### Requirements

* Unity 2019.4 or Later, with
  + Android Build Support
    - Android SDK & NDK Tools
    - OpenJDK
* MacOS X

# Quickstart

The purpose of this guide is to get BLE-Unity up and running in your Unity project as quick as possible.

## Setup

### Authentication

This section shows how to install the guest access token to the GripAble Unity Package Registry on your machine:

Open Terminal

Create this file in your home directory using nano or vim

|  |
| --- |
| nano ~/.upmconfig.toml |

Add these lines to the file

|  |
| --- |
| [npmAuth."http://34.105.177.10:4873"] token = "c4+ji5dQe4dBLbzAu9yTlxnWx8bz/wiSfzqzXUi77D4=" |

Make sure you save the file when exiting!

Now your machine is authorized to download GripAble packages via Unity Package Manager.

### Installation

This section shows how to download the BLE-Unity package into your Unity Project.

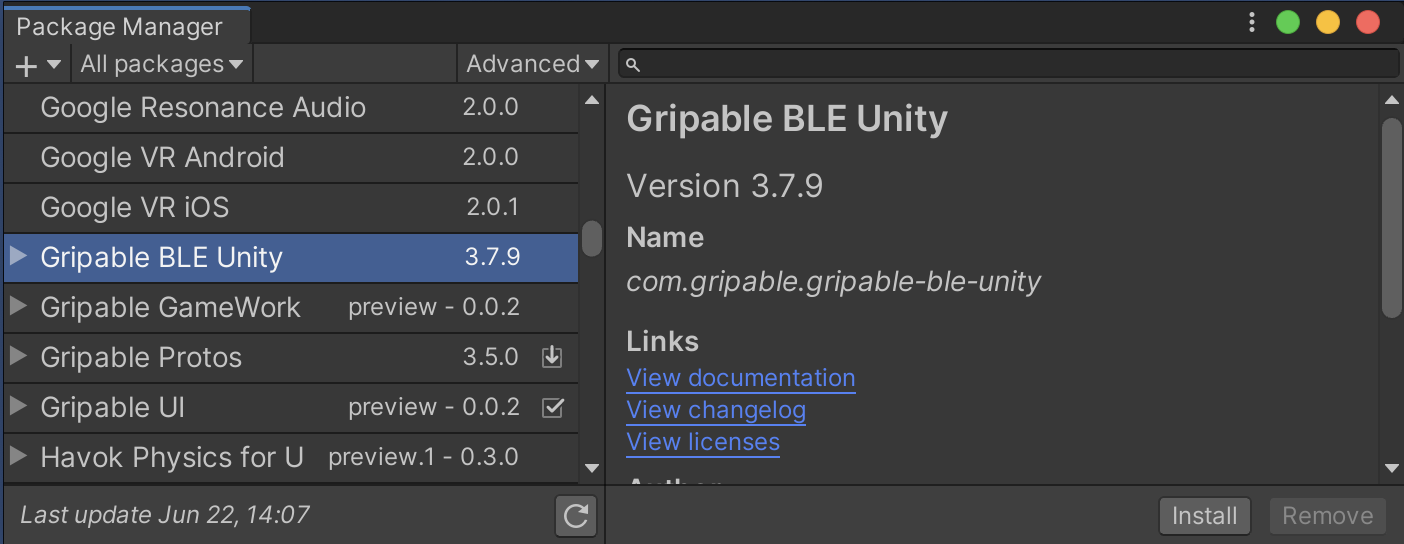
First we need to add the GripAble Unity Package Registry to the manifest.json file of the Unity Project

|  |
| --- |
| cd ~/UnityProjectFolder vi Packages/manifest.json |

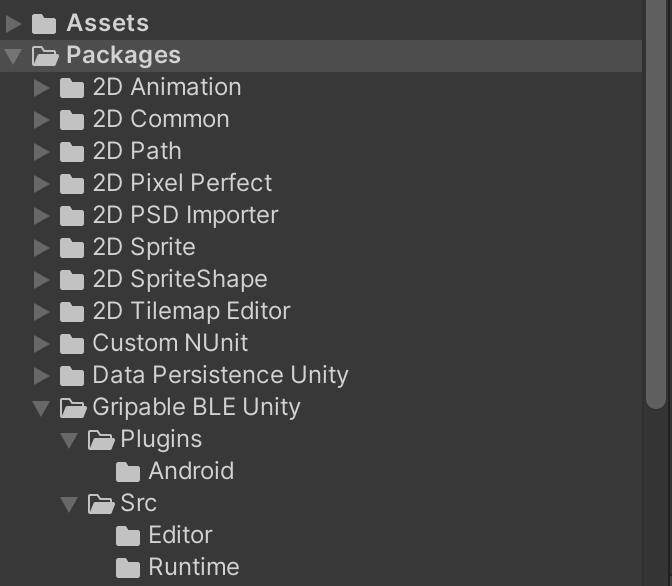
Add the registry to the top of the file, above the dependencies. It should look like this:

|  |
| --- |
| {  "scopedRegistries": [  {  "name": "Gripable Packages",  "url": "http://34.105.177.10:4873",  "scopes": [  "com.gripable"  ]  }  ],  "dependencies": {  ... |

