Bluetooth® low energy profile development

Bluegiga Technologies

Topics

- What is a profile?
- *Bluetooth* low energy profile
- How to create a *Bluetooth* low energy profiles
- Making *Bluetooth* low energy devices

What is a profile?

Profiles

- **Why they are needed?**
  - To make devices interoperable

- **How profiles are created?**
  - Based on market's needs

- **Who is making them?**
  - Member companies of Bluetooth SIG under working groups
    - Can be a slow process.

  - What if companies or even individuals could make profiles by themselves if market demands for it?

Profiles

- **Profiles in *Bluetooth* BR/EDR**
  - Specifies low level protocol (L2CAP, RFCOMM, SCO)
  - Specified high level protocol (AT-commands, IrOBEX, SBC codec)
  - Proprietary profiles closed (typically running on top of RFCOMM)

- **Profile are essentially a large concept**
  - Involves very detailed and large specifications

    → specification work takes time
  - Implementation and testing slow process

    → adaption of new profiles takes time

Bluetooth low energy profile

Profiles

- ***Bluetooth* low energy profiles**
  - Specifies data structures
  - No definition of protocol needed

    Used over a single protocol (ATT)

- **Profiles require very little specification to be written**
  - Profiles can be described using XML schema

    → Unified structure for interpreting the data specified by profiles

    → Profiles can be downloaded from Internet

  - Proprietary profiles can be adopted/interpreted by others easily

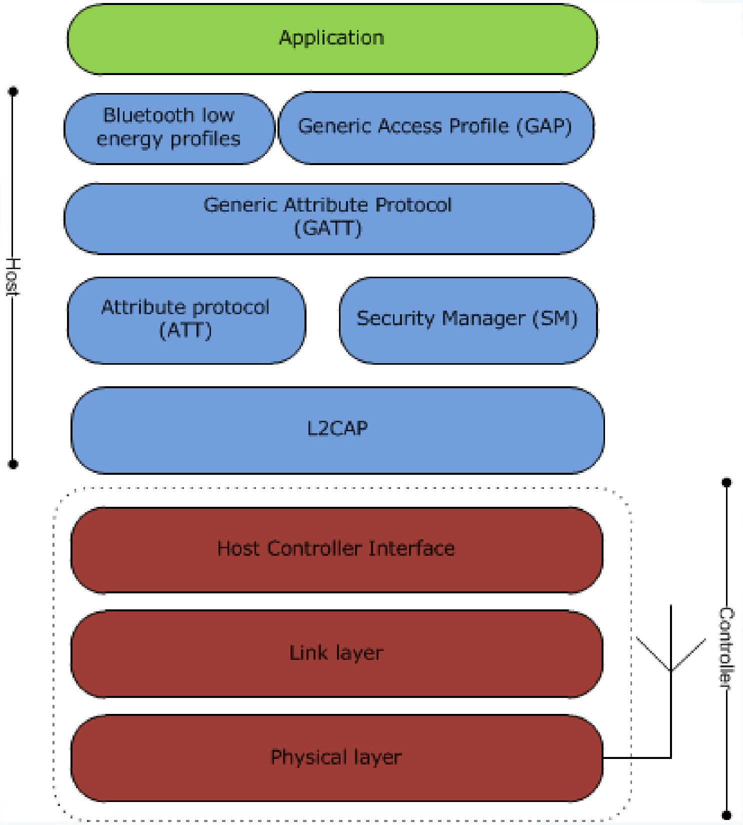
Profiles

- ***Bluetooth* low energy profiles are lightweight**
  - Anyone can create a new profile
  - Small profile footprint makes it usable in applications with memory constraints
- ***Bluetooth* low energy profiles are reusable**
  - No need to rewrite complete profiles
    - Include services from existing profiles
    - Extend with the features you need

How to create *Bluetooth* low energy profiles?

