BLED112

DATA SHEET

Friday, 04 April 2014

Version 1.1

bluegiga

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.

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Current consumption added

TABLE OF CONTENTS

BLED112 *Bluetooth*® Smart USB Dongle

DESCRIPTION

BLED112 *Bluetooth* Smart Dongle integrates all *Bluetooth* Smart features. The USB dongle can a virtual COM port that enables simple host application development using a simple application programming interface. The BLED112 can be used for *Bluetooth* Smart development. With two BLED112 dongles you can quickly prototype new *Bluetooth* Smart application profiles by utilizing Bluegiga Profile Toolkit^{TM} and also automate in module software functions with Bluegiga BGScript^{TM} .

Figure 1: BLED112 Bluetooth Smart USB dongle

KEY FEATURES:

- *Bluetooth* v.4.0, single mode compliant
  - Supports master and slave modes
  - Supports up to eight connections
- Integrated *Bluetooth* Smart stack
  - GAP, GATT, L2CAP and SMP
  - *Bluetooth* Smart profiles
- Radio performance
  - Transmit power : +0 dBm to -27 dBm
  - Receiver sensitivity: -91 dBm
- Host interfaces
  - USB (virtual COM port emualation)
- Programmable 8051 processor for stand-alone operation
- Simple Bluegiga BGScript^{TM} scripting language for quick application development
- Bluegiga Profile Toolkit^{TM} allowing the quick development of GATT based profiles
- Free Software Development Kit
- *Bluetooth*, CE, FCC, IC and South-Korea and Japan qualified

1 BLED112 Product Numbering

Available products and product codes

Product code	Description
BLED112	BLED112 USB dongle

2 Electrical Characteristics

2.1 Absolute Maximum Ratings

Note: These are absolute maximum ratings beyond which the module can be permanently damaged. These are not maximum operating conditions. The maximum recommended operating conditions are in the table 6.

Rating	Min	Max	Unit
Storage Temperature	-40	+85	°C
VBUS	-0.3	6.5	V

Table 1: Absolute Maximum Ratings

2.2 Recommended Operating Conditions

Rating	Min	Max	Unit
Operationg Temperature Range	-40	+85	°C
VBUS	3.6	5.5	V

Table 2: Recommended Operating Conditions

2.3 Current Consumption

Rating		AVG	Peak	Unit
Idle		12.1		mA
Scan			44	mA
Advertising	TX		44	mA
	RX		33	mA

Table 3: Current Consumption

3 Block Diagram

BLED112 is based on TI's CC2540 chip. Embedded 32 MHz and 32.678 kHz crystals are used for clock generation..

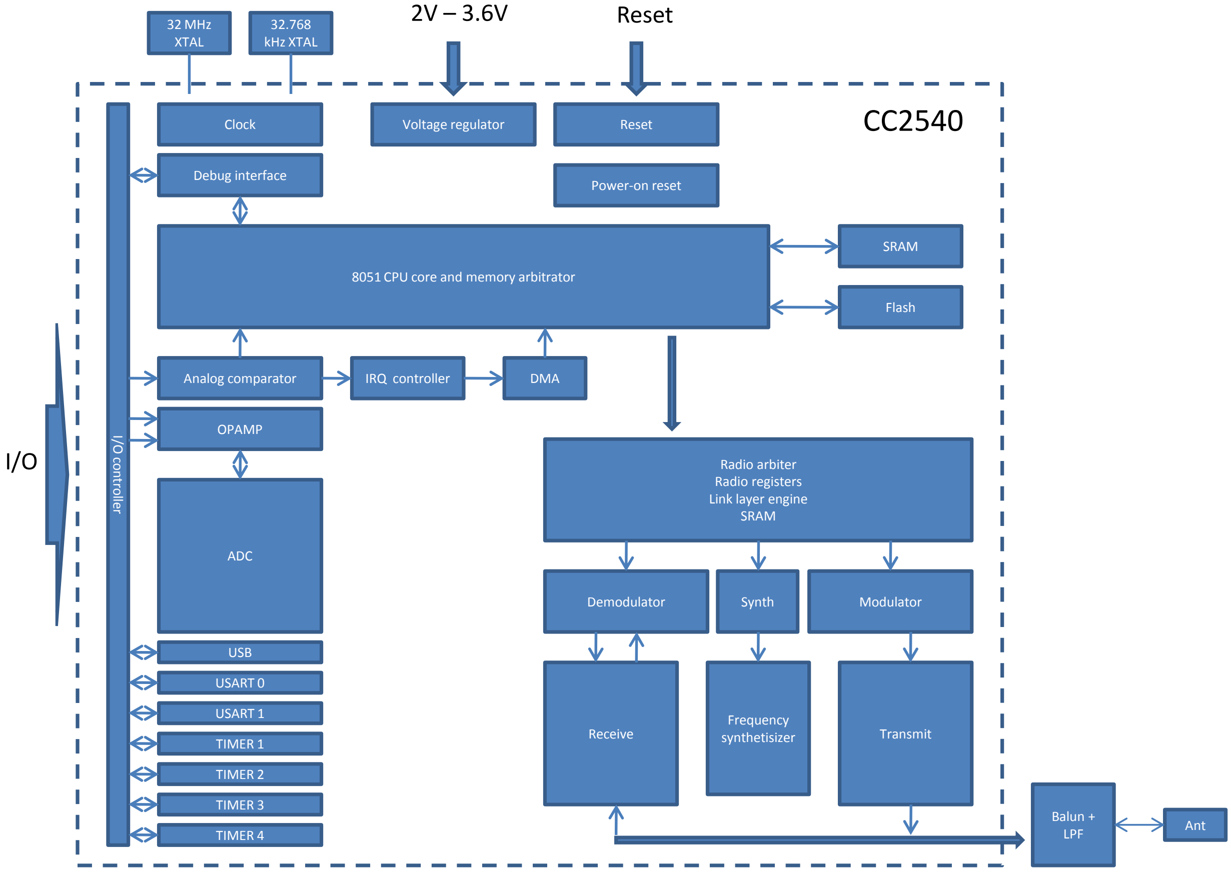


Figure 2: Simplified block diagram of BLE112

CPU and Memory

The 8051 CPU core is a single-cycle 8051-compatible core. It has three different memory access buses (SFR, DATA, and CODE/XDATA), a debug interface, and an 18-input extended interrupt unit.

The memory arbiter is at the heart of the system, as it connects the CPU and DMA controller with the physical memories and all peripherals through the SFR bus. The memory arbiter has four memory-access points, access of which can map to one of three physical memories: an SRAM, flash memory, and XREG/SFR registers. It is responsible for performing arbitration and sequencing between simultaneous memory accesses to the same physical memory.

The SFR bus is a common bus that connects all hardware peripherals to the memory arbiter. The SFR bus also provides access to the radio registers in the radio register bank, even though these are indeed mapped into XDATA memory space.

The 8-KB SRAM maps to the DATA memory space and to parts of the XDATA memory spaces. The SRAM is an ultralow-power SRAM that retains its contents even when the digital part is powered off (power modes 2 and 3).

The 128KB flash block provides in-circuit programmable non-volatile program memory for the device, and maps into the CODE and XDATA memory spaces.

4 Certifications

4.1 Bluetooth

BLED112 Bluetooth low energy module is *Bluetooth* qualified and listed as an End Product.

4.2 FCC and IC

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation.

This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one of the following measures:

- Reorient or relocate the receiving antenna
- Increase the separation between the equipment and receiver
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- Consult the dealer or an experienced radio/TV technician for help

FCC Caution : To assure continued compliance, any changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate this equipment. (Example - use only shielded interface cables when connecting to computer or peripheral devices).

FCC Radiation Exposure Statement

This equipment complies with FCC RF radiation exposure limits set forth for an uncontrolled environment. This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference, and

(2) This device must accept any interference received, including interference that may cause undesired operation.

4.3 Industry Canada

IC Statements:

This device complies with Industry Canada licence-exempt RSS standard(s). Operation is subject to the following two conditions: (1) this device may not cause interference, and (2) this device must accept any interference, including interference that may cause undesired operation of the device.

Under Industry Canada regulations, this radio transmitter may only operate using an antenna of a type and maximum (or lesser) gain approved for the transmitter by Industry Canada. To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (e.i.r.p.) is not more than that necessary for successful communication.

Déclaration d’IC :

Ce dispositif est conforme aux normes RSS exemptes de licence d’Industrie Canada. Son fonctionnement est assujetti aux deux conditions suivantes : (1) ce dispositif ne doit pas provoquer de perturbation et (2) ce dispositif doit accepter toute perturbation, y compris les perturbations qui peuvent entraîner un fonctionnement non désiré du dispositif.

Selon les réglementations d’Industrie Canada, cet émetteur radio ne doit fonctionner qu’avec une antenne d’une typologie spécifique et d’un gain maximum (ou inférieur) approuvé pour l’émetteur par Industrie Canada. Pour réduire les éventuelles perturbations radioélectriques nuisibles à d’autres utilisateurs, le type d’antenne et son gain doivent être choisis de manière à ce que la puissance isotrope rayonnée équivalente (P.I.R.E.) n’excède pas les valeurs nécessaires pour obtenir une communication convenable.

4.4 CE

BLED112 is in conformity with the essential requirements and other relevant requirements of the R&TTE Directive (1999/5/EC). The product is conformity with the following standards and/or normative documents.

- EMC EN 301 489-17 V.1.3.3 in accordance with EN 301 489-1 V1.8.1
- Radiated emissions EN 300 328 V1.7.1
- Safety EN 60950-1

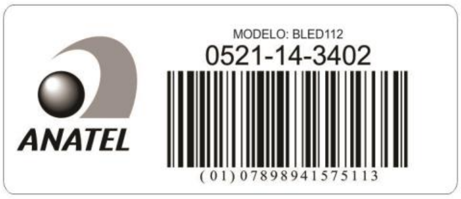
4.5 South-Korea

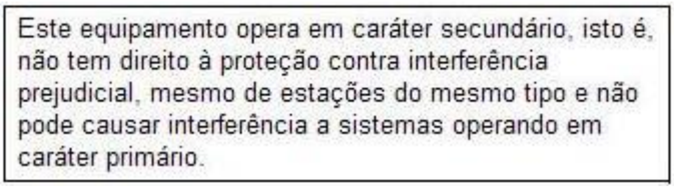
BLED112 is certified in South-Korea with certification number: KCC-CRM-BGT-BLED112

4.6 Japan

BLED112 has MIC Japan type certification with certification number: 003WWA111471

4.7 Brazil




Este equipamento opera em caráter secundário, isto é, não tem direito à proteção contra interferência prejudicial, mesmo de estações do mesmo tipo e não pode causar interferência a sistemas operando em caráter primário.

5 Contact Information

Sales: sales@bluegiga.com

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

Orders: orders@bluegiga.com

WWW: www.bluegiga.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 3972 2186
Bluegiga Technologies Ltd.
Unit 10-18
32/F, Tower 1, Millennium City 1
388 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong

BLUETOOTH SMART SDK

Developing your 1st *Bluetooth* Smart Application

Thursday, 26 September 2013

Version 2.0

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 products are not authorized for use as critical components in life support devices or systems.

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	*Bluetooth* LE description updated
1.6	Minor updates
1.7	Updated compile and installation instructions
1.8	Chapter 3 updated
1.9	Chapter 4 updated
2.0	Minor changes

TABLE OF CONTENTS

1 Introduction

This application note discusses how to start developing Bluetooth Smart applications using Bluegiga *Bluetooth* Smart modules and BLED112 Bluetooth Smart USB dongle. The application note contains a practical example of how to build Bluetooth Smart GATT based services with the profile toolkit, how to make a standalone sensor device using BGScript programming language.

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.

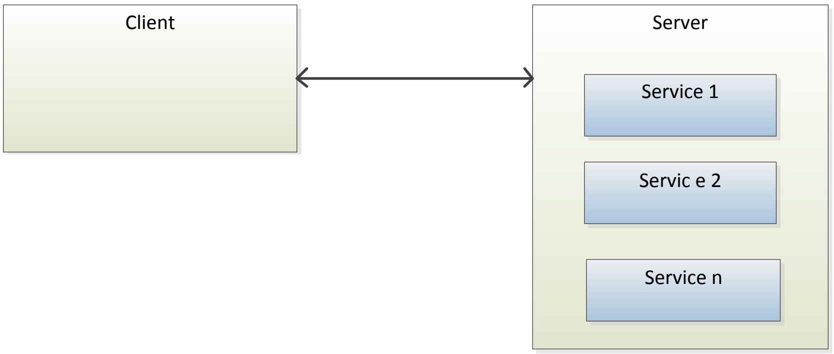


Figure 1: *Bluetooth* low energy device roles

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* Smart 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.).

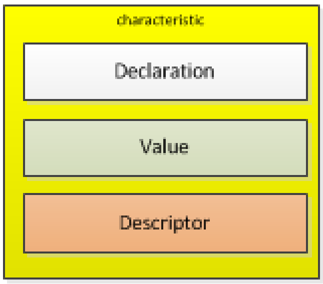


Figure 2: Characteristic structure

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.

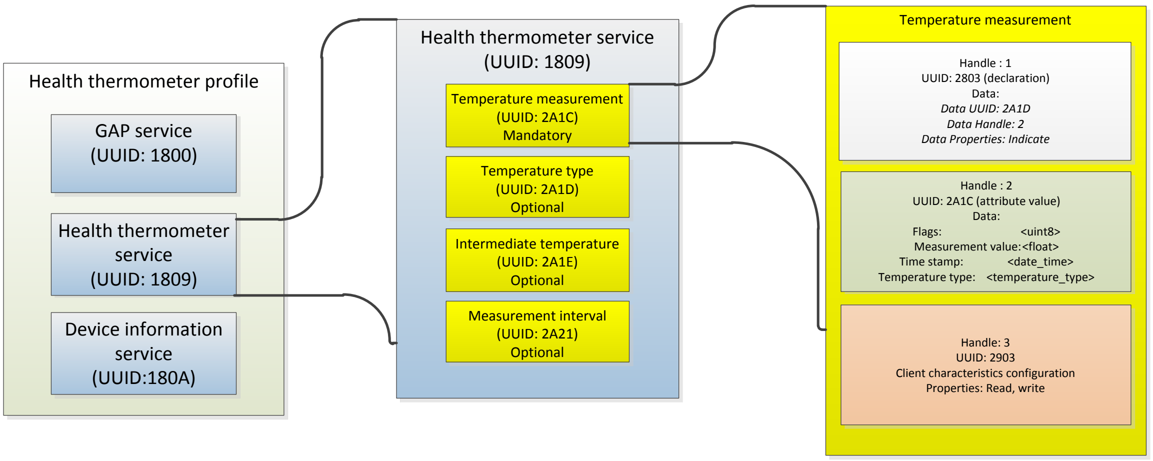

Health thermometer profile
GAP service (UUID: 1800)
Health thermometer service (UUID: 1809)
Device information service (UUID:180A)
Health thermometer service (UUID: 1809)
Temperature measurement (UUID: 2A1C) Mandatory
Temperature type (UUID: 2A1D) Optional
Intermediate temperature (UUID: 2A1E) Optional
Measurement interval (UUID: 2A21) Optional
Temperature measurement
Handle : 1
UUID: 2803 (declaration)
Data:
Data UUID: 2A1D
Data Handle: 2
Data Properties: Indicate
Handle : 2
UUID: 2A1C (attribute value)
Data:
Flags: <uint8>
Measurement value:<float>
Time stamp: <date_time>
Temperature type: <temperature_type>
Handle: 3
UUID: 2903
Client characteristics configuration
Properties: Read, write

Figure 3: Health thermometer profile

4 Introduction to the Bluegiga *Bluetooth* Smart Software

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

- The *Bluetooth* Smart Stack
- The *Bluetooth* Smart Software Development Kit (SDK)

4.1 The *Bluetooth* Smart Stack

The *Bluetooth* Smart 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* Smart is meant for the Bluegiga *Bluetooth* Smart 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* Smart SDK

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

The *Bluetooth* Smart 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* Smart 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 can be developed using a simple scripting language called BGScript^{TM}. Several profiles and examples are also offered as a part of the *Bluetooth* Smart Software in order to easily develop the *Bluetooth* Smart compatible end products.

Bluegiga's *Bluetooth Smart Software* provides a complete development framework for *Bluetooth* low energy application implementers.

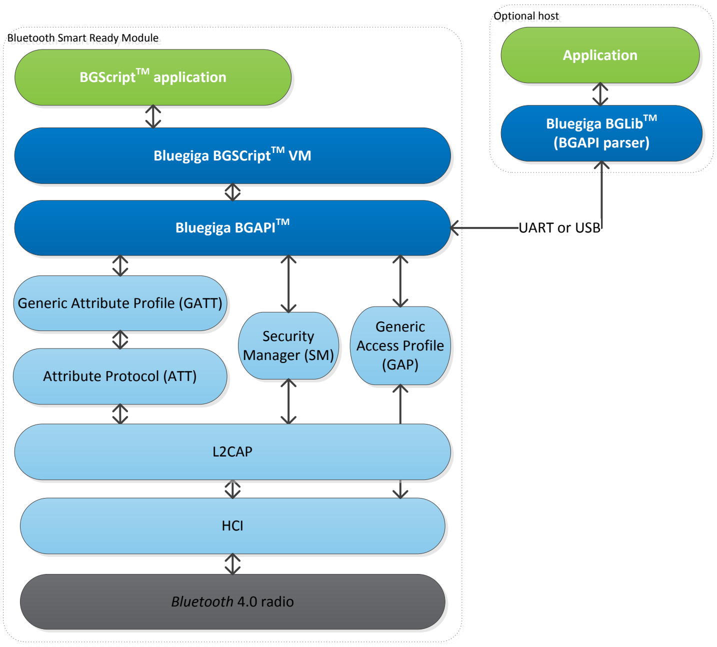


Figure 4: *Bluetooth Smart* Software

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

- The *Bluetooth* Smart stack implementing the *Bluetooth* low energy protocol
- **BGAPI™** APIs that enable the software developers to interface to the *Bluetooth* Smart Stack
- **BGScript™** Virtual Machine (VM) and scripting language which enable application code to be developed and executed directly on the *Bluetooth* Smart 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* Smart 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* Smart 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* Smart hardware) or USB interfaces.

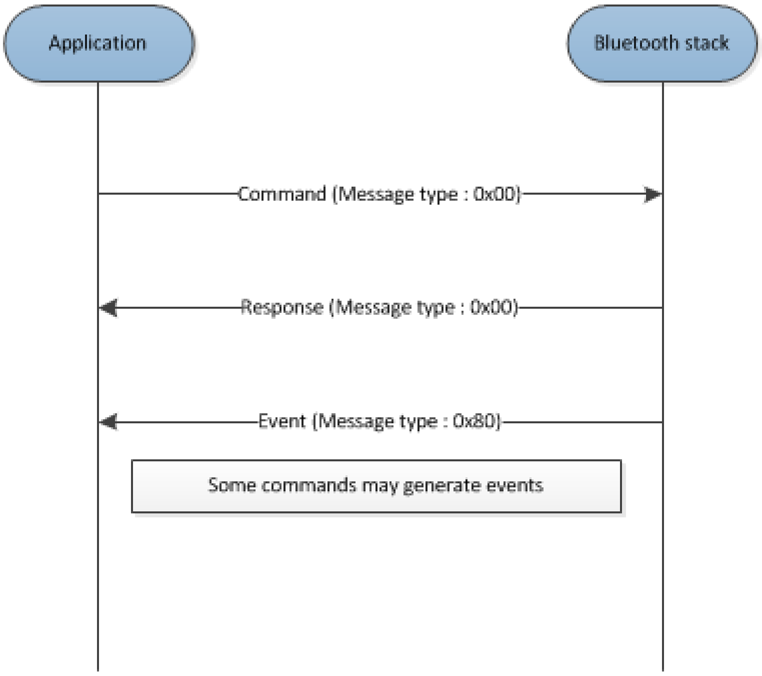

Application
Bluetooth stack
Command (Message type : 0x00)
Response (Message type : 0x00)
Event (Message type : 0x80)
Some commands may generate events

Figure 5: BGAPI protocol

The BGAPI provides access for example to the following layers in the *Bluetooth* Smart 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* Smart SDK. The purpose is to simplify the application development to various host environments.

