Developing Accessories with *Bluetooth*® Smart

bluegiga

Webinar Rules

Everyone will be **muted** during the webinar

We have dedicated time for **Questions and Answers** in the end of the webinar. You can however post your questions using the GoToMeeting toolbar already during the presentation.

Slides and **webinar recording** will shared after the webinar

Topics

- *Bluetooth*® Smart Intro
- Developing a *Bluetooth* Smart Device
- iOS and Android Considerations
- Questions and Answers

Bluetooth Smart Intro

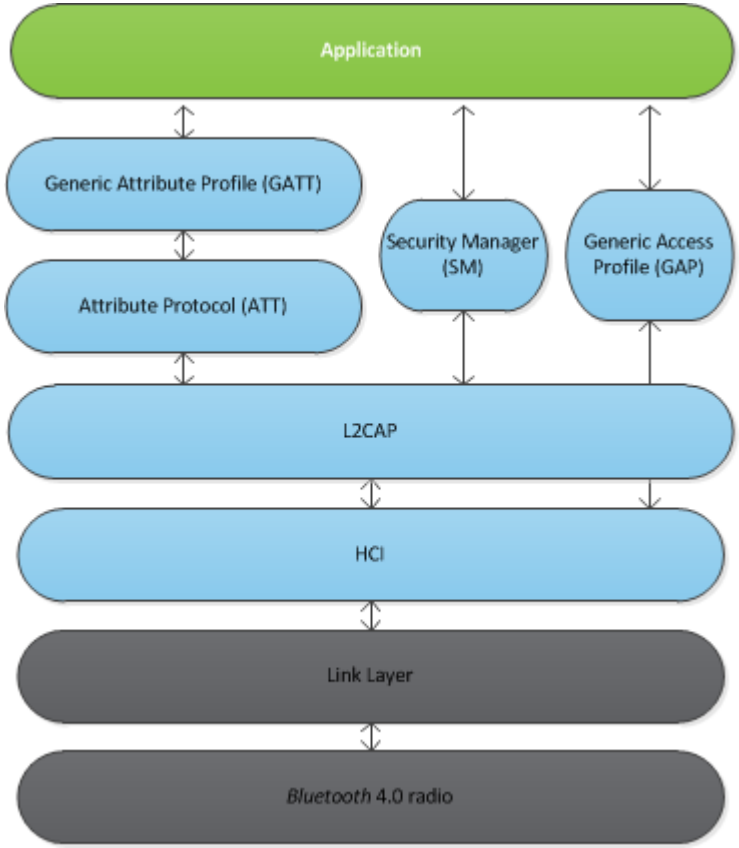
- **What is *Bluetooth* Smart technology?**
  - A new technology - designed almost on a blank sheet of paper
  - Optimized for ultra-low power consumption
  - Enables coin cell battery uses cases
    - < 15mA peak power
    - Average consumption in range of uA
  - Different from classic *Bluetooth* (BR/EDR) technology

Bluetooth Smart Intro

- **Benefits of *Bluetooth* Smart**
  - Ultra low power
  - Low cost
  - Reliable
  - Secure – paring, AES-128 and MITM protection
  - Robust – AFH, retransmissions, 24-bit CRCs and FEC
  - Enables profiles to be developed as Apps – fast deployment
  - Customer specific profiles – no need to wait OS developers
  - Connectivity to Smart phones, tablets, PCs
  - Supported by : iOS, OSX, Linux, Windows 8, Android 4.3

Bluetooth Smart Intro

- **Generic Access Profile –** Device roles, discovery and connections
- **Security Manager –** Authentication, authorization and encryption
- **Generic Attribute Profile (GATT) –** Organization of data into services
- **Attribute protocol (ATT)** – Data access protocol
- **L2CAP –** Provides connection oriented data and multiplexing services
- **Host Controller Interface (HCI)** – An interface between the host and the controller
- **Link Layer –** Defines packets and radio control procedures
- ***Bluetooth* radio (PHY)** – Transmits and receives bits



