SMART Technologies
Introducing bluetooth low energy and iBeacon
In real life you may call me
Frederick Bousson

Competence Leader Mobile @ Ordina
Smartphone as life’s remote control
Focus on Software Development

Frederick.Bousson@ordina.be  fbousson
Bluetooth Low Energy
Smart!

• Bluetooth Classic
• Bluetooth Smart
• Bluetooth Smart Ready
BLE not backward-compatible with Classic Bluetooth.

Bluetooth 4.0 permits devices to implement either or both LE and Classic.

BLE uses the same 2.4 Ghz radio frequencies as Classic.

Dual mode devices can share single radio antenna.

BLE uses a simpler modulation system.
Branding

• **Bluetooth Smart Ready** indicates
  • a dual-mode device
  • whose hardware is compatible with both Classic and LE Bluetooth peripherals.

• **Bluetooth Smart** indicates
  • an LE-only device, typically a battery-operated sensor,
  • which requires either a SMART Ready or another SMART device in order to function.
Bluetooth®

Wireless devices, streaming rich content, like video and audio.

Bluetooth® SMART READY

Devices that connect with both. The center of your wireless world

Bluetooth® SMART

Sensor devices, sending small bits of data, using very little energy.
Advantages

• **Low power requirements**, operating for "months or years" on a button cell.

• **Small size and low cost.**

• **Compatibility** with a large installed base of mobile phones, tablets and computers.
Applicable in Multiple areas

- Smart home solutions
- Healthcare & Fitness
- Security
- Home entertainment
- Toys and Remote controls
- …
Smart home solutions
Home entertainment

http://www.youtube.com/watch?v=oWu9TFJjHaM
Healthcare & fitness
Smart assistance for men
And of course

If it flies …

http://www.youtube.com/watch?v=q9bpp7zmM_A
Devices

- **Windows Phone 8:**
  - Nokia Lumia (520, 525, 620, 625, 820, 920, 925, 928, 1020, 1320, 1520)

- **Android 4.3 and later**
  - (HTC, LG, Motorola, Sony, Samsung, ...)

- **Apple iOS 5+**
  - (iPhone 4S+, iPad 3rd gen, iPod Touch 5th gen)
Some figures

<table>
<thead>
<tr>
<th>Technical Specification</th>
<th>Classic</th>
<th>BLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance / Range</td>
<td>100m</td>
<td>50m</td>
</tr>
<tr>
<td>Latency</td>
<td>100ms</td>
<td>6ms</td>
</tr>
<tr>
<td>Power consumption</td>
<td>1 as the reference</td>
<td>0,01 to 0,5 (depending on use case)</td>
</tr>
</tbody>
</table>
Software model

- All Bluetooth low energy devices use the Generic Attribute Profile (GATT).
- The Application programming interface offered by a Bluetooth LE-aware operating system will typically be based around GATT concepts.
- GATT has the following terminology:
Client

A device that initiates GATT commands and requests, and accepts responses, for example a computer or smartphone.
Server

A device that receives GATT commands and requests, and returns responses, for example a temperature sensor.
Peripheral (Slave)

A peripheral can advertise, to let other devices know that it’s there.
Central (Master)

Only a central can send a connection request to establish connection.
Roles

- Client <-> Server
- Peripheral <-> Central
Characteristic

A data value transferred between client and server.

*for example the current battery voltage.*
A collection of related characteristics, which operate together to perform a particular function.

For instance, the Health Thermometer services includes characteristics for a temperature measurement value, and a time interval between measurements.
Descriptor

A descriptor provides additional information about a characteristic.

*For instance, a temperature value characteristic may have an indication of its units (e.g. Celsius), and the maximum and minimum values which the sensor can measure.*

Descriptors are optional - each characteristic can have any number of descriptors.
Identifiers

Services, characteristics, and descriptors are collectively referred to as **attributes**, and identified by UUIDs.

• Any implementer may pick a random or pseudorandom UUID for proprietary uses, but the Bluetooth SIG have reserved a range of **UUIDs** (of the form `xxxxxxxx-0000-1000-8000-00805F9B34FB`) for standard attributes.
GATT Operations

- Discover UUIDs for all 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 all descriptors for a particular characteristic
GATT Operations

Commands are also provided to *read* (data transfer from server to client) and *write* (from client to server) the values of *characteristics*. 
Notifications: The client may request a notification for a particular characteristic from the server. The server can then send the value to the client whenever it becomes available. For instance, a temperature sensor server may notify its client every time it takes a measurement. This avoids the need for the client to poll the server, which would require the server's radio circuitry to be constantly operational.

