Securely

Get

Communication with Bluetooth Low-Energy Devices on Linux

What is Bluetooth and Bluetooth Low Energy?

Bluetooth™ is a wireless technology used to build Personal Area Networks (PAN) or piconets. Since it's original development by Ericsson in 1989, it has grown in popularity to the point where today it has become a standard feature in many segments of the consumer electronic market.

The use of Bluetooth in industrial IoT is also growing, mostly due to Bluetooth LE, a newer variant of Bluetooth optimized for minimum power usage, as is required for wireless, battery-powered sensor devices.

This blog post takes you through the important details of the Bluetooth LE technology from the perspective of an IoT developer. It is based on the presentation at the Internet of Things conference "From the sensor to the cloud" 2021, by our Founder & Chief Technologist Günter Obiltschnig.

A few facts about Bluetooth

Development of Bluetooth technology was originally initiated by Nils Rydbeck (Ericsson) in 1989. The first use case was not until ten years later, and was released at Comdex in 1999 as the first hands-free mobile headset. It won the "Best of show Technology Award". This first product had a range of 10 meters but nowadays, that range can go up to 100m with Bluetooth 5.0.

Bluetooth uses the globally unlicensed ISM (industrial, scientific, medical) radio bands from 2402 to 2480 GHz. One of the major advantages of Bluetooth is that it does not rely on a wireless network infrastructure.

The Bluetooth standards are developed and maintained by the Bluetooth Special Interest Group.

Search Posts

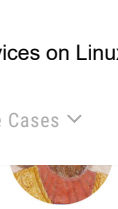
Search

Featured Posts

- Machine Learning in Manufa
  Power of AI
- macchina.io REMOTE now s
  PostgreSQL
- macchina.io EDGE Release 2
- How to Provide Secure Rem
  Edge Devices
- macchina.io REMOTE Delive
  Access for Service Partners a
- macchina.io Remote Manag
  Available
- Building an IoT Edge Applica
  macchina.io and Docker (Pa
- On the Edge

Categories

- Latest
- C++
- Company
- Edge Computing
- Events
- Internet of Things
- IoT Platforms
- macchina.io
  - macchina.io EDGE
  - macchina.io REMOTE
- my-devices.net
- OSP
- POCO C++ Libraries
- Remoting
- Security



The name Bluetooth is an anglicized version of the Scandinavian Blåtand/Blåtann, the epithet of the tenth-century king Harald Bluetooth who united dissonant Danish tribes into a single kingdom. The implication is that Bluetooth unites communication protocols in the same way king Bluetooth united tribes.

Bluetooth Low Energy (LE)

Bluetooth LE (formerly also known as Bluetooth Smart) was introduced with the Bluetooth 4.0 specification in 2010. It is intended to reduce power consumption and cost while maintaining a similar communication range. The actual connection times are reduced to a few milliseconds; significantly less than Bluetooth, which has connection times ranging from a few seconds up to a few hours.

Bluetooth LE can co-exist with Bluetooth on the same controller, however, the Bluetooth LE protocol stack is different from classic Bluetooth. It uses the same 2.4 GHz radio bands as Bluetooth, but a different communication protocol and different channels with 2 MHz bandwidth.

Bluetooth LE is targeted at devices with very low-power requirements, such as fitness and health sensors (heart rate monitors, thermometers, blood pressure, etc) as well as fitness trackers, battery-powered environmental and industrial sensors, beacons, etc.

