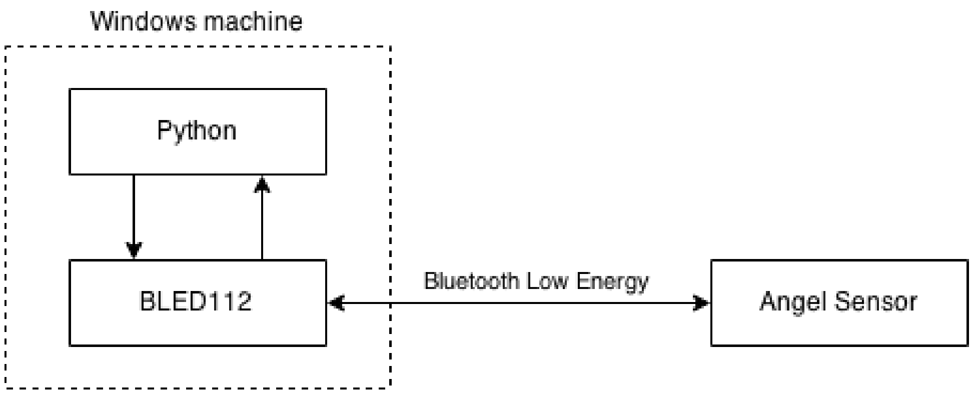
Experiment 01 - Heart Rate Notifications With Python And BLE Dongle

Learn how to grab heart rate notifications from Angel Sensor using Python and Bluetooth Low Energy dongle by Bluegiga. This experiment is about getting things to work fast.

Get the source code here.

Angel Sensor uses different Bluetooth Low Energy (BLE) profiles to transmit data to client devices. Probably the most popular health related protocol standardized by the Bluetooth Special Interest Group (SIG) is Heart Rate. In this experiment we use a BLED112 dongle and some quick Python to converse with Angel's heart rate service.



Overview

GATT Attributes

All BLE communication revolves around reading and writing attributes. This is handled by the Generic Attribute Profile (GATT). Angel Sensor acts as the server that announces its attributes to the world by means of GATT. Heart Rate Measurement is one of the attributes it exposes.

GATT Attributes are typically organized in a 3-level hierarchy. The top level lists services. Each service contains one or more characteristics (i.e. attributes or variables). Each characteristic may contain one or more descriptors that provide additional details about the characteristic. These may describe supported access modes such as read/write/notify/indicate, provide textual descriptions of the characteristic, its format, range, and so on.

Here's what the relevant GATT descriptors look like for Angel's Heart Rate service (service UUID 0x180D, handles 17 to 21):

Descriptor	**UUID**	**Handle**
GATT Primary Service Declaration	0x2800	17
GATT Characteristic Declaration	0x2803	18
Heart Rate Measurement	0x2A37	19
Characteristic User Description	0x2901	20
Client Characteristic Configuration	0x2902	21

Order is important! The GATT table itself is completely flat, but special declaration descriptors are used to create hierarchy levels. For example, services are preceded by a special declaration descriptor numbered 0x2800 or 0x2801 (primary or secondary service).

Attribute types are defined by unique identifiers (UUID) that are either assigned by the Bluetooth SIG, or specified by the protocol developer. Some descriptor types may appear more than once, so knowing their type alone is insufficient. Handles specify the exact location of the descriptor in the GATT table, and thus its specific context in the hierarchy. In our case, Client Characteristic Configuration descriptor 0x2902 is used by many services. The instance relevant to Angel's Heart Rate service is represented by handle 21.

Note: An excellent introduction to GATT services and characteristics is available on Safari. The discovery process of the GATT table is called enumeration. This experiment sources contains the relevant code to enumerate the Heart Rate Service automatically. We'll dedicate a special experiment to this subject later on.

BLED112 by Bluegiga

BLED112 by Bluegiga is a USB dongle designed to add BLE functionality to any Windows-based machine. BLED112 uses a virtual serial port for its control protocol. Serial port commands are sent and received to control the dongle and to communicate with remote devices over BLE. A simple command contains a 4-byte header and a few optional payload bytes. For example, the reset command carries a header and a single payload byte 0x00: [0x00 0x01 0x00 0x00 : 0x00]. Serial commands, responses and events are defined in *bled112.py*:

```
class SystemResetCommand(BleCommand):
    def __init__(self):
        BleCommand.__init__(self, (0, 1, 0, 0), [0])
```

When BLE messages arrive from Angel Sensor, the dongle reports them as attribute events. For example, heart rate measurement [0x80 0x00 0x04 0x05 : *<payload>*]

```
class AttClientAttributeValueEvent(BleEvent):
    def __init__(self, payload=[]):
        BleEvent.__init__(self, (0x80, 0x00, 0x04, 0x05), payload)
        if payload:
            self.connection = payload[0]
            self.attHandle = Uint16().deserialize(payload[1:3])
            self.type = payload[3]
            self.data = payload[5:]
```

