BLUETOOTH SMART PROFILE TOOLKIT

DEVELOPER GUIDE

Wednesday, 10 September 2014

Version 3.4

Table of Contents

1 Version history

Version	Comments
2.4	v1.2 SW compatibility changes; USB set inactive (false) by default, ...)
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.
3.0	Project.xml, Config.xml, Hardware.xml sections are moved to the Bluetooth Smart Module Configuration Guide document.
3.1	Improved examples
3.2	Improved examples
3.3	Improved the user characteristic property documentation and an example added
3.4	Characteristic default type description added

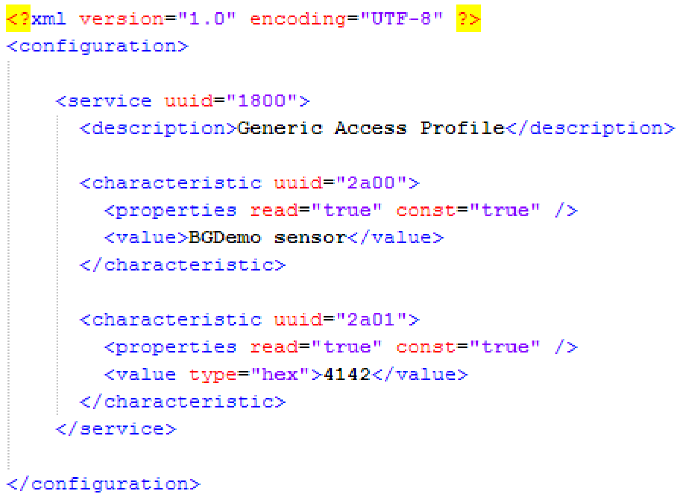
2 Introduction

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.

2.1 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
<?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: A profile toolkit example of GAP service

3 Introduction to Profiles, Services and Characteristics

3.1 Typical Bluetooth Smart Application Architecture

Bluetooth Smart 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.

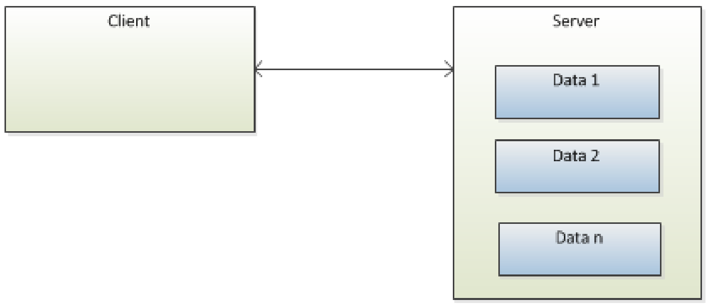


Figure: Bluetooth Smart device roles

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 Descriptor(s)** which provide additional information about the characteristic (characteristic user description, characteristic client configuration, vendor specific information etc.).

- **Characteristic Value** containing the value of a characteristic (for example temperature reading).

