Using Bluetooth

By Ben DuPont, January 31, 2012

8 Comments

Trying to communicate with a remote device with no other familiar protocol? Bluetooth provides an easy answer with well-documented specs and straightforward programming APIs.

Bluetooth programming can solve several annoying problems outside of its principal domain areas. For example, it works well if you're working on a standalone device and you want to configure the device via a custom Bluetooth profile, or you have a device that speaks a standard protocol that your Linux machine doesn't yet support. In this article, I demonstrate how to code your way out of both problems using Bluetooth.

Linux has a mature and capable Bluetooth stack called BlueZ. Many Linux distributions including Ubuntu 10.10 ship with a Bluetooth-enabled kernel as well as a set of profiles that enable your machine to do things like transfer files to and from a Bluetooth-enabled phone.

There are three main steps to making Bluetooth devices communicate:

- Scan: Scan for devices that advertise certain Bluetooth services
- Pair devices: Exchange a key to authenticate the devices and initiate secure communication
- Connect to a service: After the devices are paired, either end can initiate a connection to a particular service.

We'll use built-in tools to accomplish these tasks, and we'll use the BlueZ API to write a program that will communicate with the device. In particular, we'll implement the Headset profile (HSP), which allows your smart phone to use your computer as a headset. Sample code is provided in C.

Before diving into how to accomplish these tasks, let's cover system setup.

System Setup

The given examples were written on Ubuntu 10.10. You'll need the following packages:

- gnome-Bluetooth, a gnome applet for administering Bluetooth devices
- bluez-hcidump, communication debugging tool
- bluez, Linux Bluetooth stack and associated tools
- libBluetooth3, the BlueZ library
- libBluetooth-dev, the development files for linking to the BlueZ library
- libasound2, ALSA API for using a sound card
- libasound2-dev, ALSA API for using a sound card

Finally, you'll need a Bluetooth phone that supports the Bluetooth HSP (headset) profile and a Bluetooth adapter. I'm using a TRENDnet TBW-106UB USB Bluetooth adapter. After plugging the dongle into your machine, the Bluetooth-applet should start as indicated by the Bluetooth symbol on the top panel. (Second in from the left; see Figure 1.)

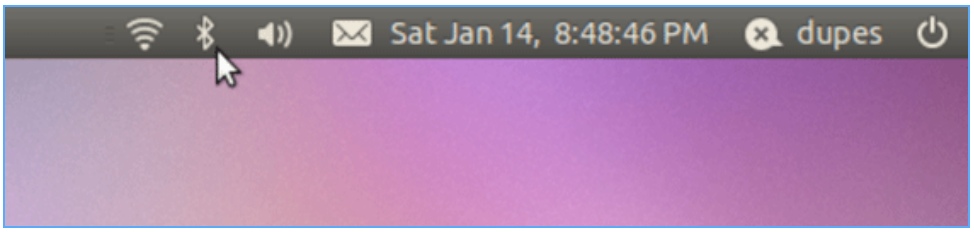

Figure 1.

Using the applet, you can scan for devices, initiate and complete the paring process, and connect Bluetooth services between devices. As the topic is programming, I'm not going to go through the process of scanning and pairing devices, but note that you will have to pair your phone with your machine in order to follow along.

BlueZ API

BlueZ exposes a socket API that's similar to network socket programming. If you're familiar with network programming in Linux, learning the BlueZ API will feel familiar. To implement the headset profile (more on what that means in a bit), we'll use two different sockets: RFCOMM and SCO.

RFCOMM sockets are like TCP sockets: They're useful in situations where reliability trumps latency. SCO sockets provide a two-way stream of audio data at a rate of 64 kb/s. SCO packets are not retransmitted.

Bluetooth Profiles

What does it mean to implement a Bluetooth profile? The standard Bluetooth profiles and core specification are published on the bluetooth.org website. The top of the page lists different version of the core specification. Below the core specification documents are the profiles.

We need two files from the website: Headset profile v1.1 (HSP), and the Core Specifcation Version 2.1 + EDR.

The core specification file tells you everything you need to know to create a Bluetooth stack. Weighing in at just over 1400 pages, it would be a hefty read. Fortunately, we'll only be using this document as a debugging aid. I chose version 2.1 + EDR because that's the same version that the dongle supports.

The HSP document is a much more manageable 27 pages. It shows how to implement the Bluetooth headset profile (HSP). With these documents in hand, let's configure the Bluetooth adapter, pair a smart phone, and finally get to coding.

Configure the Bluetooth Adapter

When scanning, Bluetooth looks for nearby devices that provide specific services. Services offered by a device are identified by the device class. The first thing we need to do is set the class of the adapter. We'll use the hciconfig tool to do this. Later, I'll show how to set the class in the code.

Setting the class requires creating a bitmask from a list of assigned constants. The Bluetooth.org site contains a series of assigned numbers documents. The definitions required to set the appropriate class are in the baseband document. I'm not going to go into detail about how the document formatted, but I'll point you to the relevant pieces for the class that we need to build. Using the second document, I'll construct the class value.

From the major service class, we'll select bits 21 and 19 (audio and capturing), and from the major device class, we'll use audio/video (00100). Because we chose the audio/video device class, scroll down to Table 7 in the baseband document where the minor device classes are listed for audio/video major class. From this table, we want hands-free device (000010). Putting the selected values together, the value of the class in hex is 0x280404.

