htc

VIVE WAVE

Build Unity OpenXR applications on VIVE Focus3

Mar, 2022

Revision History

Revision	Date	Description
1.0	November 2021	Initial release
1.1	March 2022	Added Vulkan support Added a known issue and workaround

Contents

1. Introduction

VIVE Wave runtime is going to support OpenXR standard on VIVE Focus3. If you are reading this development guide now, it means that you are the close partners with Wave, and welcome to join OpenXR Beta trial of Wave runtime.

Since Unity 2020.3, Unity engine released the OpenXR Plug-in which allowed to develop the cross-platform applications for OpenXR. In this guide, you will learn how to build an OpenXR app running on Focus3 through the following step-by-step sections.

What you will learn in this guide:

- Set up OpenXR development environment for your Unity project
- Configure VIVE Wave and Focus3 profile into your XR application
- Build and run Unity XR application on VIVE Focus3

What Wave support on Focus3 in this release version:

- HMD tracking pose and XR rendering
- Controller tracking pose, key input, and haptics

Okay. Let's get started to create your 1st VIVE Wave OpenXR content on Focus3!

2. Development Environment

2.1. Prerequisite

Unity: Version 2020.3.15f2 and 2020.3.30f1 are verified and recommended.

Note

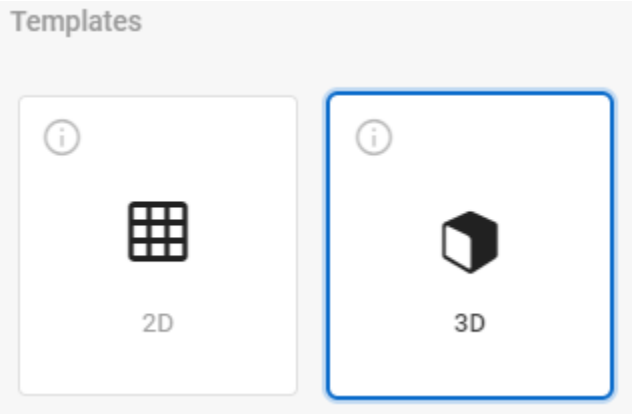
This guide uses Unity2020.3.30f1 with OpenXR Plugin 1.3.1 as the example.

(Optional) Android Studio: 181.5014246-windows

Note

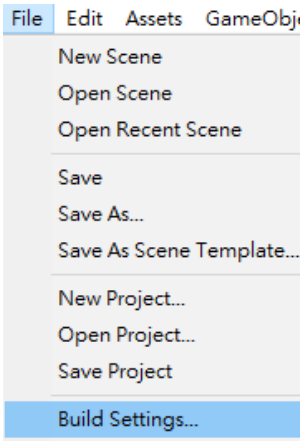
Android Studio is used to configure your Android SDK path to Unity. See detail in Chapter 6. **You can skip if your Unity env is ready for Android already.**