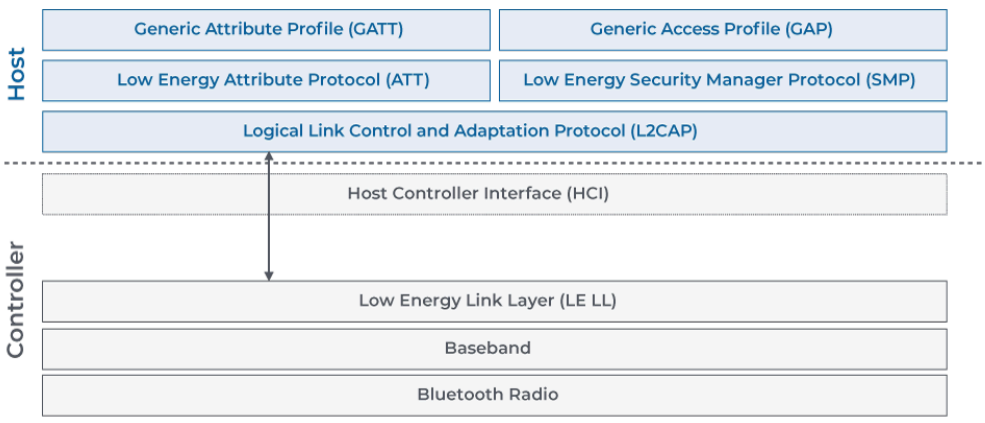
Bluetooth LE uses a unique 48-bit address per device (MAC address). Similar to Ethernet MAC addresses, the first part of the address is the “vendor prefix”, which allows to deduct the vendor of a Bluetooth device (or at least its chipset) from its address.

Bluetooth LE Radio

Bluetooth LE uses 40 channels with 2 MHz bandwidth. Within each channel, Gaussian Frequency Shift Keying (GFSK) is used, similar to classic Bluetooth 1.0. The bitrate is 1 Mbit/sec, with an option for 2 Mbit/sec in Bluetooth 5.0.

Bluetooth LE uses frequency hopping like “classic” Bluetooth, but in a different variant (direct sequence spread spectrum).

Bluetooth LE Protocol Stack



GAP – Generic Access Profile

Bluetooth LE devices use the Generic Access Profile for device discovery and advertisement.

GAP distinguishes between devices based on their roles. Central devices are typically a computer or smartphone. Peripheral devices are often small, low-power and resource-constrained devices like sensors.

- Video

Tags

airvantage c++ certifi docker docker-comp computing e framework iiot internet-of-thing internet of things iot javascript json-rpc letse logging msvc NSLog osp poc raspberry-pi remote ac remoting security sm synology thingspeak tinkerforge vulnerability web sockets wince

Securely

Get

amounts of custom data. This is used by Bluetooth LE beacons (e.g., iBeacon).

Both peripheral and central devices can assume one or both of the other GAP roles, which are Broadcaster and Observer. A very typical example would be a sensor in broadcast mode sending data to a central device in observer mode.

Securely

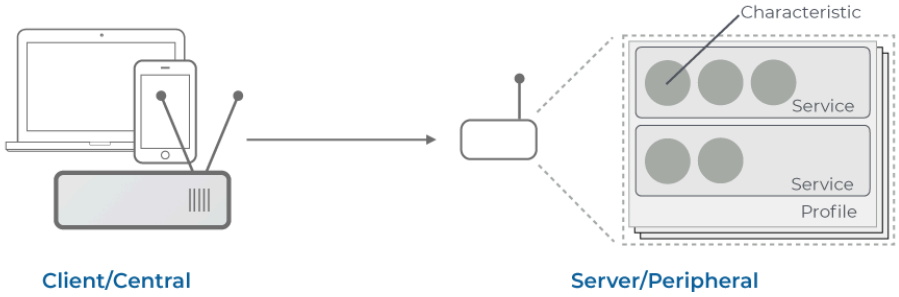
Get

GATT – Generic Attribute Profile

GATT defines the following concepts, which are often reflected in Bluetooth LE APIs.

Client: typically a computer or smartphone that sends GATT commands and requests and receives responses from servers. Also referred to as central device.

Server: a device (e.g., a sensor) that receives commands and requests and returns responses. Also referred to as peripheral device.



Characteristic: a data value transferred between client and server. Consists of multiple **attributes** containing the actual value, as well as configuration options and meta data.

