CHAPTER 5

Getting the Hands Wet

5.1 Introduction

After walking through the concepts of Bluetooth, it's a good time to start doing some practical experiments. This chapter will help you to bring up your own Bluetooth development environment in which you can start writing small scripts and programs. A bit of familiarity with the Linux systems, scripting with the shell and the C programming language is assumed.

This chapter will introduce some of the practical usage of the Bluetooth functionality. This will be extended in further chapters to Bluetooth Low Energy. So it's important that you understand the examples provided here and try out a few of them yourself.

This chapter provides examples based on the BlueZ stack. BlueZ is the "official Linux Bluetooth protocol stack." Support for BlueZ can be found in many Linux distributions. In general it is compatible with any Linux system in the market. It provides support for the core Bluetooth layers and protocols. It is flexible, efficient, and uses a modular implementation. Further details about BlueZ can be found on the BlueZ website (http://www.bluez.org).

5.2 Ingredients

You will need the following:

1. A PC running any flavor of Linux. 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.
2. A Bluetooth Dongle. If you search for Bluetooth dongle, you will find a lot of vendors offering USB based dongles. Any of those should be good. If your PC (or laptop) has an in-built Bluetooth device, then it should also be fine and you don't need an external dongle.
3. Some off-the-shelf devices that support Bluetooth. For example, mobile phone, mono headset, stereo headset, keyboard, mouse, printer. Depending on the devices that you have, you may be able to run different examples provided later in this chapter.

4. Some off-the-shelf devices that support Bluetooth Low Energy. This is the tricky part since only limited LE devices are available in the market as of writing this book. You will find development kits from some of the vendors which could be useful in developing LE applications. For the purpose of examples, a PTS dongle is used as a Bluetooth Low Energy device. This dongle is available for purchase from the Bluetooth SIG website.

5.3 Basic Bluetooth Operations

Before trying the examples provided in this section, you will need to connect the Bluetooth dongle to the PC and boot up Linux. Some of the examples provided here may also need root privileges.

5.3.1 Enabling and Disabling Bluetooth

The first command to try out is hciconfig. This command is used to configure the Bluetooth devices.

To check whether the Bluetooth dongle is properly connected and initialized, run the hciconfig command. It will give output similar to Figure 5.1. If the dongle is properly connected, then this command should show information like:

- BD_ADDR.
- Class of Device.
- Manufacturer name.
- Packet Types supported.
- Number of ACL buffers and buffer size: The screenshot below shows 10 buffers of 310 bytes each.
- Number of SCO buffers and buffer size: The screenshot below shows 8 buffers of 64 bytes each.

```
# hciconfig -a
hci0: Type: BR/EDR  Bus: USB
      BD Address: 00:1B:DC:05:B5:B3  ACL MTU: 310:10  SCO MTU: 64:8
      UP RUNNING PSCAN
      RX bytes:1127 acl:0 sco:0 events:39 errors:0
      TX bytes:655 acl:0 sco:0 commands:38 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 5.1 Hciconfig command.

To close the HCI interface, use the command:

```
hciconfig hci0 down
```

To open and initialize the HCI interface, use the command:

```
hciconfig hci0 up
```

Both these commands need root privileges. In these commands, hci0 indicates the hci interface on which the Bluetooth dongle is attached. It's possible that it's attached on hci1 or some other interface instead of hci0. The parameters of these commands will need to be changed accordingly.

5.3.2 Discovering Devices

There are two commands to discover the devices in the vicinity:

```
hcitool scan
hcitool inq
```

hcitool inq performs a Bluetooth inquiry and reports information like BD_ADDR, Clock offset and Class of Device.

hcitool scan performs the Bluetooth inquiry as well as gets the Bluetooth Device names for all the devices that are found during inquiry.

The output of these two commands is shown in Figure 5.2.

5.3.3 Browsing Services

The following command can be used to browse the services of the remote devices:

```
sdptool browse [bdaddr]
```

The command queries the SDP server on the device specified by the BD_ADDR and shows the list of services supported.

A partial output of this command is shown in Figure 5.3.

