

BLUETOOTH® 4.0

Developing *Bluetooth* 4.0 single mode applications

Friday, 19 October 2012

Version 1.7

Copyright © 2000-2012 Bluegiga Technologies

All rights reserved.

Bluegiga Technologies assumes no responsibility for any errors which may appear in this manual. Furthermore, Bluegiga Technologies reserves the right to alter the hardware, software, and/or specifications detailed here at any time without notice and does not make any commitment to update the information contained here. Bluegiga's products are not authorized for use as critical components in life support devices or systems.

The WRAP, Bluegiga Access Server, Access Point and iWRAP are registered trademarks of Bluegiga Technologies.

The *Bluetooth* trademark is owned by the *Bluetooth* SIG Inc., USA and is licensed to Bluegiga Technologies. All other trademarks listed herein are owned by their respective owners.

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Project and Hardware configuration added
1.2	BGScript and firmware update instructions added
1.3	Better screen captures and BLEGUI example added
1.4	<i>Bluetooth</i> LE description updated
1.6	Minor updates
1.7	Updated compile and installation instructions

TABLE OF CONTENTS

1	Introduction	5
2	What is <i>Bluetooth</i> low energy technology?	6
3	Typical <i>Bluetooth</i> 4.0 application architecture	7
3.1	Description	7
3.2	Profile	7
3.3	Service	7
3.4	Characteristic	8
3.5	Relationship between profiles, services and characteristics	8
4	Bluegiga's <i>Bluetooth</i> 4.0 single mode stack suite	9
4.1	<i>Bluetooth</i> 4.0 single mode stack	9
4.2	BGAPI protocol	10
4.3	BGLib library	11
4.4	BGScript™ scripting language	12
4.5	Profile toolkit	13
5	Implementation of "BGDemo" sensor	14
5.1	Creating a project	15
5.2	Hardware configuration	16
5.3	Building a GATT database with Profile Toolkit	17
5.3.1	Profile template	17
5.3.2	Generic Access Profile service	18
5.3.3	Battery service	19
5.3.4	A manufacturer specific service	20
5.4	Writing BGScript code	21
5.5	Compiling and installing the firmware	23
5.5.1	Using BLE Update tool	23
5.5.2	Compiling using the bgbuild.exe	25
5.5.3	Installing the firmware with TI's flash tool	26
6	Testing the BGDemo sensor	27
6.1	Using BLEGUI	27
6.1.1	Discovering the BGDemo sensor	27
6.1.2	Establishing the connection	28
6.1.3	Making GATT service discovery	29
6.1.4	Reading battery status	30
6.1.5	Notifying battery status	32
6.2	Reading and writing the manufacturer proprietary service	33
6.3	Writing application using BGLib C-library	34
7	Contact information	35

1 Introduction

This application note discusses how to build *Bluetooth* 4.0 applications using Bluegiga's BLE112 and BLED112 single mode products. The application note contains a practical example of how to build GATT based services with the profile toolkit, how to make a standalone sensor device using BGScript, and how to use the BGAPI binary protocol to connect to a standalone sensor and transmit data.

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 in 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 Description

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.

The figure below shows the relationship of these two roles.

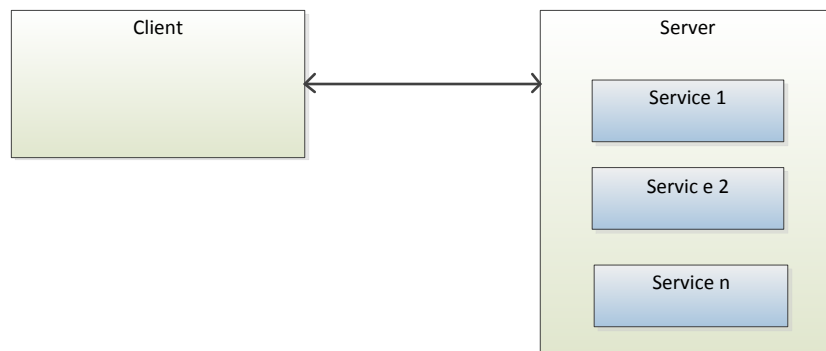


Figure 1: *Bluetooth* low energy device roles

3.2 Profile

Profile is a collection of services. A profile may implement single or multiple services.

Profiles are defined in profiles specifications, which are available at <http://developer.bluetooth.org>

3.3 Service

Service is a collection of characteristics and optionally references to other services. 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.

Services are defined in service specifications, which are available at <http://developer.bluetooth.org>.

3.4 Characteristic

Services consist of one or several characteristics. Characteristic is a value, with a known type, and a known format.

Characteristics are defined in “Characteristic Specification” available at <http://developer.bluetooth.org>.

Characteristics consist of:

- **Characteristic Declaration**

Describes the properties of *characteristic value* (read, write, indicate etc.), *characteristic value* handle and *characteristic value* type (UUID)

- **Characteristic Value**

Contains the value of a characteristic (for example temperature reading)

- **Characteristic Descriptor(s)**

Provide additional information about the characteristic (characteristic user description, characteristic client configuration, vendor specific information etc.).

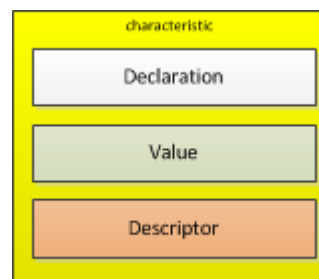


Figure 2: Characteristic structure

3.5 Relationship between profiles, services and characteristics

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

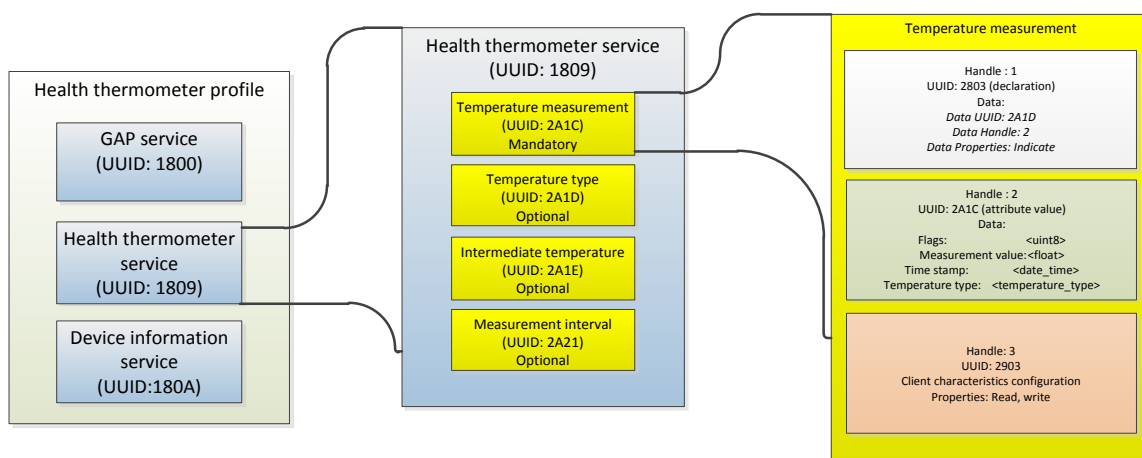


Figure 3: Health thermometer profile

4 Bluegiga's Bluetooth 4.0 single mode stack suite

Bluegiga's *Bluetooth* 4.0 single mode stack suite provides a complete development framework for *Bluetooth* low energy application implementers.

The *Bluetooth* 4.0 single mode stack suite framework supports two architectural modes:

- All software including: *Bluetooth* 4.0 single mode stack, profiles and end user application all run on the Bluegiga's *Bluetooth* 4.0 single mode hardware
- The *Bluetooth* 4.0 single mode stack and profiles run on the Bluegiga 4.0 single mode hardware but the end user application runs on a separate host (a micro controller for example)

The benefits of the development suite in either of the use cases is that it provides a complete *Bluetooth* 4.0 single mode stack so that no *Bluetooth* development is required, a well-defined transport protocol exists between the host and the *Bluetooth* hardware and also simple development tools are available for embedding the end user applications on the *Bluetooth* 4.0 single mode hardware.