Indication: similar to a notification, except that it requires a response from the client, as confirmation that it has received the message.
Profiles

• specifications for how a device works in a particular application
• Manufacturers are expected to implement the appropriate specifications for their device in order to ensure compatibility.
• A device may contain implementations of multiple profiles.
• Health Care: Blood Glucose, Blood Pressure, …
• Sports and fitness: Heart Rate, Running speed and cadence, …
Is it secure?

Short answer: No!

Passive eavesdropping possible
Ubertooth
Secure?

Long answer: Use Ubertooth + Wireshark for network sniffing

http://www.youtube.com/watch?v=4POOiVrdnX8
Your Smartphone Is Now Your Key

Keep your phone in your pocket or purse. No more fumbling for your keys… just touch the lock to open for the ultimate in convenience.
Bluetooth 4.1

- IPV6
- Platform for Internet of things
- Wireless charging?
Sensortag

- No embedded software knowledge required.
- No embedded compiler required.
- Android and iOS sample apps to demonstrate sensor use.
- [www.ti.com/sensortag](http://www.ti.com/sensortag)
- Cheap! 25$
Sensors

1. Temperature (Infrared + ambient)
2. Humidity
3. Pressure
4. Accelerometer
5. Gyroscope
6. Magnetometer
7. Buttons
BLE SensorTag

BLE DEVICE LIST

SensorTag
1C:BA:8C:20:C5:48
Rssi: -57 dBm

Connect

Scan

TEXAS INSTRUMENTS
/**
 * As a last-second hack i'm storing the barometer coefficients in a global.
 * /

public enum BarometerCalibrationCoefficients {
    INSTANCE;
    volatile public List<Integer> barometerCalibrationCoefficients;
    volatile public double heightCalibration;
}
private void scanLeDevice(final boolean enable) {
    if (enable) {
        // Stops scanning after a pre-defined scan period.
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                _devices.clear();
                mScanning = false;
                _bluetoothAdapter.stopLeScan(mLeScanCallback);
            }
        }, SCAN_PERIOD);

        mScanning = true;
        _bluetoothAdapter.startLeScan(mLeScanCallback);
    } else {
        mScanning = false;
        _bluetoothAdapter.stopLeScan(mLeScanCallback);
    }
}

// Device scan callbacks
// NB! Nexus 4 and Nexus 7 (2012) only provide one scan result per scan
private BluetoothAdapter.LeScanCallback mLeScanCallback = new BluetoothAdapter.LeScanCallback() {

    public void onLeScan(final BluetoothDevice device, final int rssi, byte[] scanRecord) {
        runOnUiThread(new Runnable() {
            public void run() {
                BleDeviceInfo bleDeviceInfo = new BleDeviceInfo(device, rssi);
                _devices.put(bleDeviceInfo.getBluetoothDevice().getAddress(), bleDeviceInfo);
                attemptConnection(device);
            }
        });
    }
};
Preferences

SENSORS ONLINE

Simple keys
Turn the Simple keys service on or off.

Accelerometer
Turn the Accelerometer service on or off.

Magnetometer
Turn the Magnetometer service on or off.

Gyroscope
Turn the Gyroscope service on or off.

IR and Ambient Temperature
Turn the IR and Ambient Temperature service on or off.

Humidity
Turn the Humidity service on or off.

Barometer
Turn the Barometer service on or off.
public class SensorTag {

  public final static UUID
          UUID_IRT_SERV =.fromString("f000aa00-0451-4000-b000-000000000000"),
          UUID_IRT_DATA = fromString("f000aa01-0451-4000-b000-000000000000"),
          UUID_IRT_CONF = fromString("f000aa02-0451-4000-b000-000000000000"), // 0: disable, 1: enable
          UUID_IRT_PERI = fromString("f000aa03-0451-4000-b000-000000000000"), // Period in tens of milliseconds

          UUID_ACC_SERV = fromString("f000aa10-0451-4000-b000-000000000000"),
          UUID_ACC_DATA = fromString("f000aa11-0451-4000-b000-000000000000"),
          UUID_ACC_CONF = fromString("f000aa12-0451-4000-b000-000000000000"), // 0: disable, 1: enable
          UUID_ACC_PERI = fromString("f000aa13-0451-4000-b000-000000000000"), // Period in tens of milliseconds

