MAKING THE FIRST CONNECTION WITH THE BL620

Application Note *v1.0*

INTRODUCTION

This document provides a comprehensive guide to your first connection from the BL620 central mode device to a BL600 BLE peripheral which is exposing a heart rate service. It will be expedited using a couple of sample *smart*BASIC applications.

REQUIREMENTS

The following equipment and utilities are required:

- One of the following configurations:
  - BL620-US Adapter (firmware 12.4.10.0 or newer) AND BL600 devkit (firmware 1.5.70.0 or newer)
  - 2 x BL600 devkit (firmware 1.5.70.0 or newer in one) AND BL620 Firmware Zip file (12.4.10.0 or newer)
- Windows PC
- UwTerminal (available on the software downloads tab of the BL620 product page at lairdtech.com)
- Mini USB Cable (included with BL600 Series development kit)
- SEGGER J-Link LITE debugger (included with BL600 Series development kits)
- 10-way ribbon cable (included with the SEGGER J-Link LITE debugger)
- *smart*BASIC application "cmd.central.bl620.sb" (in the *smart*BASIC folder of the BL620 firmware zip file)
- *smart*BASIC application "hrs.heartrate.bl600.sb" (in the *smart*BASIC folder of the BL620 firmware zip file)

Note: If you do not also have a BL600 development kit to follow this step-by-step guide then you can purchase one by accessing the following Digikey, Mouser or Farnell distributor websites:
http://www.digikey.co.uk/product-detail/en/DVK-BL600-SA/DVK-BL600-SA-ND/3995739

http://uk.mouser.com/ProductDetail/Laird-Technologies-Wireless-M2M/DVK-BL600-SA/?qs=sGAEpiMZZMu%252b6ulL5WffptndcUOx9D%252bg

http://uk.farnell.com/laird-technologies/dvk-bl600-sa/bl600-sa-class2-bluetooth-dev-kit/dp/2321472

ASSUMPTIONS

It is assumed you are familiar with interacting with a Laird BL600 or BL620 module and AT commands using the UwTerminal terminal emulation utility, available for free from Laird.

It is assumed you are familiar with downloading a *smart*BASIC application into a device using the UwTerminal utility.

PREPARATION

To prepare your modules to connect for the first time, complete the steps in the following sections.

Verify Firmware Versions

If you have a BL620-US USB adapter and a BL600 development kit, ensure that the BL620-US is running firmware version 12.4.10.0 or newer and the BL600 development kit is running firmware version 1.5.70.0 or newer. Using exactly those versions will ensure an exact replication of the responses in this step by step guide.

If you have two BL600 devkits, ensure one has BL620 firmware which is version 12.4.10.0 or newer and the other has BL600 firmware v1.5.70.0 or newer. Again using exactly those versions will ensure an exact replication of the responses in this step by step guide.

The command AT I 3 terminated by a carriage return will result in the module responding with its firmware version number.

For information on upgrading module firmware, see the Firmware Upgrade Application Note available on the documentation tab of the BL620 product page at lairdtech.com.

Download *smart*BASIC Applications

Download the correct *smart*BASIC scripts to the modules. On the BL620 device, download the *smart*BASIC sample application called "**cmd.central.bl620.sb.**" On the BL600, download the *smart*BASIC sample application called "**hrs.heartrate.bl600.sb.**"

First Time Connection

This section describes a step by step guide to performing various actions like scanning for adverts, connecting and performing GATT Client operations such as discovery, read and writes.

Preparation

To begin, first start from a known reset state. To do so, close any UwTerminal applications currently running and power cycle the BL620 and BL600 devices. You may unplug and reconnect the modules from the PC's USB port to achieve this state.

Launch two instances of UwTerminal and connect to the appropriate COM ports for the BL620 and the BL600.

Note: Active COM ports may be found in the Windows Device Manager.

In this example, the UwTerminal window connected to the BL620 is positioned on the left side of the screen and the UwTerminal window connected to the BL600 is positioned on the right side of the screen. When a connection is made from the BL620 to the BL600, it may be thought of as happening from left to right.

Note: Subsequent sections of this guide make reference to the left (BL620) and right (BL600) screens.

