QSG124: BT111

LINUX USAGE

QUICK START GUIDE

Thursday, 16 May 2013

Version 1.1

SILICON LABS

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Small updates

TABLE OF CONTENTS

1 Introduction

This guide will walk you through getting started with the BT111 *Bluetooth* Smart Ready HCI module on the Linux operating system, using both classic *Bluetooth* and *Bluetooth* Smart.

BT111 is preconfigured at the factory to provide the Bluetooth HCI over USB interface, so it is very straightforward to set up in Linux.

2 Prerequisites

2.1 Linux Operating System

For example the Ubuntu Linux distribution comes with the *BlueZ* Bluetooth stack pre-installed. *BlueZ* contains all the tools needed for testing the BT111 module in both Classic and Low Energy operation.

Ubuntu Linux can be downloaded from http://www.ubuntu.com/download. You need to download Ubuntu Desktop Disc Image File (.iso) which can be installed on top of a virtual machine.

2.2 BlueZ + GATTtool

The version of BlueZ distributed with Ubuntu as of this writing has a bug in **gatttool**, which prevents the writing of attributes over Bluetooth low energy connection. The bug has been fixed for some time in the BlueZ source code, and for your convenience you can download a compiled binary of **gatttool** at http://techforum.bluegiga.com.

The binary was compiled from the bluez-4.101.tar.gz source tar ball available at http://bluez.org.

Copy the **gatttool** over your current one at **/usr/bin/gatttool**. It may be useful to save a backup of the original.

Verifying BT111 Operation

As mentioned BT111 is preconfigured, so it should be recognized automatically by the Linux operating system.

In order to verify this, do the following steps:

1. Once you have Linux operating system running on your device, simply access the Linux command prompt.
2. Type the command shown below to verify the BT111 Bluetooth Module has been recognized correctly.

```
bgt@bgt:~$ dmesg | tail –n 3
[ 2464.288303] usb 1-2: new full-speed USB device number 3 using ohci_hcd
[ 2466.716561] Bluetooth: Generic Bluetooth USB driver ver 0.6
[ 2466.732997] usbcore: registered new interface driver btusb
```

The three lines in the response indicate that BT111 was recognized as a *Bluetooth* radio.

You can also check which version of BlueZ you have installed by typing ***hcitool***. The first line of output specifies the version. This guide was written with version 4.98, except for **gatttool**, which was from 4.101.

3. The third step is to confirm hcitool can find the *Bluetooth* radio:

```
bgt@bgt:~$ hcitool dev
Devices:
        hci0    00:07:80:12:34:56
```

radios is the BT111 by looking at its Bluetooth address: the Bluegiga range begins with 00:07:80.

3 Testing basic *Bluetooth* functions

3.1 *Bluetooth* classic device discovery

One of the basic operations in *Bluetooth* classic is device discovery. With the BlueZ *Bluetooth* stack you can run device discovery with the following command:

```
bgt@bgt:~$ hcitool --dev=hci0 inquiry --length=5

Inquiring ...
          00:1A:6B:B0:F8:FF     clock offset: 0x21ad  class: 0x6e0100
          00:07:80:ff:ed:fa     clock offset: 0x2140  class: 0x100108
```

3.2 *Bluetooth* low energy scan

With *Bluetooth* low energy technology the device discovery is performed using a scan operation, which can be executed with the following command:

```
bgt@bgt:~$ sudo hcitool --dev=hci0 lescan

LE Scan ...
00:07:80:11:22:33 (unknown)
00:07:80:11:22:33 BGScript - Basic tests
00:07:80:C0:FF:EE Bluegiga Heart Rate Demo
00:07:80:4A:AA:C3 (unknown)
00:07:80:4A:A8:2B Test
00:07:80:4A:A8:2B (unknown)
^C
```

Note:

You must be a super user in order to perform the scan operation.

Ctrl+C can be used to stop the scan operation.

