# BLUETOOTH LOW ENERGY PROFILE TOOLKIT

DEVELOPER GUIDE

Wednesday, 2 December 2020

Version 3.12

# Table of Contents

# 1 Version history

| Version | Comments |
|---|---|
| 2.1 | 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 instructions |
| 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 Low Energy 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 |
| 3.5 | Minor edits, textual style aligned, terminology check, GATT database limitations added for BLExxx and BT121 modules |
| 3.6 | encrypted_write and encrypted_read plus other minor clarifications added. |
| 3.7 | Attribute operation figures corrected |
| 3.8 | GATT limitations documentation corrected.<br><br>Chapter 3.3.7 added |
| 3.9 | Corrected description of *write_no_response* attribute in <properties> section |
| 3.10 | Added note about *advertise* parameter, added limitation note for *encrypted_write*, *encrypted_read* and *encrypted_notify* parameters. |
| 3.11 | Added note about new capabilities of service *id* parameter in BT121 modules. |
| 3.12 | Renamed "Bluetooth Smart Ready" to "Bluetooth Dual Mode", "Bluetooth Smart" to "Bluetooth Low Energy" and "Classic" to "BR/EDR" according to the official Bluetooth SIG nomenclature |

# 2 Introduction

Bluetooth Low Energy services and characteristics are the basis of Bluetooth low-energy data exchange. They are used to describe the structure, access type, and security properties of the data exposed by a device, such as a heart-rate monitor. The BLE services and characteristics have a well-defined and structured format, and they can be easily described using XML mark-up language.

The Profile Toolkit is an XML-based mark-up language for describing the Bluetooth Low Energy services and characteristics, also known as the GATT database, in an easy human-readable and machine-readable format. The Bluetooth Low Energy Profile Toolkit Developer Guide walks you through the XML syntax with the Profile Toolkit and instructs you how to easily describe your own Bluetooth Low Energy services and characteristics, configure the access and security properties, and include the GATT database as a part of the firmware.

This guide also contains practical examples of both standardized Bluetooth, as well vendor proprietary services, which can serve as an example for your own development.

## 2.1 Understanding Profile, Services, Characteristics and the Attribute Protocol

### 2.1.1 GATT-Based Bluetooth Profiles

The profile specifies the structure in which data is exchanged. The profile defines elements, such as services and characteristics, used in a profile, but it may also contain definitions for security and connection-establishment parameters. Typically a profile consists of one or more services which are needed to accomplish a high-level use case, such as heart-rate or cadence monitoring. Standardized profiles allow device and software vendors to build interoperable devices and applications.Bluetooth SIG standardized profiles are defined in profiles specifications.

These are available at: https://developer.bluetooth.org/gatt/profiles/Pages/ProfilesHome.aspx

### 2.1.2 Services

Services are a collection of data composed of one or more characteristics used to accomplish a specific function of a device, such as battery monitoring or temperature data, rather than a complete use case.

Standardized Bluetooth SIG are defined in service specifications, which are available at: https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

