UG118：*Bluetooth*®Profile Toolkit 开发人员指南

Bluetooth GATT 服务和特性是 Bluetooth 数据交换的基础。它们可用于描述设备的结构、存取类型和安全属性，如作为心率监测器。Bluetooth 服务和特性采用明确的结构化格式，便于使用 XML 标记语言进行描述。

Profile Toolkit 是一种基于 XML 标记语言的程序，用于说明 Bluetooth 服务和特性，也被称为“GATT 数据库”，其格式既方便人类阅读，同样便于计算机读取。本指南将引导您认识 Profile Toolkit 中使用的 XML 语法，指导您轻松描述您的 Bluetooth 服务和特性、配置访问和安全属性，以及将 GATT 数据库作为固件的一部分。

本指南也包含各类实用示例，展示使用 Bluetooth 标准化服务和供应商专有服务的情形。这些示例可作为您开发工作的良好开端。

内容要点

- 了解 Bluetooth GATT 配置文件、服务、特性和属性协议
- 使用 Profile Toolkit 建立 GATT 数据库

 Copyright © 2023 by Silicon Laboratories

1. 了解配置文件、服务、特性和属性协议

本节大致介绍了 Bluetooth 配置文件、服务和特性，以及如何在 GATT 服务器和客户端的数据交换中使用属性协议。有关这些主题的更多信息，请访问 Bluetooth SIG 网站：https://www.bluetooth.com/specifications/specs/。

1.1 基于 GATT 的 Bluetooth 配置文件和服务

Bluetooth 配置文件指定数据交换的结构。配置文件对其中使用的服务和特性等元素进行定义，也可以包含安全和建立联系参数的定义。通常来说，配置文件由一种或多种服务组成，用于实现高水平的使用实例，如心率或节奏监测。标准化配置文件使设备和软件供应商能够建立交互运作的设备和应用。

1.2 服务

服务是指由一个或多个特性组成的数据集合，这些特性用于完成某设备的特定功能，如电池监测或温度数据，而不是用于完整的使用实例。

1.3 特性

特性是指服务中使用的某个值，可用来展现自身内容、交换数据或控制信息。特性有明确定义的已知格式。特性中还包含获取特性值的方法、需要履行的安全要求以及显示或解释特性值的方法（可选）。特性也可以包含各类描述符，用于说明特性值或允许配置特性数据指示或通知。

1.4 属性协议

属性协议能实现 GATT 服务器和 GATT 客户端之间的数据交换。该协议也提供一系列操作，包括如何对数据进行查询、写入、指示或通知，以及/或控制 GATT 服务器和客户端之前的信息往来。

