This article describes the steps required to create a GATT based profile to enable a use case. Such a Profile consists of Roles and which Services each role implements. A Service is comprised of one or more Characteristics. A fundamental premise for Services is that they follow the client server design pattern. These building blocks are combined to define the use case in a GATT based profile which will contain a description of the use case, the roles required to support the use case, the one or more services exposed which enable the use case as well as the .xml definition of the server required to implement the use case.

For illustrative purposes, two examples will be used throughout this paper:

1. Obtaining the temperature in a room
2. Turning lights on and off

As each example is developed, concepts such as Characteristics, Services, Profiles and reuse will be expanded upon. Tools provided by the Bluetooth SIG which aid in the creation of GATT based profiles will also be introduced via the examples. After reading and understanding these concepts, you will have all the tools necessary to create GATT based profiles.

Articulating the Use Case

The first step in creating a profile is clearly defining what will be enabled and what won't. While this step seems trivial, properly defining the use case is paramount in completing the profile. Leaving the use case ambiguous often makes design decisions more complex and leads to feature creep and poor design.

Example: Obtaining the temperature in a room

A device, the thermometer, will expose the current ambient room temperature where the temperature sensor is located within the device and reports in 100^{th} of degrees Celsius, such that another device, the monitor, can read the current temperature it is measuring. Whether the device takes a new temperature reading each time the temperature is requested or stores the last ready temperature which is updated on a regular bases is an implementation issue and outside the scope of this example. As such, for this example, it is assumed the data can be returned from the sensor immediately. The format of the temperature data returned is a signed 8 bit integer.

Example: Turning the lights on and off

In the simplest form, turning the lights on and off is obtained by flipping a switch on the wall. Traditionally in this case, the wall switch physically opens and closes the electrical connection enabling the light to turn off and on. However, things quickly get more complex when 2-way switches are introduced. This is typically handled by using two physical switches with a traveler wire between them such that toggling either switch would turn the light on or off whichever is the opposite of the current state of the lights. The complexity rises substantially as 3-way, 4-way or higher control is implemented with physical switches. Furthermore, incandescent bulbs can be

dimmed, while fluorescent bulbs cannot. In many situations, switches controlling incandescent lights consist of a slider or wheel that set the brightness of the lights.

Modern lightening control implements a controller that physically enables or disables the current to one or more lights and allows one or more switches to alter the on/off state as well as the brightness of the controller. For simplicity, a controller may also contain a switch locally, but that isn't required. This topology is depicted in **Error! Reference source not found.**~~Figure 1~~.

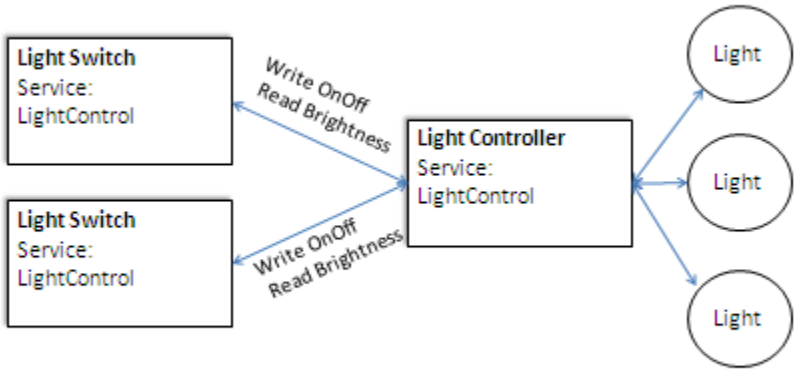

Figure 1 Client and Server roles for Light Switch and Controllers

Defining Characteristics

Now that the use case is clearly articulated, the next step for designing a GATT based profile is to answer the question: *what Characteristics are required for enabling this use case*. A Characteristic is a piece of data, a Value, with metadata to describe the data. The Value can be an atomic element, a Field, such as weight or pressure or a collection of related Fields such as a year, month and day combined into a date. Every GATT based profile exposes at least one Service which exposes at least one Characteristic; some will have many more. It is therefore critically important to first understand what Characteristic information is necessary to implement the use case.

To define a Characteristic, the GATT profile has defined data types based off the International System of Units (SI Units) and their derivatives which are used when defining Fields.