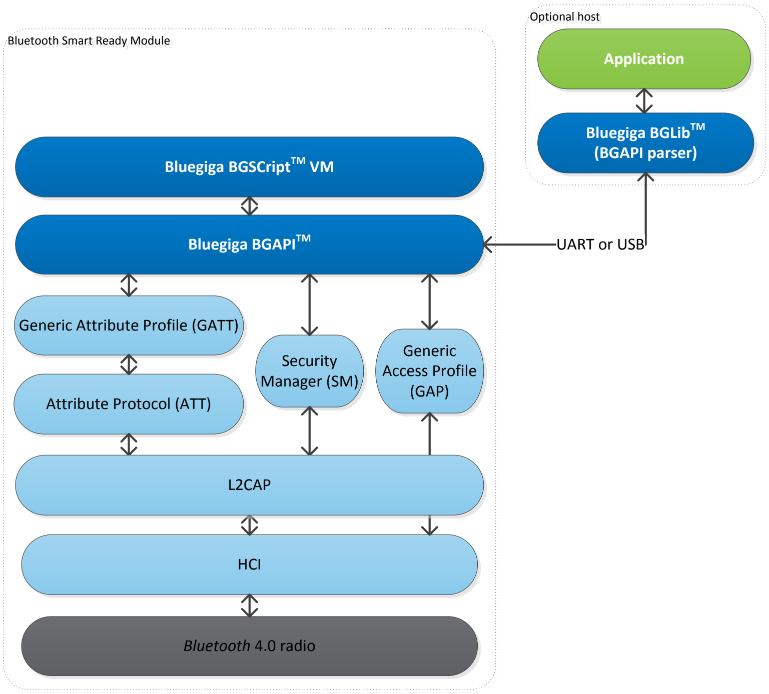


Figure 6: BGLib host library

4.5 BGScript^{TM} Scripting Language

The *Bluetooth* Smart 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* Smart Hardware. The *Bluetooth* Smart modules can run simple applications along the *Bluetooth* Smart stack and this provides a benefit when one needs to minimize the end product’s size, cost and current consumption. For developing standalone *Bluetooth* Smart 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.

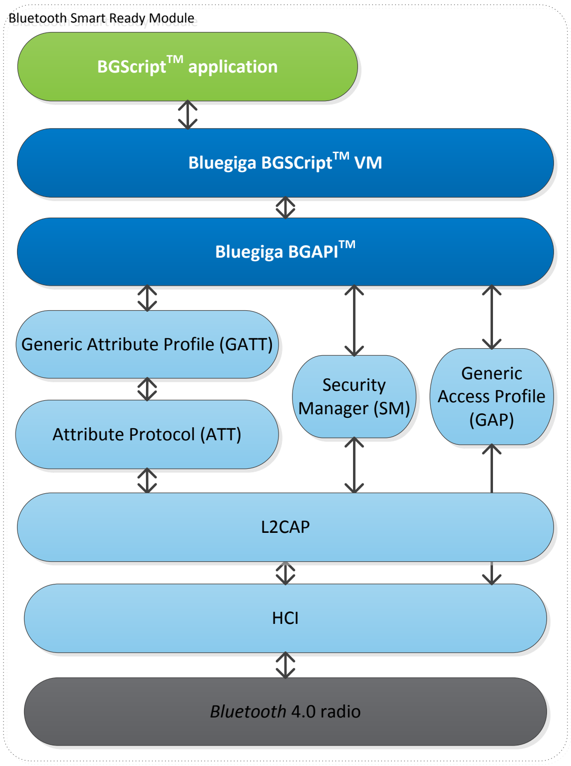


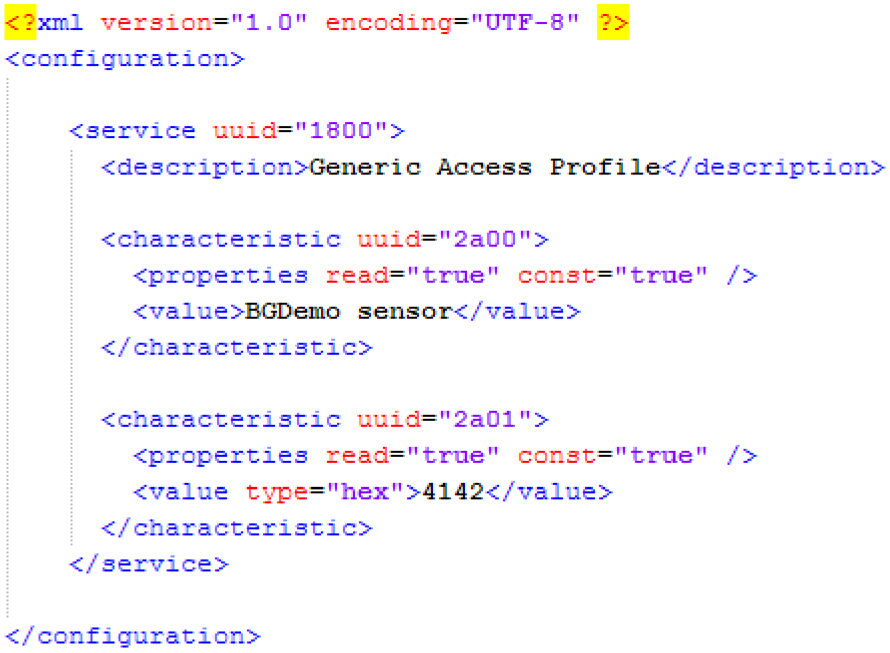
Figure 7: BGScript application model

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* Smart profile toolkit a simple set of tools, which can used to describe GATT based *Bluetooth* Smart 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

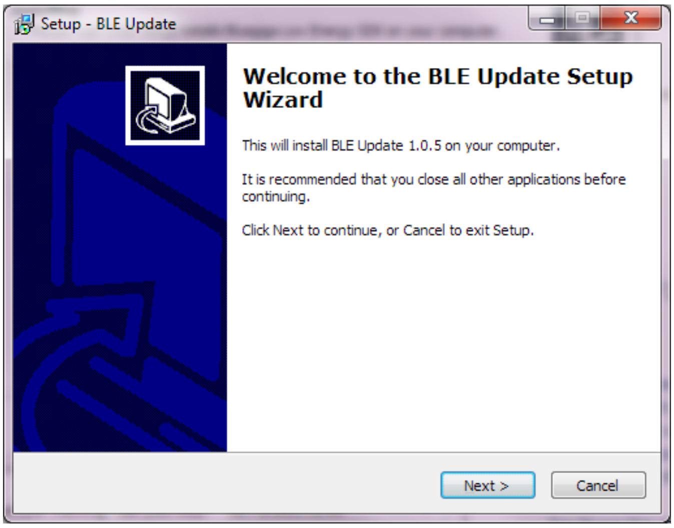
5 Implementation of “BGDemo” Sensor

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

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

5.1 Installing the Tools

1. Download the latest install the Bluegiga *Bluetooth* Smart SDK from the Bluegiga web site
2. Run the executable
3. Follow the on-screen instructions and install the SDK to the desired directory
4. Perform a Full Installation (BLE SDK and TI tools)



Welcome to the BLE Update Setup Wizard

This will install BLE Update 1.0.5 on your computer.

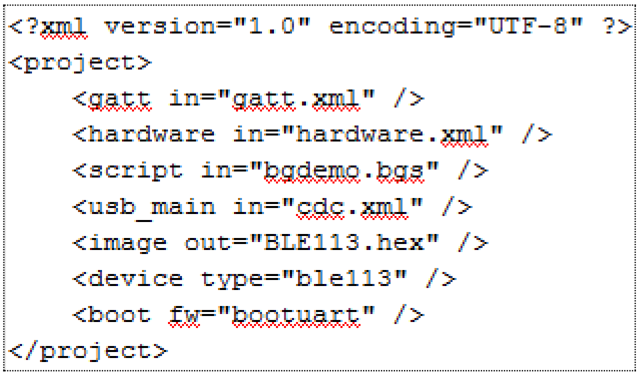
It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 9: Installing Bluegiga *Bluetooth* Smart SDK

5.2 Creating a Project

The project is started by creating a project file. The file is a simple XML formatted 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="BLE113.hex" />
    <device type="ble113" />
    <boot fw="bootuart" />
</project>
```

Figure 10: Project file

- The project configuration is described within the <project> tags
- <gatt> tag defines the .XML file containing the GATT data base
- <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
- <device type> tag defines if the project is meant for BLE112 or BLE113 hardware
- <boot fw> tag defines which interface is enabled for DFU firmware upgrades

The exact syntax and options of the project file can be found from the *BLE112 and BLE113 Configuration Guide* and the syntax is not fully described in this document.

NOTE:

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

WARNING:

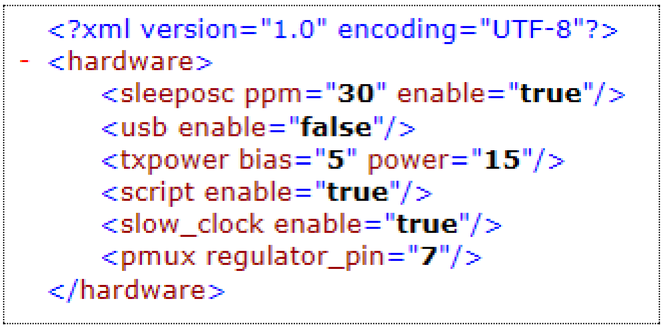
If the firmware is to be installed into the BLED112 USB dongle the USB CDC configuration MUST BE included in the project file. If this is not included in the project file and the compiled firmware is installed into the BLED112 USB dongle, the USB interface will be disabled and the dongle stops from working.

5.3 Hardware Configuration

Once the project is configured the next logical step is the hardware configuration of your Bluetooth Smart module. In this document we use the BLE113 *Bluetooth* Smart Module as a target platform.

If the default project template is used, the file where the hardware configuration remains 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 ppm="30" enable="true"/>
      <usb enable="false"/>
      <txpower bias="5" power="15"/>
      <script enable="true"/>
      <slow_clock enable="true"/>
      <pmux regulator_pin="7"/>
  </hardware>
```

Figure 11: Hardware configuration for the BLE113 *Bluetooth* Smart Module

- 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 BLE113 does incorporate the sleep oscillator so this value should be set to true especially in the applications where power consumption matters. The PPM value defines the sleep oscillator accuracy and MUST not be changed.
- <usb> tag defines if USB is to be enabled or not. The BLE113 (unlike BLE112 or BLED112) does not have USB interface so we leave the setting to **false**.
- <txpower> tag defines the TX power level and the value 15 configures the maximum TX power level.
- <script enable> tag defines if BGScript VM and application are present. Since our example uses a BGscript application we set this value to **true.**
- <slow clock> tag enabled the slow the MCU clock when there is radio activity and reduces the peak power consumption. The option is enabled by setting the value to **true.**
- <pmux regulator_pin> configuration defines which GPIO pin is used to control an external DC/DC converter. An external DC/DC converter can be used to lower the peak power consumption during radio activity and the Bluetooth Smart software will automatically enable or disable the DC/DC based on the software status. The DKBLE112 and DKBLE113 development kits have the DC/DC converter, so this feature is enabled.

NOTE:

- Enabling the <slow clock> feature will corrupt high speed UART transmissions, so if UART is used in your application this feature MUST NOT be enabled.

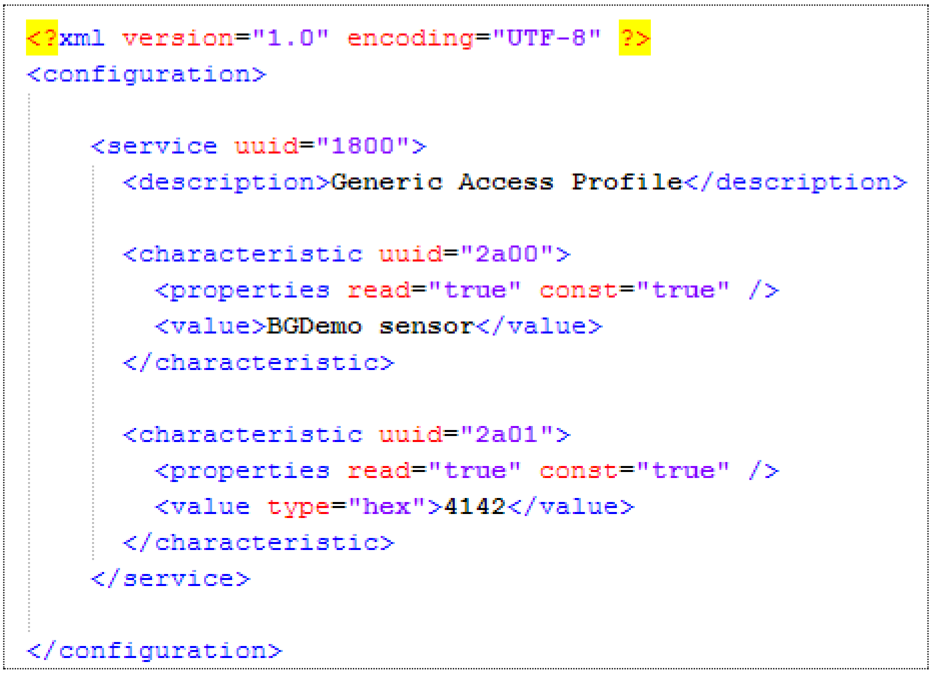
:

5.4 Building a GATT Database with Profile Toolkit

This section discusses the implementation of a GATT database so the services and characteristics exposed by a device. The service database is created with the Profile Toolkit^{TM} tools, which simply is are just XML based description language and templates.

5.4.1 Generic Access Profile Service

Every *Bluetooth* Smart 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

5.4.1.1 Service Description

A Bluetooth Smart service is described within the <service> tags. For every service you need to define a UUID as shown in the example above.

For the GAP service the globally unique 16-bit UUID is : **1800**

In the example above we also use optional <description> tag is used to identify the service name. This is optional tag and can be considered to be a comment in the XML file and is not used in the actual device.

The Bluetooth SIG standardized services and UUIDs are available at:

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

5.4.1.2 Characteristics Description

Characteristics belonging to a service are described within <characteristic”> tags and they must be inside the <service> tags of the service they belong to.

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 (similar to the friendly name used in *Bluetooth* classic)

- **Device appearance (UUID: 2A01)**

  This identifies the devices type (similar to the Class-of-Device used in *Bluetooth* classic)

Characteristics also must have unique UUIDs and they need to be defined in the GATT database as shown above.

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

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

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 defined read only. Now since the values are read only they can be marked as **const** meaning the values are constant and they will be stored on the flash memory during the firmware installation.

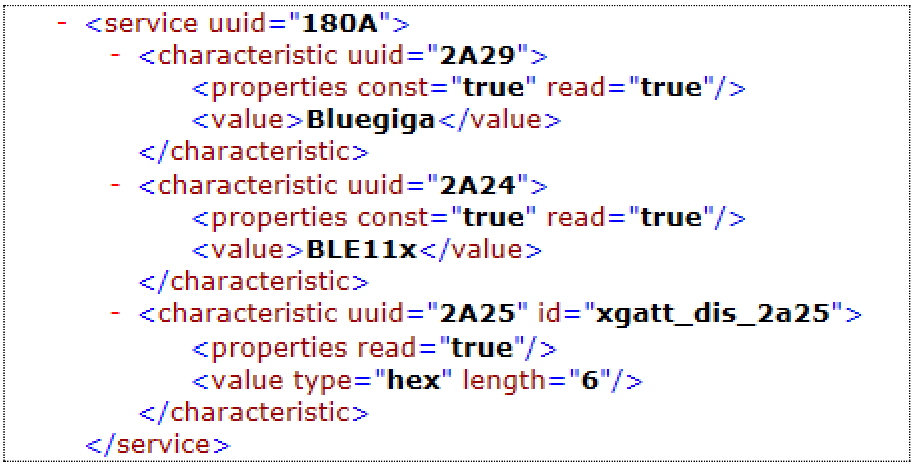
Characteristics values:

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

The exact syntax and more examples of services and characteristics definitions can be found from the *Profile Toolkit Developer Guide*.

5.4.2 Device ID

The second service implemented in our example is the Device ID service. The DI service exposes information about the vendor of the device and optional information for example about the devices firmware and hardware versions. In this example we implement a fairly minimalistic DI service with only a few characteristics. The DI service description is very similar to the GAP service and has only a few differences. The XML description is shown below.



```
- <service uuid="180A">
  - <characteristic uuid="2A29">
      <properties const="true" read="true"/>
      <value>Bluegiga</value>
    </characteristic>
  - <characteristic uuid="2A24">
      <properties const="true" read="true"/>
      <value>BLE11x</value>
    </characteristic>
  - <characteristic uuid="2A25" id="xgatt_dis_2a25">
      <properties read="true"/>
      <value type="hex" length="6"/>
    </characteristic>
  </service>
```

Figure 13: Device ID service

The global UUID for the DI service is: 180A

Link: Device ID Service

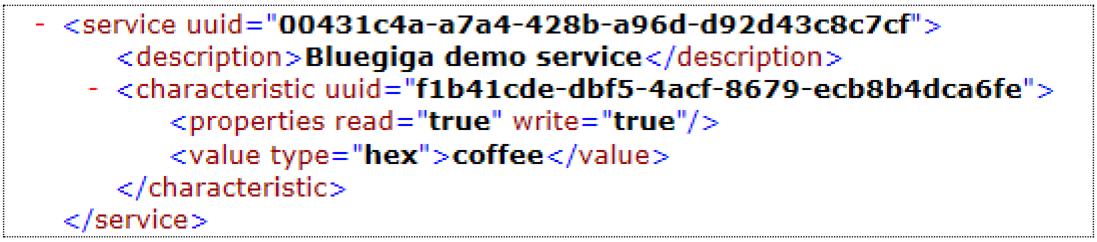
Three characteristic are defined and they are Manufacturer Name String, Model Number String and Serial Number String. All of these characteristics have UTF-8 format and they are **ready only** values. The two first values we permanently store to the flash and mark them **const**, but the third value is unique to every device and later we want to be able to modify the value with our BGscript code. Therefore we do no mark it const and we also define and ID for the value **xgatt_dis_2a35** which we later use in the BGScript code **to** write the devices Bluetooth address to the serial number string.

Link: Manufacturer Name String, Model Number String and Serial Number String

5.4.3 A Manufacturer Specific Service

The third service used in this example is a manufacturer specific service. *Bluetooth* Smart devices can have services defined by manufacturers which are not standardized by the *Bluetooth* SIG. The service structure is exactly the same however, manufacturer specific services MUST use 128-bit long UUIDs instead of the 16-bit UUIDs reserved for the standardized services.

