USB Bluetooth Guide (C language)

by Globe Tronners, Fall 2015

Formatting and Terms

1. Host (desktop) commands starting with $ are Linux console commands:
   ```
   $ echo "Hi! I am mr.Meeseeks"
   ```
2. Target (board) commands start with #:
   ```
   # echo "Meow on the embedded board"
   ```
3. All the code will be in boxes
   ```
   printf("I am code");
   ```
4. Almost all commands are case sensitive
5. The guide will refer to BeagleBone Black as BBB

Introduction

It has to be noted first of all that the **BBB does not have any Bluetooth hardware built in**, and the cape provided for the course did not have any as well. If you are in the same predicament as we are, you have 2 options for getting Bluetooth connection with the board. First is to get a Bluetooth cape, the choice we did not go with, since we wanted to use the provided cape's hardware. Therefore this guide will not provide any information regarding usage of the Bluetooth cape and there are many variants. The second option is to use a USB Bluetooth dongle, **which must be purchased separately**, and the guide will be focus on that.

Choosing the USB dongle

As the Bluetooth dongle must be purchased separately, and there a lot of options on the market. The dongle we ended up using was a dongle from Kinivo ($35, $15 on sale). We have also tried a couple cheap-o dongle from an unknown manufacturers, and ran into problems of dongles (of the same type) sharing the same MAC address. Beware of odd problems like that, your experience may vary.

Sanity check for USB Bluetooth

Once you have the Bluetooth dongle, make sure it works and will be sufficient for your purposes.

1. Download BlueZ library
   ```
   # apt-get install bluez
   ```
2. Plug in your USB dongle
3. Run the ifconfig but for bluetooth
   ```
   # hciconfig
   ```
   a. If your dongle is detected you should see something like this (not coloured):
   ```
   hci0:   Type: BR/EDR  Bus: USB
           BD Address: 5C:F3:70:6D:D0:CE  ACL MTU: 1021:8  SCO MTU: 64:1
           UP RUNNING PSCAN
           RX bytes:1351 acl:0 sco:0 events:60 errors:0
           TX bytes:1333 acl:0 sco:0 commands:60 errors:0
   ```

`hci0` - is your device
`5C:F3:70:6D:D0:CE` - is the MAC address of your Bluetooth dongle.
`UP RUNNING` - means the device is ON
`PSCAN` - means the device is able to search for other devices around
`ISCAN` - (not shown) means the device is visible to other Bluetooth devices

b. If the dongle is not detected, you will get no output. Check if the dongle is plugged in. If the board is taking too long to boot or showing something like this:

```
[   167.741290] usb 1-1: device not accepting address 2, error -110
```

Try rebooting the board without the dongle plugged in; try using the cold-reset button on the board.

4. By default, Bluetooth is configured for scanning but will not be visible by other Bluetooth devices. To make your device visible to others run:
   a. Get the `hci0` - of your device (0 in my case) by running
      `# hciconfig`
      By default it will show: `UP RUNNING PSCAN`
   b. `# hciconfig hci0 piscan`
   c. Check it worked

```
# hciconfig
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 5C:F3:70:6D:D0:CE  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN ISCAN
        RX bytes:1351 acl:0 sco:0 events:60 errors:0
        TX bytes:1333 acl:0 sco:0 commands:60 errors:0
```

5. If your device is configured for scanning (should be by default): `PSCAN`, to check devices around you, can run:

```
# hcitool scan
      Scanning …
          00:12:6F:3B:4C:72       Music Receiver
          24:0A:64:DE:4A:0E       MyPhone
          C4:9A:02:ED:47:41       MyPC
```

   This will be the stage you make sure your dongle or dongles have appropriate MAC addresses and are working. If your boards with other Bluetooth dongles do not show up in the list, they may be too far, not running, or not made visible (check step 4 above).