Achitecture

- **All profiles build on top of**
  - L2CAP
    - Segmentation and reassembly of **packets**
  - ATT
    - Ability to read/write **attribute values**
  - GATT
    - Grouping of attributes as **services**


Application
Bluetooth low energy profiles
Generic Access Profile (GAP)
Generic Attribute Protocol (GATT)
Attribute protocol (ATT)
Security Manager (SM)
L2CAP
Host Controller Interface
Link layer
Physical layer
Host
Controller

Attribute Protocol (ATT)

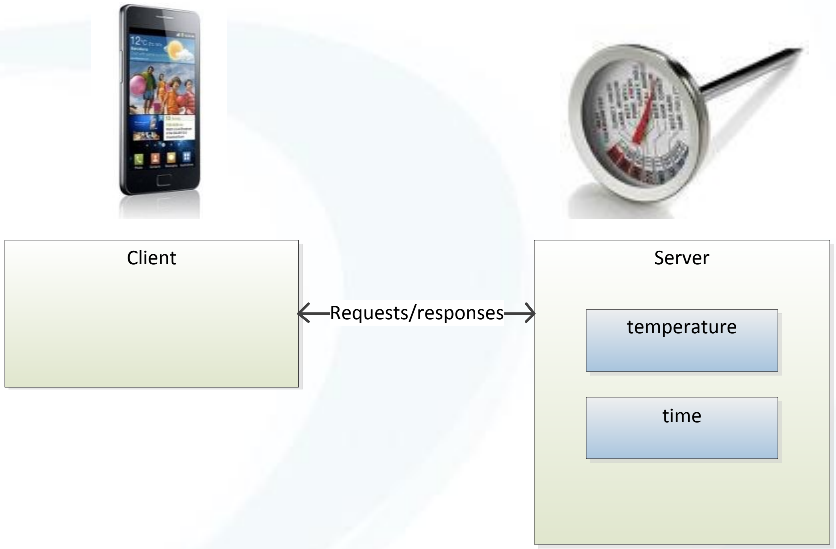
The only protocol used in *Bluetooth* low energy

Uses client server architecture

- servers store data
- clients request data from server
- clients writes data to server

Protocol Methods

- Client to server: Read, write
- Server to client: Notify, indicate



Client

Requests/responses

Server

temperature

time

Attribute Protocol

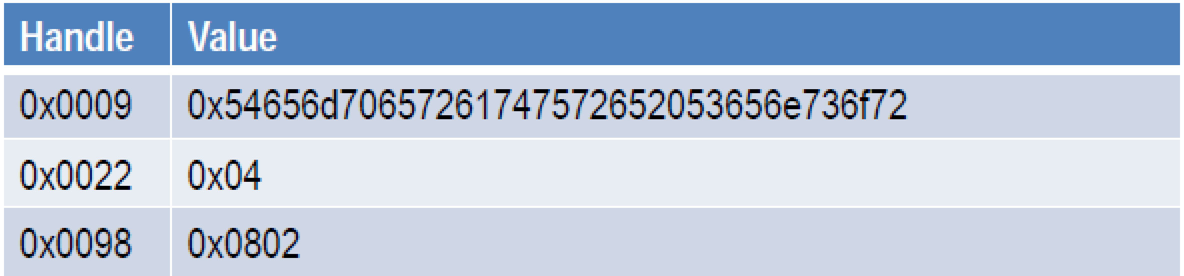
The data is exposed as attributes

- Attributes have values
- 0 to 512 octets
- Fixed or variable length

Attributes have handles

- Used to address individual attributes

Read 0x0022 -> 0x04



Handle	Value
0x0009	0x54656d70657261747572652053656e736f72
0x0022	0x04
0x0098	0x0802