The 128-bit UUIDs do not need to be applied or registered, but can be generated using for example online tools such as this site: http://www.uuidgenerator.net/



```
- <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>
```

Figure 14: Proprietary service

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

5.5 Writing BGScript Code

This example implements a standalone sensor device without an external host processor. The sensor side application is created with 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.

```
dim tmp(10)
dim addr(6)

# Boot Event listener
event system_boot(major ,minor ,patch ,build ,ll_version ,protocol_version ,hw )

           #Read local BT address
           call system_address_get( )(addr(0:6))

           # Write BT address to DI service serial number string
           call attributes_write(xgatt_dis_2a25,0,6,addr(0:5))

           #set bondable mode
           call sm_set_bondable_mode(1)

           #set to advertising mode
           call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
end

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

The BGScript has two event listeners defined:

- `system_boot(…)` **event listener**

  When the system is started (power up) a boot event is generated and this event listener will catch it. This event is the entry point for all the BGScript applications. In the example above, when the system is started, the BGSCript code reads the local devices MAC (Bluetooth) address, stores it to the Device ID services serial number string, enables bonding mode in case the remote device wants to do pairing and finally starts the advertisement procedure so the device becomes visible and connectable to other devices.

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

  The second event handler is executed when a *Bluetooth* Smart connection is lost or closed by the remote device and it simply enables the advertisement mode again.

The BGScript functions and events can be found from the *Bluetooth Smart Software API reference* document.

5.6 Compiling and Installing the Firmware

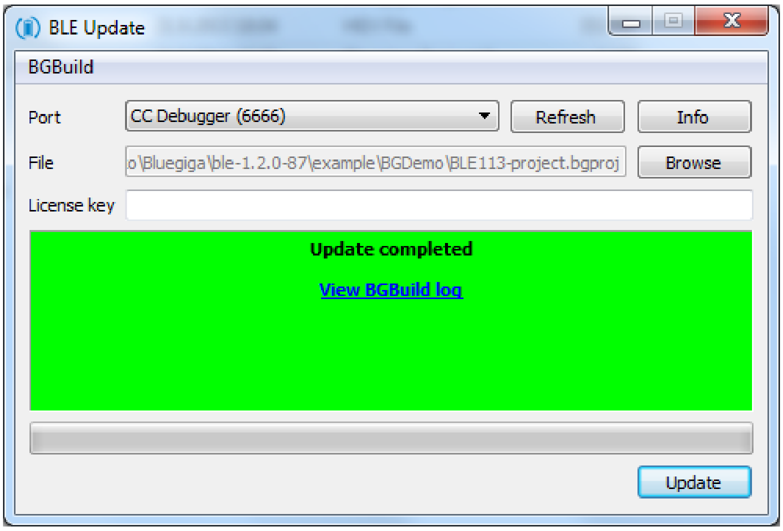
5.6.1 Using BLE Update tool

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 Smart 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.



BLE Update

BGBuild

Port: CC Debugger (6666)

File: o\Bluegiga\ble-1.2.0-87\example\BGDemo\BLE113-project.bgproj

License key:

Update completed

View BGBuild log

Figure 15: Compile and install with BLE Update tool

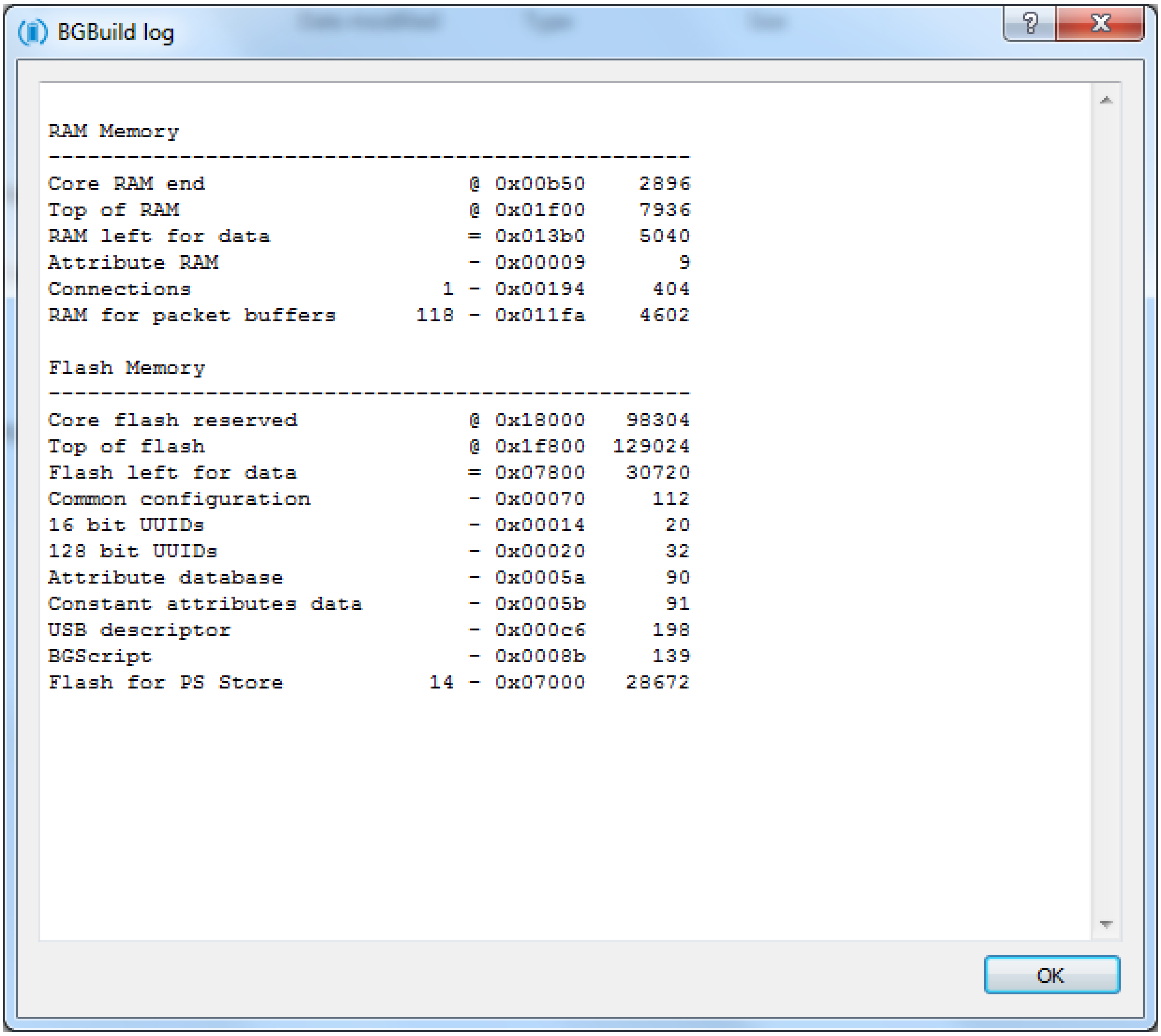
Note:

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.



```
RAM Memory
-------------------------------------------------
Core RAM end                      @ 0x00b50    2896
Top of RAM                        @ 0x01f00    7936
RAM left for data                 = 0x013b0    5040
Attribute RAM                     - 0x00009       9
Connections                     1 - 0x00194     404
RAM for packet buffers        118 - 0x011fa    4602

Flash Memory
-------------------------------------------------
Core flash reserved               @ 0x18000   98304
Top of flash                      @ 0x1f800  129024
Flash left for data               = 0x07800   30720
Common configuration              - 0x00070     112
16 bit UUIDs                      - 0x00014      20
128 bit UUIDs                     - 0x00020      32
Attribute database                - 0x0005a      90
Constant attributes data          - 0x0005b      91
USB descriptor                    - 0x000c6     198
BGScript                          - 0x0008b     139
Flash for PS Store             14 - 0x07000   28672
```

Figure 16: BLE Update build log

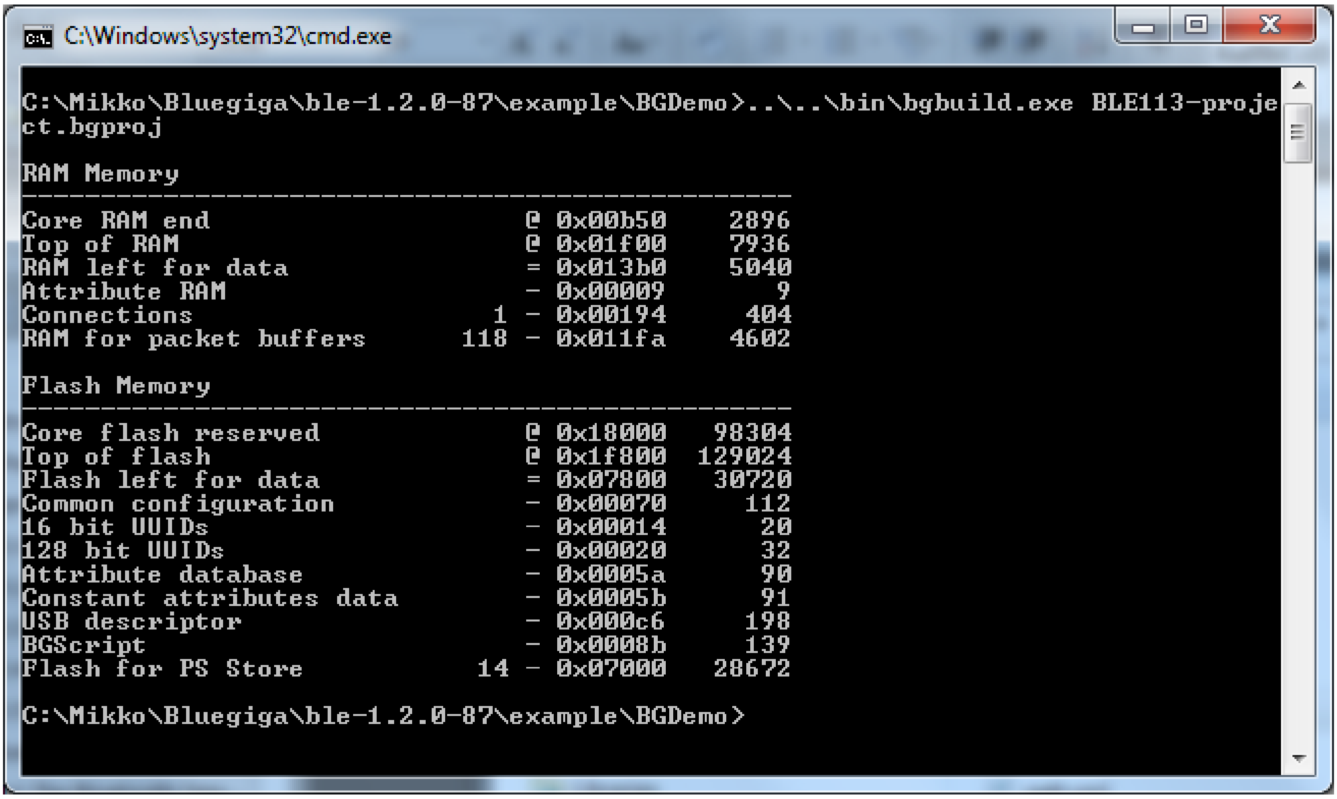
5.6.2 Compiling Using bgbuild.exe

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>***



```
C:\Mikko\Bluegiga\ble-1.2.0-87\example\BGDemo>..\..\bin\bgbuild.exe BLE113-proje
ct.bgproj

RAM Memory
-------------------------------------------------
Core RAM end                      @ 0x00b50    2896
Top of RAM                        @ 0x01f00    7936
RAM left for data                 = 0x013b0    5040
Attribute RAM                     - 0x00009       9
Connections                     1 - 0x00194     404
RAM for packet buffers        118 - 0x011fa    4602

Flash Memory
-------------------------------------------------
Core flash reserved               @ 0x18000   98304
Top of flash                      @ 0x1f800  129024
Flash left for data               = 0x07800   30720
Common configuration              - 0x00070     112
16 bit UUIDs                      - 0x00014      20
128 bit UUIDs                     - 0x00020      32
Attribute database                - 0x0005a      90
Constant attributes data          - 0x0005b      91
USB descriptor                    - 0x000c6     198
BGScript                          - 0x0008b     139
Flash for PS Store             14 - 0x07000   28672

C:\Mikko\Bluegiga\ble-1.2.0-87\example\BGDemo>
```

Figure 17: Compiling with BGBuild.exe

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

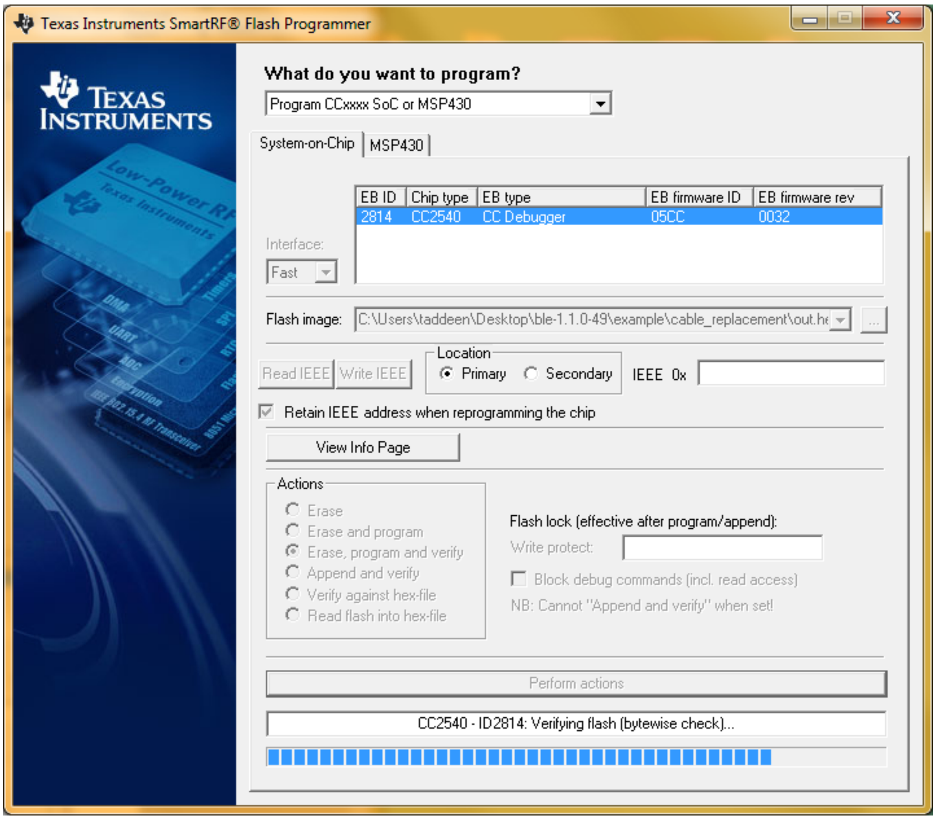
On the other hand if the compilation fails due to syntax errors in the BGScript or GATT files, and error message is printed.

5.6.3 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



Texas Instruments SmartRF® Flash Programmer

What do you want to program?

Program CCxxxx SoC or MSP430

System-on-Chip | MSP430

EB ID	Chip type	EB type	EB firmware ID	EB firmware rev
2814	CC2540	CC Debugger	05CC	0032

Interface: Fast

Flash image: C:\Users\taddeen\Desktop\ble-1.1.0-49\example\cable_replacement\out.he

Read IEEE | Write IEEE

Location: ● Primary ○ Secondary

IEEE 0x

☑ Retain IEEE address when reprogramming the chip

View Info Page

Actions
○ Erase
○ Erase and program
● Erase, program and verify
○ Append and verify
○ Verify against hex-file
○ Read flash into hex-file

Flash lock (effective after program/append):

Write protect:

□ Block debug commands (incl. read access)

NB: Cannot "Append and verify" when set!

Perform actions

CC2540 - ID2814: Verifying flash (bytewise check)...

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 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 Testing the BGDemo Sensor

6.1 Using BLEGUI

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

BLEGUI is a simple PC utility that can be used to control a Bluegiga *Bluetooth* Smart device over UART or USB. BLEGUI software sends the BGAPI commands to the device and parses the reponses and has a simple user interface to display device data.

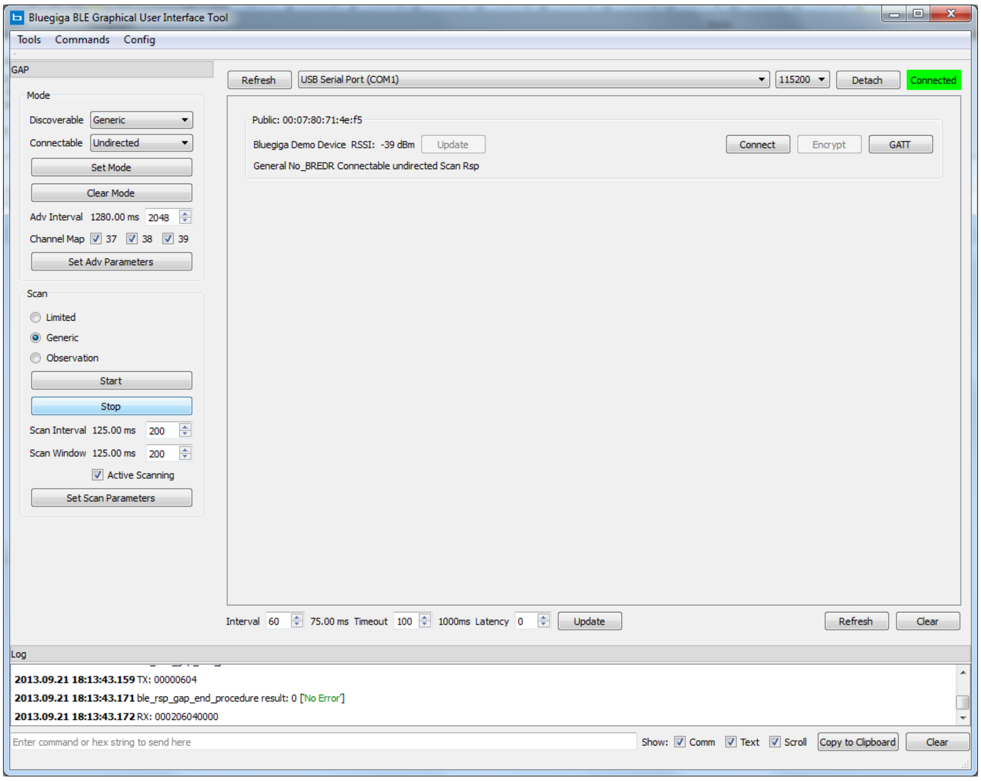
6.1.1 Discovering the BGDemo Sensor

- Connect for example a BLED112 USB dongle to your PC
- Make use the USB/CDC driver gets installed and a Virtual COM port gets created
- Open BLEGUI software and attach the device in the virtual COM port to the BLEGUI

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.

- Enable **Active Scanning**
- Press **Set Scan Parameters**
- Select **Generic** scan mode
- Press **Scan**

If the BGDemo device is power on and the BGDemo application is installed to is you should see the device in the BLEGUI software.



Bluegiga BLE Graphical User Interface Tool

USB Serial Port (COM1) 115200 Connected

Mode

Discoverable: Generic

Connectable: Undirected

Adv Interval 1280.00 ms 2048

Channel Map ☑ 37 ☑ 38 ☑ 39

Scan

○ Limited

● Generic

○ Observation

Scan Interval 125.00 ms 200

Scan Window 125.00 ms 200

☑ Active Scanning

Public: 00:07:80:71:4e:f5

Bluegiga Demo Device RSSI: -39 dBm

General No_BREDR Connectable undirected Scan Rsp

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:13:43.159 TX: 00000604
2013.09.21 18:13:43.171 ble_rsp_gap_end_procedure result: 0 ['No Error']
2013.09.21 18:13:43.172 RX: 000206040000
```

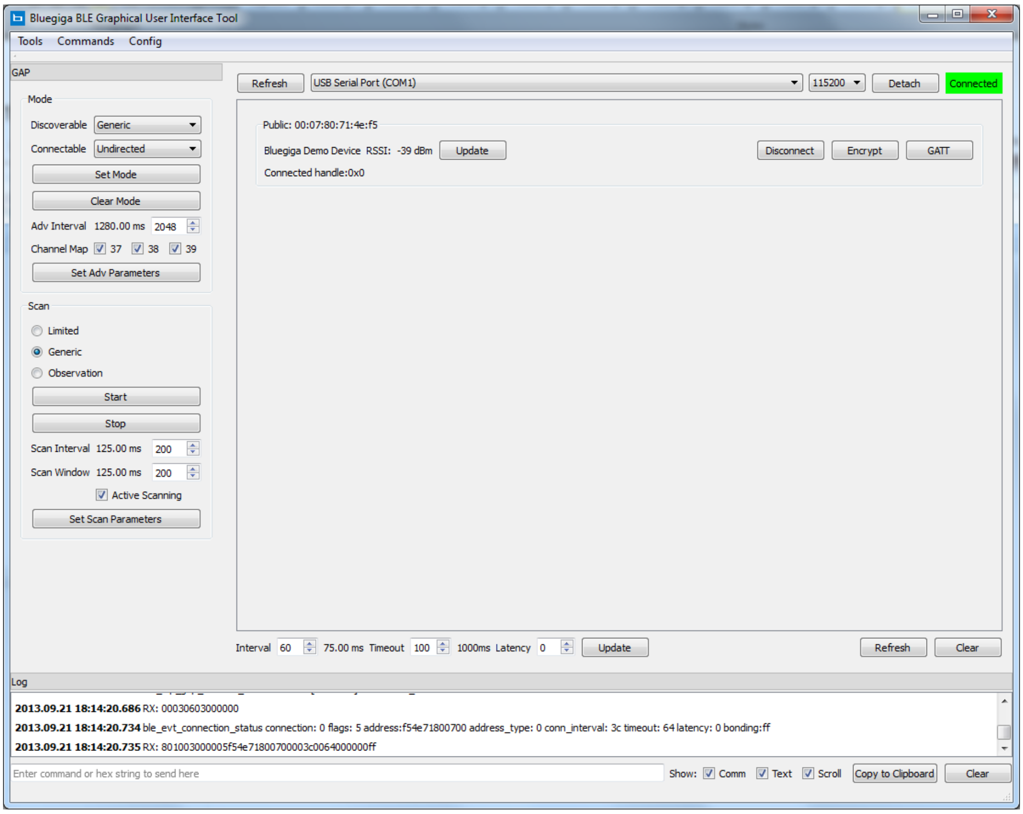
Enter command or hex string to send here

Show: ☑ Comm ☑ Text ☑ Scroll

Figure 19: Discoverting the BGDemo device

6.1.2 Establishing a Connection

Simply select the BGDemo sensor device and press the **Connect** button in the BLEGUI application.