Press *enter* in both UwTerminal screens and confirm that you see the 00 response. The 00 response confirms that the modules are connected. If you do not receive the 00 response, close UwTerminal, power cycle the device and reconnect. If it is still does not respond with 00 then it is possible you have opened UwTerminal on an incorrect COM port. Do not proceed until both modules are confirmed to be connected.

On the left screen (BL620 device) enter the command **AT I 0** and ensure you see the following response:

```
10 0 BL620
00
```

Then enter the command **AT I 3** to check the firmware version and ensure you see the following response (you may have newer firmware than the version shown):

```
10 3 12.4.10.0
00
```

On the right screen (BL600 device) enter the command **AT I 0** and ensure you see the following response:

```
10 0 BL600r2
00
```

Then enter the command **AT I 3** to check the firmware version and ensure you see the following response (you may have newer firmware than the version shown):

```
10 3 1.5.70.0
00
```

Then enter the command **AT I 4** and ensure you see the following response:

```
10 4 00 0016A40B1620
00
```

In this example, **00 0016A40B1620** is the mac address (including the leading 00). Make a note of that address, which begins either with a leading 00 or 01 (00 if you have modified the address using the AT+MAC command, as is the case for this module) and the rest of the 12 hex digits. The noted address will be a 14 hex digit string without any spaces and you will be using that string in this guided tour whenever you see the text <<14HexDigitAddr>>.

Note: In this guide, text surrounded in carats such as <<14HexDigitAddr>> is used as a variable, to suggest that you must fill in your own specific values.

Starting Applications

Then finally start the applications in both devices by entering the command **cmd** in the left screen and the command **hrs** in the right screen.

On the left screen confirm that you see the following response:

```
LAIRD BL620
OK
>
```

On the right screen confirm that you see the following response:

```
Start Adverts 0
LAIRD_HRM
OK
>
```

At this stage if you have a smartphone and are familiar with BLE apps that allow you to scan for devices, then you will see that the BL600 is advertising. It is worth doing that as a sanity check if you have used such apps before.

Scan for Adverts

This section shows how to scan for all devices that are advertising. We will scan for 1 second.

On the left screen, enter the command **scan start 1000 0 0** and you will see received adverts displayed as follows:

```
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
Scanning stopped via timeout, advert count = 0
```

Each line is an advertisement. In this example, the same device with address 000016A40B1620 has been picked up 5 times. The trailing *advert count = 0* does not count the adverts themselves, but the max that was requested in the scan command (the last 0 in the command **scan start 1000 0 0**).

The address will be the same as the one noted for the BL600 in the previous startup section. If you do not see the same address, it is very possible that you have many other BLE devices advertising in the vicinity.

The last value of each line, RS:-43, is the RSSI value. The smaller this number is, the further away the remote device is. As general rule, Laird has found that the BL600 module that reports an RSSI value of -60 is roughly 1 meter away from the scanning device. In this case -43 means a stronger signal. In reality, the devices are about 10 cm apart.

To limit the number of adverts displayed to just 3, on the left screen enter command **scan start 1000 0 3** and you will see 3 received adverts displayed as follows:

```
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
ADV:000016A40B1620 AD:020106031940030A094C414952445F48524D07030A180D180F18 XX:0 RS:-43
Scanning stopped via timeout, advert count = 3
```

The BL620 has limited memory and so the memory used for scanning has to be released, so enter the command **scan stop** and confirm you get the OK response.

To prepare for GATT Client operations while in a connection, enter the command **gattc open 0 0** and confirm you get an OK response.

Establish the Connection

This section shows how to connect to the BL600 device so that you can interact with its GATT table. You will need the Bluetooth address of the BL600 that was noted above.

On the left screen , enter the command **connect <<14HexDigitAddr>> 4000 90000 120000 2000000** where <<14HexDigitAddr>> is replaced by your noted 14-digit address, e.g: **connect 000016A40B1620 4000 90000 120000 2000000 .**

The connection is established. On the left screen you will see the following:

```
--- Connect: (0001FF00) handle=1
Conn Interval 120000
Conn Supervision Timeout 2000000
Conn Slave Latency 0
```

On the right screen you will see the following:

```
--- Connect : 3731
Conn Interval 120000
Conn Supervision Timeout 2000000
Conn Slave Latency 0
 --- Hrs Notification : 0
```

You may see a different number, if you are running a later firmware version.