Different data types are supported, but on protocol level an attribute value is always a sequence of bytes. Characteristics and their attributes are used for reporting sensor values (e.g., temperature) and also for configuring devices (e.g., setting sample rate on an accelerometer). The maximum length is 512 bytes. Numeric values are usually stored in little-endian format.

Service: a collection of related characteristics, which operate together, e.g. the sensor value and configuration parameters.

Descriptor: provides additional information about a characteristic's attribute, e.g. physical unit or minimum and maximum values.

Identifier: services, characteristics and descriptors are attributes identified by UUIDs. The 128-bit UUIDs are mapped to device-specific 16-bit values (handles) for use in the protocol. Bluetooth reserves UUIDs in range xxxxxxxx-0000-1000-8000-00805F9B34FB for standard attributes, which are transmitted as 16-bit or 32-bit values in the protocol.

GATT Operations

GATT comes into play once a connection is made between two devices. For data to be delivered and received, some or all of the following operations can be executed.

- discover UUIDs for primary services
- find a service with a given UUID
- find secondary services for a given primary service
- discover all characteristics for a given service
- find characteristics matching a given UUID
- read descriptors for a particular characteristic
- read the value of a characteristic (a sequence of bytes)
- write the value of a characteristic (a sequence of bytes)

Servers can send notifications and indications to clients on certain state changes (e.g., the sensor value has changed); indications must be acknowledged by the client.

Securely

Get

Bluetooth LE Profiles

Like classic Bluetooth, Bluetooth LE defines a number of profiles. All Bluetooth LE profiles are based on GATT, therefore, a Bluetooth LE profile defines the services and characteristics (and their IDs) that a device must support.

Examples:

- BLP – Blood Pressure Profile
- HTP – Health Thermometer Profile
- GLP – Glucose Profile
- HRP – Heart Rate Profile
- CPP – Cycling Power Profile
- WSP – Weight Scale Profile
- LNP – Location and Navigation Profile
- ESP – Environmental Sensing Profile

BlueZ – Bluetooth & Bluetooth LE on Linux

BlueZ is the official Linux Bluetooth protocol stack. It was originally developed by Max Krasnyansky at Qualcomm (starting in 2001), and is available for Linux kernels 2.4.6 and newer. Currently the main developers are Marcel Holtmann and Johan Hedberg, currently at Intel.

BlueZ is listed as a qualified protocol stack on the Bluetooth Qualification website, which simplifies the product qualification process. It consists of kernel modules and user space libraries and is included in all major Linux distributions. BlueZ has a low-level C API and a high-level D-Bus API.

BlueZ is licensed under the GPL, but is (apparently) in the process of being moved to LGPL.

bluepy

bluepy by Ian Harvey is a Python interface to Bluetooth LE on Linux. bluepy consists of a Python module and an executable (*bluepy-helper*) written in C. The *bluepy-helper* executable wraps the BlueZ C API in a command-line interface.

The output of *bluepy-helper* commands is easily parseable by programs. The bluepy python module starts *bluepy-helper* and sends command via a pipe. It can also be used interactively, which is great for making the first steps with a Bluetooth LE peripheral.

bluepy-helper is licensed under the GPL, like the BlueZ libraries it uses (derived work), however programs launching *bluepy-helper* and talking to it via a pipe need not be under GPL.

Getting Started (Debian/Ubuntu/Raspberry Pi OS)

Install Dependencies

```
$ sudo apt-get update && \
sudo apt-get install bluetooth build-essential libglib2.0-dev \
libdbus-1-dev
```

Get and Build bluepy-helper

Securely

Get

Enable Bluetooth

```
$ sudo hciconfig hci0 up
```

Search for Bluetooth LE devices (address and name)

```
$ sudo hcitool lescan
64:BB:12:3B:05:3E (unknown)
79:77:B8:8C:E3:F7 (unknown)
58:2D:34:35:D7:1D (unknown)
58:2D:34:35:D7:1D MJ_HT_V1
3B:85:C4:72:1A:29 (unknown)
3B:85:C4:72:1A:29 (unknown)
27:05:78:F6:C9:F3 (unknown)
80:6F:B0:F0:0D:81 Multi-Sensor
80:6F:B0:F0:0D:81 (unknown)
```

