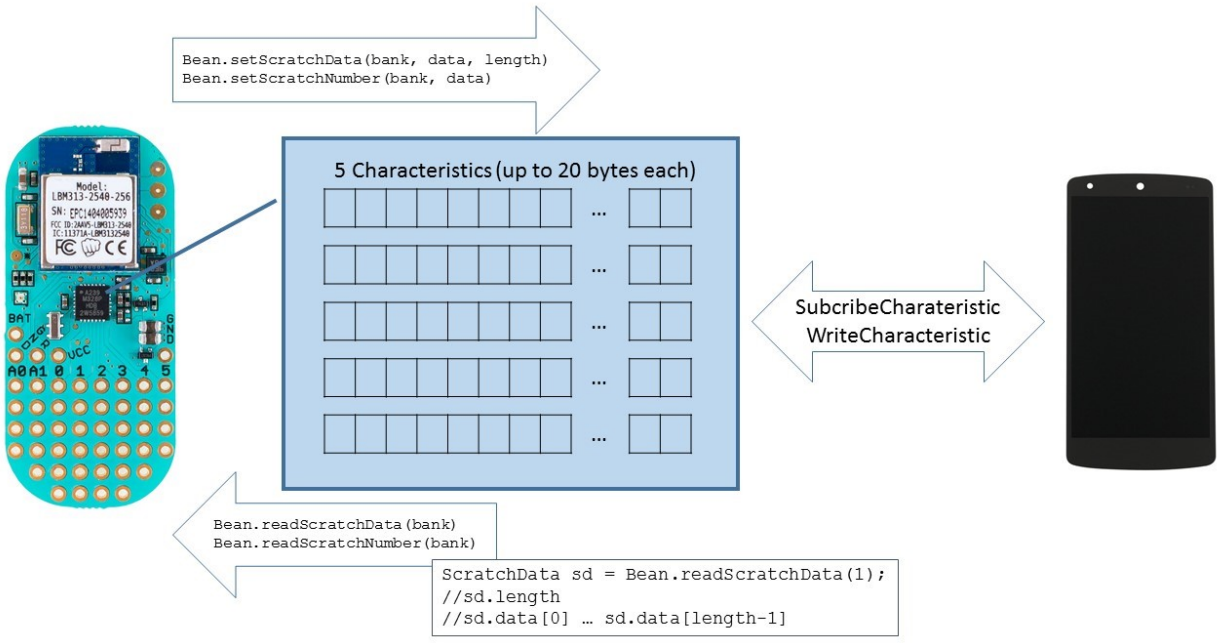
Android app that talks to a BLE device

By Victor Cheung

Last edit: Jun 2017

The Concept

BLE devices provide multiple **Services** to communicate with others. For example, battery, heart rate. Within a Service is a set of **Characteristics** that carries the actual values of interest. Each Service and Characteristics is identified by a 128-bit identifier (UUID). Here we are mostly interested in using a generic characteristic, which could be abstracted by the manufacturer to make it easier to use.



The above diagram shows how a Bean communicates with a mobile phone using the **Scratch** (not the IDE for Arduino) service. The Bean updates any of the 5 characteristics of the Scratch service when there is some new data from it (using the code Bean.set...), and reads data from any of the 5 characteristics periodically (using the code Bean.read...). Since each characteristics are both readable and writable, you are advised to perform R/W in different characteristics.

After connecting to the Bean, the mobile device **subscribes** to one of the 5 characteristics. This process follows the subscription metaphor, that is, when there is an update in the characteristic, the connected mobile device will be notified. If the mobile device has something to send, it **writes** to a characteristic.

The tools described in this document automates the **connect**, **subscribe**, and **write** part at the mobile device. Since the code at the BLE device side is hardware-specific, this document only provides sample code to achieve the communication. We'll be using Bean as the example here.

