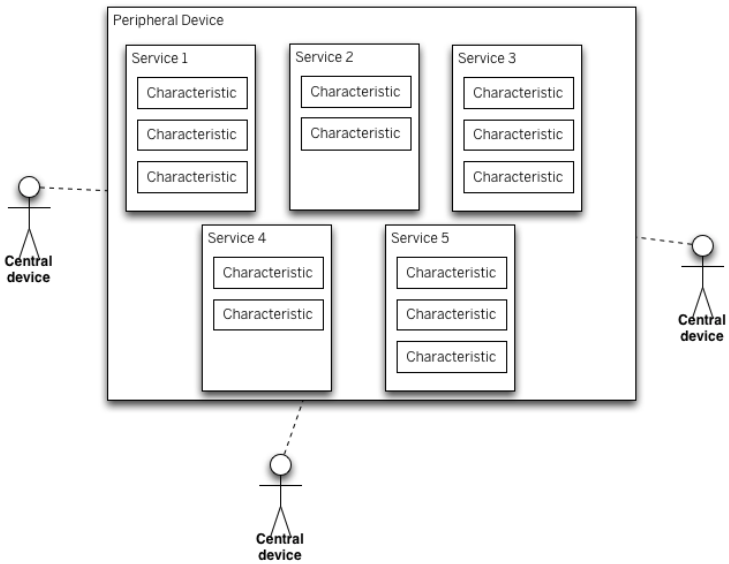
CurieBLE library

With the Arduino/Genuino 101 (//www.arduino.cc/en/Main/ArduinoBoard101), using this library, it is possible to use BLE features to communicate and interact with other devices like smartphones and tablet

A quick introduction to BLE

Bluetooth 4.0 includes both traditional Bluetooth, now labeled "Bluetooth Classic", and the new Bluetooth Low Energy (Bluetooth LE, or BLE). BLE is optimized for low power use at low data rates, and was designed to operate from simple lithium coin cell batteries.

Unlike standard bluetooth communication basically based on an asynchronous serial connection (UART) a Bluetooth LE radio acts like a community bulletin board. The computers that connect to it are like community members that read the bulletin board. Each radio acts as either the bulletin board or the reader. If your radio is a bulletin board (called a peripheral device in Bluetooth LE parlance) it posts data for all radios in the community to read. If your radio is a reader (called a central device in Blueooth LE terms) it reads from any of the bulletin boards (peripheral devices) that have information about which it cares. You can also think of peripheral devices as the servers in a client-server transaction, because they contain the information that reader radios ask for. Similarly, central devices are the clients of the Bluetooth LE world because they read information available from the peripherals.



(//www.arduino.cc/en/uploads/Reference/ble-bulletin-board-model.png)
Think of a Bluetooth LE peripheral device as a bulletin board and central devices as viewers of the board. Central devices view the services, get the data, then move on. Each transaction is quick (a few milliseconds), so multiple central devices can get data from one peripheral.

The information presented by a peripheral is structured as **services**, each of which is subdivided into **characteristics**. You can think of services as the notices on a bulletin board, and characteristics as the individual paragraphs of those notices. If you're a peripheral device, you just update each

BLEPeripheral class

- BLEPeripheral (//www.arduino.cc/en/Reference/BLEPeripheralConstructor)
- begin (//www.arduino.cc/en/Reference/BLEPeripheralBegin)()
- poll (//www.arduino.cc/en/Reference/BLEPeripheralPoll)()
- end (//www.arduino.cc/en/Reference/BLEPeripheralEnd)()
- setAdvertisedServiceUuid (//www.arduino.cc/en/Reference/BLEPeripheralSetAdvertisedServiceUuid)()
- setLocalName (//www.arduino.cc/en/Reference/BLEPeripheralSetLocalName)()
- setDeviceName (//www.arduino.cc/en/Reference/BLEPeripheralSetDeviceName)()
- setAppearance (//www.arduino.cc/en/Reference/BLEPeripheralSetAppearance)()
- setEventHandler (//www.arduino.cc/en/Reference/BLEPeripheralSetEventHandler)()
- addAttribute (//www.arduino.cc/en/Reference/BLEPeripheralAddAttribute)()
- disconnect (//www.arduino.cc/en/Reference/BLEPeripheralDisconnect)()
- central (//www.arduino.cc/en/Reference/BLEPeripheralCentral)()
- connected (//www.arduino.cc/en/Reference/BLEPeripheralConnected)()

BLEDescriptor class

Descriptors are defined attributes that describe a characteristic value.

- BLEDescriptor (//www.arduino.cc/en/Reference/BLEDescriptorConstructor)

BLECentral class

The device the board is connected to

- BLECentral (//www.arduino.cc/en/Reference/BLECentralConstructor)

service characteristic when it needs updating and don't worry about whether the central devices read them or not. If you're a central device, you connect to the peripheral then read the boxes you want. If a given characteristic is readable and writable, then the peripheral and central can both change it.

Notify

The Bluetooth LE specification includes a mechanism known as **notify** that lets you know when data's changed. When notify on a characteristic is enabled and the sender writes to it, the new value is automatically sent to the receiver, without the receiver explicitly issuing a read command. This is commonly used for streaming data such as accelerometer or other sensor readings. There's a variation on this specification called **indicate** which works similarly, but in the indicate specification, the reader sends an acknowledgement of the pushed data.