Bluetooth Smart Intro

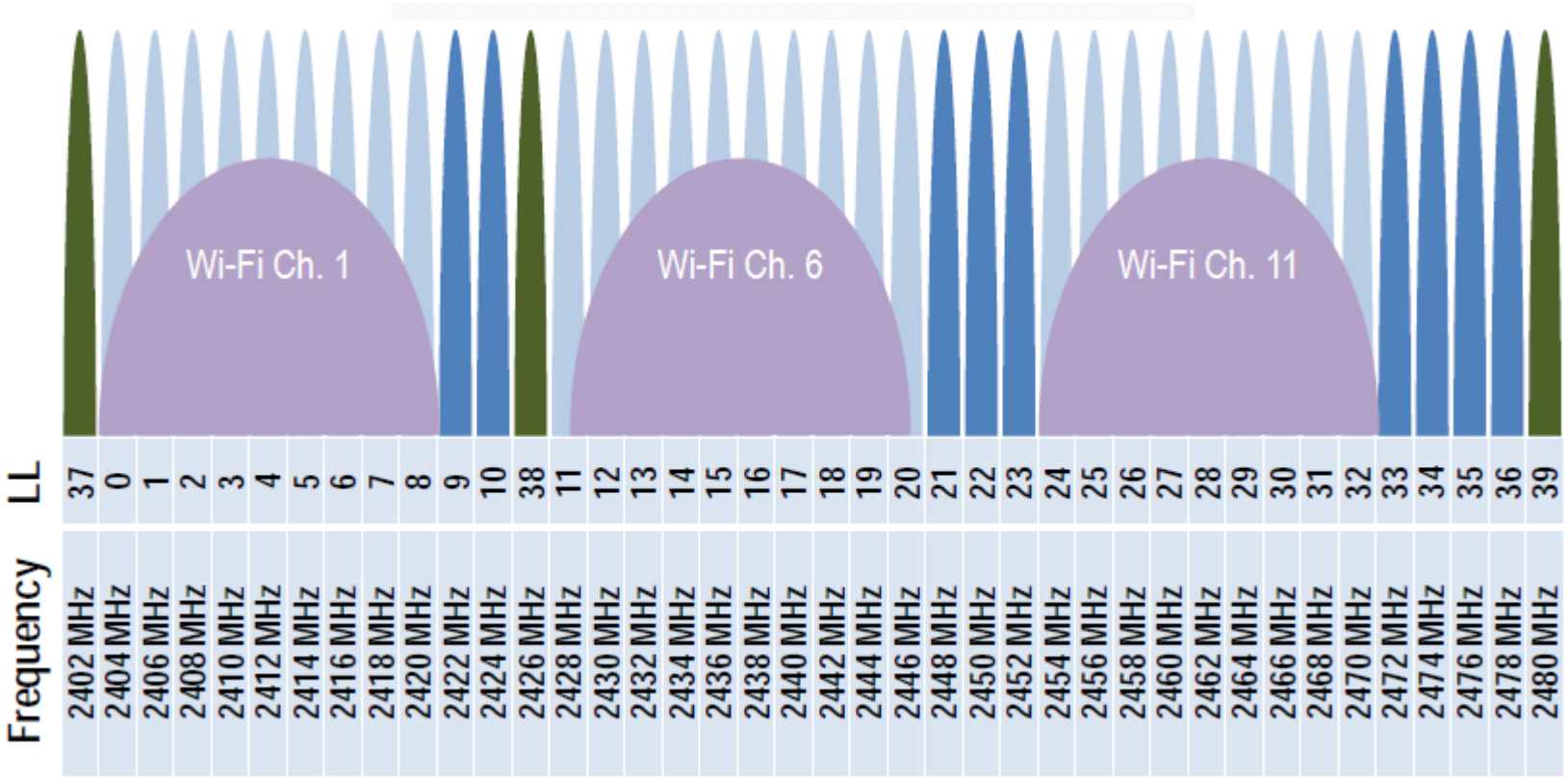
2.4 GHz radio
- Industrial, Scientific and Medical ISM band
- 2400 to 2480 MHz
- License free
- Maximum 10dBm transmit power

GFSK Modulation
- 1Mbps symbol rate
- 0.5 Modulation index – better SNR compared to classic *Bluetooth*

Frequency Hopping Spread Spectrum (FHSS) Radio
- FHSS radio for better interference tolerance and robustness

40 RF channels
- Each channel is 1 MHz
- 1 MHz channel spacing
- 3 channels used for advertisement (discovering devices)
- 37 used for transmitting data