10/6/11

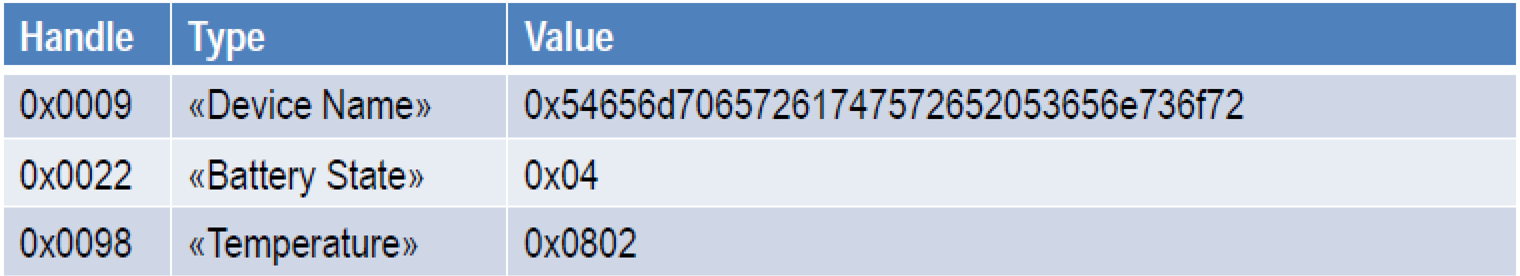
Attribute Protocol

Attributes have a type

- Identified by UUIDs
- UUIDs are 16 bit or 128 bit

Types are defined is specifications

- Characteristics specifications
- Generic Access Profile
- Generic Attribute Profile



Handle	Type	Value
0x0009	«Device Name»	0x54656d70657261747572652053656e736f72
0x0022	«Battery State»	0x04
0x0098	«Temperature»	0x0802

0x54656d70657261747572652053656e736f72 = “Temperature Sensor”

Attribute Protocol

Attributes have permissions:

- Readable / not readable
- Writeable / not writeable
- Readable & writeable / not readable & not writeable

Attribute values may require:

- Authentication to read / write
- Authorization to read / write
- Encryption / pairing to read / write

These are defined in *Bluetooth* LE profile specifications

Attribute Protocol

Attribute Protocol is stateless

Transactions:

- Request -> Response
- Command
- Notification
- Indication -> Confirmation

Attribute Protocol is sequential

- Only one request at a time

Simple!

Attribute Protocol

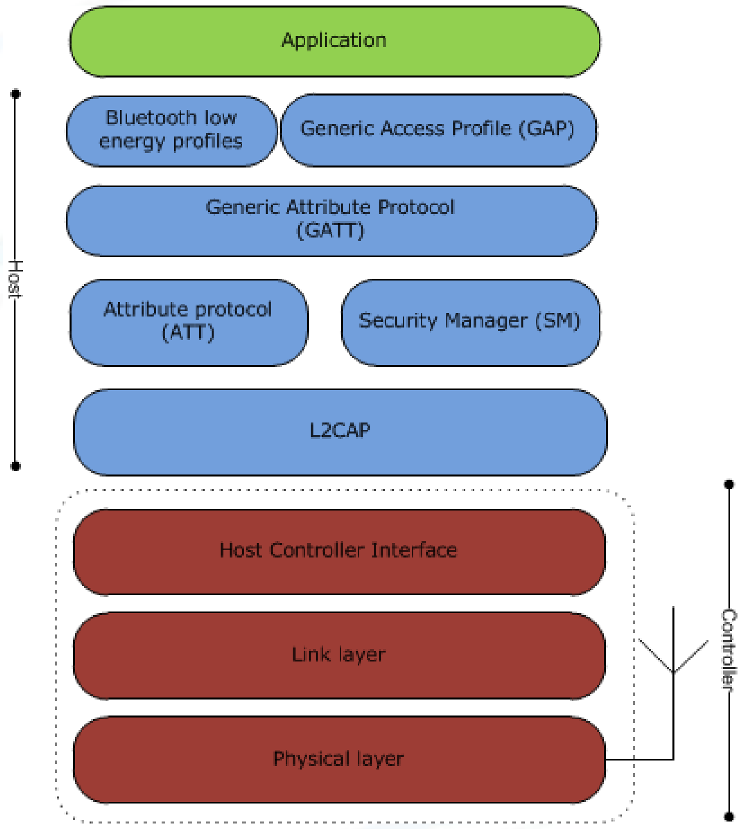
- **Attribute operations: read**
  - Client requests data when it needs it
  - Client polls server for attribute value
  - – This may be inefficient if data doesn’t change often
  - – Shouldn’t be used for frequently changing data that you are monitoring
- **Attribute operations: write**
  - Client can set attributes to configure a server
  - – E.g. set the room temperature to 22ºC