          UUID_HUM_SERV = fromString("f000aa20-0451-4000-b000-000000000000"),
          UUID_HUM_DATA = fromString("f000aa21-0451-4000-b000-000000000000"),
          UUID_HUM_CONF = fromString("f000aa22-0451-4000-b000-000000000000"), // 0: disable, 1: enable
          UUID_HUM_PERI = fromString("f000aa23-0451-4000-b000-000000000000"), // Period in tens of milliseconds

          UUID_MAG_SERV = fromString("f000aa30-0451-4000-b000-000000000000"),
          UUID_MAG_DATA = fromString("f000aa31-0451-4000-b000-000000000000"),
          UUID_MAG_CONF = fromString("f000aa32-0451-4000-b000-000000000000"), // 0: disable, 1: enable
          UUID_MAG_PERI = fromString("f000aa33-0451-4000-b000-000000000000"), // Period in tens of milliseconds

          UUID_BAR_SERV = fromString("f000aa40-0451-4000-b000-000000000000"),
          UUID_BAR_DATA = fromString("f000aa41-0451-4000-b000-000000000000"),
          UUID_BAR_CONF = fromString("f000aa42-0451-4000-b000-000000000000"), // 0: disable, 1: enable
          UUID_BAR_CALI = fromString("f000aa43-0451-4000-b000-000000000000"), // Calibration characteristic
          UUID_BAR_PERI = fromString("f000aa44-0451-4000-b000-000000000000"), // Period in tens of milliseconds

          UUID_GYR_SERV = fromString("f000aa50-0451-4000-b000-000000000000"),
          UUID_GYR_DATA = fromString("f000aa51-0451-4000-b000-000000000000"),
          UUID_GYR_CONF = fromString("f000aa52-0451-4000-b000-000000000000"), // 0: disable, bit 0: enable x, bit 1: enable
          UUID_GYR_PERI = fromString("f000aa53-0451-4000-b000-000000000000"), // Period in tens of milliseconds

          UUID_KEY_SERV = fromString("0000ffe0-0000-1000-8000-00805f9b34fb"),
          UUID_KEY_DATA = fromString("0000ffe1-0000-1000-8000-00805f9b34fb");
}
public enum Sensor {
    IR_TEMPERATURE(UUID_IRT_SERV, UUID_IRT_DATA, UUID_IRT_CONF) {
        @Override
        public Point3D convert(final byte[] value) {

            /**
             * The IR Temperature sensor produces two measurements; Object (AKA target or IR) Temperature,
             * and Ambient (AKA die) temperature.
             * Both need some conversion, and Object temperature is dependent on Ambient temperature.
             * They are stored as [ObjLSB, ObjMSB, AmbLSB, AmbMSB] (4 bytes)
             * Which means we need to shift the bytes around to get the correct values.
             */

            double ambient = extractAmbientTemperature(value);
            double target = extractTargetTemperature(value, ambient);
            return new Point3D(ambient, target, 0);
        }
    }
}
private void enableSensors(boolean enable) {
    for (Sensor sensor : mEnabledSensors) {
        UUID servUuid = sensor.getService();
        UUID confUuid = sensor.getConfig();

        // Skip keys
        if (confUuid == null)
            break;

        // Barometer calibration
        if (confUuid.equals(SensorTag.UUID_BAR_CONF) && enable) {
            calibrateBarometer();
        }

        BluetoothGattService serv = mbtGatt.getService(servUuid);
        BluetoothGattCharacteristic charac = serv.getCharacteristic(confUuid);
        byte value = enable ? sensor.getEnableSensorCode() : Sensor.DISABLE_SENSOR_CODE;
        mbtLeService.writeCharacteristic(charac, value);
        mbtLeService.waitIdle(GATT_TIMEOUT);
    }
}
Keys

Accelerometer

<table>
<thead>
<tr>
<th>Axis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>-0.30 g</td>
</tr>
<tr>
<td>y</td>
<td>-0.41 g</td>
</tr>
<tr>
<td>z</td>
<td>-0.83 g</td>
</tr>
</tbody>
</table>