### 2.1.3 Characteristics

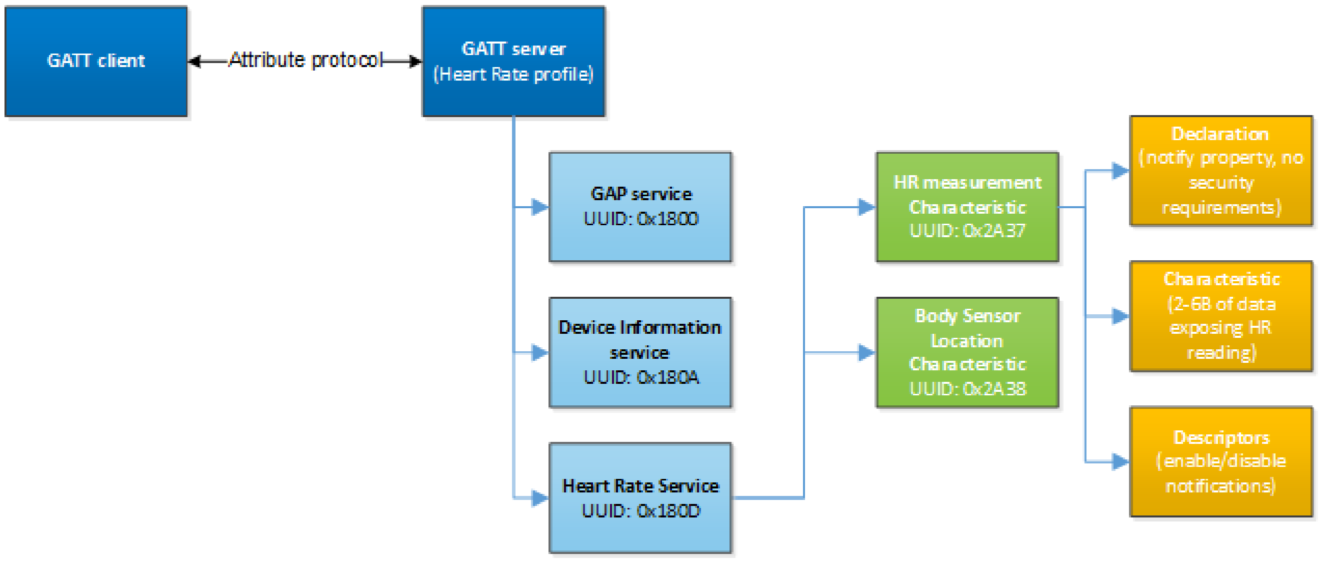
A characteristic is a value used in a service, either to (1) expose and/or exchange data and/or (2) control information. Characteristics have a well-defined, known format. They also contain information about how the value can be accessed, what security requirements must be fulfilled, and ,optionally, how the characteristic value is displayed or interpreted. Characteristics may also contain descriptors that describe the value or permit configuration of characteristic data indications or notifications.

Standardized characteristics are defined in the Characteristic Specification, which are available at:

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

## 2.1.4 The Attribute Protocol

The Attribute protocol enables the data exchange between the GATT server and the GATT client. The protocol also provides a set of operations: how to query, write, indicate or notify the data and/or control information between the two GATT parties.


GATT client
Attribute protocol
GATT server (Heart Rate profile)
GAP service UUID: 0x1800
Device Information service UUID: 0x180A
Heart Rate Service UUID: 0x180D
HR measurement Characteristic UUID: 0x2A37
Body Sensor Location Characteristic UUID: 0x2A38
Declaration (notify property, no security requirements)
Characteristic (2-6B of data exposing HR reading)
Descriptors (enable/disable notifications)