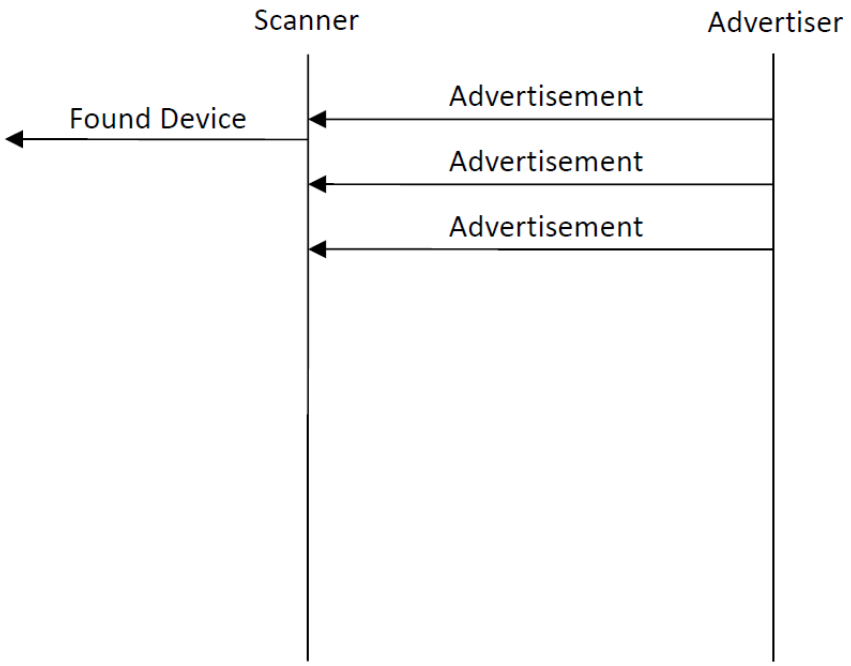
Bluetooth Smart Intro



Bluetooth Smart Intro

- **Discovering Devices**
  - Devices advertise themselves
  - They broadcast advertisement packets of the ADV channels
  - Scanners listen for advertisements to discover devices

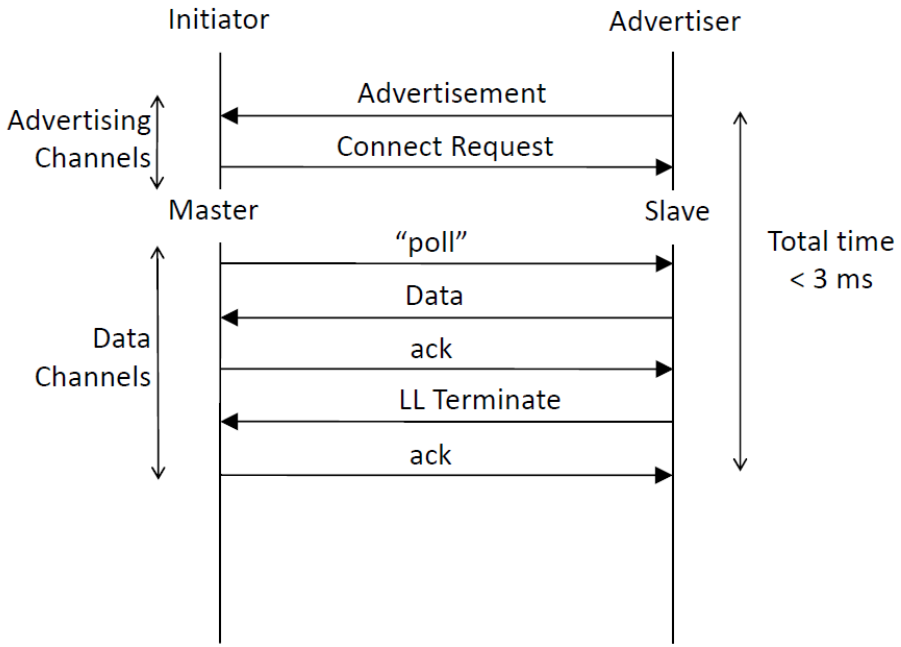
- **Benefits of Advertisements**
  - **Low power** – 1.3ms to TX three ADV packets
  - **Quick** – Again 1.3 ms to TX
  - Can be sent out from every 20ms to every 10240ms – **flexible**
  - Can contain data ~20B – enable **broadcasting** of data



Bluetooth Smart Intro

- **Connections**
  - Enable **reliable data transfer** - connections use ACKs, retrasmits, 24-bit CRC etc.
  - Connections enable the use of **encryption**

