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

2.51

Added a new method ReadRSSI to read the RSSI value when already connected to a device.

2.50

Added automatic packetizing of packets larger than the Bluetooth MTU size. Removed the need to have a deinit button when using the editor. This is now done in the OnApplicationQuit method if the editor is being used.

2.30

All new method of packaging the plugin due to Unity requirement. You must now unpack the plugin package included in the base package.
Added MTU support.
Added internal packetizing of byte arrays that are larger than the MTU size.
Updated examples to remove OnGUI.

2.25

Added a new StarterExample for connecting to an ESP32 device.
Bug Fixes
EXPERIMENTAL: Support for OSX Editor and Stand Alone. Use with caution. This could crash the editor unless you DeInitialize before stopping playback.

2.24

Improved initialization when Bluetooth is initially off

2.23

Added support for the new Android permissions API in 2018.3

2.20

Fixed the android ConnectionPriority setting call.

2.19

Added iBeacon support for iOS.
Added iBeacon example for iOS.

2.18

Added ScanMode and ConnectionPriority settings for Android.
Added iBeacon support for Android.
Added iBeacon example for Android.

2.17   Added HM10 example.

2.16   Modification to Android Manifest xml for Unity 2018.

2.15   Bug fix for iOS in the native Objective-C code.

2.14   Bug fix for Android in the native Java code. We were getting a null reference in the StopScan method if it was called before scanning.

2.13   Added iOS specific preprocessor directive for post process build script.
Removed unneeded recoreType parameter to iOS scan method.

2.12   Added usage of api21 for scanning and stop scanning to support filtering on multiple UUID service values.
Added parameter to scanning method that is only used in iOS, but had to be in the generic api. This can be ignored for android.
Removed the TestScene example because it was confusing.

2.11   Fixed a bug in the AndroidManifest.xml file to use the correct android theme from Unity.
Fixed a bug in the StopScan method.
Added the Scanner example for scanning and showing RSSI values.

2.10   Nothing changed for Android. All changes are in iOS version.

2.9   Nothing changed for Android. All changes are in iOS version.

2.8   Added a new parameter to the scan method that allows you to get RSSI values even if the scanned device doesn't have manufacturer specific data.

2.7   Added SimpleTest. This is an example of connecting to an RFduino at startup using callbacks, states and a timer.

2.6   Fixed bugs when getting characteristics from multiple devices at the same time.
Added automatic lower casing of all UUID values for Android since that is what the OS does.

2.5   Added support for subscriptions to different characteristics to have their own callback handler.
Added a new example based on the Adafruit Bluefruit LE with the Nordic BLE Chip.

2.4   Added support for both Notification and Indication subscriptions in the Android library.

Added DontDestroyOnLoad to the Bluetooth Receiver Script so it can survive loading a new level.

2.3

Added SubscribeCharacteristicWithDeviceAddress
Fixed a bug in the android advertising packet request

2.2

Added disconnect callback from connect
Added callback for the RSSI and manufacturer specific data in the advertising packet

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
4. NOTE: For Android 6.x and later there is a new permission that is required. Please add the following line to the AndroidManifest.xml in the proper location:

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

Example Code

There are several eamples included in the plugin. The best example to demonstrate how to use the plugin is the StartingExample project. Study careful how it uses a state machine and the Update method to determine when to call the next method.

The StartingExample is designed to communicate with an esp32 based device that is referenced in the example .cs file. The source code to matching firmware that will run on an Arduino based esp32 device is included in the .cs file as well as a comment at the bottom.

iBeaconExample shows how to scan for beacons. iOS hides the signal power of the iBeacon protocol and returns a proximity value of unknown, far, near or immediate. For Android you get the signal power and rssi and this example shows how to calculate the distance from those values.

If you start and operation before the previous operation is finished then it cancels the first operation.

This is why you can't call the interface methods one after another in code. You must wait for the callback to an operation to be called before you initiate the next operation.

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" androi
d: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 examples are detailed in the section above.

Support

For email support you can email support@shatalmic.com

Notes

As mentioned above all operations are asynchronous. This means you can't start a new operation until you have received the callback from the previous operation. If you do you will probably cancel the previous operation.

You can use timeouts and hope that you are done with the previous operation, but this is not as deterministic.

It is most reliable to scan for peripherals and store the address of the peripheral that you find during the scan and use that device address in all further API calls for that device.