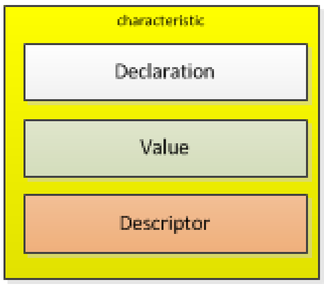


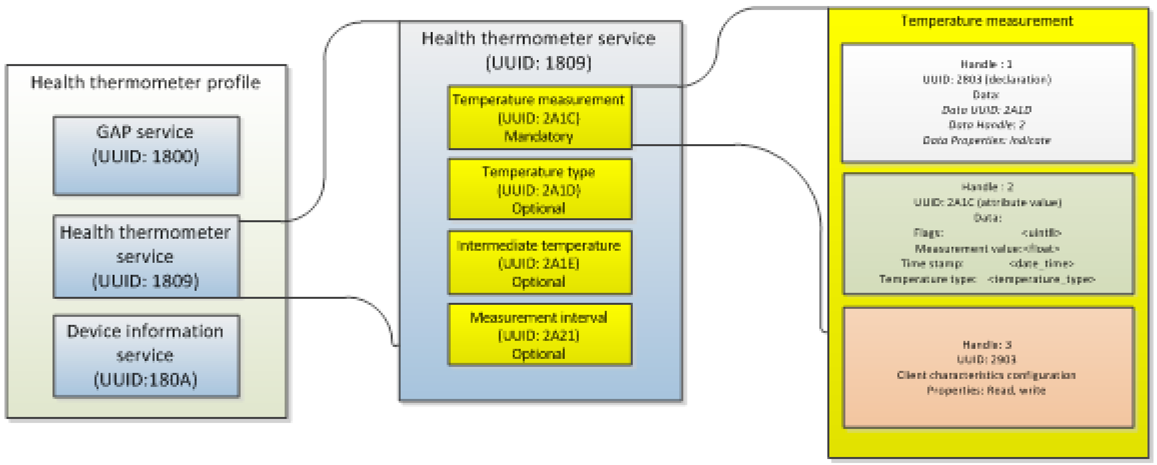
Figure: Characteristic structure

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

4 GATT Database File (gatt.xml)

This section of the document describes how to build Bluetooth Smart services and characteristics using the Bluegiga Bluetooth Smart Profile Toolkit and shows practical examples how to implements services and characteristics.

4.1 Defining Services

4.1.1 <service>: A Service Definition Tag

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. **Options:** **0000** to **FFFF:** 16-bit UUIDs are reserved for the services standardized by the Bluetooth SIG and must not be used for vendor specific implementations. **00000000-0000-0000-0000-000000000000** to **FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF:** 128-bit UUIDs are reserved for vendor 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. At the moment the **id** feature does not have any meaningful use.
type	The type field defines whether the service is a **primary** or a **secondary** service. When omitted, it will default to *type="primary"* At the moment the **type** feature does not have any meaningful use.
advertise	This option can be used to include the service UUID is the advertisement packet payload. **Options:** **true**
Example: A 128-bit vendor specific service which UUID is included in the advertisement data: *<service uuid="0bd51666-e7cb-469b-8e4d-2742f1ba77cc" advertise="true">*	
Example: A Bluetooth SIG standardized Heart Rate service which is included in the advertisement data: *<service uuid="180a" advertise="true">*	

4.1.2 <description> : A Service Description Tag

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

4.1.3 <include> : A Service Include Tag

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.

4.2 Defining Characteristics

4.2.1 <characteristic> : A Characteristic Definition Tag

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

Attribute	Description
uuid	Universally Unique IDentifier. The UUID uniquely identifies a characteristic. **Options:** **0000** to **FFFF:** 16-bit UUIDs are reserved for the characteristics standardized by the Bluetooth SIG and must not be used for vendor specific implementations. **00000000-0000-0000-0000-000000000000** to **FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF:** 128-bit UUIDs are reserved for vendor specific implementations.
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.
Examples: A Bluetooth SIG standardized characteristic with UUID 2a00 (device name): *<characteristic uuid="2a00">*	
Example: A vendor specific characteristic with UUID e7add780-b042-4876-aae1-112855353cc1 and ID **xgatt_data:** *<characteristic uuid="e7add780-b042-4876-aae1-112855353cc1" id="xgatt_data">*	

4.2.2 <properties> : A Characteristic Properties Tag

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

Attribute	Description
read	Characteristic value can be read over a *Bluetooth* connection using attribute read procedure.
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 using attribute write procedure.
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 that devices are bonded and link encrypted. ***read***-attribute must also be set to true.
authenticated_write	Writing characteristic value over a *Bluetooth connection requires that devices are bonded and link encrypted.* **write** *or* **write_no_response** attribute must also be set to true.
reliable_write	Allows using reliable write procedure to modify attribute, this is just a hint to GATT client. Stack always allows using reliable writes to be used to modify attributes.