3.3 Checking *Bluetooth* radio details

For detailed information and statistics about the *Bluetooth* radios currently in use, you can use the ***hciconfig*** command.

```
bgt@bgt:~$ hciconfig -a
hci0:       Type: BR/EDR  Bus: USB
            BD Address: 00:07:80:12:34:56  ACL MTU: 310:10  SCO MTU: 64:8
            UP RUNNING PSCAN
            RX bytes:844 acl:0 sco:0 events:35 errors:0
            TX bytes:646 acl:0 sco:0 commands:35 errors:0
            Features: 0xff 0xff 0x8f 0xfe 0xdb 0xff 0x5b 0x87
            Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
            Link policy: RSWITCH HOLD SNIFF PARK
            Link mode: SLAVE ACCEPT
            Name: 'bgt-VirtualBox-0'
            Class: 0x6e0100
            Service Classes: Networking, Rendering, Capturing, Audio, Telephony
            Device Class: Computer, Uncategorized
            HCI Version: 4.0 (0x6)  Revision: 0x22bb
            LMP Version: 4.0 (0x6)  Subversion: 0x22bb
            Manufacturer: Cambridge Silicon Radio (10)
```

4 Testing BT111 under Ubuntu

4.1 *Bluetooth* Classic

The most intuitive way for testing Classic Bluetooth with BT111 is to use it as any user would. Right-click on the Bluetooth icon in the top right corner, and add any Bluetooth device you have, be it a headset, phone, or keyboard.

A reasonable throughput test would be to add an OBEX-capable phone with "***Set up New Device…***", then connect to it with the "***Browse Files on Device…***" button, then send and receive files and check the data rate. Browsing files on a phone is very easy with Ubuntu, as it will mount the phone's files as a file system that can be browsed like an SD card or a USB stick.

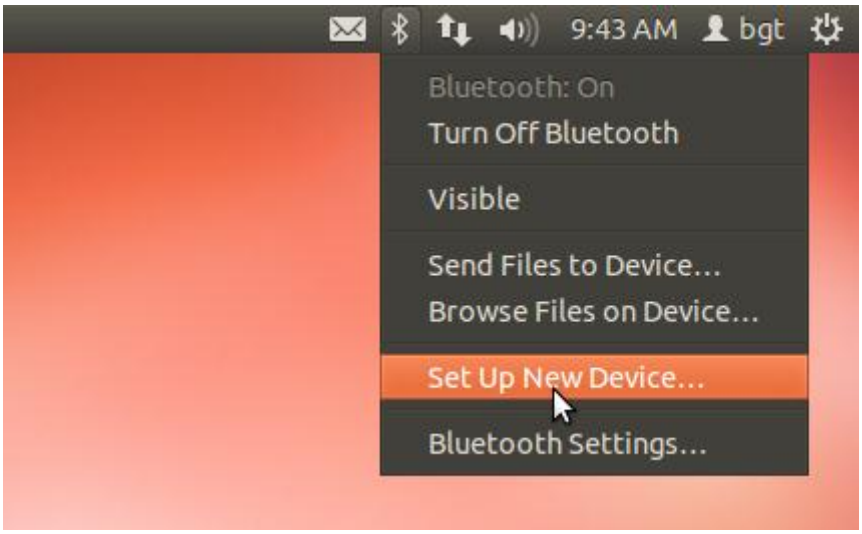


Figure 1: Setting up a new Bluetooth device

4.2 *Bluetooth* Smart

This example will guide you with reading heart rate data from a separate Bluegiga BLE112 *Bluetooth* Smart evaluation board running the example Heart Rate Demo application. The Heart Rate application is done according to the corresponding profile specification, so this example should work with any *Bluetooth* Smart heart rate device.

After you have powered on the BLE112 *Bluetooth* Smart evaluation board and discovered its *Bluetooth* address via ***hcitool lescan***, you can use ***gatttool*** to connect to it.