Attribute Protocol

- **Attribute operations: notify**
  - Server sends the data when it changes

- **Attribute operations: indicate**
  - Server sends the data when it changes
  - Client confirms that is has received the data

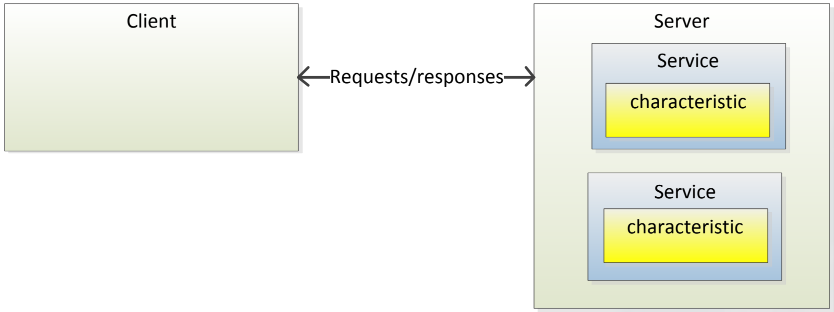
Generic Attribute Profile (GATT)


Application
Bluetooth low energy profiles
Generic Access Profile (GAP)
Generic Attribute Protocol (GATT)
Attribute protocol (ATT)
Security Manager (SM)
L2CAP
Host Controller Interface
Link layer
Physical layer
Host
Controller

Generic Attribute Profile

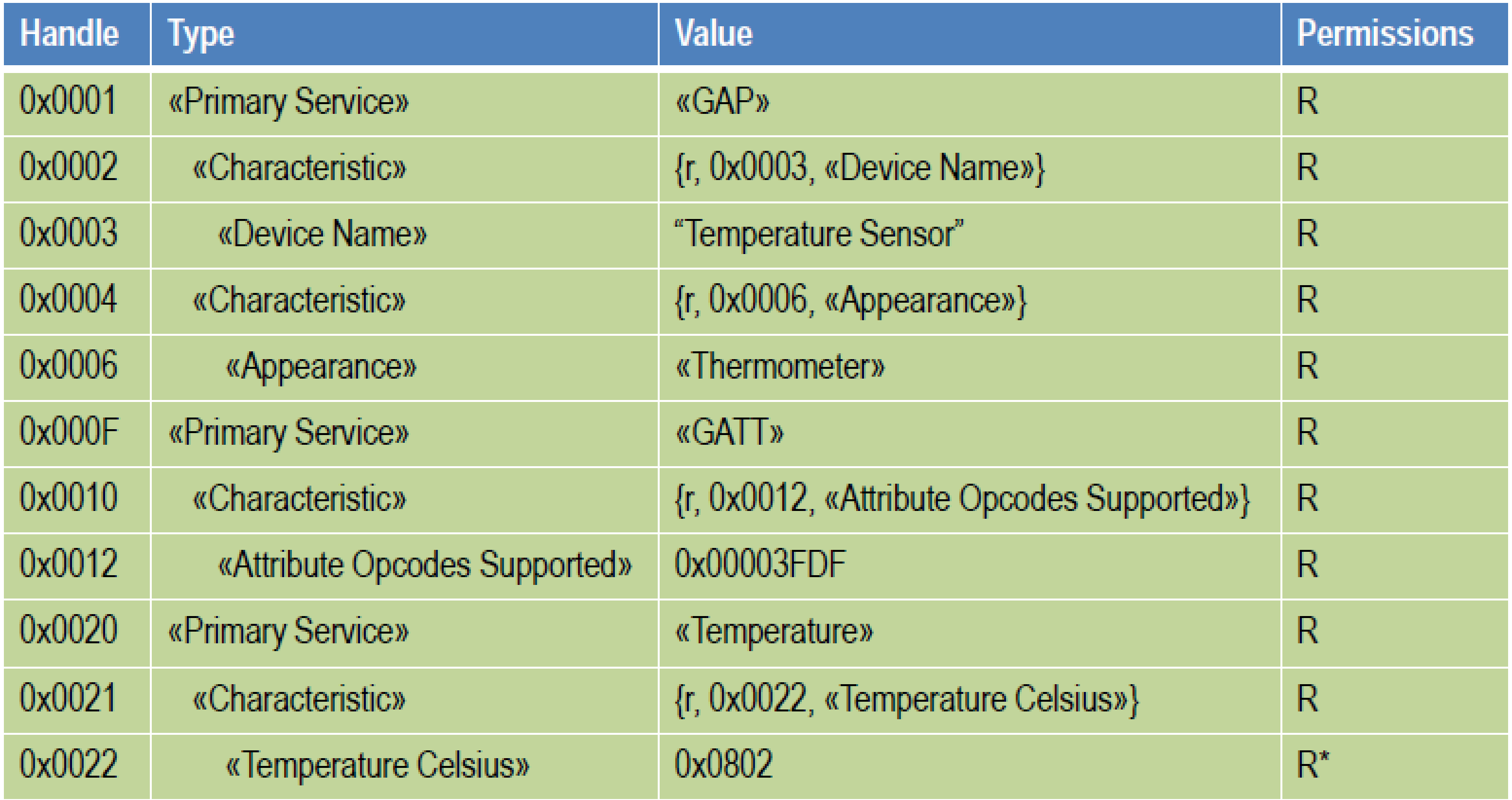
- Attribute Protocol is just a flat structure
  - Profiles require hierarchical structures

