Android 4.3 and Bluetooth SMART

Ben Von Handorf

M3 CONFERENCE

OUR GENEROUS SPONSORS

GOLD

CHASE

CareWorksTech

Compuware

Microsoft

QSI Quick Solutions Inc.

NOKIA

telerik deliver more than expected

SILVER

OCLC The world's libraries. Connected.

Fine CITIZENS

pluralsight hardcore developer training

Grange Insurance

MARKETING WORKS

manta

KEYNOTE

LeadingEDJE

adept

Who am I?

Developer (mostly in the Microsoft stack) since 1995

Android since 2010

Multiple apps in multiple industries

Still learning!

Goals

Explain the basics of Bluetooth SMART (BLE)

Discuss the API details

Briefly discuss compatibility and limitations of the Android 4.3 Bluetooth SMART API

Bluetooth SMART (BLE)

Sister protocol to Bluetooth Classic, but not compatible. Adopted into Bluetooth standard in 2010

Uses the same frequencies, so many devices support both protocols

Bluetooth Low Energy (BLE) is the technical name, marketed as Bluetooth SMART

Protocol Benefits

- Low Power Usage
- Low bandwidth and shorter range
- Low latency (~6ms vs 100ms)
- No pairing between devices

Optimal for sensors & simple data transmission where ease of connectivity is paramount

Sample Applications

Healthcare and Excercise

- Heart Rate Sensor
- Nike+ sensors

Temperature, Humidity, Gyroscope

- TI SensorTag

Proximity

- Key Loss detectors
- iBeacon

Terminology

GATT - Generic Attribute Profile

Underlies all Bluetooth SMART operations, so it shows up throughout the API

(Think of BluetoothGatt as a synonym for Bluetooth SMART)

Server/Peripheral - A device which provides Bluetooth SMART services

Client/Central - A device which consumes those services (i.e. our Android devices)

Getting Connected

1. Get a BluetoothAdapter
2. Look for available Bluetooth SMART devices
3. Connect to a device
4. Scan for Services that the device provides

Getting a Bluetooth Adapter

```
BluetoothManager bluetoothManager = (BluetoothManager) getSystemService
(BLUETOOTH_SERVICE);

BluetoothAdapter bluetoothAdapter = bluetoothManager.getAdapter();

// Ensures Bluetooth is available on the device and it is enabled. If not,
// displays a dialog requesting user permission to enable Bluetooth.
if (bluetoothAdapter == null || !bluetoothAdapter.isEnabled()) {
     Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
     startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
} else {
     //Scan for devices...
}
```

Scan for Devices

```
private BluetoothAdapter.LeScanCallback _bluetoothLeScanCallback = new
BluetoothAdapter.LeScanCallback() {
    @Override
    public void onLeScan(BluetoothDevice bluetoothDevice, int i, byte[]
bytes) {
        Log.v(TAG, String.format("Scan found device: %s at %s",
bluetoothDevice.getName(), bluetoothDevice.getAddress()));
        //Connect to the device if it is interesting to us
    }
};

//Start the scan with the BluetoothAdapter you obtained earlier
bluetoothAdapter.startLeScan(_bluetoothLeScanCallback);
```

Connecting

```
//Once we have a device that we're interested in, get
//GATT connection.

_bluetoothGatt = bluetoothDevice.connectGatt(context, false,
_bluetoothGattCallback);
```

Scan for Services

```
//Detect the device becoming connected:
@Override
public void onConnectionStateChange(BluetoothGatt gatt, int status, int
newState) {
    super.onConnectionStateChange(gatt, status, newState);
    if (newState == BluetoothProfile.STATE_CONNECTED) {
        gatt.discoverServices();
    }
}
```

Analyze the available services

```
@Override
public void onServicesDiscovered(BluetoothGatt gatt, int status) {
    super.onServicesDiscovered(gatt, status);
    for (BluetoothGattService service : gatt.getServices()) {
        //Examine the available services on the device

    }
}
```

What's a Service?

A Bluetooth Service is a known collection of Characteristics that, together, represent a set of behaviors

Many well known and documented Services

Manufacturers can create their own (or ignore the specification)

We’re connected, let’s do something

When we find a Service we understand, we can start using it

Interacting with a service consists of reading and writing Characteristics

Writing to a Characteristic

```
//Find the service and characteristic you care about
BluetoothGattService service =
    _bluetoothGatt.getService(Sensors.IR_SENSOR.SERVICE);

BluetoothGattCharacteristic characteristic =
    service.getCharacteristic(Sensors.IR_SENSOR.CONFIG);

//Queue up the value you'd like to set
characteristic.setValue(new byte[]{0x01});

//Request that the value be written
if (!_bluetoothGatt.writeCharacteristic(characteristic)) {
    Log.v(TAG, "Failed writeCharacteristic");
}
```

Reading from a Characteristic

```
BluetoothGattService service =
    _bluetoothGatt.getService(Sensors.IR_SENSOR.SERVICE);

BluetoothGattCharacteristic characteristic =
    service.getCharacteristic(Sensors.IR_SENSOR.DATA);

//Request that the value be read from the characteristic
if(!_bluetoothGatt.readCharacteristic(characteristic)){
    Log.v(TAG, "Read characteristic failed");
}
```