The *Bluetooth* 4.0 single mode development suite consists of several components:

- A *Bluetooth* 4.0 single mode stack
- Binary based communication protocol (called BGAPI) between the host and the *Bluetooth* stack
- A C library (called BGLib) for the host that implements the BGAPI protocol
- BGScript™ scripting language and interpreter for implementing applications on the *Bluetooth* 4.0 single mode hardware
- A Profile Toolkit for quick and easy development of GATT based *Bluetooth* services

4.1 Bluetooth 4.0 single mode stack

The *Bluetooth* 4.0 single mode stack is a full, embedded implementation of *Bluetooth* v.4.0 compatible stack software and it's dedicated for Bluegiga's *Bluetooth* 4.0 single mode modules. The stack implements all mandatory functionality for a single mode device. The structure and layers of the stack are illustrated in the figure below.

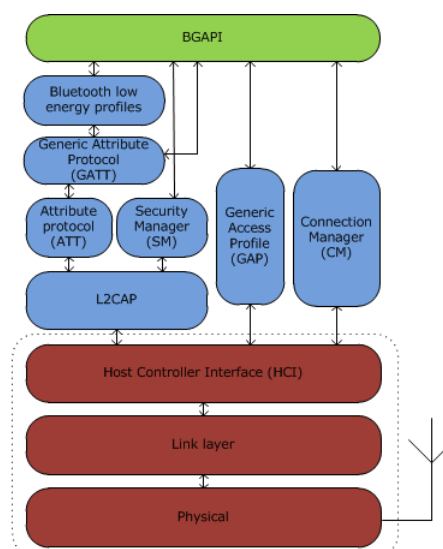


Figure 4: *Bluetooth* 4.0 single mode stack

4.2 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 binary based communication protocol designed specifically for ease of implementation within host devices with limited resources.

The BGAPI provides access to the following layers:

- **Generic Access Profile** - GAP allows the management of discoverability and connectability modes and open connections
- **Security manager** - Provides access to the *Bluetooth* low energy security functions
- **Attribute database** - An interface 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

The BGAPI protocol is intended to be used with:

- a serial UART link or
- a USB connection or
- a SPI connection

4.3 BGLib 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* 4.0 single mode energy development suite. The purpose is to simplify the application development to various host environments.

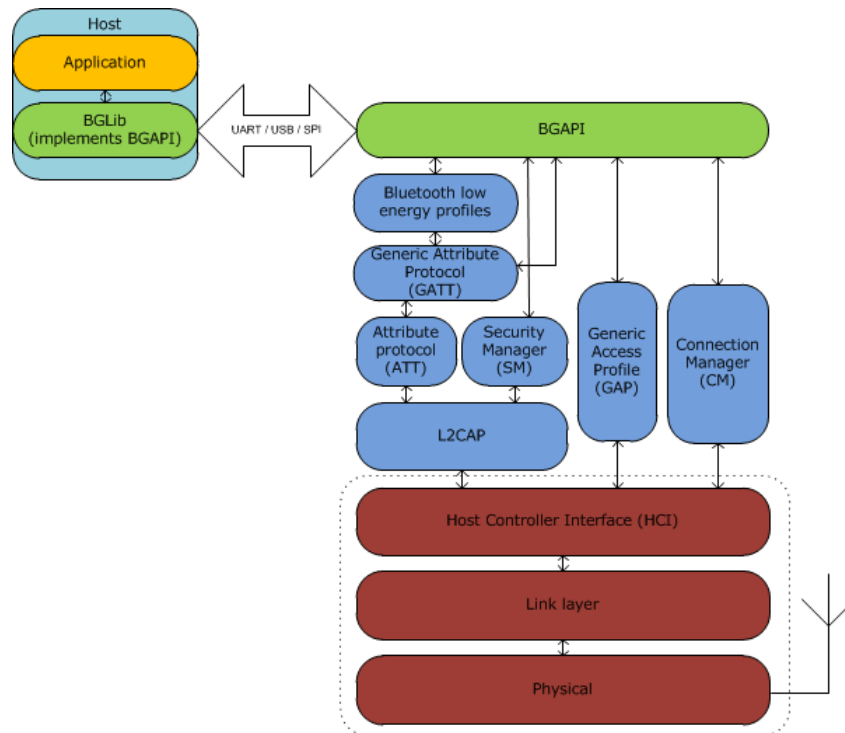


Figure 5: BGLib host library

4.4 BGScript™ scripting language

Bluegiga's *Bluetooth* 4.0 single mode energy products allow application developers to create standalone devices without the need of a separate host. The *Bluetooth* low energy modules can run simple applications along the *Bluetooth* 4.0 single mode stack and this provides a benefit when one needs to minimize the end product size, cost and current consumption. For developing standalone *Bluetooth* low energy applications the development suite provides a simple BGScript scripting language. With BGScript provides access to the same software and hardware interfaces as the BGAPI protocol. The BGScript code can be developed and compiled with free tools provided by Bluegiga.