Gripable Packages should now appear in the Unity Package Manager window. Inside Unity, go to Window -> Package Manager

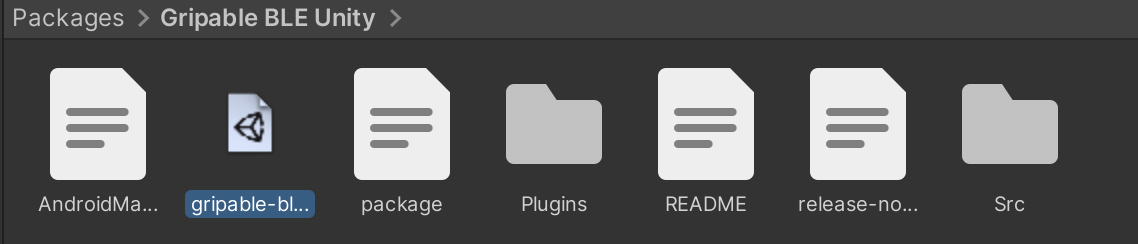


Press the “Install” button and the “Gripable BLE Unity” package will be imported into the Packages folder of the project:

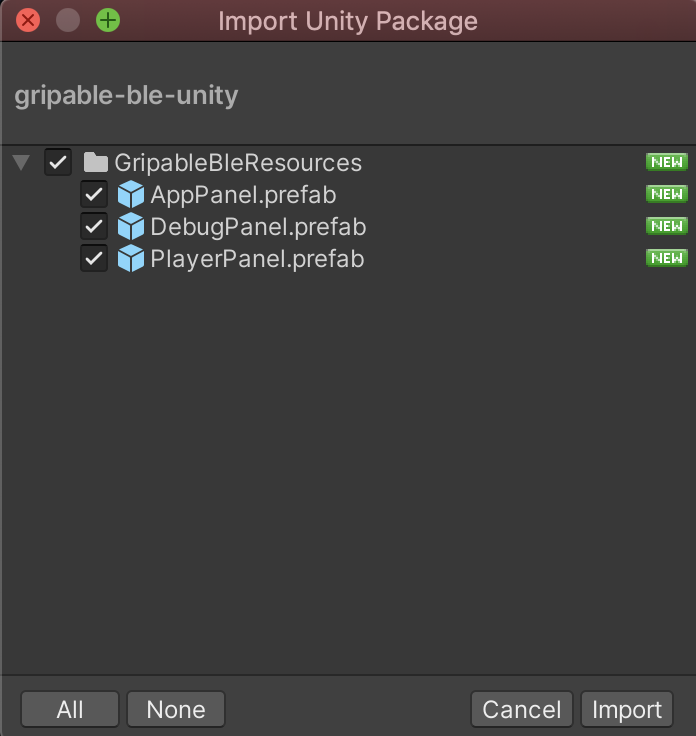


**NOTE: THE FOLLOWING STEP IS OBSOLETE SINCE BLE-UNITY 4.0.0, ONWARDS. INSTEAD, JUST DRAG THE PREFABS DIRECTLY FROM THE “PREFABS” FOLDER INTO YOUR SCENE**

The final step is to import the debugging prefabs from the .unitypackage file within this new Packages folder. Navigate to Packages -> Gripable BLE Unity and double click on the gripable-ble-unity.unitypackage file



This will open up the Import Package window in Unity. Make sure all the options are selected and press the “Import” button:



The GripableBleResources folder will now appear inside your Project’s Assets folder. This contains 3 prefabs dedicated to debugging the GripAble device.

### 

### 

### Building

In order to build to an Android device, we need to set up the correct build environment.