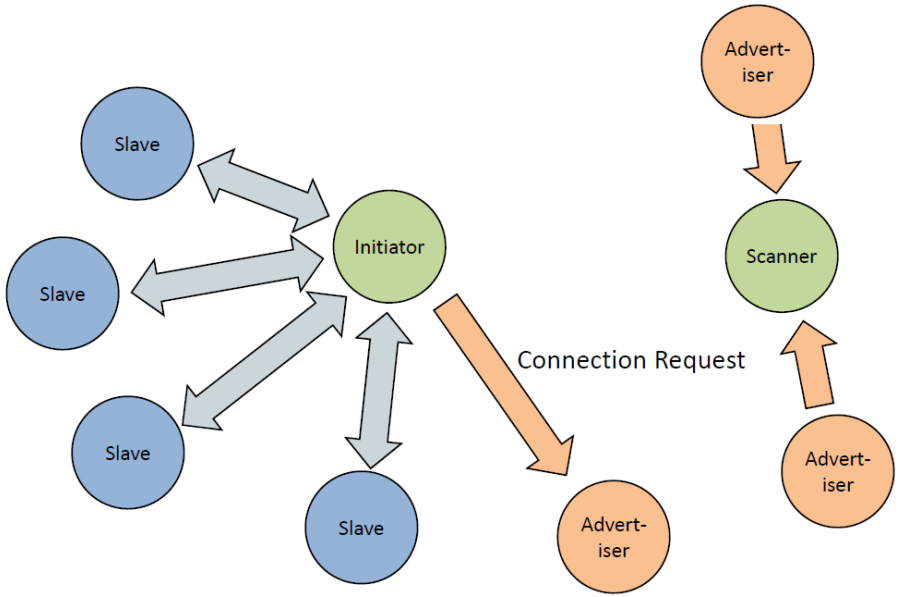
- **Properties**
  - Connection interval from 7.5ms to 4000ms
  - Data payload between 20 to 22B
  - Slave devices can use **slave latency** – enables them to skip N connection intervals when there is no data to transmit



10ms connection interval + ACK allow data to be sent every 20ms.
At 20B payload this is about 1000B/sec.

Bluetooth Smart Intro

- **Device roles and topologies**
  - Some devices can only scan or advertise – never connect
  - Point-to-Point and Star topologies are supported
    - Slave devices support only a single connection
    - Master devices can have multiple connections
    - Number of supported connections vary based on vendor



Bluetooth Smart Intro

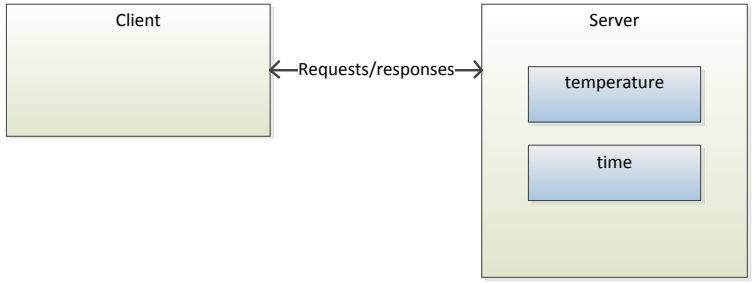
- **Transfering data – the ATT protocol**

  - Uses client – server architecture
    - Server stores the data
    - Client requests data from the server

  - Data is stored as attributes
    - From 0 up to 512 bytes
    - Can be fixed or variable length

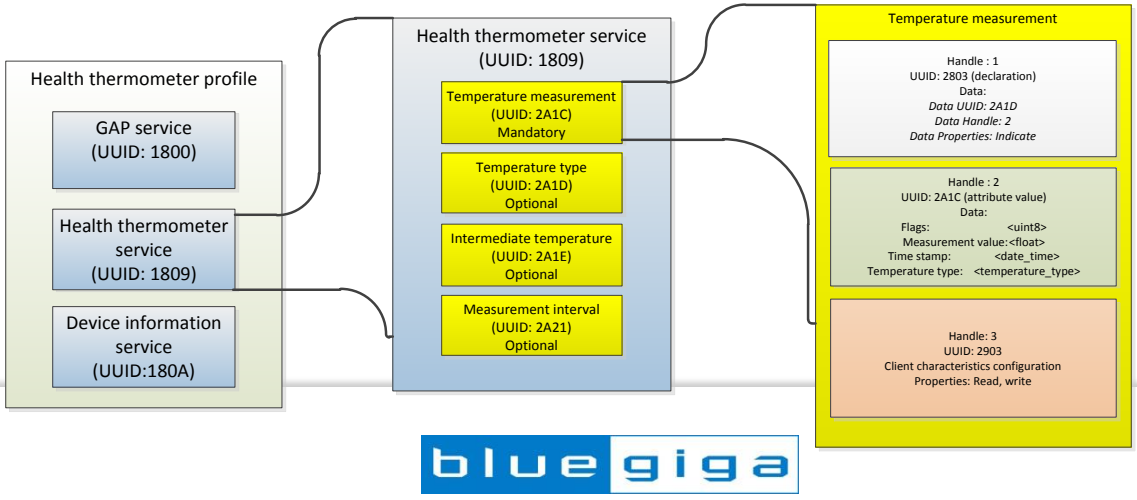
  - Data is accessed using handles and/or UUIDs
    - Every attribute has a unique handle (ID) and UUID
    - UUID describes the data type
      - f.ex 0x2a2b equeals to Current Time
    - 16-bit UUID used for standardized profiles
    - 128-bit UUID used for vendor specific profiles
      - Do not need to be allocted