```
#hcitool inq
Inquiring ...
      68:ED:43:25:0E:99   clock offset: 0x298e      class: 0x7a020c
      00:17:83:DC:72:E9   clock offset: 0x7596      class: 0x1a0114


# hcitool scan
Scanning ...
      00:17:83:DC:72:E9   WM_nareshg
      68:ED:43:25:0E:99   BlackBerry 8520
```

Figure 5.2 Discovering devices.

Figure 5.3 SDP browsing.

5.4 Real World Application—Café Bluebite

This section explains how to use the Bluetooth commands to implement a simple real world application. It is assumed that the reader is familiar with writing simple shell scripts using bash shell.

Let's say you have been requested by Café Bluebite (fictitious name) to create a Bluetooth based advertisement application for them. They are located in a shopping mall and they want to send the deal of the day to anyone who visits the mall. The deal is to be sent on the person's mobile phone via Bluetooth assuming that the person has his Bluetooth switched on when he enters the Mall.

5.4.1 Requirements Specification

The requirements specification provided by Café Bluebite is as follows:

- Advertise the deal of the day to any user who enters the shopping mall.

- Send this advertisement to the person's mobile phone using Bluetooth.

5.4.2 High Level Design

The high level design of the application is provided in the form of a flow chart in Figure 5.4

Some key points to note in the high level design are as follows:

- Step 1c
  - We switch off the discoverable and connectable mode of our own device. This is because we only want to send advertisements messages and we don't want to receive any messages from the remote devices.
- Step 7
  - We want to send messages to only Mobile phones and Tablets. There is no point in sending messages to headsets, keyboards, etc. because they don't have display capabilities.
- Step 11
  - This example uses an endless loop. In practice it will be needed to terminate this loop based on certain conditions.

5.4.3 Code

Before we start writing the code, let's look at the pre-requisites to run this program.

This program will use OBEX to send the advertisements to the remote device. It's possible that the obexftp software is not installed on your Linux system.

To check this you may run the command:

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

This indicates the obexftp is not installed on your system. To install it, run the following command:

```
# apt-get install obexftp
```

You will need to be online to download the obexftp package. During installation it will ask for a few confirmations and then install obexftp on your system.

Now let's start writing the code for each of the steps mentioned above.

We will write the code as bash scripts for simplicity. You may either use a programming language or scripts in another shell depending on what you are conversant with.

Step 1: Initialize Bluetooth. The following steps assume that the Bluetooth controller is attached to hci0 interface. If it's attached to another interface, then the name of the correct hci device needs to be put here. This step also disables inquiry

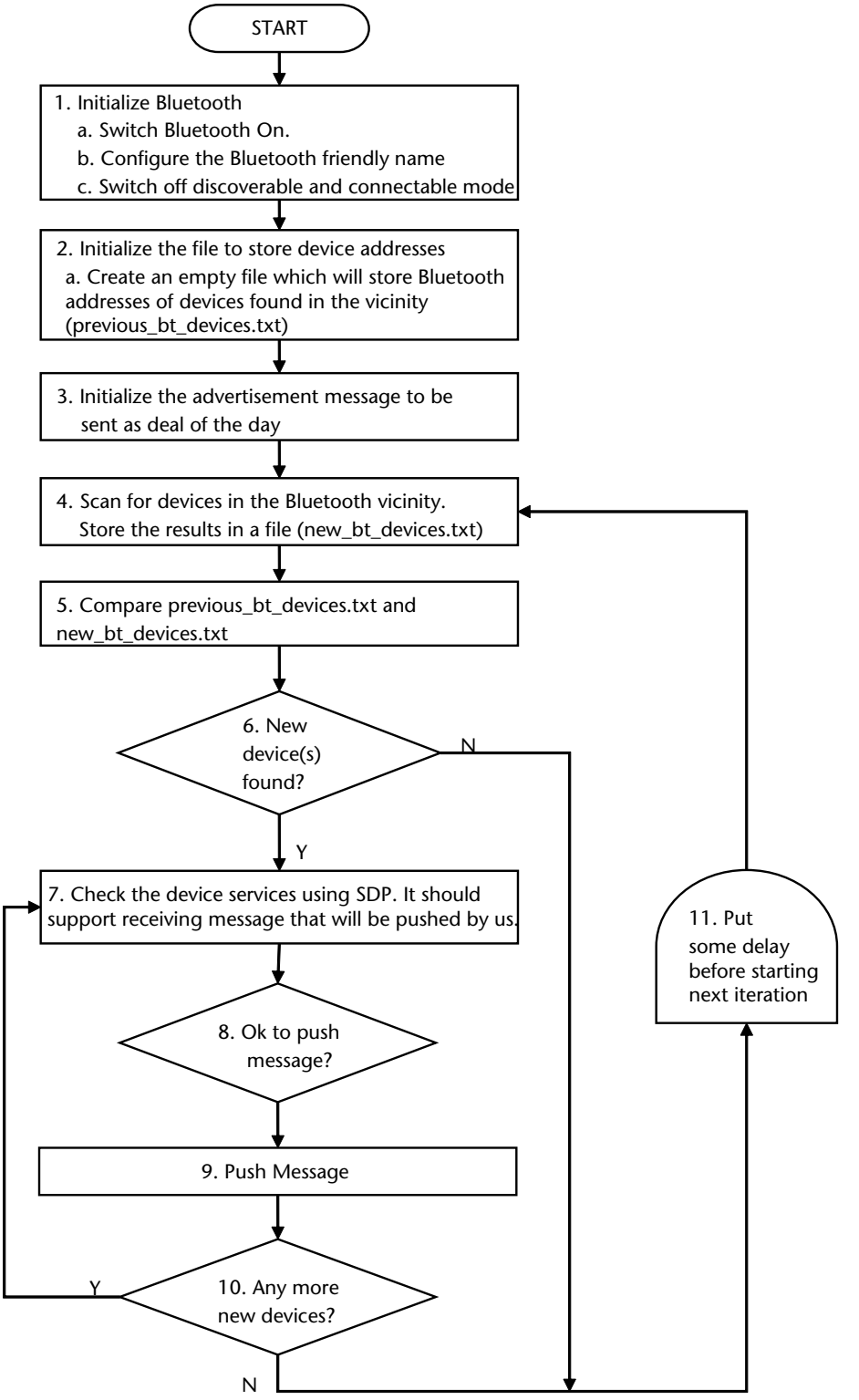

Figure 5.4 High level design—flow chart for Bluebite.

and page scans. After this other devices will not be able to discover our device or connect to our device.

