluetooth SIG website.

16.2.1 Installing hcidump

BlueZ comes with a very useful tool called *hcidump* which can be used to find out the packets being exchanged on the HCI interface. By default this tool is not installed on the Ubuntu system. The message shown in Figure 16.1 is received on invoking hcidump if this tool is not installed.

There are two methods for installing hcidump:

```
dualpc# hcidump
The program 'hcidump' is currently not installed. You can install it by
typing:
apt-get install bluez-hcidump
```

Figure 16.1 Invoking hcidump.

Method 1: The following command may be given to download and install it.

```
apt-get install bluez-hcidump
```

This should install hcidump on the system.

Method 2: The source code of the hcidump command may be downloaded from the bluez website (Reference [3]) and built. This method is explained in detail here.

The commands needed to build hcidump are shown in Figure 16.2. Some of the output of the commands is removed for ease of understanding.

```
dualpc# gunzip bluez-hcidump-2.3.tar.gz
dualpc#

dualpc# tar xvf bluez-hcidump-2.3.tar
bluez-hcidump-2.3/
.
.
.
bluez-hcidump-2.3/README
dualpc#

dualpc# cd bluez-hcidump-2.3
dualpc#
dualpc# ./configure
checking for ...
.
.
.
dualpc#

dualpc# make
make -no-print-directory all-am
  CC     src/bpasniff.o
.
.
.
dualpc#

dualpc# make install
test -z ...
.
.
.
dualpc#
```

Figure 16.2 Building hcidump.

The following are the steps needed to build hcidump.

1. Unpack the source code and prepare to build.

```
gunzip bluez-hcidump-2.3.tar.gz
tar xvf bluez-hcidump-2.3.tar
cd bluez-hcidump-2.3.tar
```

2. Configure the source code:

```
./configure
```

3. Build and Install

```
./make
./make install
```

Note that before doing a make install, root privileges may be required. This can be done by executing the command:

```
sudo su
```

16.2.2 Basic Bluetooth operations

Before trying the examples provided in this chapter, the two LE dongles will need to be connected to each of the two PCs: *lepc* and *dualpc*. Note that some of the hciconfig commands are supported only if the user has root privileges. So it's better to do the following before trying out the commands mentioned in this chapter:

```
sudo su
```

16.2.2.1 Invoking hcidump

It will be useful to start hcidump before giving any of the commands mentioned in the following section. This tool will show the various commands given on the HCI interface and the events received. Apart from the commands and events this tool also decodes the parameters of the commands and events. It also shows the higher layer transactions for ACL data packets. For example, for ACL data packets, this tool shows the transactions that happen at the ATT protocol layer. Figure 16.3 shows how to invoke hcidump. (The –i option is used to specify the HCI interface for which the tool should display the various commands and events. This will be explained shortly).

```
dualpc# hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
```

Figure 16.3 Invoking hcidump.

The best way to use this tool is to execute it in a separate terminal window so that all commands and events can be seen in that separate window and these don't garble the display of commands and events on the main window.

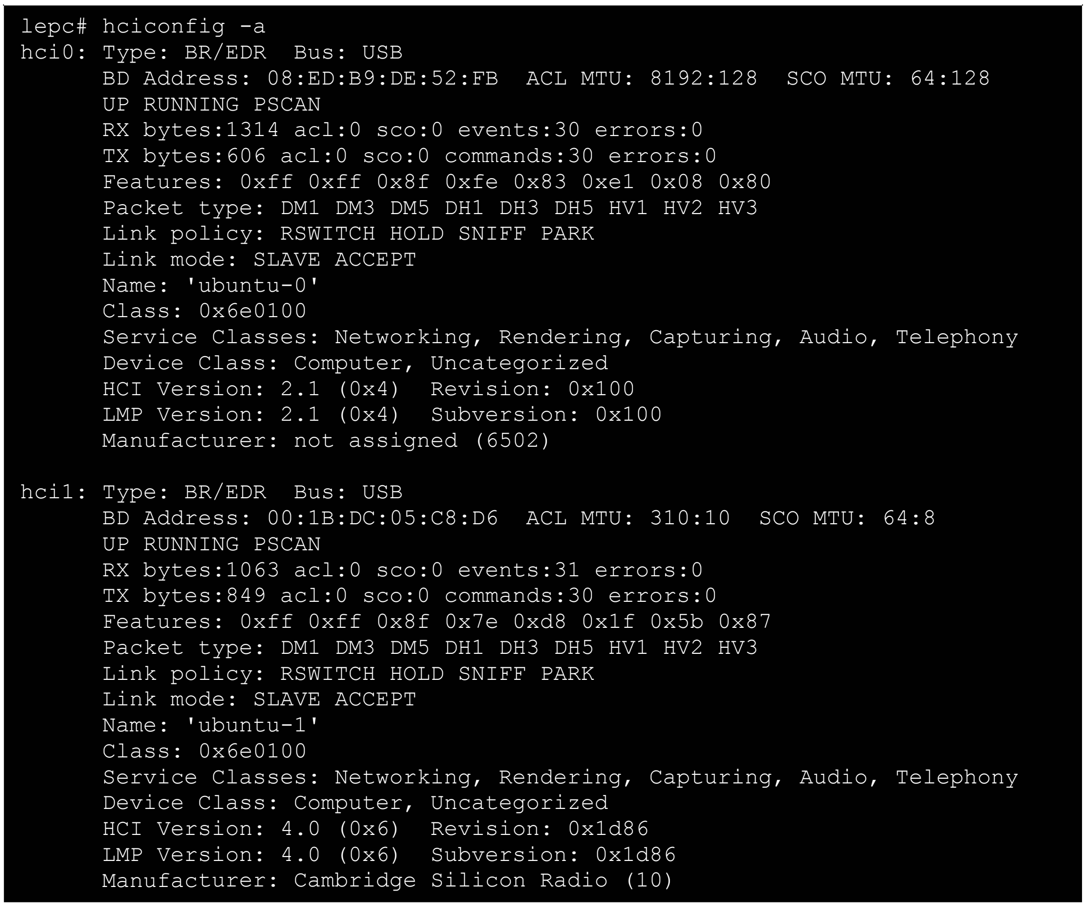
16.2.2.2 Accessing the LE device

Once a dongle (also known as Bluetooth adapter) is connected to a PC, it should be accessible through an *hci* interface. The *hci* interfaces are referred to by BlueZ as *hci0*, *hci1* and so on, depending on the number of Bluetooth adapters connected to the PC.

The *hciconfig* command can be used to configure the HCI adapter(s). The following command is used to find information about all HCI adapter(s) attached to the *lepc* PC:

```
hciconfig -a
```

The output of this command is shown in Figure 16.4. The following things may be observed:



```
lepc# hciconfig -a
hci0: Type: BR/EDR  Bus: USB
      BD Address: 08:ED:B9:DE:52:FB  ACL MTU: 8192:128  SCO MTU: 64:128
      UP RUNNING PSCAN
      RX bytes:1314 acl:0 sco:0 events:30 errors:0
      TX bytes:606 acl:0 sco:0 commands:30 errors:0
      Features: 0xff 0xff 0x8f 0xfe 0x83 0xe1 0x08 0x80
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy: RSWITCH HOLD SNIFF PARK
      Link mode: SLAVE ACCEPT
      Name: 'ubuntu-0'
      Class: 0x6e0100
      Service Classes: Networking, Rendering, Capturing, Audio, Telephony
      Device Class: Computer, Uncategorized
      HCI Version: 2.1 (0x4)  Revision: 0x100
      LMP Version: 2.1 (0x4)  Subversion: 0x100
      Manufacturer: not assigned (6502)

hci1: Type: BR/EDR  Bus: USB
      BD Address: 00:1B:DC:05:C8:D6  ACL MTU: 310:10  SCO MTU: 64:8
      UP RUNNING PSCAN
      RX bytes:1063 acl:0 sco:0 events:31 errors:0
      TX bytes:849 acl:0 sco:0 commands:30 errors:0
      Features: 0xff 0xff 0x8f 0x7e 0xd8 0x1f 0x5b 0x87
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy: RSWITCH HOLD SNIFF PARK
      Link mode: SLAVE ACCEPT
      Name: 'ubuntu-1'
      Class: 0x6e0100
      Service Classes: Networking, Rendering, Capturing, Audio, Telephony
      Device Class: Computer, Uncategorized
      HCI Version: 4.0 (0x6)  Revision: 0x1d86
      LMP Version: 4.0 (0x6)  Subversion: 0x1d86
      Manufacturer: Cambridge Silicon Radio (10)
```

Figure 16.4 Get information about all Bluetooth devices connected to the PC.

