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.

Listening for SCO Connections

Once the RFCOMM connection is established, the audio gateway will connect and release an SCO connection as needed. Recall that the SCO connection carries audio data in both directions.

Creating an SCO connection is nearly identical to creating a RFCOMM connection. The main difference is, SCO connections do not specify a channel: to connect to a remote host, only the host address is specified. Of course, SCO sockets also use different constants and structs. Listing Four provides an example:

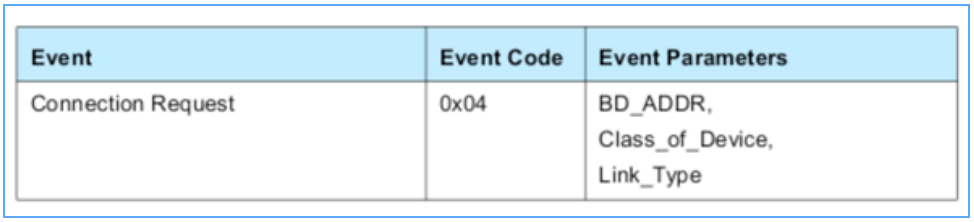
Listing Four

```
#include "btinclude.h"

int main()
{
    int sock;
    int client;
    unsigned int len = sizeof(struct sockaddr_sco);

    struct sockaddr_sco local;
    struct sockaddr_sco remote;

    char pszremote[18];

    sock = socket(PF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_SCO);

    local.sco_family = AF_BLUETOOTH;

    local.sco_bdaddr = *BDADDR_ANY;

    if (bind(sock, (struct sockaddr*)&local,
sizeof(struct sockaddr_sco)) < 0)
            return -1;

    if (listen(sock, 1) < 0)
        return -1;

    client = accept(sock, (struct sockaddr*)&remote, &len);

    ba2str(&remote.sco_bdaddr, pszremote);

    printf("sco received connection from: %s\n", pszremote);

    return 0;
}
```

Following the Signaling Diagrams in the HSP Profile Document

From a BlueZ API standpoint, all of the pieces required to implement the headset profile have been covered. What we haven't discussed is how the headset and audio gateway interact. Interaction between the devices is described in signaling diagrams.

Understanding the signaling diagrams is important in order to know what types of scenarios your program needs to handle. While looking at the diagrams, keep in mind that we're implementing the headset (HS) side. Let's examine the *incoming audio connection establishment* from page 210 of the HSP profile document. Figure 3 is the table from the document:



Event	Event Code	Event Parameters
Connection Request	0x04	BD_ADDR, Class_of_Device, Link_Type

Figure 3.

The signaling diagram can be viewed as a sequence of mandatory and optional events that happen from the top down. The first event is the establishment of an RFCOMM connection, represented by the arrow labeled *Connection establishment.* This connection is initiated by the audio gateway. Note that the RFCOMM connection can also be initiated by the headset side as indicated on page 212 of the HSP document.

Next is a `RING` event, which is generated by an incoming phone call. When the audio gateway receives an incoming phone call, it will send the text `RING` to the headset. Optionally, the headset will establish a SCO connection and send an in-band ring tone. The audio gateway will continually send `RING` to the headset until the headset responds with `AT+CKPD=200`. After receiving this response, the audio gateway responds with `OK` and then initiates the SCO connection if the connection does not yet exist.

Debugging with hcidump

If you plan to do Bluetooth programming in Linux, hcidump is an excellent debugging tool. I'm going to cover the very basics of how to translate the output from hcidump using the Bluetooth core specification document.

One place where hcidump becomes useful is when your Bluetooth device doesn't seem to be connecting to your program. If you have hcidump running and you don't see any output, the device is not even communicating with your Bluetooth adapter.

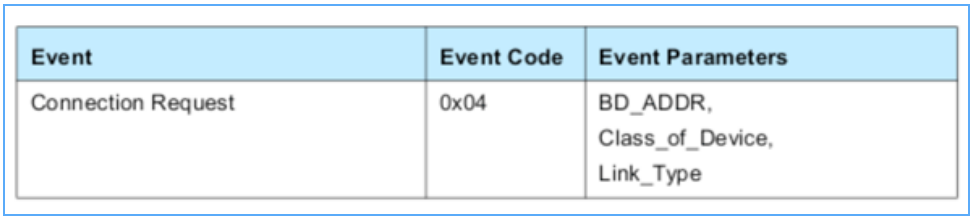
The hcidump program contains a number of options for filtering and logging to files or sockets. We won't filter any events and we'll let the output write to the screen. The only option we'll turn on is the hex and ascii output option. Here's the command: `sudo hcidump -X`.