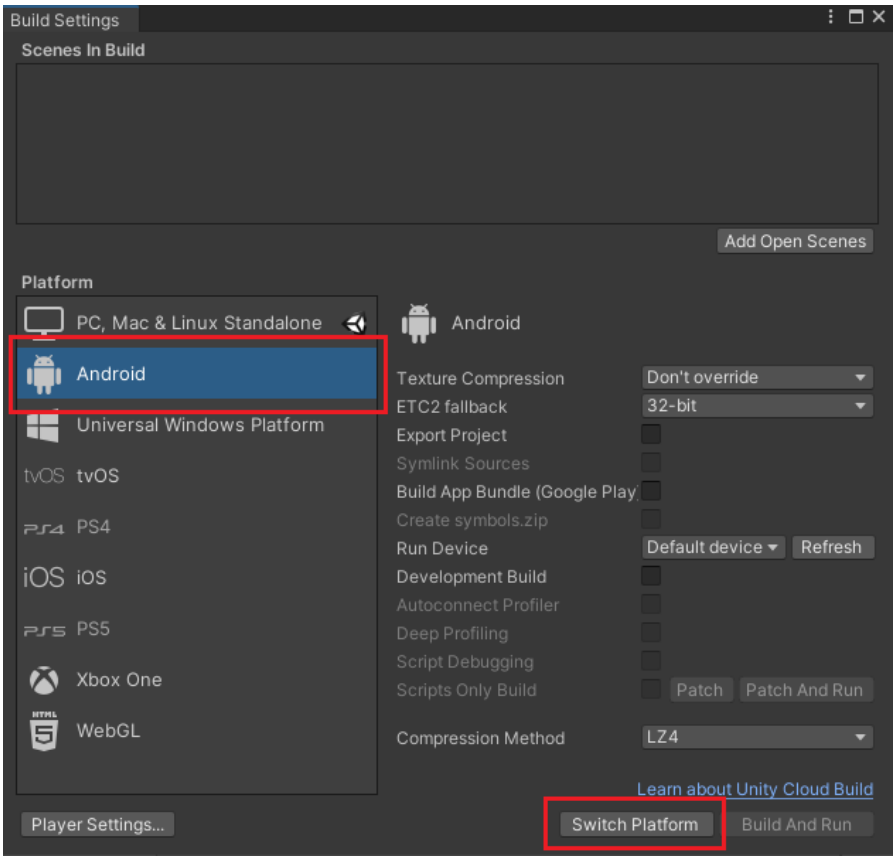
2.2. Create a new Unity 3D project



2.3. Build Settings

From menu File > Build Settings, switch platform to **“Android”**.