USB Serial Port (COM1) 115200

Public: 00:07:80:71:4e:f5

Bluegiga Demo Device RSSI: -39 dBm

Connected handle:0x0

Mode

Discoverable: Generic

Connectable: Undirected

Adv Interval 1280.00 ms 2048

Channel Map ☑ 37 ☑ 38 ☑ 39

Scan

○ Limited

● Generic

○ Observation

Scan Interval 125.00 ms 200

Scan Window 125.00 ms 200

☑ Active Scanning

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:14:20.686 RX: 00030603000000
2013.09.21 18:14:20.734 ble_evt_connection_status connection: 0 flags: 5 address:f54e71800700 address_type: 0 conn_interval: 3c timeout: 64 latency: 0 bonding:ff
2013.09.21 18:14:20.735 RX: 801003000005f54e71800700003c0064000000ff
```

Enter command or hex string to send here

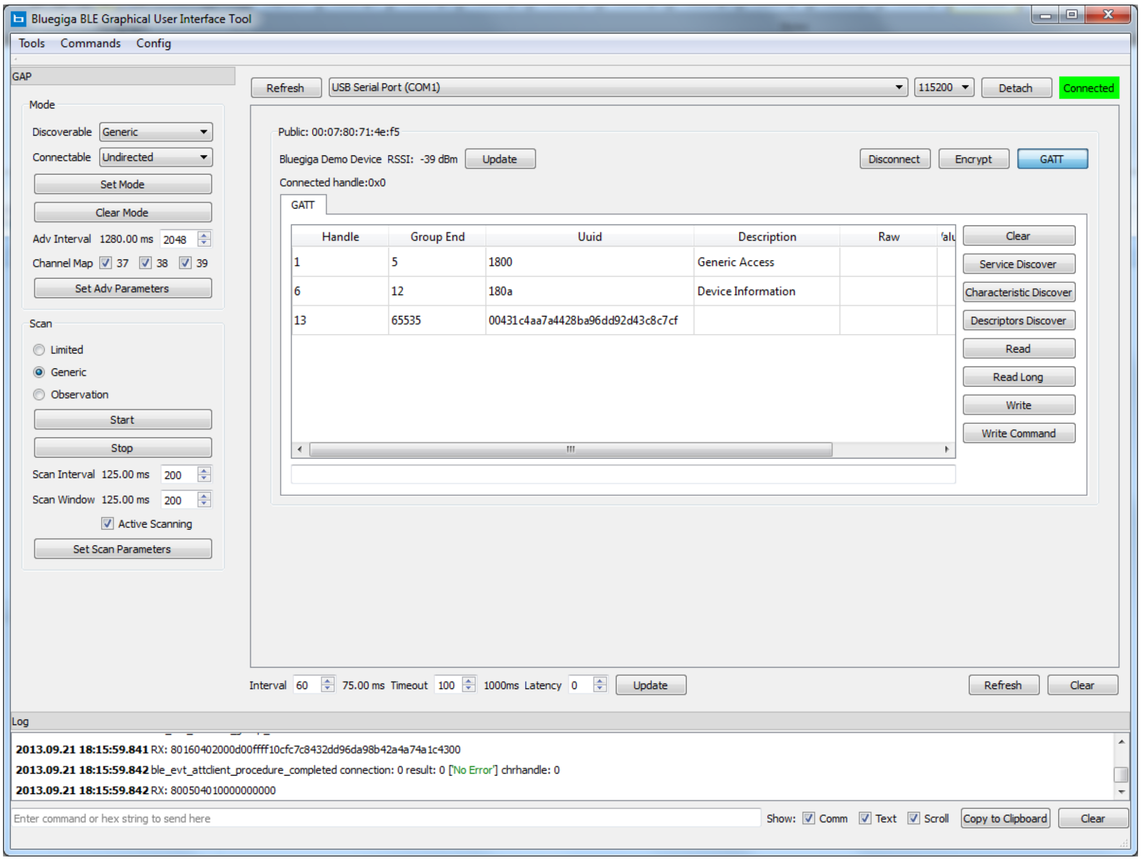
Show: ☑ Comm ☑ Text ☑ Scroll

Figure 20: Opening a connection

6.1.3 Making GATT Service Discovery

In order to see the supplied services in the BGDemo device do the following steps

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



USB Serial Port (COM1) 115200

Public: 00:07:80:71:4e:f5

Bluegiga Demo Device RSSI: -39 dBm

Connected handle:0x0

Mode

Discoverable: Generic

Connectable: Undirected

Adv Interval 1280.00 ms 2048

Channel Map ☑ 37 ☑ 38 ☑ 39

○ Limited
● Generic
○ Observation

Scan Interval 125.00 ms 200

Scan Window 125.00 ms 200

☑ Active Scanning

Handle	Group End	Uuid	Description	Raw	'alu
1	5	1800	Generic Access		
6	12	180a	Device Information		
13	65535	00431c4aa7a4428ba96dd92d43c8c7cf			

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:15:59.841 RX: 80160402000d00ffff10cfc7c8432dd96da98b42a4a74a1c4300
2013.09.21 18:15:59.842 ble_evt_attclient_procedure_completed connection: 0 result: 0 ['No Error'] chrhandle: 0
2013.09.21 18:15:59.842 RX: 8005040100000000000
```

Enter command or hex string to send here

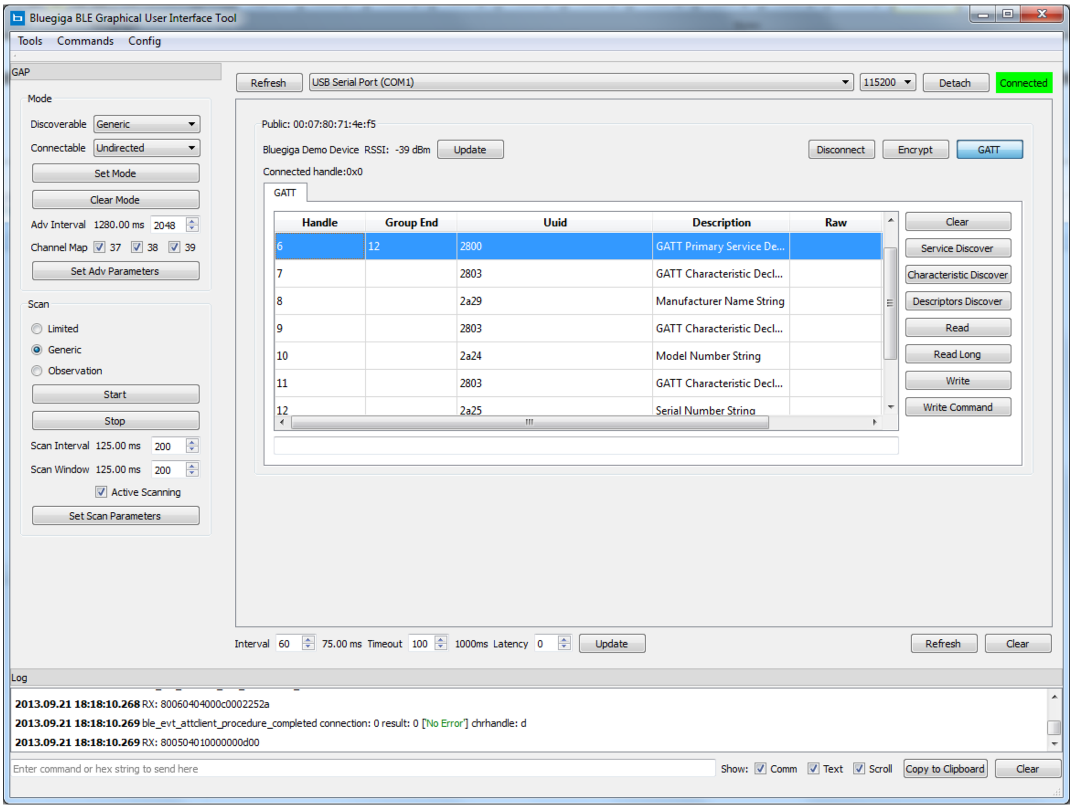
Show: ☑ Comm ☑ Text ☑ Scroll

Figure 21: GATT service discovery

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

6.1.4 Reading the Serial Number String

- To read the DI service's serial number string, which contains the MAC address, do the following steps:
  - Select the Device ID service (UUID: 180A)
  - Make **Descriptors discovery**



USB Serial Port (COM1) 115200

Mode

Discoverable: Generic

Connectable: Undirected

Adv Interval 1280.00 ms 2048

Channel Map ☑ 37 ☑ 38 ☑ 39

Scan

○ Limited
● Generic
○ Observation

Scan Interval 125.00 ms 200

Scan Window 125.00 ms 200

☑ Active Scanning

Public: 00:07:80:71:4e:f5

Bluegiga Demo Device RSSI: -39 dBm

Connected handle:0x0

Handle	Group End	Uuid	Description	Raw
6	12	2800	GATT Primary Service De...	
7		2803	GATT Characteristic Decl...	
8		2a29	Manufacturer Name String	
9		2803	GATT Characteristic Decl...	
10		2a24	Model Number String	
11		2803	GATT Characteristic Decl...	
12		2a25	Serial Number String	

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:18:10.268 RX: 80060404000c0002252a
2013.09.21 18:18:10.269 ble_evt_attclient_procedure_completed connection: 0 result: 0 ['No Error'] chrhandle: d
2013.09.21 18:18:10.269 RX: 800504010000000d00
```

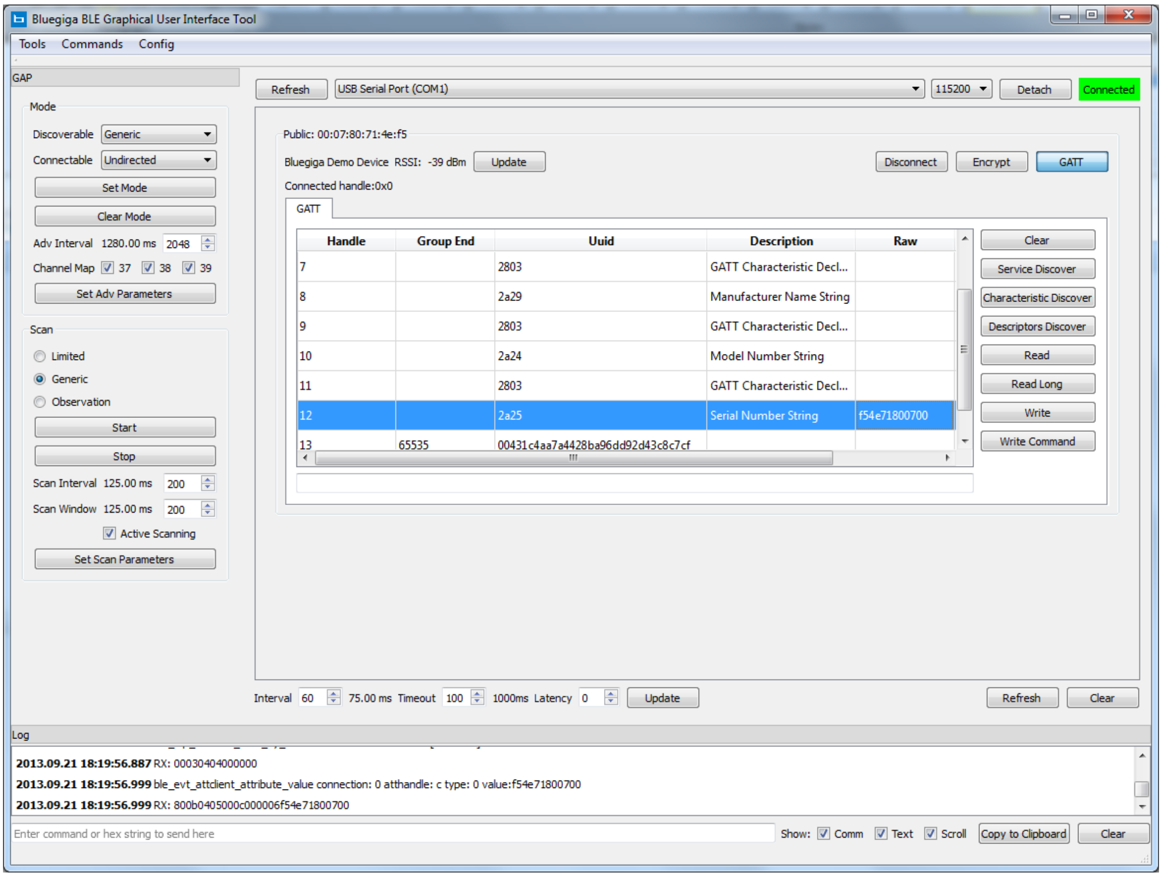
Enter command or hex string to send here

Show: ☑ Comm ☑ Text ☑ Scroll

Figure 22: GATT descriptor discovery

The Serial Number String is stored in the UUID a2a5 as defined in the GATT database. The value is read only and to read it:

- Select the **Serial Number String** characteristic
- Press **Read**



Discoverable: Generic
Connectable: Undirected
Adv Interval 1280.00 ms 2048
Channel Map ☑ 37 ☑ 38 ☑ 39
○ Limited
● Generic
○ Observation
Scan Interval 125.00 ms 200
Scan Window 125.00 ms 200
☑ Active Scanning

USB Serial Port (COM1) 115200

Public: 00:07:80:71:4e:f5
Bluegiga Demo Device RSSI: -39 dBm
Connected handle:0x0

Handle	Group End	Uuid	Description	Raw
7		2803	GATT Characteristic Decl...	
8		2a29	Manufacturer Name String	
9		2803	GATT Characteristic Decl...	
10		2a24	Model Number String	
11		2803	GATT Characteristic Decl...	
12		2a25	Serial Number String	f54e71800700
13	65535	00431c4aa7a4428ba96dd92d43c8c7cf		

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:19:56.887 RX: 00030404000000
2013.09.21 18:19:56.999 ble_evt_attclient_attribute_value connection: 0 atthandle: c type: 0 value:f54e71800700
2013.09.21 18:19:56.999 RX: 800b0405000c000006f54e71800700
```

Enter command or hex string to send here
Show: ☑ Comm ☑ Text ☑ Scroll

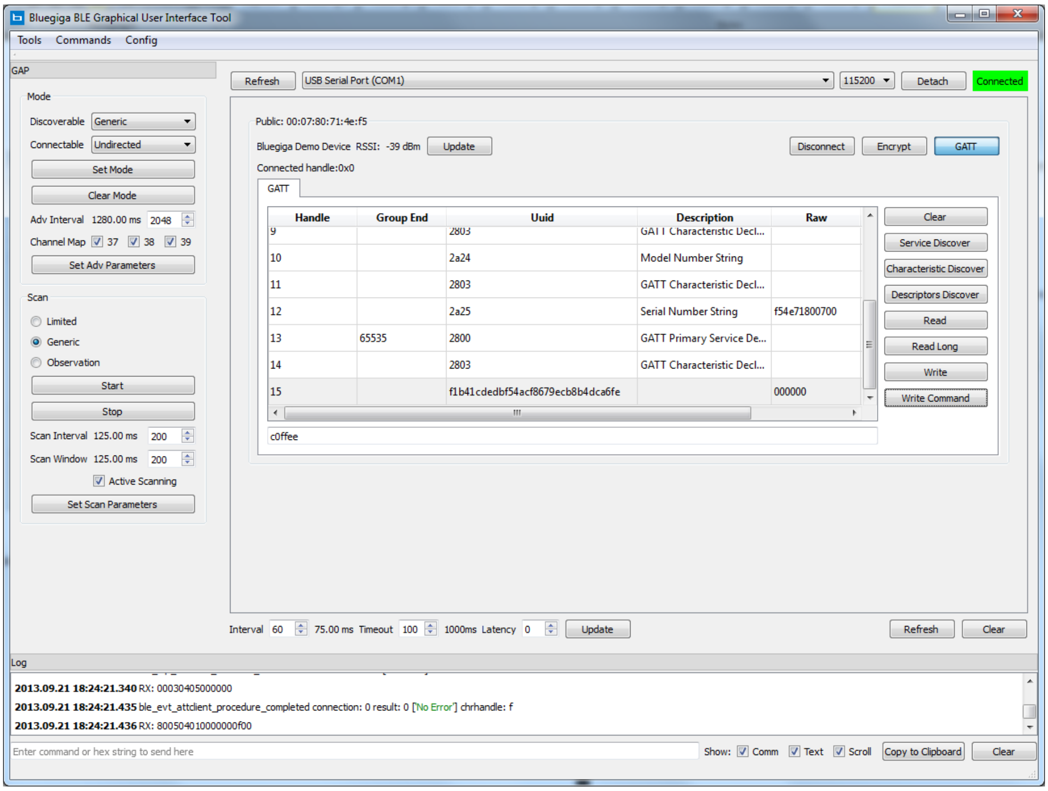
Figure 23: Reading Serial Number String

The MAC address is displayed in the **Raw** column.

