AN983: *BLUETOOTH*® 4.0 HEART RATE SENSOR

APPLICATION NOTE

Wednesday, 02 December 2020

Version 1.5

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Services added
1.2	Updated BGscript code examples
1.3	Changed broadcast=”true” to advertise=”true”
1.4	Updated compile and installation instructions
1.5	Renamed "Bluetooth Smart" to "Bluetooth Low Energy" according to the official Bluetooth SIG nomenclature

TABLE OF CONTENTS

1 Introduction

This application note discusses how to build *Bluetooth* 4.0 Heart Rate Profile (HRP) sensor using Bluegiga's *Bluetooth* 4.0 software and DKBLE112 hardware development kits. The application note contains a practical example of how to build GATT based Heart Rate Profile and how to make a standalone sensor device using BGScript scripting language.

Notice that this application note only focuses on the Heart Rate Profile sensor implementation, not the Heart Rate Profile Collector implementation.

2 What is *Bluetooth* Low Energy Technology?

Bluetooth Low Energy (*Bluetooth* 4.0) is a new, open standard developed by the *Bluetooth* SIG. It's targeted to address the needs of new modern wireless applications such as ultra-low power consumption, fast connection times, reliability and security. *Bluetooth* Low Energy consumes 10-20 times less power and is able to transmit data 50 times quicker than classical *Bluetooth* solutions.

Link: How Bluetooth low energy technology works?

Bluetooth Low Energy is designed for new emerging applications and markets, but it still embraces the very same benefits we already know from the classical, well established *Bluetooth* technology:

- **Robustness and reliability** - The adaptive frequency hopping technology used by *Bluetooth* Low Energy allows the device to quickly hop within a wide frequency band, not just to reduce interference but also to identify crowded frequencies and avoid them. On addition to broadcasting *Bluetooth* Low Energy also provides a reliable, connection oriented way of transmitting data.
- **Security** - Data privacy and integrity is always a concern is wireless, mission critical applications. Therefore *Bluetooth* Low Energy technology is designed to incorporate high level of security including authentication, authorization, encryption and man-in-the-middle protection.
- **Interoperability** - *Bluetooth* Low Energy technology is an open standard maintained and developed by the *Bluetooth* SIG. Strong qualification and interoperability testing processes are included in the development of technology so that wireless device manufacturers can enjoy the benefit of many solution providers and consumers can feel confident that equipment will communicate with other devices regardless of manufacturer.
- **Global availability** - Based on the open, license free 2.4GHz frequency band, *Bluetooth* Low Energy technology can be used in world wide applications.

There are two types of *Bluetooth* 4.0 devices:

- ***Bluetooth* 4.0 single-mode** devices that only support *Bluetooth* Low Energy and are optimized for low-power, low-cost and small size solutions.
- ***Bluetooth* 4.0 dual-mode** devices that support *Bluetooth* Low Energy and classical *Bluetooth* technologies and are interoperable with all the previously *Bluetooth* specification versions.

Key features of *Bluetooth* Low Energy wireless technology include:

- Ultra-low peak, average and idle mode power consumption
- Ability to run for years on standard, coin-cell batteries
- Low cost
- Multi-vendor interoperability
- Enhanced range

Bluetooth Low Energy is also meant for markets and applications, such as:

- Automotive
- Consumer electronics
- Smart energy
- Entertainment
- Home automation
- Security & proximity
- Sports & fitness

3 Typical *Bluetooth* 4.0 Application Architecture

3.1 Overview

Bluetooth Low Energy applications typically have the following architecture:

- **Server**

  Service is the device that provides the information, so these are typically the sensor devices, like thermometers or heart rate sensors. The server exposes implements services and the services expose the data in characteristics.

- **Client**

  Client is the device that collects the information for one or more sensors and typically either displays it to the user or passes it forward. The client devices typically do not implement any service, but just collect the information from the service provided by the server devices. Clients are typically devices like mobile phones, tablets and PCs.

The figure below shows the relationship of these two roles.

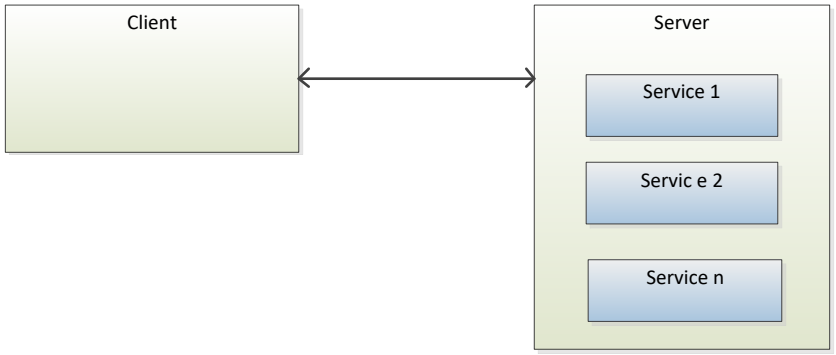

Figure 1: *Bluetooth* Low Energy device roles

3.2 What is a Profile?

Profiles are used to describe devices and the data they expose and also how these devices behave. The data is described by using services, which are explained later and a profile may implement single or multiple services depending on the profile specification. For example a Heart Rate Service specification mandates that the following services need to be implemented:

- Heart Rate Service
- Device Information Service

Profile specifications might also define other requirements such as security, advertisement intervals and connection parameters.

The purpose of profile specifications is to allow device and software vendors to build standardized interoperable devices and software. Standardized profiles have globally unique 16-bit UUID, so they can easily identify.

Profiles are defined in profiles specifications, which are available at:

https://developer.bluetooth.org/gatt/profiles/Pages/ProfilesHome.aspx

3.3 What Is a Service?

Services such as a Heart Rate service describes what kind of data a device exposes, how the data can be accessed and what the security requirements for that data are. The data is described using characteristics and a service may contain single or multiple characteristics and some characteristics might be optional where as some are mandatory.

Two types of services exist:

- **Primary Service**

  A primary service is a service that exposes primary usable functionality of this device. A primary service can be included by another service.

- **Secondary Service**

  A secondary service is a service that is subservient to another secondary service or primary service. A secondary service is only relevant in the context of another service.

Just like the profiles also the services are defined in service specifications and the Bluetooth SIG standardized services are available at:

https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

Every service standardized by the Bluetooth SIG has a globally unique 16-bit UUID so just like the profiles also the services can be easily identified.

However not every use case can be fulfilled by the standardized service and therefore the *Bluetooth* Low Energy specification enables device vendors to make proprietary service. The proprietary services are described just as the standardized services, but 128-bit UUIDs need to be used instead of use 16-bit UUIDs reserved for the standard services.

3.4 What is a Characteristic?

Characteristics are used to expose the actual data. Characteristic is a value, with a known type (UINT8, UINT16, UTF-8 etc.), a known presentation format. Just like profiles and services also characteristics have unique UUID so they can be easily identified and the standardized characteristics use 16-bit UUIDs and vendor specific characteristics use 128-bit UUIDs.

Characteristics consist of:

- **Characteristic Declaration** describing the properties of characteristic value such as:
  - characteristic (UUID)
  - Access control *(*read, write, indicate etc.)
  - *Characteristic value* handle (unique handle within a single device)
- **Characteristic Value** containing the value of a characteristic (for example temperature reading).
- **Characteristic Descriptor(s)** which provide additional information about the characteristic (characteristic user description, characteristic client configuration, vendor specific information etc.).

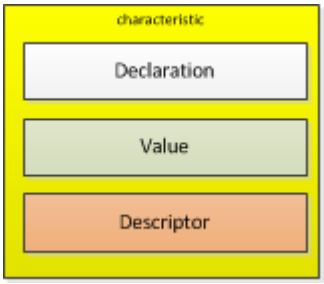

Figure 2: Characteristic structure

Standardized characteristics are defined in Characteristic Specification and the standardized characteristics are available at:

https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx

3.5 Relationship Between Profiles, Services and Characteristics

The illustration below shows the relationship between profiles, services and characteristics.

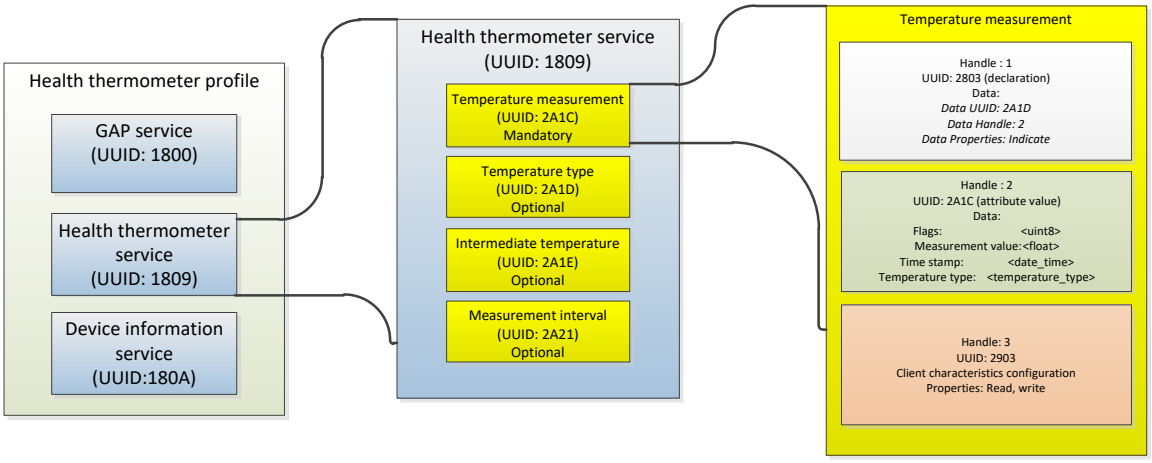

Figure 3: Health thermometer profile

4 Introduction to the Bluegiga *Bluetooth* Low Energy Software

The Bluegiga *Bluetooth* Low Energy Software enables developers to quickly and easily develop *Bluetooth* Low Energy applications without in-depth knowledge of the *Bluetooth* Low Energy technology. The *Bluetooth* Low Energy Software consist of two parts:

- The *Bluetooth* Low Energy Stack
- The *Bluetooth* Low Energy Software Development Kit (SDK)

4.1 The *Bluetooth* Low Energy Stack

The *Bluetooth* Low Energy stack is a fully *Bluetooth* 4.0 single mode compatible software stack implementing slave and master modes, all the protocol layers such as L2CAP, Attribute Protocol (ATT), Generic Attribute Profile (GATT), Generic Access Profile (GAP) and security and connection management.

The *Bluetooth* Low Energy is meant for the Bluegiga *Bluetooth* Low Energy products such as BLE112, BLE113 and BLED112 and it runs on the embedded MCU used in these products, so no host is needed.

4.2 The *Bluetooth* Low Energy SDK

The *Bluetooth* Low Energy SDK is a software development kit, which enables the device and software vendors to develop products on top of the Bluegiga's *Bluetooth* Low Energy hardware and software.

The *Bluetooth* Low Energy SDK supports multiple development models and the software developers can decide whether the application software runs on a separate host (a low power MCU) or whether they want to make fully standalone devices and execute their code on the MCU embedded in the Bluegiga *Bluetooth* Low Energy modules. The SDK also contains documentation, tools for compiling the firmware, installing it into the hardware and lot of example application speeding up the development process.

fully standalone applications using a simple scripting language called BGScript™. Several profiles and examples are also offered as a part of the *Bluetooth* Low Energy Software in order to easily develop the *Bluetooth* Low Energy compatible end products.

