Movesense-Plugin for Unity 2018

Version 1.0

Description and usage for all methods and events

Company: Kaasa Solution GmbH
Editor: Alexander Kern
Support: info@movesense.com

Table of Contents

1. Introduction Movesense

Movesense is an open development environment for motion sensing solutions. Its easy and innovative tools revolutionize measuring and sensing sports. With **Movesense**, you can make better sense of your favorite sports activities.

You can track motion, analyze the data, and gain valuable insights while unlocking a whole new level of sports experience for participants and athletes. And you don't need to restrict to sports **Movesense** can track anything that moves.

Build your own wearables or make existing gear smart and connected. Use the **Movesense** toolkit to develop, test and take your idea to market faster and easier than ever.

Capabilities

Do you have a brilliant measurement concept that is vitally needed in your sport? Get a developer kit and make it happen! If you are not a software expert, find a partner and do it together. With **Movesense**, taking your idea to practice is fast and simple.

Here is a link to Movesense Wiki

You can purchase Movesense development kit from www.movesense.com/shop

2. Links Movesense API and documentiation

- **Movesense API**
- **Movesense documentation**

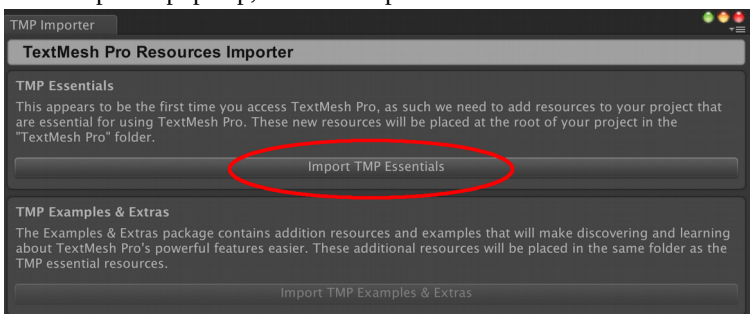
3. Setup:

First of all: the plugin is designed to run on Android- and iOS-devices; NOT in the Editor!

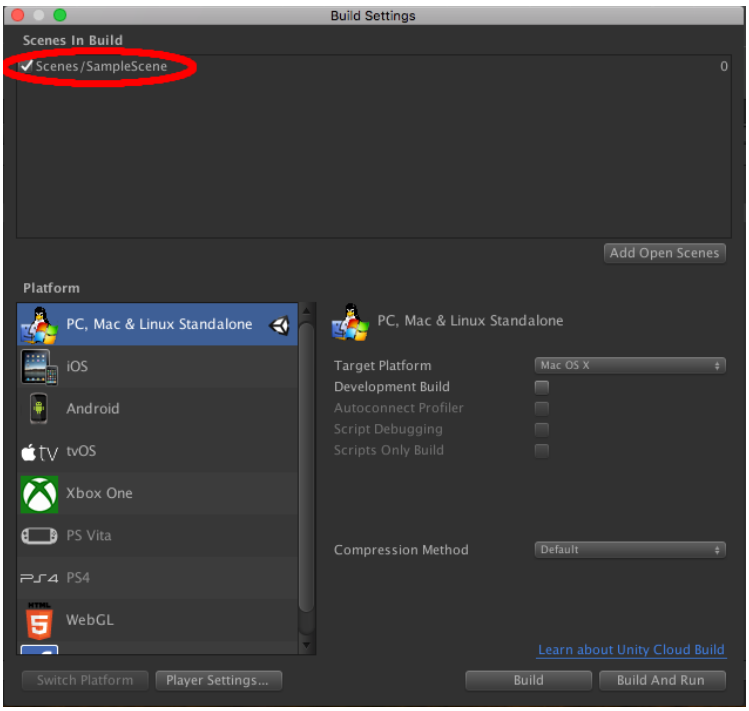
Here is how to setup Unity for usage of this plugin