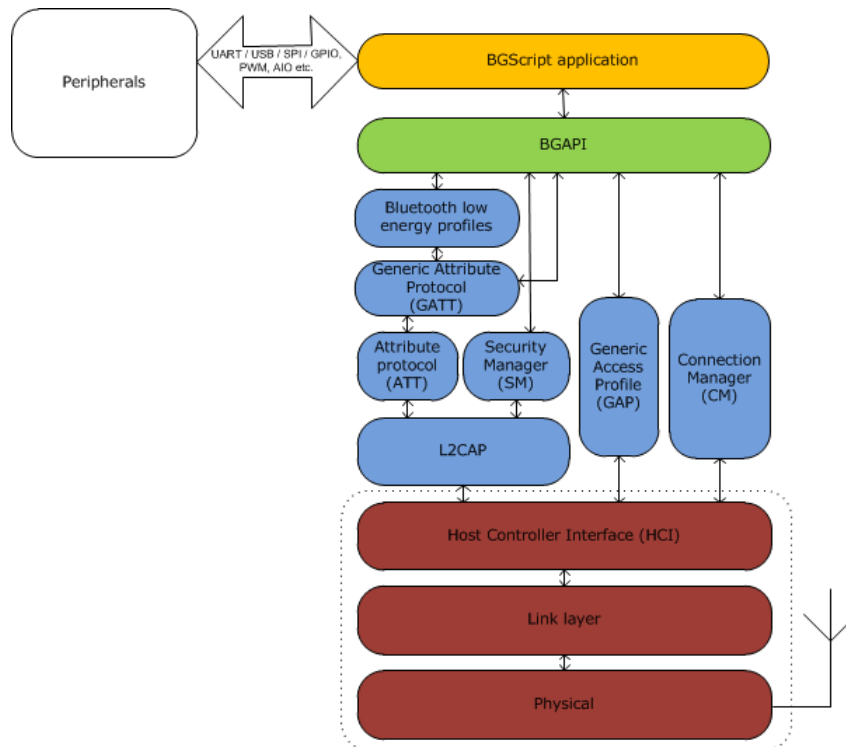


Figure 6: BGScript application model

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
```

4.5 Profile toolkit

The *Bluetooth* low energy profile toolkit a simple set of tools, which can used to create GATT based *Bluetooth* services. The profile toolkit consists of a simple XML based service description language template, which describes the devices local GATT database as a set of services. 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 7: A profile toolkit example of GAP service

5 Implementation of “BGDemo” sensor

In this chapter we discuss an actual implementation of a standalone *Bluetooth* low energy sensor called “BGDemo”. The implementation consists of following steps:

1. Starting the project
2. Defining hardware configuration
3. Building a GATT based service database with profile toolkit
4. Writing a simple BGScript that defines the sensors functionality
5. Compiling the GATT data base and BGScript into a binary firmware
6. Installing the firmware into BLE112 or BLED112 hardware

5.1 Creating a project

The project is started by creating a project file. The file is a simple XML document and defines all the other files the included in the project. An example of a complete project file is shown below:

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

Figure 8; Project file

- The project configuration is described within the `<project>` tags
- `<gatt>` tag defines the .XML file containing the GATT data base description
- `<hardware>` tag defines the .XML file containing the hardware configuration
- `<script>` tag defines the .BGS file containing the BGScript code. If the project does not contain a BGScript code, this tag can be simply left out.
- `<usb_main>` tag defines the .XML file containing the USB descriptors description. If the project does not use USB interface, this tag can be simply left out.
- `<image>` tag defines the output .HEX file containing the firmware image

WARNING:

If the firmware is to be installed into the BLED112 USB dongle the USB CDC configuration **MUST BE** included in the project file. If this is not included in the project file and the compiled firmware is installed into the BLED112 USB dongle, the USB interface will be disabled and the dongle stops from working.

NOTE:

For applications targeted for BLE112 module, the USB should be disabled as the USB interface will continually draw around 1mA of power.

5.2 Hardware configuration

The hardware configuration of the BLE112 is described in the hardware configuration file. If the default project template is used, the file is called **hardware.xml**. An example of a hardware configuration used in BGDemo application is shown below.

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

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

Figure 9: Example of hardware.xml file for BLE112 USB dongle

- The hardware configuration is described within the `<hardware>` tags
- `<sleeposc>` tag defines whether the sleep oscillator is enabled or not. The Sleep oscillator allows low power sleep modes to be used. The BLE112 USB dongle does not incorporate the sleep oscillator so the value is set to false. PPM defines the accuracy of the sleep oscillator.
- `<usb>` tag defines if USB is to be enabled or not. This MUST always be enabled for the BLE112 USB dongle. `endpoint="API"` defines that the USB interface is reserved for the BGAPI (binary protocol) usage.
- `<txpower>` tag defines the TX power level.

The example below reveals the hardware configuration for BLE112 module used in the demo.

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

<hardware>
    <sleeposc enable="true" ppm="30" />
    <usb enable="false" endpoint="none" />
    <txpower power="15" bias="5" />
    <usart channel="0" alternate="2" baud="57600" endpoint="api" />
    <wakeup_pin enable="true" port="1" pin="1" />
</hardware>
```

Figure 10: Example of hardware.xml file for BLE112 module

- BLE112 module includes the sleep oscillator so it should be enabled in the hardware configuration file.
- To minimize the power consumption the USB interface is disabled.
- The UART0 interface is enabled with 57600bps baud rate and reserved for BGAPI usage.
- Sleep mode 3 wake-up pin is enabled and port 1, pin 1 is used for that.

5.3 Building a GATT database with Profile Toolkit

This section discusses the implementation of a GATT service database. The service database is created with the profile toolkit, which is a simple XML based description language used to describe *Bluetooth* low energy profiles (service) and their characteristics and access rights.

5.3.1 Profile template

When you start to build the GATT service database, a simple template is needed. An example of such a template is shown in Figure 11 below. An empty template only needs to have a line stating the version number and `<configuration>` tag, inside which the actual services are described.

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

Figure 11: Profile template

5.3.2 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.

```
<?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 12: GAP service

Service description:

A service is described within `<service uuid="nnnn">` and `</service>` tags. All services defined by *Bluetooth* SIG use fixed 16-bit UUID's and the UUID assigned for GAP service is 1800.

In the example above an optional `<description>` tag is used to identify the service name. This is optional tag and can be considered to be a comment.

Characteristics description:

Service characteristics are described within `<characteristic uuid="nnnn">` and `</characteristic>` tags. A service may have one or more characteristics. The GAP service used as an example contains two characteristics, which are:

- **Device name, UUID: 2a00**

This is a device's user friendly name and is similar to the friendly name used in *Bluetooth* BR/EDR.

- **Device appearance, UUID: 2a01**

This identifies the devices type and is similar to class-of-device used in *Bluetooth* BR/EDR.

The characteristics also have UUIDs and the standardized *Bluetooth* low energy services use globally unique UUIDs for characteristics.

Characteristics properties:

Service characteristics are described using the `<properties>` tag. The properties define how the characteristic can be accessed by a remote device. In the GAP service both the values are read only. Since the values are read only they can be marked as `const` meaning the values are constant. This will reduce the amount of memory the compiled GATT data base will use.

Characteristics values:

The characteristic's value is defined within the `<value>` and `</value>` tags. The device appearance is a hex value, so `hex` flag is used.

5.3.3 Battery service

The second service implemented in our example is the battery service. The battery service exposes the state of a battery connected to a device. This example implementation follows the actual battery service being implemented by the *Bluetooth* SIG, but only implements the mandatory characteristics. The battery service description is very similar to the GAP service and has only a few differences. The XML description is shown below.

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

  <service uuid="1800">
    <service uuid="e001">
      <description>Battery status</description>
      <characteristic uuid="e101" id="xgatt_battery">
        <properties read="true" notify="true" />
        <value type="hex">ABCD</value>
      </characteristic>
    </service>
  </configuration>
```

Figure 13: Battery status service

The global UUID for battery status service is: e0001

The global UUID for battery level characteristic is: e101

- This is the only mandatory characteristic in battery service
- The battery level tells the battery level in mV
- Battery level can be read or notified
- The value is in hex

With BGScript we want to access this value and update the battery level so an `id` tag is used in the characteristic description. The id `xgatt_battery` can be used by BGScript to easily access the characteristic and update its value.

5.3.4 A manufacturer specific service

The third service used in our example is a manufacturer specific service. *Bluetooth* low energy devices can have services defined by manufacturers and are not standardized by the *Bluetooth* SIG. The service structure is exactly the same however, manufacturer specific services have to use 128-bit long UUIDs instead of the 16-bit UUIDs used by the standardized services.

The 128-bit UUIDs do not need to be applied for with the *Bluetooth* SIG, but can be generated using online tools such as this site: <http://uuidgenerator.net/>

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

  <service uuid="1800">


---


  <service uuid="e001">


---


    <service uuid="00431c4a-a7a4-428b-a96d-d92d43c8c7cf">
      <description>Bluegiga demo service</description>
      <characteristic uuid="f1b41cde-dbf5-4acf-8679-ecb8b4dca6fe">
        <properties read="true" write="true"/>
        <value type="hex">coffee</value>
      </characteristic>
    </service>
  </configuration>
```

Figure 14: Custom service

The example service above has one characteristic which can be either read or written.

5.4 Writing BGScript code

The example implements a standalone sensor device where no external host processor is needed. The sensor side application is created with Bluegiga's BGScript scripting language and the code is shown below.

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.

Our example BGDemo application uses the following BGScript code.