To see the current configuration of your Bluetooth hardware, run `hciconfig -a`. This command displays various data items about your hardware, most notably, the Bluetooth address and the class. Here's sample output from my machine:

```
hci0:   Type: BR/EDR  Bus: USB
    BD Address: 00:11:22:33:44:55  ACL MTU: 310:10  SCO MTU: 64:8
    UP RUNNING PSCAN ISCAN
    RX bytes:7383 acl:0 sco:0 events:175 errors:0
    TX bytes:3583 acl:0 sco:0 commands:175 errors:0
    Features: 0xff 0xff 0x8f 0xfe 0x9b 0xff 0x59 0x83
    Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
    Link policy: RSWITCH HOLD SNIFF PARK
    Link mode: SLAVE ACCEPT
    Name: 'CSR - bc4'
    Class: 0x280404
    Service Classes: Capturing, Audio
    Device Class: Audio/Video, Device conforms to the Headset profile
    HCI Version: 2.1 (0x4)  Revision: 0x149c
    LMP Version: 2.1 (0x4)  Subversion: 0x149c
    Manufacturer: Cambridge Silicon Radio (10)
```

Under the class line, the output contains a text description of the service classes and device class. My machine shows a class of 0x280404, service classes capturing and audio, and device classes audio/video, and device conforms to the headset profile.

Run the following command to set the class: `sudo hciconfig hc0 class 0x280404`.

After setting the class, run `hciconfig -a` again and verify that the class, service class, and device class lines match my output. If it does, your phone should recognize the machine as supporting the headset profile. You can scan for your computer from your phone and pair them. Of course, we haven't implemented the headset service yet so your phone has nothing to connect to yet.

Bluetooth Addresses and Conversion Functions

Bluetooth addresses are a 6-byte hex number similar to an Ethernet MAC address. BlueZ provides convenience functions for converting between a 6-byte string in the format 00:11:22:33:44:55 and the `bdaddr_t struct`. Here are the function prototypes:

- `int ba2str(const bdaddr_t *ba, char *str);`
- `int str2ba(const char *str, bdaddr_t *ba);`

The function `ba2str` converts from the internal `bdaddr_t` to a zero-terminated string (the `str`parameter should have at least 18 bytes), and `str2ba` provides the opposite conversion. The first example makes use of the `ba2str` function.

Implementing the Headset Profile (HSP)

The HSP profile document describes how to implement the HSP profile. The profile overview (on page 204) contains a diagram that shows two distinct components of the HSP: the audio gateway (AG) and the headset (HS); see Figure 2:

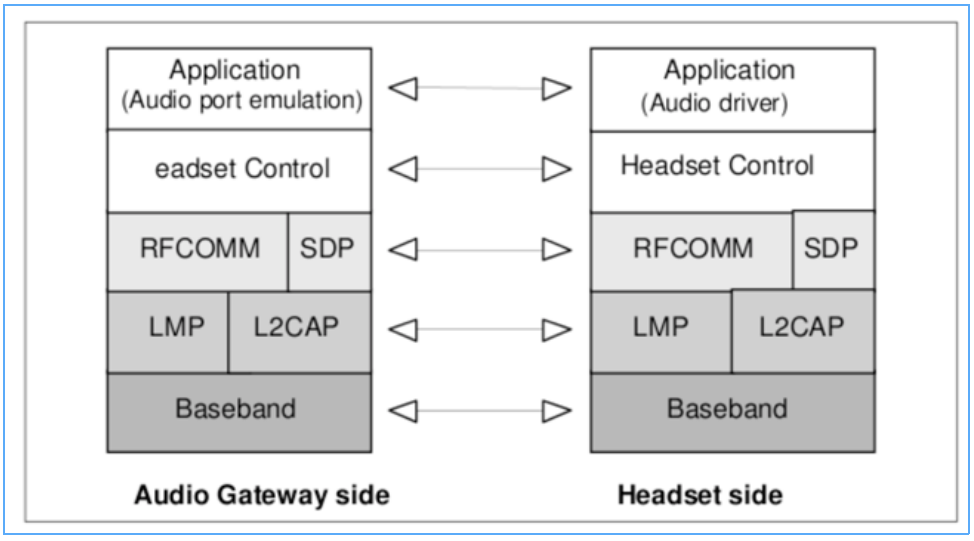

Figure 2: Diagram copied from page 204 of the HSP profile document.

We're going to implement the headset side, (but implementing the audio gateway side is almost identical — the audio gateway can be implemented by making a few small changes to the code).

Here are the steps:

- Set the class (as we did earlier, except through a function call)
- Register with the SDP server
- Listen for RFCOMM connections
- Listen for SCO connection
- Process data from the connections

The following sections provide sample code to accomplish these tasks.

Setting the Class

In a previous example, we used the hciconfig tool to set the device class. In this example, we'll set the class in code with a call to `hci_write_class_of_dev` as shown in Listing One. Modifying the device attributes is a privileged function so this example must be run as root.

Listing One

```
#include "btinclude.h"

int main()
{
    int id;
    int fh;
    bdaddr_t btaddr;
    char pszaddr[18];

    unsigned int cls = 0x280404;
    int timeout = 1000;

    printf("this example should be run as root\n");

// get the device ID
    if ((id = hci_get_route(NULL)) < 0)
        return -1;

// convert the device ID into a 6 byte bluetooth address
    if (hci_devba(id, &btaddr) < 0)
        return -1;

// convert the address into a zero terminated string
    if (ba2str(&btaddr, pszaddr) < 0)
        return -1;

// open a file handle to the HCI
    if ((fh = hci_open_dev(id)) < 0)
        return -1;

// set the class
    if (hci_write_class_of_dev(fh, cls, timeout) != 0)
    {
        perror("hci_write_class ");
        return -1;
    }

// close the file handle
    hci_close_dev(fh);

    printf("set device %s to class: 0x%06x\n", pszaddr, cls);

    return 0;
}
```