  - ATT Operations
    - Read and Write
    - Indicate and Notify
    - Write command



Bluetooth Smart Intro

- **Describing and keeping the data organized – the GATT**
  - GATT organizes the data into services
    - Services have UUIDs (16-bit or 128-bit)
  - Services contain one or multipe attributes
    - Every attribute has a unique UUID
    - Attributes typically contain 0 - 512B of data
  - Attributes declarations describe how data is accessed
    - Read, Write, notify, indicate etc.
    - Also access right: the need of bonding and/or encryption
  - Profile descriptions describe, which services need to be included
    - Also include guidelines about connection and re-connection parameters and the use of security



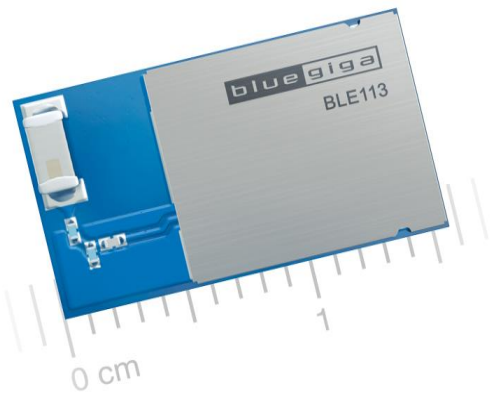
Developing a *Bluetooth* Smart Device

bluegiga

Developing a *Bluetooth* Smart Device

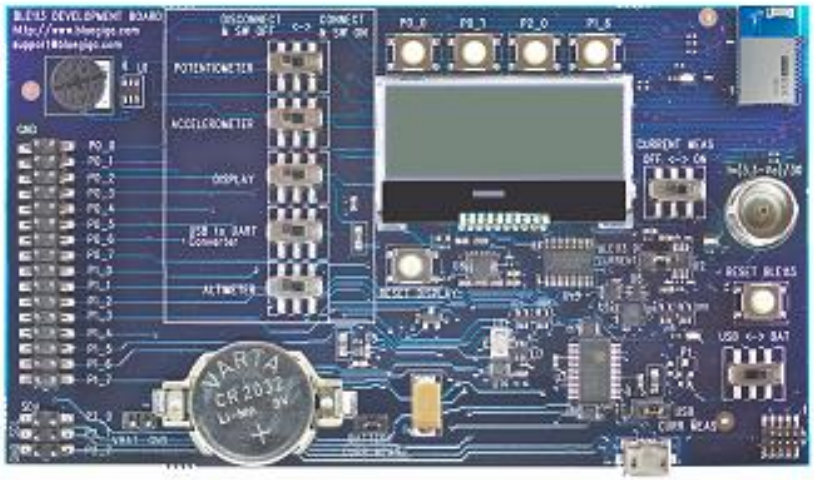
- In this section we briefly describe how to develop a *Bluetooth* Smart Glucose Sensor using the Bluegiga *Bluetooth* Smart products
  - **1st:** Short introduction of the Bluegiga *Bluetooth* Smart Products
  - **2nd:** Developing the GATT services with Bluegiga Profile Toolkit™
  - **3rd**: Developing the sensor's application code with Bluegiga BGScript™ scripting language
  - **4th**: Considerations and tips for iOS and Android application development

BLE113 *Bluetooth* Smart Module



- ***Bluetooth* v.4.0, single mode compliant**
  - Supports master and slave modes
  - Up to 8 connections
- **Integrated *Bluetooth* Smart stack**
  - GAP, GATT, L2CAP and SMP
  - Any *Bluetooth* Smart profile
- **Radio performance**
  - Transmit power : +0 dBm
  - Receiver sensitivity: -93 dBm
- **Ultra low current consumption**
  - Transmit: 14.7 mA (0 dBm)
  - Sleep mode 3: 0.5 uA
- **Flexible peripheral interfaces**
  - UART or SPI
  - I2C
  - PWM, GPIO
  - 12-bit ADC
- **Host interfaces**
  - UART
- **Programmable 8051 processor for stand-alone operation**
- ***Bluetooth*, CE, FCC, IC, South-Korea and Japan qualified**