```
# 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

# Timer expired
event hardware_soft_timer(handle)
    # Start ADC read
    call hardware_adc_read(15,3,0)
end

# ADC read complete
event hardware_adc_result(input, value)
    #adc read complete, write result to attribute
    call attributes_write(xgatt_battery,2,value)
end

# Disconnection event
event connection_disconnected(handle, result)
    #connection disconnected, continue advertising
    call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
end
```

The BGScript has four event listeners defined:

- `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.

- `hardware_soft_timer(...)` **event listener**

When the timer expires this event is generated. In our example the timer is used to read the ADC connected to the battery every second.

- `hardware_adc_result(...)` **event listener**

The ADC read function generates an ADC event, which this event listener captures. When the ADC value is received, it's written to the GATT databases battery service.

- `connection_disconnected(...)` **event listener**

The last event handler is executed when a *Bluetooth* low energy connection is lost or closed by the remote device and simply puts the device back to advertisement mode.

The BGScript functions and events can be found from the *Bluetooth* low energy stack API reference document.

5.5 Compiling and installing the firmware

5.5.1 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 BLE112 module using a CC debugger.

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
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.xml** file
7. Press **Update**

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

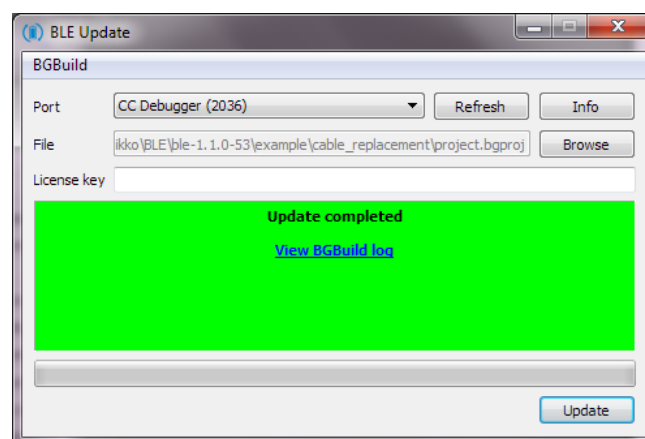


Figure 15: Compile and install with BLE Update tool

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

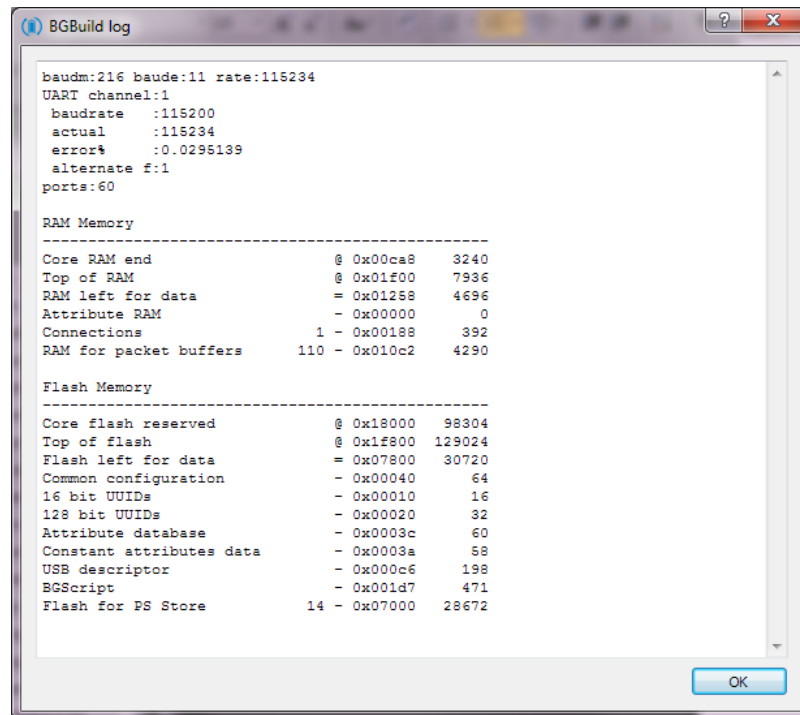


Figure 16: BLE Update build log

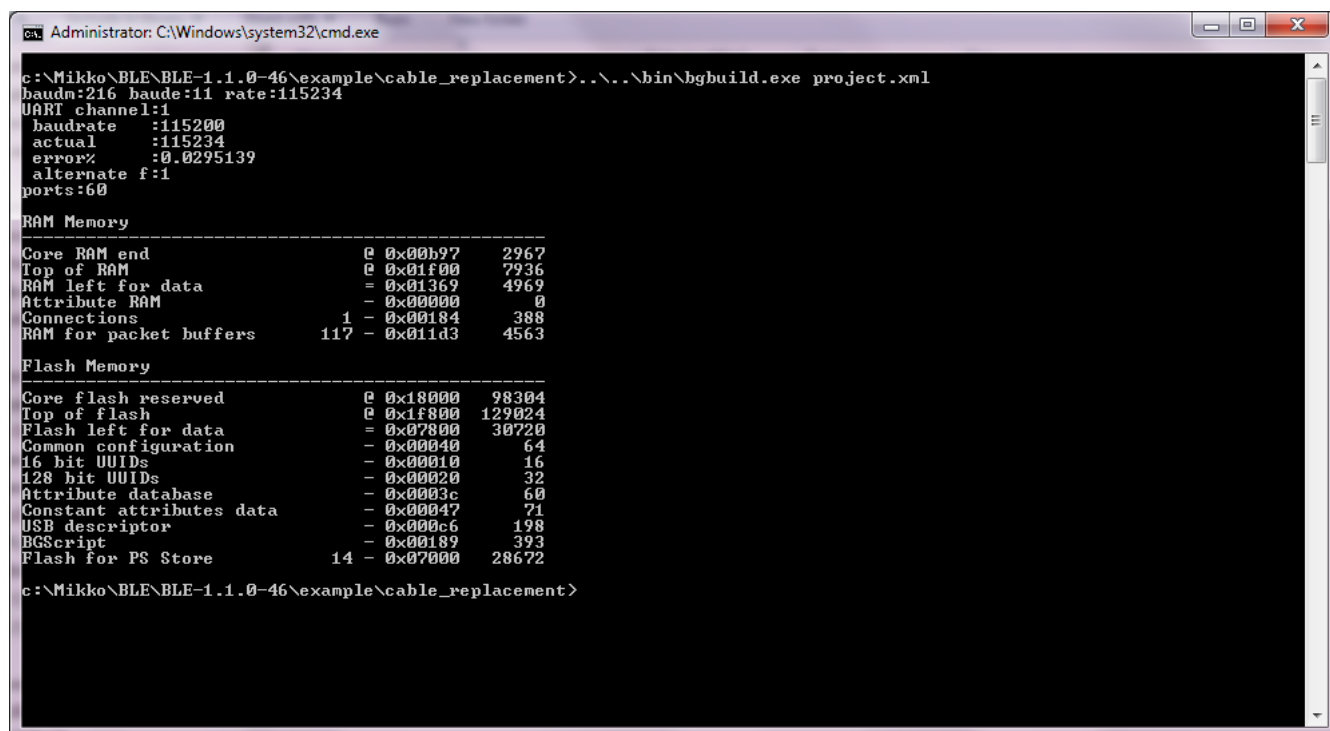
5.5.2 Compiling using the bgbuild.exe

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.

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>*



```
Administrator: C:\Windows\system32\cmd.exe

c:\Mikko\BLE\BLE-1.1.0-46\example\cable_replacement>..\..\bin\bgbuild.exe project.xml
baudm:216 baudr:11 rate:115234
UART channel:1
  baudrate :115200
  actual   :115234
  error%   :0.0295139
  alternate f:1
ports:60

RAM Memory
-----
Core RAM end          @ 0x00b97 2967
Top of RAM            @ 0x01f00 7936
RAM left for data     = 0x01369 4969
Attribute RAM         - 0x00000 0
Connections           1 - 0x00184 388
RAM for packet buffers 117 - 0x011d3 4563