3.1 Usage with example scenes:

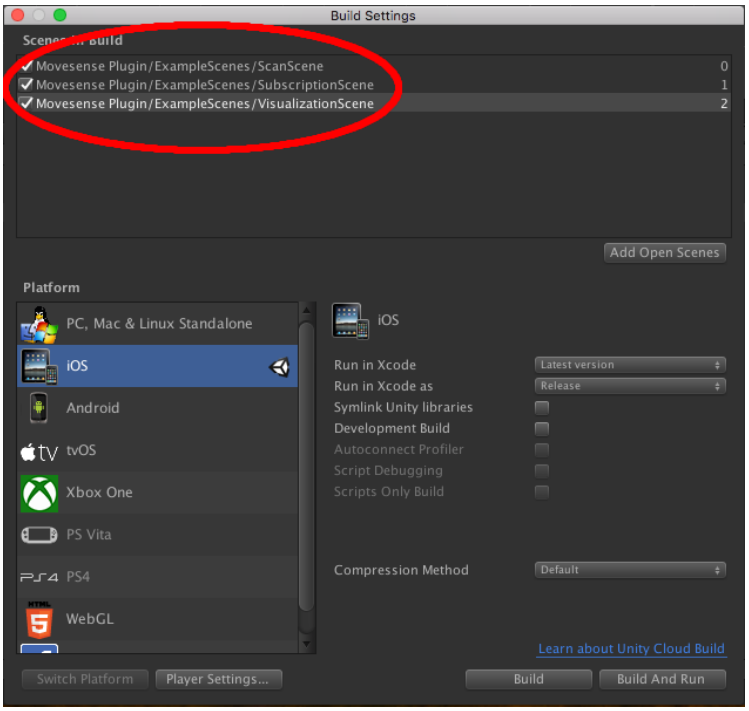
- Import package
- Load scene: Movesense Plugin/ExampleScenes/ScanScene
- TMP Importer pups up, click on Import TMP Essentials



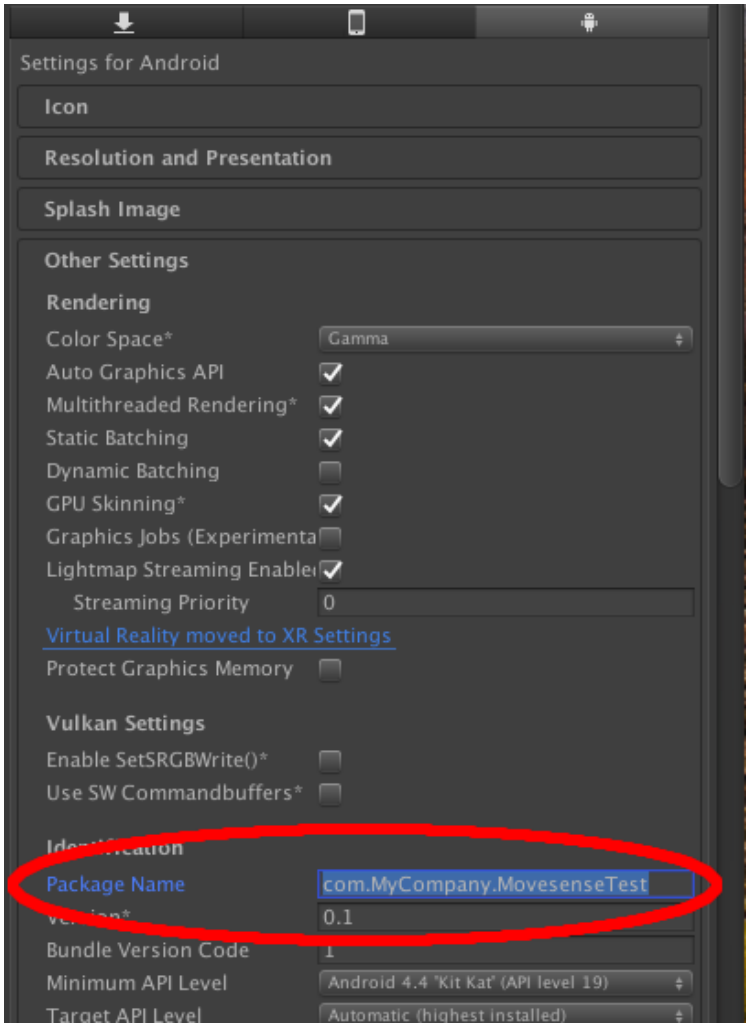
- To make the errors in the console disappear, load another scene and again load ScanScene.
- Open Build Settings (File/Build Settings) and remove unneeded scenes (backspace or delete)