- Same client server architecture as in ATT, except:
  - Data is encapsulated in services
  - Data is exposed in characteristics



Generic Attribute Profile

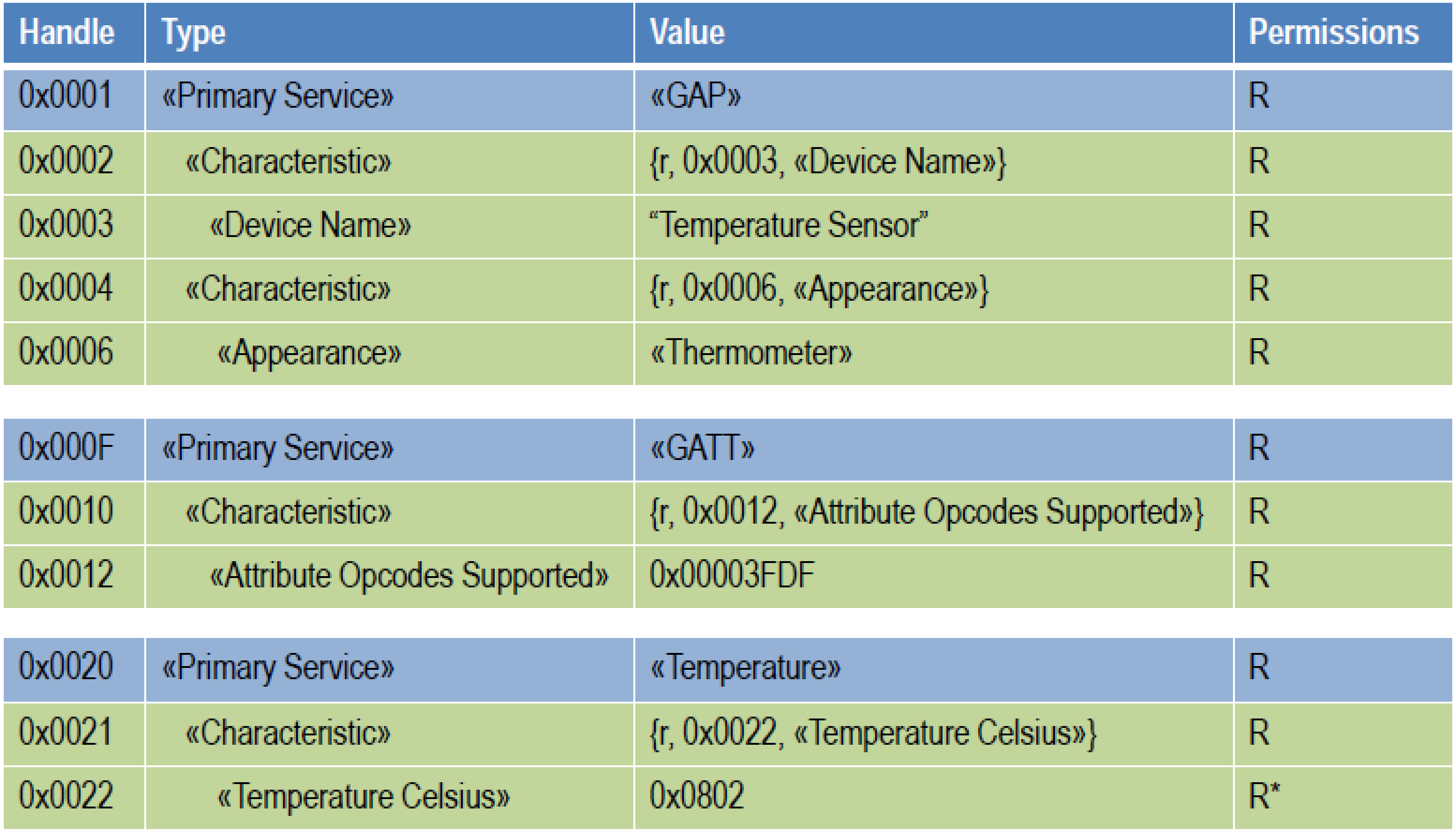
Attributes are flat



Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

Generic Attribute Protocol (GATT)

Grouping gives structure



Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

Services

- **Is a collection of characteristics**

- **Has an UUID**

Generic Access	1800
Device Information	180A
Heart Rate	180D

  - Proprietary services have 128 bit UUID
    Can be generated online : www.uuidgenerator.com

- **Primary Service**
  - Exposes primary usable functionality of this device
  - Can be included by another service

- **Secondary Service**
  - Is subservient to another secondary service or primary service
  - Is only relevant in the context of another service

Characteristic

Characteristic Declaration

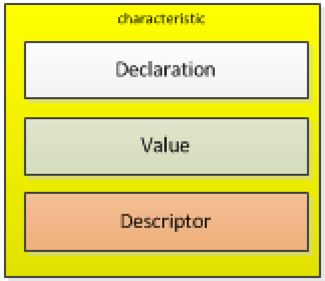
- Describes the properties of *characteristic value* (read, write, indicate etc.), *characteristic value* handle and *characteristic value* type (UUID)

Characteristic Value

- Contains the value of the characteristic.

Characteristic Descriptor(s)

- Provide additional information about the characteristic (characteristic user description, characteristic client configuration, vendor specific information etc.)



Characteristic

- **UUID for each characteristic type**

Device name	2A00
Date time	2A08
Temperature in Celsius	2A1F

  - Proprietary characteristics again have 128-bit UUID

- **You can use already specified characteristics**
  - http://developer.bluetooth.org
  - Makes profile development faster
  - Allows better interoperability

Profile vs Service

Profiles	Services
Network Availability	Network Availability
Proximity	Immediate Alert
Fine Me	Tx Power
Soft Button	Link Loss Alert
Device Power	Generic Control
Battery	Battery
Heart Rate Belt	Time
Weight Scale	Manufacturer
Glucose Meter	Glucose Meter
Light Switch	

Making Bluetooth low energy devices

Making BLE enabled device

- **You need**
  - *Bluetooth* low energy radio
  - *Bluetooth* low energy stack
  - A Profile for your application

- **What if there is no profile that would suit your needs?**
  - Make your own one!