```
echo "Initializing the Bluetooth Controller on hci0..."
hciconfig hci0 up      # Initialize the Bluetooth controller.

echo "Configuring the name to Cafe Bluebite..."
hciconfig hci0 name "Cafe Bluebite"  # Configure the BT name.

echo "Switching off inquiry and page scans..."
hciconfig hci0 noscan     # Disable page and inquiry scan.
```

Step 2: Create an empty file to store device names. This step is quite easy. We can just do a simple "> previous_bt_devices.txt" to create such a file.

```
# Create an empty file to store the previous list of devices
> previous_bt_devices.txt
```

Step 3: Initialize the advertisement message to be sent as deal of the day.

```
# Initialize the advt message to be sent as deal of the day
echo "Welcome to the shopping mall !!" > advt.txt
echo "Visit Cafe Bluebite for exciting deals" >> advt.txt
echo "Get 15% off if you show this message" >> advt.txt
```

Step 4: Scan for devices in the Bluetooth vicinity. Store the results in a file (new_bt_devices.txt). "hcitool inq" performs a Bluetooth inquiry for remote devices in the vicinity and reports the Bluetooth device address, clock offset and Class of Device for each device that is found.

We redirect the output of this command to the file temp_bt_devices.txt. The file temp_bt_devices.txt will contain the list of devices in the following format:

```
#hcitool inq
Inquiring ...
  68:ED:43:25:0E:99  clock offset: 0x298e  class: 0x7a020c
  00:17:83:DC:72:E9  clock offset: 0x7596  class: 0x1a0114
```

We need only the BD_ADDRs out of this list. So the remaining information can be removed. We will use a combination of the following two commands to remove the remaining information:

tail –n +2: To remove the first line.

cut –c2-12: To keep only columns 2 to 18 which contain BD_ADDR.

```
# Perform an inquiry and store the results in a temporary file.
hcitool inq > temp_bt_devices.txt
# Remove extra info from the temporary file
tail -n +2 temp_bt_devices.txt | cut -c2-18 > new_bt_devices.txt
```

If we use these two commands on the shell prompt, the file new_bt_devices.txt will contain the following.