Development Tools



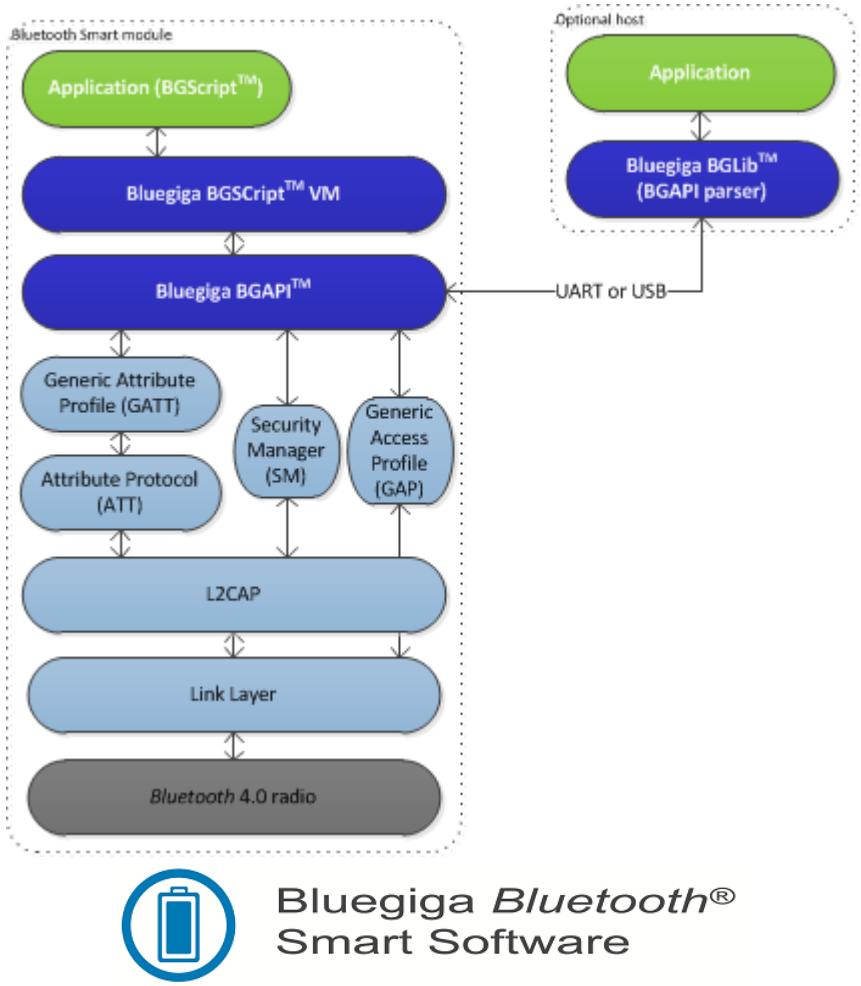
- **BLE113 Development Kit**
  - BLE113-A
  - Display
  - On-board accelerometer
  - On-board altimeter
  - Potentiometer
  - CR2032 battery holder
  - USB and RS232 interfaces
  - Programming interface
  - Current measurement point
  - External DC/DC converter
  - I/O headers

  + Firmware programming tools
  + BLED112 USB dongle
  + 2 x BLE113-A modules

- ***Bluetooth* Smart SDK**
  - BGAPI™ documentation
  - BGScript™ development tools
  - BGLib™ source code
  - Profile Toolkit™
  - BGScript and BGLib examples
  - Profile examples
  - Documentation
  - iOS example applications