1. The PC has two Bluetooth adapters attached to it on hci0 and hci1.
2. The information about hci0 is as follows:
   a. BD_ADDR is 08:ED:B9:DE:52:FB.
   b. The ACL MTU size is 8192 bytes and the device has 128 buffers.
   c. The SCO MTU size is 64 bytes and the device has 128 buffers.
   d. The bitmap indicating features supported by the device is: ff ff 8f fe 83 e1 08 80. (This indicates the set of features that the link manager supports out of the feature mask definition.)
   e. The name of the device is 'ubuntu-0.'
   f. The Class of Device (CoD) is 0x6e0100. This means that it supports the following: Device Class: Computer, Uncategorized; Service Class: Networking, Rendering, Capturing, Audio, Telephony.
   g. The LMP version is 2.1. This means that this device support Bluetooth 2.1 specification.
3. The information about hci1 is as follows:
   a. BD_ADDR is 00:1B:DC:05:B5:B3.
   b. The ACL MTU size is 310 bytes and the device has 10 buffers.
   c. The SCO MTU size is 64 bytes and the device has 8 buffers.
   d. The bitmap indicating features supported by the device is: ff ff 8f 7e d8 1f 5b 87 (This indicates the set of features that the link manager supports out of the feature mask definition.)
   e. The name of the device is 'ubuntu-1.'
   f. The Class of Device (CoD) is 0x6e0100. This means that it supports the following: Device Class: Computer, Uncategorized; Service Class: Networking, Rendering, Capturing, Audio, Telephony.
   g. The LMP version is 4.0. This means that this device support Bluetooth 4.0 specification.

This indicates that hci0 is a BR/EDR device and hci1 is a dual mode device.

16.2.2.3 Opening and Closing HCI interface

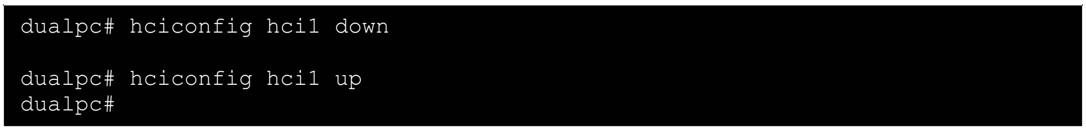
The HCI interface may be opened and closed by the hciconfig command. By default the HCI interface is open when Ubuntu boots up. The command to open the HCI interface is:

```
hciconfig hci1 up
```

The command to close the HCI interface is:

```
hciconfig hci1 down
```

These are shown in Figure 16.5.



```
dualpc# hciconfig hci1 down

dualpc# hciconfig hci1 up
dualpc#
```

Figure 16.5 Opening and Closing HCI interface.

The hcidump of the previous command that was used to bring up the HCI interface (hciconfig hci1 up) throws up several interesting bits of information. This is shown in Figure 16.6. Some of the lines of output are removed for easier understanding.

Some of the key points to note are:

1. The HCI Reset command is given at the beginning to reset the controller and bring it to a known state.
2. The Read Local Version Information command is used to find out the version of Bluetooth specification supported by the controller.
3. The LE Read Buffer Size command is used to read the number of LE buffers. This command returns 0 as the length of the LE packets. This means that the controller is using shared ACL buffers for LE and BR/EDR. The number of shared buffers in the controller is read by the HCI Read Buffer Size command. This was explained in Chapter 9.
4. The Write LE Host Supported command is used to inform the controller that the host supports LE.

16.2.2.4 Link Manager Supported States

The various states of the link manager were explained in Chapter 8. BlueZ supports the following command to get the list of states supported by the link manager:

```
hciconfig hci1 lestates
```

Note that this command takes the HCI adapter as an argument. In this case, the LE dongle is attached on hci1 interface. So the command is used with the first argument as hci1.

The output of this command is shown in Figure 16.7. The device in this case supports the following LM states:

- Non-connectable Advertising State.
- Scannable Advertising State.
- Connectable Advertising State.

There are many more possible states that the device supports. The complete list is shown in Figure 16.7.

