Programming on Raspberry Pi

Bluetooth Low Energy

https://goo.gl/9KwHGi

Bluetooth Low Energy (BLE)

- BLE is a wireless personal area network technology intended to provide **reduced power** consumption and **reduced cost** while maintaining similar communication range as compared to Classic Bluetooth.
- BLE uses the same 2.4 GHz radio frequencies as Classic Bluetooth, allows dual-mode devices to share a single radio antenna.

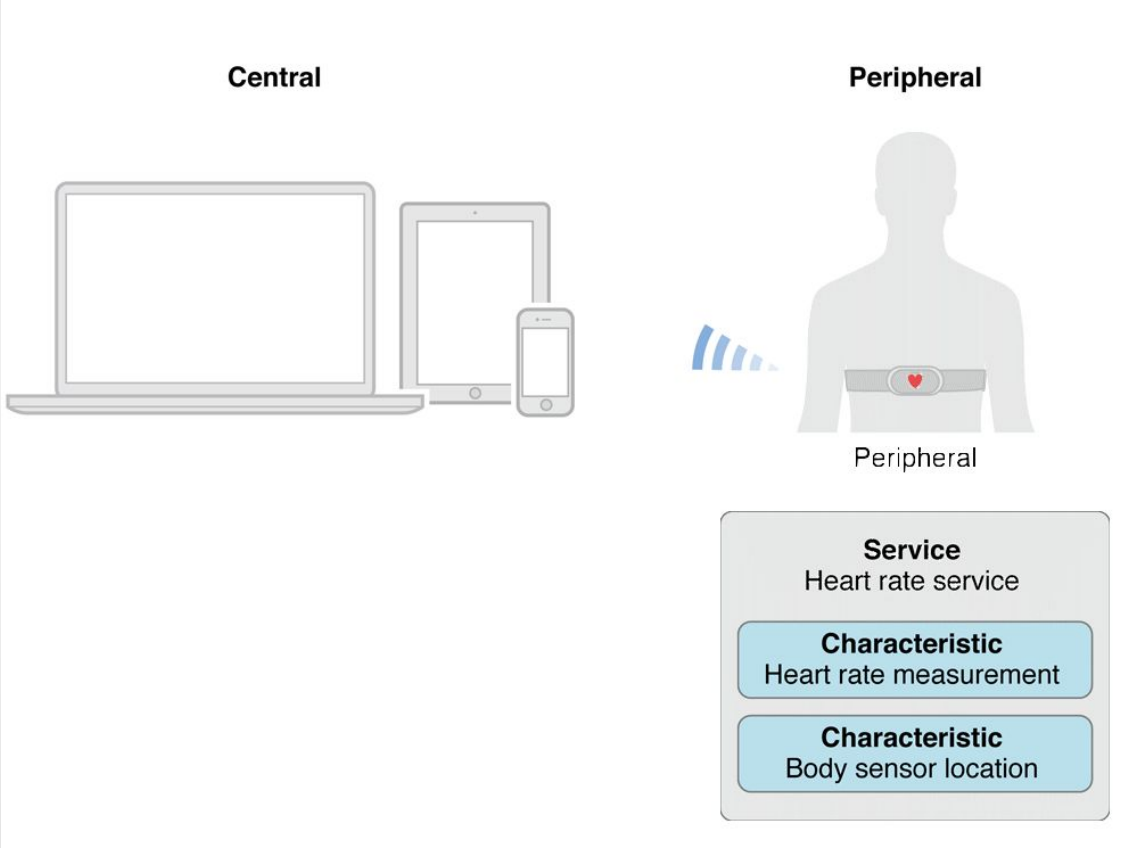
BLE Central vs Peripheral role

- BLE **Generic Access Profile (GAP)** defines various roles for BLE devices which determines how two devices can or can not interact with each other.
- A BLE connection can only be established between a device in the central role and a device in the peripheral role:
  - The device in the **peripheral** role broadcasts BLE advertisements about themselves so that a connection can be established.
  - The device in the **central** role scans and looking for these advertisement.

BLE GATT Terminology

- BLE devices use the **Generic Attribute Profile (GATT)**, a general specification for sending and receiving short pieces of data known as **attributes** over a BLE link.
- A GATT **characteristic** is a basic data element used to construct a GATT service, e.g. the heart rate of a heart rate sensor with a value of 72.
- A GATT **service** is a collection of GATT characteristics that operate together to perform a particular function, e.g. the heart rate service with a heart rate measurement characteristic and a body sensor location characteristic.