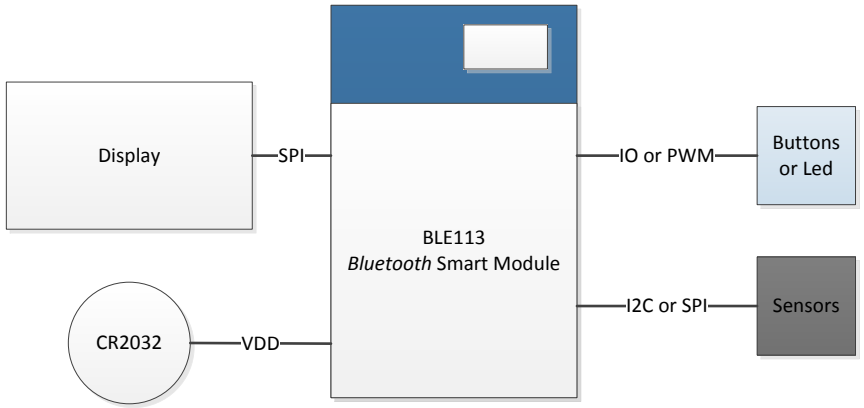
The Bluegiga *Bluetooth* Smart Software

- ***Bluetooth v.4.0*, single mode compliant**
  - Supports master and slave modes
  - Up to 8 simultaneous connections
- **Implements all *Bluetooth* Smart functionality**
  - GAP, L2CAP, ATT, GATT
  - Security manager: bonding, encryption
  - *Bluetooth* Smart profiles
- **Simple API for external host processors**
  - BGAPI™ : A simple protocol over UART or USB interfaces
  - BGLIB™ : A C library for host processors implementing BGAPI
- **Supports standalone applications as well**
  - BGScript™ : A simple scripting language for writing applications
  - Native C application development with IAR Embedded Workbench
  - **No separate host needed**
- **Blutoooth Smart Profile Toolkit™**
  - XML based development tool for Bluetooth Smat profiles
  - Fast and simple profile development



Implementing Standalone Devices

- **Standalone architecture**: No separate host processor
  - Sensors and peripherals are directly connected to the BLE113 via the IO interfaces
  - Application executed on the on-board 8051
  - Application developed with BGScript™ or ANSI C and services and profiles with Profile Toolkit™



Developing the Glucose Profile

- **The Glucose Profile:**
  - The Glucose Profile is used to enable a device to obtain glucose measurement and other data from a Glucose Sensor that exposes the Glucose Service.
  - Described at the Bluetooth SIG developer profile (link) http://developer.bluetooth.org
  - Needs to implement the following services:
    - Glucose Service (link)
    - Device Information Service (link)
    - Also : the GAP service (link)

Developing the Glucose Service

- **The Glucose Service:**
  - Needs to implement the following attributes:

Characteristic	UUID	Length	Type	Support	Security	Properties
Glucose Measurement	2A18	Variable (max 17B)	Hex	Mandatory	None	Notify
Glucose Measurement Context	2A34	Variable (max 17B)	Hex	Optional	None	Notify
Glucose Feature	2A51	2 bytes	Hex	Mandatory	None	Read
Record Access Control Point	2A52	Variable (typical 2B)	Hex	Mandatory	Writeable with Authentication	Write, Indicate

Profile Toolkit Implementation of the Glucose Service

```
<service uuid="1808" advertise="true">
    <description>Glucose Service</description>

    <characteristic uuid="2A18" id="c_glucose_measurement">
        <description>Glucose Measurement</description>
        <properties notify="true" />
        <value length="17" variable="true" />
    </characteristic>

    <characteristic uuid="2A34" id="c_glucose_measurement_context">
        <description>Glucose Measurement Context</description>
        <properties notify="true" />
        <value length="17" variable="true" />
    </characteristic>

    <characteristic uuid="2A51" id="c_glucose_feature">
        <description>Glucose Feature</description>
        <properties read="true" const="true" />
        <value length="2" type="hex">07FF</value>
    </characteristic>

    <characteristic uuid="2A52" id="c_record_access_control_point">
        <description>Record Access Control Point</description>
        <properties indicate="true" write="true" authenticated_write="true" />
        <value length="17" variable="true" />
    </characteristic>
</service>
```

Developing the Application Code with BGScript

- **BGScript™ scripting language** : A very simple BASIC-like application scripting language
  - Used when applications are implemented on the BLE113's 8051 controller
  - Enables very fast application development and allows programs to be executed directly on the BLE113 without the need of an external MCU

```
# System boot event listener : Executed when BLE112 is started
event system_boot(major ,minor ,patch ,build ,ll_version ,protocol_version ,hw )


    # Configure ADV interval to 1000ms and start advertisements an all channels
    call gap_set_adv_parameters(1600, 1600, 7)

    # Start generic advertisement and enable connections
    call gap_set_mode(2,2)

   #Start a continuous software timer, which generates interrupts every 1000ms
   call hardware_set_soft_timer(32768, 1, 0)
end
```