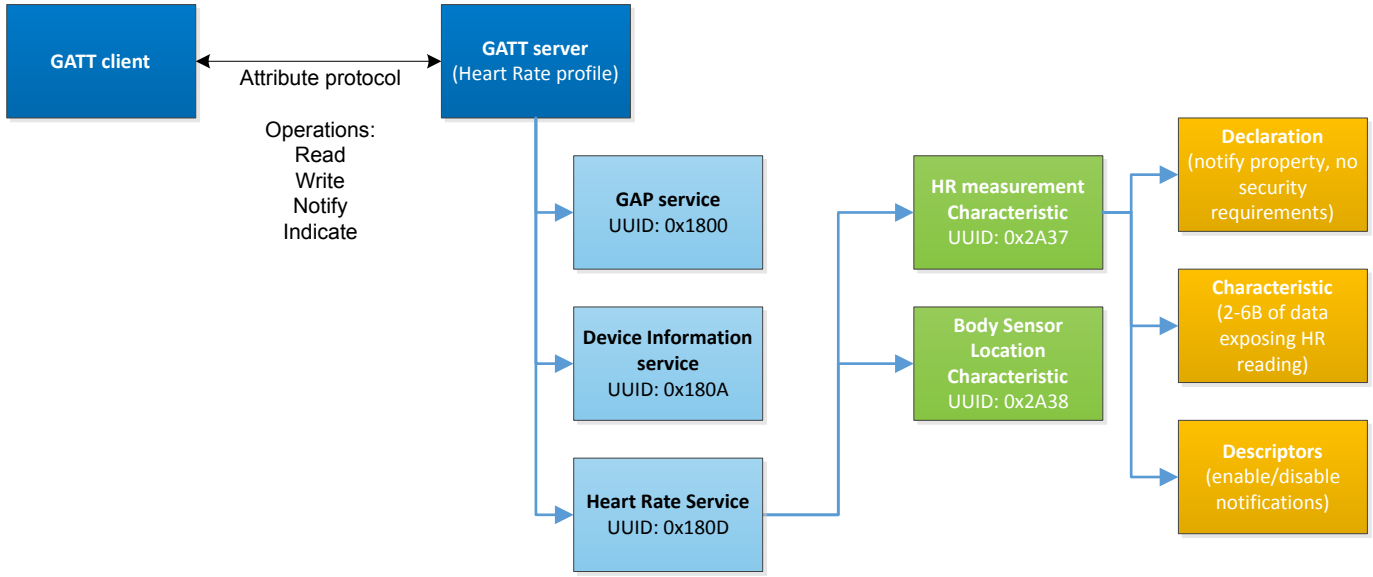


Figure 1.1. 配置文件、服务及特性的关系图

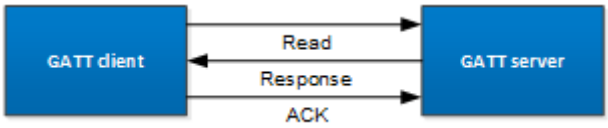


Figure 1.2. 属性读取操作

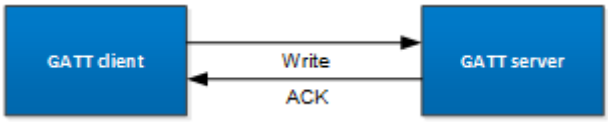


Figure 1.3. 属性写入操作

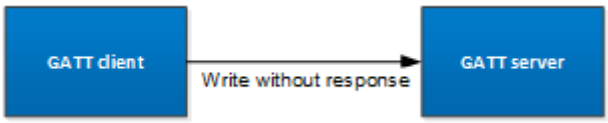


Figure 1.4. 无回应属性写入操作

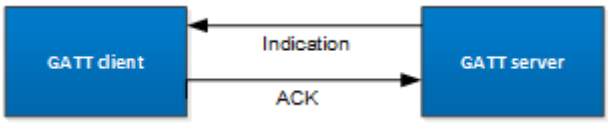


Figure 1.5. 属性指示操作

Figure 1.6. 属性通知操作

2. 使用 Profile Toolkit 建立 GATT 数据库

本节内容对 Bluetooth Profile Toolkit 使用的 XML 语法进行说明，并引导您认识建立 Bluetooth 服务和特性时可以使用的不同选项。

还给出了一些实用的 GATT 数据库示例。

2.1 一般局限性

下表显示了 EFR32BG 设备支持的 GATT 数据库的局限性。

项目	限制	局限
特性最大数量	不限；操作中受到数据库中属性总数的限制	所有不含 **const="true"** 属性的特性都包含在内。
type="user" 特性最大长度	255 字节	这些特性由应用处理，这意味着可用的 RAM 数量会限制最大长度。 注意：这些特性不支持 GATT 程序**写入长特性值**、**可靠的写入**和**读取多个特性值**。
type="UTF-8/Hex" 特性最大长度	255 字节	如果 **const="true"**，那么设备上的可用闪存量决定了此限制。 如果 **const="false"**，则 RAM 将被分配用于存储其值的特性。所用设备上可用闪存量对此进行了定义。
单个 GATT 数据库中的最大属性数	255	一个特性通常使用 3-5 个属性。
最大可通知特性数量	64	
最大功能数量	16	功能的逻辑状态将确定每个服务/特性的可见性。

2.2 <gatt>

GATT 数据库，连同各服务和特性，必须通过 XML 的 **<gatt>** 属性进行说明。

Parameter	Description
out	Filename for the GATT C source file **Value:** Any UTF-8 string. Must be valid as a filename and end with '.c' **Default:** gatt_db.c
header	Filename for the GATT C header file **Value:** Any UTF-8 string. Must be valid as a filename and end with '.h' **Default:** gatt_db.h
db_name	GATT database structure name and the prefix for data structures in the GATT C source file. **Value:** Any UTF-8 string; must be valid in C. **Default:** bg_gattdb_data
name	Free text, not used by the database compiler **Value:** Any UTF-8 string **Default:** Nothing
prefix	Prefix to add to each 'id' name for defining the handle macro that can be referenced from the C application. **Value:** Any UTF-8 string. Must be valid in C. **Default:** gattdb_ For example: If prefix="gattdb_" and id="temp_measurement" for a particular characteristic, then the following will be generated in the GATT C header file: #define gattdb_temp_measurement X (where X is the handle number for the temp_measurement characteristic)
generic_attribute_service	If it is set to true, Generic Attribute service and its service_changed characteristic will be added in the beginning of the database. The Bluetooth stack takes care of database structure change detection and will send service_changed notifications to clients when a change is detected. In addition, this will enable the GATT-caching feature introduced in Bluetooth 5.1. **Values:** **true:** Generic Attribute service is automatically added to the GATT database and GATT caching is enabled. **false:** Generic Attribute service is not automatically added to the GATT database and GATT caching is disabled. **Default:** false
gatt_caching	The GATT caching feature is enabled by default if generic_attribute_service is set to true. However, it can be disabled by setting this attribute to false.