4.2.3 <value> : A Characteristic Value Definition Tag

Characteristic value description

Attribute	Description
length	Maximum length for attribute. Length is fixed. **Range:** **0 - 255 (Bytes)**
variable_length	Attribute is variable in length. Maximum length needs also to be defined.

Attribute	Description
type	How to interpret element value. **Options:** **hex**: Value is hex **utf-8**: Value is string **user:** When this property is used in a characteristic and a remote device tries to read (with ATT read operation) the value, an *User Read Request* event is generated to the application (via BGscript or BGAPI). The application must then provide the attributes value or an error code to the remote device with the command *User Read Response*. This feature enables the application to dynamically generate the attributes value whenever it's being requested by a remote device. The user property can also be used with attributes that can be written. When a remote device writes an attribute with the user property an *Attribute Value* event is generated where the reason code is *attributes_attribute_change_reason_write_request_user.* The application must then accept or reject the write to the remote device with *User Write Response* command. **Default**: **utf-8**: Value is string
Example: A simple characteristic value length definition (20 bytes) and UTF-8 type (since it's the default). *<value length="20" />*	
Example: A variable length characteristic definition, where length can be from 0 up to 20 bytes *<value variable_length="true" length="20" />*	
Example: Characteristic type definition *<value type="hex" />*	
Example: A characteristic type and length definition *<value length="20" type="hex" />*	
Example: A characteristic length and user type property definitions. *<value length="20" type="user" />*	

There are two ways to specify the length (in bytes) of a characteristic. One way is to use the length="..." attribute inside the <value> tag, and the other is to provide the information as the tag content. In the case of a fixed/const value, you must provide the data as the tag content, since that is the only way to pre-populate the const value and it cannot be written at runtime.

When you specify type="hex", this causes the bgbuild compiler to interpreted the data inside as hexadecimal, which affects the interpreted length. So this:

<value type="hex">0001</value>

...is two bytes, but this:

<value type="utf-8">0001</value>

...is four bytes.

The value given as the tag content will take precedence over a specified length="..." attribute in the <value> tag.

Non-const values will not be pre-populated with the data given, but in fact they have to be initialized at runtime. That is, the value provided for a non-const characteristic is only used for memory allocation, and not for initialization.

4.2.4 <description> : A Characteristic User Description Tag

Characteristic User Description. A user friendly description for the characteristic. This is exposed by the GATT database to remote devices and can for example be used to populate user interfaces with user friendly strings.

Constant string example:

Characteristic Description Example

```
<characteristic uuid="2a37">
    <properties notify="true" />
    <value length="2" />
    <description>Heart Rate Measurement</description>
</characteristic>
```

Properties tag can be used to allow remote modification of attribute, for example to allow remote reading but writing required bonding:

Characteristic Description Example

```
<characteristic uuid="2a37">
    <properties notify="true" />
    <value length="2" />
    <description>
        <properties read="true" write="true" authenticated_write="true" />
        <value variable_length="true" length="20" />
    </description>
</characteristic>
```

NOTE: If description is writable then GATT Parser automatically adds extended properties attribute with writable_auxiliaries bit set to be Bluetooth compliant.

4.2.5

4.2.6 <aggregate> : Characteristic Aggregated Format Definition Tag

4.2.7

This tag helps creating aggregated characteristic format descriptor, by automatically converting id:s to attribute handles.

Attributes id:s should refer to characteristic presentation format descriptors.

Example how to add characteristic aggregate.

4.2.8

Characteristic Aggregate Format Example

```
<characteristic uuid="da8a80c0-829d-498f-b70b-e85c95e0f839">
    <properties notify="true" read="true"/>
     <value length="10" />
    <aggregate>
        <attribute id="format1" />
        <attribute id="format2" />
    </aggregate>
</characteristic>
```

4.2.9 <descriptor> : A Characteristic Descriptor Definition Tag

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 how to add characteristic descriptor with type UUID 2908.