* In Unity, open File -> Build Settings…
* Ensure Android is the current platform (if not, select “Android” and then press the “Switch Platform” button)
* Press the “Player Settings…” which will open the Project Settings -> Player window
* Under “Other Settings” make the following changes:
  + Set the Minimum API Level to Android 7.0 ‘Nougat’ (API Level 24)
  + Set Target API Level to Android 9.0 ‘Pie’ (API Level 28)
  + Ensure API Compatibility Level is .NET 4.x
  + Set Install Location to “Force Internal”

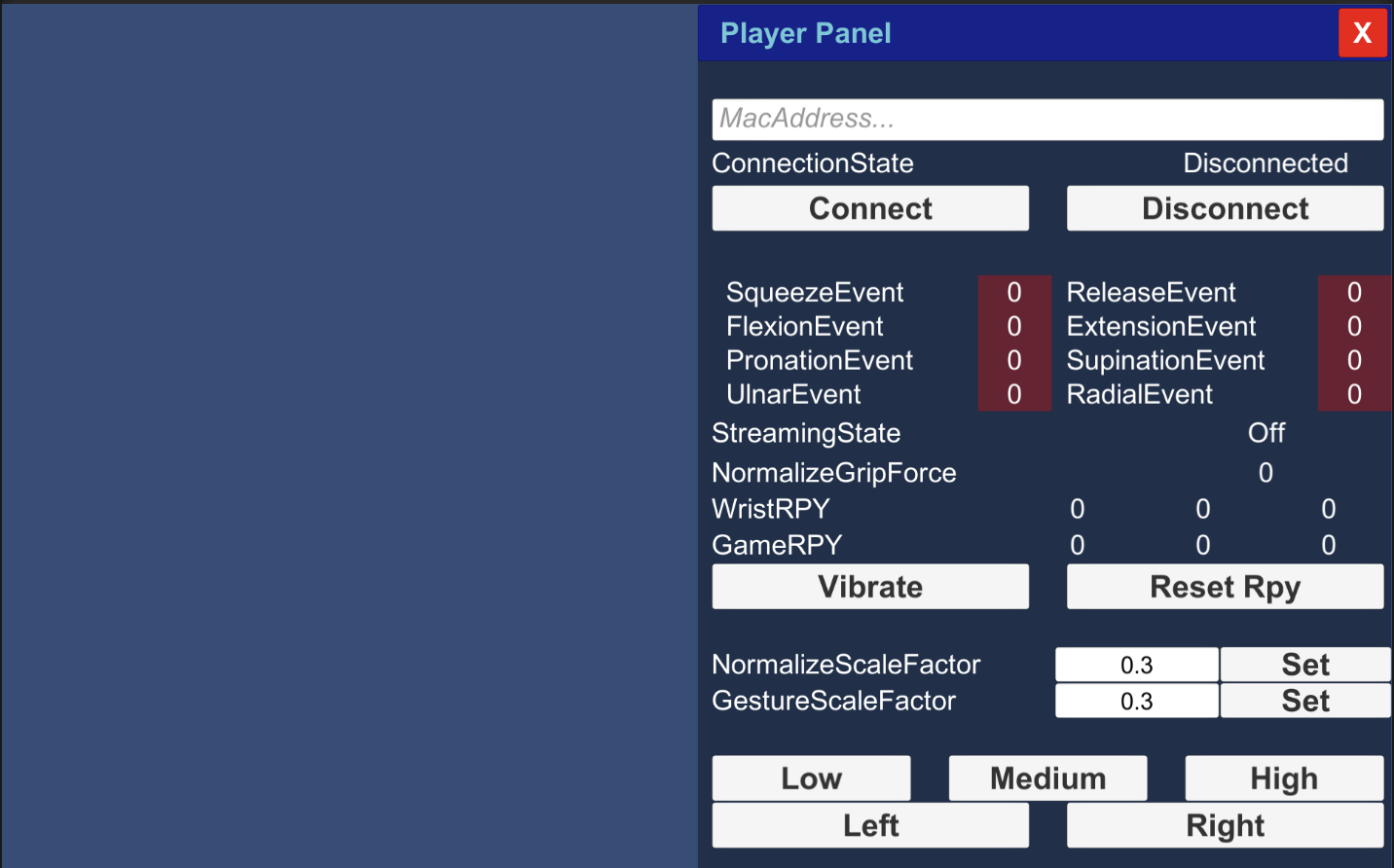
### Testing

We will set up a testing method to make sure the GripAble is outputting data correctly to Unity.

**TEMPORARY: Each time you restart your tablet you will need to scan for the GripAble device using the Android OS.** For this, go to Bluetooth settings on your tablet and let the scan run (no need to select the actual device from the list).

Simply drag and drop the PlayerPanel prefab from Assets > GripableBleResources into the scene. The best scene to place this into is your game’s entry point. The prefab has a DontDestroyOnLoad option so it can persist across all scenes.