Note: The line *handle=1* means that from the BL620 end this connection may be referred to with the handle "1."

At the BL600 end, the *Hrs Notification* line is saying that the Heart Rate service confirms that notifications are disabled.

Extract the GATT Table Map

This section shows how to obtain the GATT table map so that you can interact with individual characteristics, and provides a general overview of the table in the peripheral.

The GATT Client at the BL620 has already been opened (the command **gattc open 0 0** in the steps above). Submit the command **gattc tablemap 1** and you will see the remote GATT table displayed as below.

Note: If the command **gattc tablemap 1** returns `ERROR 00006059`, the GATT client manager has not been opened. If so, submit **gattc open 0 0.** If this command returns `ERROR 00000201` it means the module's available RAM is running low, most likely because the advert scan manager is using that memory. To resolve this, submit the command **scan stop** and then **gattc open 0 0** again.

The GATT table appears as follows:

```
S:1 ,(7) ,FE011800
 C:3 ,0000000A ,FE012A00 ,0
 C:5 ,00000002 ,FE012A01 ,0
 C:7 ,00000002 ,FE012A04 ,0
S:8 ,(11) ,FE011801
 C:10 ,00000020 ,FE012A05 ,0
 D:11 ,FE012902
S:12 ,(30) ,FE01180A
 C:14 ,00000002 ,FE012A29 ,0
 C:16 ,00000002 ,FE012A24 ,0
 C:18 ,00000002 ,FE012A25 ,0
 C:20 ,00000002 ,FE012A27 ,0
 C:22 ,00000002 ,FE012A26 ,0
 C:24 ,00000002 ,FE012A28 ,0
 C:26 ,00000002 ,FE012A23 ,0
 C:28 ,00000002 ,FE012A2A ,0
 C:30 ,00000002 ,FE012A50 ,0
S:31 ,(36) ,FE01180D
 C:33 ,00000010 ,FE012A37 ,0
 D:34 ,FE012902
 C:36 ,00000002 ,FE012A38 ,0
S:37 ,(65535) ,FE01180F
 C:39 ,00000002 ,FE012A19 ,0
```

Each line starting with **S:n** is a start of a service definition, where **n** is the starting handle and the number that follows is the end handle value. The last hex value is the UUID handle for that service.

Each line starting with **C:n** is the definition of a characteristic, where **n** is the handle value for it, the next number is the characteristic properties (see *smart*BASIC User Guide for details), the third number is the UUID for that characteristic. The last value (0 in the example above is a uuid handle and will be nonzero if the characteristic is part of an included secondary service)

Each line starting with **D:n** is the definition of a descriptor where **n** is the handle value and the last number is the UUID handle for he descriptor. There is one instance above, which reads `D:34 , FE012902`. The 2902 is the UUID for a CCCD descriptor for the heart rate service.

UUIDs in *smart*BASIC Explained

In the Bluetooth Low Energy specification up to v4.0, all Service, Characteristic and Descriptor types are either 16-bit (2 bytes) or 128-bit (16 byes). In specification 4.1 and newer they can also be 32-bit (4 bytes). The non 128-bit values are just offsets from a known published 128-bit base.

In *smart*BASIC there are 2 types of variables (using DIM statement) : INTEGER and STRING. The former are 4 byte entities and the later can be arbitrary length. This means 16bit and 32bit UUIDs can be manipulated using INTEGERs and 128bit UUIDs have to be defined using STRING variables.

Given *smart*BASIC does not offer polymorphism (as in object oriented languages where the same function or method name can accept different variable types) all *smart*BASIC functions that take a UUID as a parameter are designed to take a UUID handle which is always a 32-bit (4 byte) entity. Conversely it has been arranged that when GATT table responses arrive any 16-bit or 128-bit UUIDs are first internally converted to a 32-bit UUID and then presented (there are only a few exceptions when this does not happen and this will be clearly obvious). For example, in the **tablemap** example above, virtually all lines have a hex value starting with FE01, for example the characteristic beginning with C:39 has the value FE012A19.