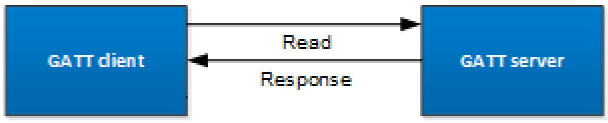


**Figure: Read operation**

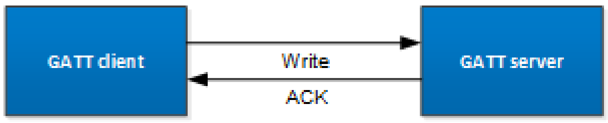


**Figure: Write operation**

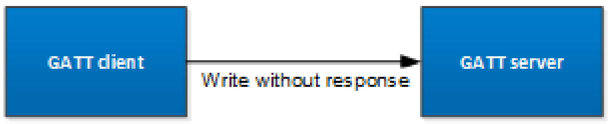


**Figure: Write without response operation**

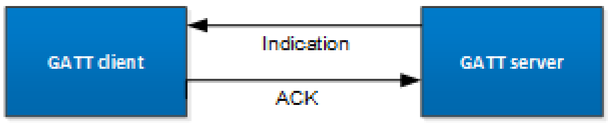


**Figure: Attribute indication**

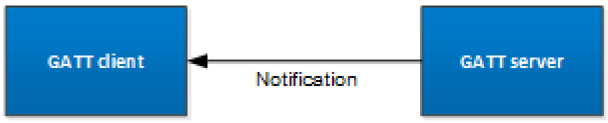
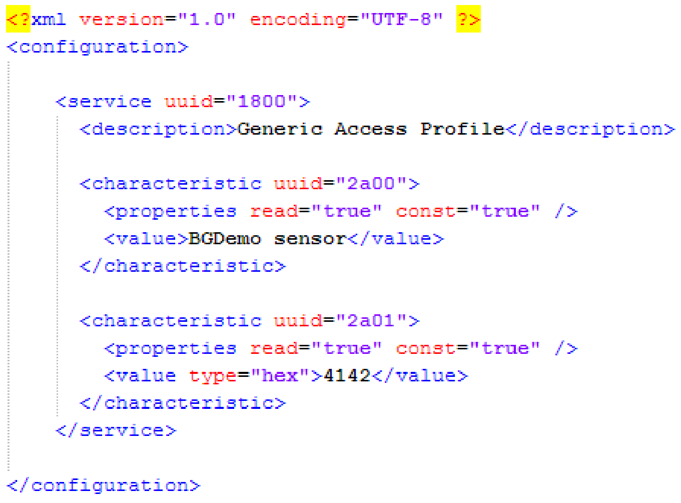


**Figure: Attribute notification**

## 2.1.5 The Profile Toolkit

The *Bluetooth* Low Energy Profile Toolkit is a simple set of tools, which can be used to describe GATT based *Bluetooth* Low Energy *services* and *characteristics*. The Profile Toolkit consists of a simple XML based description language and templates, which can be used to describe the devices GATT database. The Profile Toolkit also contains a compiler, which converts the XML in 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: A profile toolkit example of GAP service**

# 3 GATT Database file (gatt.xml)

This section of the document describes the XML syntax used in the *Bluetooth* Low Energy Profile Toolkit and walks you through the different options you can use when building *Bluetooth* Low Energy services and characteristics. A few practical GATT database examples are also shown.

## 3.1 Generic GATT Limitations

The following table lists limitations regarding GATT database.

| **Item** | **BLE112, BLE113, BLE121LR and BLED112** | **BT121** |
|---|---|---|
| Maximum number of characteristics | 64<br><br>All characteristics which do not have *const="true"* are counted in this | 64<br><br>All characteristics which do not have *const="true"* are counted in this |
| Maximum length of a characteristic value<br><br>**a** type="user"<br><br>**b** const="true"<br><br>**c** const="false" | **a** Unlimited<br><br>**b** Available free flash<br><br>**c** 255 bytes | **a** 511 bytes<br><br>**b** Available free flash, up to 255 bytes<br><br>**c** Available free memory, up to 255 bytes |
| Maximum number of attributes<br>in a single GATT database | 255 | Limited by available free flash |

## 3.2 Defining Services

### 3.2.1 <service>: Service Definition element

The XML element *<service>* is used to define a service.

The attributes are described in more detail in the table below.

| 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 are used for manufacturer specific implementations.<br><br>**Options:**<br><br>**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.<br><br>**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 below). This ID is not stored in the GATT database and a t the moment it does not have other practical uses in BLE products. In BT121 this ID is also used to generate an handle (and a constant revealed in the **constants** file) to be used with the new *dumo_cmd_gatt_server_set_service_status*. The ID can be used in the application code (BGScript) to manage the service instead of using its handle number. Every service must use a unique ID.<br><br>**Value:**<br><br>Any UTF-8 string |
| ***type*** | The type field defines whether the service is a **primary** or a **secondary** service. Typically this does not need to be used.<br><br>If not defined, the value of this attribute will default to *type="primary"* |
| **advertise** | This option can be used to include the service UUID in the advertisement packet payload.<br><br>Up to eight 16-bit UUIDs or one 128-bit UUID will fit into the advertisement data of BLE firmware.<br><br>Up to thirteen 16-bit UUIDs or one 128-bit UUID will fit into the advertisement data of BT121 firmware.<br><br>Note: you can use custom advertisements with the dedicated BGAPI/BGScript configuration commands, in which case this option will not be valid.<br><br>**Options:**<br><br>**true** : UUID is included in advertisement data<br><br>**false** : UUID is not included in advertisement data (default)<br><br>**NOTE!** Service with ***advertise*** attribute set to *"true"* must be placed in one of the 8 first services of the gatt.xml tree. |

