

Mobile Computing Bluetooth Low Energy on Android Smartphones

CC BY-SA, T. Amberg, FHNW

Slides: tmb.gr/mc-cen



[Source](#) on GDocs

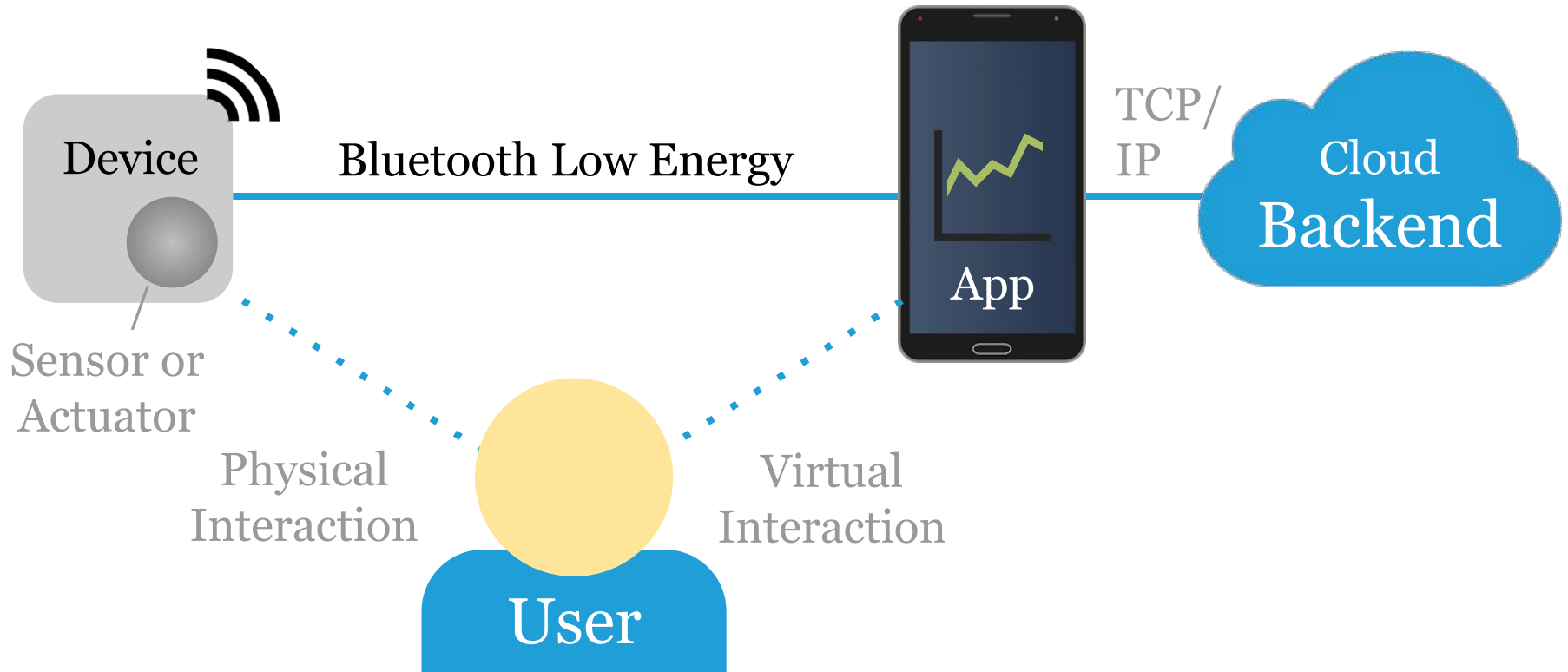
Overview

These slides introduce *BLE on Android smartphones*.

How to implement the central role, as a GATT client.

Scanning, reading, writing and getting notifications.

Reference model



BLE on Android

Android has built-in **BLE support*** since **API level 18**.

A smartphone can take the central or peripheral role.

The Android SDKs BLE API is rather tricky to use.

Luckily there are some good 3rd party libraries.

*Also Bluetooth Classic for Headset, A2DP, HDP.

Adding permissions

Add these [permissions](#) to your app manifest file:

`android.permission.BLUETOOTH` 18+

`android.permission.BLUETOOTH_ADMIN` 18+

Also, as BLE discovery is privacy critical*, add:

~~`android.permission.ACCESS_COARSE_LOCATION` 23+~~

`android.permission.ACCESS_FINE_LOCATION` 29+

*A scan reveals nearby devices/beacons, often such a "fingerprint" is all it takes to derive a user's location. 5

Checking hardware capability

Indicate usage of BLE in your app manifest file:

```
<uses-feature  
    android:name="android.hardware.bluetooth_le"  
    android:required="true" />
```

And test if BLE is available on the current system:

```
if (getPackageManager().hasSystemFeature(  
    PackageManager.FEATURE_BLUETOOTH_LE)) { ... }
```