示例: GATT 数据库定义。

```
<?xml version="1.0" encoding="UTF-8" ?> <gatt out="my_gatt_db.c" header="my_gatt_db.h" db_name="my_gatt_db_" prefix="my_gatt_"
generic_attribute_service="true" name="My GATT database"> … </gatt>
```

2.3 <capabilities_declare>

使用**功能**可使 GATT 数据库服务和特性显示/隐藏。一项功能必须在 <capability> 元素中声明，GATT 数据库中的全部功能必须先在包含一系列 <capability> 元素的 <capabilities_declare> 元素中声明。数据库中功能的最大数量为 16。

该新功能不会影响旧式 GATT XML 数据库（比 Silicon Labs Bluetooth 协议栈版本 2.4.x 更早的版本）。由于没有明确声明新功能的任何兼容性，所有服务和特性将仍对远程 GATT 客户端显示。

示例：功能声明

```
<capabilities_declare> <capability enable="false">feature_1</capability> <capability enable="false">feature_2</capability> </capabilities_declare>
```

2.3.1 <capability>

每项功能必须使用 <capability> 元素在 <capabilities_declare> 元素中单独声明。<capability> 元素拥有一个名为“enable”的属性，该属性表明功能在数据库初始化的默认状态。

<capability> 元素的文本值将作为已生成数据库 C 头文件中功能的标识符名。因此，该文本值必须在 C 中有效。

功能的继承

服务和特性可声明希望使用的功能。如果未声明任何功能，则需遵守以下继承规则：

1. 如果某一服务未声明任何功能，则拥有 <capabilities_declare> 元素的全部功能。
2. 如果某一特性未声明任何功能，则拥有所属服务的全部功能。如果在 <capabilities_declare> 服务中说明了功能的子集，则特性仅继承该子集。
3. 某一特性的全部属性继承该特性的功能。

可见性

可启用/禁用功能，以按以下逻辑使服务和特性对 GATT 客户端显示/隐藏：

1. 当**启用**某一服务的**至少一项**功能时，则**显示**该服务及其全部特性。
2. 当**禁用**某一服务的**全部**功能时，则**隐藏**该服务及其全部特性。
3. 当**启用**某一特性的**至少一项**功能时，则**显示**该特性及其全部属性。
4. 当**禁用某一特性的全部功能时，则隐藏该特性及其全部属性。**

Parameter	Description
enable	Sets the default state of a capability at database initialization. **Values:** **true:** Capability is enabled. **false:** Capability is disabled. **Default:** true

示例：功能声明

```
<capabilities_declare> <!-- This capability is enabled by default and the identifier is cap_light --> <capability enable="true">cap_light</capability> <!-- This capability is disabled by default and the identifier is cap_color --> <capability enable="false">cap_color</capability> </capabilities_declare>
```

2.4 <service>

GATT 服务定义通过 XML 的 **<service>** 属性及其参数进行定义。

下表中给出了可用于定义相关值的参数。

Parameter	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 vendor specific implementations. **Range:** **0x0000 – 0xFFFF**: Reserved for Bluetooth SIG standardized services **0x00000000-0000-0000-0000-000000000000 - 0xFFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF**: Reserved for vendor specific services.
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). Typically, this does not need to be used. **Value:** 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. **Values:** **primary**: a primary service **secondary**: a secondary service **Default**: primary
advertise	This field defines if the service UUID is included in the advertisement data. The advertisement data can contain up to 13 16-bit UUIDs or one (1) 128-bit UUID. **Values:** **true**: UUID included in advertisement data **false**: UUID not included in advertisement data **Default**: false **Note:** You can override the advertisement data with the GAP API, in which case this is not valid.