| Attribute | Description |
|---|---|
| **sdp** | This option can be used to exclude the service from BT BR/EDR SDP database.<br><br>**Options:**<br><br>**true** : UUID is included in SDP database (default)<br><br>**false** : UUID is not included in SDP database |
| **Example:** A 128-bit vendor specific service the UUID which will be included in the advertisement data:<br><br>*<service uuid="0bd51666-e7cb-469b-8e4d-2742f1ba77cc" advertise="true">*<br><br>You can generate your own 128-bit UUIDs at *http://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx* | |
| **Example**: A *Bluetooth SIG* standardized *Heart Rate Service* which is included in the advertisement data:<br><br>*<service uuid="180a" advertise="true">* | |

### 3.2.2 <description>: Service Description element

The XML element *<description>* can be used for informative purposes (commenting) and are not exposed in the actual GATT database.

### 3.2.3 <include>: Service Include element

The service element *<include>* and its attribute are used to define another service in the gatt.xml that should be included within this service.

| Attribute | Description |
|---|---|
| ***id*** | This attribute refers to the service ID of the other service that should be included within this service. |
| Example: Including Hear Rate service (having id of "hrs" in its definition element) within GAP service<br><br>*<!-- Generic Access Service -->*<br><br>*<service uuid="1800">*<br><br>*<!-- Include HR Service -->*<br><br>*<include id="hrs" />*<br><br>*…*<br><br>*</service>* | |

## 3.3 Defining Characteristics

### 3.3.1 <characteristic>: Characteristic Definition element

The XML element *<characteristic>* is used to define define a characteristic, it's UUID and internal ID used in applications.

The attributes are described in more detail in the table below.

| Attribute | Description |
|---|---|
| ***uuid*** | Universally Unique IDentifier. The UUID uniquely identifies a characteristic.<br><br>**Options:**<br><br>**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.<br><br>**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 can be used in the application code (BGScript) to manage the characteristic instead of using its handle number.<br><br>When the project is compiled with the **BGBuild** compiler a text file called **attributes.txt** for BLE and **constants** for BT121 is generated. This file contains the **id**s and corresponding handle values (dec).<br><br>The value of an ID can be any UTF-8 string. Every characteristic must use a unique ID. |
| **Examples:** A *Bluetooth SIG* standardized characteristic with UUID 2a00 (corresponding to device name):<br><br>*<characteristic uuid="2a00">* | |
| **Example:** A vendor specific characteristic with UUID e7add780-b042-4876-aae1-112855353cc1 and ID **xgatt_data:**<br><br>*<characteristic uuid="e7add780-b042-4876-aae1-112855353cc1" id="xgatt_data">* | |

### 3.3.2 <properties>: Characteristic Properties element

The XML element *<properties>* is used to define the properties of a characteristic. A characteristic may have one or more properties. The attributes are described in more detail in the table below. Possible values for all attributes are **true** or **false** and in all cases default value is false.