Flash Memory
-----
Core flash reserved   @ 0x18000 98304
Top of flash          @ 0x1f800 129024
Flash left for data   = 0x07800 30720
Common configuration  - 0x00040 64
16 bit UUIDs          - 0x00010 16
128 bit UUIDs         - 0x00020 32
Attribute database    - 0x0003c 60
Constant attributes data - 0x00047 71
USB descriptor        - 0x000c6 198
BGScript              - 0x00189 393
Flash for PS Store    14 - 0x07000 28672

c:\Mikko\BLE\BLE-1.1.0-46\example\cable_replacement>
```

Figure 17: Compiling with BGBuild.exe

If the compilation is successful a .HEX file is generated, which can be installed into a BLE112 module.

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

5.5.3 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

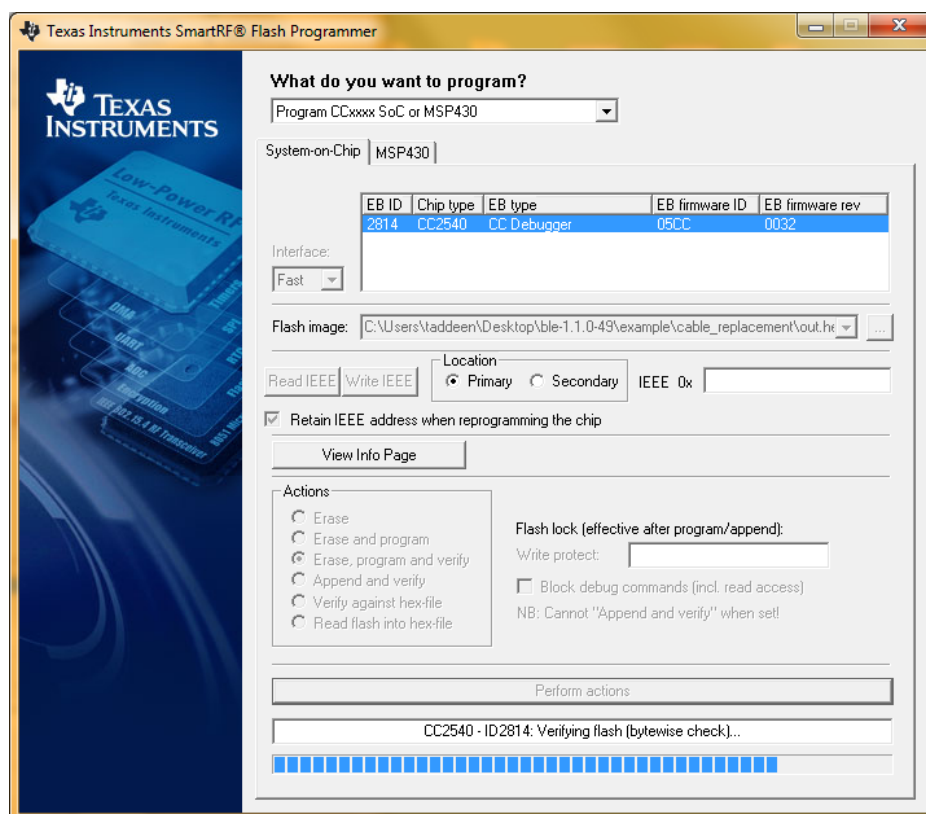


Figure 18: TI's flash programmer tool

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 BLE112 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 Testing the BGDemo sensor

6.1 Using BLEGUI

This section describes how to test the BGDemo sensor implementation using BLEGUI software.

6.1.1 Discovering the BGDemo sensor