Glucose Sensor - The Application Code

```
event system_boot(major, minor, patch, build, ll_version, protocol, hw)

      # initialize connection status as DISCONNECTED
      connected = 0
      # set tick counter to zero
      tick = 0
      # set sequence number to zero
      seq_num = 0
      # initialize the alternating animation state
      alternating = 1
      # initialize countdown ticks to 0 (no special message to show temporarily on display)
      wait_ticks = 0

      # set advertisement interval to 20-30ms, and use all advertisement channels
      # (note min/max parameters are in units of 625 uSec)
      call gap_set_adv_parameters(32, 48, 7)

      # put module into discoverable/connectable mode
      call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)

      # enable bondable
      call sm_set_bondable_mode(1)

      # enable interrupt on P0_0 rising edge (triggers glucose reading)
      # also on P0_1 rising edge (triggers PS key userdata reset)
      call hardware_io_port_config_irq(0, 3, 0)

      # configure timer for ~1 second intervals
      call hardware_set_soft_timer(32000, 0, 0)
end
```

Developing the Application Code

```
# catch result of ADC read
event hardware_adc_result(input, value)

      if input = 15 then
        # battery level reading received, store to GATT DB for reading
        call attributes_write(c_battery_level, 0, 2, value)
      end if
      if input = 6 then
            # potentiometer value received, so we'll pretend it's
            # a glucose measurement for the purposes of this demo
            # set <Flags> value to 00011011 in binary
            measure_buf(0:1) = $1B
            # set <Sequence Number> (incremented later, after the context report is built)
            measure_buf(1:1) = seq_num          # 16-bit LSB
            measure_buf(2:1) = seq_num / 256    # 16-bit MSB
            # set <Base Time> to example value of 2009-07-24 16:30:00
            measure_buf(3:2) = $07D9    # 0x07D9 = 2009
            measure_buf(5:1) = $07      # 0x07 = July
            measure_buf(6:1) = $18      # 0x18 = 24
            measure_buf(7:1) = $10      # 0x10 = 16
            measure_buf(8:1) = $1E      # 0x1E = 30
            measure_buf(9:1) = $00      # 0x00 = 0, of course.

            # set <Time Offset> to seq_num, to imitate passage of time
            measure_buf(10:1) = seq_num         # 16-bit LSB
            measure_buf(11:1) = seq_num / 256   # 16-bit MSB
            tmp_float = (value / 30) + 720
            measure_buf(12:2) = sfloat(tmp_float, -6)
            # …
            measure_buf(15:2) = $0000        # write glucose measurement characteristic value
            call attributes_write(c_glucose_measurement, 0, 17, measure_buf(0:16))
            # …
end
```

iOS Device Considerations

- iOS can operate as central and peripheral
- Advertise the service UUIDs in advertisement packet
  - The iOS App can filter devices based on UUIDs
- Minimum connection interval ~20ms
  - When App is put to background the connection interval might be increased
- iOS devices cache services
  - Implement the generic GATT service and iOS will refresh services on every connection

iOS Device Considerations

- Need xCode developer license and OSX developer tools
  - Available at the Apple's developer site
- MFI
  - You do not need to be part of MFI in order to develop *Bluetooth* Smart Apps for iOS
- Bluegiga example iOS App available
  - Download from www.bluegiga.com
  - Available in source code

Android Device Considerations

- *Bluetooth* Smart APIs available since 4.3
  - In API level 18
- Currentle supported devices: Nexus 4 and 7. Samsung Galaxy S3/4 (updates rolling out)
- Android only supports central mode (master)
- Backgroud applications supported
- Android supports secure connections and insecure connections
- However Bluetooth Smart implementation is not very robust – improvements expected in Android 4.4 (KitKat)

Android Device Considerations

- You need Android Development Kit (ADK)
  - Available on Android developer web site
  - Free-of-Charge
- You need the latest API level 18 access
- Bluegiga example Android App available
  - Download from www.bluegiga.com
  - Available in source code and as APK

More Information

- Bluegiga BLE Software and SDK, Example Applications, Documents and Smart Phone examples
  - Documentation and Downloads
- *Bluetooth* Smart SDK v.1.2 Introduction
  - Presentation
- Over-the-Air Firmware Update
  - Application Note
- iBeacons example and discussion
  - Example: Bluegiga Forums
  - Discussion: Bluegiga Forums

More Information

- Bluegiga
  - www.bluegiga.com
  - www.bluegiga.com/support
- *Bluetooth* SIG
  - www.bluetooth.org
  - www.bluetooth.com
  - http://developer.bluetooth.org
- iOS Development
  - iOS Dev Center
- Android Development
  - Android Developers

Questions and Answers