Documentation BluetoothLEPlugin 7.0 for Unity 2017

The BluetoothLEPlugin exposes methods for usage in your project. (Please read the important notice at the end of this document, if you are having any troubles)

This section will cover the methods that you need for building a connection to a bluetooth low energy capable device. First up, you have to attach the following events to your Start()-method:
-BluetoothLEPlugin.HRControlPointFeatureEvent
-BluetoothLEPlugin.BleDeviceScanEvent
-BluetoothLEPlugin.BleDeviceDisconnectEvent

To start scanning for devices you have to initialize the bluetoothadapter first by calling the method: *BluetoothLEPlugin.Initialize(b_Loggin).* Set b_Loggin to true if you want detailed logging.

Then you have to call the scan-method depending on what platform you are running:
StartCoroutine(StartDelayedScan(0)). 0 for androis systems and 0.2f for Mac- or iOS.
Please be sure to copy the StartDelayedScan-method from one of the samplescenes in your code, it includes the call:
BluetoothLEPlugin.Scan(90). The value 90 is the time in seconds the bluetoothadapter is searching for devices. You can modify this value.
Before you call the DelayedScan-method, you have to attach the *BluetoothLEPlugin.BleDeviceFoundEvent*.

If your systems bluetoothadapter is ready and turned on you will get the found devices at the *BleDeviceFoundEvent*.
Now you can get the device-name, -adress, -serviceUUID and connectionstate using: *BleDevice.Name, .Adress, .DeviceServUUID, and .Connected*.

Before you connect to a device with *BleDevice.Connect()* you have to detach the *BluetoothLEPlugin.BleDeviceFoundEvent* (otherwise it will keep on firing) and attach to *BluetoothLEPlugin.BleDeviceConnectEvent* which is fired, if the device is connected successfully.

If *BleDeviceConnectEvent* is fired, please attach the *BluetoothLEPlugin.BleReadyForSyncEvent* which is fired, if the device is ready for transmitting data.

If the *BleReadyForSyncEvent is fired*, please detach *BluetoothLEPlugin.BleDeviceConnectEvent* and attach ***HeartRate****.OnHRMDataReceived*. (important: HeartRate not BluetoothLEPlugin!).

Now you call the *BleDevice.Sync(b_Logging)*-method.
The data is now being transmitted to the OnHRMDataReceived-Event.

To get the values please use:
HeartRate_Measurement Measurement = BleDevice.GetHeartRateMeasurement();
Measurement.pulsrate for the actual pulsrate as uint

If featured by your device, you can get:
(HRM_BodySensorLocation)Measurement.SensorLocation for the sensorlocation as string
(HRM_SensorContactStatus)Measurement.SCStatus for the sensorcontactstatus as string
Measurement.energyExpended for the energy expended as uint
Measurement.rrInterval[0] gives you the number of available RR-Intervals on the device
Measurement.rrInterval[1...n] for the RR-Interval as uint in miliseconds, where the Value[1] is older than [2] and so on.

If you are done with the measurement and you want to disconnect the device simply call *BleDevice.Reset()*. This is used to detach the events in the background and *BleDevice.Disconnect()*