Bluegiga's *Bluetooth Low Energy Software* provides a complete development framework for *Bluetooth* Low Energy application implementers.

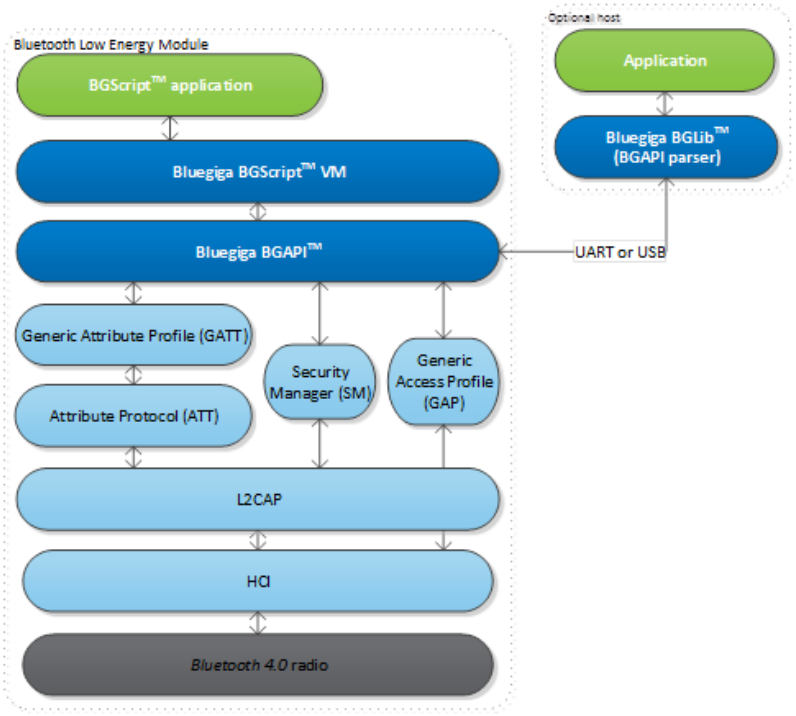

Figure 4: *Bluetooth Low Energy* Software

The *Bluetooth Low Energy* Software architecture is illustrated and it consists of the following components

- The *Bluetooth* Low Energy stack implementing the *Bluetooth* Low Energy protocol
- **BGAPI™** APIs that enable the software developers to interface to the *Bluetooth* Low Energy Stack
- **BGScript™** Virtual Machine (VM) and scripting language which enable application code to be developed and executed directly on the *Bluetooth* Low Energy hardware
- **BGLib™** lightweight host library which implements the BGAPI binary protocol and parser and is target for applications where separate host processor is used to interface to the *Bluetooth* Low Energy modules over UART or USB.
- **Profile Toolkit™** is a GATT based profile development tool that enables software developers quickly and easily to describe the *Bluetooth* Low Energy profiles, services and characteristics using simple XML templates

Each of these components are described in more detail in the following chapters.

4.3 The BGAPI Protocol

For applications where a separate host is used to implement the end user application, a transport protocol is needed between the host and the *Bluetooth* stack. The transport protocol is used to communicate with the *Bluetooth* stack as well to transmit and receive data packets. This protocol is called BGAPI and it's a lightweight binary based communication protocol designed specifically for ease of implementation within host devices with limited resources.

The BGAPI protocol is a simple command, response and event based protocol and it can be used over UART SPI (at the moment not supported by the *Bluetooth* Low Energy hardware) or USB interfaces.

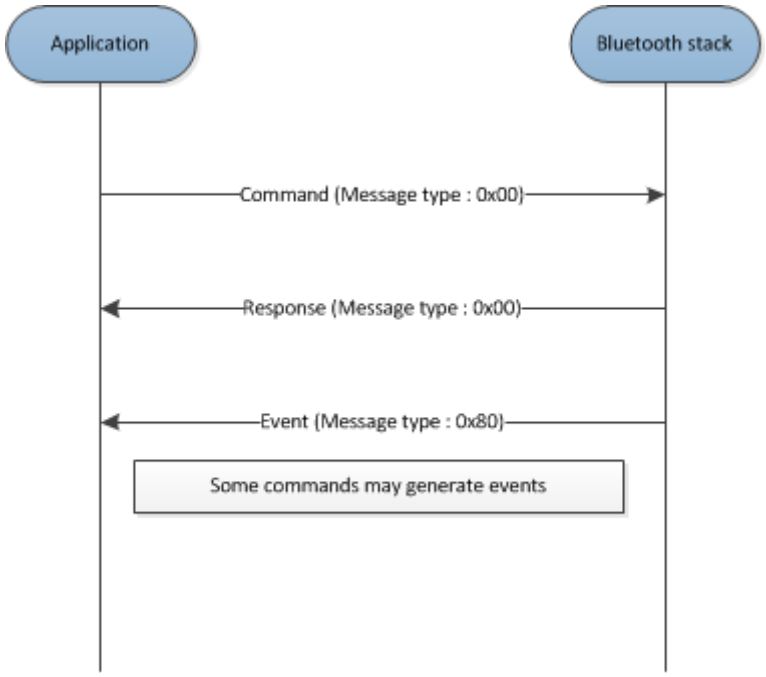

Figure 5: BGAPI protocol

The BGAPI provides access for example to the following layers in the *Bluetooth* Low Energy Stack:

- **Generic Access Profile** - GAP allows the management of discoverability and connetability modes and open connections
- **Security manager** - Provides access the *Bluetooth* Low Energy security functions
- **Attribute database** - An class to access the local attribute database
- **Attribute client** - Provides an interface to discover, read and write remote attributes
- **Connection** - Provides an interface to manage *Bluetooth* Low Energy connections
- **Hardware -** An interface to access the various hardware layers such as timers, ADC and other hardware interfaces
- **Persistent Store** - User to access the parameters of the radio hardware and read/write data to non-volatile memory
- **System** - Various system functions, such as querying the hardware status or reset it

4.4 The BGLib Host Library

For easy implementation of BGAPI protocol an ANSI C host library is available. The library is easily portable ANSI C code delivered within the *Bluetooth* Low Energy SDK. The purpose is to simplify the application development to various host environments.

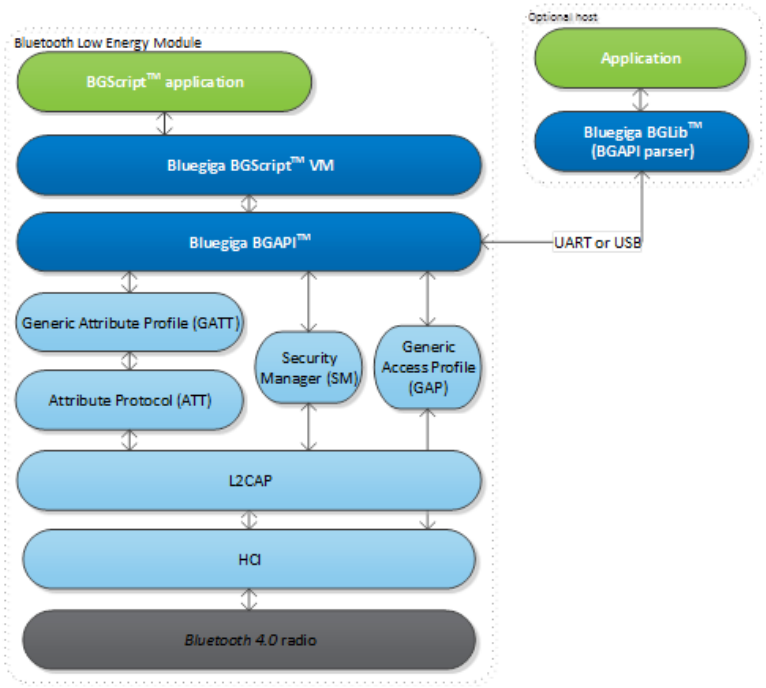

Figure 6: BGLib host library

4.5 BGScript™ Scripting Language

The *Bluetooth* Low Energy SDK Also allows the application developers to create fully standalone devices without a separate host MCU and run all the application code on the Bluegiga *Bluetooth* Low Energy Hardware. The *Bluetooth* Low Energy modules can run simple applications along the *Bluetooth* Low Energy stack and this provides a benefit when one needs to minimize the end product's size, cost and current consumption. For developing standalone *Bluetooth* Low Energy applications the SDK includes the Script VM, compiler and other BGScript development tools. BGScript provides access to the same software and hardware interfaces as the BGAPI protocol and the BGScript code can be developed and compiled with free-of-charge tools provided by Bluegiga.

Typical BGScript applications are only few tens to hundreds lines of code, so they are really quick and easy to develop and lots of readymade examples are provides with the SDK.

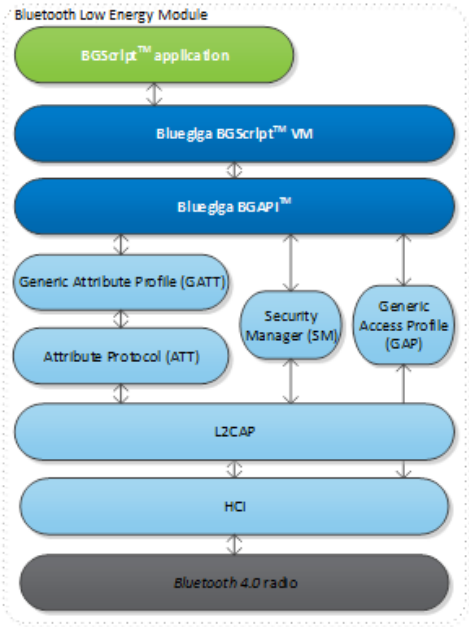

Figure 7: BGScript application model

BGScript code example:

```
# System Started
event system_boot(major, minor, patch, build, ll_version, protocol_version,hw)
            #Enable advertising mode
            call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
            #Enable bondable mode
            call sm_set_bondable_mode(1)
            #Start timer at 1 second interval (32768 = crystal frequency)
            call hardware_set_soft_timer(32768)
end
```

4.6 The Profile Toolkit

The *Bluetooth* Low Energy profile toolkit a simple set of tools, which can used to describe GATT based *Bluetooth* Low Energy services and characteristics. The profile toolkit consists of a simple XML based description language and templates, which can be used to describe the devices GATT database. The profile toolkit also contains a compiler, which converts the XML to binary format and generates API to access the characteristic values.