示例: 一例通用访问配置文件 (GAP) 服务定义。

```
<!-- Generic Access Service -->
<service uuid="1800">
      …
</service>
```

示例：一例供应商特定服务定义。

```
<!-- A vendor specific service -->
<service uuid="25be6a60-2040-11e5-bd86-0002a5d5c51b">
      …
</service>
```

示例: 一例带有 UUID 的心率服务定义，包含在广告数据内，ID 为“hrs”。

```
<!-- Heart Rate Service -->
<service uuid="180D" id="hrs" advertise=” true” >
      …
</service>
```

Note: 您可以通过如下网址生成自己的 128-位 UUID：http://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx

2.4.1 <capabilities>

一项服务可使用 <capabilities> 元素声明其拥有的功能。该元素包含一系列 <capability> 元素，这些元素的标识符**必须同时属于** <capabilities_declare> 元素。“enable”属性不影响此上下文中声明的功能，因此可排除在外。

如果某一服务未声明任何功能，按照**继承规则**，该服务拥有 **<capabilities_declare>** 的**全部功能**。

当**启用**某一服务的**至少一项**功能时，则**显示**该服务及其全部特性；当**禁用**某一服务的**全部功能**时，则**隐藏**该服务及其全部特性。

示例：功能声明

```
<capabilities> <capability>cap_light</capability> <capability>cap_color</capability> </capabilities>
```

2.4.2 <informativeText>

XML 元素 <informativeText> 可以用于提供信息（注释），并且不会在实际 GATT 数据库中可见。

2.4.3 <include>

利用 XML 的 **<include>** 属性，可以在一项服务内调用另一项服务。

Parameter	Description
id	ID of the included service **Value:** ID of another service

示例: 在 GAP 服务中调用心率服务。

```
<!-- Generic Access Service -->
<service uuid="1800">

      <!-- Include HR Service -->
      <include id="hrs”  />
      …
</service>
```

2.5 <characteristic>

设备所展示的所有特性均使用 XML 的 **<characteristic>** 属性及其参数进行定义，并且必须用在 **<service>** XML 属性标签内。

下表中给出了可用于定义相关值的参数。

Parameter	Description
uuid	Universally Unique Identifier. The UUID uniquely identifies a characteristic. 16-bit values are used for the services defined by the Bluetooth SIG and 128-bit UUIDs can be used for vendor specific implementations. **Range:** **0x0000 – 0xFFFF**: Reserved for Bluetooth SIG standardized characteristics. **0x00000000-0000-0000-0000-000000000000 to 0xFFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF** : Reserved for vendor specific characteristics.
id	The ID is used to identify a characteristic. The ID is used within a C application to read and write characteristic values or to detect if notifications or indications are enabled or disabled for a specific characteristic. When the project is built, the generated GATT C header file contains a macro with the characteristic 'id' and corresponding handle value. **Value:**Any UTF-8 string
const	Defines if the value stored in the characteristic is a constant. **Default:** false
name	Free text, not used by the database compiler. **Value:**Any UTF-8 string **Default:** Nothing

示例: 为 GAP 服务添加设备名称特性。

```
<!-- Generic Access Service -->
<service uuid="1800">

      <!-- Device name -->
      <characteristic uuid="2a00">
            …
      </characteristic>
      …
</service>
```

示例: 为带有 ID 属性的供应商特定服务添加供应商特定特性。

```
<!-- A vendor specific service -->
<service uuid="25be6a60-2040-11e5-bd86-0002a5d5c51b">

      <!-- My proprietary data -->
      <characteristic uuid="59cd69c0-2043-11e5-a717-0002a5d5c51b" id="mydata” >
            …
      </characteristic>
      …
</service>
```

2.5.1 <capabilities>

一项特性可使用 <capabilities> 元素声明其拥有的功能。该元素包含一系列 <capability> 元素，所包含元素的标识符**必须同时由父服务声明（或部分继承）**。“enable”属性不影响此上下文中声明的功能，因此可排除在外。

如果某一特性未声明任何功能，按照**继承规则**，该特性拥有其所属**服务**的**全部功能**。某一特性的全部属性继承该特性的功能。

当**启用**某一特性的**至少一项**功能时，则**显示**该特性及其全部属性；当**禁用**某一特性的**全部功能**时，则**隐藏**该特性及其全部属性。

示例：功能声明

```
<capabilities> <capability>cap_light</capability> <capability>cap_color</capability> </capabilities>
```

2.5.2 <properties>

特性的访问属性及其权限等级通过 XML 属性的子属性 **<properties>** 进行定义，并且必须用在 **<characteristic>** XML 属性标签内。一个特性可以同时包含多个访问属性，例如，它可以是可读、可写或两者都可以。每个访问属性都可以具有不同的权限等级（例如，加密或验证）。