Characteristic Descriptor Example

```
<characteristic uuid="2a4d" id="hid_input">
    <properties notify="true" read="true"/>
 <value length="3" />
    <descriptor uuid="2908">
        <properties read="true" const="true" />
        <value type="hex" >0001</value>
    </descriptor>
</characteristic>
```

Max amount of attributes

The compiler might output the following message: "Dynamic attribute count is xx which is larger than maximum supported of 64"

In fact, the amount of attributes, other than the ones which are constant or kept in RAM, is limited to 64

Max amount of attributes

The compiler might output the following message: "Dynamic attribute count is xx which is larger than maximum supported of 64"

In fact, the amount of attributes, other than the ones which are constant or kept in RAM, is limited to 64

4.3 Examples

The example below shows how to implement the Generic Access (GAP) service which needs to be included in every *Bluetooth* Smart device.

GAP Service

```
<?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>BLE112 Bluetooth Smart Module</value>
      </characteristic>

      <characteristic uuid="2a01">
        <properties read="true" const="true" />
        <value>0000</value>
      </characteristic>

    </service>

</configuration>
```

- GAP service has 16-bit UUID 1800. In this example the service UUID is not included in the advertisement packet payload.
- **<description>Generic Access Profile</description>** is simply used as a comment to identify the service name and type.
- The GAP service implements a two mandatory GAP service characteristics with UUID 2a00 (device name) and UUID 2a01 (appearance)
  - Both characteristics have **read** property, so they can be read over a *Bluetooth* connection
  - Both characteristics are marked const, so the values are constant and cannot be changed by the application, but are permanently stored on the flash
  - The value of the device name characteristic is: "BLE112 Bluetooth Smart Module"
  - The value of the appearance characteristic is: 0000

The second example below shows how to implement the Heart Rate (HRS) service.

GAP Service

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

   <service uuid="180d" advertise="true">

      <characteristic uuid="2a37" id="xgatt_hrs_2a37">
          <properties notify="true" />
          <value length="2" />
      </characteristic>

    </service>

</configuration>
```

- HR service has 16-bit UUID 180d. In this example the service UUID is included in the advertisement packet payload, so a remote device can recognize the service simply by receiving an advertisement packet.
- The HR service implements only the single mandatory characteristic with UUID 2a37 called Heart Rate Measurement.
    - The HR measurement characteristics has **id xgatt_hrs_2a37**, so a for example a BGScript application can update the value when a new HR measurement is made.
    - The HR measurement value is not marked as const as the value needs to be changed by the application.
    - The HR measurement characteristics has **notify** property as defined by the standard. When the HR measurement value changes, it will be automatically sent to the remote device, if it has registered to listen the notifications.
    - The HR measurement characteristic length is 2 bytes (fixed).

The third example below shows how to implement a vendor specific service and characteristic:

GAP Service

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

    <service uuid="0bd51666-e7cb-469b-8e4d-2742f1ba77cc" advertise="true">
        <description>Cable replacement service</description>

        <characteristic uuid="e7add780-b042-4876-aae1-112855353cc1" id="xgatt_data">
            <description>Data</description>
            <properties write="true" indicate="true" />
            <value variable_length="true" length="20" type="user" />
        </characteristic>
    </service>

</configuration>
```

- HR service has 128-bit UUID e7add780-b042-4876-aae1-112855353cc1. In this example the service UUID is included in the advertisement packet payload.
- The service implements a single characteristic with UUID e7add780-b042-4876-aae1-112855353cc1.
    - The characteristics has **id xgatt_data**, so a for example a BGScript application can update the value when need.
    - The characteristics is not marked as const as the value needs to be changed by the application.
    - The characteristics has **write** ad **indicate** properties so it can be written by a remote device or indicated to a remote device when the value changes.
    - The characteristic length is variable from 0 bytes up to the maximum value of 20 bytes.
    - Characteristic has also type **user** so the actual data is queried from the user application. See for example the *Cable Replacement Application Note* to see how this feature is used.

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