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

    <service uuid="1800">
      <description>Generic Access Profile</description>

      <characteristic uuid="2a00">
        <properties read="true" const="true" />
        <value>BGDemo sensor</value>
      </characteristic>

      <characteristic uuid="2a01">
        <properties read="true" const="true" />
        <value type="hex">4142</value>
      </characteristic>
    </service>

</configuration>
```

Figure 8: A profile toolkit example of GAP service

5 Heart Rate Profile v1.0

5.1 Description

Heart Rate Profile enables a Heart Rate Collector to connect and exchange data with a Heart Rate Sensor in sports and fitness applications.

Heart Rate Profile defines two roles:

- **The Heart Rate Sensor**

  The Heart Rate Sensor measures the heart rate and exposes it via the Heart Rate Service. The sensor also contains the Device Information Service, which contains information for example about the manufacturer of the device. The Heart Rate Sensor is the GATT server.

- **The Heart Rate Collector**

  The Heart Rate Collector accesses the information exposed by the Heart Rate Sensor and can for example display it to the end user or store it on non-volatile memory for later analysis. The Heart Rate Collector is the GATT client.

The figure below shows the relationship of these two roles.

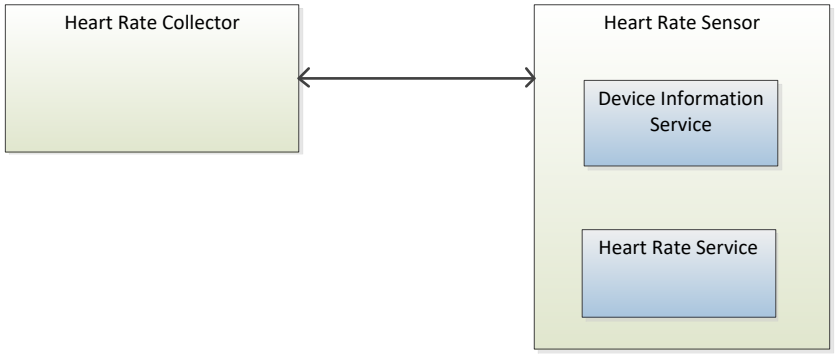

Figure 9: Heart Rate profile roles

5.2 Service requirements

The table below describes the service requirements.

Service	UUID	Heart Rate Sensor
Heart Rate Service	180A	Mandatory
Device Information Service	180D	Mandatory

Table 1: Service requirements

The Heart Rate Sensor implements one and only one instance of Heart Rate Service.

The Heart Rate Sensor implements one instance of Device Information Service.

5.3 Heart Rate Service requirements

The table below describes the structure and requirements for the Heart Rate Service

Characteristic	UUID	Type	Support	Security	Properties
Heart rate measurement	2A37	8bit	Mandatory	none	Notify
Body sensor location	2A38	8bit	Optional	none	Read
Heart rate control point	2A39	8bit	Conditional	none	Write

Table 2: Heart Rate Service structure

5.4 Device Information Service requirements

The table below describes the structure and requirements for the Device Information Service when used in the context of Heart Rate Service.

Characteristic	UUID	Type	Support	Security	Properties
Manufacturer name string	2A29	UTF-8	Mandatory	none	Read

Table 3: Device Information Service structure

5.5 Other requirements

The Heart Rate Sensor should include the Heart Rate Service UUID in the advertisement data.

The Heart Rate Sensor should include the device name in the advertisement or scan response data.

The Heart Rate Sensor may support write property for the local name for the device name characteristic so the Collector can write its value.

5.6 Connection establishment requirement

5.6.1 Un-bonded devices

Advertisement duration	Parameter	Value
First 30 seconds	Advertising interval	20ms to 30ms
After 30 seconds	Advertising interval	1000ms to 2500ms

Table 4: Advertising parameters for un-bonded Heart Rate Sensor

1. The Heart Rate Sensor shall accept any valid values for connection interval and slave latency set by the Collector until service discovery, bonding and/or encryption have are complete. After this the sensor may request the change of connection parameters.
2. If the connection is not established within a time limit, the sensor may exit GAP Connectable mode.
3. If bonded the Heart Rate Sensor should write the address of the Collector to the white list and should set the filtering policy so that scan and connection requests are only accepted from devices on the white list.
4. When the Heart Rate Sensor no longer senses the heart rate it should terminate the connection for example within 10 or 20 seconds.
5. When Heart Rate Sensor is disconnected by the Collector and ready to receive a connection (i.e. senses the heart rate) it should initiate the connection procedure.

5.6.2 Bonded devices

The following produce is uses for bonded devices:

1. The Heart Rate Sensor should use GAP General discoverable mode with connectable undirected advertisement events.
2. For the first 10 seconds the white list should be used to allow only connections from bonded devices. After 10 seconds the white list should not be used to allow connections from other devices.
3. The advertisement parameters should be as in Table 4.
4. The Heart Rate Sensor shall accept any valid values for connection interval and slave latency set by the Collector until service discovery, bonding and/or encryption have are complete. After this the sensor may request the change of connection parameters.
5. If the connection is not established within a time limit, the sensor may exit GAP Connectable mode.
6. When the Heart Rate Sensor no longer senses the heart rate it should terminate the connection for example within 10 or 20 seconds.
7. When Heart Rate Sensor is disconnected by the Collector and ready to receive a connection (i.e. senses the heart rate) it should initiate the connection procedure.

5.6.3 Link loss procedure

When connection is terminated due to link loss the sensor should attempt reconnection with the Collector by entering the GAP connectable mode using the recommended parameters from Table 4.

5.7 Security requirements

1 The Heart Rate Sensor may bond with the Collector.

2 When bonding is not used:

2.1 The Heart Rate Sensor should use the *Slave Security Request* procedure to inform the Collector of its security requirements.

3 When bonding is used:

3.1 The Heart Rate Sensor shall use LE security Mode 1 and either Security Level 2 or 3.

3.2 The Heart Rate Sensor shall use the *Slave Security Request* procedure.

3.3 All supported characteristics specified by the Heart Rate Service shall be set to Security Mode 1 and either Security Level 2 or 3.

All supported characteristics specified by the Device Information Service should be set to the same security mode and level as the characteristics in the Heart Rate Service.

6 Implementing a Heart Rate Sensor

The chapter contains step by step instructions how to implement a stand-alone Heart Rate Sensor with Bluegiga's Bluetooth 4.0 Software Development Kit. The chapter is split into following steps:

1. Creating a project
2. Defining hardware configuration
3. Building Heart Rate and Device Information Services with Profile Toolkit
4. Writing a BGScript code
5. Compiling the GATT database and BGScript into a binary firmware
6. Installing the firmware into BLE112 or DKBLE112 hardware

The actual project comes as an example with the Bluegiga's Bluetooth Low Energy Software Development Kit.

Note: This application note is written for firmware version 0.2.0 build 30.

6.1 Creating a project

The Heart Rate Sensor implementation is started by first creating a project file (**project.xml**), which defines the resources use by the project and the firmware output file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="HRP.xml" />
    <hardware in="hardware.xml" />
    <script in="hr_sensor.bgs" />
    <image out="HRP_v10.hex" />
</project>
```

Figure 10: Project file

- **<gatt>** Defines the XML-file containing the GATT database.
- **<hardware>** Defines the XML-file containing the hardware configuration.
- **<script>** Defines the BGScript-file which contains the BGScript code.
- **<image>** Defines the output HEX file containing the firmware image.

WARNING:

This example MUST only be installed on BLE112 module or DKBLE112 development kit. The example does not use USB or UART interfaces, so the firmware can be installed only via the debug interface using CC debugger. Installing the example into BLED112 USB dongle will brock the device.

6.2 Hardware configuration

The **hardware.xml** file contains the hardware configuration for BLE112 device. It describes which interfaces and functions are in used and their properties.

```
<?xml version="1.0" encoding="UTF-8" ?>

<hardware>
    <sleeposc enable="true" ppm="30" />
    <usb enable="false" />
    <txpower power="15" bias="5" />
</hardware>
```

Figure 11: Hardware configuration for Heart Rate Sensor

- **<sleeposc>** Sleep oscillator is enabled to allow the device to enter low power modes like Power mode 3. If this configuration is not used, the BLE112 device will not go to power mode 3. 30 refers to the crystal accuracy used in BLE112. **Do not modify the value.**
- **<usb>** USB interface is disabled to save power and allow the device to go to low power modes.
- **<txpower>** TX power is set to +3dBm value. Every step represents 1 dBm change. Range is 15 to 0, corresponding TX power values from +3 dBm to -12 dBm.

This example uses a single ADC to read the heart rate value. DKBL112 development kit contains a potentiometer, which can be read with the ADC.

Typically in a real heart rate sensor a GPIO would be used to indicate the heartbeat and a GPIO pin would be used to detect it. This requires that one GPIO pin is configured as an input.

6.3 Heart Rate Profile GATT database

This section describes how to define the Heart Rate Profile services using Bluegiga's Profile Toolkit.

The Heart Rate Profile contains three services:

- Generic Access Profile (GAP) service
- Device Information Service (DIS)
- Heart Rate Service (HRS)

This example contains a minimum implementation of the above services, so only mandatory characteristics are used. You may also implement the optional characteristics.

6.3.1 Generic Access Profile service

Every *Bluetooth* Low Energy device needs to implement a GAP service. The GAP service is very simple and consists of only two characteristics. An example implementation of GAP service is show below.

The service has two characteristics, which are explained in Table 5. In this example the characteristics are read-only, so they are also marked as **const.** Constant values are stored on the flash of BLE112 and the value is defined in the GATT database. Constant values cannot be changed.

```
<service uuid="1800">
  <description>Generic Access Profile</description>

  <characteristic uuid="2a00">
    <properties read="true" const="true" />
    <value>Bluegiga HR Profile</value>
  </characteristic>

  <characteristic uuid="2a01">
    <properties read="true" const="true" />
    <value type="hex">4142</value>
  </characteristic>
</service>
```

Figure 12: GAP service

Characteristic	UUID	Type	Support	Security	Properties
Device name	2a00	UTF8	Mandatory	none	Read (optionally write)
Appearance	2A01	16bit	Mandatory	none	Read

Table 5: GAP (UUID: 1800) service description

6.3.2 Heart Rate Service

The Heart Rate Sensor must also implement the Heart Rate Service. The example implementation uses a simplified Heart Rate Service with just the mandatory characteristics, but optionally you may implement the full Hear Rate Service with the optional characteristics also.

Heart Rate Service is described below:

Characteristic	UUID	Type	Support	Security	Properties
Heart Rate Measurement	2a37	16 bits to 56 bits	Mandatory	none	Notify
Body Sensor location	2a38	8 bit	Optional	none	Read
Heart Rate Control Point	2a39	8 bit	Conditional	none	Write

Table 6: Heart Rate Service (UUID: 180D) description

Hear rate service specification: Hear Rate Service specification: Hear rate service specification

Hear Rate service at Bluetooth developer web site: Heart Rate Service

The example implementation of the minimum Heart Rate Service is shown below:

```
<service uuid="180D" advertise="true">
  <description>Heart Rate</description>

    <characteristic uuid="2a37" id="xgatt_HRS_2a37">
        <properties notify="true" />
        <value type="hex">1122</value>
        <description>Heart Rate Measurement</description>
    </characteristic>
</service>
```

Figure 13: Minimal Heart Rate Service

A few explanations are needed:

- First of all the **advertise="true**" option is needed for the Hear Rate Service UUID to be broadcasted in the advertisement packets. For example the Apple iPhone 4S is not able to discover devices, if the service UUIDs are not broadcasted.
- The **id="xgatt_HRS_2a37**" defines the attribute ID, which the BGScript application can use to update the Heart Rate measurement values.
- The length of the Heart Rate measurement is 16-bits in this example. The first 8 bits define the flags for the Heart Rate measurement value and the next 8 bits contain the actual measurement value.
- *Body Sensor location* and *Heart Rate Control Point* characteristics are not used in this example as they are not mandatory.