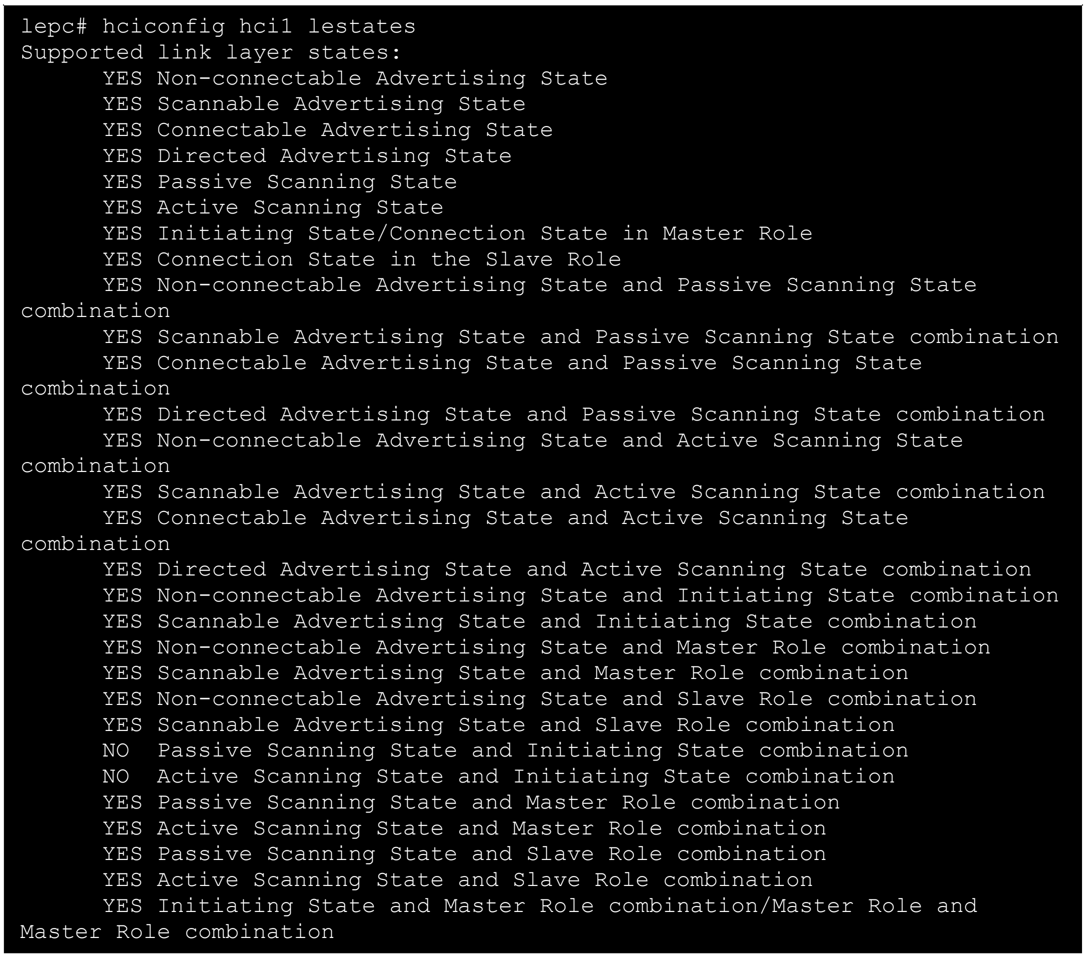
The commands and events exchanged on the HCI interface as shown in Figure 16.8. It shows that the HCI command LE_Read_Supported_States is used to get the list of supported states. This command was explained in Chapter 9.

```
< HCI Command: Reset (0x03|0x0003) plen 0
> HCI Event: Command Complete (0x0e) plen 4
    Reset (0x03|0x0003) ncmd 1
    status 0x00
< HCI Command: Read Local Supported Features (0x04|0x0003) plen 0
> HCI Event: Command Complete (0x0e) plen 12
    Read Local Supported Features (0x04|0x0003) ncmd 1
    status 0x00
    Features: 0xff 0xff 0x8f 0x7e 0xd8 0x1f 0x5b 0x87
< HCI Command: Read Local Version Information (0x04|0x0001) plen 0
> HCI Event: Command Complete (0x0e) plen 12
    Read Local Version Information (0x04|0x0001) ncmd 1
    status 0x00
    HCI Version: 4.0 (0x6) HCI Revision: 0x1d86
    LMP Version: 4.0 (0x6) LMP Subversion: 0x1d86
    Manufacturer: Cambridge Silicon Radio (10)
< HCI Command: Read Buffer Size (0x04|0x0005) plen 0
> HCI Event: Command Complete (0x0e) plen 11
    Read Buffer Size (0x04|0x0005) ncmd 1
    status 0x00
    ACL MTU 310:10 SCO MTU 64:8
< HCI Command: Read BD ADDR (0x04|0x0009) plen 0
> HCI Event: Command Complete (0x0e) plen 10
    Read BD ADDR (0x04|0x0009) ncmd 1
    status 0x00 bdaddr 00:1B:DC:05:B5:B3
< HCI Command: Set Event Mask (0x03|0x0001) plen 8
    Mask: 0xfffffbff07f8bf3d
> HCI Event: Command Complete (0x0e) plen 4
    Set Event Mask (0x03|0x0001) ncmd 1
    status 0x00
< HCI Command: Read Local Supported Commands (0x04|0x0002) plen 0
> HCI Event: Command Complete (0x0e) plen 68
    Read Local Supported Commands (0x04|0x0002) ncmd 1
    status 0x00
    Commands: ffffff03feffffffffffffff30fe8fe3ff783ff1c00000061f7ffff7f
< HCI Command: Write LE Host Supported (0x03|0x006d) plen 2
  0000: 01 01                                              ..
> HCI Event: Command Complete (0x0e) plen 4
    Write LE Host Supported (0x03|0x006d) ncmd 1
    0000: 00                                                .
< HCI Command: LE Read Buffer Size (0x08|0x0002) plen 0
> HCI Event: Command Complete (0x0e) plen 7
    LE Read Buffer Size (0x08|0x0002) ncmd 1
    status 0x00 pktlen 0x0000 maxpkt 0x00
< HCI Command: Write Simple Pairing Mode (0x03|0x0056) plen 1
    mode 0x01
> HCI Event: Command Complete (0x0e) plen 4
    Write Simple Pairing Mode (0x03|0x0056) ncmd 1
    status 0x00
< HCI Command: Write LE Host Supported (0x03|0x006d) plen 2
  0000: 01 01                                              ..
> HCI Event: Command Complete (0x0e) plen 4
    Write LE Host Supported (0x03|0x006d) ncmd 1
    0000: 00                                                .
```

Figure 16.6 HCI transactions during hciconfig hci1 up.

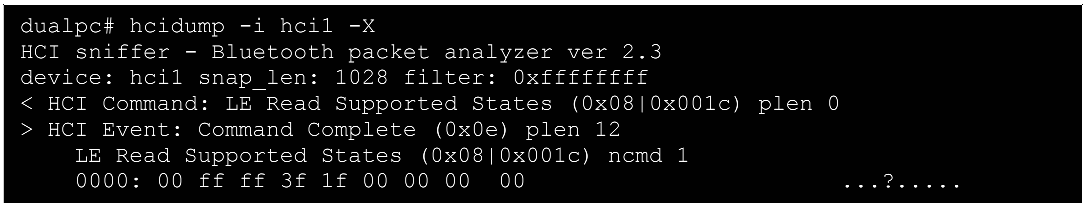
16.3 Advertising and Scanning

The advertising and scanning procedures were explained in Chapter 8.



```
lepc# hciconfig hci1 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
      NO  Passive Scanning State and Initiating State combination
      NO  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
```

Figure 16.7 Retrieving the list of LM states supported.



```
dualpc# hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
< HCI Command: LE Read Supported States (0x08|0x001c) plen 0
> HCI Event: Command Complete (0x0e) plen 12
    LE Read Supported States (0x08|0x001c) ncmd 1
    0000: 00 ff ff 3f 1f 00 00 00  00                       ...?.....
```

Figure 16.8 HCI transactions for getting LE states.

The following command is used to enable advertising on the PC *lepc*.

```
hciconfig hci1 leadv
```

The following command is used for scanning on PC *dualpc*.

```
hcitool –i hci1 lescan
```

The advertising command is shown in Figure 16.9. The corresponding output of hcidump command is also shown in Figure 16.9. As may be noted, the HCI command LE_Set_Advertise_Enable is used to enable advertising.

The command for scanning is shown in Figure 16.10. The BD_ADDR for lepc is 00:1B:DC:05:C8:D6. When *dualpc* does a scanning procedure, this device is shown amongst the list of devices found.

The corresponding output of hcidump command is also shown in Figure 16.10. As may be noted, the HCI commands LE_Set_Scan_Parameters and LE_Set_Scan_Enable are used to enable scanning. The remote devices are informed by the controller by sending LE_Advertising_Report events.

16.4 Creating a Connection

Creating an LE connection is quite simple. Once the *lepc* is in advertising mode (as shown in previous section) the following command can be given to create an LE connection by *dualpc:*

```
hcitool -i hci1 lecc BD_ADDR_OF_lepc
```

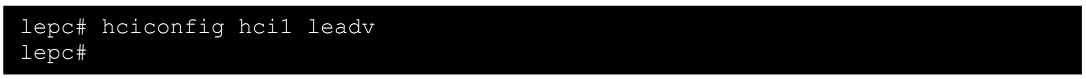
This is shown in Figure 16.11. It may be noted that on successful connection this command returns the connection handle. For example, in this case, the connection handle assigned to the LE connection is 39.