下表列出了可能的访问属性。表中的每行定义了 **<properties>** 属性下的新属性。

Attribute	Description
read	Characteristic can be read by a remote device. **Values:** **true**: Characteristic can be read **false**: Characteristic cannot be read **Default**: false
write	Characteristic can be written by a remote device **Values:** **true**: Characteristic can be written **false**: Characteristic cannot be written **Default**: false
write_no_response	Characteristic can be written by a remote device. Write without response is not acknowledged over the Attribute Protocol. **Values:** **true**: Characteristic can be written **false**: Characteristic cannot be written **Default**: false
notify	Characteristic has the notify property and characteristic value changes are notified over the Attribute Protocol. Notifications are not acknowledged over the Attribute Protocol. **Values:** **true**: Characteristic has notify property. **false**: Characteristic does not have notify property. **Default**: false **Note**: The *notify* attribute is stored in the SIG defined Client Characteristic Configuration Descriptor (a descriptor with the UUID 0x2902, which will be autogenerated when notifications are enabled). If you manually add a CCCD to the characteristic, the descriptor’s value will overwrite this setting.
indicate	Characteristic has the indicate property and characteristic value changes are indicated over the Attribute Protocol. Indications are acknowledged over the Attribute Protocol. **Values:** **true**: Characteristic has indicate property. **false**: Characteristic does not have indicate property. **Default**: false **Note**: The *indicate* attribute is stored in the SIG defined Client Characteristic Configuration Descriptor (a descriptor with the UUID 0x2902, which will be autogenerated when indications are enabled). If you manually add a CCCD to the characteristic, the descriptor’s value will overwrite this setting.

Attribute	Description
reliable_write	Allows using a reliable write procedure to modify an attribute; this is just a hint to a GATT client. The Bluetooth stack always allows the use of reliable writes to modify attributes. **Values:** **true**: Reliable write enabled. **false**: Reliable write disenabled. **Default**: false

下表所述为权限级别或安全要求。任何安全要求都可以分配给任何访问属性。

Attribute	Description
authenticated	Accessing the characteristic value requires an authentication. To access the characteristic with this property the remote device has to be bonded using MITM protection and the connection must be also encrypted. **Values:** **true**: Authentication is required **false**: Authentication is not required **Default**: false
encrypted	Accessing the characteristic value requires an encrypted link. Devices with iOS 9.1 or higher must also be bonded at least with Just Works pairing. **Values:** **true**: Encryption is required **false**: Encryption is not required **Default**: false
bonded	Accessing the characteristic value requires an encrypted link. Devices must also be bonded at least with Just Works pairing. **Values:** **true**: Bonding and encryption are required **false**: Bonding is not required **Default**: false

示例：含*常量*和*读取*属性的设备名称特性。

```
<!-Device Name-->
<characteristic const = true uuid="2a00">
<properties>
    <read authenticated="false" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

示例：含*读取*和*写入*属性的设备名称特性，允许远程设备修改特性的值。

```
<!-Device Name-->
<characteristic uuid="2a00">
<properties>
    <read authenticated="false" bonded="false" encrypted="false"/>
        <write authenticated="false" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

示例：含*通知*属性的心率测量特性。

```
<!-Heart Rate Measurement -->
<characteristic uuid="180D">
<properties>
    <notify authenticated="false" bonded="false" encrypted="false"/>
```

```
</properties>
</characteristic>
```

示例：含 *加密读取*属性的特性。

```
<!-Device Name-->
<characteristic uuid="1234">
<properties>
        <read authenticated="false" bonded="false" encrypted="true"/>
</properties>
</characteristic>
```

示例：含 *验证写入*属性的特性。

```
<!-Device Name-->
<characteristic uuid="1234">
<properties>
    <write authenticated="true" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

示例：含 *验证指示*属性的特性。

```
<!-Descriptor value changed -->
<characteristic uuid="2A7D">
<properties>
    <indicate authenticated="true" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