6.3.3 Device Information Service

The third mandatory service the Heart Rate Sensor must implement is the Device Information Service. This service exposes information about the manufacturer of the device and optionally other information about the device, which is for example device model number and software version. The example implementation uses a simplified Device Information Service with just the mandatory characteristics, but optionally you may implement the full Device Information Service with the optional characteristics also.

Device Information Service is described below:

Characteristic	UUID	Type	Support	Security	Properties
Manufacturer name string	2a29	UTF-8s	Conditional	none	Read
Model number string	2a24	UTF-8s	Conditional	none	Read
Serial number string	2a25	UTF-8s	Conditional	none	Read
Hardware revision string	2a27	UTF-8s	Conditional	none	Read
Firmware revision string	2a26	UTF-8s	Conditional	none	Read
Software revision string	2a28	UTF-8s	Conditional	none	Read
System ID	2a35	uint40 or uint64	Conditional	none	Read
IEEE 11073-20601 Regulatory Certification Data List	2a36	reg-cert-data-list	Conditional	none	Read

Table 7: Device Information Service (UUID: 180A) description

Device Information Service specification: Hear Rate Service specification: Device Information Service specification

Device Information Service at Bluetooth developer web site: Device Information Service

The example implementation of the minimum Device Information Service is shown below:

```
<service uuid="180A">
  <description>Device Information</description>

  <characteristic uuid="2a29">
    <properties read="true" const="true" />
    <value>Bluegiga</value>
    <description>Manufacturer Name String</description>
  </characteristic>

  <characteristic uuid="2a24">
    <properties read="true" const="true" />
    <value>BLE112</value>
    <description>Model Number String</description>
  </characteristic>

</service>
```

Figure 14: Minimal Device Information Service

A few explanations are needed:

- The Heart Rate Profile service only mandates that *Manufacturer Name String* characteristic is implemented, but the example also implements the *Model Number String* characteristic.

6.3.4 Summary

The full GATT database implementation is shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

    <service uuid="1800">
      <description>Generic Access Profile</description>

      <characteristic uuid="2a00">
        <properties read="true" const="true" />
        <value>DKBLE112 heart rate</value>
      </characteristic>

      <characteristic uuid="2a01">
        <properties read="true" const="true" />
        <value type="hex">4142</value>
      </characteristic>
    </service>

    <service uuid="180A">
      <description>Device Information</description>

      <characteristic uuid="2a29">
        <properties read="true" const="true" />
        <value>Bluegiga</value>
        <description>Manufacturer Name String</description>
      </characteristic>

      <characteristic uuid="2a24">
        <properties read="true" const="true" />
        <value>BLE112</value>
        <description>Model Number String</description>
      </characteristic>
    </service>

    <service uuid="180D" advertise="true">
      <description>Heart Rate</description>

        <characteristic uuid="2a37" id="xgatt_HRS_2a37">
            <properties notify="true" />
            <value type="hex">1122</value>
            <description>Heart Rate Measurement</description>
        </characteristic>
    </service>

</configuration>
```

Figure 15: Heart Rate Profile GATT database

6.4 Writing BGScript application

The example implements a standalone Heart Rate Sensor device where no external host processor is needed. The Heart Rate Sensor application is created as a BGScript script application and the BGScript code explained in this chapter.

BGScript uses an event based programming approach. The script is executed when an event takes place, and the programmer may register listeners for various events.

The Heart Rate Sensor BGScript uses the following event listeners:

1. **`system_boot(…)` event listener**

   When the system is started a boot event is generated and this event listener should be the entry point for all the BGScript applications. In the example above, when the system is started, the unit starts to advertise, enables bonding mode, and starts a timer.

```
# System start/boot listener
event system_boot(major,minor,patch,build,ll_version,protocol,hw)

    # Device is not connected yet
    connected = 0

    # Set advertisement interval to 20 to 30ms. Use all advertisement channels
    call gap_set_adv_parameters(32,48,7)

    # Start advertisement (generic discoverable, undirected connectable)
    call gap_set_mode(2,2)

    # Start single shot timer with 30 second interval. Handle ID 1
    # 1 second = $8000 (32.768kHz crystal)
    call hardware_set_soft_timer($F0000, 1, 1)
end
```

Figure 16: system_boot(...) event listener

2. **`hardware_soft_timer(…)` event listener**

When the timer expires this event is generated. In the Heart Rate Sensor example the timers are used to alter the advertising intervals and read HR sensor.

```
event hardware_soft_timer(handle)

    # 30 second timer expired
    if handle = 1 then
        # No connection
        if connected = 0 then
            # Stop advertisement
            call gap_set_mode(0, 0)

            # Reconfigure advertisement parameters
            # Min interval 1000ms, max interval 2500ms, use all 3 channels
            call gap_set_adv_parameters(1600, 4000, 7)

            # Enabled advertisement
            # Limited discovery, Undirected connectable
            call gap_set_mode(1, 2)

            # Start single shot timer with 30 second interval. Handle ID 2
            # This is used to stop advertisemnts after 60 seconds to save power
            call hardware_set_soft_timer($F0000, 2, 1)
        end if
    end if

    # 60 second timer expired
    if handle = 2 then
        # No connection
        if connected = 0 then
            # Stop advertisement. Device will enter Power Mode 3 to save battery
            call gap_set_mode(0, 0)
        end if
    end if

    # HR timer expired
    if handle = 3 then
        #read DKBLE112 potentiometer, decimation 128, use avdd5 as reference
        call hardware_adc_read(6,1,2)
    end if
end
```

Figure 17: hardware_soft_timer(…) event listener

3. `hardware_adc_result(…)` event listener

The ADC read function generates an ADC event, which this event listener captures. The ADC result event is used to read the HR value and write it to GATT database.

```
#ADC event listener for HR measurement
event hardware_adc_result(input,value)

    #potentiometer value is measured
    if input = 6 then
            # Heart Rate Measurement flags field (8 bits)
            # RR not present, EE not present, SC feature supported, but no contact detected, HR format is UINT8
            hrm(0:1)=2
            #calculate some valid hr value 20-224
            hrm(1:1)=value/160+20
            # Write value to GATT database
            call attributes_write(xgatt_HRS_2a37,0,2,hrm(0:2))
    end if
end
```

Figure 18: hardware_adc_result(…) event listener

4. `connection_status(…)` event listener

This event takes place when the device is connected. The code changes the connection status parameter and starts the HR measurement timer.

```
# Connection event listener
event connection_status(connection, flags, address, address_type, conn_interval, timeout, latency)
    # End advertisement timers, so HR timer can be started
    call hardware_set_soft_timer(0, 2, 1)
    call hardware_set_soft_timer(0, 1, 1)
    # Device is connected.
    # Set <connected> to true, or otherwise the advertisemnt timers will disconnect the device
    connected = 1
    # Start HR monitoring timer: 1 second interval, ID 3, continuous timer
    call hardware_set_soft_timer($8000, 3, 0)
end
```

Figure 19: connection_status(…) event listener

5. **`connection_disconnected(…)` event listener**

The last event handler is executed when the *Bluetooth* is lost or closed by the remote device. The event listener restarts the advertisement procedure.

