BLUETOOTH SMART PROFILE TOOLKIT

DEVELOPER GUIDE

Monday, 11 March 2013

Version 2.9

TABLE OF CONTENTS

1 Version history

Version	Comments
2.2	v.1.1 beta 2 updates added
2.3	Updated firmware compile and installation instructrions
2.4	Improved hardware.xml and gatt.xml examples and documentation
2.5	UART packet mode documentation updated
2.6	Updated compilation and installation instructions
2.7	Config.xml documentation improved and example added
2.8	<port> description improved
2.9	Updated with the latest SW 1.1 release (build 71). BLE113 (project.xml) is also added.

2 Introduction

The *Bluetooth* Smart profile toolkit guide developer guide instructs you how to make your own GATT based *Bluetooth* services and profiles and how to configure the settings of your Bluegiga *Bluetooth* Smart device.

The guide also contains basis instructions how to make projects with the *Bluetooth* Smart development environment, how to compile and install them into your Bluegiga *Bluetooth* Smart device.

3 Project file (project.xml)

The ***project.xml*** is the file that contains all the components like for example hardware configuration, BGScript file and GATT database file included in a *Bluetooth* Smart project.

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

⚠ If the project file is named as ***project.bgproj*** the Bluegiga BLEUpdate tool will automatically recognize it and compile the project and try to install it using a CC debugger.

3.1 <device>

Device configuration

XML tag	Description
type	Device type, ble112 or ble113 Default: ble112 Example: <device type="ble113" />

3.2 <gatt>

GATT database file

XML tag	Description
gatt	This tag tag is used to describe the XML file, which contains the GATT data base description **Example:** *<gatt in="gatt.xml" />*

3.3 <hardware>

Hardware configuration file

XML tag	Description
hardware	This tag is used to describe the XML file, which contains the hardware configuration of your Bluegiga *Bluetooth* Smart device. **Example:** *<hardware in="hardware.xml" />*

3.4 <config>

Application configuration file

XML tag	Description
config	This tag is used to describe the XML file, which contains generic application configuration of your Bluegiga _Bluetooth _Smart device. **Example:** *<config in="config.xml" />*

3.5 <script>

BGScript file (optional)

XML tag	Description
script	This tag is used to describe the BGScript file, which contains the BGscript code of your standalone *Bluetooth* Smart application. If you use BGAPI protocol and a separate host and do not use BGScript code, this tag should be left out. **Example:** *<script in="bgscript.bgs" />*

3.6 <usb_main>

USB descriptor definition (optional)

XML tag	Description
usb_main	This tag is used to describe the XML file, which 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. **Example:** *<usb_main in="cdc.xml" />*

3.7 <image>

Firmware binary output file

XML tag	Description
image	This tag is used to describe the 128kB firmware output file for the compiler. **Example:** *<image out="out.hex" />*

3.8 Example

Below is an example of the hardware configuration file for the "USBCDC" project.

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

Figure 1: Project file example

4 Hardware configuration file (hardware.xml)

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

4.1 <sleeposc>

Sleep oscillator settings:

Attribute	Value - Description
enable	**="true"** - The external 32.768KHz sleep oscillator is enabled. Sleep oscillator allows the BLE112 to enter power mode 1 or 2 between *Bluetooth* operations, for example between connection intervals. **="false"** - The 32.768KHz sleep oscillator is not enabled and the internal 32.000KHz RC oscillator is used for timings. Using this options increases the current consumption. **(Default)** **In BLE112 this SHOULD be enabled and in BLED112 this MUST not be used.**
ppm	**="30"** - Defines the sleep oscillator accuracy. **Do not modify, but always use value 30 with BLE112.** **Example for BLE112 *Bluetooth* Smart Module:** *<sleeposc enable="true" ppm="30" />* **Example for BLED112 USB dongle:** *<sleeposc enable="false" ppm="30" />*

4.2 <script>

BGScript settintgs

Attribute	Value - Description
enable	**="true"** - BGScripting is enabled **="false"**- BGScripting is disabled

4.3 <slow_clock>

Slow system clock when radio is active, in order to lower peak-power consumption:

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

UART and PWM uses system clock for timings. If this feature is enabled timings in peripherals are invalid. This feature must only be enabled when peripherals requiring stable clock is not used.