Setting up a BluetoothAdapter

Get the **BluetoothAdapter** via **BluetoothManager**:

```
final BluetoothManager bluetoothManager =  
    (BluetoothManager) getSystemService(  
        Context.BLUETOOTH_SERVICE);  
BluetoothAdapter bluetoothAdapter =  
    bluetoothManager.getAdapter();
```

Ensuring Bluetooth is enabled

If Bluetooth is not enabled, ask the user to enable it:

```
if (bluetoothAdapter != null &&  
    bluetoothAdapter.isEnabled()) { ...  
} else {  
    Intent enableBtIntent = new Intent(  
        BluetoothAdapter.ACTION_REQUEST_ENABLE);  
    startActivityForResult(  
        enableBtIntent, REQUEST_ENABLE_BT);  
}
```


Getting a scanner

Get a **BluetoothLeScanner** to scan for BLE devices:

```
BluetoothLeScanner scanner =  
    bluetoothAdapter.getBluetoothLeScanner();
```

The **Handler** runs a *Runnable** on another thread:

```
static final long SCAN_PERIOD_MS = 10000;  
Handler handler =  
    new Handler(Looper.getMainLooper());
```

*See slide *Starting a Scan*, p.11

Asking for location permission (again)

```
String[] permissions = new String[]{  
    Manifest.permission.ACCESS_FINE_LOCATION  
};
```

```
ActivityCompat.requestPermissions(  
    MainActivity.this, permissions, 0);
```

```
@Override  
public void onRequestPermissionsResult (...) {  
    scan();  
}
```

Starting a scan

```
void scan() {  
    handler.postDelayed(new Runnable() {  
        @Override  
        public void run() {  
            scanner.stopScan(scanCallback);  
        }  
    }, SCAN_PERIOD_MS);  
    scanner.startScan(scanCallback);  
}
```

Getting scan results

```
ScanCallback scanCallback = new ScanCallback() {  
    @Override  
    public void onScanResult(  
        int callbackType, ScanResult result) {  
        super.onScanResult(callbackType, result);  
        Log.d(TAG, result.getDevice());  
    }  
};
```

Hands-on, 10': Android BLE scanner

Build and run the [MyBleScannerApp example code](#).

Check the Logcat output with filter "MainActivity".

If it works, you should see BLE devices around you.

Test the app by disabling Bluetooth, location, etc.

Done? Add a button to the UI to start a scan.

Heart rate service

This service is intended for fitness heart rate sensors:

Heart Rate Service UUID (16-bit): 0x**180D**

This service includes the following characteristics:

Heart Rate Measurement UUID: 0x**2A37** [N]

Body Sensor Location UUID: 0x**2A38** [R]

Heart Rate Control Point UUID: 0x**2A39** [W]*

Standard service, defined by the Bluetooth SIG.

Base UUID for registered services

For a custom service UUID we define all 128-bits, e.g.
0x6E400001-B5A3-F393-E0A9-E50E24DCCA9E

For well known, [registered services](#) the *base UUID* is*
0x00000000-0000-1000-8000-00805F9B34FB

A 16-bit service "UUID", e.g. 0x**180D** corresponds to:
0x0000180D-0000-1000-8000-00805F9B34FB

*See [Bluetooth spec v5.3](#), p.1181 ff.

Connecting to discover GATT services

```
final BluetoothGattCallback gattCallback =  
    new BluetoothGattCallback() {  
        @Override  
        public void onConnectionStateChange(...) { ... }  
            gatt.discoverServices();  
        } ...  
    };  
  
BluetoothGatt gatt = device.connectGatt(  
    this, /* reconnect */ true, gattCallback);
```


Reading a GATT characteristic

```
final BluetoothGattCallback gattCallback =  
    new BluetoothGattCallback() { ...  
    @Override  
    public void onServicesDiscovered(...) { ...  
        BluetoothGattCharacteristic characteristic  
            = gattService.getCharacteristic(uuid);  
        gatt.readCharacteristic(characteristic);  
    } ...  
};
```

Getting a GATT characteristic value

```
final BluetoothGattCallback gattCallback =  
    new BluetoothGattCallback() { ...  
    @Override  
    public void onCharacteristicRead(...) { ...  
        if (characteristic.getUuid().equals(...)) {  
            byte[] value = characteristic.getValue();  
        }  
    } ...  
};
```

Writing a GATT characteristic

```
final BluetoothGattCallback gattCallback =  
    new BluetoothGattCallback() { ...  
    @Override  
    public void onServicesDiscovered(...) { ...  
        BluetoothGattCharacteristic characteristic  
            = gattService.getCharacteristic(uuid);  
        characteristic.setValue(value, format, 0);  
        gatt.writeCharacteristic(characteristic);  
    } ... };
```