6.2 Reading and Writing the Manufacturer Specific Service

In order to write and read the value of our proprietary characteristic

- **Connect** to the BGDemo sensor
- Make GATT **service discovery**
- Select the proprietary service and make **descriptor discovery**
- Press **Read** in order to read the value:
  - Note that the value does not contain any real data by default, since it was not marked as **const** but zero's are returned
- To write the value:
  - Select the proprietary characteristic
  - Write the desired value to the line below the GATT view (c0ffee)
  - Press **Write**



Discoverable: Generic
Connectable: Undirected
Adv Interval 1280.00 ms 2048
Channel Map ☑ 37 ☑ 38 ☑ 39

○ Limited
● Generic
○ Observation

Scan Interval 125.00 ms 200
Scan Window 125.00 ms 200
☑ Active Scanning

USB Serial Port (COM1) 115200

Public: 00:07:80:71:4e:f5
Bluegiga Demo Device RSSI: -39 dBm
Connected handle:0x0

Handle	Group End	Uuid	Description	Raw
9		2803	GATT Characteristic Decl...	
10		2a24	Model Number String	
11		2803	GATT Characteristic Decl...	
12		2a25	Serial Number String	f54e71800700
13	65535	2800	GATT Primary Service De...	
14		2803	GATT Characteristic Decl...	
15		f1b41cdedbf54acf8679ecb8b4dca6fe		000000

c0ffee

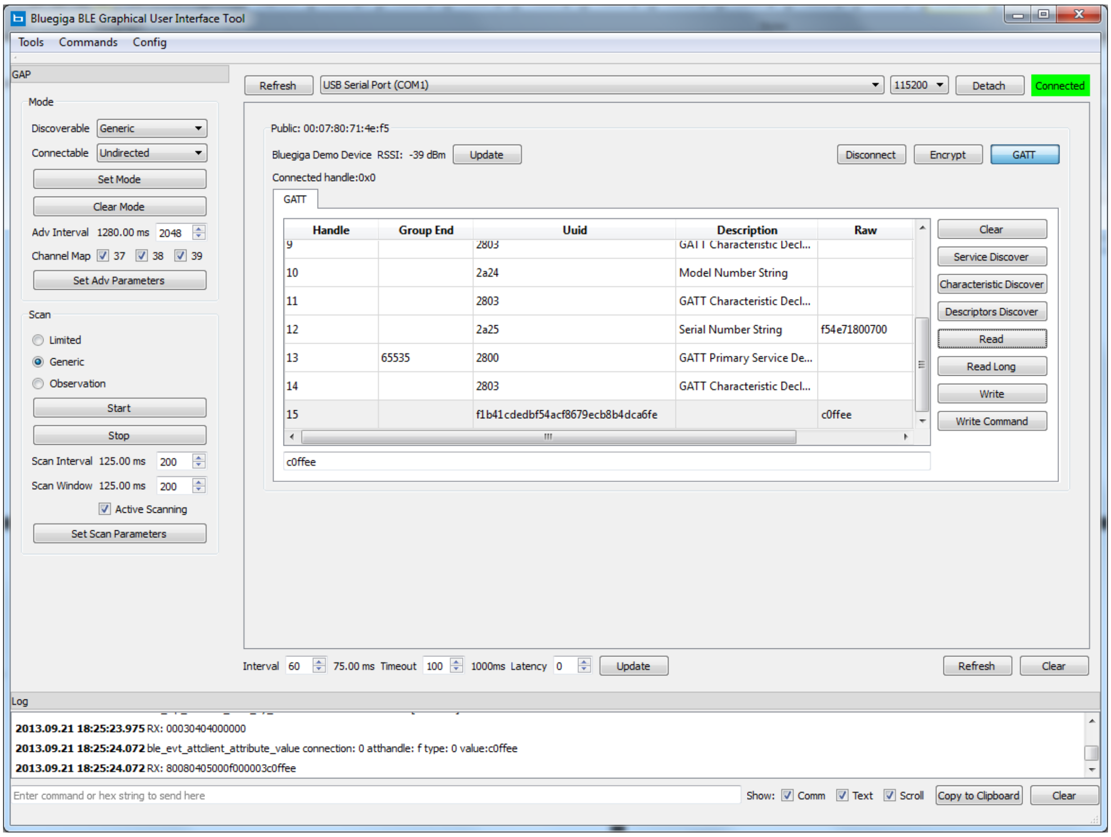
Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:24:21.340 RX: 00030405000000
2013.09.21 18:24:21.435 ble_evt_attclient_procedure_completed connection: 0 result: 0 ['No Error'] chrhandle: f
2013.09.21 18:24:21.436 RX: 80050401000000f00
```

Enter command or hex string to send here
Show: ☑ Comm ☑ Text ☑ Scroll

Figure 24: Writing a characteristic value

- To make sure the value got written, simply read it again.



USB Serial Port (COM1)

Discoverable: Generic
Connectable: Undirected
Adv Interval 1280.00 ms 2048
Channel Map ☑ 37 ☑ 38 ☑ 39

○ Limited
● Generic
○ Observation

Scan Interval 125.00 ms 200
Scan Window 125.00 ms 200
☑ Active Scanning

Public: 00:07:80:71:4e:f5

Bluegiga Demo Device RSSI: -39 dBm

Connected handle:0x0

Handle	Group End	Uuid	Description	Raw
9		2803	GATT Characteristic Decl...	
10		2a24	Model Number String	
11		2803	GATT Characteristic Decl...	
12		2a25	Serial Number String	f54e71800700
13	65535	2800	GATT Primary Service De...	
14		2803	GATT Characteristic Decl...	
15		f1b41cdedbf54acf8679ecb8b4dca6fe		c0ffee

c0ffee

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2013.09.21 18:25:23.975 RX: 00030404000000
2013.09.21 18:25:24.072 ble_evt_attclient_attribute_value connection: 0 atthandle: f type: 0 value:c0ffee
2013.09.21 18:25:24.072 RX: 80080405000f000003c0ffee
```

Enter command or hex string to send here

Show: ☑ Comm ☑ Text ☑ Scroll

Note:

If you reset the BGDemo sensor the value written to the proprietary characteristic will be lost, since the example BGScript code will not store the value to the flash memory.

If you want to store the value permanently use for example the PS key API commands to write the value to the PS key storage in your BGScript code.

Disconnecting from the device will keep the characteristic value, since as long as the software runs, the value will be kept in RAM.

7 Contact information

Sales:	sales@bluegiga.com
Technical support:	www.bluegiga.com/support/
Orders:	orders@bluegiga.com
WWW:	www.bluegiga.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 3972 2186
Bluegiga Technologies Ltd.
Unit 10-18
32/F, Tower 1, Millennium City 1
388 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong

BLUETOOTH SMART MODULE

CONFIGURATION GUIDE

Tuesday, 2 September 2014

Version 3.6

Table of Contents

1 Version History

Version	Comments
3.0	This document is separated from the Profile Toolkit Developer Guide document. Compatibility changes for the Bluetooth Smart Software v.1.2 added: • Added BLE113 reference for <txpower> in hardware.xml • Added fixed passkey documentation to config.xml • Bootloader definition added for OTA update. • USB interface is disabled by default • Default maximum power mode defined to be 3 • Wake up pin functionality added • Defrag tag added for running the defragmentation automatically in the boot-up. In addition, editorial improvements done within the document.
3.1	Improved examples and configuration option descriptions.
3.2	Compatibility changes for the Bluetooth Smart Software v.1.2.2: • 256kB variant configuration supported added for BLE113 • Binary image and memory configurability added for OTA SW update under the <ota> and <otaboot> tags
3.3	Improved examples
3.5	Compatibility changes for the Bluetooth Smart Software v.1.3.0: • Hardware configuration and TX power parts added for BLE121LR product variant • OTA firmware update instructions added to create an OTA file just containing the BGScript and GATT portions
3.6	Compatibility changes for the Bluetooth Smart Software v.1.3.1: • <DFU> tag introduced for allowing that DFU boot-mode is not allowed.

2 Introduction

The *Bluetooth* Smart configuration guide instructs you how to how to create a project file for your application and how to configure your *Bluetooth* Smart Modules hardware and application configuration settings.

3 Project Configuration File

The project file (typically ***project.bgproj*** *or* ***project.xml)*** is the file that describes all the components included in a Bluetooth Smart software and hardware project. Typically these files are name something like this:

- **hardware.xml** - Hardware configuration file
- **gatt.xml** - GATT database file
- **config.xml** - Optional application configuration file
- **script.bgs** - Optional BGScript application source code
- **cdc.xml** - Optional USB descriptor file (not used with BLE113, BLE121LR)

The project file also defines other features of the project like the hardware version (BLE112, BLE113, and BLE121LR), firmware output files and selected bootloader.

The project file itself is a simple XML file with only few tags on it, which are described below.

⚠ If the project file is named as ***project.bgproj*** (or any other file with a ***.bgproj*** extension), then the Bluegiga **BLE Update** application will automatically recognize it in the Windows Explorer interface and allow you to easily open, compile, and flash the project to a module using the CC debugger.

3.1 <device>

Hardware type configuration

XML tag	Description
type	This tag defines the type of your *Bluetooth* Smart Module **Options:** **ble112**: Use if you have BLE112 or BLED112 **ble113**: Use if you have BLE113 **ble113-m256k**: Use if you have BLE113-M256K **ble121lr-m256k**: Use if you have BLE121LR longrange module **Default:** ble112
Example: Defining device type BLE112 *<device type="ble112" />*	
Example: Defining device type BLE113 (128kB flash variant) *<device type="ble113" />*	
Example: Defining device type BLE113 (256kB flash variant) *<device type="ble113-m256k" />*	

3.2 <gatt>

GATT database file

XML tag	Description
in	This tag points to the XML file that contains the GATT database structure.
Example: Defining the GATT database file *<gatt in="gatt.xml" />*	

3.3 <hardware>

Hardware configuration file

XML tag	Description
in	This tag points to the XML file which contains the hardware configuration for your Bluegiga *Bluetooth* Smart device.
Example: Defining the hardware configuration file *<hardware in="hardware.xml" />*	

3.4 <config>

Application configuration file (optional)

XML tag	Description
in	This tag points to the XML file which contains the generic application configuration of your Bluegiga *Bluetooth* Smart device.
Example: Defining the application configuration file *<config in="config.xml" />*	

3.5 <script>

BGScript application file (optional)

XML tag	Description
in	This tag points to the BGScript file that contains the BGScript source code for your standalone *Bluetooth* Smart application. If you use the BGAPI protocol and a separate host (which cannot be used simultaneously with BGScript code), then this tag should be left out.
Example: Defining the BGScript file *<script in="script.bgs" />*	

3.6 <usb_main>

USB descriptor definition (optional)

XML tag	Description
in	This tag points to the XML file that contains the USB descriptor for BLED112 or BLE112 *Bluetooth* Smart devices. If USB interface is disabled in the hardware configuration, this tag is not needed.

XML tag	Description
Example: Defining the USB descriptor file *<usb_main in="cdc.xml" />*	

USB enumeration

The USB only descriptors which may be used the ones contained in the **cdc.xml** file that is present in many of the example projects which come with the stack, providing USB CDC functionality (virtual serial port). There is no support in the current BLE stack for other types of USB enumeration such as USB HID or other protocols.

3.7 <image>

Firmware binary output file

XML tag	Description
out	This tag names the firmware output file for the compiler. The output file can be uploaded to the device using the CC debugger or an available wired DFU method (USB or UART). The DFU option depends on which bootloader is present in the firmware that is already on the module from a previous full CC debug reflash, and the module must be specifically rebooted into DFU mode first. The BLEGUI utility implements both of these methods (USB and UART) via the **Commands -> DFU** menu.
Example: Defining the binary output file for the compiler *<image out="out.hex" />*	

3.8 <ota>

This optional tag is used to generate a firmware file that can be uploaded to the device using Over-the-Air (OTA) update.

XML tag	Description
out	This tag names the OTA firmware output file from the compiler. The output file can be uploaded to the device using an Over-the-Air (OTA) firmware update process/tool (such as BLEGUI).
firmware	This tag defines if only the GATT and configuration portions are included in the OTA output file. **Note:** The GATT and configuration portions must match with the SDK version against which they will be updated. **Options:** **true**: Include Bluetooth Stack in the OTA update (firmware) **false**: Do not Bluetooth Stack in the OTA update (firmware) **Default:** True
Example: Defining a full OTA firmware update file *<ota out="out.ota" firmware="true" />*	

XML tag	Description
Example: Defining an OTA firmware update file containing just BGScript and GATT database *<ota out="out.ota" firmware="false" />*	

3.9 <boot>

Selects the bootloader interface used for In-the-Field or Over-the-Air firmware updates.

XML tag	Description
fw	This tag is used to describe the boot loader used in the firmware. The boot loader also devices which interface is used for the on-the-field firmware updates. **Only one bootloader can be active in the device.** **Options:** **boot:** Configures the bootloader for the USB interface. Use only with the BLE112 module or BLED112 dongle. **bootuart:** Configures the bootloader for the UART interface. **bootota:** Configures the bootloader for Over-the-Air (OTA) interface. **Default:** **boot**
Example: Enabling UART bootloader *<boot fw="bootuart" />*	
Example: Enabling USB boot loader *<boot fw="boot" />*	
Example: Enabling OTA boot loader *<boot fw="bootota" />*	

3.10 Examples

Below is an example of a project file for BLE112 Bluetooth Smart Module or BLED112 USB dongle using USB interface:

BLE112 Project

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware.xml" />
    <usb_main in="cdc.xml" />
    <config in="config.xml" />
    <device type="ble112" />
    <boot fw="boot" />
    <image out="BLE112_usbcdc.hex" />
</project>
```

Below is an example of a project file for BLE113 Bluetooth Smart Module using UART interface for potential DFU updates:

BLE113 Project

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware.xml" />
    <config in="config.xml" />
    <device type="ble113" />
    <boot fw="bootuart" />
    <image out="BLE113.hex" />
</project>
```

Below is an example of a project file for BLE113 Bluetooth Smart Module running a BGScript application and OTA bootloader:

BLE113 Project

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware.xml" />
    <config in="config.xml" />
    <device type="ble113" />
    <boot fw="bootota" />
    <image out="BLE113.hex" />
</project>
```

4 Hardware Configuration File (hardware.xml)

The hardware configuration file is used to configure the hardware features such as TX power, UART, SPI, hardware timers, and GPIO settings of your Bluegiga *Bluetooth* Smart device.

4.1 <sleeposc>

Sleep oscillator settings

Attribute	Value - Description
enable	This setting can be used to enable or disable the external sleep clock. **Options:** **true:** This enables the external 32.7680KHz sleep oscillator. This sleep oscillator allows the BLE112, BLE113, or BLE121LR to enter power mode 1 or 2 whenever radio is not active, transmitting or receiving, for example also during radio silence between connection intervals. **false**: This disables the external 32.7680KHz sleep oscillator, so the TI's chipset internal 32.7530KHz RC oscillator is used for timings. Using this setting increases current consumption because power modes 1 and 2 are prevented during any *Bluetooth* activity (connection - scanning - advertising), due to limited accuracy of internal RC oscillator. **Default:** **false** **Note:** In BLE112, BLE113, and BLE121LR this options MUST be configured to enable the external sleep oscillator, while in the BLED112 this option MUST be set to "false", since the USB dongle does not contain the required external oscillator.
ppm	This setting defines the sleep clock accuracy and **must always be 30**. **Options:** **30** **Note:** Do not modify**!**
Example : Configuration for BLE112, BLE113, BLE121LR *Bluetooth* Smart Modules: *<sleeposc enable="true" ppm="30" />*	
Example : Configuration for BLED112 USB dongle: *<sleeposc enable="false" ppm="30" />*	

4.2 <script>

BGScript settings

Attribute	Value - Description
enable	This setting can be used to enable or disable BGScript application execution. **Options:** **true:** BGScript application and VM are enabled. **false:** BGScript application and VM are disabled and BGAPI should be used instead. **Default:** **false**
Example: Enable scripting *<script enable="true" />*	

4.3 <slow_clock>

This setting can be used to slow the system clock from 32MHz to 250KHz when radio is active, in order to reduce the peak power consumption. The average current consumption reduction between normal clock speed and slow clock speed is approximately 5-6 mA.

Attribute	Value - Description
enable	**Options:** **true:** System clock is slowed down. **false:** System clock is not slowed down **Default:** **false**
Example: Enable slow clock *<slow_clock enable="true" />*	

UART and PWM interfaces use system clock for timings. If the system clock is allowed to slow down (notice that it will happen only when radio is active) the peripheral interface timings become variable, thus invalid. This feature must only be enabled when peripherals requiring stable clock are not used.

SPI Master sends clock signal with transmission which allows enabling the slow clock feature.

4.4 <lock_debug>

This feature can be used to lock down the debug interface (CC debugger interface, P2_1/P2_2) on the BLE112, BLE113, and BLE121LR *Bluetooth* Smart Modules in order to protect application code and data. If this feature is enabled, then only a **full erase** of the firmware can be done with the CC debugger using the TI's Smart RF Flash Programmer. Notice that Bluegiga's own re-flash tool would not be usable anymore, and for instance it would become impossible to retrieve the serial number and license key of a module.

Attribute	Value - Description
enable	**Options:** **true:** Debug interface is locked. **false:** Debug interface is available. **Default:** **false**
Example: Lock debug interface *<lock_debug enable="true" />*	

4.5 <sleep>

This setting can be used to enable or disable sleep modes.

Attribute	Value - Description
enable	**Options:** **true:** All power modes can be enabled. Selection of power modes is done automatically by the firmware. Firmware will select the best power saving mode automatically to achieve lowest possible power consumption. **false:** Use this to prevent the firmware from entering any of the sleep modes. **Default:** **true**
max_mode	Maximum power mode device is allowed to use. **Range:** **1-3** **Default:** **3**
Example : Allow power modes 1 and 2 and disable power mode 3. *<sleep enable="true" max_mode="2" />*	

⚠ When sleep mode (power mode 1, 2, or 3) is enabled and the module is not kept awake (for example by using the wake-up pin), then the *Bluetooth* Smart module **will not respond** to any BGAPI commands or process any other incoming sent to it via UART. If you want to enable sleep mode and use the UART interface to communicate with the module, you need to enable the wake-up pin feature (described below) and provide a wake-up signal from an external host.

4.6 <wakeup_pin>

This feature is used to prevent the *Bluetooth* Smart module from entering any sleep modes like power mode 3, or alternatively used to to wake it up if it has entered a low power mode. If you use UART to communicate with the module, then you need to enable this feature and assert the relevant pin before sending any streaming data or BGAPI commands to the module, and keep it asserted until the last byte has been clocked into the module over the UART RX pin.

The wake-up pin functionality can only be assigned to a single GPIO, but you can still assign normal GPIO interrupts to other pins using BGAPI/BGScript commands. The difference between this special wake-up pin operation and normal GPIO interrupts is that this pin will not only generate the interrupt which wakes the module, but will also keep the module awake as long as it is held in the asserted state. Normal GPIO interrupts can wake the module from any state (even power mode 3), but after the interrupt event handler completes, the module will return to sleep (if sleep is enabled and not prevented via the wake-up pin).

The correct procedure for using the wake-up pin to send BGAPI packets over UART is as follows:

1. Assert the wake-up pin from an external host
2. Process the "**hardware_io_port_status**" BGAPI event packet which is generated and sent out the module's TX pin
3. Send the desired BGAPI command packet to the module
4. Wait until you receive **at least the first byte of the BGAPI response packet** before de-asserting the wake-up pin

Important:

- Step 2 above is critical because some sent data may be ignored if you do not process the port status event before starting to send data.
- Step 4 above is critical because if you de-assert the wake-up pin too soon (e.g. immediately after the last byte is placed in the TX buffer of the attached UART host), then the last byte or two may not be properly clocked into the module before it goes to sleep again, resulting in lost or corrupt data.

attribute	description
enable	Used to enable wake-up pin feature. Wake-up pin wakes the device up from a sleep mode or prevents it from entering it again. **Options:** **true:** wake-up pin is enabled **false:** wake-up pin is disabled
port	Defines the port where wake-up pin is. **Options:** **0-1**
pin	Defines the pin inside the selected port. **Options:** **0-7**
state	Logic state for wake-up pin. **Options:** **up** **down**

attribute	description
	Default: **up**
Example: Enabling wake-up on P0_0 *<wakeup_pin enable="true" port="0" pin="0" state="up" />*	

When this pin is pulled, the *Bluetooth* Smart module does not enter any sleep modes which increases power consumption.

When this pin is used to wake up the *Bluetooth* Smart module from sleep mode, a **hardware_io_port_status** API event is triggered immediately, since it's handled as a normal GPIO interrupt. You should expect this event to occur and either handle it or ignore it intentionally if you are using external control via the BGAPI protocol.

4.7 <host_wakeup_pin>

This pin can be used to wake up an external host from sleep when the *Bluetooth* Smart module has data to send over the UART interface. The external host should then use flow control signals (or wake immediately) so that the module can send data to it.

Notice that the host wake-up pin is only meant to wake up the host from a sleep mode and it does not necessarily remain active during the UART transmission. The host therefore should not go back to sleep after the host wake-up pin is de-asserted, but only after all the expected data has been received over UART.

attribute	description
enable	Use to enable the host wake-up pin feature. Host wake-up pin is asserted when the *Bluetooth* Smart module has data to send. **Options:** **true:** host wake-up pin is enabled **false:** host wake-up pin is disabled
port	Defines the port used for the host wake-up. **Options:** **0-2**
pin	Defines the pin inside the selected port. **Options:** **0-7**
state	Logic state for host wake-up signal. **Options:** **up** **down** **Default:** **up** **Example:**
Example: Enabling wake-up on P1_1 *<host_wakeup_pin enable="true" port="1" pin="1" state="up" />*	

4.8 <txpower>

This can be used to configure the TX output power used since boot. Values represent roughly equal linear divisions between the minimum and maximum output power as noted in the ***power*** attribute description.

Attribute	Value - Description
power	Default TX power setting **Range:** **0-15** **BLE112 (BLED112):** 15 is the highest TX power setting and equals roughly to 3dBm (0dBm), while 0 is the lowest value and corresponds to around -24dBm. **BLE113:** 14 is the highest TX power setting and equals roughly to +0dBm, while 0 is the lowest value and corresponds to around -24dBm. **BLE121LR:** 9 is the highest TX power setting and equals roughly to +8dBm, while 0 is the lowest value and corresponds to around -10dBm. Using a value of 15 with the BLE113, or using any value between 10 and 15 with the BLE121LR, is the same as using respectively their max values of 14 or 9.
bias	TX power amplifier bias setting. Do not modify. **Options:** **5**
Example: BLE112 with +3 dBm TX power *<txpower power="15" bias="5" />*	
Example: BLE112 with 0 dBm TX power *<txpower power="13" bias="5" />*	
Example: BLE113 with 0 dBm TX power *<txpower power="15" bias="5" />*	

4.9 <pmux>

This setting is used to configure the control pin for an external DC/DC converter which can be used to reduce the peak TX and RX power consumption. A GPIO pin needs to be dedicated to control the DC/DC converter's **enable** or **bypass** modes. Any GPIO pin from Port 1 can be dedicated as the DC/DC control pin and the firmware will automatically control the pin depending on the Bluetooth transmission and reception states.

The BLE development kits contain an external DC/DC converter which is specifically designed to work with the internal CC254x radio chipset. When the GPIO pin defined with **<pmux>** is high, the DC/DC converter is enabled, and when the GPIO pin is low, the converter is disabled. Note that the circuit is design to **disable** the converter at all times except when the radio is active. By doing this, the input voltage is dropped to 2.1V only when the radio is on and the resulting current consumption is less during transmissions. This is particularly beneficial because of the battery chemistry of most small coin cells. The reduced current draw during transmissions will notably extend the life of a typical CR2032 cell.

attribute	description
regulator_pin	Defines the output pin for the external DC/DC converter in Port 1. **Range:** **0-7** **Note:** Only pins of **Port 1** can be used to control the DC/DC converter. With the BLE121LR only pin P1_7 can be used.
clock_pin	Defines the output pin in Port 0 for a 32.768 kHz clock signal, which can be used to provide the clock value to external devices. **Range:** **0-7** **Note:** Only **Port 0** can be used for clock signal output.
Example: This is for DKBLE112 and DKBLE113 with DC/DC control on P1_7 and no clock signal in use *<pmux regulator_pin="7" />*	

4.10 <port>

This setting is used for the I/O port configuration settings (input only).

attribute	description
index	Port index to configure **Range:** **0-2**
tristatemask	Tristate configuration (bit mask) for port. For the pins defined with this bit mask, no high/low pull will be used, but the pins will be in tristate mode. **Range:** **0x00 - 0xFF** For example 0x02 means pin number 1 is configured to be tristated instead of being pulled high/low.
pull	Defines the pull direction. **Options:** **up**: Pins are pulled up **down**: Pins are pulled down **Note:** The pull direction will affect the whole port and and individual pin directions cannot be configured.
Example : pulling all pins in Port 0 down *<port index="0" tristatemask="0" pull="down" />*	

⚠ By default all the ports except P1_0 and P1_1 are configured as inputs with pull-ups. P1_0 and P1_1 should be configured as outputs or pulled up externally.

All unused I/O pins should have a defined level and should not be left floating. This can be done by leaving the pin unconnected and by configuring the pin as a general-purpose I/O input with a pull-up resistor. Alternatively the pins can be configured as a general-purpose I/O output. In either case, the pins should not be connected directly to VDD or GND, in order to avoid excessive power consumption.

⚠ Port 2 pins currently do not support interrupts. They may still be pulled up or down with the above configuration in hardware.xml, but BGScript/BGAPI commands to enable interrupts on P2_* pins will not have any effect. Only Port 0 and Port 1 pins support interrupts.

4.11 <usb>

USB interface settings:

Attribute	Value - Description
enable	Enables or disables the USB interface. **Options:** **true:** Use this to enable the USB interface. **false:** Use this to disable the USB interface. **Default:** **false**
endpoint	Configures the USB interface usage purpose. **Options:** **none:** USB can be controller with a BGScript application **api:** USB is used for the BGAPI protocol **test:** See endpoint section for more information **script:** do not use **usb:** See endpoint section for more information **uart0:** See endpoint section for more information **uart1:** See endpoint section for more information See: Endpoints available below.
Example : Enabling BGAPI over USB *<usb enable="true" endpoint="api" />*	
Example : Enabling USB access for BGScript *<usb enable="true" endpoint="none" />*	

In the BLED112, the interface must always be enabled or the dongle becomes non-communicative, resulting in a potentially bricked device.
In the BLE112, this should be set to false, unless the interface is really needed, since USB constantly uses 5+ mA of current.
In the BLE113 and BLE121LR, this must always be set to false, since this module does not have a USB interface.

4.12 <usart>

This setting is used to configure the USART interface of the BLE112, BLE113, or BLE121LR *Bluetooth* Smart modules.

In UART mode, the number of data bits is 8 and parity is set to none. Number of data bits and parity cannot be reconfigured.

attribute	description
channel	USART channel to configure **Options:** **0:** USART channel 0 **1:** USART channel 1
baud	USART baudrate and SPI master clock. **Range:** **1200 - 2000000**
alternate	Alternate configuration option for USART. **Options:** **1:** Alternative configuration 1 (see data sheet for details) **2:** Alternative configuration 2 (see data sheet for details)
endpoint	Configures the UART interface usage purpose. **Options:** **none:** USART interface can be controller with a BGScript application **api:** USART is configured as the host interface making use of the BGAPI protocol **Note:** The BGAPI protocol is not available over the interface operating in SPI mode.
mode	USART operation mode. **Options:** **uart:** USART is configured as UART interface. When BGAPI is used over UART in this mode, hardware flow control MUST be used. **packet:** USART is configured as UART interface using the BGAPI packet mode. This allows BGAPI to be used over UART without hardware flow control. **spi_master:** USART is configured as SPI master. **spi_slave:** USART is configured as SPI slave. **Not recommended to be used due to the SPI slave interface limitations (see below).** **Default:** **uart** **Note:**

attribute	description
	See the BGAPI protocol description from the API reference manual for more information about the packet mode.
polarity	SPI polarity configuration **Options:** **positive:** Configures the SPI clock polarity to be positive **negative:** Configures the SPI clock polarity to be negative **Default:** **negative**
phase	SPI clock phase **Options:** **0** **1** **Default:** **1**
endianness	SPI bit ordering **Options:** **msb:** most signigicant bit **lsb:** least significant bit
flow	UART flow control setting **Options:** **true:** Hardware flow control (RTS and CTS) enabled **false:** Hardware flow control (RTS and CTS) disabled **Default:** **true**
stop	UART stop bit logic **Options**: **high** **low** **Default:** **high**
start	UART start bit logic **Options**: **high** **low**

attribute	description
	Default: **low** **Note:** Must be different than stop bit logic.
stopbits	UART stop bits **Options**: **1:** One stop bit **2:** Two stop bits **Default:** **1**
Example : Enabling BGAPI over UART on DKBLE *<usart channel="1" alternate="1" baud="115200" endpoint="api" />*	
Example : Enabling UART access for BGScript on DKBLE *<usart channel="1" alternate="1" baud="115200" endpoint="none" />*	
Example : Enabling SPI master interface on DKBLE to control the display *<usart channel="0" mode="spi_master" alternate="2" polarity="positive" phase="1" endianness="msb" baud="57600" endpoint="none" />*	

⊖ **SPI slave limitations**

The Bluegiga BLE modules are really only practical as a SPI master. It has only a 1-byte hardware buffer in the USART which implements SPI functionality, and the BLE stack doesn't currently provide any methods for generating an API-level interrupt when there is new data coming in from the master (e.g. when the Slave Select pin is asserted or when data is clocked in). This means that SPI slave functionality requires constant polling and very slow data transfers. Additionally, **there is no BGAPI control possible over the SPI interface**, so even this very limited implementation is only usable with a BGScript-based application.

4.13 <timer_ticks>

This configuration controls a global prescaler for Timer 1, Timer 3, and Timer 4. The prescaler value (***speed*** attribute) can be set to a value between 0.25 MHz to 32 MHz (while the system clock is fixed at 32 MHz, that is, when <slow_clock> is set to false).

This setting can be used to slow down the clock value to give to the timer and generate longer values when using for example PWM output signals.

attribute	description
speed	Timer tick settings. **Options:** **0**: 32 MHz **1**: 16 MHz **2**: 8 MHz **3**: 4 MHz **4**: 2 MHz **5**: 1 MHz **6**: 500 kHz **7**: 250 kHz
Example : 32 MHz timer *<timer_ticks speed="0" />*	

4.14 <timer>

This configuration is used to configure the hardware timer(s) of the BLE112/113 module. **Timer 2** is reserved for internal use by the BLE stack.

attribute	description
index	Timer index to configure. **Options:** **1**: Timer 1 **3**: Timer 3 **4**: Timer 4
enabled_channels	Enabled channels for specified timer. **Range:** **0x00 - 0xFF**
divisor	Divisor for specified timer . **Timer 1:** **0**: Tick frequency/1 **1**: Tick frequency/8 **2**: Tick frequency/32 **3**: Tick frequency/128 **Timer 3 and Timer 4:** **0**: Tick frequency/1 **1**: Tick frequency/2 **2**: Tick frequency/4 **3**: Tick frequency/8 **4**: Tick frequency/16 **5**: Tick frequency/32 **6**: Tick frequency/64 **7**: Tick frequency/128
mode	Operating mode for specified timer. **Timer 1:** **0** : Suspended **1** : Free running **2** : Modulo **3** : Up/Down

attribute	description
	Timer 3 and Timer 4: **0** : Free running **1** : Down **2** : Modulo **3** : Up/Down
alternate	Alternate configuration for specified timer. **Options:** **1:** Alternative configuration 1 (see data sheet for details) **2:** Alternative configuration 2 (see data sheet for details)
Example: 4-channel PWM configuration *<timer index="1" enabled_channels="0x1f" divisor="0" mode="2" alternate="2" />*	

4.15 <otaboot>

Bootloader configuration for Over-the-Air update.

attribute	description
source	Source where image is updated from. **Options:** **external:** External SPI flash memory is used **internal:** Internal memory is used (requires 256kB internal flash module variant)
uart	SPI USART channel to which external flash chip is connected. **Options:** **0:** USART channel 0 **1:** USART channel 1
cs_port	Chip select port for SPI memory **Options:** **0:** Port 0 **1:** Port 1
cs_pin	Chip select pin for SPI memory **Options:** **0-7:** Pin 0 to pin 7
power_port	Power port for SPI memory **Options:** **0:** Port 0 **1:** Port 1 **Note:** P1_0 and P1_1 are recommended since they can provide high power output and can power the flash chip directly.
power_pin	Power pin for SPI memory **Options:** **0-7:** Pin 0 to pin 7 **Note:** P1_0 and P1_1 are recommended since they can provide high power output and can power the flash chip directly.
Example: Enabling external SPI flash board on DKBLE *<otaboot source="external" uart="0" cs_port="1" cs_pin="2" power_port="1" power_pin="0" />*	

4.16 Endpoints

The possible endpoint values used either for USB or UART are listed below:

Value	description
none	Data can be read from/written to BGScript when using ***system_endpoint_tx*** command and ***system_endpoint_rx*** event in BGScript code.
api	Endpoint is connected to BGAPI protocol.
test	Endpoint is connected to UART *Bluetooth* testing purposes.
script	**Do not use.**
usb	Endpoint is connected to USB interface.
uart0	Endpoint is connected to UART0 interface.
uart1	Endpoint is connected to UART1 interface.

4.17 Examples

Example for BLED112 USB dongle to enable BGAPI protocol over USB interface:

```
<?xml version="1.0" encoding="UTF-8" ?>
   <hardware>
       <txpower power="15" bias="5" />
       <usb enable="true" endpoint="api" />
       <sleeposc enable="false" ppm="30" />
   </hardware>
```

Below is an example of hardware configuration file used with BLE112, BLE113 or BLE121LR module, which uses BGAPI protocol over UART on DKBLE. Also the DC/DC control pin is enabled to control the external DC/DC converter and the wake-up pin is enabled in P0_0 pin (button).

⚠ **Never use the configuration below with a BLED112 USB dongle.**

```
<?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="1" alternate="1" baud="115200" flow="true" endpoint="api" />
    <wakeup_pin enable="true" port="0" pin="0" />
    <port index="0" tristatemask="0" pull="down" />
    <pmux regulator_pin="7" />
</hardware>
```

5 Application Configuration File (config.xml)

This application configuration file is used to configure some of the *Bluetooth* Smart Software's features such as the number of maximum connections. This file is optional.

5.1 <connections>

Defines the maximum number of connections that are supported by the firmware.

Attribute	Value - Description
value	Defines how many connections are supported. Affects how much RAM to reserve for connections. **Range:** **1 - 8** **Default:** **1**
Example : Enabling one (1) connection *<connections value="1" />*	
Example : Enabling eight (8) connections *<connections value="8" />*	

⚠ When more then one (1) connection is supported in the **config.xml** file, then connection interval values (minimum and maximum) used in **all** connection commands must be divisible by ***connections*** * ***2.5ms***

Examples:

If three (3) connections are supported, then the connection interval range has to contain limit values that are divisible by ***3 * 2.5ms*** = ***7.5ms***. In this case, any multiple value of **7.5ms** can be used, such as **7.5ms**, **15ms**, **22.5ms**, **30ms**, etc.

Alternatively, if two (2) simultaneous connections are supported, the interval values must be divisible by **5ms**. Notice that in this case, the lowest possible interval of **7.5ms** cannot be used because it is not divisible by **5.0ms**, so only larger connection intervals such as **10ms**, **15ms**, etc. can be used.

If only one (1) connection is supported, then any connection interval can be used when issuing connection commands.

5.2 <defrag>

Defines whether the persistent store is defragmented automatically at boot time.

Attribute	Value - Description
enable	Defragmentation enabled **Options:** **true:** Defragmentation run at boot **false:** Defragmentation during boot disabled

Attribute	Value - Description
	Default: **true**

5.3 <manual_confirm>

If this tag exists in the **config.xml** file, then manual confirmation of attribute indications will be enabled. **Note that it only needs to exist** and does not take any attributes.

When the *Bluetooth* Smart stack receives attribute indications from a remote device, it produces an ***attclient_attribute_value*** event to the host, where the type is ***attclient_attribute_value_type_indicate_rsp_req***. The host (application) must respond to this event with the ***attclient_indicate_confirm*** command after it has properly handled the indication to acknowledge that the data has been received.

This feature can be used by the host software to acknowledge the indication data, and this provides extra reliability in some kinds of application. If this tag is not present, then the BLE stack will automatically acknowledge indications upon reception.

Attribute	Value - Description
	Enables or disables manual indication confirmations.
Example: Enabling manual confirmations *<manual_confirm />*	

5.4 <script_timeout>

Defines maximum number of steps (commands) a BGScript can run within an event before a **system_script_failure** is raised.

Attribute	Value - Description
value	Maximum number of steps a BGScript can take. **Range:** **0 - 65535** **Default:** **1000**
Example : disabling script timeout feature *<script_timeout value="0" />*	
Example : Limiting BGScript steps to 10000 *<script_timeout value="10000" />*	

⚠ This timeout is especially recommended to be used when developing BGScript applications into BLED112 USB dongle.

5.5 <throughput>

Defines how data packets are sent over the air during each connection interval.

Attribute	Value - Description
optimize	Throughput optimization setting **Options:** **power:** Only a single packet is sent at each connection interval. This setting minimizes power consumption, but might limit throughput. **balanced:** Sends only packets that fit in the transmission buffer, which is 128 bytes. Normally 3-4 packets will fit, depending on user payload and overhead. **performance:** Maximizes throughput by loading new packets into transmission buffer and sending them as soon as the previous packets have been successfully transmitted. Increases power consumption. **Default:** **balanced**
Example : Optimizing data throughput *<throughput optimize="performance" />*	
Example : Optimizing power consumption *<throughput optimize="power" />*	

5.6 <passkey>

This configuration defines a fixed passkey to be used during MITM paring instead of a randomly generated passkey.

If this tag is not used, then the passkey for Man-in-the-Middle pairing will be randomly generated, as described in the *Bluetooth* specification.

Attribute	Value - Description
passkey	Defines a six (6) digit fixed passkey used during MITM pairing. **Range:** **000000 - 999999** **Default:** **disabled**
Example : Use fixed MITM passkey 246802 *<passkey value="246802" />*	

⚠ When this configuration is enabled, the device will default to a ***display only*** I/O capability setting. The remote device pairing with this device must have ***keyboard only*** or ***keyboard/display*** capabilities, or else J*ust Works* pairing is used automatically.

5.7 <user_data>

Defines how much continuous flash space will be allocated for user data. This space is taken from the pool that would otherwise been used for PS keys. Data size allocated will be rounded up to nearest 2KB.

When implementing the Over-the-Air (OTA) firmware update by storing the update image to the module's built-in flash, this space must be pre-allocated. The allocated size must be at least the size of the firmware update for the update to be possible. If you allocate flash for user data and want to also support OTA firmware update, make sure there is enough flash space reserved for the firmware update as well. When you compile the firmware with bgbuild.exe the compiler output will give an indication of the required flash allocation.

Attribute	Value - Description
size	Defines how much data is allocated for the user data. **Default:** **0**
file	Optionally initialize the data from a file. If both the ***file*** and ***size*** attributes are used than the allocated flash space will be the larger of the two rounded up to closest 2kB.
Example: Allocating 1280 bytes from the flash for user data *<user_data size="0x500" />*	

5.8 <dfu>

This configuration option can be used to disable DFU firmware update feature.

Attribute	Value - Description
enable	**Options:** **true:** Booting to DFU mode is allowed **false:** Booting to DFU mode is not allowed **Default:** **true**
Example: Disabling DFU firmware update *<dfu enable="false" />*	

⚠ If using this option with a BLED112 device or another end-product, which does not expose the HW debugging interfaces for re-flashing the BLE firmware, the firmware of the device is permanently locked.

5.9 Examples

Below is an example of **config.xml** that enables a single (1) connection, disables BGScript timeout and configures the throughput for balanced mode.

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <connections value="1" />
    <script_timeout value="0" />
    <throughput optimize="balanced" />
</config>
```

Contact information

Sales: sales@bluegiga.com

Technical support: http://www.bluegiga.com/support/

Orders: orders@bluegiga.com

WWW: http://www.bluegiga.com

Head Office / Finland:

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

Head 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.
Unit 10-18, 32/F, Tower 1, Millennium City 1,
388 Kwun Tong Road, Kwun Tong, Kowloon,
Hong Kong

USING BLED112 WITH RASPBERRY PI

QUICK START GUIDE

Tuesday, 15 April 2014

Version 1.1

bluegiga

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Minor changes

TABLE OF CONTENTS

1 Introduction

This guide contains the basic setup instructions needed to start using the BLED112 *Bluetooth* Smart USB dongle with the Raspberry Pi. **Note that some general Linux experience is assumed and will greatly help with development or troubleshooting in this process.**

Figure 1: BLED112 dongle

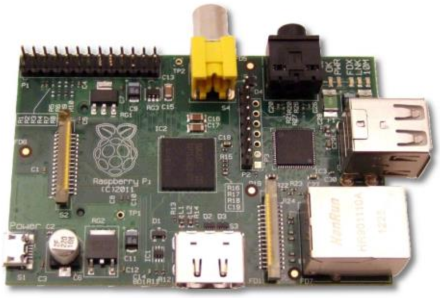


Figure 2: Raspberry Pi Model B

Since the BLED112 is a simple USB dongle, the only action necessary is to plug it into one of the Raspberry Pi's available USB ports. This may be done at any time, before or after the system has power or has finished booting.