```
68:ED:43:25:0E:99
00:17:83:DC:72:E9
```

This file will be used in the following steps for comparing with the previous list of Bluetooth devices stored in previous_bt_devices.txt.

Step 5: Compare previous_bt_devices.txt and new_bt_devices.txt to find out the devices that have got added.

There are several methods to do this with varying level of complexity and simplicity. In the interest of simplicity we choose a very rudimentary method here.

We read each line of the file new_bt_devices.txt and check if that line was present in previous_bt_devices.txt. This comparison is done using the grep command. The grep command returns 0 if the line is found and 1 if it is not found.

One interesting thing in the code below is that we redirect the output of grep command to /dev/null. This is because the output of the grep command is the list of matching lines. Besides that it also sets the exit status ($?) to 0 if a match was found and 1 if a match was not found. Since we are only interested in the exit status and not the list of matching lines, we let the list of matching lines go to /dev/null.

```
# Run a loop for all lines in new_bt_devices.txt
for line in $(cat new_bt_devices.txt)
do
    # Check if a match is found in previous_bt_devices.txt
    grep $line previous_bt_devices.txt > /dev/null
    # If a match is not found, then it's a new device
    if [ "$?" -eq "1" ]
    then
        echo "New Device found: $line"

  #### Some more code will be put here in Step 7 ####
    fi
done
```

Step 7: Check the remote device services using SDP. For simplicity, let's create a function check_sdp that will do this.

check_sdp function will search the SDP records of the remote device to see if it supports the OBEX Object Push service. This service is referred to by SDP as OPUSH.

It first creates two temporary files temp_sdp.txt and temp_sdp1.txt. It then searches for the OPUSH service on the remote device using the sdptool command. If the OPUSH service is found, it finds the RFCOMM Channel number on which the service is present.

```
function check_sdp ()
{
    # Create a blank file to store the results of SDP search
    > temp_sdp.txt
    > temp_sdp1.txt



    # Do an SDP search for OPUSH service on the BD_ADDR
    sdptool search --bdaddr $1 OPUSH > temp_sdp.txt

    # The output is in temp_sdp.txt
    # if the device has OBEX Object Push service then we must
    # be able to find the string OBEX Object Push in the file
```

```
    grep "OBEX Object Push" temp_sdp.txt > /dev/null
    retval=$?

    if [ $retval -eq 0 ]
    then
        # Check for the Channel number
        grep "Channel: " temp_sdp.txt > temp_sdp1.txt
        retval=$?

        if [ $retval -eq 0 ]
        then

            # Channel number is in the format
            # Channel: 6
            # So extract the field after colon (:)
            channel_num=`cut -d: -f2 temp_sdp1.txt`
            echo OBEX Object Push service found on Channel Num-
ber: $channel_num
        fi
    fi

    # grep returns 0 if found. 1 otherwise.
    # this is the same that our function has to return.
    return $retval
}
```

Step 8: If the check_sdp function returned 0, it means that the device supports OPUSH and we can push the advertisement.

Step 9: Push the advertisement.

```
            # Check if the device supports Object Push
            echo "Checking Services..."
            check_sdp $line

            if [ "$?" -eq "0" ]
            then
                echo "Device supports OPUSH. Pushing advertisement"
                push_advt $line
            else
                echo "Device does not supports OPUSH."
            fi

function push_advt ()
{
    echo executing obexftp --bluetooth $1 -B $channel_num -p
advt.txt
    obexftp --bluetooth $1 -B $channel_num -p advt.txt}
```

Step 10: Repeat this for any more devices found. This is already done by the for loop in Step 5.

Step 11: Put some delay before starting next iteration. Before doing this also copy the temp_bt_devices.txt file to previous_bt_devices.txt so that this file can be used for comparison to find whether any new devices came in the vicinity.

```
cp temp_bt_devices.txt previous_bt_devices.txt
sleep 5s
```

5.4.4 Complete Code

The previous section provided code in pieces. The complete code for Bluebite.sh is provided below.