As soon as the BGDemo sensor is powered on it starts to advertise. A BLED112 USB dongle can for example be used to scan for the sensor.

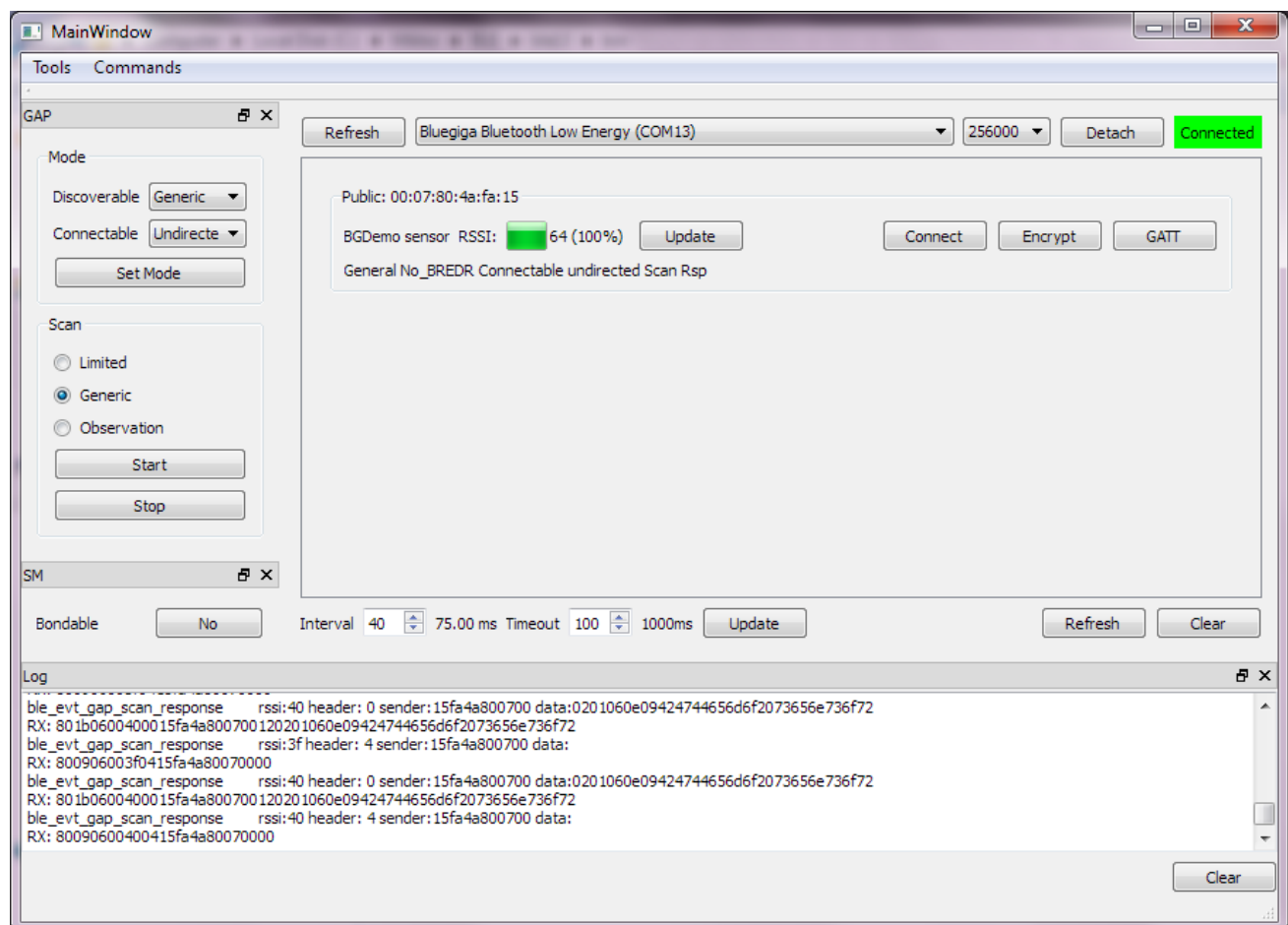


Figure 19: Scanning with BLEGUI

6.1.2 Establishing the connection

Simply select the BGDemo sensor device and press the connect button in the BLEGUI user interface.

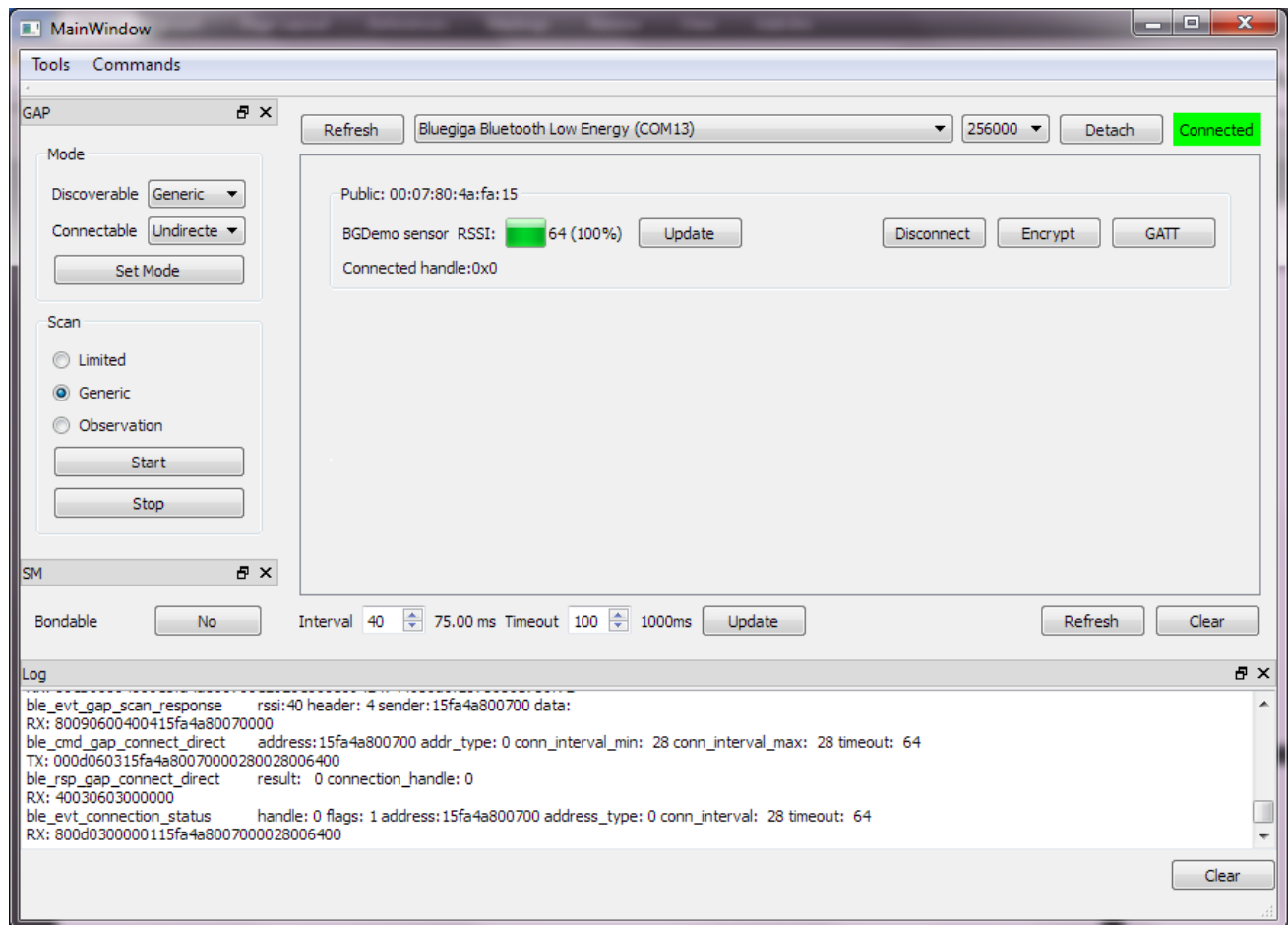


Figure 20: Connecting with BLEGUI

6.1.3 Making GATT service discovery

- Press the **GATT** button to start GATT tool
- Press **Service discover** button to start a GATT primary service discovery procedure

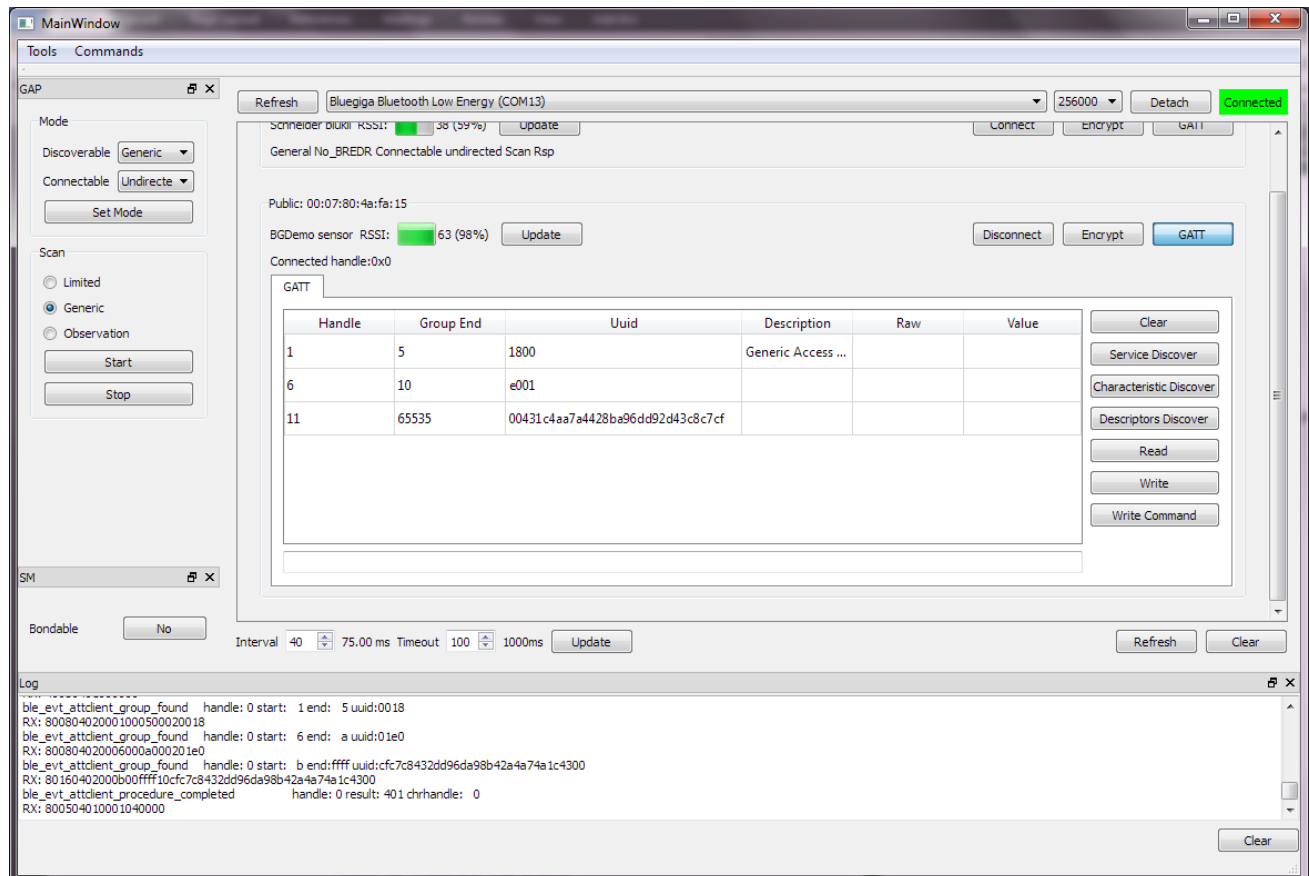


Figure 21: GATT service discovery

The three services defined in the GATT data base are visible in the device.

6.1.4 Reading battery status

- To read the battery status, simply select the Battery service (UUID: e001) and make **Descriptors discovery**
- A list of service descriptors are shown

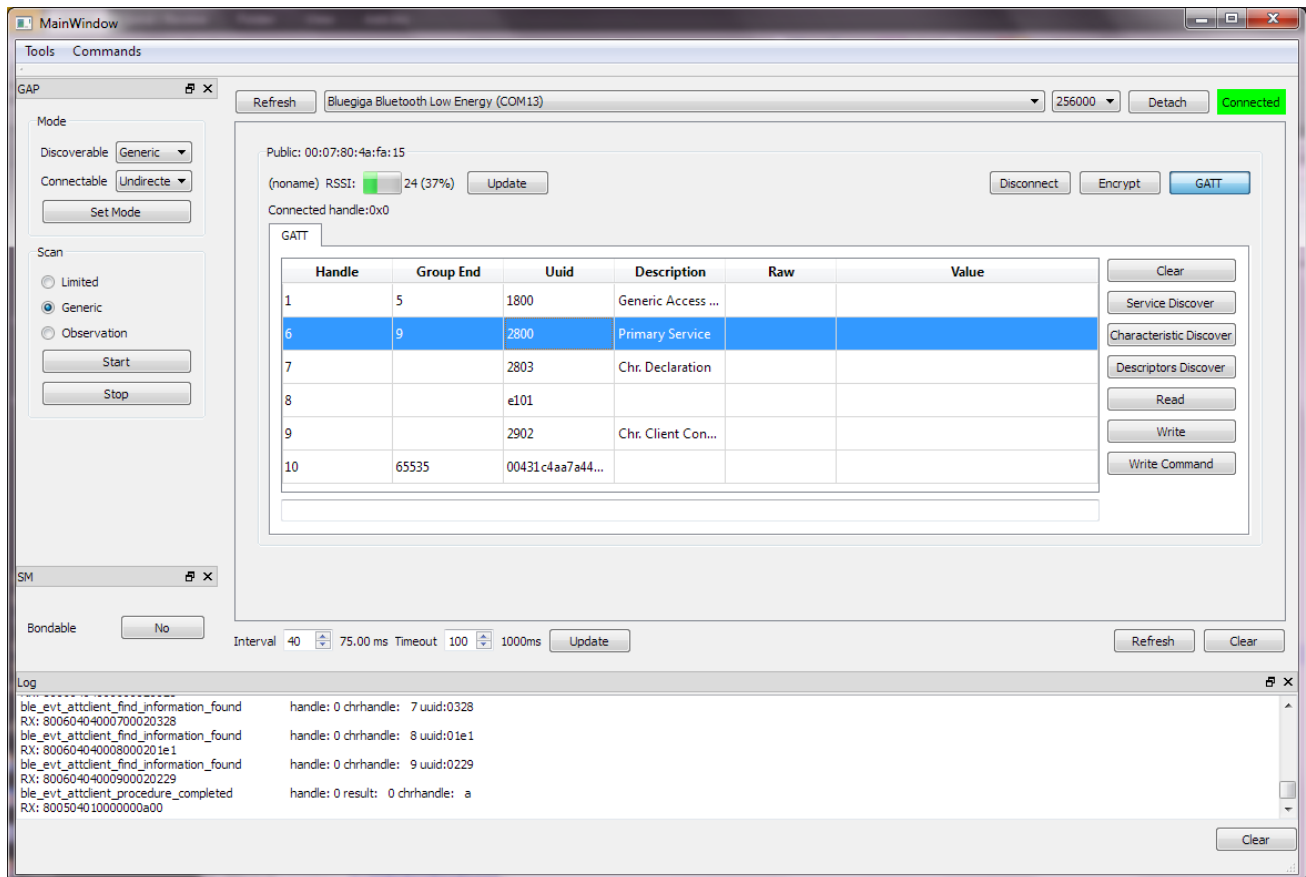


Figure 22: GATT descriptor discovery

- The battery level value is stored in the UUID e001. The reads the value, simply select UUID e001 and press the **Read** button. Every time the read operation is made the battery level value is updated.