BLE GATT Example



Source:apple.com

GATT UUID

- Each GATT attribute (Service, Characteristic) is uniquely identified by a **Universally Unique Identifier (UUID)**, which is a standardized 128-bit format for a string ID.
- The attribute **handle** is a unique 16-bit identifier for each attribute on a particular GATT server.
- The GATT protocol support these operations:
  - Discover UUIDs for all primary services
  - Find a service with a given UUID
  - Discover UUIDs for all characteristics for a given service
  - Find characteristic with a given UUID

GATT Specification

https://developer.bluetooth.org/gatt/Pages/default.aspx

SpecificationName	SpecificationType	AssignedNumber	SpecificationLevel
Alert Notification Service	org.bluetooth.service.alert_notification	0x1811	Adopted
Automation IO	org.bluetooth.service.automation_io	0x1815	Adopted
Battery Service	org.bluetooth.service.battery_service	0x180F	Adopted
Blood Pressure	org.bluetooth.service.blood_pressure	0x1810	Adopted
Body Composition	org.bluetooth.service.body_composition	0x181B	Adopted
Bond Management	org.bluetooth.service.bond_management	0x181E	Adopted
Continuous Glucose Monitoring	org.bluetooth.service.continuous_glucose_monitoring	0x181F	Adopted
Current Time Service	org.bluetooth.service.current_time	0x1805	Adopted
Cycling Power	org.bluetooth.service.cycling_power	0x1818	Adopted
Cycling Speed and Cadence	org.bluetooth.service.cycling_speed_and_cadence	0x1816	Adopted
Device Information	org.bluetooth.service.device_information	0x180A	Adopted
Environmental Sensing	org.bluetooth.service.environmental_sensing	0x181A	Adopted
Generic Access	org.bluetooth.service.generic_access	0x1800	Adopted
Generic Attribute	org.bluetooth.service.generic_attribute	0x1801	Adopted

GATT Characteristic

- GATT Characteristic operations include:
  - A value may be **read** either with the characteristic UUID, or by a handle value
  - A value may be **written** to a characteristic either with the characteristic UUID, or by a handle value
  - A **notification** for a characteristic may be requested, the value will be transferred whenever it becomes available

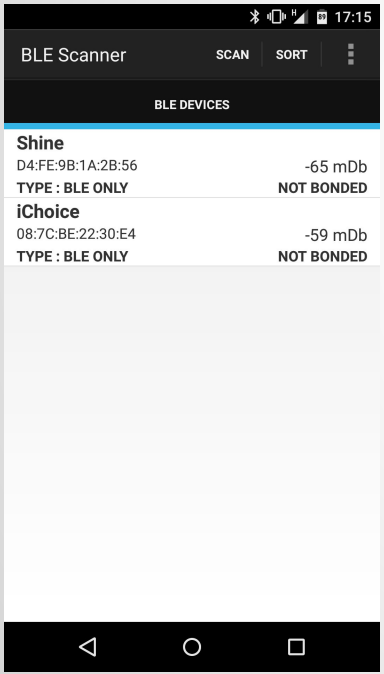
BLE with Raspberry Pi

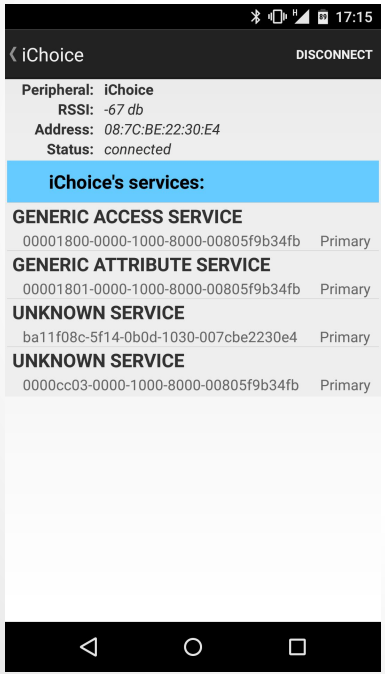
- Raspberry Pi 3 has built-in BLE capability
- Adding BLE capability to other Raspberry Pi models by adding a **Bluetooth 4.0 USB dongle**
- RPi can take the role of
  - Central Device - controlling BLE light bulb, reading sensor data etc.
  - Peripheral Device - communicating with Android device which is usually in Central role as Peripheral role only available in Android 5.0.