3. Package Manager

3.1. Register source server

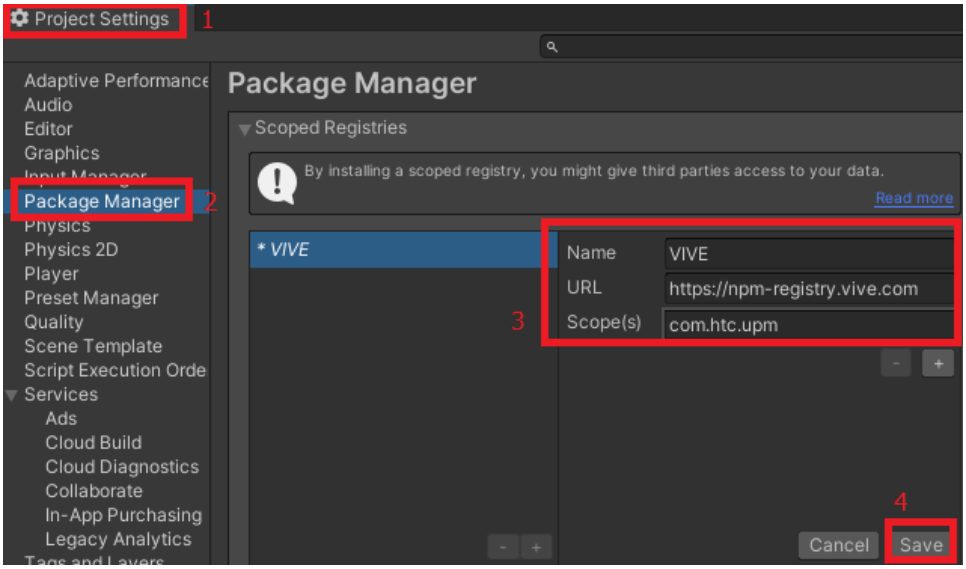
1 From menu Edit > Project Settings

2 Select Package Manager

3 Input VIVE Registry

Name	VIVE
URL	https://npm-registry.vive.com
Scope	com.htc.upm

4 Apply the change.



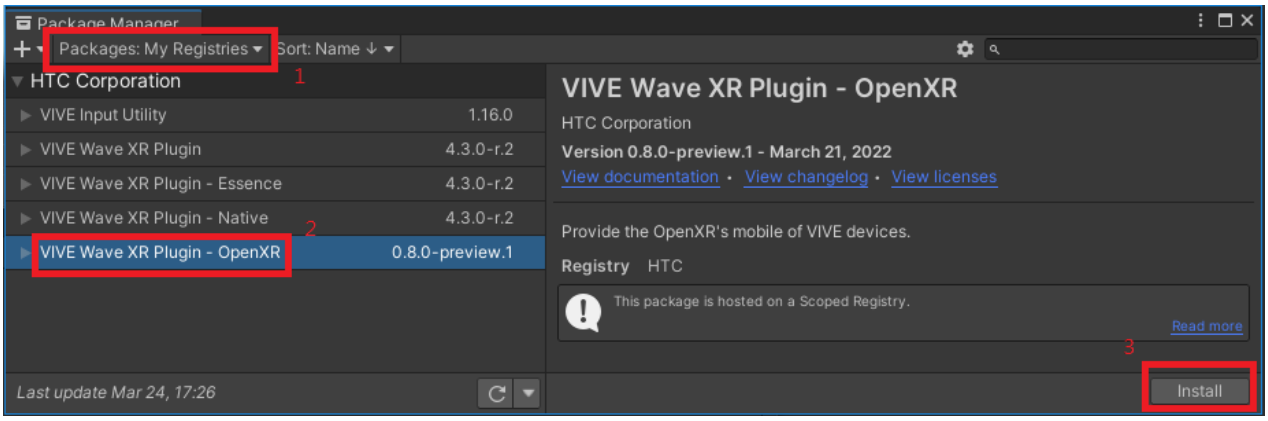
Note

All space in “URL” and “Scope(s)” must be trimmed.