To facilitate defining Characteristics as well as the other portions of a GATT based profile an XML language which defines the capabilities contained in the GATT profile has been defined. Once the GATT based profile is defined in this language, it can be compiled into the structure required by the device implementing the profile. This language will be used throughout this paper in the examples.

There is a library of Characteristics which have been developed and published by the Bluetooth SIG. Before creating a new Characteristic, the published ones should be reviewed to see if what you are developing has already been defined. If so, use that Characteristic rather than creating a new one.

A Characteristic is a fundamental building block used in the creation of GATT based profiles. However, they are not allowed to be instantiated in a device without being enclosed in a Service which is defined in the next step.

Example: Obtaining the temperature in a room

In the use case above, a single temperature was defined which exposes its Value in 100th of degrees Celsius via a signed 8 bit integer. Thus, a single Characteristic exposing the temperature along with the associated metadata will be sufficient. The XML to define this Characteristic is provided in Listing 1.

```
<!--
    The Temperature Characteristic.
    This Characteristic defines a single Field representing Temperature
    in 100th of degrees Celsius.
-->
<Characteristic name=”Temperature Celsius Example”
xsi:noNamespaceSchemaLocation="http://schemas.bluetooth.org/Documents/charact
eristic.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
uuid="0x0000000"
type="org.bluetooth.characteristic.TemperatureExample">

      <InformativeText>
              <Abstract>
                    A 8 bit signed temperature
            </Abstract>
      </InformativeText>

      <Value>
        <Field Name="Temperature Example">
            <Requirement>Mandatory</Requirement>
            <Format>SInt8</Format>
            <Exponent>-2</Exponent>
        </Field>
    </Value>
</Characteristic>
```

Listing 1 Temperature Characteristic (NOTE: based on a hypothetical example)

A Characteristic contains a single Value. However, the Value may contain multiple Fields. In the temperature example, only a single Field is required.

The Value section defines the data contained in the Characteristic. This is the information which is returned from the server when read or supplied by the client when written. When multiple Fields are defined, they are returned as one attribute with the Fields concatenated together. The

Type is used to define the Characteristic type. The name is defined in reverse domain order. For Characteristics published by the Bluetooth SIG the type will always begin with "org.bluetooth.characteristic." When the Characteristic hasn't been published by the Bluetooth SIG, the reverse domain name of the company using this Characteristic in their product must be used. For example, if a company owns the domain "mycompany.com" the Type must begin with "com.mycompany.characteristic" and then be followed by a companywide unique type name for the Characteristic defined by that company. The UUID attribute specifies the globally unique ID of the Characteristic. For Characteristics published by the Bluetooth SIG, this value will be assigned by the SIG. For Characteristics created by a company, this ID must be supplied.

Within the Value section itself, one or more Fields are defined. Each Field consists of a name which must be unique within that Characteristic as well as the type of the data contained in the Field and optional exponent definition. For Fields with enumerated meanings, that meaning is defined here. The type is selected from one of those defined in the GATT specification. The exponent can be used for integer types to "shift" the decimal place of the Field by multiplying the Field value by 10 raised to the Exponent amount. For example, if the value for the Temperature Field is 5935, multiplying that by ten to the Exponent amount (10^{-2}) yields the value of this Field as 59.35.

Example: Turning the lights on and off

Turning lights on and off could have been represented many ways such as a 0 representing off and a 1 representing on. Or the light status could have been represented by 0 representing off, 100 representing fully on and any number in between the brightness level. Furthermore, a different Characteristic could be used to read the current status of the lights from that used to set that status. However, the use case clearly defines the status of the lights to be represented by a binary value that can be read and written to control the light status with a separate value representing the brightness setting. This allows the lights to be turned on to the previous lighting level without knowing what that level was. As such, the light controller can be represented by a Characteristic containing two Fields. The first Field contains a Boolean representing the On/Off status of the lights and the second contains an unsigned integer representing the brightness setting of the lights. The XML for this Characteristic is show in Listing 2.

```
<!--
     A Characteristic which defines two Fields:
         A boolean status where true represents "on" and false "off"
         An unsigned integer representing the brightness where 0 is none
         and 100 is fully on
-->
<Characteristic
xsi:noNamespaceSchemaLocation="http://schemas.bluetooth.org/Documents/charact
eristic.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
UUID="0x0000000"
Type="org.bluetooth.characteristic.OnOffBrightnessExample">

      <Value>
        <Field Name="OnOff">
            <Format>Boolean</Format>
            <Enumeration>
```

```
                <BooleanFalse>Off</BooleanFalse>
                <BooleanTrue>On</BooleanTrue>
            </Enumeration>
        </Field>
        <Field Name="Brightness">
            <Format>UInt8</Format>
            <Exponent>0</Exponent>
        </Field>
    </Value>
</Characteristic>
```