16.5 GATT Operations

16.5.1 Enable GATT Functionality on Server

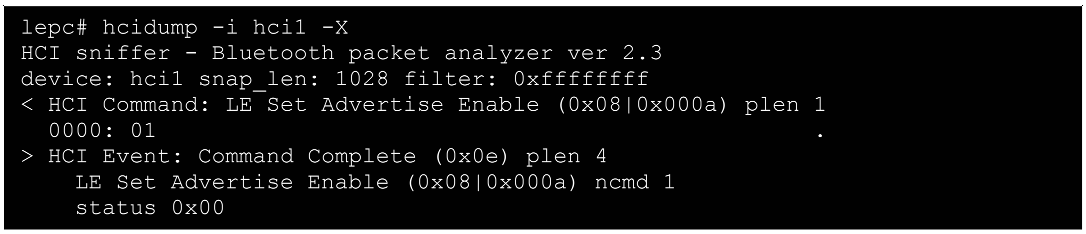
BlueZ on ubuntu includes a GATT server which provides support for FindMe and Proximity profiles. It includes the following services:

1. Generic Access Service.
2. Generic Attribute Service.



```
lepc# hciconfig hci1 leadv
lepc#
```

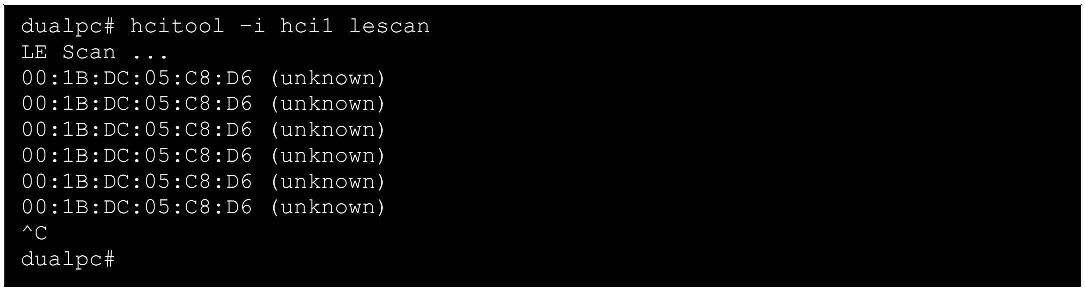
Advertising on lepc



```
lepc# hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
< HCI Command: LE Set Advertise Enable (0x08|0x000a) plen 1
  0000: 01                                             .
> HCI Event: Command Complete (0x0e) plen 4
    LE Set Advertise Enable (0x08|0x000a) ncmd 1
    status 0x00
```

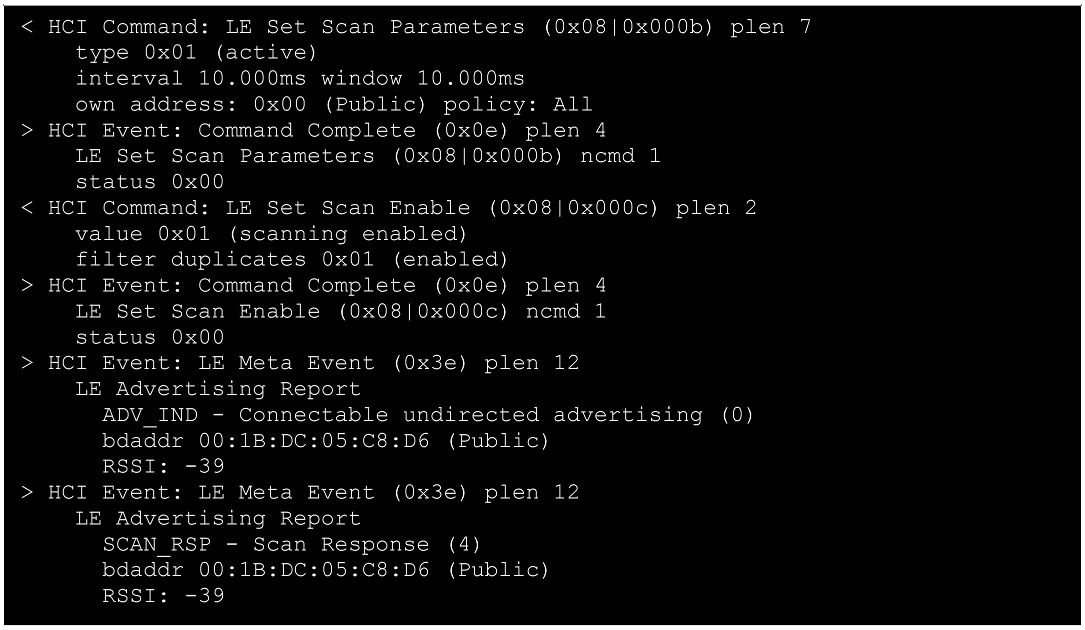
HCI commands and events for advertising

Figure 16.9 Advertising.



```
dualpc# hcitool -i hci1 lescan
LE Scan ...
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
00:1B:DC:05:C8:D6 (unknown)
^C
dualpc#
```

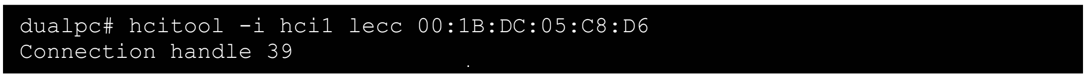
Scanning on dualpc



```
< HCI Command: LE Set Scan Parameters (0x08|0x000b) plen 7
    type 0x01 (active)
    interval 10.000ms window 10.000ms
    own address: 0x00 (Public) policy: All
> 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 12
    LE Advertising Report
      ADV_IND - Connectable undirected advertising (0)
      bdaddr 00:1B:DC:05:C8:D6 (Public)
      RSSI: -39
> HCI Event: LE Meta Event (0x3e) plen 12
    LE Advertising Report
      SCAN_RSP - Scan Response (4)
      bdaddr 00:1B:DC:05:C8:D6 (Public)
      RSSI: -39
```

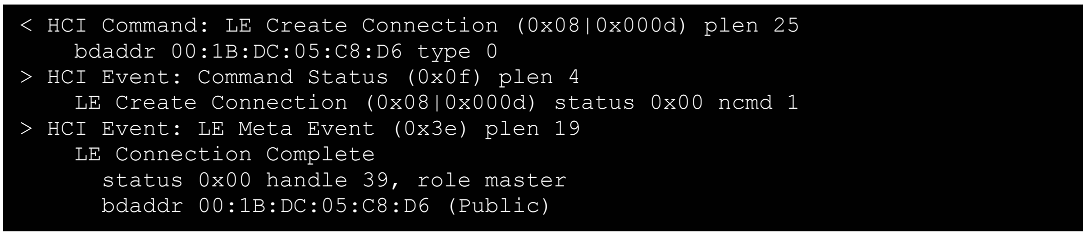
HCI commands and events for scanning

Figure 16.10 Scanning.



```
dualpc# hcitool -i hci1 lecc 00:1B:DC:05:C8:D6
Connection handle 39
```

Creating connection from dualpc to lepc



```
< HCI Command: LE Create Connection (0x08|0x000d) plen 25
    bdaddr 00:1B:DC:05:C8:D6 type 0
> 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 39, role master
      bdaddr 00:1B:DC:05:C8:D6 (Public)
```

HCI commands and events for creating a connection

Figure 16.11 Creating an LE connection.

3. Link Loss Service.
4. Immediate Alert Service.
5. Tx Power Service.

This section configures *lepc* as a GATT server. To do this, the GATT server needs to be enabled. The steps to do this are as follows:

1. Set *EnableGatt* to *true* in /etc/bluetooth/main.conf.
2. Kill the Bluetooth daemon by giving the command.

```
sudo killall bluetoothd
```

3. Start Bluetooth daemon again by giving the command.

```
sudo /usr/sbin/bluetoothd
```

16.5.2 Execute GATT Procedures from the Client

This section configures dualpc as the GATT client and uses the gatttool application. The gatttool application can be used to send a single request to the GATT layer or it can also be used in an interactive mode where a series of requests can be sent interactively. In the interactive mode, gatttool comes up with a command prompt where the requests can be given. This section will explain how to run the gatttool in the interactive mode.