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

4.4 <lock_debug>

Lock debug interface in generated .HEX firmware file. If this feature is enabled only a full erase of the firmware can be done with CC debugger.

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

4.5 <sleep>

Enable/disable sleep modes:

Attribute	Value - Description
enable	**="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. (**Default**) =**"false"** - Use this to prevent the firmware from entering any of the sleep modes. **Example:** *<sleep enable="true" />*

4.6 <wakeup_pin>

This PIN is used to prevent CPU from entering any sleep modes, or it is used to get out of a sleep mode. Use it for example to prevent sleeping or to wake up from sleeping before sending commands to device through UART. This is needed when the module is expected to go to power mode 3 otherwise the module will remain sleeping forever.

attribute	description
enable	Use to enable wake-up pin. Wake-up pin wakes the device up from a sleep mode or prevents the device to go into a sleep mode.
port	Defines the port where wake-up pin is
pin	Defines the pin inside the selected port
state	Logic state for wakeup use state=up or state=down (state=up by default if this parameter is not given) **Example:** *<wakeup_pin enable="true" port="0" pin="0" />*

⚠ When this pin is pulled, CPU does not enter any sleep modes which increases power consumption.

4.7 <txpower>

Transmit power settings:

Attribute	Value - Description
power	**="0-15"** - 15 is the highest TX power setting and equals roughly to +3dBm with BLE112. 0 is the lowest value and corresponds to -24 dBm with BLE112. **Range: 0-15**
bias	**="0-15"** - Sets the TX power amplifier bias. Do not modify. **Range: 0-15 / Always use the default value of 5.** **Example (3 dBm TX power):** *<txpower power="15" bias="5" />* **Example (0 dBm TX power):** *<txpower power="13" bias="5" />* **Example (-24 dBm TX power):** *<txpower power="0" bias="5" />*

4.8 <pmux>

External DC/DC converter settings. If an external DC/DC converter (like TPS62730) is used to reduce the peak current consumption an IO pin needs to be dedicated to control the DC/DC converter. The firmware_ _automatically enabled and disables the DC/DC converter when it's needed, so the application does not need to handle it.

attribute	description
regulator_pin	Defines the output pin for the external DC/DC converter in port 1. **Range: 0-7**
clock_pin	Output pin of the 32.768 kHz clock. Can be used to provide the clock value to external devices. **Range: 0-7** **Example (DKBLE112):** *<pmux regulator_pin="7" />*

4.9 <port>

I/O port configuration settings (input only)

attribute	description
index	Port index to configure
tristatemask	tristate configuration (bit mask) for port. For the pins defined in this bitmask, there will no high/low pull used, but the pin will be in tristate mode. For example 0x02 means pin number 1 is configured to be tristated instead of being pulled high/low.
pull	Defines "up"/"down" pull direction. The pull direction can only be configured for the whole port, not individual pins. **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.

4.10 <usb>

USB interface settings:

Attribute	Value - Description
enable	**="true"** - Use this to enable the USB interface. (Default) =**"false"** - Use this to disable the USB interface.
endpoint	**="none | api | test | script | usb | uart0 | uart1"** - Defines where the USB interface is connected in the firmware. 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 BLED112 the interface must always be enabled or the dongle becomes unusable.
In BLE112 this should be set to false, since USB constantly uses 5+ mA of current, unless USB interface is really needed.

4.11 <usart>

This setting is used to configure the USART interface of the BLE112 module.

In UART mode, the number of data bits is 8 and parity is set to None. Number of data bits and parity cannot be re-configured.