Getting Bluetooth Libraries and cross-compiling

1. If using the cross-compiler toolchain for ARM which calls the software floating point ABI, namely *arm-linux-gnueabi-gcc* (rather than *arm-linux-gnueabi**hf**-gcc*), on the BBB, you need to install the soft-float version of the Bluetooth library.

```
# apt-get install libbluetooth3:armel
```

   Note that you must specify *:armel* at the end of the package you wish to install. This gives you the soft-float version. Without it (i.e. simply *libbluetooth3*), this installs the hard-float version of the library since the BBB natively calls the hard-float ABI. Check references at the bottom, for more information about hard-float and soft-float.

The command above should get you both *libbluetooth3:armel* and *libbluetooth-dev:armel*. libbluetooth3 is the library to use the BlueZ Linux Bluetooth stack whereas libbluetooth-dev is the development library for using the BlueZ Linux Bluetooth library. If the development library is not obtained in the process, be sure to install it by running:

```
# apt-get install libbluetooth-dev:armel
```

The newly acquired library will be available in the directory /usr/lib/arm-linux-gnueabi/ under the name libbluetooth.so. It is possible, and highly likely, that there will be multiple files with the name libbluetooth.so and different extensions. Only one is the actual library; the others are symbolic links to the actual library. To discover which file is the library you want, run the commands:

```
# cd /usr/lib/arm-linux-gnueabi/
# ls -la libbluetooth.so*
```

```
root@meow-beagle:/usr/lib/arm-linux-gnueabi# ls -la libbluetooth.so*
lrwxrwxrwx  1  root root     23 Jan  3  2015 libbluetooth.so.3 ->
libbluetooth.so.3.17.11
-rw-r--r--  1  root root 103368 Jan  3  2015 libbluetooth.so.3.17.11
```

Notice that the first file, *libbluetooth.so.3*, has an arrow pointing to *libbluetooth.so.3.17.11*, representing a symbolic link. The file you want is the one without any links (your filename may differ).

2. To cross-compile for the BBB using the Bluetooth library, the host must have access to this library, but as of now, the library is only accessible by the BBB. Therefore, you must create a copy of the library where the host can access it as well.

   (Assuming /mnt/remote/ on BBB is mounted to the shared folder ~/cmpt433/public/ on host)

   ```
   # mkdir /mnt/remote/shared_libraries
   ```
   OR
   ```
   $ mkdir ~/cmpt433/public/shared_libraries
   ```

   ```
   # cp /usr/lib/arm-linux-gnueabi/libbluetooth.so.3.17.11     \
   /mnt/remote/shared_libraries/libbluetooth.so
   ```
   (Change **bolded filename** to whichever file is the actual library found in step 2)

   NOTE: the actual command is one line; the \ indicates the split of the command

3. On the host, you must let the cross-compiler know where to find the Bluetooth library to link against:

-L${HOME}/cmpt433/public/shared_libraries

And you must tell the cross-compiler which library to include:

-lbluetooth

Putting it all together, your command to cross-compile your application should look something like the following:

```
$ arm-linux-gnueabi-gcc -Wall -Werror -g -std=c99
-D_POSIX_C_SOURCE=200809L main.c -o
${HOME}/cmpt433/public/bluetoothApp
-L${HOME}/cmpt433/public/shared_libraries -lbluetooth
```

C Bluetooth programming

We used several BBBs with Bluetooth dongles, so we had a need for a server and several clients. However, the code should still be applicable in cases where you need only one of the, for example communicating with a music receiver or a smartphone, for instance. Bluetooth programming is very similar to socket programming which is convenient. Any module that will need to use bluetooth must include:

```
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/rfcomm.h>
```