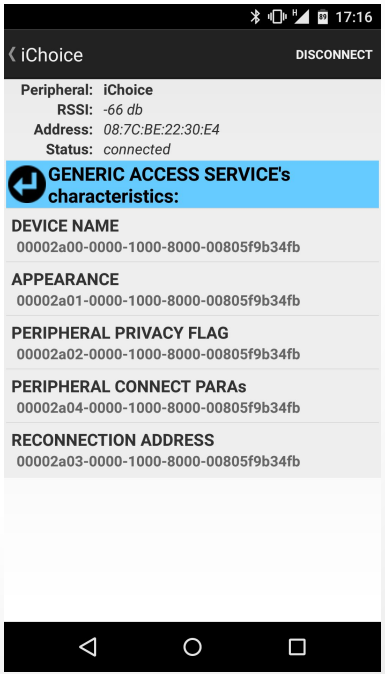
Android Device as BLE Central Device

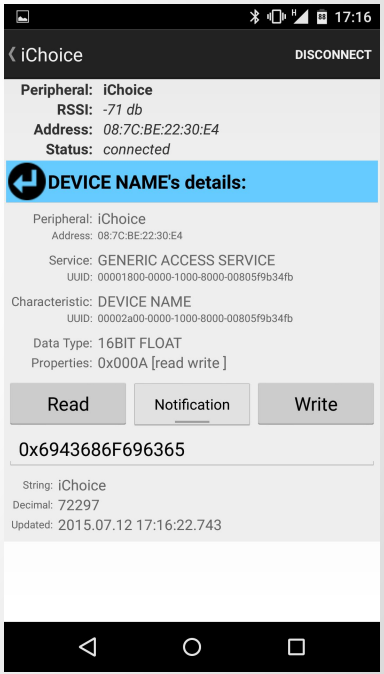
- Install BLE Scanner on Android device with BLE support

https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner









Raspberry Pi as BLE Peripheral Device

- Using Bleno
  - A Node.js module for implementing BLE (Bluetooth Low Energy) peripherals
  - Details at https://github.com/sandeepmistry/bleno
  - Install prerequisites

sudo apt-get install bluetooth libbluetooth-dev libudev-dev

  - Install Bleno via Node.js Package Manager (npm)

npm install bleno

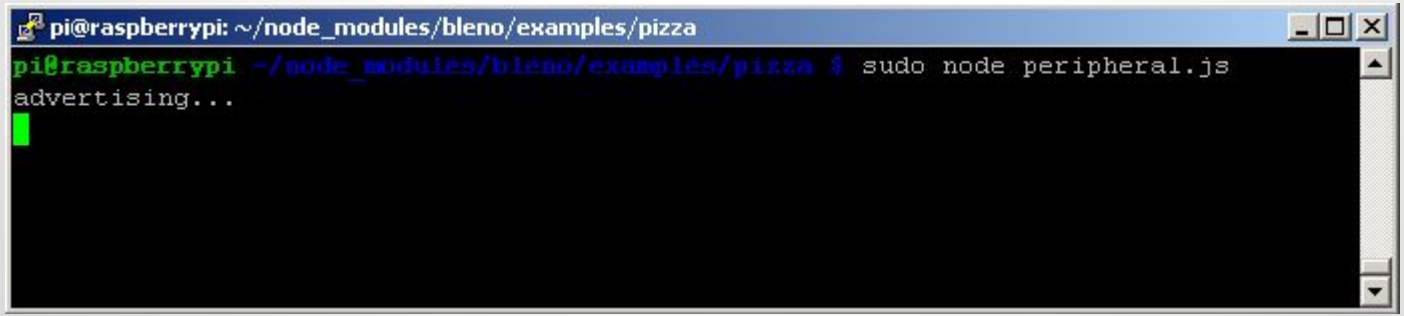
Bleno Example

- Plug USB CSR 4.0 BLE dongle into Raspberry Pi
- Change directory to Bleno Pizza example

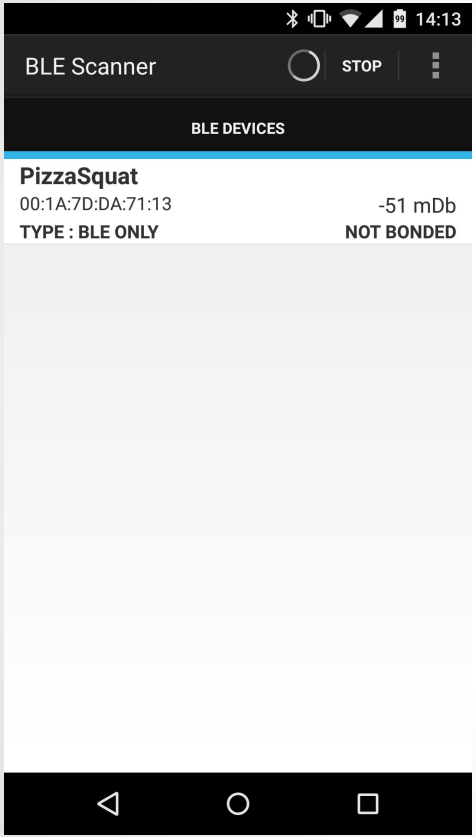
  cd /home/pi/node_modules/bleno/examples/pizza