2.5.3 <value>

特性的数据类型和长度使用 XML 的 **<value>** 属性及其参数进行定义，并且必须用在 **<characteristic>** XML 属性标签内。

下表中给出了可用于定义相关值的参数。

Parameter	Description
length	Defines a fixed length for the characteristic or the maximum length if ***variable_length*** is true. If length is not defined and there is a value (e.g. data exists inside <value></value>), then the value length is used to define the length. If both ***length*** and value are defined, then the following rules apply: 1. If ***variable_length*** is false and ***length*** is bigger than the value's length, then the value will be padded with 0's at the end to match the attribute's ***length***. 2. If ***length*** is smaller than the value's length, then the value will be clipped to match ***length***, regardless of whether ***variable_length*** is true or false. **Range:** **0 – 255**: Length in bytes if ***type*** is 'hex', 'utf-8', or 'user' **0 – 512:** Length in bytes if **type** is 'user' **Default**: 0
variable_length	Defines that the value is of variable length. The maximum length must also be defined with the ***length*** attribute or by defining a value. If both ***length*** and value are defined, then the rules described in ***length*** apply. **Values:** **true**: Value is of variable length **false**: Value has a fixed length **Default**: false
type	Defines the data type. **Values:** **hex**: Value type is hex **utf-8**: Value is a string **user:** When the characteristic type is marked as type="user", the application is responsible for initializing the characteristic value and also providing it, for example, when read operation occurs. The Bluetooth stack does not initialize the value or automatically provide the value when it is being read. When this is set, the Bluetooth stack generates ***gatt_server_user_read_request*** or ***gatt_server_user_write_request***, which must be handled by the application. **Default**: utf-8

示例：含*通知*属性的心率测量特性，固定长度为两个 (2) 字节。

```
<!-Heart Rate Measurement -->
<characteristic uuid="180D">
<value length="2" type="hex" variable_length = "false"/>
<properties>
<notify authenticated="false" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

示例：供应商特定特性长度可变，最大长度为 20 字节。

```
<!-My proprietary data -->
<characteristic uuid="59cd69c0-2043-11e5-a717-0002a5d5c51b" id="mydata">
<value variable_length="true" length="20" type="hex" />
<properties>
    <notify authenticated="false" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

示例: 在 <value> 标签中输入实际值，也可以定义特性的值和参数。

```
<characteristic const="true" id="device_name" name="Device Name" sourceId="org.bluetooth.characteristic.gap.device_name" uuid="2A00">
<value length="17" type="utf-8" variable_length="false">EFR32 BGM111</value>
<properties>
        <read authenticated="false" bonded="false" encrypted="false"/>
</properties>
</characteristic>
```

在上述示例中，该标签的值为“**EFR32 BGM111**”，长度为 17 字节。

示例：使用**大于**值长度的长度来定义长度和值。

```
<!-- Device name -->
<characteristic uuid="2a00">
     <properties read="true" />
     <value type="hex" length="4" variable_length="false">0102</value>
</characteristic>
```

在上述示例中，由于该***长度***大于该值的长度，该值用 0 填充，该标签的值将为“**01020000**”。

示例：使用**小于**值长度的长度来定义长度和值。

```
<!-- Device name -->
<characteristic uuid="2a00">
     <properties read="true" />
     <value type="hex" length="2" variable_length="false">01020304</value>
</characteristic>
```

在上述示例中，由于该***长度***小于该值的长度，为与***长度***匹配，因此减少了该值的长度，该标签的值将为“**0102**”。

2.5.4 <descriptor>

XML 元素 <descriptor> 可用于定义通用特性描述符。

描述符属性由 <properties> 元素定义且只允许读和/或写访问。值由 <value> 元素定义，与特性值的定义方式相同。

注意：如果您手动添加客户端特性配置描述符（UUID：0x2902），其值会覆盖特性的 *notify/indicate* 属性 。如果没有手动添加客户端特性配置描述符，且特性启用了通知或指示， 则系统会自动生成客户端特性配置描述符。

示例：使用 UUID 2908 类型添加特性描述符。

```
<characteristic uuid="2a4d" id="hid_input">
<properties notify="true" read="true" />
<value length="3" />
      <descriptor const="false" discoverable="true" id="" name="Custom Descriptor" sourceId="" uuid="2908">
        <properties>
          <read authenticated="false" bonded="false" encrypted="false"/>
        </properties>
        <value length="0" type="hex" variable_length="false">00</value>
      </descriptor>
</characteristic>
```

2.5.5 <description>

特性的用户描述值通过 XML 的 <description> 属性定义，并且必须用在 <characteristic> XML 属性标签内。

特征用户描述是可选值。对远程设备可见，且能发挥一定作用。例如，简洁描述应用的用户界面中显示的特性。

示例：常量字符串“心率测量”

```
<characteristic uuid="2a37">
<properties>
    <notify authenticated="false" bonded="false" encrypted="false"/>
</properties>
<description> Heart Rate Measurement </description>
</characteristic>
```