For this firmware release, a handle starting with FE01 means the handle is for an adopted UUID that is 16-bit (2 bytes). For example, in FE012A19 the 16-bit UUID is 2A19 and according to the Bluetooth specification that is allocated for Battery Level Characteristic.

Note that a 16-bit UUID as published by the Bluetooth SIG is just an offset into a predefined 128-bit base UUID which has the value `00000000-0000-1000-8000-00805F9B34FB`.

If you encounter FF00, then means a 128-bit UUID with an unknown base has been encountered and the lower 16-bit of that handle is the offset into that unknown base. **At that point the underlying stack momentarily knows the actual 128-bit UUID but it only provides a 4 byte handle**.

To enable the underlying stack to provide you with a handle that does not have the unknown FF00 'prefix', there is mechanism to register 128-bit base UUID values. In *smart*BASIC this is done using the function **BleHandleUuid128()** which takes a 16 byte string and returns the handle.

Given that a typical GATT client is only interested in services and characteristics that are defined to it, and WILL ignore all that are not, the process of interacting with a GATT server with custom services and characteristic is to first register the 28-bit UUIDs using **BleHandleUuid128()** which will enable the underlying stack to use that base to provide you with appropriate UUID handles.

Find Specific Services

In most GATT client-to-server interactions, the client does not require knowledge of the map of the GATT server (as described above) but only attempts to locate a specific service identified by a UUID.

This section describes how to find specific services using UUID handles. This means the starting point will either be a 16-bit or a 128-bit UUID which has to be first converted into a UUID handle. The sample application enables that conversion and also makes it easier by storing the handle in an integer array and then returning an index value in the range 1 to N. For the purposes of this application note, there is no need to remember and type the actual 8 hex digit number.

For this example, assume that the client is attempting to locate the Heart Rate Service, for which the Bluetooth SIG has allocated the 16-bit UUID value 0x180D. This means the starting point for this exercise is the value 0x180D.

First convert 0x180D into a handle and associate it with the index value 1 to be used in successive operations. To do so submit the command **uuid new 1 180D**. It will respond with 'OK.'

Now '1' can be used in commands requiring UUID handles to mean the 16-bit UUID corresponding to 180D (until you associate a new handle for that index).

Note: Recall that when the connection was established a connection handle of '1' was provided. The '1' is an artefact of this sample application. As with the UUID example above, the actual connection handle is a 32-bit integer. The 'contrived' index values 1..N are just for human convenience in this particular sample application. A developer is free to invent other schemes to manage the 32-bit integer handles that are the base currency for connections and UUIDs. It is expected that for applications where human interaction is not required there will be no need to cache handles in INTEGER arrays.

To locate the service 0x180D, submit the command **gattc svc first 1 0 1** where the first 1 is the connection handle index. The last 1 is the UUID handle index obtained using the command **uuid new 1 180D.**

The command will return the following response:

```
EVDISCPRIMSVC(hConn=0001FF00,hUuid=FE01180D,hStart=31,hEnd=36)
```

The confirmation **hUuid=FE01180D** shows that a service with UUID 0x180D has been located. The confirmation **hStart=31,hEnd=36** means that the service starts at GATT table attribute handle 31 and ends at GATT table attribute handle 36.

It is possible that a GATT table contains multiple instance of a service and it is possible to iterate through them all. To look for more instances of the service with UUID 0x180D, enter the command **gattc svc next 1**.

The command will return the following response:

```
EVDISCPRIMSVC(hConn=0001FF00,hUuid=00000000,hStart=0,hEnd=0)
```

Note that the start and end handles are 0, and the UUID handle is also 0. This implies that another instance of the service was not found.

The BL600 does not have a service with UUID of say 0xABCD, so create a handle in index 2 for it using the command **uuid new 2 ABCD**. Then, search for it using the command **gattc svc first 1 0 2**. You will see the following response:

```
EVDISCPRIMSVC(hConn=0001FF00,hUuid=00000000,hStart=0,hEnd=0)
```

Note that the start and end handles are 0 and the UUID handle is also 0. This implies the service was not found.