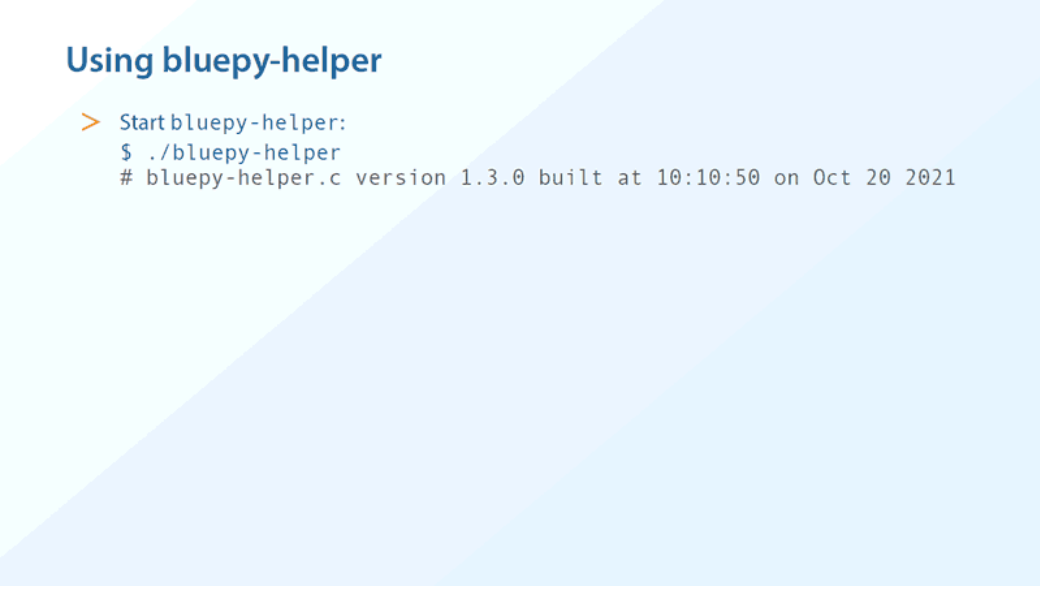
Interacting with Bluetooth LE Devices

Interacting with Bluetooth LE devices typically comprises the following operations:

- Detecting supported services and characteristics
- Writing characteristics to configure device
- Reading characteristics to read sensor and other data
- Handling notifications and indications

We can perform these steps with bluepy-helper

Using bluepy-helper



Start bluepy-helper

```
$ ./bluepy-helper
# bluepy-helper.c version 1.3.0 built at 10:10:50 on Oct 20 2021
```

```
conn 58:2D:34:35:D7:1D
rsp=$stat▫state=$tryconn▫dst='58:2D:34:35:D7:1D▫mtu=h0▫sec='low
rsp=$stat▫state=$conn▫dst='58:2D:34:35:D7:1D▫mtu=h0▫sec='low
```

Securely

Get

Discover Services Offered by Device

```
svcs
rsp=$find▫hstart=h1▫hend=h7▫uuid='00001800-0000-1000-8000-00805f9b34fb▫
hstart=h8▫hend=hB▫uuid='00001801-0000-1000-8000-00805f9b34fb▫
hstart=hC▫hend=h15▫uuid='226c0000-6476-4566-7562-66734470666d▫
hstart=h16▫hend=h19▫uuid='0000180f-0000-1000-8000-00805f9b34fb▫
hstart=h1A▫hend=h24▫uuid='0000180a-0000-1000-8000-00805f9b34fb▫
hstart=h25▫hend=h2C▫uuid='00001530-1212-efde-1523-785feabcd123▫
hstart=h2D▫hend=hFFFF▫uuid='0000fe95-0000-1000-8000-00805f9b34fb
```

Examining a Service (Battery)

```
hstart=h16▫hend=h19▫uuid='0000180f-0000-1000-8000-00805f9b34fb▫

char 16 19
rsp=$find▫
hnd=h17▫props=h12▫vhnd=h18▫uuid='00002a19-0000-1000-8000-00805f9b34fb
```

Reading the Battery Level

```
rd 18
rsp=$rd▫d=b38
```

The returned hexadecimal value 38 is decimal 56, which means 56 % battery level.