Getting GATT characteristic write status

```
final BluetoothGattCallback gattCallback =  
    new BluetoothGattCallback() { ...  
    @Override  
    public void onCharacteristicWrite(...) { ...  
        if (characteristic.getUuid().equals(...)) {  
            Log.d(TAG, "write, status = " + status);  
        }  
    } ...  
};
```

Enabling GATT char. notifications

@Override

```
public void onServicesDiscovered(...) { ...  
    gatt.setCharacteristicNotification(..., true);  
    UUID configUuid = UUID.fromString("2902");  
    BluetoothGattDescriptor descriptor =  
        characteristic.getDescriptor(configUuid);  
    descriptor.setValue(BluetoothGattDescriptor.  
        ENABLE_NOTIFICATION_VALUE);  
    gatt.writeDescriptor(descriptor);  
}
```

Getting GATT characteristic notifications

```
final BluetoothGattCallback gattCallback =  
    new BluetoothGattCallback() { ...  
        @Override  
        public void onCharacteristicChanged(...) { ...  
            if (characteristic.getUuid().equals(...)) {  
                byte[] value = characteristic.getValue();  
            }  
        } ...  
    };
```

Hands-on, 10': Android BLE central

Build and run the [MyBleCentralApp](#) example code.

Use the nRF52840 with [HRM BLE peripheral code](#).

Check the Logcat output with filter "MainActivity".

If it works, you should see heart rate measurements.

Done? Read the Android app source code in detail.

What's missing in the example code

BLE related code should be moved into a **Service**.

Starting a scan should be triggered by the user*.

Discovered devices should be presented in a list.

Read, write, etc. operations should be queued.

*On **API level 26+**, try **companion device pairing**.

Hands-on, 10': Android BLE issues

It looks easy, but BLE on Android has many issues.

Read some of the tips how to [make it actually work](#).

Also check this [list of issues](#) to get an impression.

iOS BLE seems more stable. Why could that be?

Done? Find some stats about the Android OS.

BLE libraries from 3rd parties

Writing robust apps based on the BLE API is hard.

There are a number of 3rd party libraries to fix this:

The [Nordic Android BLE library](#) is written in Java.

Nordic also has a [Kotlin BLE library](#) for Android*.

[RxAndroidBle library](#) uses the [RxJava](#) framework.

*Uses Kotlin coroutines, license still unclear.

Hands-on, 10': nRF Blinky app example

[Android nRF Blinky](#) is a complete BLE app example.

It shows how to use the [Nordic Android BLE library](#).

Read the library docs and study the app source code.

Which parts become easier by using this library?

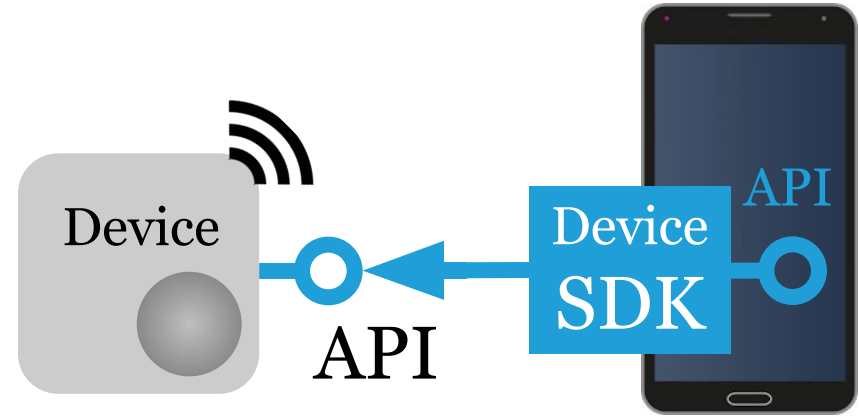
Done? Sketch the BLE API of the peripheral.

Device API vs. SDK

A *device API* specifies how to talk to the device, from any client (here via BLE).

A platform specific *device SDK* simplifies integration.

E.g. *iOS device SDK* to talk to a device API from iOS.



Battery Service

Battery Level [R]

vs.

```
p = ble.conn(addr);  
b = sdk.getBatt(p);  
x = b.getLevel();
```

Summary

The Android BLE API allows to implement a central.
Scan, read, write and notify operations use callbacks.
Service and characteristic UUID define the interface.
3rd party libraries can help to write robust BLE code.

Challenge: nRF Toolbox plugin

nRF Toolbox is a "container app" for BLE demos.

Build and run the [nRF Toolbox app source code](#).

Write a plugin for a [custom SHT30 BLE service](#).

Test the app with a [nRF52840 BLE peripheral](#).

Done? Create a custom icon for your service.

Feedback or questions?

Join us on [MSE TSM MobCom](#) in MS Teams

Or email thomas.amberg@fhnw.ch

Thanks for your time.