- add examplescenes to build settings



- select platform Android or iOS and click Switch Platform
- Player settings, mandatory for Android:
  - Enter a Package Name



  - The Minimum API Level is automatically set to 19

3.2 Usage without example scenes:

- Import Package uncheck ExampleScenes
- Create two empty objects in the Hierarchy, rename them to ScanController- and MovesenseController objects and add the identically named scripts to these objects.
  The MovesensController object will be kept alive all the times (minded about in the added script) and is needed, to receive the callbacks from the sensor.
- Player settings:
  - Mandatory for Android: Enter a package name
  - The Minimum API Level is automatically set to 19
- Continue with Topic 5 Usage

4. Classes and events

4.1 ScanController

4.1.1 ScanControllerCallbackEvent

This Event contains the EventType, the method from where the event is risen and the MacID.
Its types are:

- REMOVE_UNCONNECTED:
  Informs you, that all unconnected sensors are removed from MovesenseDevice.Devices when you start scanning. MacID: null
- SYSTEM_SCANNING:
  Informs you, that the ScanController started scanning. MacID: null
- SYSTEM_NOT_SCANNING:
  Informs you, that the ScanController stopped scanning. MacID: null
- RSSI:
  Informs you, that a scanned sensor changed its RSSI
- NEW_DEVICE:
  Informs you, that a new sensor was found
- REFRESH:
  Every 2 seconds the list with found devices is refreshed. This will inform you, if a device got out of reach and is no more found during scanning. MacID: null

Subscription to the event:
ScanController.Event += yourMethod;
void yourMethod(object sender, ScanController.EventArgs e) {}

4.1.2 Methods

- public static void StartScan(): starts scanning
- public static void StopScan(): stops scanning

4.2 MovesenseController

4.2.1 MovesenseControllerCallbackEvent

This event contains the EventType, the method from where the event is risen and a list of events from ConnectCallback, NotificationCallback and ResponseCallback(sometimes more events arrive per frame, so they are put in this list).

Its types are:

- CONNECTING:
  Informs you, that currently a sensor is connecting.
- CONNECTED:
  Informs you, that the a sensor got connected
- DISCONNECTED:
  Informs you, that a sensor got disconnected
- NOTIFICATION:
  Informs you, that the sensor sent data
- RESPONSE:
  informs you of the result of a request

Subscription to the event:

```
MovesenseController.Event += yourMethod;
void yourMethod(object sender, MovesenseController.EventArgs e) {}
```

If you want further information from the **ConnectionEvent**, cast to its EventArgs:

```
for (int i = 0; i < e.OriginalEventArgs.Count; i++) {
        var Foo = (ConnectCallback.EventArgs) e.OriginalEventArgs[i];
}
```

and get access to:

- Foo.IsConnect:
  sensor is connected
- Foo.Serial:
  Serialnumber of the sensor
- Foo.MacID:
  either MAC-Address in Android or Identifier in iOS

If you want further information from the **NotificationEvent**, cast to its EventArgs-List:

```
for (int i = 0; i < e.OriginalEventArgs.Count; i++) {
        var Foo = (NotificationCallback.EventArgs) e.OriginalEventArgs[i];
}
```

and get access to:

- Foo.Serial:
  Serialnumber of the sensor
- Foo.SubscriptionPath:
  The subscriptionpath which is used in the sensor library
- Foo.Data:
  The raw data from the sensor formatted as JSON-String

The plugin provides deserialized data from the raw data for linear acceleration, angular velocity (gyroscope), magnetic field, heart rate and temperature. To access it cast from the NotificationCallback.EventArgs to (be sure before casting to check the SubscriptionPath) :