Listing 2 OnOffBrightness Characteristic

The OnOffBrightness Characteristic contains a concatenation of two Fields into a single Value which is returned whenever the Characteristic is read. The OnOff Fields is a Boolean where false represents "off" and true represents "on." The second Field is Brightness which is represented by an unsigned 8 bit integer.

Defining Services

A Service is a collection of one or more Characteristics intended to work together to perform a function. When describing the Characteristics within a Service, the way in which those Characteristics are utilized is defined. This includes defining Properties as well as Descriptors for each Characteristic. The Property metadata defines how the Characteristic can be accessed and includes such capabilities as readable, writeable, broadcast and if authentication is required. The Descriptors define metadata such as description and presentation information. A Service may contain both mandatory and optional Characteristics.

Example: Obtaining the temperature in a room

The Service definition for reading the room temperature is very simple. It only needs to use the one Temperature Characteristic defined above as mandatory and add the metadata describing how that Characteristic can be utilized in the Service. The Service description is shown in Listing 3.

```
<!--
    A Service containing a single mandatory Temperature Characteristic which
    can be read without authentication, but not written or notified.
-->
<Service BluetoothType="org.bluetooth.service.Temperature"
      BluetoothUUID="0x000000">
    <Characteristics>
        <Include Name="Temperature" Mandatory="true"
            BluetoothType="org.bluetooth.characteristic.Temperature">
            <Properties>
                <Read>true</Read>
                <Write>false</Write>
                <Notify>false</Notify>
                <Authenticated>false</Authenticated>
            </Properties>
            <Descriptors>
```

```
                <UserDescription>The temperature in the
room</UserDescription>
                <Range>
                    <Minimum>-5</Minimum>
                    <Maximum>40</Maximum>
                </Range>
                <PresentationFormat>
                    <Format>
                        <SInt8/>
                    </Format>
                    <Unit>Degrees Celsius</Unit>
                    <Description>The temperature in the room</Description>
                </PresentationFormat>
            </Descriptors>
        </Include>
    </Characteristics>
</Service>
```

Listing 3 Temperature Service

Since a suitable Temperature Characteristic has already been defined, that Characteristic is included in the Service. Then, the properties section defines access to the Temperature Characteristic. A true value implies that access mode is supported and a false that it is not. The default value for each property is false. Thus in the example above, Write, Notify and Authenticated could have been omitted without changing the semantics of how the Temperature Characteristic is utilized in the Service. They were expressly added here to emphasize the fact that this Characteristic is readonly.

The second section of metadata for the included Temperature Characteristic is the Descriptors. While the UserDescription and Range Descriptors are somewhat self explanatory, the PresentationFormat Descriptor isn't so easy to understand. The PresentationFormat Descriptor is used as a hint to the client regarding how the data contained in the Characteristic can be displayed. For a simple Characteristic such as this, the exponent defined in the Characteristic can be used to "shift" the decimal place of the data element. The Description provides text to be displayed with the number. Thus if the value for this Characteristic was 5935, multiplying that by ten to the Exponent amount (10^{-2}) and then adding the Description would yield a display for the current temperature as: "59.35 Degrees Celsius".

Example: Turning the lights on and off

In the previous step, we defined the Characteristic to enable lights to be turned on and off as well as setting a brightness parameter. To enable the desired use case, this Characteristic will be included into a light controller service as shown in Listing 4.

```
<!--
    The LightControl service that has a single Characteristic to turn
    lights on and off as well as control the brightness of the lights.
-->
<Service BluetoothType="org.bluetooth.service.LightControl"
      BluetoothUUID="0x00000">
    <Characteristics>
        <Include Name="Status" Mandatory="true"
```

```
            BluetoothType="org.bluetooth.characteristic.OnOffBrightness">
            <Properties>
                <Read>true</Read>
                <Write>true</Write>
                <Notify>true</Notify>
                <Authenticated>true</Authenticated>
            </Properties>
            <Descriptors>
                <UserDescription>
                  The current status of the lights controlled
                </UserDescription>
            </Descriptors>
        </Include>
    </Characteristics>
</Service>
```

Listing 4 LightControl Service

The LightControl Service includes the previously defined OnOffBrightness Characteristics and defines how this Characteristic may be accessed.