Object Temperature

+31.80 deg/C

Ambience Temperature

+17.66 deg/C
public BluetoothGatt connectToGatt(String address) {
    final BluetoothDevice device = mBtAdapter.getRemoteDevice(address);
    int connectionState = mBluetoothManager.getConnectionState(device, BluetoothProfile.GATT);
    if (connectionState == BluetoothProfile.STATE_DISCONNECTED) {
        //do something..
    }
    BluetoothGatt bluetoothGatt = device.connectGatt(this, false, gattCallBacks);
    return bluetoothGatt;
}

private BluetoothGattCallback gattCallBacks = new BluetoothGattCallback() {
    @Override
    public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
    }
    @Override
    public void onCharacteristicRead(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic, int status) {
    }
    @Override
    public void onCharacteristicWrite(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic, int status) {
    }
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
    }
    @Override
    public void onDescriptorRead(BluetoothGatt gatt, BluetoothGattDescriptor descriptor, int status) {
    }
    @Override
    public void onDescriptorWrite(BluetoothGatt gatt, BluetoothGattDescriptor descriptor, int status) {
    }
    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
    }
};
public boolean setCharacteristicNotification(BluetoothGattCharacteristic characteristic, boolean enable) {
    if (!checkGatt())
        return false;

    if (!mBluetoothGatt.setCharacteristicNotification(characteristic, enable)) {
        Log.w(TAG, "setCharacteristicNotification failed");
        return false;
    }

    BluetoothGattDescriptor clientConfig = characteristic.getDescriptor(GattInfo.CLIENT_CHARACTERISTIC_CONFIG);
    if (clientConfig == null)
        return false;

    if (enable) {
        Log.i(TAG, "enable notification");
        clientConfig.writeValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
    } else {
        Log.i(TAG, "disable notification");
        clientConfig.writeValue(BluetoothGattDescriptor.DISABLE_NOTIFICATION_VALUE);
    }

    mBusy = true;
    return mBluetoothGatt.writeDescriptor(clientConfig);
}
private double extractTargetTemperature(byte[] v, double ambient) {
    Integer twoByteValue = shortSignedAtOffset(v, 0);
    double Vobj2 = twoByteValue.doubleValue();
    Vobj2 *= 0.00000015625;
    double Tdie = ambient + 273.15;
    double S0 = 5.593E-14; // Calibration factor
    double a1 = 1.75E-3;
    double a2 = -1.678E-5;
    double b0 = -2.94E-5;
    double b1 = -5.7E-7;
    double b2 = 4.63E-9;
    double c2 = 13.4;
    double Tref = 298.15;
    double S = S0 * (1 + a1 * (Tdie - Tref) + a2 * pow((Tdie - Tref), 2));
    double Vos = b0 + b1 * (Tdie - Tref) + b2 * pow((Tdie - Tref), 2);
    double fObj = (Vobj2 - Vos) + c2 * pow((Vobj2 - Vos), 2);
    double tObj = pow(pow(Tdie, 4) + (fObj / S), .25);
    return tObj - 273.15;
}
10 min break ??

After the break:

The world of iBeacons
THE WORLD IS CHANGING!
Objects are getting smarter, from detection, analyzing and interaction towards taking decisions. The innovative applications created by this, change the world at a rapid pace.

ARE YOU CHANGING ALONG?
Ordina SMART Technologies makes future life and work easier by creating sustainable solutions together. We make innovation applicable by cleverly combining:

- **Sensoring** (artificial senses)
- **Wearables** (portable technology)
- **Robotica** (programmable machines)
- **Augmented reality** (reality with digital additions)
- **Virtual reality** (world other than where you are physically)
- **Machine 2 machine** (communication between objects)

START SMALL, GROW SMART
Take the first step! Contact:

- **Jurgen.deschepper@ordina.be**
  + 32 (0) 495 26 88 28

- **Johan.steppe@ordina.be**
  +32 (0) 485 07 11 93

- **Frederick.Bousson@ordina.be**
  +32 (0) 486 89 53 38
iBEACON: Cool or Creepy?
iBeacon overview

• Introduced in iOS7

• Indoor positioning system.

• Contextual information

• Uses Bluetooth low energy Proximity sensing to transmit a Universally unique Identifier (UUID)
Android port

- https://github.com/RadiusNetworks
- Apache Licence 2.0, you can use it in your own application
- Active community
Mode of operation: Ranging

Calculate estimated distance to iBeacon using signal strength (Received Signal Strength Indicator, RSSI).