| Attribute | Description |
|---|---|
| ***read*** | This attribute defines that a characteristic value can be read over a *Bluetooth* connection using the **Attribute Read** procedure. |
| ***const*** | This attribute defines that the characteristic value is stored in flash memory and that it cannot be modified after programming.<br><br>The benefit of constant values is that no RAM is allocated for them leaving more RAM to the application.<br><br>If the const attribute is not used the data must be initialized during runtime. |
| ***write*** | This attribute defines that the characteristic value can be written over a *Bluetooth* connection using the **Attribute Write** procedure. |
| ***write_no_response*** | This attribute defines that the characteristic value can be written only using the **Write Without Response** procedure.<br><br>Note that this means the write operation is not confirmed over the attribute protocol unlike the regular write operation. |
| ***notify*** | This attribute defines that the characteristic value can be notified.<br><br>Notification is **not** confirmed over the attribute protocol. |
| ***indicate*** | This attribute defines that the characteristic value can be indicated.<br><br>Indication is confirmed over the attribute protocol. |
| ***authenticated_read*** | This attribute defines that reading the characteristic value over a *Bluetooth* connection requires that the devices are bonded using MITM protection and connection is encrypted.<br><br>**NOTE!** Attribute ***read*** must be set to *"true"*. |
| ***authenticated_write*** | This attribute defines that writing the characteristic value over a *Bluetooth* connection requires that the devices are bonded using MITM protection and connection is encrypted.<br><br>**NOTE!** Attribute ***write*** *or* ***write_no_response*** must also be set to *"true"*. |
| ***authenticated_notify*** | This attribute defines that Notifications and Indications require that the devices are bonded using MITM protection and connection is encrypted. This attribute is only valid with the Bluetooth stack in the BT121.<br><br>**NOTE!** Attribute ***notify*** *or* ***indicate*** must also be set to *"true"*. |
| ***encrypted_read*** | This attribute defines that reading the characteristic value over a *Bluetooth* connection requires that the connection is encrypted. With iOS 9.1 and newer, devices must also be bonded at least with Just Works pairing. This attribute is only valid with the Bluetooth stack in the BT121.<br><br>**NOTE!** Attribute ***read*** must be set to *"true"*. |

| Attribute | Description |
|---|---|
| ***encrypted_write*** | This attribute defines that writing the characteristic value over a *Bluetooth* connection requires that the connection is encrypted. With iOS 9.1 and newer, devices must also be bonded at least with Just Works pairing. This attribute is only valid with the Bluetooth stack in the BT121.<br><br>**NOTE!** Attribute ***write*** *or* ***write_no_response*** must also be set to *"true"*. |
| ***encrypted_notify*** | This attribute defines that Notifications and Indications require the connection to be encrypted. This attribute is only valid with the Bluetooth stack in the BT121. With iOS 9.1 and newer, devices must also be bonded at least with Just Works pairing. This attribute is only valid with the Bluetooth stack in the BT121.<br><br>**NOTE!** Attribute ***notify*** *or* ***indicate*** must also be set to *"true"*. |
| **reliable_write** | This attribute defines that reliable write procedures to modify attributes are allowed. This is just to inform the GATT client of this possibility. The stack always allows using reliable writes to modify attributes. |

### 3.3.3 <value>: Characteristic Value Definition element

The XML element *<value>* is used to define a characteristic value description.

The attributes are described in more detail in the table below.

| Attribute | Description |
|---|---|
| ***length*** | This atttribute defines the fixed length of an attribute or the maximum length of an attribute if the attribute *variable_length* (see below) is used.<br><br>**Range:**<br><br>**0 - 255 (Bytes)** |
| ***variable_length*** | This attribute defines that the length of an attribute is variable.<br><br>**Range:**<br><br>**0 - 255 (Bytes)** |
| ***type*** | This attribute defines how to interpret the element value.<br><br>**Options:**<br><br>**hex**: Value is hex<br><br>**utf-8**: Value is string<br><br>**user:** When this property is used in a characteristic and a remote device tries to read the value (with ATT read operation), 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.<br><br>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.<br><br>**Default:**<br><br>**utf-8**: Value is string |
| **Example**: A simple characteristic value length definition (20 bytes) and UTF-8 type (since it's the default).<br><br>*<value length="20" />* | |
| **Example:** A variable length characteristic definition, where length can be from 0 up to 20 bytes.<br>*<value variable_length="true" length="20" />* | |
| **Example:** Characteristic type definition.<br>*<value type="hex" />* | |
| **Example:** A characteristic type and length definition.<br><br>*<value length="20" type="hex" />* | |
| **Example:** A characteristic length and user type property definitions.<br><br>*<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>* element.