BluetoothLEPlugin.cs exposes the following methods:

```
// Initializes the Hardware, enables detailed logging if shouldLog is set to true. Must be called on start.
public static void Initialize(bool shouldLog)

// Checks and returns the bluetooth-hardware-state on the BleDeviceScanEvent, scans for devices and, if
// found, returns the devices on the BleDeviceFoundEvent.
// Stopps scanning after SecondsToScan.
public static void Scan(int SecondsToScan)

// (Optional) Stopps scanning at any time.
public static void StopScan()

// Discrete method for getting the actual status of the bluetooth-hardware. Status returned on the
// BleDeviceScanEvent.
public static void CheckBleStatus()

// Turns on bluetooth on the device. Only for iOS and Android
public static void EnableBluetooth()

// Connects to the delicered device
public static void Connect(BleDevice) BluetoothLEPlugin.cs fires following events:

// Plugin events
// This Event is fired, if the device supports writing ControlPoints to it.
public static event HRControlPointFeature HRControlPointFeatureEvent;

// This Event shows the actual scanning-state.
public static event BleDeviceScan BleDeviceScanEvent;

    SCAN_READY = 0,
    NOT_SUPPORTED = 1, (Your system (or Smartphone) does not support bluetooth low energy
    NOT_AVAILABLE = 2, (The bluetoothadapter of your system is busy)
    POWERED_OFF = 3, (Bluetooth is powered off)
    TIMEOUT = 4 (Scanning stopped due to a timeout, to save energy

// Fired whenever a device is found
public static event BleDeviceFound BleDeviceFoundEvent;

// Fired whenever a device is connected
public static event BleDeviceConnect BleDeviceConnectEvent;

// Fired whenever a device is disconnected
public static event BleDeviceDisconnect BleDeviceDisconnectEvent;

    DESCRIPTOR                  = 5
    DEVICE_NOT_AVAILABLE        = 6
    DEVICE_DID_DISCONNECT       = 7
    MANUAL_DISCONNECT           = 8
    DISCOCERSERVICE             = 9
    CBErrorUnknown              = 10
```

```
        CBErrorInvalidParameter         = 11
        CBErrorInvalidHandle            = 12
        CBErrorNotConnected             = 13
        CBErrorOutOfSpace               = 14
        CBErrorOperationCancelled       = 15
        CBErrorConnectionTimeout        = 16
        CBErrorPeripheralDisconnected   = 17
        CBErrorUUIDNotAllowed           = 18
        CBErrorAlreadyAdvertising       = 19
        DEVICE_DID_NOT_RESPOND          = 20
        DEVICE_BATTERY_LOW              = 21
        SUCCESSFUL_MEASUREMENT          = 22

// Fired whenever the device is ready for syncing
public static event BleReadyForSync BleReadyForSyncEvent;
```

HeartRate.cs exposes the following methods:

```
// Starts syncing data from the device.
public static void Sync()

// Reports the synced data to your class.
public static void GetHeartRateMeasurement()

// If supported, writes a control point to the device.
public static void writeControlPoint()

// Resets the HeartRate-class and detaches events in the background.
public static void Reset()

// Disconnects the actual connected device
public static void Disconnect()
```

HeartRate.cs fires following events:

```
// Plugin events
// Fired whenever the device sends data
public static event HRMDataReceived OnHRMDataReceived;
```

For using Android please notice that if the smartphone hasn´t installed at least Android 4.3 or isn´t Bluetooth 4.0 LE capable, the plugin returns m_IsBleEnabled = false.

The Ble_Device_Android.aar library has a AndroidManifest included, which contains:
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
For Androidversions > 6 (Api 23) there is also added
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
This is required and be sure, you have turned on your locationservice (GPS)

Important notice:

- Using Mac-platform, you have to select Ble_Device_MAC.bundle in the Heartrate/Plugins folder and check, if Editor and Standalone are ticked at Select platforms for plugin. For Platform settings set CPU and OS to any.
- Using iOS-platform, you have to select libBle_Device_iOS in the Heartrate/Plugins folder and check, if iOS is ticked at Select platforms for plugin. For Platform settings check, if at Rarely used frameworks CoreBluetooth is ticked
- Using Android-platform, you have to select Ble_Device_Android in the Heartrate/Plugins folder and check, if Android is ticked at Select platforms for plugin.
- Never forget to add the current scene to your Build Settings.
- Please be sure, to attach the BluetoothLEPlugin script to your main object
- Bluetooth and GPS have to be turned on

We added some Examplescenes to show you, what you can do with the Plugin:

-ExampleSceneBumpingHeart: pictures a bumping Heart pulsing with the Bpm from the Device
-ExampleSceneEKG: pictures a electrocardiogram (ECG) known from medical devices
-ExampleSceneImpulse: pictures a simplified ECG based on a sawtooth diagram
-ExampleSceneSinus: pictures a simplified ECG based on a sine wave
-ExampleSceneTriangle: pictures a simplified ECG based on a triangle wave