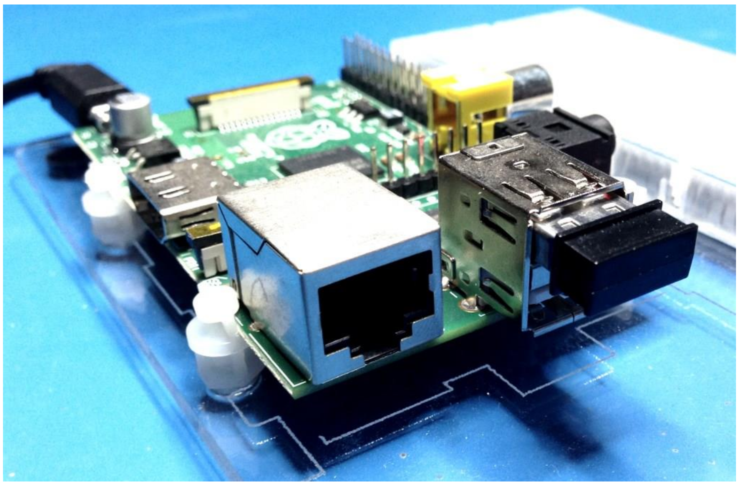


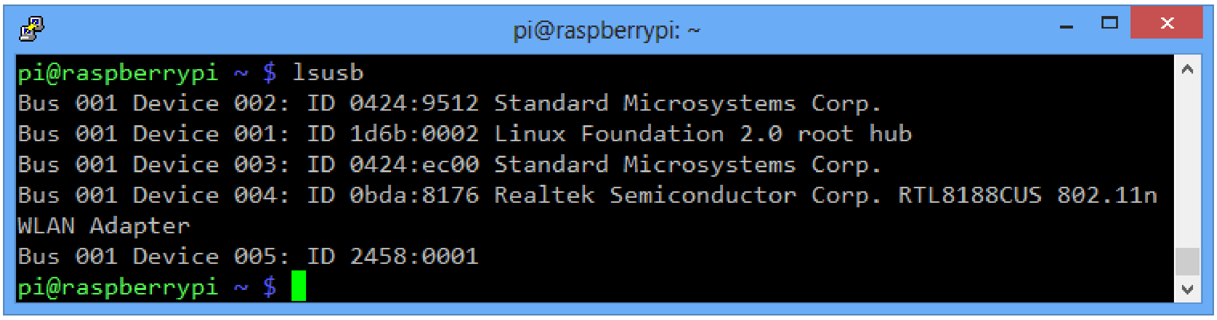
Figure 3: BLED112 in a Raspberry Pi USB port

2 Setting up Software

Use the current standard Raspbian “Wheezy” Linux build available here:

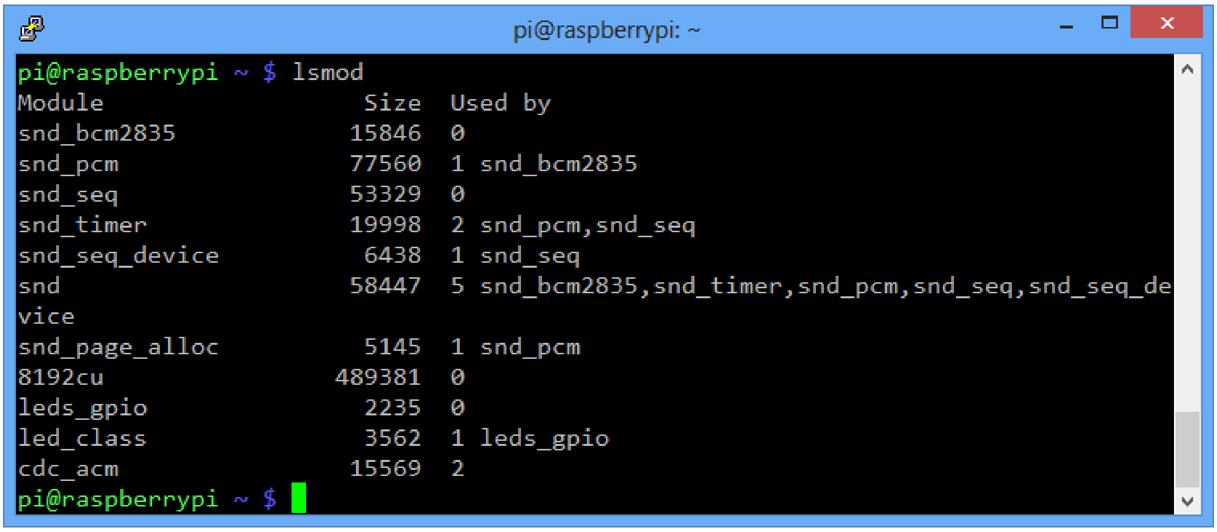
- http://www.raspberrypi.org/downloads

No drivers should be necessary, since the BLED112 is a full-stack device which enumerates as a USB CDC port. It does not require any Bluetooth stack such as “bluez” to be present on the host. However, you can still verify device connectivity and that the correct kernel module has been loaded by connecting to the Raspberry Pi via SSH (or a direct terminal session with a keyboard and display), and then checking the output of “**lsusb**”, “**lsmod**”, and “**ls /dev/ttyACM***” as shown here:



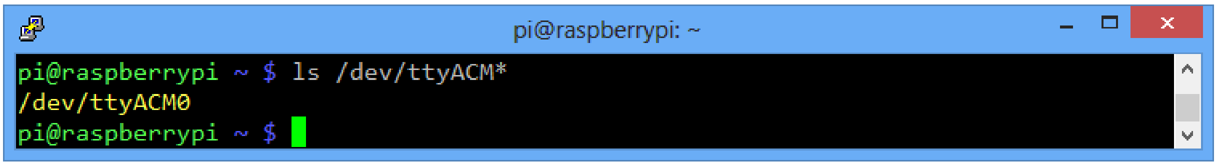
```
pi@raspberrypi ~ $ lsusb
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS 802.11n
WLAN Adapter
Bus 001 Device 005: ID 2458:0001
pi@raspberrypi ~ $
```

Figure 4: “`lsusb`” output (BLED112 appears as **`ID 2458:0001`**)



```
pi@raspberrypi ~ $ lsmod
Module                  Size  Used by
snd_bcm2835            15846  0
snd_pcm                77560  1 snd_bcm2835
snd_seq                53329  0
snd_timer              19998  2 snd_pcm,snd_seq
snd_seq_device          6438  1 snd_seq
snd                    58447  5 snd_bcm2835,snd_timer,snd_pcm,snd_seq,snd_seq_de
vice
snd_page_alloc          5145  1 snd_pcm
8192cu                489381  0
leds_gpio               2235  0
led_class               3562  1 leds_gpio
cdc_acm                15569  2
pi@raspberrypi ~ $
```

Figure 5: “`lsmod`” output (BLED112 uses **`cdc_acm`** module)



```
pi@raspberrypi ~ $ ls /dev/ttyACM*
/dev/ttyACM0
pi@raspberrypi ~ $
```

Figure 6: “`ls /dev/ttyACM*`” output (BLED112 appears as **`/dev/ttyACM0`**)

2.1 Testing *Bluetooth* Smart communications

Testing *Bluetooth* Smart communication:

This guide was written with the following starting configuration:

- Raspberry Pi model B, 256MB version
- Fresh 4GB SD card written with **2013-02-09-wheezy-raspbian.img**
- Bluegiga BLED112 running factory default "**usbcdc**" firmware
- Wired Ethernet connection to a local network connected to the internet
- Any Bluetooth Smart peripheral device nearby

Your configuration may be slightly different, but the instructions here should still apply. To prepare and run a basic Python-based BLE scanner, enter the following commands from a terminal or SSH session:

```
sudo apt-get install python-serial
wget https://raw.github.com/jrowberg/bglib/master/Python/Examples/bled112_scanner.py
chmod +x ./bled112_scanner.py
./bled112_scanner.py
```

This will install the **python-serial** package (which provides PySerial), then download the Python BGAPI-based BLE scanner script, make it executable, and finally run it using all of the default parameters. If there are any advertising BLE peripheral devices nearby, then you should begin seeing output like the following:

```
================================================================
BLED112 Scanner for Python v2013-05-10

================================================================

Serial port:    /dev/ttyACM0
Baud rate:      115200
Scan interval:  200 (125.00 ms)
Scan window:    200 (125.00 ms)
Scan type:      Passive
UUID filters:   None
MAC filter(s):  None
RSSI filter:    None
Display fields: - Time
                - RSSI
                - Packet type
                - Sender MAC
                - Address type
                - Bond status
                - Payload data
Friendly mode:  Disabled
----------------------------------------------------------------
Starting scan for BLE advertisements...
1368200810.904 -52 0 000780535BB4 0 255 020106020A0306FFFFFFB1B2B3
1368200812.410 -52 0 000780535BB4 0 255 020106020A0306FFFFFFB1B2B3
1368200813.915 -52 0 000780535BB4 0 255 020106020A0306FFFFFFB1B2B3
```

You can explore the various argument options for this scanner script by running it with the **-h** or **--help** argument.

Note that if you do not have a BLE peripheral device nearby but you do have an iPhone 4S+ or iPad 3+ or iPad Mini running iOS 6+, then you can use some freely available iOS apps to emulate a heart rate sensor or temperature sensor peripheral device.

3 Testing more complex communication with BGAPI

The **bled112_scanner.py** script implements only a limited subset of the full BGAPI communication protocol, but you can do a lot more with BGAPI than just scan for other devices. You can act as a BLE master (central/manager) device and connect to other peripherals and use them, or you could even act as a BLE peripheral device to allow a BLE master such as an iPhone or iPad to connect and control the Raspberry Pi.

To get started with something more complicated, you can download the Python **bglib** library, which is a port of the C wrapper we provide in our SDK. To download this code and a couple of example scripts which use it, enter the following commands in a terminal or SSH session:

```
wget https://raw.github.com/jrowberg/bglib/master/Python/bglib.py
wget https://raw.github.com/jrowberg/bglib/master/Python/Examples/bglib_test_scanner.py
wget https://raw.github.com/jrowberg/bglib/master/Python/Examples/bglib_test_htm_collector.py
wget https://raw.github.com/jrowberg/bglib/master/Python/Examples/bglib_test_hr_collector.py
chmod +x ./bglib_test_*
```

Note that you will still need to install the **python-serial** package as shown in the earlier scanner example if you do not already have it.

Each of the three example scripts above are built around the event-driven **bglib** Python library. The library itself is a single-file Python module which can be imported into any Python application. The examples above have the following functionality:

- **bglib_test_scanner.py:**
  Similar to the "bled112_scanner.py" example, this application uses the command/ response/event structure of BGAPI to scan for nearby BLE devices and display any resulting advertisement packets.

- **bglib_test_htm_collector.py:**
  This application scans for BLE devices which are advertising the official "Health Thermometer" service, then automatically connects when it finds one and configures the remote device to send temperature readings. These readings are displayed as they come in.

- **bglib_test_hr_collector.py:**
  This application scans for BLE devices which are advertising the official "Heart Rate" service, then automatically connects when it finds one and configures the remote device to send heart rate measurements. These are displayed as they come in.

4 Additional information

The latest data sheets, design references, and full API Reference Guide are available in Bluegiga's BLED112 product page:

- http://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/bled112-bluetooth-smart-dongle/documentation/

Always refer to the latest documentation when working with the BLED112 *Bluetooth* dongle.

5 Support

Technical support is available online: http://www.bluegiga.com/support

PRODUCT COMPARISON GUIDE

2014

bluegiga

ABOUT US

Founded in 2000 and headquartered in Espoo, Finland, Bluegiga is committed to providing innovative, easy-to-use, short-range wireless connectivity solutions to product designers throughout the world. Our competitive advantage comes from a unique combination of high performance radios, connectivity software, and superior customer support.

Our product line-up includes ultra-low power Bluetooth Smart modules, high performance Bluetooth Classic audio and data modules, and high-speed Wi-Fi modules. Bluegiga customers include industry leaders in consumer electronics, health and wellness, automotive aftermarket, sports and fitness, M2M and industrial telemetry.

OUR VALUE PROPOSITION

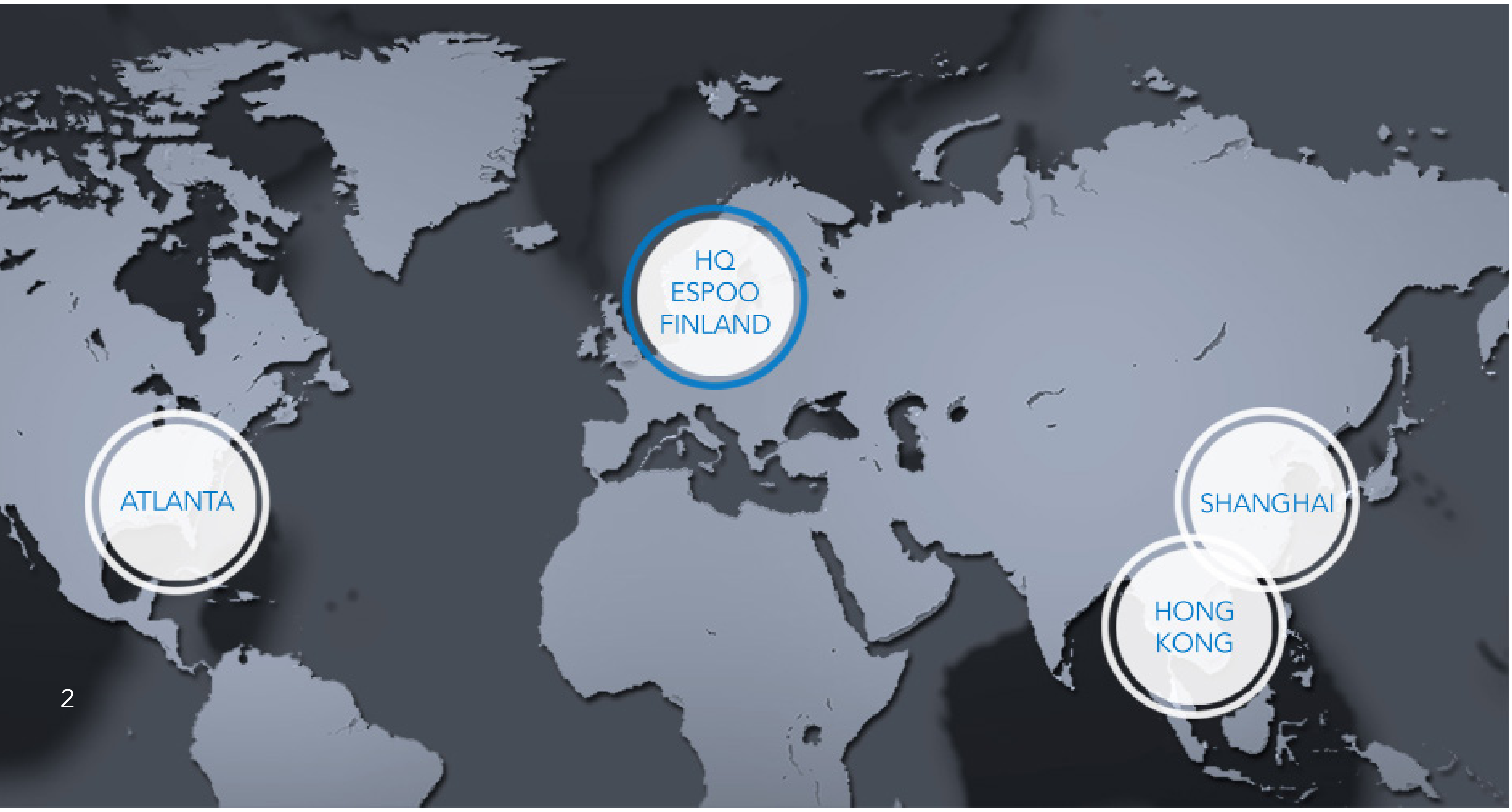
RF EXPERTISE AND PERFORMANCE – We design high quality and certified radio modules that deliver the reliability and performance required, even for the most demanding applications.

INTEGRATED CONNECTIVITY SOFTWARE – Bluegiga has developed our own Bluetooth classic, Smart and Wi-Fi software stacks giving us a unique advantage to support, maintain, and further develop or customize our software to match our customers' needs.

CUSTOMER SERVICE - We have a highly skilled and global Field Application Engineering team, who will review your design, help optimize your software, and instruct you in how to use our products.

QUALITY AND PRODUCT LIFE TIMES - The products we manufacture are designed to last for many, many years and meet the most rigorous operating specifications.

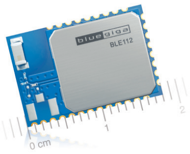
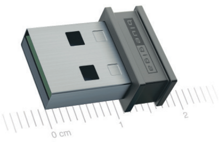
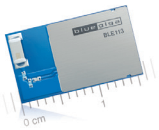
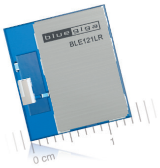
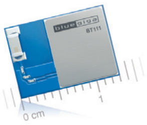
GLOBAL CHANNELS AND PARTNERS – Our products are available globally through a network of more than 50 partners ranging from worldwide distributors and online stores to locally focused distributors and representatives who sell and support our products in over 60 countries.



TECHNICAL SPECIFICATIONS	CLASSIC BLUETOOTH	BLUETOOTH SMART	BLUETOOTH SMART READY	WI-FI
Radio frequency	2.4GHz	2.4GHz	2.4GHz	2.4GHz or 5.8GHz
Distance/Range	~10-100 meters	~10-100 meters	~10-100 meters	~10-100 meters
Symbol rate	1-3Mbps	1Mbps	1-3Mbps	1.69Gbps
Application throughput	up to 2.1 Mbps	up to 250 kbps	up to 2.1 Mbps or 250kbps*	up to 100 Mbps
Nodes/Active slaves	7	Unlimited	7 / unlimited*	32
Security	56 to 128 bit	128-bit AES	128 bit AES	WPA2/AES
Modulation	FHSS	FHSS	FHSS	DSSS and OFDM
Latency (not from connected state to send data)	100+ ms	<6ms	<6ms or 100+ms"	150ms
Government regulation	Worldwide	Worldwide	Worldwide	Worldwide
Certification body	Bluetooth SIG	Bluetooth SIG	Bluetooth SIG	IEEE/WECA
Voice capable	Yes	No	Yes	Yes
Network topology	Point-to-point, star, scatternet	Point-to-point, star, scatternet	Point-to-point, star, scatternet*	Star, point-to-point
Power consumption	1 (reference value)	0.01 to 0.5 (use case dependent)	1 (reference value)	2 (use case dependent)
Service discover	Yes	Yes	Yes	N/A
Profile concept	Yes	Yes	Yes	No
Primary use cases	Mobile phones, headsets, stereo audio, automotive, PCs etc.	Mobile phones, tablets, gaming, PCs, sport & fitness, medical, automotive, industrial automation, home electronics etc.	Mobile phones, headsets, stereo audio, automotive, PCs etc.	Mobile phones, Tablets, PCs, Servers, medical, industrial automation, home electronics, etc.
Profiles	Serial Port, Hands-free, HID, A2DP, AVRCP, etc.	Proximity profile, battery status, weight scale, heart rate monitor, humidity, etc.	Classic Bluetooth profiles and/or Bluetooth smart profiles*	N/A

*Capabilities will vary based on operating mode between classic Bluetooth or Bluetooth Smart

Bluetooth 4.0 Module Comparison

	BLE112 *Bluetooth* Smart Module	BLED112 *Bluetooth* Smart Dongle	BLE113 *Bluetooth* Smart Module	BLE121LR *Bluetooth* Smart Long Range Module	BT111 *Bluetooth* Smart Ready HCI Module
					

BLUETOOTH

	BLE112	BLED112	BLE113	BLE121LR	BT111
Version	*Bluetooth* 4.0	*Bluetooth* 4.0	Bluetooth 4.0	Bluetooth 4.0	Bluetooth 4.0
Single mode	Yes	Yes	Yes	Yes	-
Dual mode	-	-	-	-	Yes
Roles	central/peripheral	central/peripheral	central/peripheral	central/peripheral	

RADIO

	BLE112	BLED112	BLE113	BLE121LR	BT111
TX power	+2-3 dBm	0 dBm	0 dBm	+8 dBm	+8 dBm
RX sensitivity	-92 dBm	-89 dBm	-93 dBm	-98 dBm	-89 dBm
Class	-	-	-	1	1
Typical Range*	150m	30m	100m	250-450m	100m

ANTENNA OPTIONS

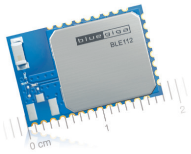
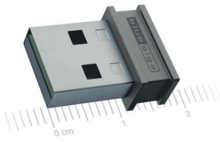
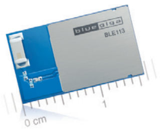
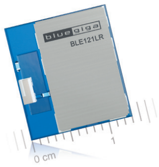
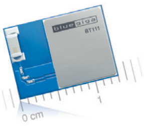
	BLE112	BLED112	BLE113	BLE121LR	BT111
Integrated	Yes	Yes	Yes	Yes	Yes
U.FL	Yes	-	-	-	-

INTERFACES

	BLE112	BLED112	BLE113	BLE121LR	BT111
UART	2 (or SPI)	-	2 (or SPI)	2 (or SPI)	-
USB	2.0 device	2.0 device	-	-	2.0 device
SPI	2 (or UART)	-	2 (or UART)	2 (or UART)	-
I2C	1 (soft I2C)	-	1	1	-
PWM	4 (or timer)	-	4 (or timer)	4 (or timer)	-
GPIO	19 configurable (shared)	-	19 configurable (shared)	16 configurable (shared)	6
ADC	7 x 12 bit	-	7 x 12-bit	7 x 12-bit	-
Wake-up interrupt	Yes	-	Yes	Yes	-
Comparator	1	-	1	1	-
Timers	2x8-bit and 1x16-bit (or PWM)	-	2x8-bit and 1x16-bit (or PWM)	2x8-bit and 1x16-bit (or PWM)	-
Op-amp	Yes	-	Yes	Yes	-
Battery monitor	Yes	-	Yes	Yes	-
Temperature sensor	Yes	Yes	Yes	Yes	-
Debug	Yes	-	Yes	Yes	Yes
IR generation	Yes	-	Yes	Yes	-
PCM	-	-	-	-	Yes
I2S	-	-	-	-	Yes