```
#! /bin/bash

#####################################################################
# function check_sdp
# Input: BD_ADDR of the device for which SDP search is to be done
# Output: 0 if OPUSH record is found. 1 otherwise
# Synopsis: This function does an SDP Search for OBEX Object Push
#           service on the remote device
#####################################################################
function check_sdp ()
{

    # Create a blank file to store the results of SDP search
    > temp_sdp.txt
    > temp_sdp1.txt

    # Do an SDP search for OPUSH service on the BD_ADDR
    sdptool search --bdaddr $1 OPUSH > temp_sdp.txt

    # The output is in temp_sdp.txt
    # if the device has OBEX Object Push service then we must
    # be able to find the string OBEX Object Push in the file
    grep "OBEX Object Push" temp_sdp.txt > /dev/null
    retval=$?

    if [ $retval -eq 0 ]
    then
        # Check for the Channel number
        grep "Channel: " temp_sdp.txt > temp_sdp1.txt
        retval=$?

        if [ $retval -eq 0 ]
        then
          # Channel number is in the format
          # Channel: 6
          # So extract the field after colon (:)
          channel_num=`cut -d: -f2 temp_sdp1.txt`
          echo OBEX Object Push service found on Channel Number:
$channel_num
        fi
    fi

    # grep returns 0 if found. 1 otherwise.
    # this is the same that our function has to return.
    return $retval
```

```
#####################################################################
# function push_advt
# Input: BD_ADDR of the device to which advertisement is to be
#        pushed
# Output: None
# Synopsis: This function pushes the advertisement to the remote
#           device
#####################################################################
function push_advt (){

    echo executing obexftp --bluetooth $1 -B $channel_num -p
advt.txt
    obexftp --bluetooth $1 -B $channel_num -p advt.txt
}

echo "Initializing the Bluetooth Controller on hci0"
hciconfig hci0 up          # Initialize the Bluetooth controller.

echo "Configuring the name to Cafe Bluebite"
hciconfig hci0 name "Cafe Bluebite"  # Configure the BT name.

echo "Disabling Simple Secure Pairing"
hciconfig hci0 sspmode 0

echo "Switching off inquiry and page scans."
hciconfig hci0 noscan            # Disable page and inquiry scan.

# Create an empty file to store the previous list of devices
> previous_bt_devices.txt

# Initialize the advt message to be sent as deal of the day
echo "Welcome to the shopping mall !!" > advt.txt
echo "Visit Cafe Bluebite for exciting deals" >> advt.txt
echo "Get 15% off if you show this message" >> advt.txt

echo "Starting infinite loop. Press Ctrl-C to exit program"

while [ 0 -eq 0 ]
do
    # Perform an inquiry and store the results.
    hcitool inq > temp_bt_devices.txt

    # Remove extra information from the temporary file and
    # store results in new_bt_devices.txt
    tail -n +2 temp_bt_devices.txt | cut -c2-18 > new_bt_devices.
txt

    num_devices=`cat new_bt_devices.txt | wc -l`
    echo Found $num_devices devices in this iteration

    # Run a loop for all lines in new_bt_devices.txt
    for line in $(cat new_bt_devices.txt)
    do
       # Check if a match is found in previous_bt_devices.txt
        grep $line previous_bt_devices.txt > /dev/null

        # If a match is not found, then it's a new device
        if [ "$?" -eq "1" ]
```

```
            then
                echo "New Device found: $line"

                # Check if the device supports Object Push
                echo "Checking Services..."
                check_sdp $line

                if [ "$?" -eq "0" ]
                then
                  echo "Device supports OPUSH. Pushing advertisement"
                  push_advt $line
                else
                  echo "Device does not supports OPUSH."
                fi
            fi
        done
        # Sleep for 5 seconds
        cp temp_bt_devices.txt previous_bt_devices.txt
        sleep 5s
    done # Infinite While loop
```

5.5 Disclaimer

The code provided here is only for educational purposes to illustrate the use of different Bluetooth related commands and features. It has been tested with only a few mobile phones and may not work with all phones or lead to some unknown problems. So you are advised to use it at your own risk. To make the code suitable for commercial use several enhancements, error checks, and exhaustive testing would be needed.

5.6 Summary

This chapter explained the basic requirements of bringing up a setup to try out some Bluetooth operations like enabling and disabling Bluetooth, discovering devices, etc. An example application was also developed to get accustomed to how the various Bluetooth operations can be invoked through simple shell commands.

With this chapter, the background of Bluetooth is completed. The next chapter onwards will focus on Bluetooth Low Energy.

Bibliography

Bluetooth Core Specification 4.0 http://www.bluetooth.org.
Bluetooth SIG, Specifications of the Bluetooth System, Profiles http://www.bluetooth.org.
BlueZ website (http://www.bluez.org).