The syntax of the gatttool application is:

```
gatttool [option…]
```

It can be invoked as follows:

```
gatttool -i hciX -I -b BD_ADDR_OF_REMOTE_DEVICE
```

where:

-I hciX specifies the HCI Interface (local adapter interface)

-b BD_ADDR_OF_REMOTE_DEVICE specifies the BD_ADDR of the remote device

-I specifies to use the interactive mode

In this example, the gatttool command is run as follows:

```
gatttool -i hci1 -I -b 00:1B:DC:05:C8:D6
```

The –I switch is used to run gatttool in interactive mode so that further commands can be given at the gatttool command prompt. Once this command is executed, the gatttool shows its own prompt where further commands can be given. The prompt shown by gatttool is shown in Figure 16.12.

```
dualpc# gatttool -i hci1 -I -b 00:1B:DC:05:C8:D6
[   ][00:1B:DC:05:C8:D6][LE]>
```

Figure 16.12 Running gatttool in interactive mode.

16.5.2.1 Connecting from the Client

The command to create a connection is:

```
connect
```

Once this command is given, the display changes as shown in Figure 16.13. Note that the prompt in the first square bracket has changed to CON to indicate that GATT client is now connected to a GATT server.

16.5.2.2 Getting the Primary Services

The command to get the primary services is:

```
primary
```

The output of this command is shown in Figure 16.14.

The output shows the following for each primary service:

- Attribute Handle;
- End Group Handle;

```
[   ][00:1B:DC:05:C8:D6][LE]> connect
[CON][00:1B:DC:05:C8:D6][LE]>
```

Figure 16.13 Connecting from the client.

```
[CON][00:1B:DC:05:C8:D6][LE]> primary
[CON][00:1B:DC:05:C8:D6][LE]>
attr handle: 0x0001, end grp handle: 0x0008 uuid: 00001800-0000-1000-8000-
00805f9b34fb
attr handle: 0x0009, end grp handle: 0x000b uuid: 00001803-0000-1000-8000-
00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001804-0000-1000-8000-
00805f9b34fb
attr handle: 0x0010, end grp handle: 0x0010 uuid: 00001801-0000-1000-8000-
00805f9b34fb
attr handle: 0x0011, end grp handle: 0x0013 uuid: 00001802-0000-1000-8000-
00805f9b34fb
attr handle: 0x0014, end grp handle: 0x0017 uuid: 0000a002-0000-1000-8000-
00805f9b34fb
attr handle: 0x0022, end grp handle: 0x002c uuid: 0000a004-0000-1000-8000-
00805f9b34fb
attr handle: 0xfffa, end grp handle: 0xfffe uuid: feee74dc-a8de-3196-1149-
d43596c00a4f
[CON][00:1B:DC:05:C8:D6][LE]>
```

Figure 16.14 Getting the primary services.

- UUID of that service (128-bit UUID of the service).

It indicates the primary services as shown in Table 16.1. Note that the service UUID has been converted from 128-bit to 16-bit. As explained in Chapter 12, a 16-bit UUID can be derived from the 128-bit bit UUID for the UUIDs that are already defined by the Bluetooth SIG. It is shown as xxxx in the 128-bit representation when the representation is in the form of 0000xxxx-0000-1000-8000-00805F9B34FB.

The service UUIDs are mapped to the Service Names by referring to the Bluetooth Assigned Numbers for GATT (see Reference [4]) and GATT based Service UUIDs (see Reference [5]).

The transactions that are initiated on the ATT level for getting the primary services can be captured by using the hcidump tool. The output of the hcidump tool for getting the primary services is shown in Figure 16.15. The following points may be noted:

1. The ATT Read_By_Group_Type request is used to read the services. It is executed with the following parameters:
   a. Start Handle: 0x0001.
   b. Ending Handle: 0xffff.
   c. UUID: 0x2800 (This means that <<Primary Service>> is requested).
2. All ATT methods use the ACL packets to send and receive data.
3. The ATT Read_By_Group_Type response returns the first three services with the End Group Handle of the last service being 0x0f.
4. The ATT Read_By_Group_Type request is used again to read the next set of services. It is executed with the following parameters:
   a. Start Handle: 0x0010.
   b. Ending Handle: 0xffff.
   c. UUID: 0x2800 (This means that <<Primary Service>>) is requested.
5. This continues till an Error Response is received from the remote side.

16.5.2.3 Getting the Characteristics

The command to get the characteristics that are included in the service is:

Table 16.1 Primary Services on the GATT Server

S. No	*Service UUID*	*Service Name*	*Starting Handle*	*Ending Handle*
1	1800	<<Generic Access Service>>	0x0001	0x0008
2	1803	<<Link Loss Service>>	0x0009	0x000b
3	1804	<<Tx Power Service>>	0x000c	0x000f
4	1801	<<Generic Attribute Servie>>	0x0010	0x0010
5	1802	<<Immediate Alert Service>>	0x0011	0x0013
6	a002		0x0014	0x0017
7	a004		0x0022	0x002c
8	User define 128-bit	User defined service	0xfffa	0xfffe

```
dualpc# hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
< ACL data: handle 39 flags 0x00 dlen 11
    ATT: Read By Group req (0x10)
      start 0x0001, end 0xffff
      type-uuid 0x2800
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 24
    ATT: Read By Group resp (0x11)
      attr handle 0x0001, end group handle 0x0008
      value 0x00 0x18
      attr handle 0x0009, end group handle 0x000b
      value 0x03 0x18
      attr handle 0x000c, end group handle 0x000f
      value 0x04 0x18
< ACL data: handle 39 flags 0x00 dlen 11
    ATT: Read By Group req (0x10)
      start 0x0010, end 0xffff
      type-uuid 0x2800
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 24
    ATT: Read By Group resp (0x11)
      attr handle 0x0010, end group handle 0x0010
      value 0x01 0x18
      attr handle 0x0011, end group handle 0x0013
      value 0x02 0x18
      attr handle 0x0014, end group handle 0x0017
      value 0x02 0xa0
```

Figure 16.15 ATT procedures during GATT primary service discovery.

```
characteristics start_handle end_handle
```

The output of this command is shown in Figure 16.16.

Table 16.2 shows the following characteristics for service <<Generic Access Profile>> (UUID=0x1800).

```
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0001 0x0008
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x0004, char properties: 0x02, char value handle: 0x0006, uuid:
00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0007, char properties: 0x02, char value handle: 0x0008, uuid:
00002a01-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]>
```

Figure 16.16 Getting the characteristics of a service.