- Linear acceleration, angular velocity and magnetic field:
  var notificationFieldArgs = (NotificationCallback.FieldArgs) Foo;
  access the values:
  - notificationFieldArgs.Values[].x
  - notificationFieldArgs.Values[].y
  - notificationFieldArgs.Values[].z

- Heart rate:
  var notificationHeartRateArgs = (NotificationCallback.HeartRateArgs) Foo;
  access the values:
  - notificationHeartRateArgs.Pulse
  - notificationHeartRateArgs.rrData[]
- Temperature:
  var notificationTemperatureArgs = (NotificationCallback.TemperatureArgs) Foo;
  access the value:
  - notificationTemperatureArgs.Temperature

If you want further information from the **ResponseEvent**, cast to its EventArgs:

```
for (int i = 0; i < e.OriginalEventArgs.Count; i++) {

        var Foo = (ResponseCallback.EventArgs) e.OriginalEventArgs;

}
```

and get access to:

- Foo.Uri
- Foo.Method
- Foo.Data

4.2.2 Methods

- public static void **Connect(string MacID)**:
  connects to the sensor with its MacID
- public static void **Disconnect(string MacID)**:
  disconnects the sensor with its MacID.
- public static void **Subscribe(string Serial, string Subscriptionpath, int? Samplerate)**:
  subscribe to the sensor with Serial, Subscriptionpath and Samplerate.
  Subscriptionpath can be chosen from: SubscriptionPath
  For Samplerate you can use numeric values [13,26,52,104,208,416] or choose from SampleRate

  The following SubscriptionPaths are available:
  - SubscriptionPath.LinearAcceleration
  - SubscriptionPath.AngularVelocity
  - SubscriptionPath.MagneticField
  - SubscriptionPath.HeartRate
  - SubscriptionPath.Temperature

  The following Samplerates are available:
  - SampleRate.slowest
  - SampleRate.slower
  - SampleRate.medium
  - SampleRate.fast
  - SampleRate.faster
  - SampleRate.fastest
- public static void **UnSubscribe(string Serial, string SubscriptionPath)**:
  unsubscribes Serial from the SubscriptionPath
- **request methods**:
  How to use is described in the Movesense-API
  - public static void **ResponseGet(string Serial, string Path)**
  - public static void **ResponsePut(string Serial, string Path, string JsonParameters)**
  - public static void **ResponsePost(string Serial, string Path)**
  - public static void **ResponseDelete(string Serial, string Path)**

4.3 MovesenseDevice

4.3.1 Devices

A list containing all scanned and connected sensors, sorted by RSSI.
When you connect to a sensor, the connected sensors move to the bottom of the list.
If you don’t want the list to be sorted, set ShouldSortByRssi = false. Also you can pass null for RSSI.

4.3.2 Methods

- public static string GetSerial(string MacID)
  returns the serialnumber of its MacID
- public static string GetMacID(string Serial)
  returns the MacID of its Serial
- public static bool ContainsMacID(string MacID)
  returns if a MacID is in Devices
- public static int ContainsSerialAt(string Serial)
  returns the list position of a device with the Serial
- public static void RemoveUnconnected()
  remove all unconnected devices
- public static bool GetConnectingState(string MacID)
  returns if device is currently connecting
- public static bool GetConnectedState(string MacID)
  returns if device is currently connected
- public static bool IsAnyConnectedOrConnecting()
  returns if any device is connecting or connected
- public static int NumberOfConnectedDevices()
  returns the number of connected devices
- public static int NumberOfConnectDevices()
  returns the number of connecting and connected devices
- public static List<string> GetAllSubscriptionPaths(string Serial)
  returns a list with all subscriptions for a Serial
- public static bool isAnySubscribed()
  returns if there is any subscription for any serial active

5. Usage

A simple example script is added to the plugin. At SuperSimpleExample.cs you can check out, what is needed to get the plugin up and running.