Receiving the Characteristic value

```
@Override
public void onCharacteristicRead(BluetoothGatt gatt,
    BluetoothGattCharacteristic characteristic, int status) {
    super.onCharacteristicRead(gatt, characteristic, status);

    if (Sensors.IR_SENSOR.DATA.equals(characteristic.getUuid())) {
        double ambient = extractAmbientTemperature(characteristic);
        double target = extractTargetTemperature(characteristic, ambient);

        Log.v(TAG, String.format("Got IR Sensor Data: %.1f %.1f", ambient,
target));
    }
}
```

Wait, where is that code?

All operations in Bluetooth SMART are asynchronous

Results of all operations are delivered to the BluetoothGattCallback class registered when we connected to the device

```
//Once we have a device that we're interested in, get
//GATT connection.

_bluetoothGatt = bluetoothDevice.connectGatt(context, false,
                 _bluetoothGattCallback);
```

BluetoothGattCallback

BluetoothGattCallback Methods:

onConnectionStateChange

onServicesDiscovered

onCharacteristicRead

onCharacteristicWrite

onCharacteristicChanged

onDescriptorRead

onDescriptorWrite

onReliableWriteCompleted

onReadRemoteRssi

Inconvenient.
But it gets worse.

If you start a new BluetoothGatt operation before the previous one succeeds, both may fail.

You must wait for commands to complete before issuing the next command.

Dealing with Async

You cannot start a new command before the previous one completes

Options:

- State machine - Transition states at known points
- Command Queue - Each command completing triggers the next command in the queue

A brief, raw GATT Example

Uses a simple state machine to push the device through the various states

State transitions are triggered in the BluetoothGattCallback

BluetoothGattCallback class is very smart and very specific

[demo]

Creating a better interface

A device-type specific class can determine if the device is available and can provide strongly typed access to services & characteristics

Command queue is embodied in the device class

Commands only need to know if they have completed when certain events occur

(Note! This API was created as a teaching tool. I make no assertions about applicability to a production app)

[demo]

BluetoothDeviceWrapper example

```
public void onDeviceReady(Device device) {
_sensorTagDevice.getIrService().getDataCharacteristic()
    .setCharacteristicListener(new Characteristic.CharacteristicListener()
{
        public void onValueChanged(Characteristic characteristic) {
            irDataCharacteristic irDataCharacteristic =
(IrDataCharacteristic) characteristic;
            displayTemperatureData(irDataCharacteristic);
        }
    });

    //Enable the IR Sensor
    _sensorTagDevice.getIrService().getConfigCharacteristic().enable();
    _sensorTagDevice.getIrService().getDataCharacteristic().read();
    _sensorTagDevice.getIrService()
         .getDataCharacteristic().enableNotification();
}
```

Notifications

Many (not all) characteristics support server-initiated notification

Notifications can either be timing based or change based, implementation dependent

Prevents the need to poll, but may not match the necessary frequency

Example:

IR Sensor Notification: ~1000ms

IR Sensor Fast Polling: ~20ms

Key API Points

Almost all APIs are async through the `BluetoothGattCallback` interface

Operations must be performed serially

A real implementation should not be handled by the Activity, but rather by a Service to eliminate lifetime issues. (Sample code != Production code)

API Minutiae

Using Bluetooth SMART on Android requires the following permissions:

- `android.permission.BLUETOOTH`
- `android.permission.BLUETOOTH_ADMIN`

You can check for Bluetooth SMART support with this manifest feature:

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

(This will also provide Google Play store filtering so your app will not be installed on incompatible devices)

API Limitations

Currently, **Android** API is limited to **4** characteristic notifications

Device support is not universal, even among new devices.

- Nexus devices are your best bet

Connection is not as stable as we'd like

- Nexus 7 seems more stable than the Nexus 4

Devices

Some good development devices

- TI SensorTag
- Many Polar Heart Rate Monitors
- Arduino shields for the Makers
- Many others
  - Look for Bluetooth SMART support
  - Look for support on the iPhone 4S and higher

Resources

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

Developer Guide - http://developer.android.com/guide/topics/connectivity/bluetooth-le.html

Google IO - Best Practices Video

https://developers.google.com/events/io/sessions/326240948

iOS Examples

- iOS has a similar API
- Good for cross-referencing UUIDs

Another BLE Talk Today!

BLUETOOTH LOW ENERGY: THE RESURGENCE

Eric McGary

13:10

Great Hall Meeting Room 3 *(that would be this room)*

Thanks!

Ben Von Handorf

@benvonhandorf

ben@benvonhandorf.com

Code at https://github.com/benvonhandorf

Slides at:

https://docs.google.com/presentation/d/1gnLivZO87kbKEQF0Jisi0XZkLdOOKkNhlWhnn1kAxQ4/edit?usp=sharing

Thanks!

Questions?