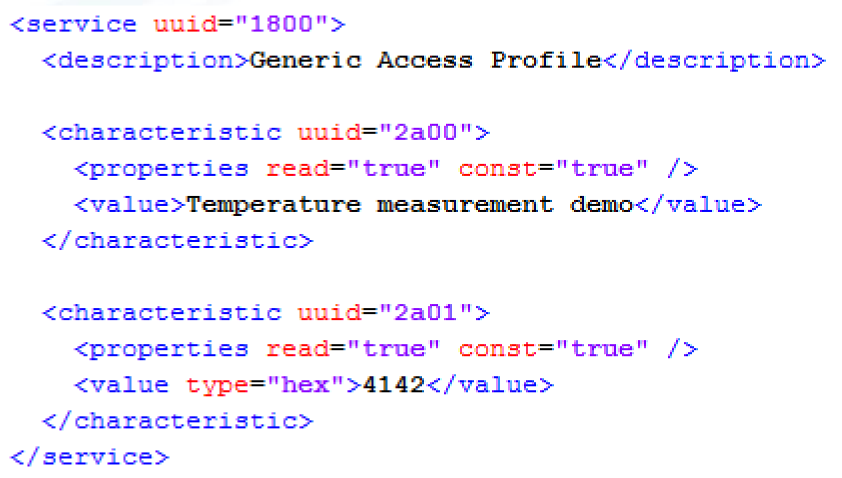
Making your own profile

- **What information do you want to transfer**
  - Sensor readings
  - Key presses
  - Alarms
  - ...
- **Is there an existing service for some of the features?**
  - https://developer.Bluetooth.org/gatt/services
- **Is there an existing characteristic for measurements?**
  - https://developer.Bluetooth.org/gatt/characteristics
- **Streaming or read once?**
  - Indicate or Notify or Read
- **Is security required?**

Thermometer - example

- **Information (GAP) 1800 (Primary service)**
  - Device name 2A00 (Characteristic)
    - Readable, constant
  - Appearance 2A01 (Characteristic)
    - Readable, constant

- **XML description**

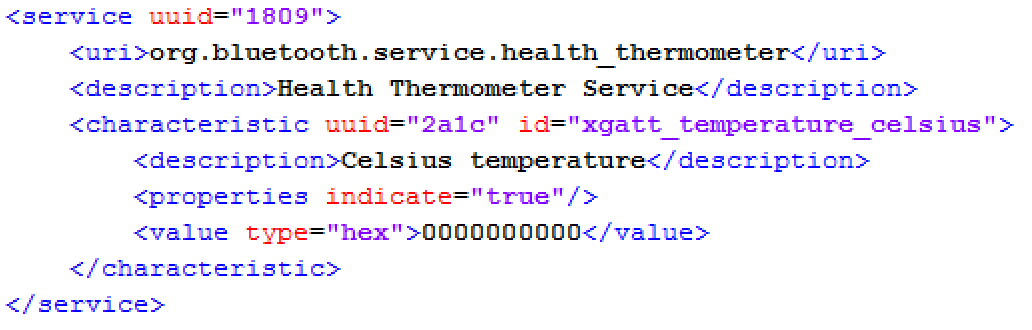


```
<service uuid="1800">
  <description>Generic Access Profile</description>

  <characteristic uuid="2a00">
    <properties read="true" const="true" />
    <value>Temperature measurement demo</value>
  </characteristic>

  <characteristic uuid="2a01">
    <properties read="true" const="true" />
    <value type="hex">4142</value>
  </characteristic>
</service>
```

