Unity Bluetooth LE Plugin for Android

Introduction

This plugin provides basic access to the GATT Bluetooth API provided in Android. You can use this plugin from Unity to create Centrals. At the time of this release, Android does not support being a Peripheral.

For information about Android Bluetooth GATT you will need to consult the Android developer documentation.

Version Changes

2.1

Added an all new example that connects to RFduino and TI SensorTag devices

2.0

Added support for Android Centrals

Setup Guide

Setting up this plugin involves installing the package and writing scripts to interact with the plugin. Follow these steps:

1. Import the package into your Unity project
2. Create a game object that you can attach a script to
3. Create a script that makes calls into the BluetoothHardwareInterface class using static methods for the Bluetooth functionality you require

Example Code

This package includes a sample scene with a simple GUI exposing the most common functions that you will need. The example is located under the Example folder.

The example scene just includes the Main Camera object that is created in a new project. The TestScript sample script has been attached to the main camera.

The TestScript builds a simple GUI. When each button is clicked some very basic code is executed to demonstrate using the basic features of this plugin.

The buttons on the are used to connect to Bluetooth LE peripherals that are near the device. This is known as being a Bluetooth LE Central.

All output showing the device states and connections and data transfer is sent to the debug output. In order to see this you must have an active debug session in the Android SDK IDE. You can also use Unity's script debugging.

There is also a second example called BluetoothLETest. This example can connect to RFduino and TI SensorTag devices.

First select central as the type of device to simulate. Next start scanning. Any BLE devices will show up in the list. Nothing will happen if you select a device that is not an RFduino or TI SensorTag.

If you choose an RFduino or TI SensorTag device you will get a page specific to those devices.

The RFduino page shows an image of an RFduino device. If you select button1 on the RFduino that button on the image will have a green circle on it.

If you hit the LED button on the page the LED should show up on the RFduino device.

The TI SensorTag just shows temperature. It is left up to the user to implement other sensors on the TI SensorTag.

Plugin Layout
The plugin has several parts.

Android Java Class Library
The Plugins/Android folder contains the Android Java code library along with an AndroidManefest.xml file to setup the proper permissions and settings for the plugin.

The AndroidManifest.xml file contains Android permissions. There is also a setting that you can use to control whether your app shows up for devices that don't support Bluetooth Low Energy. If you edit the xml file you will see the following line of code:

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

If the required value is false as you see there, your app will be able to be installed on all devices that are otherwise compatible. During initialization of this plugin the device will be checked for Bluetooth Low Energy compatibility. If the device does not support Bluetooth Low Energy, you will receive and error in the error callback.

If the required value is true then your app will not show up in the Google Play store unless the target device supports Bluetooth Low Energy.

C# Scripts

The Plugins folder contains the C# scripts. These scripts provide the Unity – Plugin interface and some helper methods. The BluetoothHardwareInterface is a class that contains static methods to make calls into the Java code. The BluetoothDeviceScript is used to receive messages passed back to Unity from the Java code.

Example

The example code is detailed in the section above.

Support

For email support you can email support@shatalmic.com

API Reference

Since this library also works with iOS, some of the values are only used for iOS.

Initialization Errors

When Initialize is called there are several errors that can occur. You will receive the error text as the parameter to the errorAction callback. Here is a list of those errors:

> Bluetooth LE Not Enabled

BluetoothHardwareInterface Methods

public static void Log (string message)

> Log the string message to the Android console window

public static BluetoothDeviceScript Initialize (bool asCentral, bool asPeripheral, Action action, Action<string> errorAction)

> Initialize the Bluetooth system as either a central, peripheral or both. Acting as a peripheral is only available for iOS.
> When completed the action callback will be executed. If there is an error the errorAction callback will be executed.

public static void DeInitialize (Action action)

> DeInitialize the Bluetooth system.
> When completed the action callback will be executed.

public static void FinishDeInitialize ()

> This method is automatically called by the BluetoothDeviceScript when it has been notified by the Java code that everything else has been deinitialized.

public static void PauseMessages (bool isPaused)
> This method notifies the bluetooth system that the app is going to be paused or unpaused.

public static void ScanForPeripheralsWithServices (string[] serviceUUIDs, Action<string, string> action)
> This method puts the device into a scan mode looking for any peripherals that support the service UUIDs in the serviceUUIDs parameter array. If serviceUUIDs is NULL all Bluetooth LE peripherals will be discovered. As devices are discovered the action callback will be called with the ID and name of the peripheral.

public static void RetrieveListOfPeripheralsWithServices (string[] serviceUUIDs, Action<string, string> action)
> This method will retrieve a list of all currently connected peripherals with the UUIDs listed in the serviceUUIDs parameter. If serviceUUIDs is NULL all Bluetooth LE peripherals will be discovered. As devices are discovered the action callback will be called with the ID and name of the peripheral.

public static void StopScan ()
> This method stops the scanning mode initiated using the ScanForPeripheralsWithServices method call.

public static void ConnectToPeripheral (string name, Action<string> connectAction, Action<string, string> serviceAction, Action<string, string, string> characteristicAction)
> This method attempts to connect to the named peripheral. If the connection is successful the connectAction will be called with the name of the peripheral connected to. Once connected the serviceAction is called for each service the peripheral supports. Each service is enumerated and the characteristics supported by each service are indicated by calling the characteristicAction callback.

public static void DisconnectPeripheral (string name, Action<string> action)
> This method will disconnect a peripheral by name. When the disconnection is complete the action callback is called with the ID of the peripheral.

public static void ReadCharacteristic (string name, string service, string characteristic, Action<byte[]> action)
> This method will initiate a read of a characteristic using the name of the peripheral, the service and characteristic to be read. If the read is successful the action callback is called with the raw bytes of the read.

public static void WriteCharacteristic (string name, string service, string characteristic, byte[] data, int length, bool withResponse, Action<string> action)
> This method will initiate a write of a characteristic by the name of the peripheral and the service and characteristic to be written. The value to write is a byte buffer

with the length indicated in the data and length parameters. The withResponse parameter indicates when the user wants a response after the write is completed. If a response is requested then the action callback is called with the message from the Bluetooth system on the result of the write operation.

public static void SubscribeCharacteristic (string name, string service, string characteristic, Action<string> notificationAction, Action<byte[]> action)
  This method will subscribe to a characteristic by peripheral name and the service and characteristic. The notificationAction callback is called when the notification occurs and the action callback is called whenever the characteristic value is updated by the peripheral.

public static void UnSubscribeCharacteristic (string name, string service, string characteristic, Action<string> action)
  This method is unsubscribe from a characteristic by name, service and characteristic. When complete the action callback is called.