```
# Disconnection event listener
event connection_disconnected(handle, result)
    # End HR timer, so advertisement timer can be restarted
    call hardware_set_soft_timer(0, 3, 0)
    # Connection disconnected, reinitiate connection procedure
    connected = 0
    call gap_set_adv_parameters(32,48,7)
    call gap_set_mode(2,2)
    call hardware_set_soft_timer($F0000, 1, 1)
end
```

Figure 20: connection_disconnected(…) event listener

6.5 Compiling and Installing the Firmware

6.5.1 Using BLE Update tool

When you want to test your project, you need to compile the hardware settings, the GATT data base and BGScript code into a firmware binary file. The easiest way to do this is with the BLE Update tool that can be used to compile the project and install the firmware to a Bluetooth Low Energy Module using a CC debugger tools

In order to compile and install the project:

1. Connect CC debugger to the PC via USB
2. Connect the CC debugger to the debug interface on the BLE112 or BLE113
3. Press the button on CC debugger and make sure the led turns green
4. Start **BLE Update** tool
5. Make sure the CC debugger is shown in the **Port** drop down list
6. Use Browse to locate your **project** file (for example **BLE113-project.bgproj**)
7. Press **Update**

   BLE Update tool will compile the project and install it into the target device.

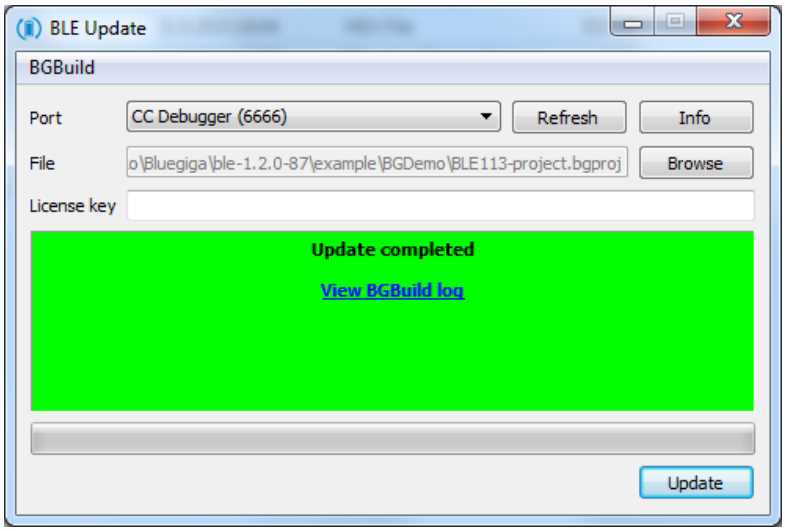

Figure 21: Compile and install with BLE Update tool

Note:

You can also double clikc the .BGPROJ file and it will automatically open the BLE Update tool.

If you have BLE113 Development Kit v.1.2 the CC debugger component is already placed on the kit and you simply need to:

- Connect the **DEBUGGER** USB port to the PC
- Turn the **DEBUGGER** switch to **MODULE**
- Press the **RESET DEBUGGER** button and make sure the **DEBUGGER** led turns green

The **View Build Log** opens up a dialog that shows the bgbuild compilere output and the RAM and Flash memory allocations.

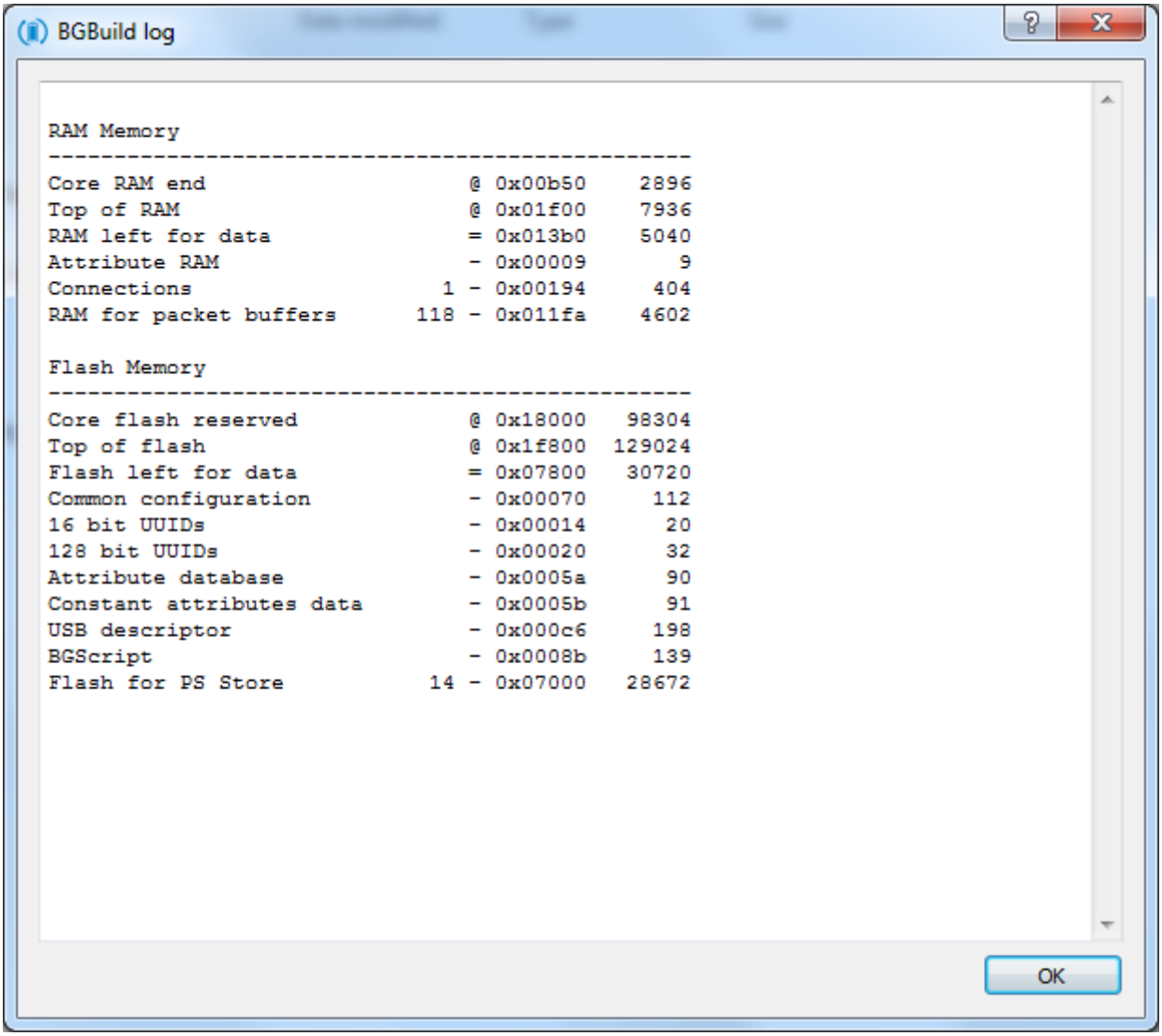

Figure 22: BLE Update build log

6.5.2 Compiling Using bgbuild.exe

The project can also be compiled with the **bgbuild.exe** command line compiler. The BGBuild compiler simply generates the firmware image file, which can be installed to the BLE112 or BLE113.

In order to compile the project using BGBuild:

1. Open Windows Command Prompt (cmd.exe)
2. Navigate to the directory where your project is