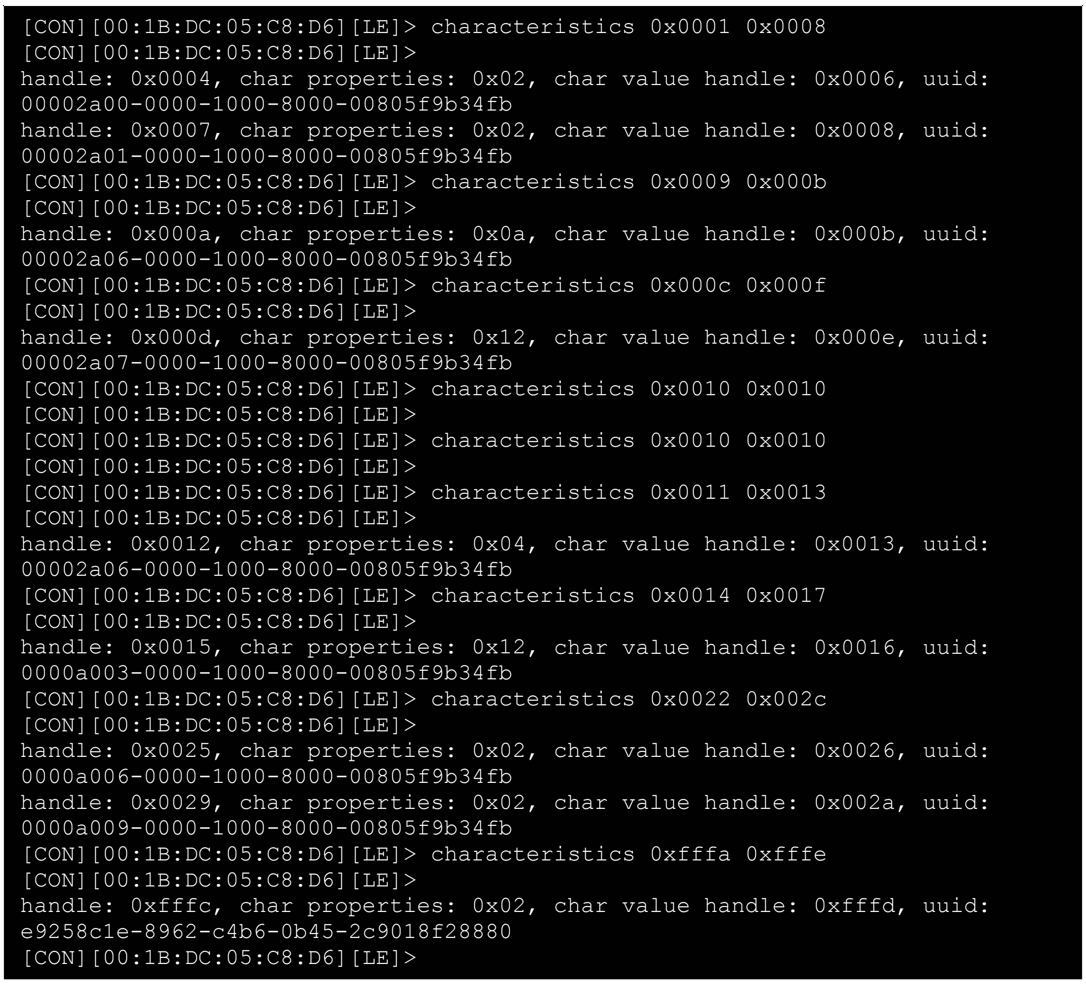
Table 16.2 Characteristics of Generic Access Profile

S. No	*Characteristic UUID*	*Characteristic Name*	*Characteristic Handle*	*Characteristic Properties*	*Characteristic Value Handle*
1	2a00	<<Device Name>>	0x0004	0x02	0x0006
2	2a01	<<Appearance Characteristic>>	0x0007	0x02	0x0008

Note that the Characteristic UUID is mapped to Characteristic Name in the Bluetooth Assigned Numbers. (See Reference [6]). Similarly, the characteristics of each of the service can also be retrieved. Figure 16.17 shows the commands used to retrieve the characteristics of all the services. The meaning of each of the characteristics is shown in Table 16.3.

A few points worth noting in the table are:

1. The Generic Access Service contains two characteristics—Device Name and Appearance Characteristic. Both these services were explained in detail in Chapter 14.
2. The Generic Attribute Service does not contain any characteristic in this example. It can only contain one characteristic—Service Changed Characteristic which was explained in Chapter 13. In this particular device this characteristic is missing. This means that the list of services supported by this device cannot change. In general, if this service is absent for an LE only implementation, this would have given information to the clients that the services cannot change over the entire lifetime of the device.



```
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0001 0x0008
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x0004, char properties: 0x02, char value handle: 0x0006, uuid:
00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0007, char properties: 0x02, char value handle: 0x0008, uuid:
00002a01-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0009 0x000b
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x000a, char properties: 0x0a, char value handle: 0x000b, uuid:
00002a06-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x000c 0x000f
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x000d, char properties: 0x12, char value handle: 0x000e, uuid:
00002a07-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0010 0x0010
[CON][00:1B:DC:05:C8:D6][LE]>
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0010 0x0010
[CON][00:1B:DC:05:C8:D6][LE]>
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0011 0x0013
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x0012, char properties: 0x04, char value handle: 0x0013, uuid:
00002a06-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0014 0x0017
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x0015, char properties: 0x12, char value handle: 0x0016, uuid:
0000a003-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0x0022 0x002c
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0x0025, char properties: 0x02, char value handle: 0x0026, uuid:
0000a006-0000-1000-8000-00805f9b34fb
handle: 0x0029, char properties: 0x02, char value handle: 0x002a, uuid:
0000a009-0000-1000-8000-00805f9b34fb
[CON][00:1B:DC:05:C8:D6][LE]> characteristics 0xfffa 0xfffe
[CON][00:1B:DC:05:C8:D6][LE]>
handle: 0xfffc, char properties: 0x02, char value handle: 0xfffd, uuid:
e9258c1e-8962-c4b6-0b45-2c9018f28880
[CON][00:1B:DC:05:C8:D6][LE]>
```

Figure 16.17 Getting the characteristics of all services.

Table 16.3 List of All Characteristics within the Primary Services

				Characteristic Value Related	
S. No	*Characteristic Handle*	*Characteristic UUID*	*Characteristic Name*	*Characteristic Properties*	*Characteristic Value Handle*
Generic Access Service [0x0001 – 0x0008]					
1	0x0004	2a00	<<Device Name>>	0x02	0x0006
2	0x0007	2a01	<<Appearance Characteristic>>	0x02	0x0008
Link Loss Service [0x0009 – 0x000b]					
3	0x000a	2a06	<<Alert Level>>	0x0a	0x000b
Tx Power Service [0x000c – 0x000f]					
4	0x000d	2a07	<<Tx Power Level>>	0x12	0x000e
Generic Attribute Service [0x0010 – 0x0010] – No Characteristics					
Immediate Alert Service [0x0011 – 0x0013]					
5	0x0012	2a06	<<Alert Level>>	0x04	0x0013
a002 [0x0014 – 0x0017]					
6	0x0015	a003		0x12	0x0016
a004 [0x0022 – 0x002c]					
7	0x0025	a006		0x02	0x0026
8	0x0029	a009		0x02	0x002a
feee74dc-a8de-3196-1149-d43596c00a4f [0xfffa – 0xfffe]					
9	0xfffc	User defined 128-bit UUID		0x02	0xfffd

3. The meaning of the different bits in the bit-map of characteristic values was explained in Chapter 13. For example:
    a. 0x02 means that only read is permitted on the Characteristic Value.
    b. 0x4 means that only write without response is permitted.
    c. 0x0a means that the Characteristic Value can be read and written.
    d. 0x12 means that it can be read and notified.
4. There are two instances of <<Alert Level>> characteristic:
    a. First time as a part of Link Loss Service where it has permissions of 0x0a (read, write permitted).
    b. Second time as a part of Immediate Alert Service where it has permissions of 0x04 (It can only be written). This was also explained in Chapter 15.
5. There are certain user defined UUIDs apart from the ones defined in Bluetooth Assigned Numbers.
6. The handle to the Characteristic Value is provided in the last column. This is the handle that would be used to access the characteristic provided there are sufficient permissions as per Characteristic Properties.

The transactions that are initiated on the ATT level for getting the characteristics are shown in Figure 16.15.