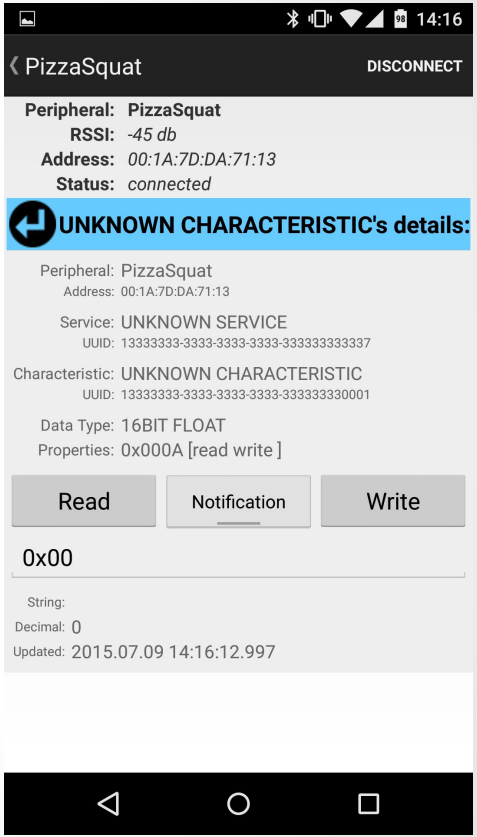
- Run the example

  sudo node peripheral.js



Bleno Example





Capstone Project BLE Communication Example

- BLE communication between Raspberry Pi as peripheral and Android as central
- Download from

  https://github.com/wenjiun/capstonedemo
- To run the Node.js script on Raspberry Pi, plug in a supported USB BLE dongle and run

  sudo node capstone.js
- To run the Android App, open the Android project in Android Studio, edit the BLE MAC address and run

Using Bleno

```
var bleno = require('bleno');
```

Using Bleno - setup GATT characteristic

```
var CapstoneCharacteristic = bleno.Characteristic;
var capstoneCharacteristic = new CapstoneCharacteristic({
    uuid: '13333333333333333333333330001',
    properties: ['read', 'write'],
    descriptors: [new bleno.Descriptor({
        uuid: '2901',value: 'Capstone demo string'
    })],
    …..
```

Using Bleno - setup GATT characteristic - handling read

```
….
onReadRequest: function(offset, callback) {
    var data = new Buffer(1);
    data.writeUInt8(value, 0);
    console.log('Data read: ' + value);
    callback(this.RESULT_SUCCESS, data);
},
….
```

Using Bleno - setup GATT characteristic - handling write

```
    ….
    onWriteRequest: function(data, offset, withoutResponse,
        callback) {
        value = data.readUInt8(0);
        console.log('Data written: ' + value);
        callback(this.RESULT_SUCCESS);
    }
});
```

Using Bleno - setup GATT service

```
var CapstoneService = bleno.PrimaryService;
var capstoneService = new CapstoneService({
    uuid: '13333333333333333333333333333337',
    characteristics: [
        capstoneCharacteristic
    ]
});
```

Using Bleno - advertising

```
bleno.startAdvertising(
    'Capstone Project',
    ['13333333333333333333333333333337'],
    function(err) {
        if (err) {
            console.log(err);
        }
    });
```

Using Bleno - waiting for connection

```
bleno.on('stateChange', function(state) {
    if (state === 'poweredOn') {
        // start advertising
    }  else {
        bleno.stopAdvertising();
    }
});
```

Using Bleno - start GATT services

```
bleno.on('advertisingStart', function(err) {
    if (!err) {
        console.log('Advertising...');
        bleno.setServices([capstoneService]);
    }
});
```

Bluetooth Low Energy on Android

Capstone Project

- a.k.a. Bluetooth Smart, adopted in Bluetooth 4.0
- low power consumption & low cost for wireless personal area network in health-care, security etc.
- Android supports Bluetooth Low Energy since JellyBean 4.3
  - Central role – consume BLE data
  - Peripheral role (Android 5.0) – to advertise BLE data