The properties are defined such that this Characteristic is to be both written and read while requiring authentication. When the value changes, interested Clients are notified of the change. Upon Client connection, they are immediately notified of the current status of the Characteristic.

The Descriptors are used to provide a UserDescription for this service.

Defining Profiles

A Profile is a collection of one or more Services intended to work together to enable a use case. The Profile may contain both mandatory and optional Services. However, a Characteristic which is not part of a Service may not be included in a Profile.

In both the "Obtaining the temperature in a room" and "Turning the lights on and off" examples, the use case is fully defined by a single Service.

Example: Obtaining the temperature in a room

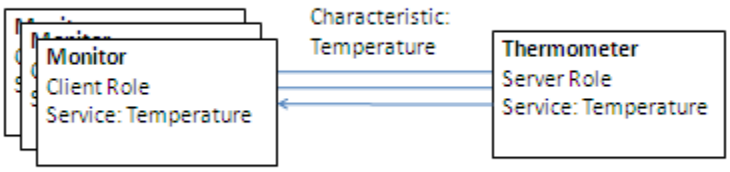

Figure 2 Client and Server roles for Light Switch and Controllers

In this example, the Temperature Profile is defined to contain two roles: the Thermometer and the Monitor. The Profile supports one and only one Thermometer Role, but there may be multiple Monitor Roles. The Thermometer Role implements a single Service; that being the Temperature Service. The Profile description is shown in Listing 5.

```
<!--
    The Temperature Profile defines two roles.
    The first, a Thermometer, implements the Temperature Service.
    The second, any number of Monitors, doesn't implement any services.
-->
<Profile BluetoothType="org.bluetooth.profile.Temperature"
      BluetoothUUID="0x000000">
      <Roles>
            <Role Name="Thermometer">
                  <Instances>
                      <ExactlyN>1</ExactlyN>
                  </Instances>
                  <Services>
                        <Service Name="Temperature"

      BluetoothType="org.bluetooth.service.Temperature"
                              BluetoothUUID="0x0000" Mandatory="true" />
                  </Services>
            </Role>
            <Role Name="Monitor">
                  <Instances>
                      <OneOrMore/>
                  </Instances>
                  <Services />
            </Role>
      </Roles>
</Profile>
```

Listing 5 Temperature Profile

Example: Turning the lights on and off

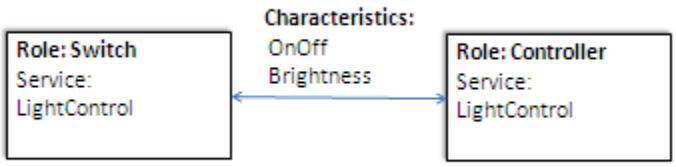

Figure 3 Client and Server roles for Light Switch and Controllers

Similar to the Temperature Profile, the Light Control Profile defines two roles: a Controller and one or more Switch. The Controller implements a single Service. The Profile description is shown in Listing 6.

```
<!--
    The Light Control Profile defines two roles.
    The first, a Controller, implements the Light Control Service.
    The second, any number of Switch, doesn't implement any services.
-->
<Profile BluetoothType="org.bluetooth.profile.LightControl"
      BluetoothUUID="0x000000">
    <Roles>
        <Role Name="Controller">
            <Instances>
                <ExactlyN>1</ExactlyN>
            </Instances>
            <Services>
                <Service Name="Light Control"
                        BluetoothType="org.bluetooth.service.LightControl"
                        BluetoothUUID="0x0000" Mandatory="true"/>
            </Services>
        </Role>
        <Role Name="Switch">
            <Instances>
                <OneOrMore/>
            </Instances>
            <Services/>
        </Role>
    </Roles>
</Profile>
```

Listing 6 Temperature Profile