You can perform a wildcard search for any service by specifying a 0 as the index for the UUID handle (in the sample application index 0 is populated with the value -1 which is an invalid UUID handle) by entering the command **gattc svc first 1 0 0.**

You will see the following response:

```
EVDISCPRIMSVC(hConn=0001FF00,hUuid=FE011800,hStart=1,hEnd=7)
```

Compare that with what was received for the tablemap command above. The information matches with *S:1 ,(7) ,FE011800.*

Get the next service by entering the command **gattc svc next 1**. You should see the following response:

```
EVDISCPRIMSVC(hConn=0001FF00,hUuid=FE011801,hStart=8,hEnd=11)
```

This again matches the next service in the tablemap example above, *S:8 ,(11) ,FE011801.*

You may submit **gattc svc next 1** repeatedly to iterate through the GATT table until the entire table is traversed. The command returns 'ERROR' when you reach the end of the table.

Find Specific Characteristic in a Service

Once you have obtained the start and end attribute handles for a service (as we did with the *gattc svc first* command) you can iterate through all characteristic (or look for specific ones identified by a UUID handle).

For example, from the tablemap example above, the heart rate service starts at handle 31 and ends at handle 36 (*S:31 ,(36) ,FE01180D*)

Find the first characteristic in that service using the command **gattc char first 1 0 31 36** which returns the following:

```
EVDISCCHAR(hConn=0001FF00,hCharUuid=FE012A37,hIncUuid=0,hVal=33,Props=00000010)
```

This means a characteristic with 16-bit UUID 0x2A37 has been located and the value for that is in attribute handle 33 and that characteristic has properties 0x00000010.

Use the command **gattc char next 1** to locate the next characteristic and repeat until there are no more.

Find Specific Descriptor in a Characteristic

Once you have obtained the attribute handle for a characteristic (as we did with the *gattc char first* command) you can iterate through all descriptors (or look for specific ones identified by a UUID handle) belonging to that characteristic.

For example, the heart rate characteristic at handle 33 has a CCCD descriptor, locate the details for it by submitting the command **gattc desc first 1 0 33** which returns the following:

```
EVDISCDESC(hConn=0001FF00,hDescUuid=FE012902,hDesc=34)
```

This means a descriptor with 16-bit UUID 0x2902 has been located and the value for that is in attribute handle 34.

You can look for the next descriptor in that characteristic by submitting the command **gattc desc next 1.**

Find a Specific Characteristic in Entire GATT Table

In most use cases a client will want to quickly locate a specific instance of a characteristic without having to first look for the service it is contained in and then look for that characteristic in that service. This section shows how to do this, as well as how to look for the heart rate characteristic with 16-bit UUID 0x2A37 in the heart rate service with 16-bit UUID 0x180D.

First create the UUID handle for the service 0x180D in handle index 1, and then in index 2 create the UUID handle for the characteristic uuid 0x2A37. To do so, submit the following 2 commands:
uuid new 1 180D
uuid new 2 2A37

Then to locate for the first instance of the characteristic 0x2A37 in the first instance of the service 0x180D submit the command **gattc findchar 1 1 0 2 0**. The two zeroes specify the first instance, given that the indexing starts at 0. You should see the following response:

```
EVFINDCHAR(hConn=0001FF00,hIncUuid=00000000,hVal=33,Props=00000010)
```

This means the characteristic was found at attribute handle 22 and has the property 0x00000010, which matches that found earlier.

You could look for the first instance of the same characteristic in the second instance of the service (which does not exist) by submitting the command **gattc findchar 1 2 1 2 0**. You should see the following response:

```
EVFINDCHAR(hConn=0001FF00,hIncUuid=00000000,hVal=0,Props=00000000)
```

The hVal is 0, which means the second instance was not found.

Find a Specific Descriptor in Entire GATT Table

In many use cases a client will be interested in say a heart rate service and want notifications to be enabled by writing 0x0001 to the CCCD descriptor and wants to do that with minimal interaction. In this case all it wants is to locate the first instance of a CCCD in the first instance of a heart rate characteristic inside the first instance of a heart rate service.