The following is slightly modified code for the Bluetooth server that is found on the guide from reference [1].

```
// allocate socket
int socketDesc = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

// binding socket to port 1 of the first available
struct sockaddr_rc server_addr = { 0 };
server_addr.rc_family  = AF_BLUETOOTH;
server_addr.rc_bdaddr  = *BDADDR_ANY;
server_addr.rc_channel = (uint8_t) 1;
bind(socketDesc, (struct sockaddr *)&server_addr, sizeof(server_addr));

// put socket into listening mode
listen(socketDesc, 1);

// accept one connection
struct sockaddr_rc client_addr = { 0 };
socklen_t opt = sizeof(client_addr);
int client = accept(socketDesc, (struct sockaddr *)&client_addr, &opt);

// print the MAC address of the connection received
char mini_buf[256] = { 0 };
```

```
ba2str( &client_addr.rc_bdaddr, mini_buf );
fprintf(stderr, "accepted connection from %s\n", mini_buf);
memset(mini_buf, 0, sizeof(mini_buf));

// read data from the client
char buf[1024] = {0};

// perform a blocking read
int bytes_read = read(client, buf, sizeof(buf));
if( bytes_read > 0 ) {
      printf("received [%s]\n", buf);
}
else {
      // potentially lost connection or error, check error codes
}

// always remember to close the connections
// especially when the connection is lost for some reason
close(client);
close(sock);
```

With the basic server setup and listening you will need a client that will connect:

```
// A total of 32 Bluetooth channels at the time of writing [1..32]
//  2 clients cannot connect on the same channel if a
//  connection is still open on the server
static const short BLUETOOTH_CHANNEL = 13;

struct sockaddr_rc addr = { 0 };

// HARD CODED MAC ADDRESS OF THE SERVER
char dest[18] = "5C:F3:70:6D:D0:CE";

// allocate a socket
int socketDesc = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

// set the connection parameters (who to connect to)
addr.rc_family  = AF_BLUETOOTH;
addr.rc_channel = BLUETOOTH_CHANNEL;
str2ba( dest, &addr.rc_bdaddr );

// connect to server
int status = connect(socketDesc, (struct sockaddr *)&addr, sizeof(addr));

// send a message
if( status == 0 ) {
      char message[] = "Mesa Back! Grlgrlgr!";
      // non-blocking write to socket
      status = write(socketDesc, message, sizeof(message));
}

if( status < 0 ) {
      printf("I cannot connect");
}

// ALWAYS close the connection, even if the connection was declined
// or FACE THE WRATH OF MIGHTY OVERFLOW
close(socketDesc);
```

Note that in code above, a server that has accepted a connection and is trying to read from the socket, the read is blocking.

If you have a need for 2-way communication, the code is going to be slightly more involved. Since it is only socket programming, the socket descriptors can be used for two-way communication as they are. However, naturally since the read() function is blocking, an issue arises when trying to read and write in the thread.

There is a way to make it non-blocking but you will need to code a lot more support for non-blocking reads from the socket.

```
int socketDesc = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
fcntl(socketDesc, F_SETFL, O_NONBLOCK);
```

We take no responsibility of what is going to happen once you add the `fcntl(...)`. It may launch nuclear missiles, make the air acidic, or threaten your family. Use it at your own risk.

Here is the guide [2] http://beej.us/guide/bgnet/output/html/multipage/advanced.html that explained it very well, and has a lot of sample code for you to explore. However, we found that their solution was too cumbersome and we did not have time, so you do not have to do what the guide above describes and can solve the issue with a reader and a writer threads. It will be more more maintenance and code to synchronize them but it has worked well for us. However, if you have time you should probably use the non-blocking version that is described in the guide above.

Useful References

1. **An Introduction to Bluetooth Programming**
   https://people.csail.mit.edu/albert/bluez-intro/x502.html
2. **A great article explaining how to do non-blocking reads/writes**
   http://beej.us/guide/bgnet/output/html/multipage/advanced.html
3. **Another guide for non-blocking I/O, slightly outdated and will need a lot of change**
   http://www.kegel.com/dkftpbench/nonblocking.html

4. **Raspbian - FAQ - Soft-float ABI and Hard-float ABI**
   https://www.raspbian.org/RaspbianFAQ#What_do_you_mean_by_.22soft_float_ABI.22_and_.22hard_float_ABI.22.3F
5. **Raspberry Pi Forums - What is hard float?**
   https://www.raspberrypi.org/forums/viewtopic.php?t=11177&p=123525