Alternatively you can simply include the data inside the *<value>* element, which for example must be done with constant data. Using this will override the length defined with *length="".*

**Example**:

*<characteristic uuid="2a00">*
*<properties read="true" const="true" />*
*<value>Device name</value>*
*</characteristic>*

When you specify *type="hex"*, bgbuild compiler will interpret the data inside as hexadecimal, which affects the interpreted length.

**Example:**

*<value type="hex">0001</value>* is two bytes

*<value type="utf-8">0001</value>* is four bytes.

### 3.3.4 <description>: Characteristic User Description element

The XML element *<description>* is used to define a user description of the characteristic. This can be used to name a characteristic in a user friendly way. This information is exposed in the GATT database can be used to populate user interfaces with user friendly strings.

Below some constant string examples:

**Characteristic Description Example**

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

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

**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>
```

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

### 3.3.5 <aggregate>: Characteristic Aggregated Format Definition element

The XML element *<aggregate>* enables the creation of an aggregated characteristic format descriptor by automatically converting ID's to attribute handles.

Attribute ID's should refer to characteristic presentation format descriptors.

Below an example how to add a characteristic aggregate:

**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>
```

### 3.3.6 <descriptor>: Characteristic Descriptor Definition element

The XML element *<descriptor>* can be used to define a generic characteristic descriptor.

Descriptor properties are defined by the *<properties>* element and only read and/or write access is allowed. Value is defined by *<value>* element the same way as for characteristic value.