attribute	description
channel	USART channel
baud	USART baudrate and SPI master clock **="1200"** - Min **...** **="115200"** - One of the standard baudrates **...** **="2000000"** - Max
alternate	alternate configuration for USART
endpoint	Defines where USART is connected in the firmware. See: endpoints
mode	**="uart"** - Use as UART. UART flow control MUST be used. (**Default**) **="packet"** - Use as UART in packet mode. This options allows UART to be used without UART flow control, but a special header needs to be used with BGAPI protocol See BGAPI description from the API reference manual for more information. **="spi_master"** - Use as SPI in master mode **="spi_slave"** - Use as SPI in slave mode
polarity	**="positive"** - Configures the SPI clock polarity to be positive **="negative"** - Configures the SPI clock polarity to be negative. (**Default**)
phase	SPI clock phase 0 or 1 **default=1**
endianness	SPI bit ordering MSB or LSB
flow	UART flow control setting: "true" or "false **default=true**
stop	UART stop bit logic high or low **default=high**
start	UART start bit logic high or low, **default=low** **Must be different than stop bit**
stopbits	UART stop bits 1 or 2 **default=1** **Example (Enabling BGAPI over UART):** *<usart channel="1" alternate="1" baud="115200" endpoint="api" />* **Example (Enabling UART access for BGScript):** *<usart channel="1" alternate="1" baud="115200" endpoint="none" />* **Example (Enabling SPI master interface on DKBLE112 to control the display):** *<usart channel="0" mode="spi_master" alternate="2" polarity="positive" phase="1" endianness="msb" baud="57600" endpoint="none" />*

4.12 <timer_ticks>

This configuration controls a global prescaler for Timer 1, Timer 3, and Timer 4. The pre-scaler value can be set to a value from 0.25 MHz to 32 MHz.

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

attribute	description
speed	Timer tick settings **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" />*

4.13 <timer>

This configuration is used to configure the TIMER of the BLE112 module.

attribute	description
index	Timer index to configure **1**: TIMER1 **3**: TIMER3 **4**: TIMER4
enabled_channels	Enabled channels for TIMER as bitmask
divisor	Divisor for timer **TIMER1:** **0**: Tick frequency/1 **1**: Tick frequency/8 **2**: Tick frequency/32 **3**: Tick frequency/128 **TIMER 3&4:** **0**: Tick frequency/1 **1**: Tick frequency/2 **2**: Tick frequency/4 **3**: Tick frequency/8 **4**: Tick frequency16 **5**: Tick frequency /32 **6**: Tick frequency/64 **7**: Tick frequency/128
mode	Timer operating mode **TIMER1:** **0** : Suspended **1** : Free running **2** : Modulo **3** : Up/Down **TIMER 3&4:** **0** : Free running, **1** : Down **2** : Modulo **3** : Up/Down
alternate	Alternate configuration for TIMER **Example (4-channel PWM):** *<timer index="1" enabled_channels="0x1f" divisor="0" mode="2" alternate="2"/>*

4.14 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 script
api	Endpoint is connected to BGAPI protocol
test	Connected to UART *Bluetooth* testing.
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

4.15 Hardware configuration examples

Below is an example of hardware configuration file used with BLED112 USB dongle, which uses BGAPI protocol over USB.

```
<?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 module, which uses BGAPI protocol over UART on DKBLE112:

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