Let's start with connecting and retrieving the primary service records.

```
bgt@bgt:~$ sudo gatttool -i hci0 -b 00:07:80:C0:FF:EE -I

[   ][00:07:80:C0:FF:EE][LE]> connect
[CON][00:07:80:C0:FF:EE][LE]> primary
[CON][00:07:80:C0:FF:EE][LE]>
attr handle: 0x0001, end grp handle: 0x0005 uuid: 00001800-0000-1000-8000-
00805f9b34fb
attr handle: 0x0006, end grp handle: 0x000c uuid: 0000180a-0000-1000-8000-
00805f9b34fb
attr handle: 0x000d, end grp handle: 0x0011 uuid: 0000180d-0000-1000-8000-
00805f9b34fb
attr handle: 0x0012, end grp handle: 0xffff uuid: 0000e001-0000-1000-8000-
00805f9b34fb
```

Note:

You MUST have discovered the device with an LE scan prior to connecting to it; otherwise ***gatttool*** cannot connect.

The first attribute handle is for the **Generic Access Service** (16-bit shorthand UUID 0x1800, as seen in the last 4 digits of the first part of the 128-bit UUID). The second is for **Device Information Service** (UUID 0x180a). The third is **Heart Rate Service** (0x180d), while the fourth is a proprietary service for testing purposes; it shall be ignored for now.

Let's proceed with reading the device's name. First, we have to retrieve the characteristics of the Generic Access Service. We will see that it has a device name characteristic (UUID 0x2a00) and an appearance characteristic (UUID 0x2a01). Now, we can either retrieve the name with the UUID, which we could have done even without retrieving the GA characteristics, since the device name is mandatory to present; or we can use the characteristic's value handle to retrieve it, as in the example:

```
[CON][00:07:80:C0:FF:EE][LE]> characteristics 0x0001 0x0005
[CON][00:07:80:C0:FF:EE][LE]>
handle: 0x0002, char properties: 0x02, char value handle: 0x0003, uuid:
00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid:
00002a01-0000-1000-8000-00805f9b34fb
[CON][00:07:80:C0:FF:EE][LE]> char-read-hnd 0x0003
[CON][00:07:80:C0:FF:EE][LE]>
Characteristic value/descriptor: 42 6c 75 65 67 69 67 61 20 48 65 61 72 74 20
52 61 74 20 52 61 74 65 20 44 65 6d 6f
[CON][00:07:80:C0:FF:EE][LE]>
```

The returned name is in UTF-8 format, and since all of the bytes have the highest bit (0x80) unset, they can be interpreted as regular ASCII character, as the values 0x00 - 0x7f represent the exact same characters in UTF-8 and ASCII): "**Bluegiga Heart Rate Demo**".