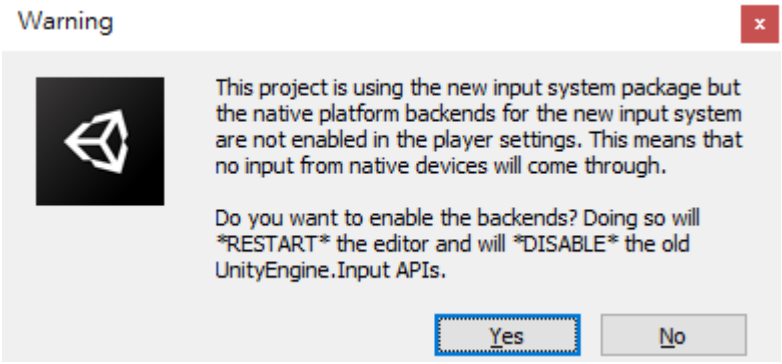
3.2. Install Wave OpenXR Feature

From menu: Window > Package Manager

1 Check sources in "My Registries".
2 Install "VIVE Wave XR Plugin - OpenXR".



3 Select “Yes” to use the Input System backends.

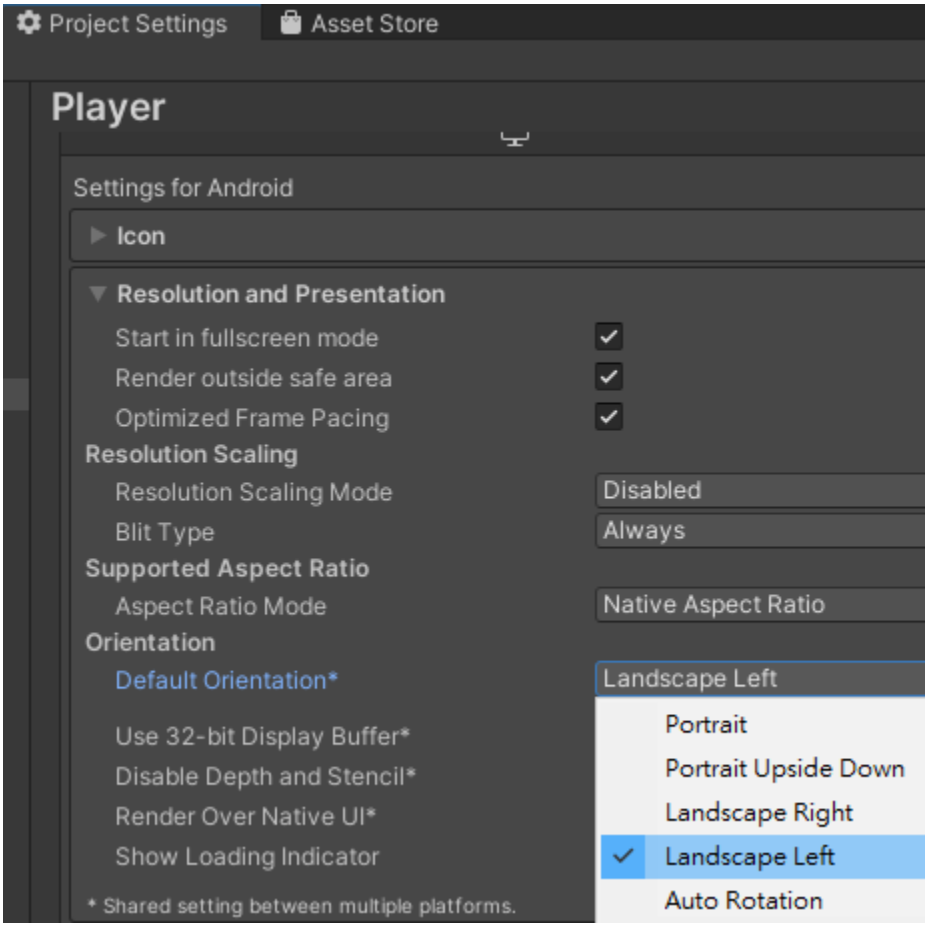


4 These dependent packages will be also installed:
  4.1 InputSystem 1.2.0
  4.2 XR Plugin Management 4.2.1
  4.3 OpenXR Plugin 1.3.1

4. Player Settings

4.1. Change Display Orientation

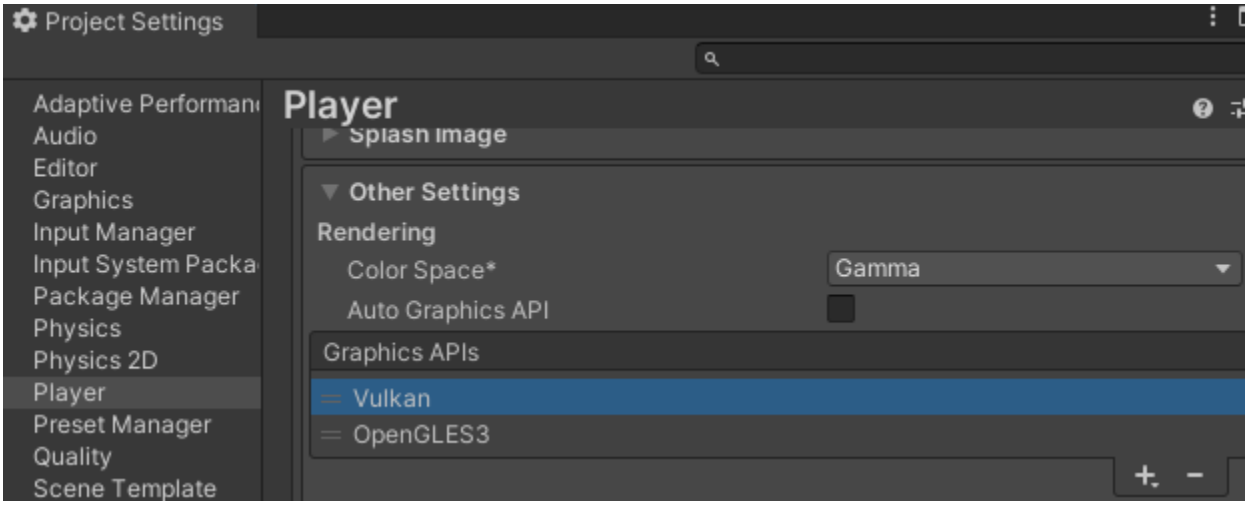
From Edit > Project Settings > Player > Resolution and Presentation > Orientation > Default Orientation, set to **Landscape Left**.