The following points may be noted:

1. The ATT Read_By_Type request is used to read the characteristics. It is executed with the following parameters:
   a. Start Handle: 0x0001.
   b. Ending Handle: 0x0008 (Handles that were received for the first service).
   c. UUID: 0x2803 (This is the UUID for means that <<Characteristic>>).
2. The ATT Read_By_Type response returns the two characteristics.

16.5.3 Reading and Writing Characteristics

Now that the handles for all the characteristics values and characteristic permissions are available, the next step is to read and write these characteristics.

16.5.3.1 Reading and Writing Alert Level in Link Loss Service

This section will show how to read and write the Alert Level Characteristic in the Link Loss Service. This has an attribute handle of 0x000b and has Characteristic Properties of 0x0a. This means that this Characteristic Value can be read and written. This is in line with the description of Link Loss Service in Chapter 15.

The read and write operations are shown in Figure 16.19. The following may be noted:

1. The syntax of the command to read Characteristic Value is:

```
dualpc # hcidump -i hci1 -X
HCI sniffer - Bluetooth packet analyzer ver 2.3
device: hci1 snap_len: 1028 filter: 0xffffffff
< ACL data: handle 39 flags 0x00 dlen 11
    ATT: Read By Type req (0x08)
      start 0x0001, end 0x0008
      type-uuid 0x2803
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 20
    ATT: Read By Type resp (0x09)
      length: 7
        handle 0x0004, value 0x02 0x06 0x00 0x00 0x2a
        handle 0x0007, value 0x02 0x08 0x00 0x01 0x2a
```

Figure 16.18 ATT procedures for reading characteristics.

```
[CON][00:1B:DC:05:C8:D6][LE]> char-write-cmd 0x000b 0x01
[CON][00:1B:DC:05:C8:D6][LE]> char-read-hnd 0x000b
Characteristic value/descriptor: 01

[CON][00:1B:DC:05:C8:D6][LE]> char-write-cmd 0x000b 0x02
[CON][00:1B:DC:05:C8:D6][LE]> char-read-hnd 0x000b
Characteristic value/descriptor: 02
```

Figure 16.19 Reading and writing the Alert Level Characteristic in the Link Loss Service.

```
char-read-hnd <handle> [offset]
```

2. The syntax of the command to write Characteristic Value is:

```
char-write-cmd <handle> <new value>
```

3. The first command writes the value 0x01 to the Alert Level Characteristic. The command is:

```
char-write-cmd 0x000b 0x01
```

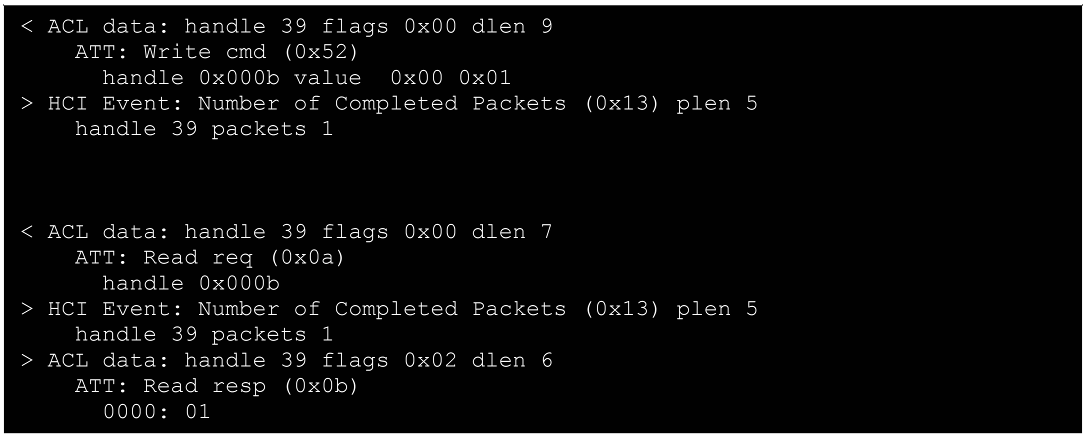
4. The second command reads the value of the Alert Level Characteristic. The command is:

```
char-read-hnd 0x000b
```

5. This returns the same value that was written. This confirms that the value was written correctly.
6. The next two commands do the same write and read operations with a value of 0x02.

The transactions that are initiated on the ATT level for reading and writing characteristics are shown in Figure 16.20. The following points may be noted:

1. The ATT Write command is used to write the characteristics. It is executed with the following parameters:
   a. Handle: 0x000b.
   b. Value: 0x0001.
2. The ATT Read request is used to read a characteristic. It is executed with the following parameters:
   a. Handle: 0x000b.



```
< ACL data: handle 39 flags 0x00 dlen 9
    ATT: Write cmd (0x52)
      handle 0x000b value  0x00 0x01
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1



< ACL data: handle 39 flags 0x00 dlen 7
    ATT: Read req (0x0a)
      handle 0x000b
> HCI Event: Number of Completed Packets (0x13) plen 5
    handle 39 packets 1
> ACL data: handle 39 flags 0x02 dlen 6
    ATT: Read resp (0x0b)
      0000: 01
```

Figure 16.20 ATT procedures for reading and writing characteristics.

3. The remote side responds with a ATT Read response. It contains the following parameters:
   a. Value: 0x0001.

16.5.3.2 Writing Alert Level in Immediate Alert Service

This section will show how to read and write the Alert Level Characteristic in the Immediate Alert Service. This has an attribute handle of 0x0013 and has Characteristic Properties of 0x04. This means that this Characteristic Value can only be written. This is in line with the description of Immediate Alert Service in Chapter 15.

The write operations are shown in Figure 16.21. The following may be noted:

1. The first command writes the value 0x01 to the Alert Level Characteristic. The command is:

   ```
   char-write-cmd 0x0013 0x01
   ```

2. The second command writes the value of 0x02 to the Alert Level Characteristic. The command is:

   ```
   char-write-cmd 0x0013 0x01
   ```

3. The third command writes the value of 0x00 to the Alert Level Characteristic. The command is:

   ```
   char-write-cmd 0x0013 0x00
   ```

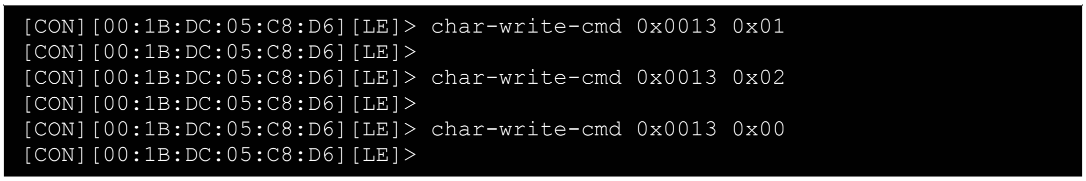
16.6 Disconnecting

16.6.1 Disconnecting the GATT Connection

The GATT connection can be disconnected by the following command:

```
disconnect
```

This is shown in Figure 16.22. Note that the CON message disappears in the second line after giving the disconnect command. This indicates that the connection is no longer there. The final command given is `quit` to come out the gatttool interactive interface.



```
[CON][00:1B:DC:05:C8:D6][LE]> char-write-cmd 0x0013 0x01
[CON][00:1B:DC:05:C8:D6][LE]>
[CON][00:1B:DC:05:C8:D6][LE]> char-write-cmd 0x0013 0x02
[CON][00:1B:DC:05:C8:D6][LE]>
[CON][00:1B:DC:05:C8:D6][LE]> char-write-cmd 0x0013 0x00
[CON][00:1B:DC:05:C8:D6][LE]>
```

Figure 16.21 Writing the Alert Level Characteristic in the Immediate Alert Service.

```
[CON][00:1B:DC:05:C8:D6][LE]> disconnect

[   ][00:1B:DC:05:C8:D6][LE]> quit
dualpc#
```

Figure 16.22 Disconnecting the GATT connection.