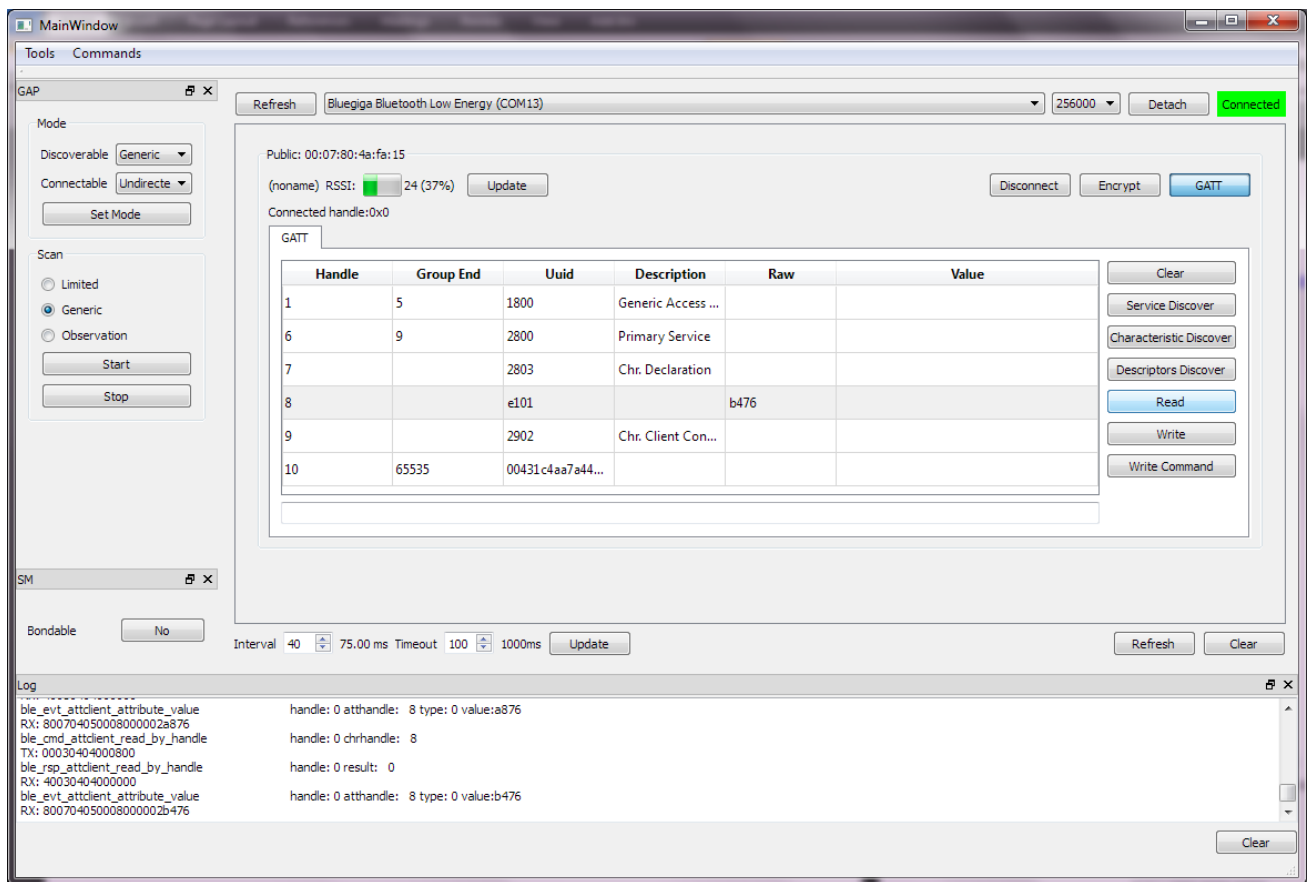


Figure 23: Reading characteristic values

6.1.5 Notifying battery status

- The battery status value can also be indicated as in the GATT data base **notify**="true" option was set. The notification makes the server automatically to send the characteristic value to the client, when the value changes. In the BGDemo example the battery value is updated once a second by the BGScript code. However the server can only indicate the value change to the client at the connection anchor points defined by the connection interval.
- To start the notifications the client needs to enable them in the server. This is done by enabling the notifications in the characteristic client configuration.
- Select the UUID 2902 (characteristic client configuration) and **Write** its value to 0100. This will star the notifications.

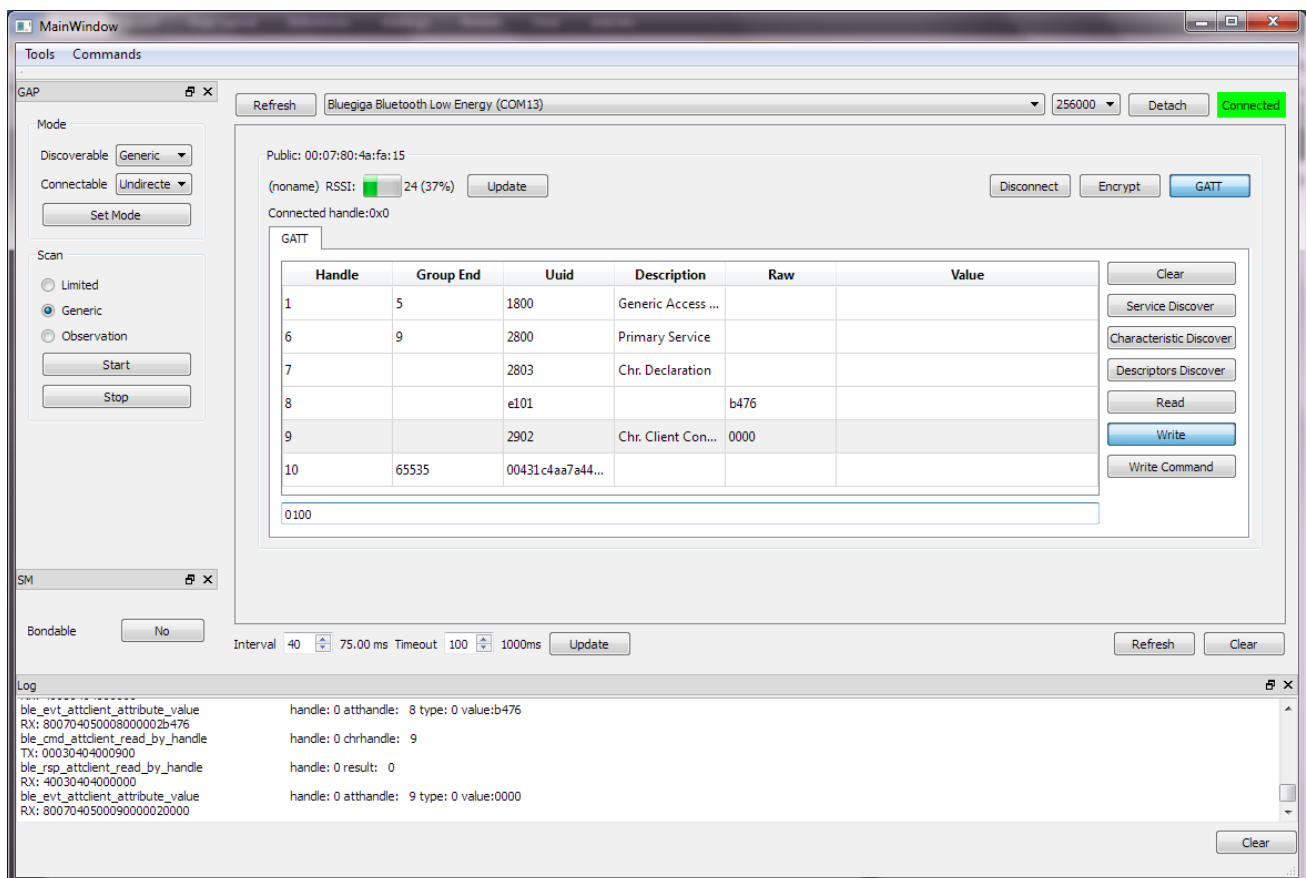


Figure 24: Enabling notifications

- To disable notifications, simply set the value of UUID 2902 to 0000.

6.2 Reading and writing the manufacturer proprietary service

The procedures to read or write the manufacturer proprietary service are similar to the battery status service.