Thermometer - example

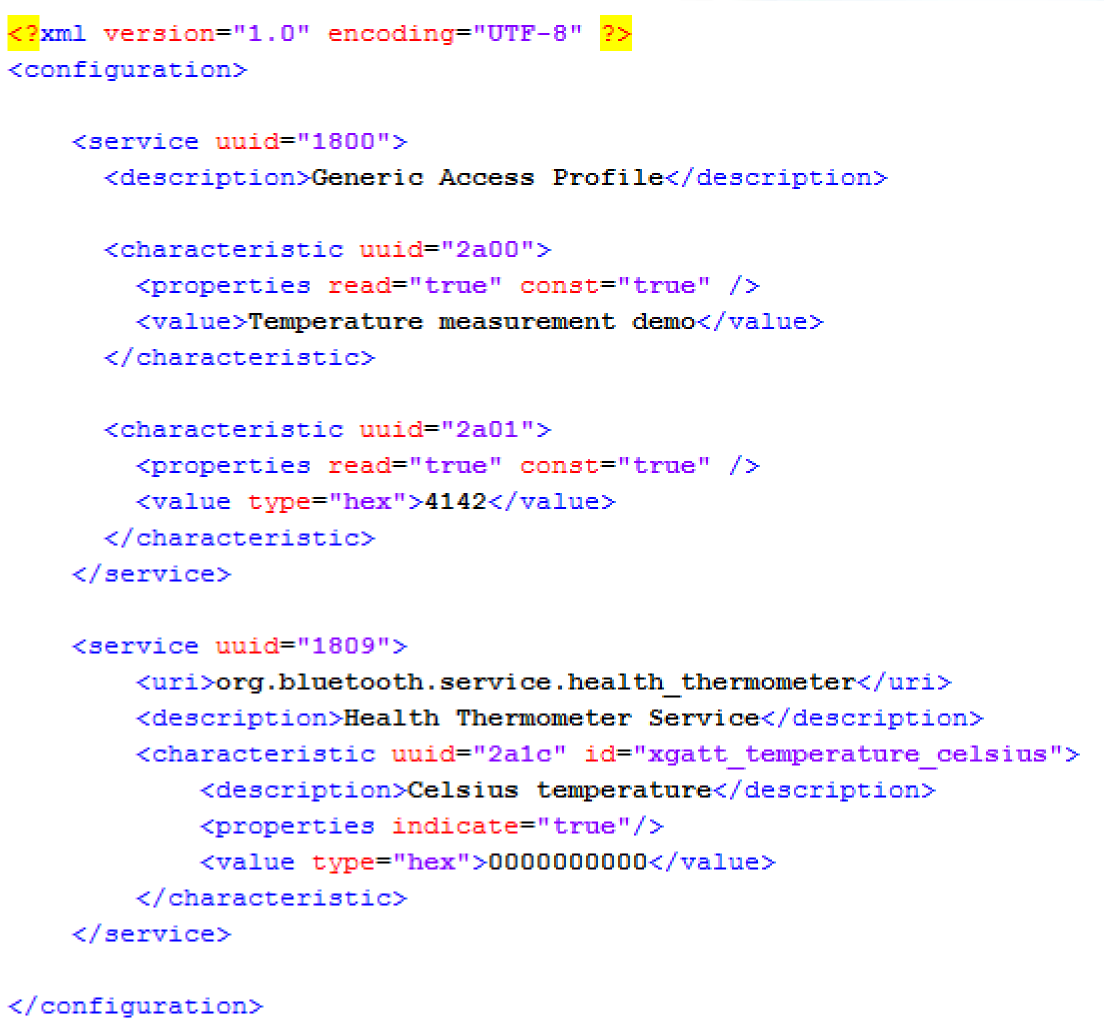
- **Health Thermometer 1809 (Primary service)**
  - Celsius temperature 2a1C (Characteristic)
    - Indicate

- **XML description**



```
<service uuid="1809">
    <uri>org.bluetooth.service.health_thermometer</uri>
    <description>Health Thermometer Service</description>
    <characteristic uuid="2a1c" id="xgatt_temperature_celsius">
        <description>Celsius temperature</description>
        <properties indicate="true"/>
        <value type="hex">0000000000</value>
    </characteristic>
</service>
```

Thermometer XML schema



```
<?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>Temperature measurement demo</value>
      </characteristic>

      <characteristic uuid="2a01">
        <properties read="true" const="true" />
        <value type="hex">4142</value>
      </characteristic>
    </service>

    <service uuid="1809">
        <uri>org.bluetooth.service.health_thermometer</uri>
        <description>Health Thermometer Service</description>
        <characteristic uuid="2a1c" id="xgatt_temperature_celsius">
            <description>Celsius temperature</description>
            <properties indicate="true"/>
            <value type="hex">0000000000</value>
        </characteristic>
    </service>

</configuration>
```

Bluetooth low energy profile toolkit

- **Fast development of BLE profiles on top of GATT**
  - XML based profile description language
  - Compiler
  - API generator

- **Standard or manufacturer specific *Bluetooth* low energy profiles**

- **Profile and application both running inside a Bluegiga BLE112 module**
  - Includes Bluetooth 4.0 single-mode radio
  - A full Bluetooth 4.0 single-mode stack
  - Standardized or proprietary profiles
  - Application functionality – no external MCU needed
  - Full qualifications : Bluetooth, CE, FCC, IC, Telec

bluegiga

Thank you!

www.bluegiga.com