Distance or Bucket (Immediate, Near, Far, Unknown) High power.
Mode of operation: **Regioning**

- **Regioning:**
  - Enter / leave a region.
  - Background
  - Low power
Configuration

• **128-bit UUID** (uint8_t[16]) - The 128-bit ID indentifying your company/store/etc

• **TX Power** (uint8_t) - This value is used to try to estimate distance based on the RSSI value
Configuration

• **Major** (uint16_t: 0-65535) - The major value (to differentiate individual stores, etc.)

• **Minor** (uint16_t: 0-65535) - The minor value (to differentiate nodes within one location, etc.)
Example: Starbucks

• **UUID:**
  • Same on all Starbucks iBeacons.

• **Major:**
  • Designates store.

• **Minor:**
  • Designates tablets / seats.
  • Could use fixed minor for cash register, food area, exit, …
Use cases

- Indoor mapping
- Shoping experience
- Tracking behaviour
- Healthcare
- ...
Indoor location

Check-in coupons

Contactless payment

Proximity marketing
Some misconceptions

1. iBeacons deliver **content**
2. iBeacons **know** when they are detected
3. iBeacons are detected **immediately**
4. iBeacons distance is **accurate**
The actual iBeacon hardware

• www.estimote.com
• www.radiusnetworks.com/
• Build your own.
• Local vendor?
• ...

...
iOS 7 will be compatible with:

- iPhone 4
- iPhone 4S
- iPhone 5
- iPod touch 5th generation 16GB
- iPod touch 5th generation 32GB/64GB
- iPad 2
- iPad with Retina display
- iPad mini
Mac OS X 10.9 Mavericks + Bluetooth 4.0 Dongle @ €10,85
MacBeacon

Apple AirLocate E2C56DB5
Beacon On

Apple AirLocate 5A4BCFCE
Beacon Off

Apple AirLocate 74278BDA
Beacon Off

Radius Networks 2F234454
Beacon Off

Status: Beacon On

Name: Apple AirLocate E2C56DB5

UUID: E2C56DB5-DFF8-48D2-8060-D0F5A71096E0

Major: 1

Minor: 4

Power: -55

Radius Networks

Revert

Apply
https://github.com/mttrb/BeaconOSX
Virtual Beacon

• VMWare / Virtualbox + cheap bluetooth 4.0 dongle

• http://developer.radiusnetworks.com/ibeacon/virtual.html
Pi Beacon

- $50~
Phonegap
Triangulation
Use your phone to locate any iBeacons around you, measure distance to an iBeacon, or to calibrate your own iBeacons.

Locate iBeacons

Buy iBeacons

Develop Apps
Visible iBeacons
Tap on a row for more information

2F234454-CF6D-4A0F-ADF2-F4911BA9FFA6
Major: 2  Minor: 5
proximity: immediate  (0.28 meters)  rssi: -52
Distance

0.1

Meters
iBeacon Calibration
2f234454-cf6d-4a0f-adf2-f4911ba9ffaf6
   major: 2
   minor: 5
 txPower: -59

RSSI: -49

If you are setting up this iBeacon, you can get its transmit power calibration value by holding this device one meter away, pressing the button below, and holding this device steady for 30 seconds to one minute.

Calibrate
iBeacon Locate

**iBeacon Region Filter**
Configures the iBeacons that are seen

**Monitoring Configuration**
Configures alerts entering/exiting the iBeacon Region

**Sharing Configuration**
Configures sharing data with wikibeacon.org
Proximity UUID

Major

Minor
iBeacon Locate

Notify on entry  ON

Notify on exit   ON
iBeacon Locate


THE WORLD IS CHANGING!
Objects are getting smarter, from detection, analyzing and interaction towards taking decisions. The innovative applications created by this, change the world at a rapid pace.

ARE YOU CHANGING ALONG?
Ordina SMART Technologies makes future life and work easier by creating sustainable solutions together. We make innovation applicable by cleverly combining:

- **Sensoring** (artificial senses)
- **Wearables** (portable technology)
- **Robotica** (programmable machines)
- **Augmented reality** (reality with digital additions)
- **Virtual reality** (world other than where you are physically)
- **Machine 2 machine** (communication between objects)

START SMALL, GROW SMART
Take the first step! Contact:

- *Jurgen.deschepper@ordina.be*
  + 32 (0) 495 26 88 28

- *Johan.steppe@ordina.be*
  +32 (0) 485 07 11 93

- *Frederick.Bousson@ordina.be*
  +32 (0) 486 89 53 38