- **1. Step scan:**
  - attach event:
    ScanController.Event += OnScanControllerCallbackEvent;
    MovesenseController.Event += OnMovesenseControllerCallbackEvent;
  - ScanController.StartScan();
  - as you get ScanController.EventType.NEW_DEVICE in OnScanControllerCallbackEvent a new sensor is added to MovesenseDevice.Devices.
- **2. Step connect**

  You can connect with the MacID from the device list or the one from the ScanController.EventType.NEW_DEVICE event.
  - MovesenseController.Connect(MovesenseDevice.Devices[position in list].MacID);
  - As you get ScanController.EventType.CONNECTED in OnMovesenseControllerCallbackEvent you can subscribe to data or sendRest requests to the sensor
- **3. Step subscribe**
  - attach event:
    MovesenseController.Event += OnMovesenseControllerCallbackEvent;
  - MovesenseController.Subscribe(((NotificationCallback.NotificationCallbackEventArgs) e.OriginalEventArgs[]).Serial, SubscriptionPath.AngularVelocity, SampleRate.slowest); (for example)
    IMPORTANT:
    - if you subscribe to SubscriptionPath.HeartRate or SubscriptionPath.Temperature, no SampleRate has to be added
    - if you subscribe to the other Subscritpionpaths, an SampleRate has to be added
  - as you get MovesenseController.EventType.NOTIFICATION in OnMovesenseControllerCallbackEvent you can handle the data
- **OR send requests**
  - attach event;
    MovesenseController.Event += OnMovesenseControllerCallbackEvent;
  - use either MovesenseController.Subscribe.ResponseGet(), MovesenseController.Subscribe.ResponsePost(), MovesenseController.Subscribe.ResponsePut() or MovesenseController.Subscribe.Delete() with parameters: string serial and string path. The path can be found at **Movesense API**
  - as you get MovesenseController.EventType.RESPONSE in OnMovesenseControllerCallbackEvent you can handle the response
- **4. Handle data**
  - the data arrives at OnMovesenseControllerCallbackEvent with MovesenseController.EventType.NOTIFICATION. Now you can handle the raw data as described in
    - var Foo = (NotificationCallback.EventArgs) e.OriginalEventArgs[];
      at Foo.Data;
      or:
    - deserialized data:
      - cast for linear acceleration, angular velocity (gyroscope) and magnetic field:
        var notificationFieldArgs = (NotificationCallback.FieldArgs) Foo;
        access:
        notificationFieldArgs.Values[].x, notificationFieldArgs.Values[].y and notificationFieldArgs.Values[].z
      - cast for heartrate:
        var notificationHeartRateArgs = (NotificationCallback.HeartRateArgs) Foo;
        access:
        notificationHeartRateArgs.Pulse and notificationHeartRateArgs.RrData[]
      - cast for temperature:
        var notificationTemperatureArgs = (NotificationCallback.TemperatureArgs) Foo;
        access:
        notificationTemperatureArgs.Temperature
        to get Celsius subtract -273.15

6. Example scenes:

We added three example scenes to the plugin, to demonstrate you, how the workflow of the plugin could be.
It starts with ScanScene, followed by SubscriptionScene and VisualizationScene. You can modify the scenes if you want, but keep in mind, that we don‘t support any changes done by you.

Please follow the Setup for usage with example scenes, then everything should work.

7. Known issues:

- you can find a list of compatible mobile devices @:
  List of compatible devices
- We found out, that the number of connectable sensors differ using different smartphones
- There was an appcrash with Unity 2013.3.0 which we could not, until now, reproduce on 2017.3.1
- The more sensors you connect, the slower gets the frame rate in the example scenes, this depends on the bandwidth of bluetooth or the cpu of your smartphone
- Examplescene: sometimes, if you connect and keep scanning, the device you are connecting to disappears for a moment. Just wait a few seconds and it will appear again as connected.
- if you don’t move a sensor, it sends (movement)data anyway. So we added a threshold (only in the example scene) for angular velocity (gyroscope) to prevent the displayed sensor from moving