Properties 元素可用于支持远程修改属性。

示例：允许远程读取，但需进行绑定才能写入

```
<characteristic uuid="2a37"> <properties> <read authenticated="false" bonded="false" encrypted="true"/> <write authenticated="false" bonded="true" encrypted="false"/> </properties> </characteristic>
```

Note: 如果描述信息可写，则 GATT 解析器会自动添加扩展属性，并将 *writable_auxiliaries* 位设置为兼容 Bluetooth。

2.5.6 <aggregate>

XML 元素 <aggregate> 通过自动将 ID 转换为属性句柄来支持创建聚合特征格式描述符。

属性 ID 应引用特征表示格式描述符。

示例：添加特征集合

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

2.6 GATT 示例

示例：一例设备名称和外观特性为常量、含*读取*属性的完整 GAP 服务。

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

<!--Generic Access-->
<service advertise="false" name="Generic Access" requirement="mandatory" sourceId="org.bluetooth.service.generic_access" type="primary" uuid="1800">
    <informativeText>Abstract: The generic_access service contains generic information about the device. All available
       Characteristics are readonly. </informativeText>
    <!--Device Name-->
   <characteristic const="true" id="device_name" name="Device Name" sourceId="org.bluetooth.characteristic.gap.device_name"
     uuid="2A00">
      <informativeText/>
      <value length="17" type="utf-8" variable_length="false">EFR32 BGM111</value>
      <properties>
        <read authenticated="false" bonded="false" encrypted="false"/>
      </properties>
    </characteristic>

    <!--Appearance-->
    <characteristic const="true" name="Appearance" sourceId="org.bluetooth.characteristic.gap.appearance" uuid="2A01">
      <informativeText>Abstract: The external appearance of this device. The values are composed of a category (10-bits) and sub-categories (6-bits).
</informativeText>
      <value length="2" type="hex" variable_length="false">0000</value>
      <properties>
        <read authenticated="false" bonded="false" encrypted="false"/>
      </properties>
    </characteristic>
  </service>
</gatt>
```

示例：链路丢失和即时报警服务。

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

<gatt>
<!--Link Loss-->
   <service advertise="false" id="link_loss" name="Link Loss" requirement="mandatory" sourceId="org.bluetooth.service.link_loss" type="primary"
uuid="1803">
    <!--Alert Level-->
    <characteristic const="false" id="alert_level" name="Alert Level" sourceId="org.bluetooth.characteristic.alert_level" uuid="2A06">
      <value length="1" type="hex" variable_length="false"/>
      <properties>
        <read authenticated="false" bonded="false" encrypted="false"/>
        <write authenticated="false" bonded="false" encrypted="false"/>
      </properties>
    </characteristic>
  </service>

  <!--Immediate Alert-->
   <service advertise="false" id="immediate_alert" name="Immediate Alert" requirement="mandatory" sourceId="org.bluetooth.service.immediate_alert"
type="primary" uuid="1802">
    <!--Alert Level-->
    <characteristic const="false" id="alert_level" name="Alert Level" sourceId="org.bluetooth.characteristic.alert_level" uuid="2A06">
      <value length="1" type="hex" variable_length="false"/>
      <properties>
        <write_no_response authenticated="false" bonded="false" encrypted="false"/>
      </properties>
    </characteristic>
  </service>

</gatt>
```

示例：拥有多种功能的 GATT 数据库

```
<gatt db_name="light_gattdb" out="gatt_db.c" header="gatt_db.h" generic_attribute_service="true">
  <capabilities_declare>
    <capability enable="true">cap_light</capability>
    <capability enable="false">cap_color</capability>
  </capabilities_declare>

  <!--Light Service-->
   <service advertise="false" id="light_service" name="Light Service" requirement="mandatory" sourceId="" type="primary" uuid="257f993d-756e-baa6-
e69c-8b101e4e6b3f">
    <informativeText>Info about custom service</informativeText>
    <capabilities>
      <capability>cap_light</capability>
      <capability>cap_color</capability>
    </capabilities>

    <!--Ligth Control-->
    <characteristic const="false" id="light_control" name="Ligth Control" sourceId="" uuid="85e82a1c-8423-610b-9fea-5ad999445231">
      <description>User description</description>
      <informativeText/>
      <capabilities>
        <capability>cap_light</capability>
      </capabilities>
      <value length="0" type="user" variable_length="false"/>
      <properties>
        <read authenticated="false" bonded="false" encrypted="false"/>
        <write authenticated="false" bonded="false" encrypted="false"/>
        <write_no_response authenticated="false" bonded="false" encrypted="false"/>
        <reliable_write authenticated="false" bonded="false" encrypted="false"/>
        <indicate authenticated="false" bonded="false" encrypted="false"/>
        <notify authenticated="false" bonded="false" encrypted="false"/>
      </properties>
    </characteristic>

    <!--Color control-->
    <characteristic const="false" id="color_control" name="Color control" sourceId="" uuid="16b90591-c54a-e7c9-413e-a82748a1e783">
      <informativeText/>
      <capabilities>
        <capability>cap_color</capability>
      </capabilities>
      <value length="0" type="user" variable_length="false"/>
      <properties>
        <read authenticated="false" bonded="false" encrypted="false"/>
        <write authenticated="false" bonded="false" encrypted="false"/>
      </properties>
    </characteristic>
  </service>
</gatt>
```