Below an example on how to add a 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>
```

### 3.3.7 Notes

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

This refers to the maximum number of characteristics seen in the table in chapter 3.1

⚠ The compiler might output the following message: *"Attribute database size exceeds maximum supported of 255 entries"*.

This refers to the maximum number of attributes seen in the table in chapter 3.1

It should be noted that each service or characteristic often uses more than one attribute handle, for example some characteristics might make use of three handles (declaration, data, description).

## 3.4 Examples

The example below shows how to implement the Generic Access (GAP) service which needs to be included in every *Bluetooth* Low Energy 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>BT121 Bluetooth Dual Mode 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 two mandatory GAP service characteristics: 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 in the flash memory.
  - The value of the device name characteristic is: *"BT121 Bluetooth Dual Mode Ready 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 characteristic has **id** of **xgatt_hrs_2a37**, so a for example a BGScript application can update its value when a new HR measurement is made by referring to the constant called **xgatt_hrs_2a37** having the value of the characteristic handle.
    - The HR measurement value is not marked as const so the value needs to be initialized/changed by the application.
    - The HR measurement characteristic has **notify** property as defined by the standard. When the HR measurement value changes, it will be automatically sent to the remote device, if the latter has registered to listen to 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**

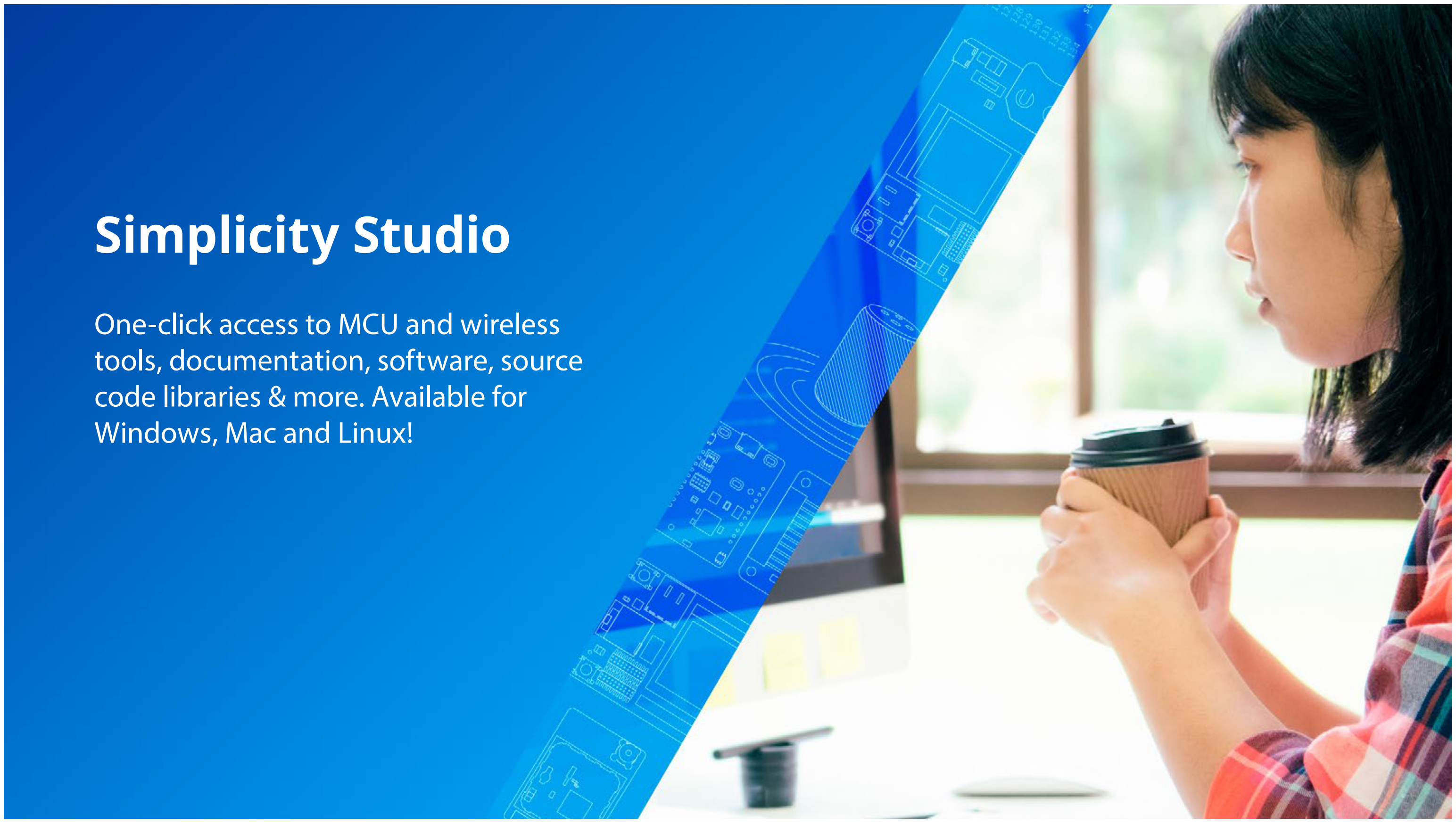
```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

    <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>
```

- The custom service has 128-bit UUID 0bd51666-e7cb-469b-8e4d-2742f1ba77cc. 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 characteristic has **id** of **xgatt_data**, so for example a BGScript application can update the value when needed by referring to the constant called **xgatt_data** having the value of the characteristic handle.
    - The characteristic is not marked as const so the value needs to be initialized/changed by the application.
    - The characteristic 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 *BLE Cable Replacement Application Note* to see how this feature is used.


Simplicity Studio
One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

**IoT Portfolio**
www.silabs.com/IoT

**SW/HW**
www.silabs.com/simplicity

**Quality**
www.silabs.com/quality

**Support & Community**
www.silabs.com/community

SILICON LABS

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

**http://www.silabs.com**