Android is case sensitive. Keep this in mind when you are comparing UUID values. There are some methods in the plugin that convert everything to lower case. The examples also show doing this.

Some of the examples have a helper method called FullUUID. This method will take a 16 bit UUID or a 128 bit UUID. If it is 16 bit, then it is folded into the BLE Specification standard UUID to make it 128 bits.

Some of the examples also have a helper method called IsEqual. This method helps you compare 2 UUID values even if they are 16 bit, 128 bit, upper or lower case. It is recommended that you use these 2 helper methods in your apps.

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 asCent
ral, bool asPeripheral, Action action, Action<string> error
Action)
      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 enum ScanMode
{
    LowPower = 0,
    Balanced = 1,
    LowLatency = 2
}
```

```
public static void BluetoothScanMode (ScanMode scanMode)
```
This method sets the Android scan mode. It affects the speed scanning is done and battery. The faster the scanning (lower latency) the more battery power. This method does nothing on iOS.

```
public enum ConnectionPriority
{
    LowPower = 0,
    Balanced = 1,
    High = 2,
}
```

```
public static void BluetoothConnectionPriority (ConnectionPriority connectionPriority)
```
This method sets the Android connection priority. It affects how fast data is passed back and forth with a connected peripheral. This method will use more battery at higher settings. This method does nothing on iOS.

```
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,
Action<string, string, int, byte[]> actionAdvertisingInfo =
null, bool rssiOnly = false, bool clearPeripheralList =
true)
>   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.
>
>   The default value for the actionAdvertisingInfo
>   callback is null for backwards compatibility. If you
>   supply a callback for this parameter it will be called
>   each time advertising data is received from a device.
>   You will receive the ID and address of the device, the
>   RSSI and the manufacturer specific data from the
>   advertising packet.
>
>   The rssiOnly parameter will allow scanned devices that
>   don't have manufacturer specific data to still send
>   the RSSI value. The reason this defaults to false is
>   for backwards compatibility.
>
>   The clearPeripheralList is only used in iOS, but is
>   here for cross platform compatibility in the api.

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 struct iBeaconData
{
    public string UUID;
    public int Major;
    public int Minor;
    public int RSSI;

```
    public int AndroidSignalPower;
    public iOSProximity iOSProximity;
}
```

```
public static void ScanForBeacons (string[] proximityUUIDs,
 Action<iBeaconData> actionBeaconResponse)
```
    This method starts scanning for the list of proximity uuids. This list of uuids is required. Only iBeacons are scanned for. The response action is called with the iBeaconData structure shown above. The iOSProximity is always "unknown" on Android. There is an example of calculating the distance using the AndroidSignalPower and RSSI value in the iBeaconExample.

```
public static void StopBeaconScan ()
```
    This method stops the scanning for beacons initiated using the ScanForBeacons method call.

```
public static void ConnectToPeripheral (string name, Action
<string> connectAction, Action<string, string> serviceActio
n, Action<string, string, string> characteristicAction,
Action<string> disconnectAction)
```
    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.

    The default value for the disconnectAction is null for backwards compatibility. If you supply a callback for this parameter it will be called whenever the connected device disconnects. Keep in mind that if you also supply a callback for the DisconnectPeripheral command below both callbacks will be called.

```
public static void DisconnectPeripheral (string name, Actio
n<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 RequestMtu(string name, int mtu,
Action<string, int> action)
```
This method will request a new MTU value. The max for
most iOS devices is 184.

```
public static void ReadRSSI(string name, Action<string,
int> action)
```
This method will read the RSSI value when you are
already connected to a device.

```
public static void ReadCharacteristic (string name, string
service, string characteristic, Action<string,
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 UUID of the
characteristic and the raw bytes of the read.

```
public static void WriteCharacteristic (string name, string
 service, string characteristic, byte[] data, int length, b
ool 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, st
ring service, string characteristic, Action<string> notific
ationAction, Action< string, 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. The first parameter is the characteristic
UUID. The second is the raw data bytes that have been
updated for the characteristic. This method is for
backwards compatibility. A new method with the device
address was added in version 2.3 (see below).

```
public static void SubscribeCharacteristicWithDeviceAddress
 (string name, string service, string characteristic, Actio
n<string, string> notificationAction, Action<string,
string, 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. The first parameter is the device address.
The second parameter is the characteristic UUID. The
third is the raw data bytes that have been updated for
the characteristic.

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