```
<?xml version="1.0" encoding="UTF-8" ?>
   <hardware>
       <txpower power="15" bias="5" />
       <usb enable="false" endpoint="none" />
       <usart channel="1" alternate="1" baud="115200" endpoint="api" />
       <sleep enable="true" />
       <sleeposc enable="true" ppm="30" />
       <pmux regulator_pin="7" />
       <wakeup_pin enable="true" port="0" pin="0" />
       <port index="0" tristatemask="0" pull="down" />
   </hardware>
```

5 Application configuration file (config.xml)

This configuration file is used to configure the application features such as the number of maximum connections.

5.1 <connections>

This configuration defines the maximum connections are supported by the firmware.

If this tag does not exist then maximum connections is limited to one (1).

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

⚠ When more then one (1) connection is supported then connection interval range (minimum and maximum values) passed to create connection command must contain value that is divisible by ***connections*** * ***2.5ms***.

Example:

If three (3) connections are supported then interval range has to contain value that is divisible by ***3 * 2.5ms*** = ***7.5ms***. Also any multiple value of 7.5ms can be used like 7.5ms, 15ms, 22.5ms, 30ms etc.

Alternatively if two (2) connections are supported, the interval must be divisible by 5ms. Notice that in in this case lowest possible interval of 7.5ms cannot be used because it is not divisible by 5.0ms.

If only one connection is supported then any connection interval can be used for create connection command.

5.2 <manual_confirm>

If this tag exists in XML file then manual confirmation of attribute indications is enabled.

When a *Bluetooth* Smart device receives indications from a remote device it produces an ***attribute value*** event to the host, where type is ***attclient_attribute_value_type_indicate_rsp_req***. The host (application) must respond to this event with ***attclient_indicate_confirm*** command after it had handled the indication.

This feature can be used by the host software to acknowledge the indication data and this provides extra reliability. If this tag is not enabled the firmware will automatically acknowledge indications upon reception.

Attribute	Value - Description
	Enables or disables manual indication confirmations **Example (Enable manual confirmations):** *<manual_confirm />*

5.3 <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 the 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.

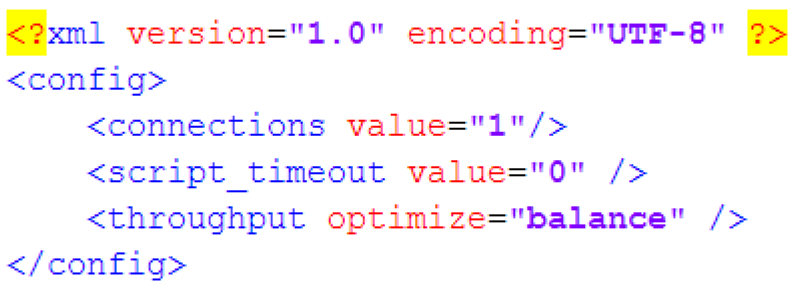
5.4 <throughput>

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

Attribute	Value - Description
optimize	**= power** - Only single packet is sent at each connection interval, minimizes power usage **= balanced** - Minimizes peak power consumption, sends only packets that fit in transmission buffer which is 128B in size (normally 3-4 packets would fit, depending on user payload and overhead) **= performance** - Maximize throughput, loads new packets to transmission buffer, and send them, as soon as previous packets have been successfully transmitted **Default: balanced** **Example (optimizing data throughput):** *<throughout optimize="performance" />* **Example (optimizing power consumption):** *<throughout optimize="power" />*

5.5 Example

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="balance" />
</config>
```

Figure 2: Example of config.xml

6 GATT database file (gatt.xml)

GATT database defines the *Bluetooth* low energy services and profiles implemented by the device. Typically a *Bluetooth* Smart device implements one or several profile and a profile consists of one to several services. Services on the other hand expose values, called characteristics.

6.1 Service description

6.1.1 <service>

The service tag starts a service definition and includes information like service UUID, ID and service type.

Attribute	Description
uuid	Universally Unique IDentifier. The UUID uniquely identifies a service. 16-bit values are used for the services defined by the Bluetooth SIG and 128-bit UUIDs can be used for manufacturer specific implementations.
id	The ID is used to identify a service within the service database and can be used as a reference from other services (include statement). This ID is not stored in the GATT database.
type	The type field defines whether the service is a **primary** or a **secondary** service. **default = primary**
advertise	If set **true**, GAP will add this service UUID to advertisement packet

6.1.2 <description>

The description tag is used only for informative purposes and not exposed by the GATT database.

6.1.3 <include>

Service included by this service

Attribute	Description
id	The include tag is used to include a service by another service. The ID refers to the service ID.

6.2 Characteristic description

6.2.1 <characteristic>

The characteristic tag defines a characteristic, it's UUID and internal ID used by BGScript.

Attribute	Description
uuid	Universally Unique IDentifier.The UUID uniquely identifies a characteristic. 16-bit values are used for the characteristics defined by the Bluetooth SIG and 128-bit UUIDs can be used for manufacturer specific characteristics.
id	The ID is used to identify a characteristic. The ID is used within a BGScript to read and write characteristic values. When the project is compiled with the **BGBuild** compiler a text file called **attributes.txt** is generated. This files contains the **id**s and corresponding handle values.

6.2.2 <properties>

The properties tag defines the characteristic properties. A characteristic may have a single or several properties.

Attribute	Description
read	Characteristic value can be read over a *Bluetooth* connection.
const	Characteristic value is stored in flash memory and it cannot be modified after programming.
write	Characteristic value can be written over a _Bluetooth _connection.
write_no_response	Characteristic value can be written only using a ***write_no_response*** command (for example by using BGLib's ***attclient_write_command***). This means the write operation is not confirmed over a *Bluetooth* connection.
notify	Characteristic value can be notified. Notification is not confirmed.
indicate	Characteristic value can be indicated. Indication is confirmed.
authenticated_read	Reading the characteristic value over a *Bluetooth* connection requires authentication. ***read***-attribute must also be set to true.
authenticated_write	Writing characteristic value over a *Bluetooth _connection requires authentication.* **write** *or* **write_no_response{_}** attribute must also be set to true.