16.6.2 Disconnecting the LE Connection

The existing LE connection can be disconnected by the following command on *dualpc:*

```
hcitool -i hci1 ledc connection_handle
```

This is shown in Figure 16.23. The connection handle is that same that was returned while creating the LE connection in Figure 16.11. The HCI Disconnect command is used to terminate the connection.

16.7 Real-World Application—Find Lost Keys

The previous sections provided all the components that are needed to make a real world application to find lost keys. These components can be used as building blocks to make a full real world LE application.

An example of such an application can be one used to find lost keys (or any other lost devices). The example of this was explained in Chapter 1. Such an application will use the Find Me profile which was explained in Chapter 15.

The broad steps this application will need to perform are as follows:

1. Search for LE devices in the vicinity.
2. Select the device that has been lost.
3. Create a connection to the device.
4. Create a GATT connection.
5. Get the primary services.
6. Check if it contains the Immediate Alert Service.

```
dualpc# hcitool -i hci1 ledc 39
dualpc#
```

Disconnecting from dualpc

```
< HCI Command: Disconnect (0x01|0x0006) plen 3
    handle 39 reason 0x13
    Reason: Remote User Terminated Connection
> HCI Event: Command Status (0x0f) plen 4
    Disconnect (0x01|0x0006) status 0x00 ncmd 1
> HCI Event: Disconn Complete (0x05) plen 4
    status 0x00 handle 39 reason 0x16
    Reason: Connection Terminated by Local Host
```

HCI commands and events for disconnection

Figure 16.23 Disconnecting the LE connection.

7. If it contains the Immediate Alert Service, then find the characteristics contained in this service.
8. This will return the Characteristic Value Handle for the Alert Level characteristic.
9. Write a “High Alert” to the Characteristic Value handle.
10. The key fob should start raising an alert (Audio or Visual indication).
11. Disconnect the GATT and LE connection.

The commands to be sent for each of these steps (along with the parameters) were explained in previous sections. These can be put together into a script or a program to perform all the steps. Applications can also use services provided by BlueZ through the *D-Bus* interface. *D-bus* is a message bus system that can be used for interprocess communication. The applications can be written in either Python or C and are simple and straightforward to implement.

16.8 Debugging LE Applications

16.8.1 Logging the HCI Interface

The first and most common mechanism used for debugging Bluetooth applications is logging the packets exchanged on the HCI interface. This includes commands sent from the host to the controller and the events that are sent back by the controller.

BlueZ includes the *hcidump* tool which provides the facility to print command and event packets exchanged on the HCI interface. A similar tool may be available on the system on which the LE application is developed. If such a tool is not available, adding the facility to log the packets may be as easy as dumping (or printing) the packets that are sent and received.

At a very preliminary level, the packets sent and received can be analyzed to check if the transactions initiated by the application or the remote side were correct. For example, if the host sends an LE connection command, the packets at the HCI level can be checked to see if the command packet was sent with the correct values and to the correct device.

At a bit more advanced level, the packets can be decoded to print the friendly names of the commands along with meaning of each of the arguments that were passed. Besides this, the higher layer protocol data can also be decoded and used for debugging instead of just confining to the HCI interface. This facility is provided by the hcidump tool (examples of this were shown earlier.) For example, the ATT level transactions at the time of discovering services of the remote device were shown in Figure 16.15. The source code for hcidump is also available from the BlueZ website. This could be used as a reference for implementation in other environments as well.

At a still more advanced level, the packets could be converted to a format which a sniffer tool can open so that the packets can directly be opened in the sniffer tool. The sniffer can provide more detailed analysis of the packets which can be useful for debugging. One such format is the *BT-Snoop* file format. The conversion from the raw send/receive packets to the BT-Snoop format is straightforward and can be done using a simple script.

16.8.2 Air Sniffer

Air sniffer is a very powerful tool to understand what is happening on the air, and to quickly find the root cause of any problems. It provides several features including:

1. Analysis of the various packets that are sent over the air. The packets can be analyzed at various levels starting from the baseband layer up to the protocol layers. The various transactions along with the parameters are displayed in an easy to understand format.
2. The profile level data may also be extracted for separate analysis.
3. Display of the used/unused channels.
4. Message sequence charts of the various transactions.
5. Facility to save logs so that these can be reopened for further analysis later on.

These are only some of the features provided by the air sniffers. Most of the sniffers available in the market provide several other advanced features that can simplify the LE development a great deal.

16.8.3 Peer Devices and Interoperability Testing

Bluetooth communications require at least two devices to interact with each other. In many of the scenarios, the application writer or device manufacturer may be focusing only on one of the roles. For example, if an LE key fob is being made, then the application writer needs to focus on the key fob functionality only which may include the Immediate Alert Service. Though the peer functionality is also important from a testing perspective, the application writer may not need to spend time on writing that functionality from scratch. Instead of that, a peer device which already has this functionality can be used. In this particular example, a Linux based PC can be used as a peer device to test all the functionality of the key fob. A good selection of peer device can help in speedier development of applications and also to ensure that the device is exhaustively tested before releasing to the marked.

Once the device is functioning well, the set of peer devices can be increased for getting further testing coverage. This is known as *interoperability testing* where the device is tested with several other devices to ensure that it works well with all those devices. It will be explained in detail in the next chapter.

16.8.4 Profile Tuning Suite (PTS)

The Profile Tuning Suite is a powerful, PC-based, black box testing tool provided by the Bluetooth SIG. It can be used during product development, testing, and qualification stages. It has support for most of the BR/EDR profiles and several LE profiles as well. It is semi-automated which leads to reduction of testing time and also ensuring that novice users can ramp-up in using this tool faster. It provides an ample amount of debugging information along with message sequence charts (MSCs) which can be useful for debugging the LE implementation. The PTS tool will be covered in further detail in the next chapter.

16.9 Disclaimer

The examples provided here are only for educational purposes to illustrate the various LE operations. These may not work or may have unpredictable results. So you are advised to use them at your own risk. To make them suitable for commercial needs several enhancements, error checks, and exhaustive testing would be required.

16.10 Summary

The previous chapters explained the various components of the LE architecture including the Link Manager, HCI, L2CAP, ATT, GATT, GAP, and GATT-based profiles. This chapter illustrated how these various components come together for an end-to-end use case like finding lost keys.

Besides the commands and events needed to perform various operations, this chapter also showed the various transactions that are happening 'behind the scenes' on the HCI level and ATT level. This would give a good overview on how the various layers interact with each other. Lastly this chapter also covered some of the tools and techniques that could be useful during development for debugging the LE applications. The next chapter will cover another aspect about developing LE applications, which is the testing and qualification of these applications.

References

[1] Bluetooth Core Specification 4.0 http://www.bluetooth.org.

[2] Bluetooth SIG, Specifications of the Bluetooth System, Profiles http://www.bluetooth.org.

[3] BlueZ website (http://www.bluez.org).

[4] Bluetooth Assigned Numbers for Generic Attribute Profile (http://www.bluetooth.org/Technical/AssignedNumbers/Generic-Attribute-Profile.htm).

[5] Bluetooth Assigned Numbers for GATT-based service UUIDs (http://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx).

[6] Bluetooth Assigned Numbers for Characteristic Descriptions (http://developer.bluetooth.org/gatt/characteristics/Pages/default.aspx).