Here's sample output from my phone connecting to the sample bths program (with modified Bluetooth addresses):

```
1  > HCI Event: Connect Request (0x04) plen 10
2    0000: 00 11 22 33 44 55 0c 02  5a 01                    u.4Q....Z.
3  < HCI Command: Accept Connection Request (0x01|0x0009) plen 7
4    0000: 00 11 22 33 44 55 00                              u.4Q...
5  > HCI Event: Command Status (0x0f) plen 4
6    0000: 00 01 09 04                                       ....
7  > HCI Event: Role Change (0x12) plen 8
8    0000: 00 00 11 22 33 44 55 00                           .u.4Q...
```

To interpret the results, turn to Volume 2 (Core System Package), Part E (Host Controller Interface Specification), 7 (HCI Commands and Events) of Core V2.1 + EDR document (this is the[core specification file] mentioned in the Bluetooth profiles section of this article).

The first line of the sample hcidump output is an event. Events are listed under 7.7. The output tells us the event is a Connect Request (event code 0x04). Connect request is documented in section 7.7.4. Table 1 shows a screenshot of the table.



Event	Event Code	Event Parameters
Connection Request	0x04	BD_ADDR, Class_of_Device, Link_Type

Table 1.

This table tells us the parameters of a connection request are a 6 byte Bluetooth address, a 3 byte class field, and a single byte link type field.

The next line in the sample output is a HCI Command Accept Connection Request which is documented in section 7.1.8.

Putting the Pieces Together

All of the examples are provided in separate files. I've also put all the pieces together in a single sample. The sample includes code that sends audio data from your phone to your computers default sound device. If you run the program, pair and connect your phone, and make a phone call, you should here the audio on your computer speakers. However, the program does not take data from your mic and send it to the phone (through the SCO socket). That's an exercise that I leave to you.

Unpack the samples and run make to compile them. Then follow these steps to run the example:

1. Disable HSP/HFP: add `Disable=Headset,Gateway` to `/etc/Bluetooth/audio.conf`under the `[General]` section
2. Restart Bluetooth: `sudo service Bluetooth restart`
3. Run `bths` as root. If the program isn't run as root, it will fail when trying to set the class:`sudo ./bths`
4. Pair your phone with your machine.
5. From your phone, initiate a connection to your machine by selecting your machine in the Bluetooth settings. Upon successful connection from the phone, the program will output`received connection from 00:11:22:33:44:55`.
6. Initiate a phone call from your phone. After initiating the call, your phone should create a SCO connection to your machine. The bths program will output `sco received connection from 00:11:22:33:44:55`, and all audio that you would normally hear on your phone will play through your machine.

The provided example several limitations. One issue is data from the RFCOMM socket is not processed until a SCO connection is initiated. Once both sockets are connected, they are both processed, but the program should process data from the RFCOMM socket even when there's no active SCO connection. This example won't handle incoming calls to your phone for this reason.

Final Observations

This article provides quite a bit of detail on how to interpret and implement Bluetooth profiles. In the introduction, I also promised that you'd learn how to implement a custom profile for configuring a standalone device. Some of the tools to do so are mentioned above, namely RFCOMM sockets, but there are other options as well. Here are a few:

- Custom protocol over an RFCOMM connection. This would be nearly identical to creating a custom protocol over TCP.
- Linux supports serial port emulation over Bluetooth. This option would allow a wireless serial link between machines, over which a custom serial protocol could be implemented. It's also useful for existing programs that require a serial interface.
- Linux supports networking over Bluetooth, often referred to as a Personal Area Network (PAN). This option creates an IP link between devices, allowing protocols such as HTTP to function over Bluetooth. With such a link, one could run a web server on one device and connect to the device with a browser, as mentioned, all over Bluetooth.

If you own an Android phone, you should know that the Bluetooth radio is exposed to the Java API. This allows you to create custom Bluetooth programs on your phone. Now what are the possibilities?

Ben DuPont is a software engineer located in Green Bay, WI.