6.2.3 <value>

Characteristic value description

Attribute	Description
length	Maximum length for attribute. Length is fixed. **Range: 0 - 255 (Bytes)** Example: ***<value length="20" />***
variable_length	Attribute is variable in length. Maximum length needs also to be defined. Example: ***<value variable_length="true" length="20" />***
type	How to interpret element value. Hex values: **hex** String values: **utf-8** Example: **<value type="hex" />** If value is **user** characteristic value is not stored in RAM. Application has to handle storing attribute value and handle length checking.

6.2.4 <description>

Characteristic User Description. A user friendly description for the characteristic. This is exposed by the GATT database to remote devices.

6.2.5 <descriptor>

Generic Characteristic descriptor definition.
Descriptor properties are defined by properties tag, only read and/or write access is allowed.
Value is defined by value tag same as in characteristic value.

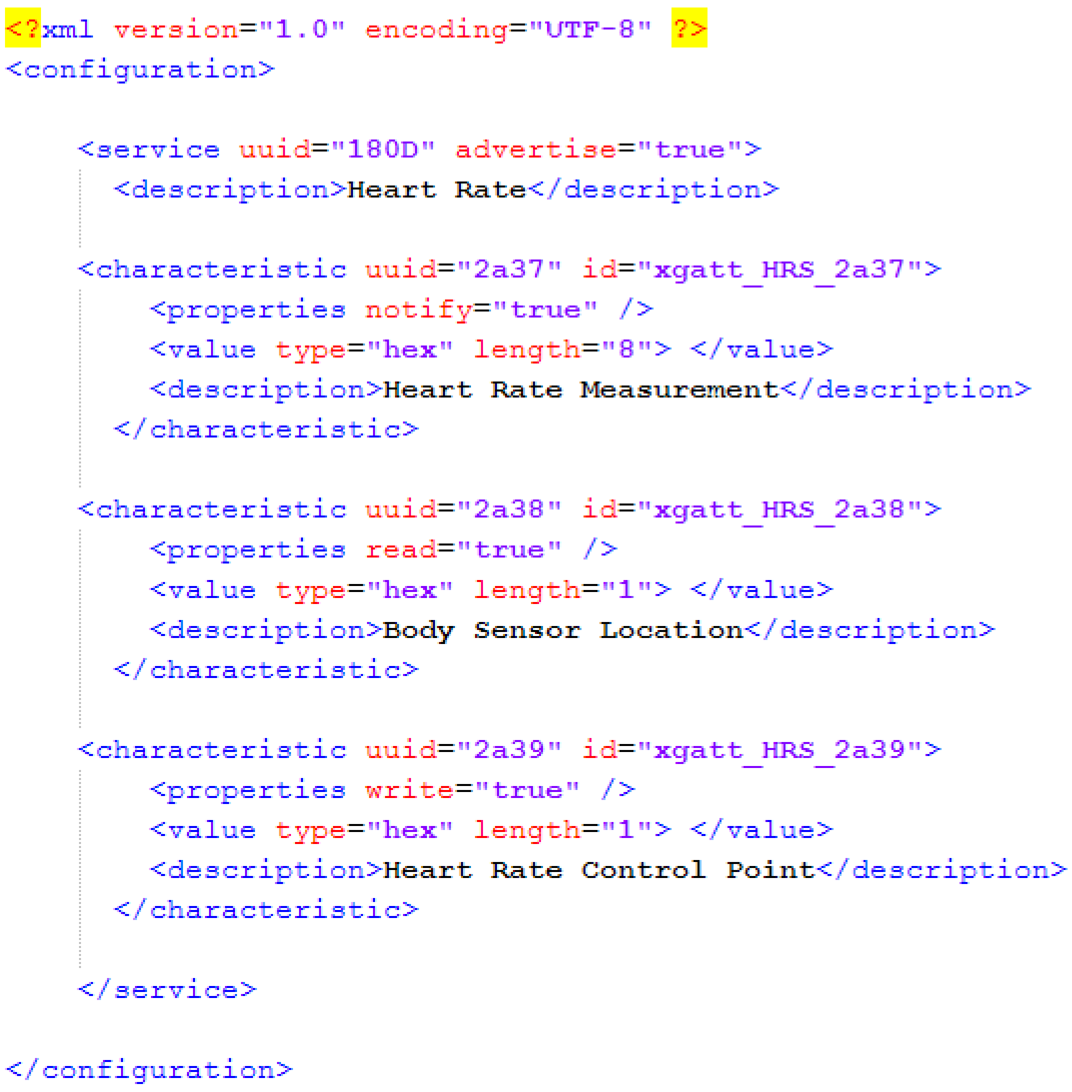
example:

```
<characteristic uuid="2a00" >
    <properties read="true" const="true" />
    <value>bluegiga-1</value>
    <descriptor uuid=9b7993c0-182c-11e2-892e-0800200c9a66"  id="mydescriptor">
        <properties read="true" const="true" />
        <value>Makkara</value>
    </descriptor>
</characteristic>
```