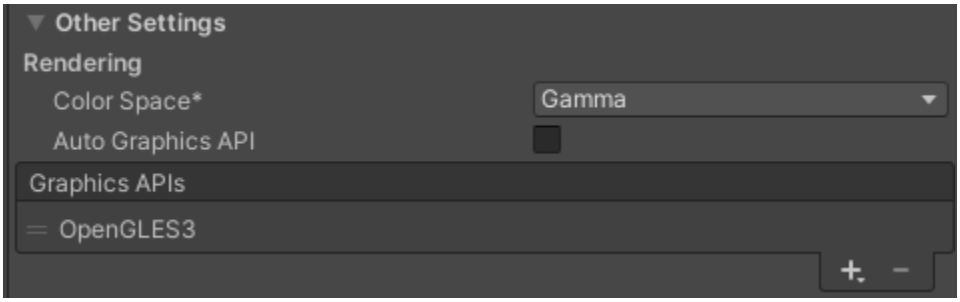
4.2. Choose Graphics APIs

Open Edit > Project Settings.

Choose Graphics APIs from Player > Other Settings > Graphics APIs.



Choose either **Vulkan** or **OpenGLES3**.



Note

When choosing "**OpenGLES3**", the Player > Other Settings > Rendering > Color Space should be Linear.

4.3. Set Android API Level

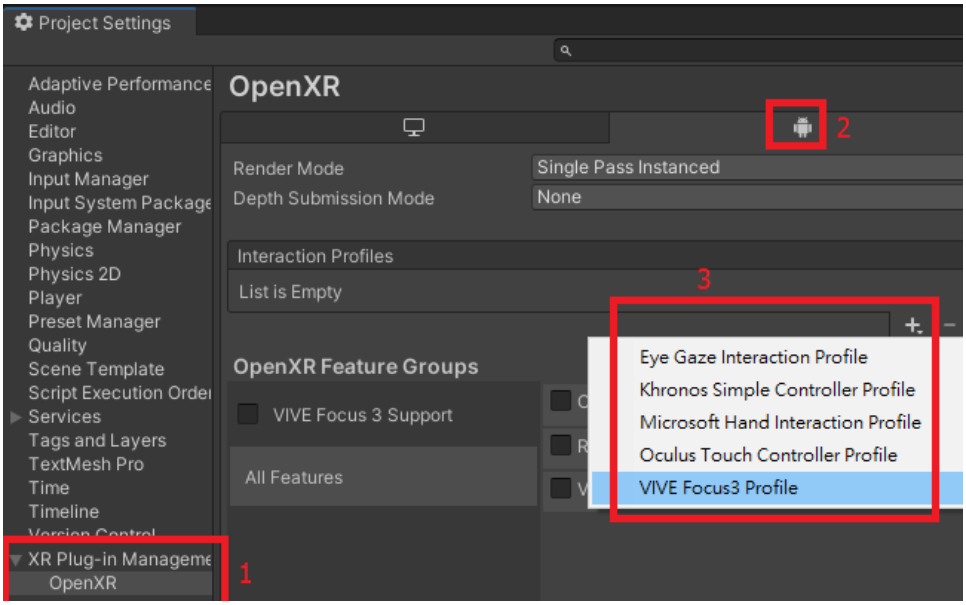
Set the API level from Player > Other Settings > Identification.

5. XR Plug-in Management Settings

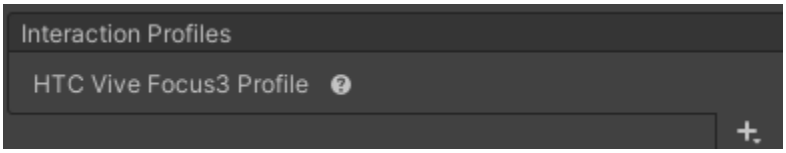
5.1.Set Interaction Profile

From Edit > Project Settings

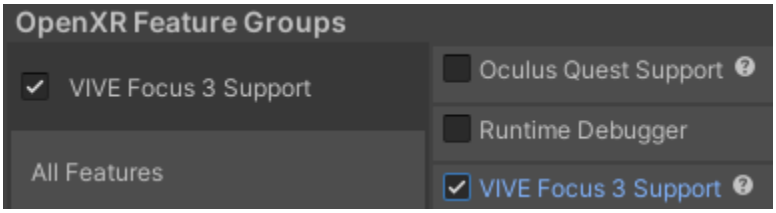
1 Find **OpenXR** page in XR Plugin-in Management
2 Select **Android** platform tab
3 Press "**+**" button of Interaction Profiles.
4 Select "**HTC Vive Focus3 Profile**"



Check the profile is added successfully.