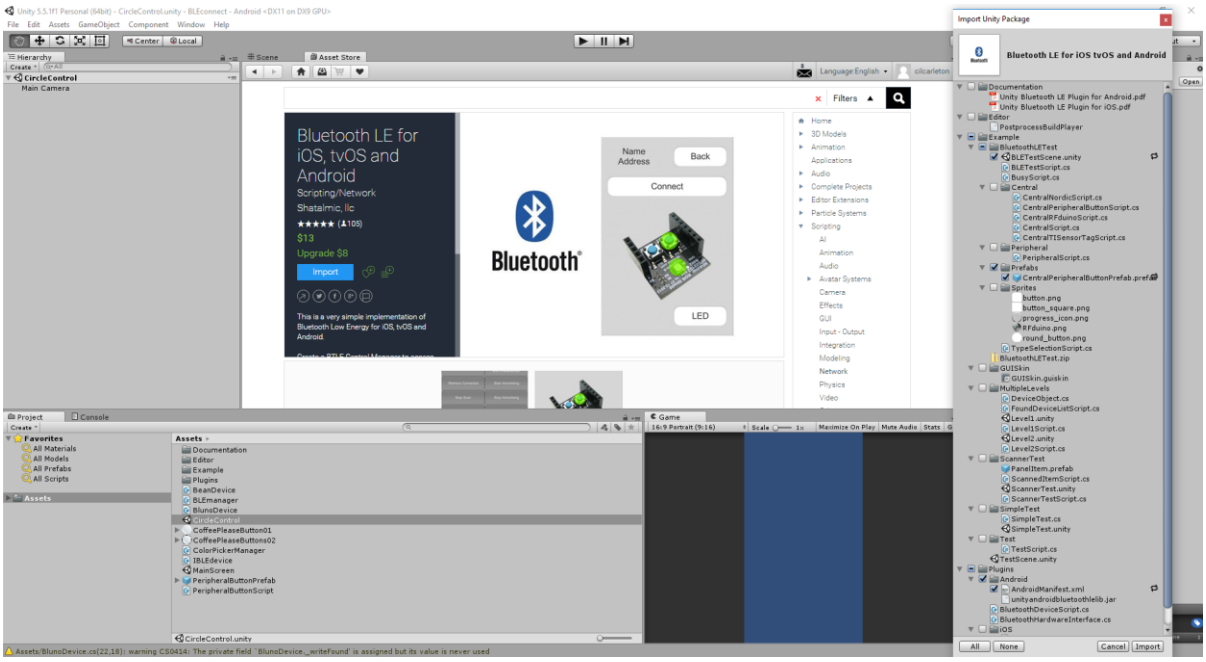
The Android Side

Here are the procedures to create an Android app in Unity (it should be the same for an iOS app, but you'll need XCode to do the final compilation and app upload). First you'll need the following:

- **Unity 5** (this document uses version 5.5.1f)
- The **Bluetooth LE for iOS, tvOS and Android** Unity package (https://www.assetstore.unity3d.com/en/#!/content/26661). Audrey has bought it for the lab, so use the CIL login for Unity and get it
- An **Android device** running Android 4.3 or above
  important: if you are using Android 6.0 or above, besides the BLUETOOH and BLUETOOTH_ADMIN permissions, you'll also need ACCESS_COARSE_LOCATION in the manifest:
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />,
  you might also want to list only the BLE enabled devices instead of everything:
  <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>

Procedures

Start Unity and create a project (e.g., BLEconnect), name the scene (e.g., CircleControl), and import the Unity package. Switch to the Android platform and set the aspect ratio to that of your Android device.