- Requires the permissions in Android Manifest
  - **android.permission.BLUETOOTH**
  - **android.permission.BLUETOOTH_ADMIN**
- Filter in Android Manifest
  - android.hardware.bluetooth_le

Detect BLE support & obtain the Bluetooth Adapter

```
boolean hasBLE =
    getPackageManager().hasSystemFeature(
PackageManager.FEATURE_BLUETOOTH_LE)
BluetoothManager bluetoothManager =
    (BluetoothManager)
    getSystemService(Context.BLUETOOTH_SERVICE);
BluetoothAdapter mBluetoothAdapter =
    bluetoothManager.getAdapter();
```

- With a BluetoothAdapter, call **startLeScan(LeScanCallback mLeScanCallback)** to start scanning for BLE devices.
- The scan results are returned in the callback
- Call **stopLeScan(LeScanCallback mLeScanCallback)** to stop
  - After a desired device is found
  - After a limited duration (e.g. 10 secs)

BLE **LeScanCallback** interface

- Has to implement **onLeScan(BluetoothDevice device, int rssi, byte[] scanRecord)**
- Obtain the device
  - friendly name with device.**getName()**
  - hardware address with device.**getAddress()**
  - Bluetooth device type with device.**getType()**
    - classic, LE, dual (classic + LE)

BLE **LeScanCallback** interface

- Obtain the RSSI value of the remote device which is useful to estimate how far is the remote device
- Obtain the content of the advertisement record advertised by BLE peripheral device, e.g. iBeacon

Obtain a GATT Server

- With a BluetoothDevice by:
  - Selecting one from the list of devices returned in onLeScan
  - Calling mBluetoothAdapter.**getRemoteDevice(** String address**)**
- BluetoothGatt mBluetoothGatt = device.**connectGatt(**Context context, **boolean** autoConnect, BluetoothGattCallback callback**)**

In BluetoothGattCallback

- Need to call **runOnUiThread**(**new** Runnable()) to modify the UI in BluetoothGattCallback
- Override **onConnectionStateChange(**BluetoothGatt gatt**,** **int** status**,** **int** newState**)**
  - Check whether newState == BluetoothProfile.*STATE_CONNECTED*
  - Call mBluetoothGatt.**discoverServices()** to discover the services and the corresponding characteristics and descriptors

- Override **onServicesDiscovered(**BluetoothGatt gatt, **int** status**)**
  - gatt.**getServices()** to return a list of GATT services List<BluetoothGattService>
  - Iterate through the list to obtain individual service

```
for(BluetoothGattService gattService:gattServices)
{
        gattService.getUuid();
       ...
}
```

gattService.**getCharacteristics()** to return a list of GATT characteristics for a given service

List<BluetoothGattCharacteristic>

- Iterate through the list to obtain individual GATT characteristic

```
for(BluetoothGattCharacteristic gattChar:gattChars)
{
        gattChar.getUuid();
        ...
}
```

For each BluetoothGattCharacteristics call gattChar.**getProperties()** to check whether the BluetoothGattCharacteristics

- Can be read
- Can be written
- Can notify

To read BluetoothGattCharacteristics

- Call gatt.**readCharacteristic(gattChar)**
- In BluetoothGattCallback, override

onCharacteristicRead(BluetoothGatt gatt,

BluetoothGattCharacteristic characteristic, **int** status**)**

- Call characteristic.**getValue()** to get the stored value in byte[]

To write BluetoothGattCharacteristics

- Obtain the byte array to be written

```
byte[] values =  …;
```

- Store the byte array into a GATT Characteristics

```
gattChar.setValue(values);
```

- Call **boolean** success = gatt.**writeCharacteristic**(gattChar);
- Check whether success is **true** for a successful write

To enable notification from BluetoothGattCharacteristic

- Call gattChar.**getDescriptors()** to return a list of GATT descriptors for a given characteristic

  List<BluetoothGattDescriptor>
- Iterate through the list to obtain individual descriptor

  for(BluetoothGattDescriptor gattDesc:gattDescs)

{

    gattDesc.getUuid();

    ...

}

To enable notification from BluetoothGattCharacteristic

- Call gattDesc.setValue(BluetoothGattDescriptor.*ENABLE_NOTIFICATION_VALUE*);
- Call gatt.**writeDescriptor**(gattDesc);
- Then call gatt.**setCharacteristicNotification**(gattDesc, **true**);

- To receive notification from BluetoothGattCharacteristic
  - In BluetoothGattCallback, override **onCharacteristicChanged**(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic)
  - Call characteristic.**getValue()** to get the stored value in byte[]

https://goo.gl/9KwHGi

Thank you