- Connect the BGDemo sensor
- Make GATT service discovery
- Select the proprietary service and make descriptor discovery
- Reading: Select the proprietary characteristic and press **Read** button
- Writing: Select the proprietary characteristic, write the new value to the field right below the GATT tool and press **Write** button.

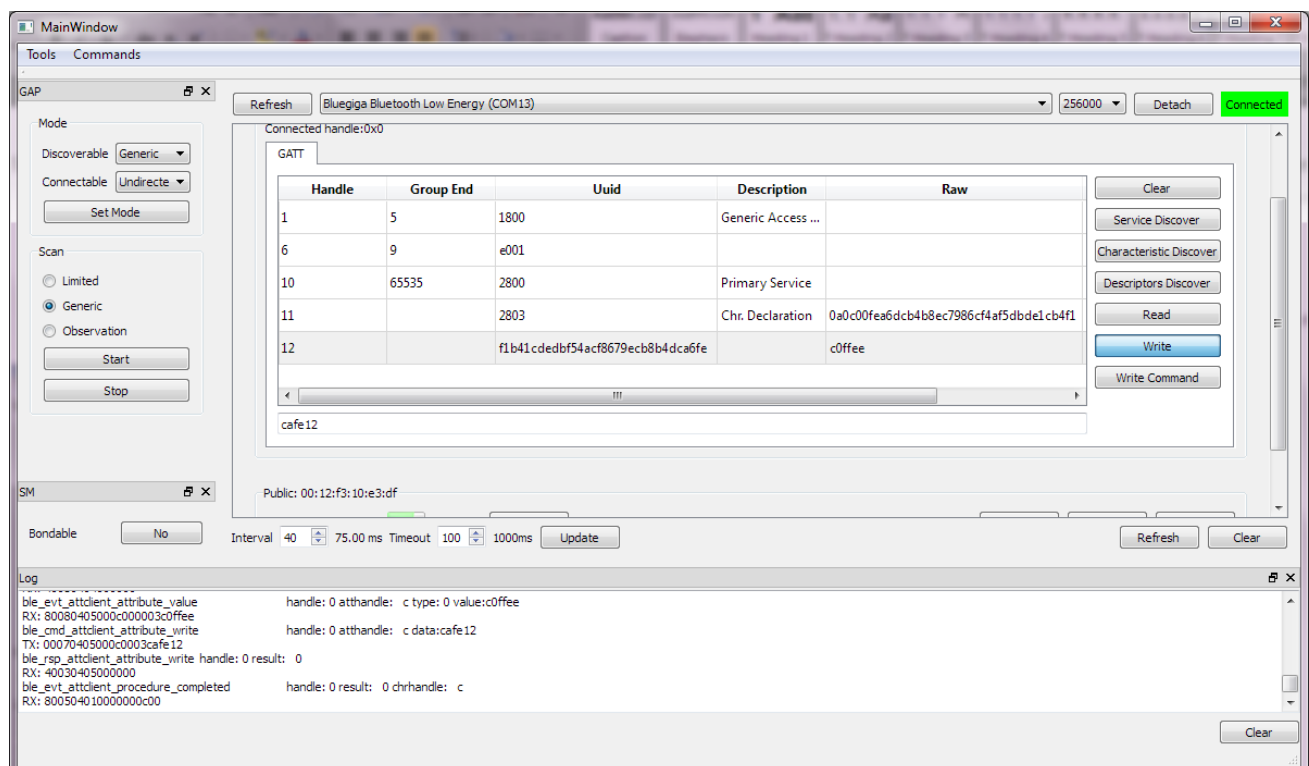


Figure 25: Writing the characteristic value

6.3 Writing application using BGLib C-library

TBA

7 Contact information

Sales: sales@bluegiga.com

Technical support: support@bluegiga.com
<http://techforum.bluegiga.com>

Orders: orders@bluegiga.com

WWW: www.bleugiga.com
www.bluegiga.hk

Head Office / Finland:

Phone: +358-9-4355 060
Fax: +358-9-4355 0660
Sinikalliontie 5A
02630 ESPOO
FINLAND

Postal address / Finland:

P.O. BOX 120
02631 ESPOO
FINLAND

Sales Office / USA:

Phone: +1 770 291 2181
Fax: +1 770 291 2183
Bluegiga Technologies, Inc.
3235 Satellite Boulevard, Building 400, Suite 300
Duluth, GA, 30096, USA

Sales Office / Hong-Kong:

Phone: +852 3182 7321
Fax: +852 3972 5777
Bluegiga Technologies, Inc.
19/F Silver Fortune Plaza, 1 Wellington Street,
Central Hong Kong