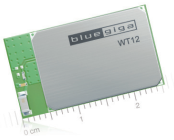
MICROCONTROLLER

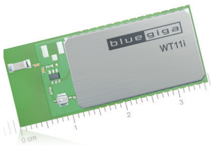
	BLE112	BLED112	BLE113	BLE121LR	BT111
Architecture	8051	8051	8051	8051	16-bit RISC (XAP2)
RAM	8 kB	8 kB	8 kB	8 kB	-
Flash	128 kB	128 kB	128 kB / 256 kB	256 kB	-
EEPROM	-	-	-		64kB

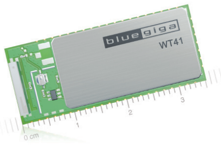
	BLE112 *Bluetooth* Smart Module 	BLED112 *Bluetooth* Smart Dongle 	BLE113 *Bluetooth* Smart Module 	BLE121LR *Bluetooth* Smart Long Range Module 	BT111 *Bluetooth* Smart Ready HCI Module 
CURRENT CONSUMPTION					
TX peak	27 mA	N/A	18.2 mA	27 mA	70 mA
RX peak	19.6 mA	N/A	17.9 mA	19.6 mA	52 mA
Sleep (timer active)	1 uA	N/A	1 uA	1 uA	-
Sleep (external wake-up)	0.5 uA	N/A	0.5 uA	0.5 uA	370uA
OPERATING VOLTAGE					
Operating voltage	2.0 - 3.6V	5V	2.0 - 3.6V	2.0 - 3.6V	2.3 - 5.7V
BLUETOOTH SOFTWARE STACK					
Integrated stack	Yes	Yes	Yes	Yes	No
ATT	Yes	Yes	Yes	Yes	-
GATT	Yes	Yes	Yes	Yes	-
GAP	Yes	Yes	Yes	Yes	-
L2CAP	Yes	Yes	Yes	Yes	-
Security manager	Yes	Yes	Yes	Yes	-
Host API	BGAPI™ protocol	BGAPI™ protocol	BGAPI™ protocol	BGAPI™ protocol	HCI
Connections	up to 8	up to 8	up to 8	up to 8	7 x *Bluetooth* classic 5 x *Bluetooth* LE
HCI	-	-	-	-	HCI over USB
IP licensing	Yes	Yes	Yes	Yes	-
SUPPORTED PROFILES					
GAP	Yes	Yes	Yes	Yes	Yes*
Manufacturer service	Yes	Yes	Yes	Yes	Yes*
Battery service	Yes	Yes	Yes	Yes	Yes*
Proximity profile	Yes	Yes	Yes	Yes	Yes*
HR profile	Yes	Yes	Yes	Yes	Yes*
Temperature profile	Yes	Yes	Yes	Yes	Yes*
SOFTWARE DEVELOPMENT					
On-board app support	Yes	Yes	Yes	Yes	-
BGScript™	Yes	Yes	Yes	Yes	-
Profile toolkit™	Yes	Yes	Yes	Yes	-
BGLib™ (Host C library)	Yes	Yes	Yes	Yes	Stack dependent
Software dev service	Yes	Yes	Yes	Yes	-
SDK/IDE	Bluegiga SDK	Bluegiga SDK	Bluegiga SDK	Bluegiga SDK	Stack dependent
C SDK	IAR Embedded Workbench	-	IAR Embedded Workbench	IAR Embedded Workbench	Stack dependent
CERTIFICATIONS					
Certifications	*Bluetooth*, CE, FCC, IC, South Korea and Japan	*Bluetooth*, CE, FCC, IC, South Korea, Japan and Brazil	*Bluetooth*, CE, FCC, IC, South Korea and Japan	*Bluetooth*, CE, FCC, IC, South Korea and Japan	*Bluetooth*, CE, FCC, IC, South Korea and Japan
DIMENSIONS					
Dimensions (W x L x H)	12 x 18 x 2.3 mm	17 x 12 x 6.5 mm	9.15 x 15.75 x 1.9 mm	14.7 x 13.0 x 1.8 mm	9.3 x 13.05 x 2.3 mm

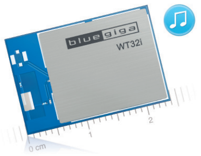
* Depends on the host *Bluetooth* stack

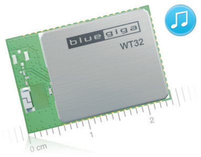
Bluetooth Classic Module Comparison







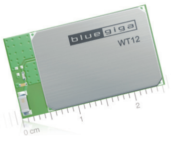
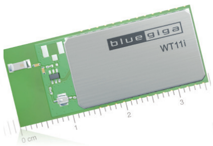
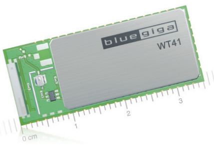
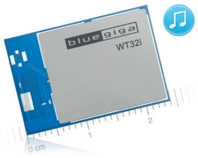
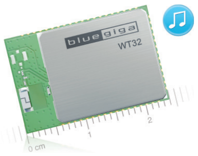




	WT12 Class 2 Module	WT11i Class 1 Module	WT41 Long Range Module	WT32i Class 2 Audio Module	WT32 Class 2 Audio Module
BLUETOOTH					
Version	Bluetooth 2.1 + EDR	Bluetooth 2.1 + EDR	Bluetooth 2.1 + EDR	Bluetooth 3.0	Bluetooth 2.1+EDR
Bluetooth low energy support	-	-	-	-	-
BR/EDR support	Yes	Yes	Yes	Yes	Yes
RADIO					
Typical TX power	+3 dBm	+17 dBm	+19 dBm	+6.5 dBm	0 dBm
Typical RX sensitivity	-83 dBm	-85 dBm	-93 dBm	-90 dBm	-90 dBm
Class	2	1	1	1.5	1.5
Typical range*	30-50m	200-400m	500-1000m	100-200m	30m
ANTENNA OPTIONS					
Integrated chip	Yes	Yes	Yes	Yes	Yes
U.FL	-	Yes	Yes	Yes	Yes (W.FL)
INTERFACES					
UART	1	1	1	1	1
USB	2.0 device	2.0 device	2.0 device	2.0 device	2.0 device
GPIO	6 configurable	6 configurable	6 configurable	10 configurable	10 configurable
AIO	-	1 x 8-bit	1 x 8-bit	2 x 10-bit	2 x 10-bit
Debug (SPI)	1	1	1	1	1
AUDIO INTERFACES					
PCM	1	1	1	1	1
I2S	-	-	-	1	1
SPDIF	-	-	-	1	1
Analogue	-	-	-	2 inputs/outputs	2 inputs/outputs
MICROCONTROLLER					
Architecture	16-bit RISC (XAP2)	16-bit RISC (XAP2)	16-bit RISC (XAP2)	16-bit RISC (XAP2)	16-bit RISC (XAP2)
RAM	48 kB	48 kB	48 kB	48 kB	48 kB
Flash	8 Mbit	8 Mbit	8 Mbit	16 Mbit	8 Mbit
DSP	-	-	-	Kalimba DSP	Kalimba DSP
OPERATING VOLTAGE					
Operating voltage	2.7 - 3.6V	2.7 - 3.6V	2.7 - 3.6V	1.8 - 4.4V	1.8 - 4.4V
PHYSICAL CONNECTION					
Type	solder pads	castellated edges	castellated edges	solder pads	castellated edges

* Line-of-sight unobstructed range measured between two identical modules

Bluetooth Classic Module Comparison

	WT12 Class 2 Module	WT11i Class 1 Module	WT41 Long Range Module	WT32i Class 2 Audio Module	WT32 Class 2 Audio Module
					

BLUETOOTH STACK FEATURES

	WT12	WT11i	WT41	WT32i	WT32
Version	2.1 + EDR / 3.0	2.1 + EDR / 3.0	2.1 + EDR / 3.0	3.0	2.1 + EDR / 3.0
Integrated Bluetooth stack	Yes	Yes	Yes	Yes	Yes
Security Simple Pairing	Yes	Yes	Yes	Yes	Yes
Connections	1-7	1-7	1-7	1-6	1-6
Host API	ASCII commands / HCI	ASCII commands / HCI	ASCII commands / HCI	ASCII commands	ASCII commands
HCI interface	UART/USB	UART/USB	UART/USB	-	-

SUPPORTED PROFILES

	WT12	WT11i	WT41	WT32i	WT32
SPP	Yes	Yes	Yes	Yes	Yes
OBEX OPP	Yes	Yes	Yes	Yes	Yes
OBEX FTP	Yes	Yes	Yes	Yes	Yes
DUN	Yes	Yes	Yes	Yes	Yes
HID	Yes	Yes	Yes	Yes	Yes
A2DP	-	-	-	Yes	Yes
AVRCP	Yes	Yes	Yes	v. 1.5	v. 1.3
HFP v.1.6	Yes	Yes	Yes	Yes	Yes
HSP	Yes	Yes	Yes	Yes	Yes
PBAP	Yes	Yes	Yes	Yes	Yes
HDP	Yes	Yes	Yes	Yes	-
MAP	Yes	Yes	Yes	Yes	Yes
DI	Yes	Yes	Yes	Yes	Yes
Apple iAP support	Yes	Yes	Yes	iAP1 and iAP2	iAP1
Over-the-Air configuration*	Yes	Yes	Yes	Yes	Yes
BGIO*	Yes	Yes	Yes	Yes	Yes

SOFTWARE DEVELOPMENT

	WT12	WT11i	WT41	WT32i	WT32
On-board applications	Yes	Yes	Yes	Yes	Yes
Software development service	Yes	Yes	Yes	Yes	Yes
SDK/IDE	CSR BlueLab	CSR BlueLab	CSR BlueLab	CSR BlueLab	CSR BlueLab

CERTIFICATIONS

	WT12	WT11i	WT41	WT32i	WT32
Bluetooth	Yes	Yes	Yes	Yes	Yes
CE	Yes	Yes	Yes	Yes	Yes
FCC	Yes	Yes	Yes	Yes	Yes
IC	Yes	Yes	Yes	Yes	Yes
South Korea	Yes	-	Yes	Yes	-
Japan	Yes	Yes	Yes	Yes	Yes
NCC (Taiwan)	-	-	WT41-E	-	-

DIMENSIONS

	WT12	WT11i	WT41	WT32i	WT32
Dimensions (W x L x H)	14 x 25.6 x 2.4 mm	14.5 x 35.8 x 2.6 mm	14.5 x 35.8 x 2.6 mm	15.9 x 23.9 x 2.4 mm	15.9 x 23.9 x 2.5 mm

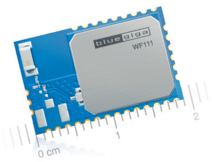
	iWRAP 5.0.2 build 764 GENERIC	iWRAP 5.0.2 build 765 AUDIO SOURCE	iWRAP 5.0.2 build 763 iAP	iWRAP 5.0.2 build 767 aptX®	iWRAP 5.0.2 build 768 aptX® SOURCE
Ordering code	AI5	AI5S	AI5IAP	AI5-APTX	AI5S-APTX
SUPPORTED HARDWARE					
WT32i	-	-	-	-	-
WT32	Yes	Yes	Yes	Yes	Yes
WT41	-	-	-	-	-
WT11i	-	-	-	-	-
WT12	-	-	-	-	-
BLUETOOTH STACK FEATURES					
Version	3.0	3.0	3.0	3.0	3.0
Secure Simple Pairing	Yes	Yes	Yes	Yes	Yes
Max connections	6	6	6	6	6
Host API	ASCII commands	ASCII commands	ASCII commands	ASCII commands	ASCII commands
Host interfaces	UART	UART	UART	UART	UART
Encryption	56/128-bits	56/128-bits	56/128-bits	56/128-bits	56/128-bits
SUPPORTED PROFILES					
SPP	Yes	Yes	Yes	Yes	Yes
OBEX OPP [1]	Yes	Yes	Yes	Yes	Yes
OBEX FTP [2]	Yes	Yes	Yes	Yes	Yes
DUN [3]	Yes	Yes	Yes	Yes	Yes
HID[4]	Yes	-	Yes	Yes	-
A2DP	A2DP sink	A2DP source	A2DP sink	A2DP sink	A2DP source
HFP	Hands-free	Hands-free AG	Hands-free	Hands-free	Hands-free AG
HSP	Headset	Headset AG	-	Headset	Headset AG
AVRCP	Yes	Yes	Yes	Yes	Yes
PBAP [5]	Yes	-	Yes	Yes	-
HDP	-	-	-	-	-
DI	Yes	Yes	Yes	Yes	Yes
MAP [7]	Yes	-	Yes	Yes	-
Apple iAP support [6]	-	-	iAP1	-	-
Over-the-Air configuration	Yes	-	-	-	-
BGIO	Yes	-	-	-	-
IEEE AGENTS (FOR HDP)					
Blood pressure monitor	-	-	-	-	-
Blood glucose	-	-	-	-	-
Weight scale	-	-	-	-	-
Pulse oximeter	-	-	-	-	-
Thermometer	-	-	-	-	-
AUDIO FEATURES					
CVC echo cancellation	-	-	-	-	-
aptX audio codec	-	-	-	Yes	Yes
AAC audio codec	-	-	-	-	-
mSBC (Wide-Band Speech)	Yes	Yes	Yes	Yes	Yes
BLUETOOTH QUALIFICATION					
QDID	37931	37931	37931	37931	37931

1) OPP server only
2) FTP client only
3) DUN terminal only
4) HID device only
5) PBAP client only
6) Delivered only to Apple MFi licensees. Please contact Bluegiga support for more information.
7) MAP client only

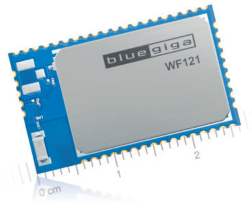
iWRAP 5.0.2 build 766 CVC5	iWRAP 5.5.0 build 891 GENERIC	iWRAP 5.5.0 build 893 iAP	iWRAP 6.0.0 build 891 GENERIC	iWRAP 6.0.0 build 892 aptX®	iWRAP 6.0.0 build 893 iAP
CVC5	AI55	AI55IAP	AI6	AI6-APTX	AI6IAP
-	-	-	Yes	Yes	Yes
Yes	-	-	-	-	-
-	Yes	Yes	-	-	-
-	Yes	Yes	-	-	-
-	Yes	Yes	-	-	-
3.0	3.0	3.0	3.0	3.0	3.0
Yes	Yes	Yes	Yes	Yes	Yes
6	7	7	6	6	6
ASCII commands	ASCII commands	ASCII commands	ASCII commands	ASCII commands	ASCII commands
UART	UART	UART	UART	UART	UART
56/128-bits	56/128-bits	56/128-bits	56/128-bits	56/128-bits	56/128-bits
Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes
-	Yes	Yes	Yes	Yes	Yes
A2DP sink	-	-	Yes	Yes	Yes
Hands-free	Yes	Hands-free	Yes	Yes	Yes
Headset	Yes	-	Yes	Yes	Yes
Yes	-	-	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes
-	Yes	-	Yes	-	-
Yes	Yes	Yes	Yes	Yes	Yes
Yes	Yes	Yes	Yes	Yes	Yes
-	-	iAP1 and iAP2	-	-	iAP1 and iAP2
-	Yes	Yes	Yes	Yes	Yes
-	Yes	Yes	Yes	Yes	Yes
-	Yes	-	Yes	-	-
-	Yes	-	Yes	-	-
-	Yes	-	Yes	-	-
-	Yes	-	Yes	-	-
-	Yes	-	Yes	-	-
Yes	-	-	-	-	-
-	-	-	-	Yes	-
-	-	-	Yes	Yes	Yes
-	-	-	Yes	Yes	Yes
37931	37931	37931	56758	56758	56758

Wi-Fi Module Comparison

WF111
Wi-Fi Module



WF121
Wi-Fi Module



WI-FI FEATURES

	WF111	WF121
Version	802.11 b/g/n	802.11 b/g/n
Frequency	2.4GHz	2.4GHz
Max. symbol rate	72.2Mbps	72.2Mbps
Soft AP mode	Yes (8 clients)	Yes (5 clients)

RADIO PERFORMANCE

Typical TX power	+17 dBm	+17 dBm
Typical RX sensitivity	-97 dBm	-97 dBm
Typical range *	300-500m	300-500m

ANTENNA OPTIONS

Integrated chip	Yes	Yes
U.FL	Yes	Yes

HOST INTERFACES

SDIO	Yes	No
CSPI	Yes	No
UART	No	Yes
USB	No	Yes
SPI	No	Yes

PERIPHERAL INTERFACES

SPI	-	up to 2
UART	-	up to 4
USB	-	1
Ethernet RMII	-	1
I2C	-	up to 2
GPIO	6	up to 38
AIO	-	up to 10

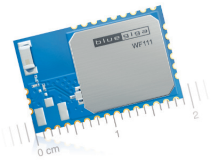
MICROCONTROLLER

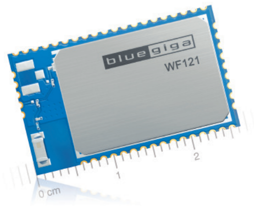
Architecture	-	MIPS 4K
MHz	-	80 Mhz
RAM	-	128 kB (<64 kB free)
Flash	-	512 kB (<256 kB free)

AVG. CURRENT CONSUMPTION

TX (17dBm, 802.11g)	192 mA	142 mA
RX	88 mA	127 mA
Idle, Associated to an AP	1.7mA	6.1mA
Sleep	70 uA	62 uA

* Line-of-sight unobstructed range measured between two identical modules





	WF111 Wi-Fi Module	WF121 Wi-Fi Module
OPERATING VOLTAGE		
Operating voltage	1.8V and 3.3V	2.7V - 3.6V
PHYSICAL CONNECTION		
Type	castellated edges	castellated edges
TCP/IP STACK FEATURES		
Integrated TCP/IP stack	-	Yes
DHCP	-	Yes
DNS	-	Yes
TCP client	-	Yes
UDP client	-	Yes
TCP server	-	Yes
UDP server	-	Yes
ICMP server	-	Yes
HTTP server	-	Yes
DHCP server	-	Yes
DNS server	-	Yes
HOST API		
BGAPI™ binary protocol	-	Yes
BGLib™ host library	-	Yes
OS DRIVERS		
Linux	Yes	Not needed
Windows	No	Not needed
Android	Yes	Not needed
SOFTWARE DEVELOPMENT		
On-board applications	-	Yes
BGScript™ support	-	Yes
Native C development	-	No
Software dev service	-	Yes
SDK	-	Bluegiga SDK
CERTIFICATIONS		
CE	Yes	Yes
FCC	Yes	Yes
IC	Yes	Yes
South Korea	Yes	Yes
Japan	Yes	Yes
DIMENSIONS		
Dimensions (W x L x H)	12 x 19 x 2.1 mm	15.4 x 26.2 x 2.1 mm

Head office
Bluegiga Technologies Oy
Sinikalliontie 5A
FI-02630 Espoo
Finland
Phone: +358 9 435 50 60
Fax: +358 9 435 50 66 0

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

Hong Kong office
Bluegiga Technologies Ltd.
Elite Business Center
15/F, Millenium City 3
370 Kwun Tong Road
Kwun Tong
Kowloon
Hong Kong

Shanghai office
Phone: +852 31078911
Shanghai Office
Room 401, #2 Building
690 Bi Bo Road,
Shanghai, 201203, China

Phone: +86 21 6104 2277

www.bluegiga.com

Bluegiga Technologies takes no responsibility for any mistakes that might appear in this document. Information subject to change. Bluegiga reserves the right to change devices, software or specifications detailed here at any time without notice, and does not make any commitment to update the information contained here.

Bluegiga products are not authorized for any use as critical components in life support devices or systems. The Bluetooth trademark and logo are registered trademarks and are owned by the Bluetooth SIG, Inc. Wi-Fi is a Registered Trademark of the Wi-Fi alliance. The Bluetooth SIG, Inc., U.S.A.

Information subject to change.

08/2014