This can done by first creating UUID handles in indices 1,2 and 3 as follows:

uuid new 1 180D
uuid new 2 2A37
uuid new 3 2902

Then, submit the command **gattc finddesc 1 1 0 2 0 3 0**. You should see the following response:

```
EVFINDDESC(hConn=0001FF00,hDesc=34)
```

This shows that the attribute handle for the descriptor is 34, which matches the information extracted using the tablemap command *D:34 ,FE012902*.

With the handle 34 one can quickly enable notifications by writing to that descriptor using commands as described in the next section.

Enabling Notifications from Heart Rate Service using a Write command

This section will show notifications for heart rate and sending of sample heart rate data from the BL600 device and will involve a write operation.

From the tablemap response above (or the **gattc finddesc** command above), we know that the CCCD has the handle 34. We need to write to it so that notifications are enabled, which means we need to use the GATT client write operation.

To enable notifications, we need to write 0x0001 to the CCCD. This is the attribute with handle 34. To do this, enter on the left screen (the BL620 window) the command **gattc write 1 34 0100.**

Note: The value is written in little endian format, so the 16-bit value 0x0001 is sent as a 2 byte array in little endian format 0100.

You should see the following response:

```
EVATTRWRITE(hConn=0001FF00,handle=34,status=0)
```

On the right screen (the BL600 window), you should see the following response:

```
--- Hrs Notification : 1
```

This confirms that heart rate measurements will now be notified when it is modified.

To notify a heart rate of 72 (which is hex 0x48), on the right screen enter the command **hr 72** and then the command **send**.

You should see the following notification on the left screen:

```
EVATTRNOTIFY()
 >BleGattcNotifyRead(hConn=0001FF00,handle=33,Dumped=0,data=0648)
```

In the line *data=0648,* the 48 corresponds to the heart rate of 72 that was sent. You may wonder why the data came through as 0648. This is because heart rate is a Bluetooth SIG adopted service and that is how the data format has been defined. For more details of the heart rate measurement characteristic see https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.heart_rate_measurement.xml.

Reading BL600 GATT Server Attributes

This section explains how to read attributes from the GATT server in the BL600.

The Heart Rate application has installed a battery service in addition to the heart rate service. The Bluetooth SIG has defined a UUID of 0x2A19 for battery characteristic. Remember that In the table map, the last line is *C:39 ,00000002 ,FE012A19 ,0* which means that the handle to read to get the battery level is 39 given that the UUIS is FE012**A19**.

In the BL600 window (the right screen), set the reported battery level to 50% (0x32) by entering the command **bl 50**. Then on the left screen, to read the battery level, enter the command **gattc read 1 39 0.**

In the BL620 window (the left screen) you should see the following response:

```
EVATTRREAD(hConn=196352,handle=39,status=0)
```

```
>BleGattcReadData(data=32,offset=0)
          (data=2)
```

The line *data=32* confirms the battery level data has arrived. Try other values to see it change.

The second line with *(data=2)* is just the data redisplayed as printable string, and the ASCII value 0x32 is the character '2'.

Disconnect

This section shows how to drop the connection. This can be done from either end. In this example, we'll disconnect from the BL620 end (the left window).

On the left screen enter the command **disconnect 1** and confirm that on the left screen you see the following response:

```
--- Disconnect: (0001FF00) handle=1 reason=22
```

On the right screen (at the BL600 end) you should see the following response:

```
--- Disconnect : 3731
```

The number 3731 in this case is the connection handle that was displayed above on the right screen when the connection was established.

Conclusion

Refer to the source for the *smart*BASIC application "cmd.central.bl620.sb" for the full list of all the commands that have been exposed by that application and which *smart*BASIC functions they invoke. For example, pairing can be initiated using the 'pair' group of commands.

In the source code, it's helpful to search for #CMD# to locate all the commands that are recognised.

Finally, feel free to enhance and modify the application as you see fit.

Revision History

Revision	Date	Description	Approved By
1.0	29 June 2015	Initial Release	Jonathan Kaye