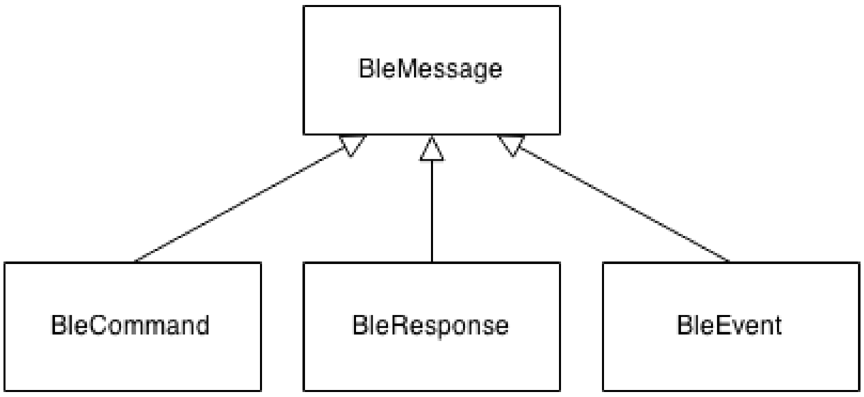
This event is of particular interest since it's how heart rate notifications are reported. Its payload contains the following fields:

Payload offset	Field
0	Connection handle
1-2	Attribute handle (HR is 19), LSB first
3	Type (read, notify, etc.)
4	Value length
5+	Value, byte array

Source Code Structure

The source *bled112.py* contains low-level definitions for using Bluegiga API over serial. The main classes are:

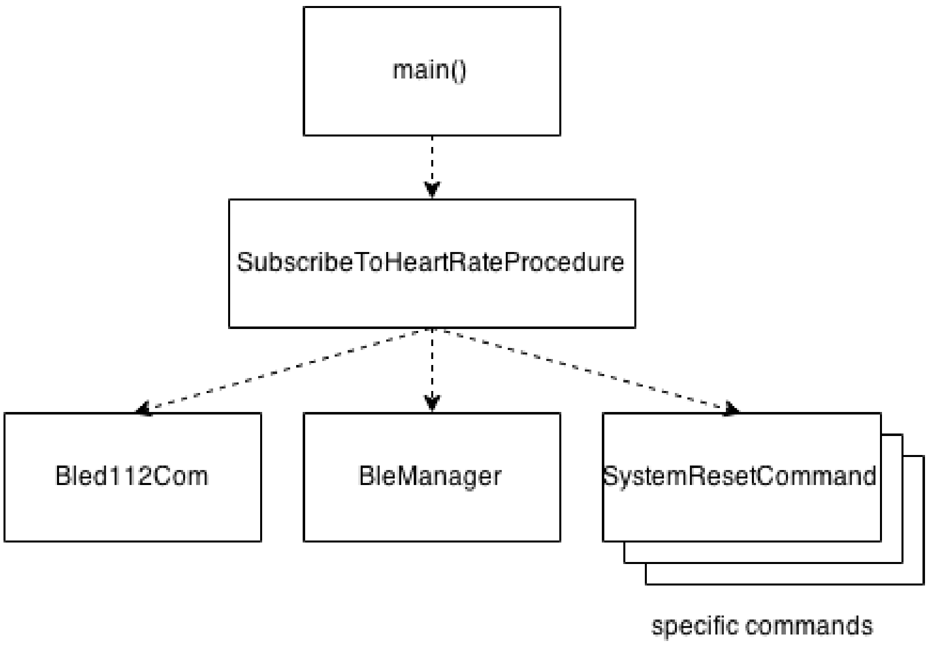
- BleMessage and its sub-classes BleCommand, BleEvent, BleResponse.
- Concrete BGAPI commands, responses and events such as SystemResetCommand.
- Bled112Com - contains the serial port listener thread and read/write methods.



BLE message types

High-level logic is handle in *main.py*, including:

- BleException and sub-classes (BleProcedureFailure, BleRemoteTimeout, etc.)
- BleManager - attribute access and remote procedures such as reading attributes, subscribing to notifications, and so on.
- App logic organized in functionality blocks: SubscribeToHeartRateProcedure, ListenToHeartRateProcedure. It may be somewhat overdoing things, but it makes the flow easier to understand.