You can then run a build on your Android device. Tapping the screen with 4 fingers will bring up the player panel on the right hand side.



If you already have other Canvas objects they may obscure the panel by rendering in front of it.

Here you can input the GripAble Device’s Mac Address and press the connect button.

Once the device connects, the panel will automatically hide itself. Bring it back up by tapping the screen with 4 fingers again. Select a calibration preset level from the 3 options below (Low, Medium or High). This should start outputting RPY data in the panel. This means the GripAble device is sending data to the Unity Game!

## Usage

There are several incoming data points which one can use as control input for their game.

The API is accessible via the static Gripable class. Anywhere in your game code you would to access the library include this line at the top of the c-sharp class:

|  |
| --- |
| **using** Gripable; |

Then the API is accessed via the GripablePlugin.Player singleton

### Connect

If you would like to connect to a device from code you can use the Connect() function:

|  |
| --- |
| GripablePlugin.Player.SetDevice("XX:XX:XX:XX:XX:XX"); GripablePlugin.Player.Connect(); |

### Calibration

The Gripable device is calibrated to the player who uses it. There are some default calibrations that can be altered using the PlayerPanel prefab (Low, Medium & High). These should be sufficient for testing and playing games so the rest of this section is very optional. Should you need to tweak these values further, please read on...

There is a calibration for each type of motion; grip force, roll, pitch & yaw and a normalize scale factor. Each calibration has a maximum value, a minimum value and a type. The maximum and minimum values are in degrees (up to 360) for roll, pitch & yaw (RPY) and in kilograms for grip force. If you would like to make your own calibration you can manually set these from code:

|  |
| --- |
| GripablePlugin.Player.SetCalibration(  **new** Calibration { Min = 0, Max = 10, Type = MovementType.Grip },  **new** Calibration { Min = 315, Max = 60, Type = MovementType.Roll },  **new** Calibration { Min = 345, Max = 40, Type = MovementType.Pitch },  **new** Calibration { Min = 310, Max = 30, Type = MovementType.Yaw },  **0.4f**); |

The normalize scale factor is used to calculate the outputs of the normalized get functions below. The higher this value, the bigger the range of movement with lower values being more sensitive. The normalize scale factor can be set separately from the entire calibration like so:

|  |
| --- |
| GripablePlugin.Player.SetNormalizeScaleFactor(0.4f); |

Using the DebugPanel prefab (placed into the scene in the same way the PlayerPanel prefab has been) will display the raw degrees and kilogram values from the Gripable. This will help setting calibration values should you want to do this.

### Grip Force

This function will get the input of the device’s grip sensor normalized by the current calibration setting to a float value between 0 and 1.

|  |
| --- |
| **float** gripForce = GripablePlugin.Player.GetGripForce(); |

### Roll, Pitch and Yaw (RPY)

The rotation of the device is described using [Aircraft Principal Axes](https://en.wikipedia.org/wiki/Aircraft_principal_axes). Each value is accessed separately and returns a degrees value up to 360 degrees, which is then normalized by the current calibration setting to a float between -1 and 1.

|  |
| --- |
| **float** roll = GripablePlugin.Player.GetRoll(); **float** pitch = GripablePlugin.Player.GetPitch(); **float** yaw = GripablePlugin.Player.GetYaw(); |

### Zeroing / Resetting RPY

It is recommended before each time a game or level starts, that the player is holding the device in an upright position or sideways position (for flexion/extension exercises):

### 

Calling the ResetWristRpy() function will zero the RPY values to whatever position the device is currently held. This is very useful to ensure consistency of control between play sessions. We have found it best to combine the function with the release gesture before a level starts. This can be done by replacing the “DoSomething()” function of the example in the following section with this line:

|  |
| --- |
| GripablePlugin.Player.ResetWristRpy(); |

### Gestures

Gestures are single-fired events that can be subscribed to from within your code. Each gesture can be monitored inside the PlayerPanel prefab. Here is an example of how you could use the release gesture (when the player releases their grip of the device’s grip sensor) to do “something”:

|  |
| --- |
| **using** Gripable;  **void** **Start**()  {  GripablePlugin.Player.OnGesture += ListenForRelease;  }  **private** **void** **ListenForRelease**(Gesture gesture) {  **if** (gesture.Type == GestureType.Release)  {  DoSomething();  } } |
|  |

Gestures are based on the minimum and maximum ranges of the current calibration. This is then multiplied by the Gesture Scale Factor. The gesture scale factor can be tweaked in the PlayerPanel prefab or from code:

|  |
| --- |
| GripablePlugin.Player.SetGestureScaleFactor(0.3f); |