3. Execute BGbuild.exe compiler

   Syntax: *bgbuild.exe <project file>*

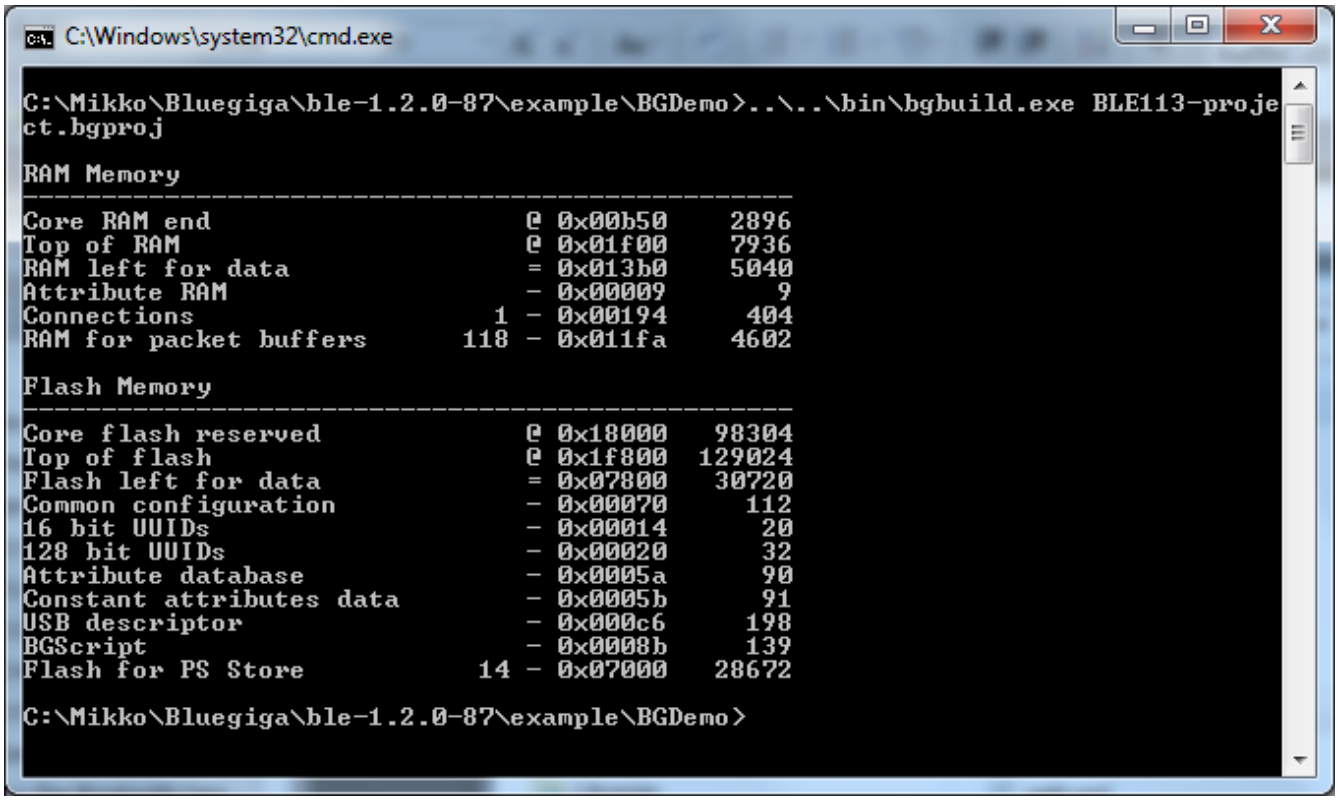

Figure 23: Compiling with BGBuild.exe

If the compilation is successful a .HEX file is generated, which can be installed into a Bluetooth Low Energy Module.

On the other hand if the compilation fails due to syntax errors in the BGScript or GATT files, and error message is printed.

6.5.3 Installing the firmware with TI's Flash Tool

Texas Instruments flash tool can also be used to install the firmware into the target device using the CC debugger.

In order to install the firmware with TI flash tool:

1. Connect CC debugger to the PC via USB
2. Connect the CC debugger to the debug interface on the BLE112
3. Press the button on CC debugger and make sure the led turns green
4. Start **TI flash tool** tool
5. Select program **CCxxxx SoC or MSP430**
6. Make sure the target device is recognized and displayed in the System-on-Chip field
7. Make sure **Retain IEEE address..** field is checked
8. Select the .HEX file you want to program to the target device
9. Select **Erase, Program and Verify**
10. Finally press **Perform actions** and make sure the installation is successful

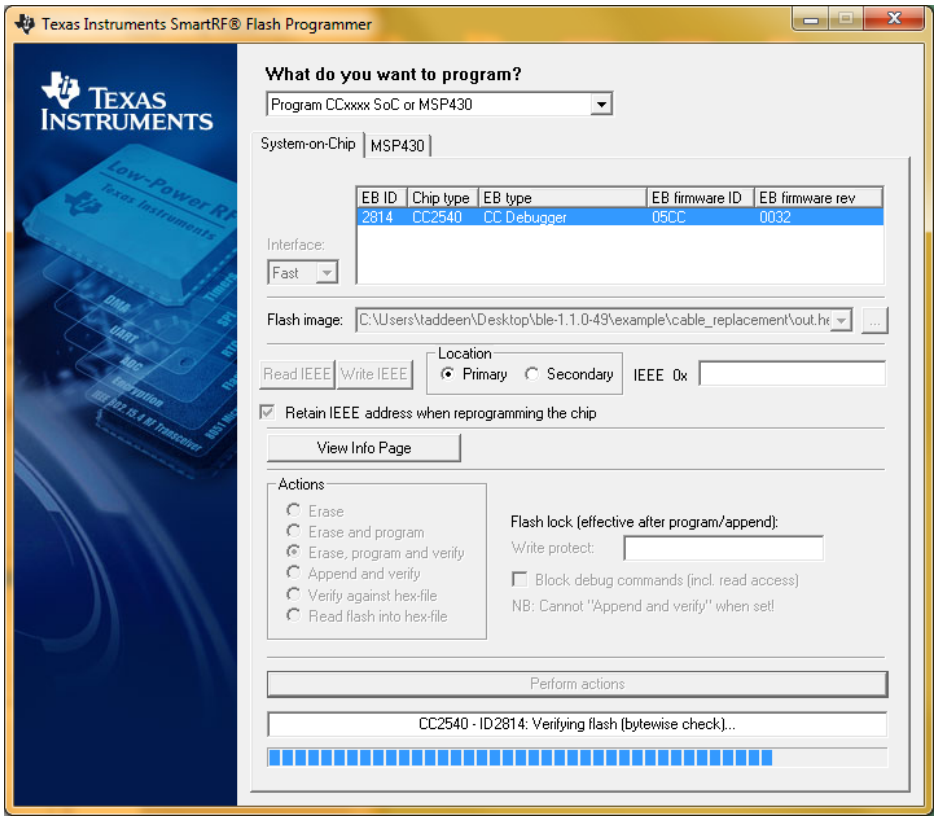

Figure 24: TI's flash programmer tool

Note:

TI Flash tool should **NOT** be used with the Bluegiga *Bluetooth* Smart SDK v.1.1 or newer, but BLE Update tool should be used instead. The BLE112 and BLED112 devices contain a security key, which is needed for the firmware to operate and if the device is programmed with TI flash tool, this security key will be erased.

6.6 Testing the Heart Rate Sensor

6.7 Testing with BLEGUI

This section describes how to test the Heart Rate Sensor application with BLEGUI software.

6.7.1 Discovering the Heart Rate Sensor

As soon as the Heart Rate Sensor is powered on it starts to advertise itself. A BLED112 USB dongle can for example be used to scan for the sensor together with BLEGUI software.

Start ***Generic Scan*** to discover the device.

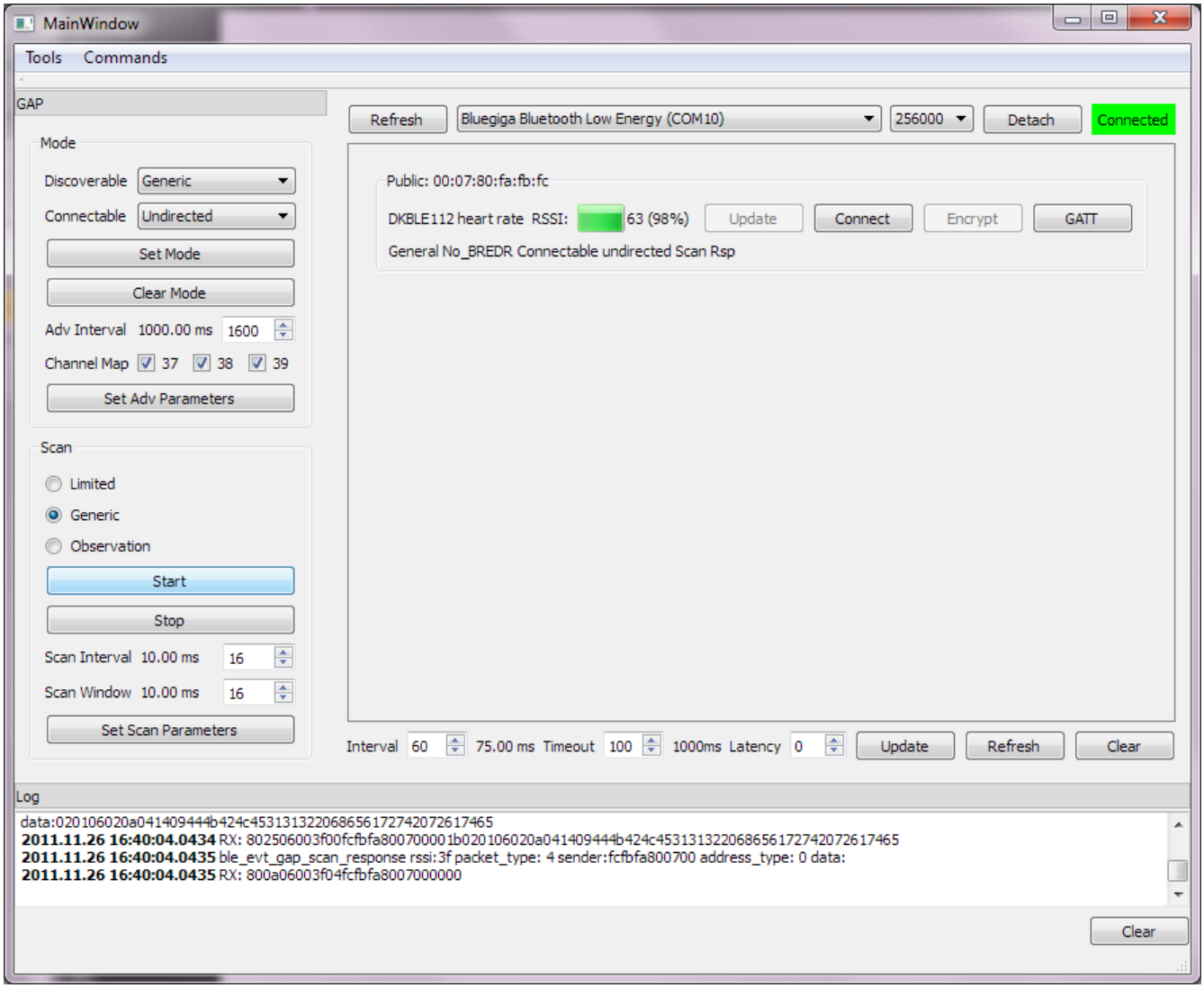

Figure 25: Scanning with BLEGUI

6.7.2 Establishing a Connection

Simply select the ***DKBLE112 heart rate*** device and press the **Connect** button in the BLEGUI user interface.

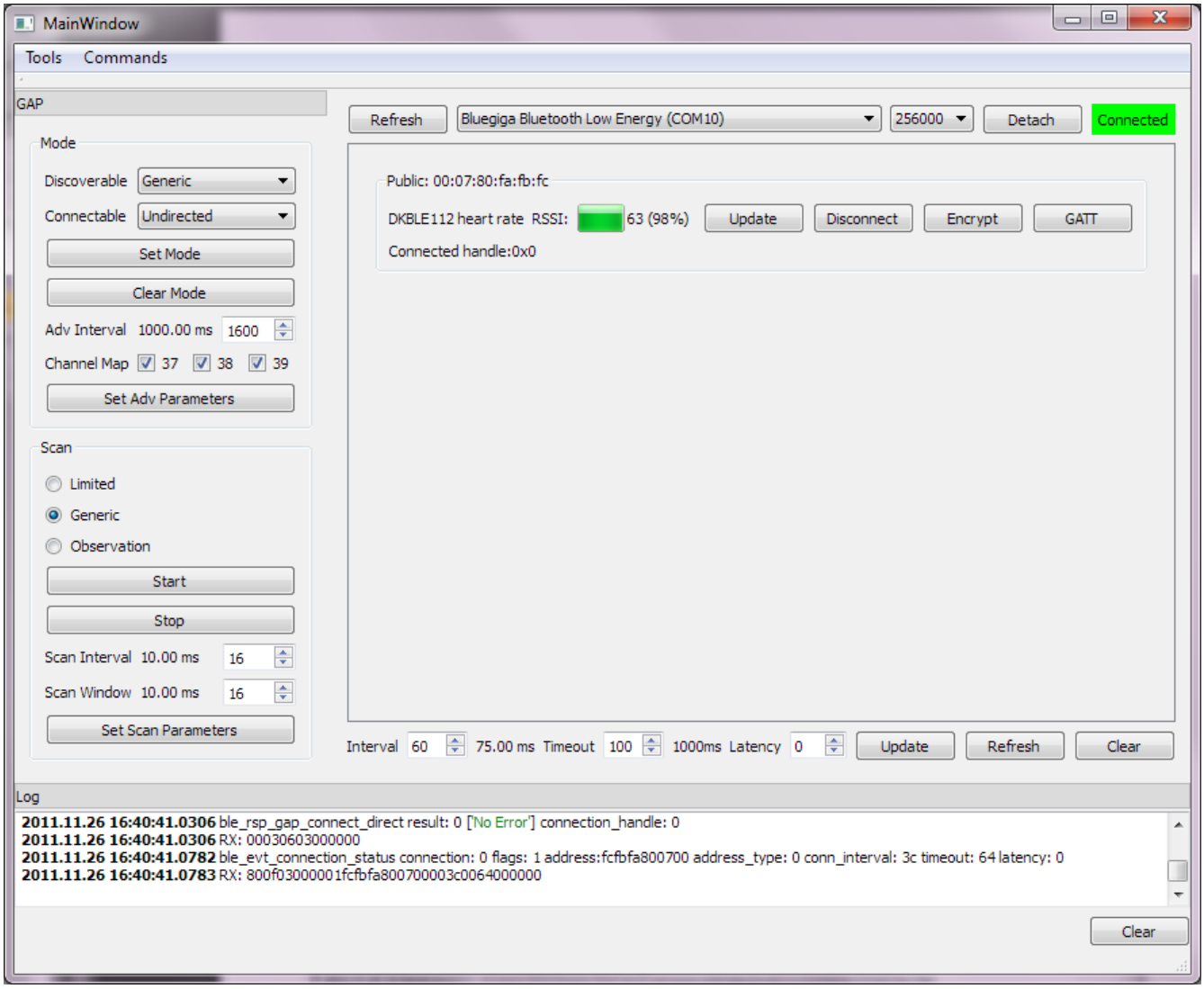

Figure 26: Establishing connection with BLEGUI

6.7.3 Making a Service Discovery