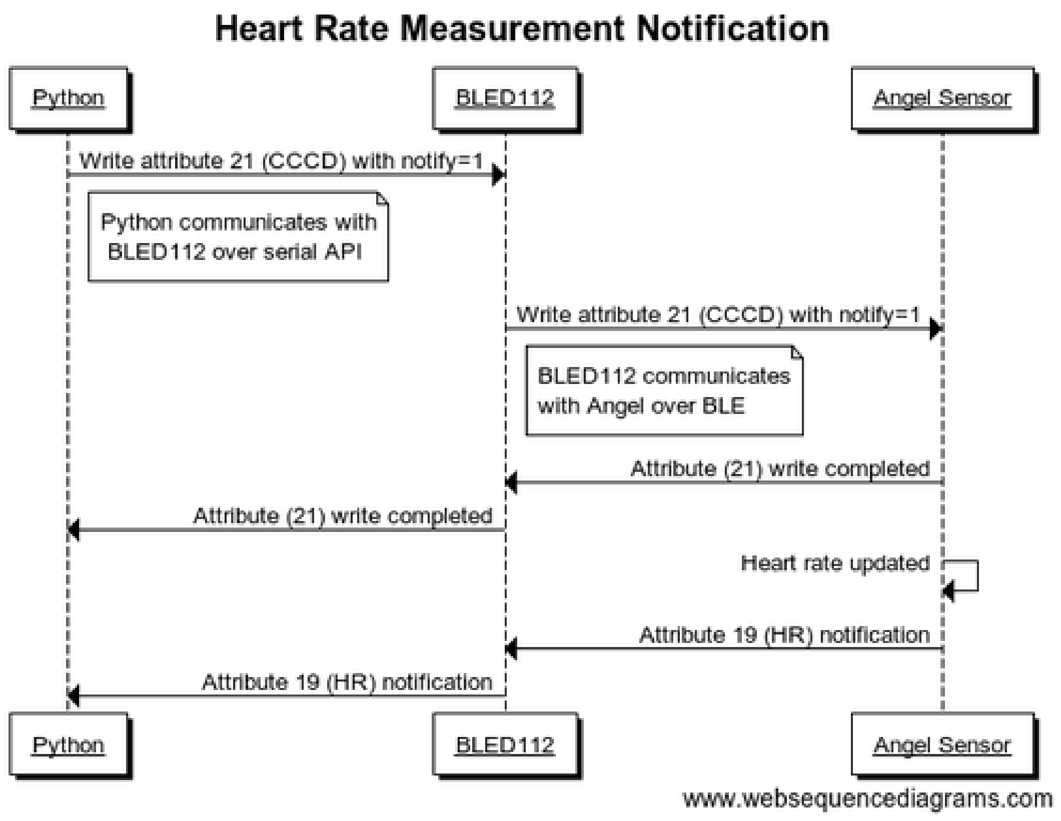


Partial class diagram

Enabling Heart Rate Notifications

Heart rate notifications from Angel are generated each time a new measurement is available. This is more efficient than reading the characteristic periodically (in fact, reading it directly is not supported by the spec). To subscribe to notifications, the Client Characteristic Configuration Descriptor (CCCD) for Heart Rate Measurement needs to be configured. The name is a mouthful, but in terms of hierarchy the CCCD is simply a descriptor containing bit flags for access modes. Setting bit 0 enables notifications, bit 1 enables indications. Notifications are unacknowledged, while indications are acknowledged. Notifications are therefore faster (i.e. use less bandwidth), but less reliable. (Read more in this quick overview.)

Heart Rate Measurement uses handle 19, its CCCD has handle 21. We aim to run the following sequence:


Heart Rate Measurement Notification
Python
BLED112
Angel Sensor
Write attribute 21 (CCCD) with notify=1
Python communicates with BLED112 over serial API
Write attribute 21 (CCCD) with notify=1
BLED112 communicates with Angel over BLE
Attribute (21) write completed
Attribute (21) write completed
Heart rate updated
Attribute 19 (HR) notification
Attribute 19 (HR) notification
Python
BLED112
Angel Sensor
www.websequencediagrams.com

Listening To Heart Rate Notifications

When notifications are enabled, Angel will trigger a *AttClientAttributeValueEvent*with attribute handle 19. The full format of Heart Rate Measurement characteristicis beyond our scope. Suffice to say, Angel Sensor reports heart rate (beats per minute) at byte 1 of the payload. Thus, the following endless loop is all we need to print heart rate values as they arrive:

```
while True:
    evt = self.ble.waitRemote(AttClientAttributeValueEvent(), 30)
    if evt.attHandle == self.heartRateHandle:
        # Assume format as reported by Angel Sensor, ignore flags
        logging.info('Heart rate = %u' % evt.data[1])
```

Running

First, download the source code for this experiment from GitHub. Assuming the MAC address of the Angel Sensor is 00:07:80:AB:CD:EF, simply execute the following in Windows command prompt, in the source code folder:

python main.py -a 00:07:80:AB:CD:EF.

This experiment uses the PySerial module which you can get from SourceForge.

Here's the output on my machine:

```
Reset BLED112
Connecting to F2:F2:02:80:07:00
Enumerating Heart Rate Service
Listening to heart rate notifications. Press Ctrl-C to quit.
Heart rate = 67
Heart rate = 66
Heart rate = 64
Heart rate = 61
Terminated by keystroke
```

Good luck!

---

* Caps: I've been reading too many BLE specs recently. As a result, I tend to follow Bluetooth SIG convention and capitalize service and characteristic names.

** I've used the wonderful WebSequenceDiagrams.com for this post and many others.