The client-server structure of Bluetooth LE, combined with the notify characteristic, is generally called a **publish-and-subscribe model**.

Update a characteristic

Your peripheral should update characteristics when there's a significant change to them. For example, when a switch changes from off to on, update its characteristic. When an analog sensor changes by a significant amount, update its characteristic.

Just as with writing to a characteristic, you could update your characteristics on a regular interval, but this wastes processing power and energy if the characteristic has not changed.

Central and Peripheral Devices

Central devices are **clients**. They read and write data from peripheral devices. **Peripheral** devices are **servers**. They provide data from sensors as readable characteristics, and provide read/writable characteristics to control actuators like motors, lights, and so forth.

Services, characteristics, and UUIDs

A BLE peripheral will provide **services**, which in turn provide **characteristics**. You can define your own services, or use standard services (https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx).

Services are identified by unique numbers known as UUIDs. You know about UUIDs from other contexts. Standard services have a 16-bit UUID and custom services have a 128-bit UUID. The ability to define services and characteristics depends on the radio you're using and its firmware.

Service design patterns

A characteristic value can be up to 20 bytes long. This is a key constraint in designing services. Given this limit, you should consider how best to store data about your sensors

- connected (//www.arduino.cc/en/Reference/BLECentralConnected)()
- address (//www.arduino.cc/en/Reference/BLECentralAddress)()
- disconnect (//www.arduino.cc/en/Reference/BLECentralDisconnect)()
- poll (//www.arduino.cc/en/Reference/BLECentralPoll)()

BLECharacteristic class

The BLE characteristics the board must show

- BLECharacteristic (//www.arduino.cc/en/Reference/BLECharacteristicConstructor)

BLEService class

The BLE service the board must use

- BLEService (//www.arduino.cc/en/Reference/BLEServiceConstructor)

and actuators most effectively for your application. The simplest design pattern is to store one sensor or actuator value per characteristic, in ASCII encoded values.

Characteristic	Value
Accelerometer X	200
Accelerometer Y	134
Accelerometer Z	150

This is also the most expensive in memory terms, and would take the longest to read. But it's the simplest for development and debugging.

You could also combine readings into a single characteristic, when a given sensor or actuator has multiple values associated with it.

Characteristic	Value
Motor Speed, Direction	150,1
Accelerometer X, Y, Z	200,133,150

This is more efficient, but you need to be careful not to exceed the 20-byte limit. The accelerometer characteristic above, for example, takes 11 bytes as a ASCII-encoded string.

Read/write/notify/indicate

There are 4 things a central device can do with a characteristic:

- **Read**: ask the peripheral to send back the current value of the characteristic. Often used for characteristics that don't change very often, for example characteristics used for configuration, version numbers, etc.
- **Write**: modify the value of the characteristic. Often used for things that are like commands, for example telling the peripheral to turn a motor on or off.
- **Indicate** and **Notify**: ask the peripheral to continuously send updated values of the characteristic, without the central having to constantly ask for it.

Advertising and GAP

BLE devices let other devices know that they exist by advertising using the **General Advertising Profile (GAP)**. Advertising packets can contain a device name, some other information, and also a list of the services it provides.

Advertising packets have a limited size. You will only be able to fit a single 128-bit service UUID in the packet. Make sure the device name is not too long, or you won't even be able to fit that.

You can provide additional services that are not advertised. Central devices will learn about these through the connection/bonding process. Non-advertised services cannot be used to discover devices, though. Sometimes this is not an issue. For example, you may have a custom peripheral device with a custom service, but in your central device app you may know that it also provides the Battery Service and other services.

GATT

The Bluetooth LE protocol operates on multiple layers. **General Attribute Profile (GATT)** is the layer that defines services and characteristics and enables read/write/notify /indicate operations on them. When reading more about GATT, you may encounter GATT concepts of a "server" and "client". These don't always correspond to central and peripherals. In most cases, though, the peripheral is the GATT server (since it provides the services and characteristics), while the central is the GATT client.

Library structure

As the library enables multiple types of functionality, there are a number of different classes.

- BLEPeripheral used to enable the BLE module
- BLEDescriptor that prepares the functions that the board will show
- BLECentral that represent the device the board is connected to
- BLECharacteristic used to enable the characteristics board offers
- BLEService used to enable the services board provides

Examples

- Heart Rate Monitor (//www.arduino.cc/en/Tutorial /Genuino101CurieBLEHeartRateMonitor)
- Battery Monitor (//www.arduino.cc/en/Tutorial /Genuino101CurieBLEBatteryMonitor)
- Button LED (//www.arduino.cc/en/Tutorial /Genuino101CurieBLEButtonLED)
- Callback LED (//www.arduino.cc/en/Tutorial /Genuino101CurieBLECallbackLED)
- LED (//www.arduino.cc/en/Tutorial /Genuino101CurieBLELED)

Reference Home (//www.arduino.cc/en/Reference/HomePage)

Corrections, suggestions, and new documentation should be posted to the Forum (http://arduino.cc /forum/index.php/board,23.0.html).

Share

NEWSLETTER

Enter your email to sign up →

Careers (//www.arduino.cc/Careers)

(https://twitter.com/arduino) (https://www.facebook.com/official.arduino)

(https://plus.google.com/+Arduino) (https://www.flickr.com/photos/arduino_cc)

(https://youtube.com/arduinoteam)