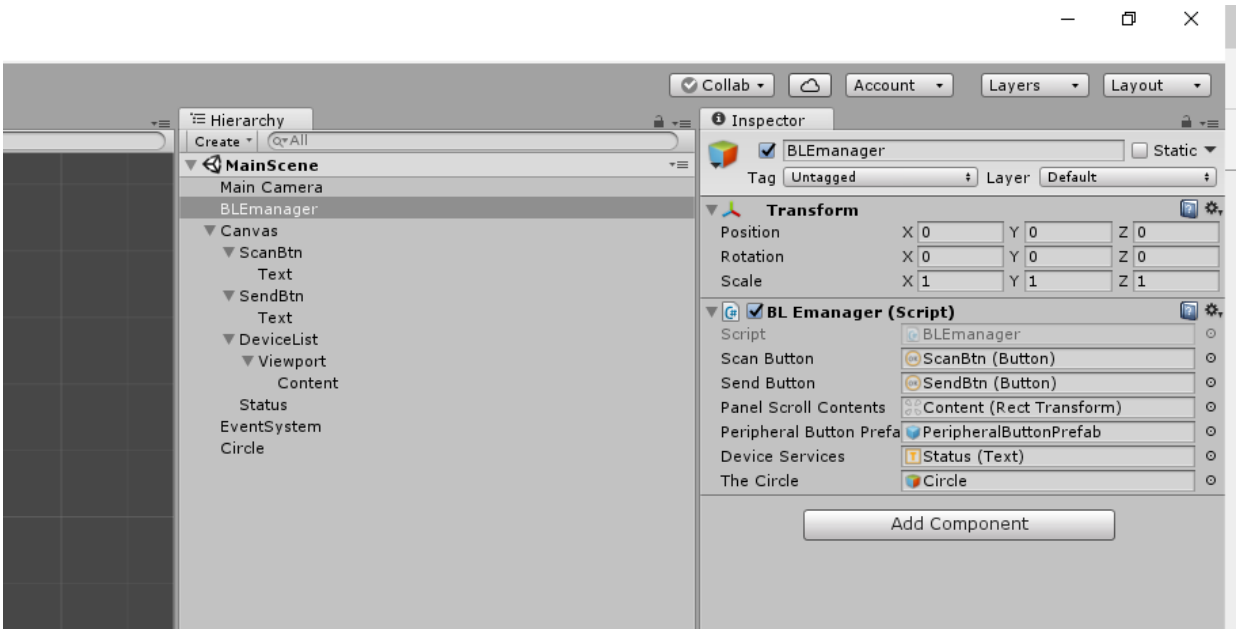
Import the following assets to your project:

- **BLEmanager.cs**: The class that does all the connection stuff
- **PeripheralButtonScript.cs**: The class that represents a BLE device as a selectable button
- **PeripheralButtonPrefab.prefab** (you might have to add the script to this prefab again)
- **IBLEdevice.cs**: Interface class that specifies what function a BLE device class should support
- **BeanDevice.cs**: The class that implements IBLEdevice and contains all the identifying information of a Bean device

- **BlunoDevice.cs**: The class that implements IBLEdevice and contains all the identifying information of a Bluno device

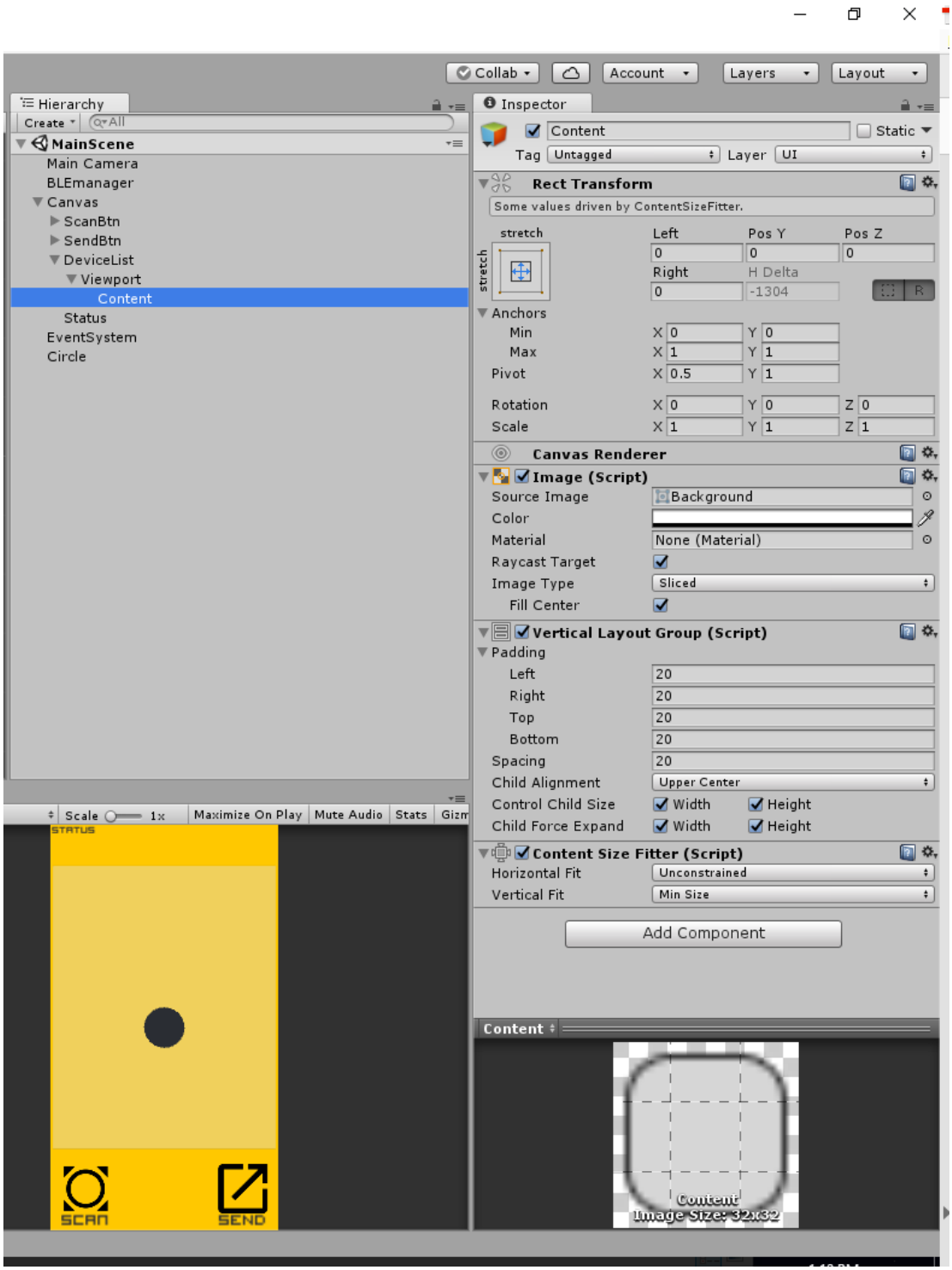
Create an Empty Object in the scene, name it as “BLEmanager”, and attach the script with the same name to it as a component. The script has a few references to the gameobjects in the scene:

- **Scan Button**: Button object that represents the control to scan for BLE devices. Set the BLEmanager.OnScan function as its OnClick() listener.
- **Send Button**: Button object that represents the control to send something (a byte array) to a connected BLE device. Set the BLEmanager.OnSend function as its OnClick() listener.
- **Panel Scroll Contents**: Container object that gets populated to show all the connectable devices. Set it to the content of a Scroll View object you create to list the devices.
- **Peripheral Button Prefab**: a Prefab from the assets that represents a connectable device in the Panel Scroll Contents. I created it based on the CentralPeripheralButtonPrefab made by the Unity BLE library we got to make it bigger.
- **Device Services**: Text object that shows the general status of the app. Good for quick debugging.
- Custom references to other game objects to control their behaviour. Just create whatever you need so the BLEmanager can make changes depending on what it receives from a connected device. Here “The Circle” is an example.



They are not all necessary depending on what you want it to do. Modify the code so it won’t throw any null reference errors.

You'll have to tweak the Scroll View at the Panel Scroll Contents to make it work. Add a **Vertical Layout Group** component and a **Content Size Fitter** component to the Content part of the view so the buttons can be placed properly. You'll probably need to add the Image component as well.



The BLE Device Side

Here are the procedures to create an arduino script at the BLE device so it measures something and notifies the subscribing Android app. First you'll need the following:

- A **BLE device** (currently we have two types: Bluno, and Bean)
- Some sensors connected to the BLE device (e.g., a bend sensor, a button)
- An IDE that compiles and upload the arduino script to the BLE device

Procedures

Since the subscribing Android device will be notified automatically when a characteristic is updated, all you need to do is to write the updated value from a sensor to a characteristic.

Notes

Due to BLE restrictions, each characteristic can only hold up to 20 bytes of data.

Because data are transferred in byte arrays, each element is 8-bit only. If you are sending values with a bigger size, for example, an integer that is 16-bit, you'll have to split the value like the following:

```
//buffer is a byte array with 2 elements [<least significant 8 bits>, <most significant two bits>]
buffer[0] = value & 0xFF;
buffer[1] = value >> 8;
// Store those two bytes in scratch characteristic 1
Bean.setScratchData(1, buffer, 2);
```

And at the app side stitch the bytes back:

```
//using System;
receivedValue = BitConverter.ToInt16(receivedData, 1); //receivedData: byte array, 1: starting point
```