1. Press the ***GATT*** button to start GATT tool
2. Press ***Service Discover*** button to start a GATT primary service discovery procedure

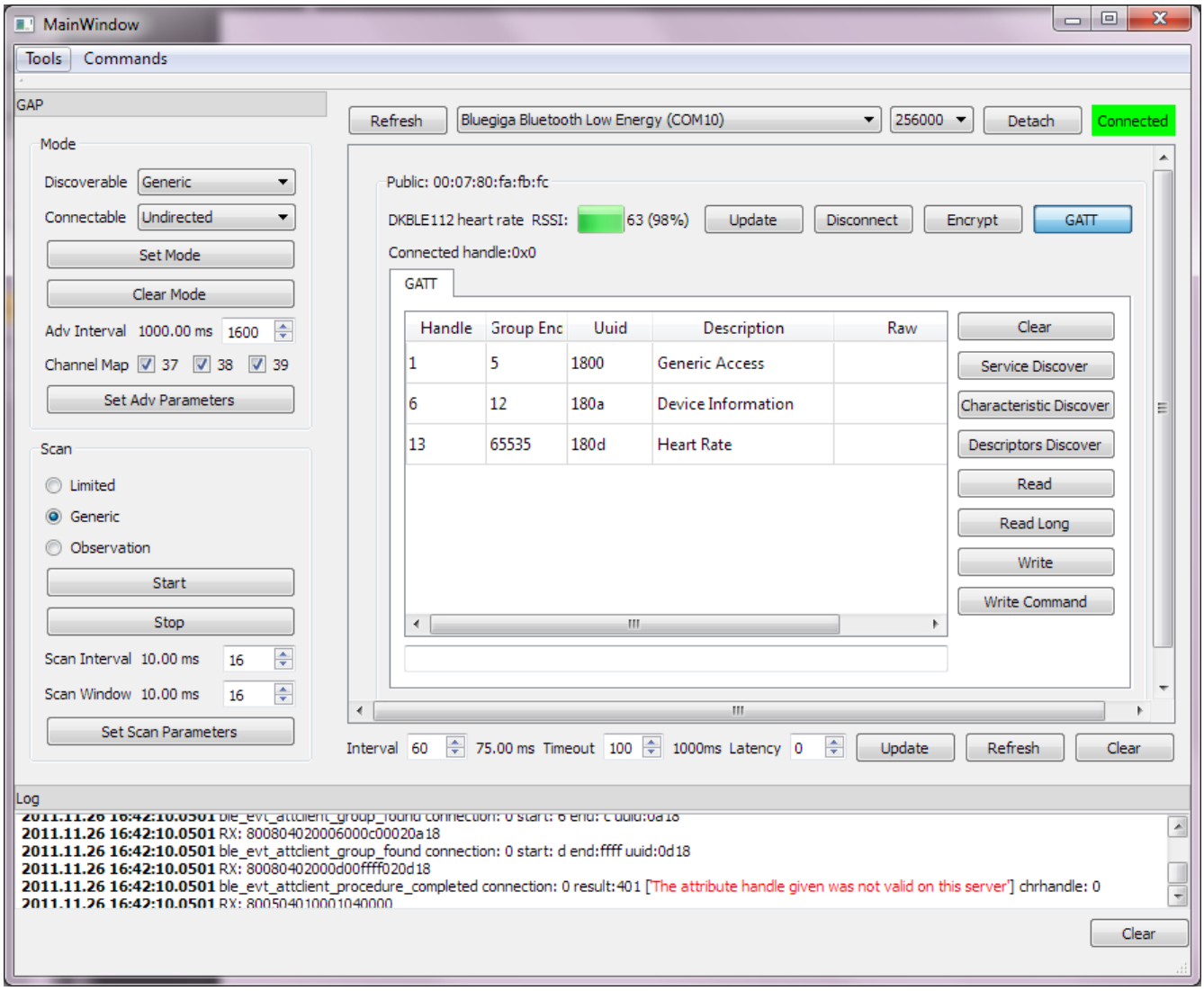

Figure 27: GATT service discovery

The three services defined in the GATT database are visible in the device.

6.7.4 Making a Descriptors Discovery

1. In order to discover the characteristics of the Device Information Service, select the service and press ***Descriptors Discover*** button
2. A list of service descriptors are shown

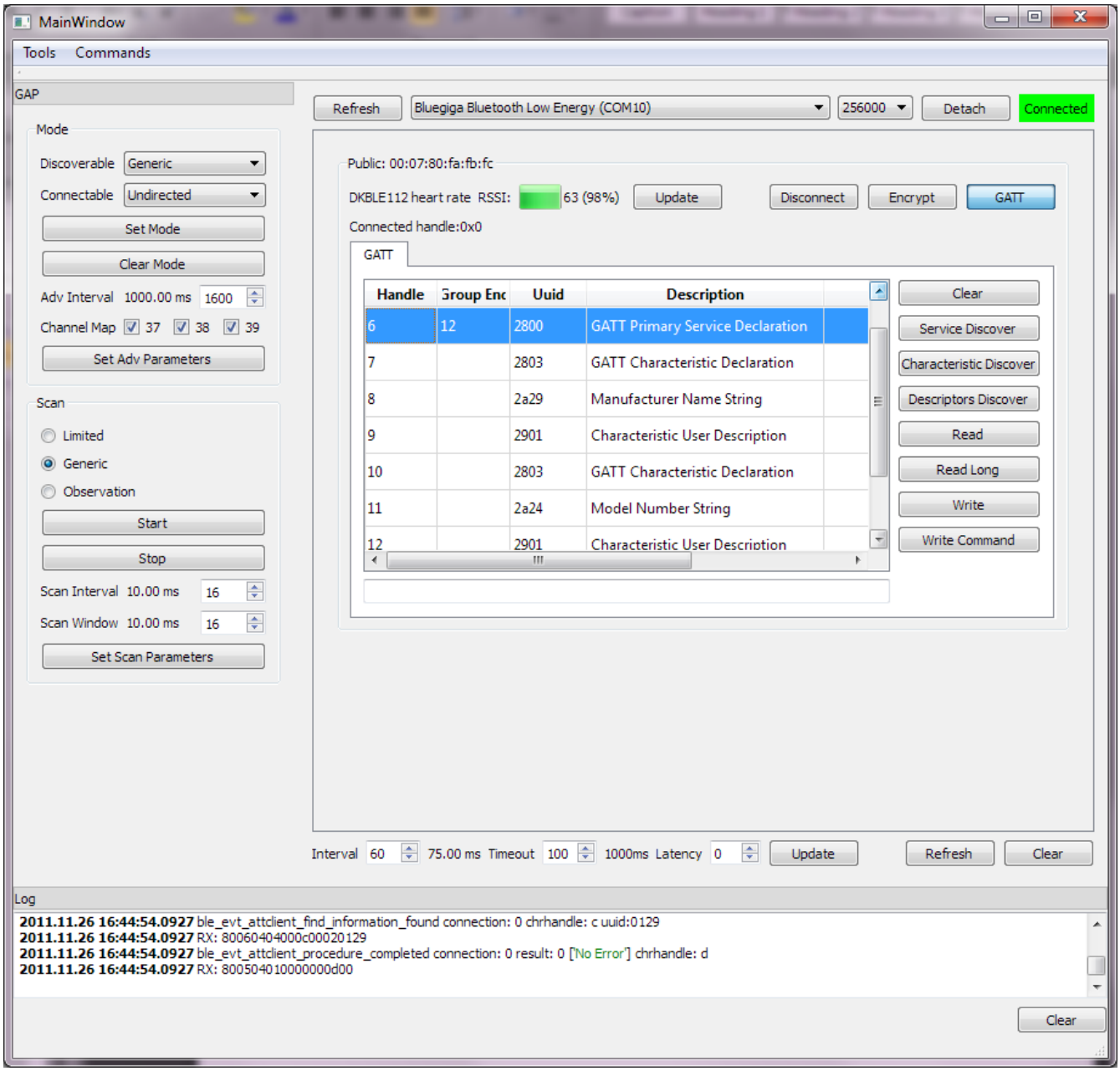

Figure 28: GATT descriptors discovery

6.7.5 Reading a Characteristics Value

1. To read a characteristic value, select the characteristic you are interested in and press the **Read** button. For example the *Manufacturer Name String* has a read property, so the value can be read by a GATT client.

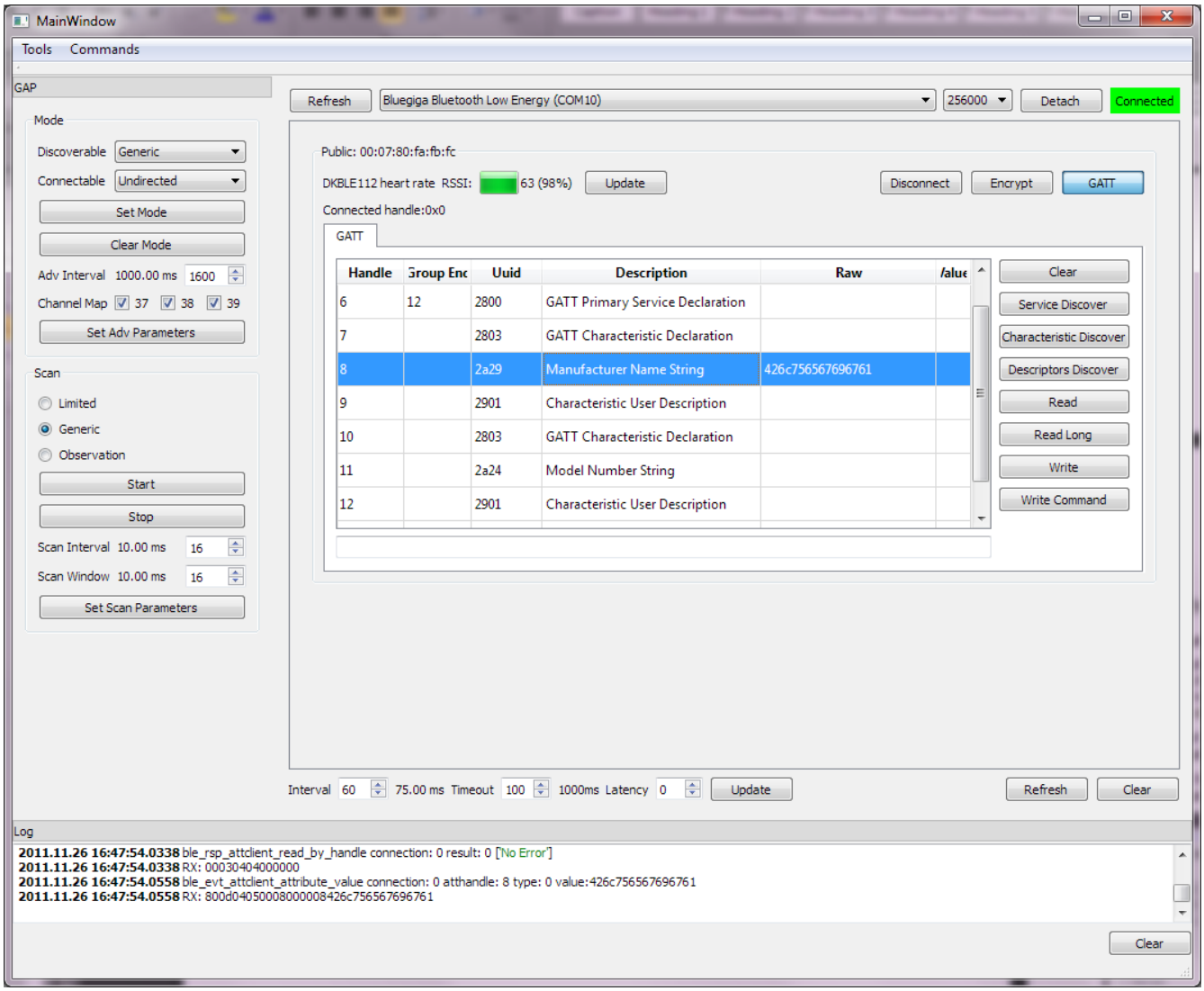

Figure 29: Reading Manufacturer Name String

6.7.6 Starting Notifications for HR Measurement

1. *Heart Rate Measurement* characteristic has a *notify* property
2. This means the Heart Rate Sensor notifies the characteristic value changes to the Heart Rate Collector, instead of the collector having to read it constantly.
3. To enable notifications:
    - Perform descriptors discovery to the Heart Rate Service
    - Write “0x01” to the *Client Characteristic Configuration*

The Heart Rate Sensor starts to notify the HR measurements at 1 second interval

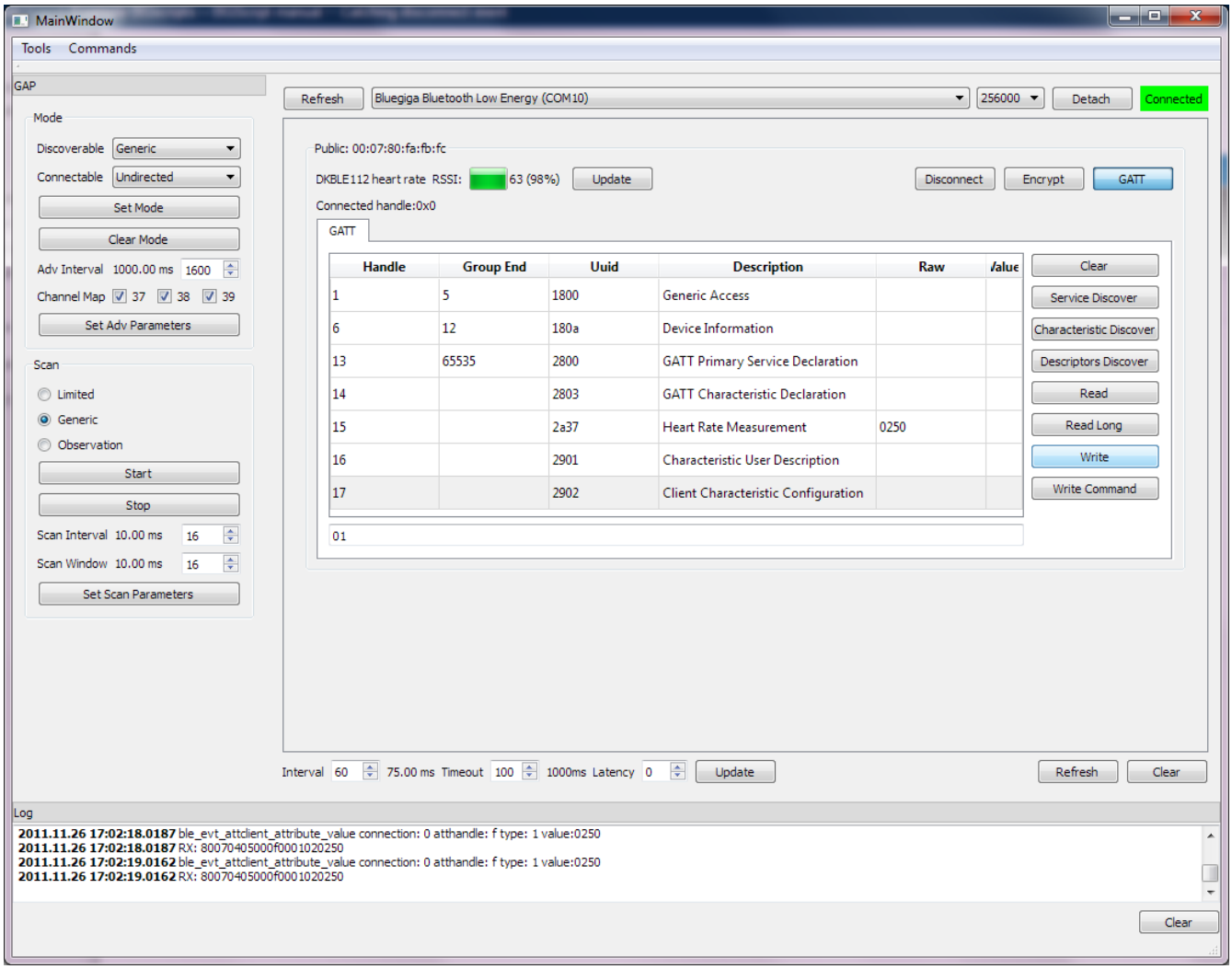

Figure 30: Enabling HR measurement notifications

6.7.7 Terminating the Connection

To terminate the connection:

1. Press the **Disconnect** button

The Heart Rate Sensor restarts the advertisement procedure for the next 60 seconds, until it stops advertisements and goes to Power Mode 3.

After 60 seconds you need to reset the device to restart the advertisements.

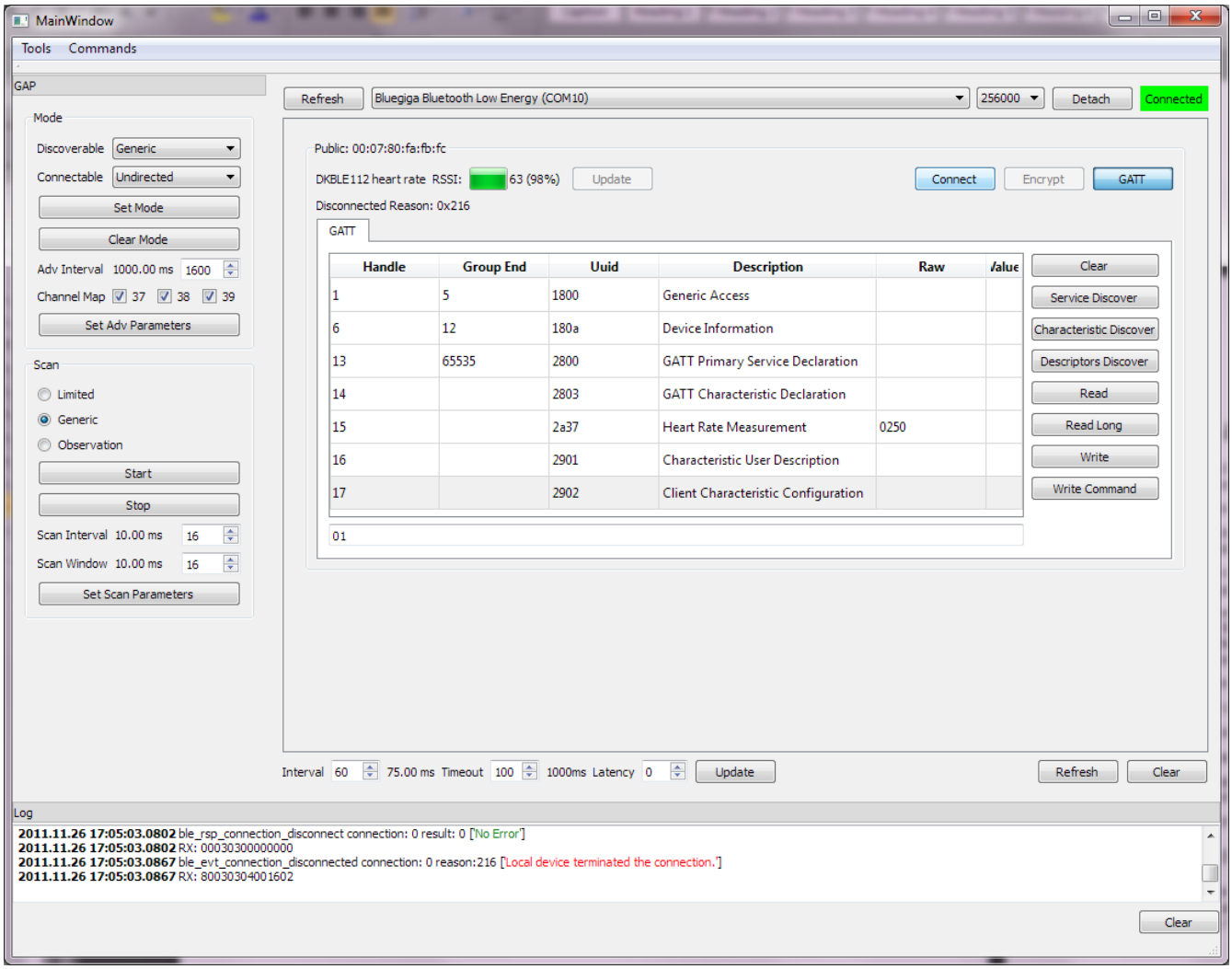

Figure 31: Terminating the connection

7 Debugging Heart Rate sensor code

Debugging BGScript is easiest achieved over the USB interface. You can add debug prints to the BGScript code and then monitor them via the USB interface for example with terminal software.

The enable debugging a few modifications are needed.

First you need to enable USB interface and give the access to it to BGScript. This can be done with the following modifications to the ***hardware.xml*** file.