- 如果启用 cap_light 和 cap_color 功能，将显示全部 light_service。
- 如果禁用 cap_light 功能，将隐藏 light_control 特性。
- 如果禁用 cap_color 功能，将隐藏 color_control 特性。

3. 生成代码文件

本部分内容讲述如何使用 Simplicity Studio 或 `bgbuild` Python 脚本和您的 IDE 从 gatt_configuration.btconf XML 文件生成 gatt_db.c/h 代码文件。

3.1 使用 Simplicity Studio

在 Simplicity Studio 中使用 GATT Configurator 时，每次保存配置都会自动生成代码文件 (gatt_db.c/gatt_db.h)。

3.2 使用 bgbuild

可通过 SDK 中提供的 `bgbuild` Python 脚本，独立于 IDE 生成代码文件：

```
$GSDK_PATH\protocol\bluetooth\bin\gatt\bgbuild.py
```

`$GSDK_PATH` 是所选 GSDK 的安装目录，例如 ~/SimplicityStudio/SDKs/gecko_sdk/。

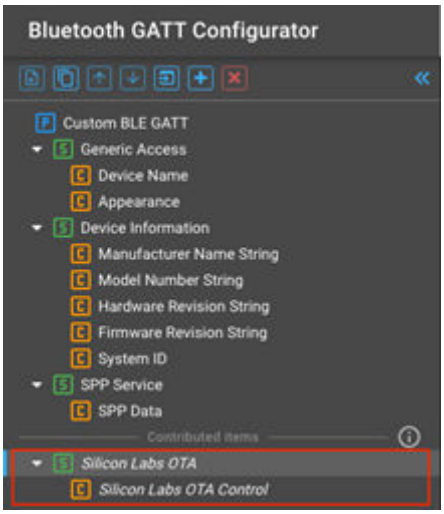
该脚本需要通过调用 `pip install jinja2` 安装 Python 3 和 Jinja2 包。该脚本只能解析由 Simplicity Studio 5/SDK v4.x 创建的或根据此用户指南手动编写的 GATT 配置。GATT Configurator 可以导入旧版 gatt.xml 格式。

必需参数：
- GATT XML 文件的路径，或用于查找 XML 文件的目录。用“；”分隔输入

可选参数：
- -h，--help：显示帮助消息
- -o OUTDIR，--outdir OUTDIR：输出目录，将在此目录中生成文件（Simplicity Studio 5 将源生成到 autogen 文件夹中。在项目结构的其他地方生成它们可能会导致潜在的冲突。）

```
python $GSDK_PATH/protocol/bluetooth/bin/gatt/bgbuild.py ~/SimplicityStudio/v5_workspace/btmesh_soc_empty/config/btconf/gatt_configuration.btconf -o ~/SimplicityStudio/v5_workspace/btmesh_soc_empty/autogen/
```

在 Bluetooth GATT Configurator 中，GATT 可以包含**贡献项目** 部分。这些是在用户界面不可修改的其他 XML 文件，但 Simplicity Studio 中的源生成将包括它们。



要使用 `bgbuild` 生成相同的结果，您必须包含整个 `config/btconf` 文件夹。

```
python $GSDK_PATH/protocol/bluetooth/bin/gatt/bgbuild.py ~/SimplicityStudio/v5_workspace/btmesh_soc_empty/config/btconf/ -o ~/SimplicityStudio/v5_workspace/btmesh_soc_empty/autogen/
```

4. 版本历史

修订版 2.8

2023 年 10 月

- 仅在日语翻译的第 3.2 使用 bgbuild 部分，将“pip install jinja22”更正为“pip install jinja2”。
- 在第 3.2 使用 bgbuild 部分中，更新了流程以手动生成代码文件。

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.
Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect , n-Link, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

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

www.silabs.com