Attribute	Description
uuid	Universally Unique IDentifier.The UUID uniquely identifies a characteristic descriptor. 16-bit values are used for the characteristics defined by the Bluetooth SIG and 128-bit UUIDs can be used for manufacturer specific characteristics.
id	The ID is used to identify a characteristic descriptor. The ID is used within a BGScript to read and write characteristic descriptor values. When the project is compiled with the **BGBuild** compiler a text file called **attributes.txt** is generated. This files contains the **id**s and corresponding handle values.

6.3 Examples

The example below describes the Heart Rate service (HRS) version 1.0 using the GATT database XML schema.



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

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

    <characteristic uuid="2a37" id="xgatt_HRS_2a37">
        <properties notify="true" />
        <value type="hex" length="8"> </value>
        <description>Heart Rate Measurement</description>
      </characteristic>

    <characteristic uuid="2a38" id="xgatt_HRS_2a38">
        <properties read="true" />
        <value type="hex" length="1"> </value>
        <description>Body Sensor Location</description>
      </characteristic>

    <characteristic uuid="2a39" id="xgatt_HRS_2a39">
        <properties write="true" />
        <value type="hex" length="1"> </value>
        <description>Heart Rate Control Point</description>
      </characteristic>

    </service>

</configuration>
```

Figure 3: Heart Rate service v.1.0

Instead of length tag the attribute length can also be defined by typing characters into the value field.

The length of *Body Sensor Location* is one (1) byte.

The values defined in GATT database are not stored on the flash, unless they are marked **const**.

If **advertise**="true" is not set for example Apple iPhone4S may not be able to discover the device.

7 Compiling and installing firmware

Once you have completed the the project and modified the necessary files you need to compile the firmware binary for you device.

Using BLE Update tool

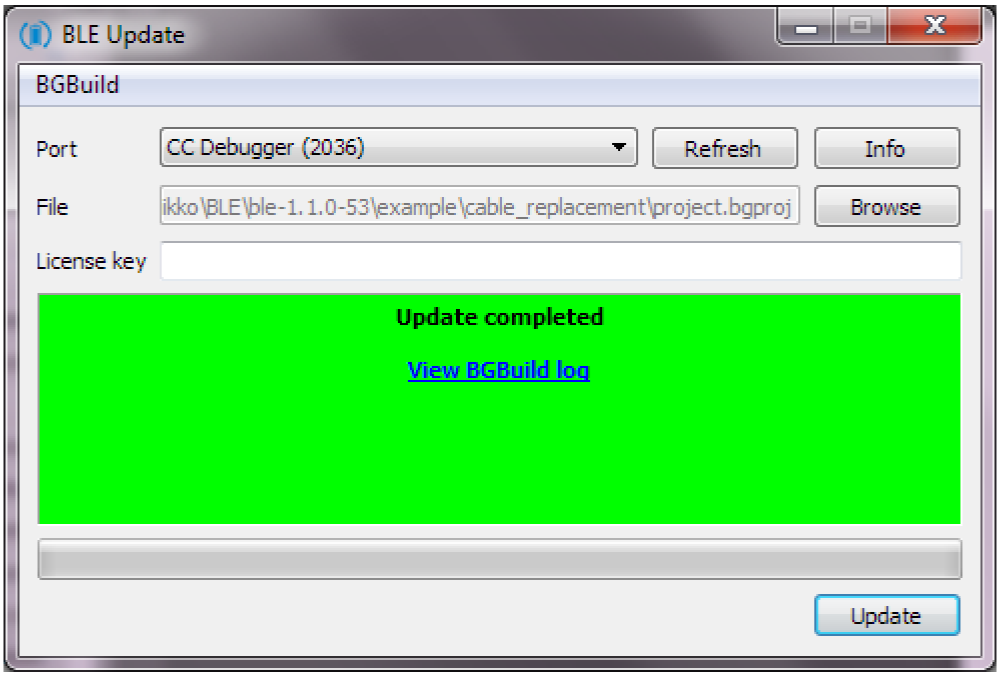
When you want to test your project, you need to compile the hardware settings, the GATT data base and BGScript code into a firmware binary file. The easiest way to do this is with the BLE Update tool that can be used to compile the project and install the firmware to a BLE112 module using a CC debugger.

In order to compile and install the project:

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

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

.



Port: CC Debugger (2036)
File: ikko\BLE\ble-1.1.0-53\example\cable_replacement\project.bgproj
License key:
Update completed

Figure 4: Compile and install with BLE Update tool

Compiling using the bgbuild.exe

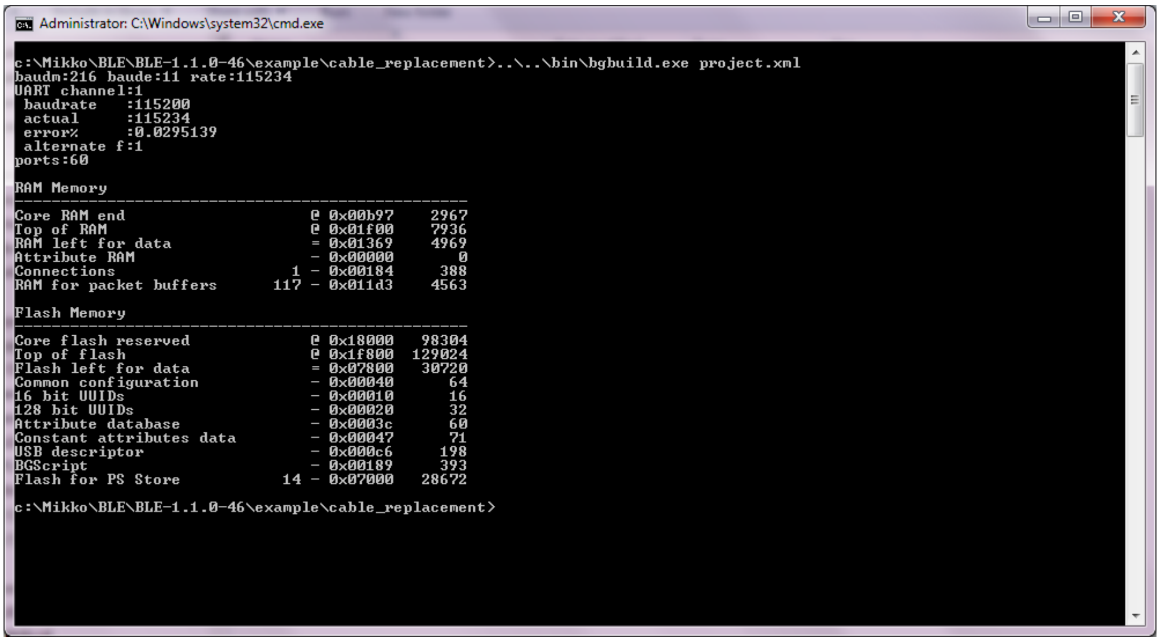
The project can also be compiled with the **bgbuild.exe** command line compiler. The BGBuild compiler simply generates the firmware image file, which can be installed to the BLE112.