```
<?xml version="1.0" encoding="UTF-8" ?>

<hardware>
    <sleeposc enable="true" ppm="30" />
    <usb enable="true" endpoint="script" />
    <txpower power="15" bias="5" />
    <script enable="true" />
</hardware>
```

Figure 32: Hardware configuration with USB enabled and access give to BGScript

To print the actual debug messages modifications to the BGScript code is needed. The debug messages are printed to USB with **system_endpoint_tx(…)** command and a few examples are shown below.

```
# System start/boot listener
event system_boot(major,minor,patch,build,ll_version,protocol,hw)

    call system_endpoint_tx(3, 5, "BOOT\n")

    # Device is not connected yet
    connected = 0

    # Set advertisement interval to 20 to 30ms. Use all advertisement channels
    call gap_set_adv_parameters(32,48,7)

    # Start advertisement (generic discoverable, undirected connectable)
    call gap_set_mode(2,2)

    # Start single shot timer with 30 second interval. Handle ID 1
    # 1 second = $8000 (32.768kHz crystal)
    call hardware_set_soft_timer($F0000, 1, 1)
end

# Timer event(s) listener
event hardware_soft_timer(handle)

    # 30 second timer expired
    if handle = 1 then

        call system_endpoint_tx(3, 10, "TIMER 30s\n")

        # No connection
        if connected = 0 then
            # Stop advertisement
            call gap_set_mode(0, 0)
```

Figure 33: BGscript code with debug prints

call system_endpoint_tx(3,5,"BOOT\n") prints the actual debug messages. **3** refers to endpoint USB, **5** means that 5 bytes are written. "**BOOT\n"** is the actual message.

NOTE:

The **"BOOT\n"** message is actually never received by a terminal software when USB interface is used, because the operating system enumerates the USB when the message is being printed. Other messages will however be visible. If UART interface is used, the also the "**BOOT\n**" message can also be received.

Monitoring debug messages can be done with a standard terminal software.

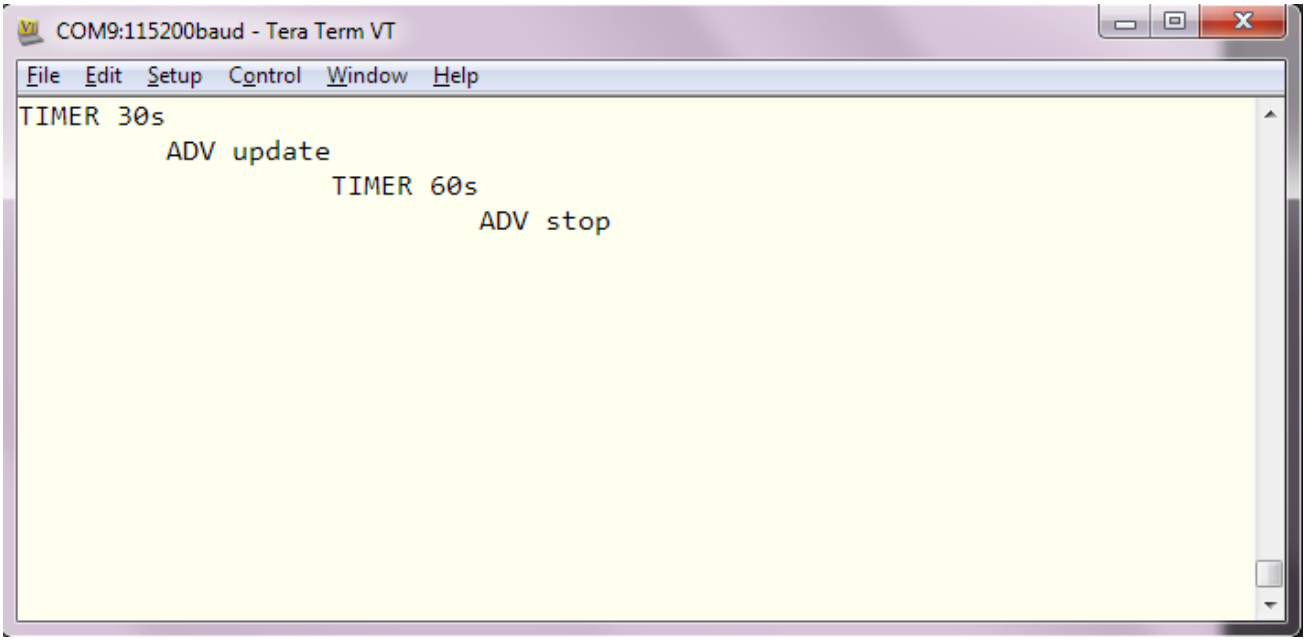

Figure 34: Monitoring debug messages

8 External resources

- *Bluetooth* 4.0 software development kit is available at : www.bluegiga.com
- BLE112 and DKBLE112 hardware documentation is available at : www.bluegiga.com
- Heart Rate Profile can be downloaded from: Heart Rate Profile
- *Bluetooth* SIG's developer portal: https://developer.*Bluetooth*.org/

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 & Community
www.silabs.com/community

SILICON LABS

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

http://www.silabs.com