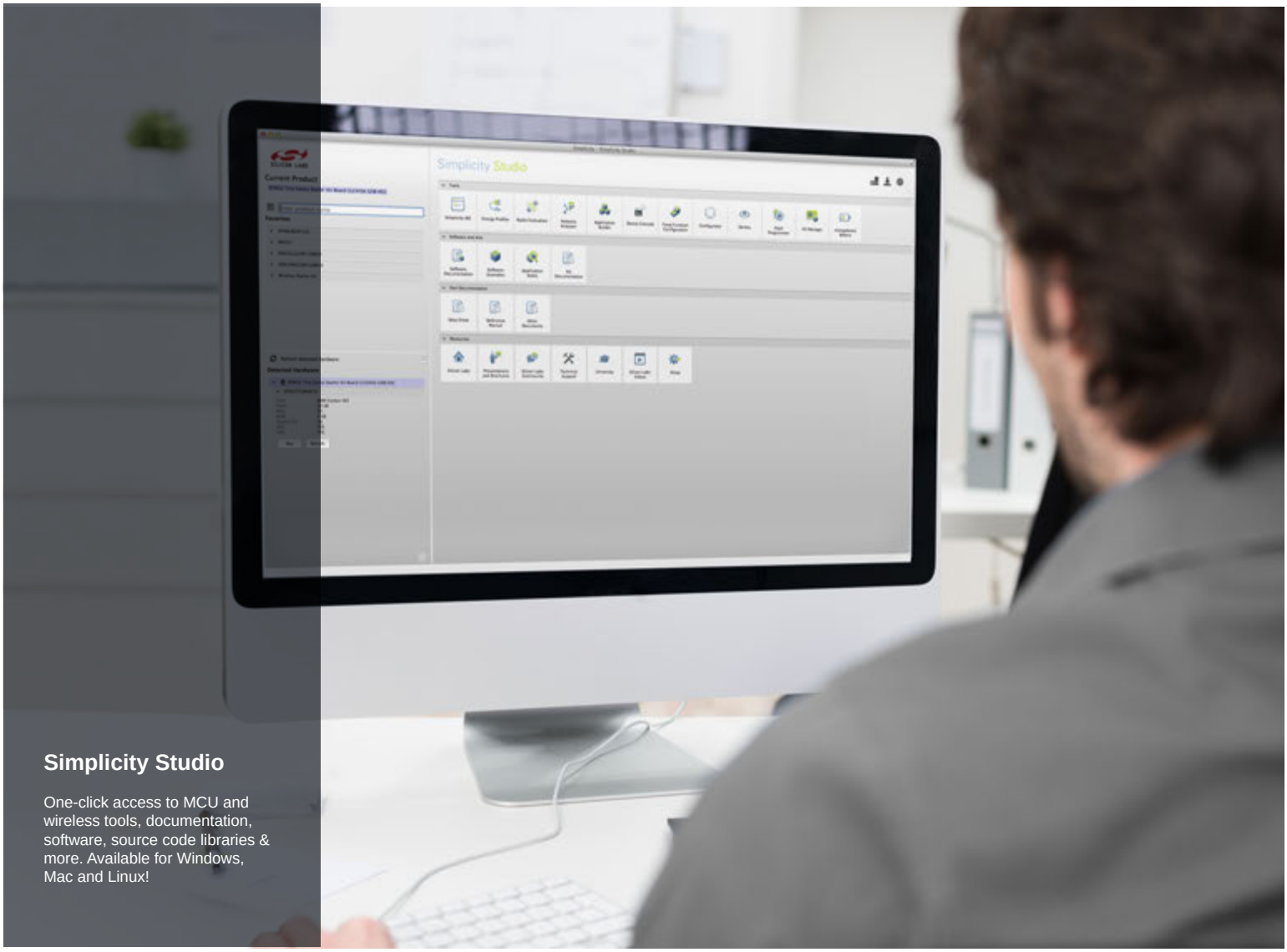
Note:

If you have not downloaded the fixed ***gatttool*** binary at the Bluegiga Tech Forum, or compiled it yourself from a recent BlueZ source, the following example will not work.

Finally we shall send a characteristic value write request to the remote device to set the Heart Rate notifications on, and then after receiving two notifications, we turn them back off.

```
[CON][00:07:80:C0:FF:EE][LE]> characteristics 0x0011
[CON][00:07:80:C0:FF:EE][LE]>
handle: 0x0014, char properties: 0x02, char value handle: 0x0015, uuid:
0000e101-0000-1000-8000-00805f9b34fb
[CON][00:07:80:C0:FF:EE][LE]> char-write-req 0x0011 0x01
[CON][00:07:80:C0:FF:EE][LE]> Characteristic value was written successfully


Notification handle = 0x0010 value: 02 14
Notification handle = 0x0010 value: 02 14
[CON][00:07:80:C0:FF:EE][LE]> char-write-req 0x0011 0x00
[CON][00:07:80:C0:FF:EE][LE]> Characteristic value was written successfully
```



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com