In order to compile the project using BGBuild:

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

Syntax: ***bgbuild.exe <project file>***

.



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

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

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

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

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

Figure 5: Compiling with BGBuild.exe

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

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

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

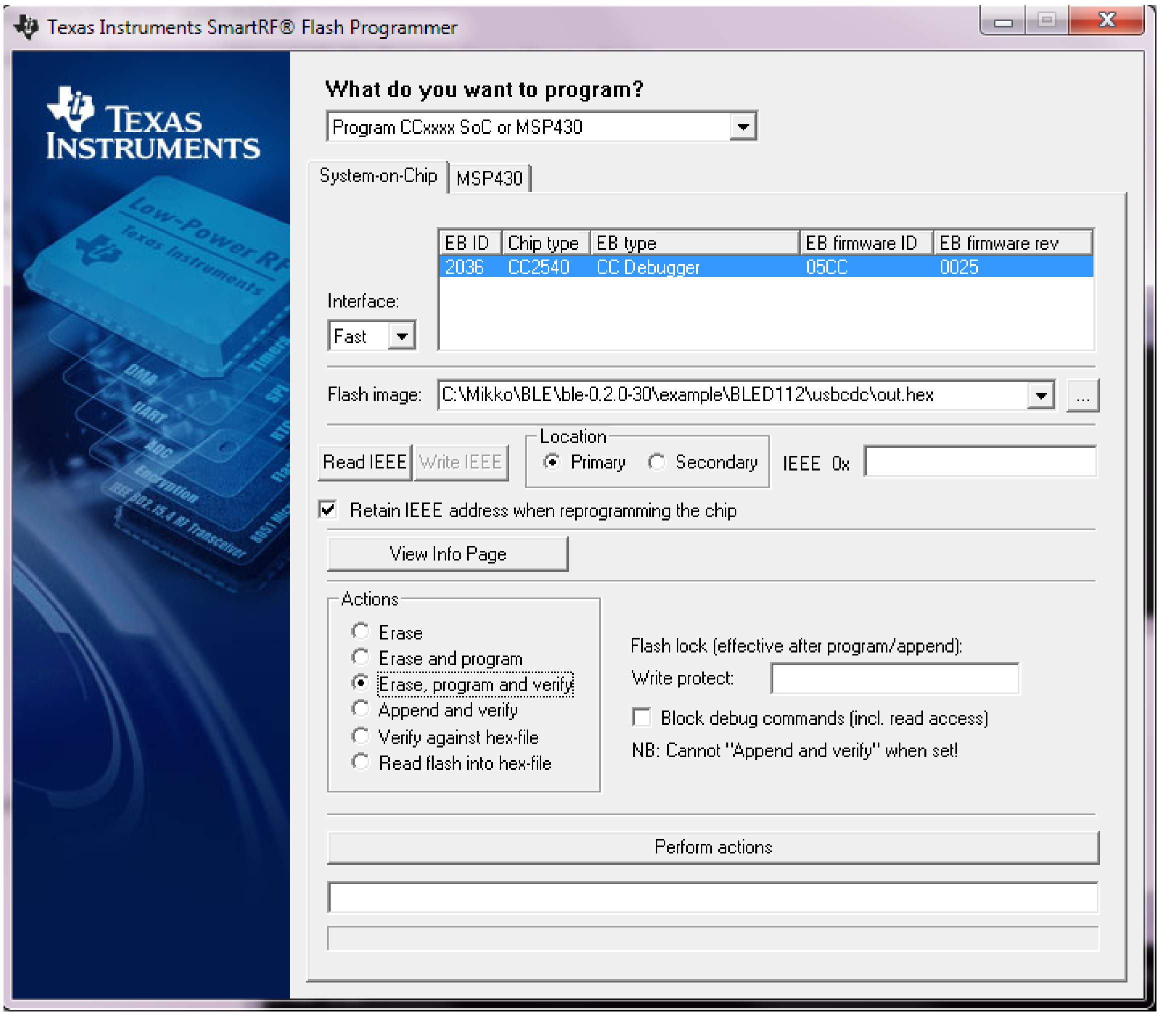
Installing the firmware with TI's flash tool

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 BLE113 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.

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**

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
2036	CC2540	CC Debugger	05CC	0025

Interface: Fast

Flash image: C:\Mikko\BLE\ble-0.2.0-30\example\BLED112\usbcdc\out.hex

Location: [✓ Primary] Secondary

IEEE 0x

☑ Retain IEEE address when reprogramming the chip

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!

Figure 6: TI's flash programmer tool

8 Contact information

Sales:	sales@bluegiga.com
Technical support:	support@bluegiga.com http://techforum.bluegiga.com
Orders:	orders@bluegiga.com
WWW:	http://www.bluegiga.com http://www.bluegiga.hk
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