Examining another service (Device Information)

```
hstart=h1A▫hend=h24▫uuid='0000180a-0000-1000-8000-00805f9b34fb

char 1A 24
rsp=$find▫
hnd=h1B▫props=h2▫vhnd=h1C▫uuid='00002a29-0000-1000-8000-00805f9b34fb▫
hnd=h1D▫props=h2▫vhnd=h1E▫uuid='00002a24-0000-1000-8000-00805f9b34fb▫
hnd=h1F▫props=h2▫vhnd=h20▫uuid='00002a25-0000-1000-8000-00805f9b34fb▫
hnd=h21▫props=h2▫vhnd=h22▫uuid='00002a27-0000-1000-8000-00805f9b34fb▫
hnd=h23▫props=h2▫vhnd=h24▫uuid='00002a26-0000-1000-8000-00805f9b34fb
```

Read Model Number String

```
rd 1E
rsp=$rd▫d=b4475636B5F52656C65617365
```

The hexadecimal string `4475636B5F52656C65617365` translates to “Duck_Release”, which actually does not make much sense, but it is what the specific device (a Xiaomi Mijia temperature and humidity sensor).

Securely

Get

Creating a High-Level Bluetooth LE API

We can now implement a high-level Bluetooth LE API for a programming language of our choice on top of *bluepy-helper*. We only need to be able to start a process and communicate with it via a pipe.

For macchina.io EDGE, we have created a C++ API that can also be used from JavaScript.

You can find the interface definition and implementation in the macchina.io EDGE (open source) repository on GitHub.

Implementation

The implementation roughly does the following:

- Connect: start a new *bluepy-helper* helper process for the peripheral.
- Disconnect**:** kill helper process.
- For every operation, send the respective command to *bluepy-helper* and parse the result.
- Handle indications and notifications.
- Perform some state-keeping.

In addition to the generic Peripheral interface, macchina.io EDGE also has high-level support for a few specific Bluetooth LE devices, such as a TI SensorTag or a Bosch XDK.

In a future article, we will show how to use the macchina.io EDGE Bluetooth support to connect to a specific device and communicate with the device using a few lines of JavaScript.

Try macchina.io REMOTE for FREE

Share this post

share share share email

By Tristan Boyd | Published 2021-11-14

Posted in Edge Computing, Internet of Things, IoT Platforms, macchina.io

« macchina.io EDGE Release 2021.1 Available

macchina.io REMOTE Delivers Remote Access for Service Partners and End Users »

macchina.io empowering things

Securely

Get

macchina.io REMOTE
Secure remote access to IoT devices
Overview
Benefits
Features
Licensing & Pricing
FAQ
Sign Up (Free Account)
Agent/Clients/SDK Downloads
Sign In

macchina.io EDGE
IoT edge device/gateway SDK for C++ and JavaScript
Overview
Benefits
Features
Licensing & Pricing
Technology
System Requirements
Examples
FAQ

Use Cases
macchina.io for IoT Gateways
Automotive Telematics & V2X
Industrial IoT and Edge Gateways
Smart Connected Infrastructure

Resources
macchina.io REMOTE Resources
macchina.io EDGE Resources

Services

Company
About Us
Blog
Imprint
Privacy Policy
Cookie Policy

GitHub
X/Twitter
YouTube

Contact Us
Preferences & Opt-Out

© 2014-2024 Applied Informatics Software Engineering GmbH. Proudly made in .