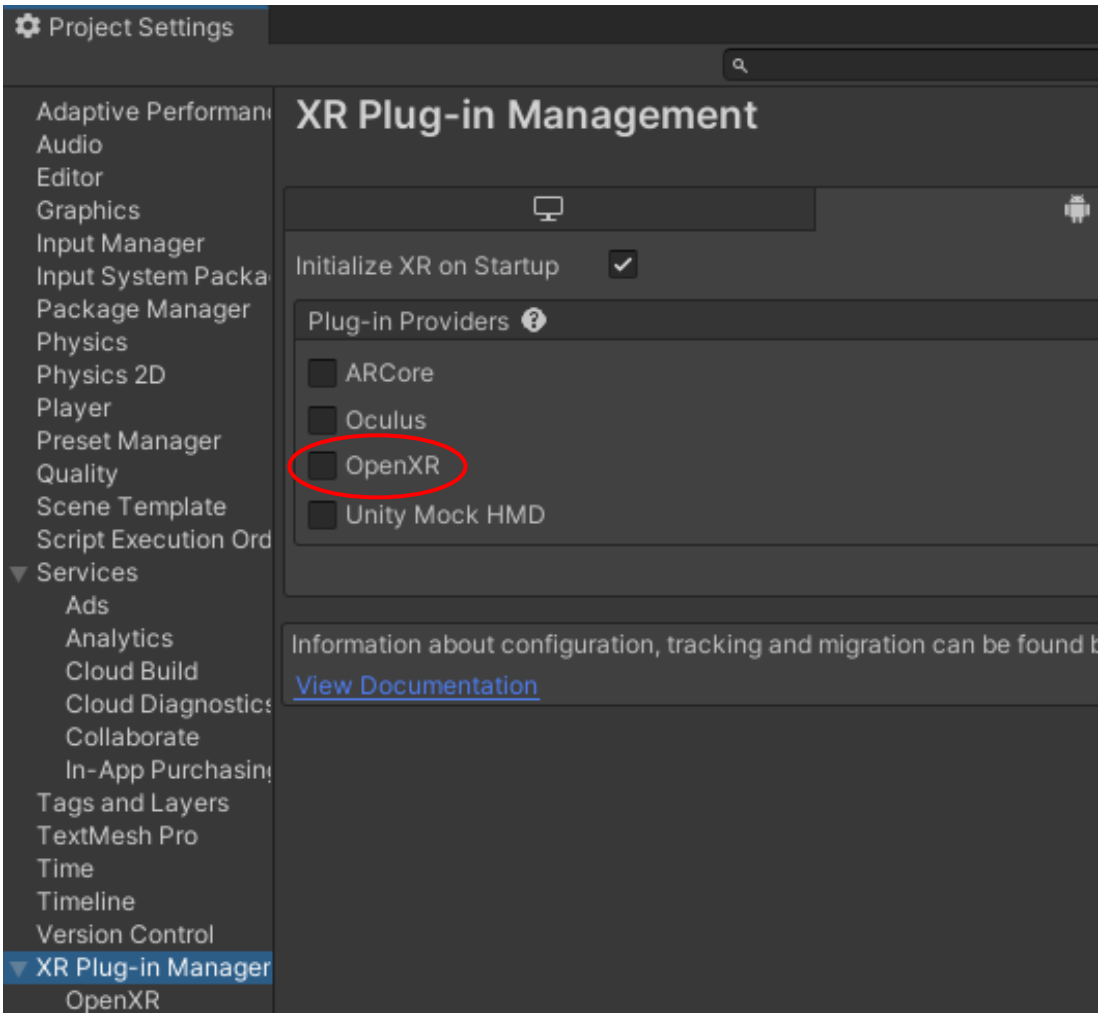


5 Check "**VIVE Focus 3 Support**" in OpenXR Feature Groups



5.2. Enable OpenXR Provider in Android

From Edit > Project Settings > XR Plugin-in Management > Android platform, check **"OpenXR"**.

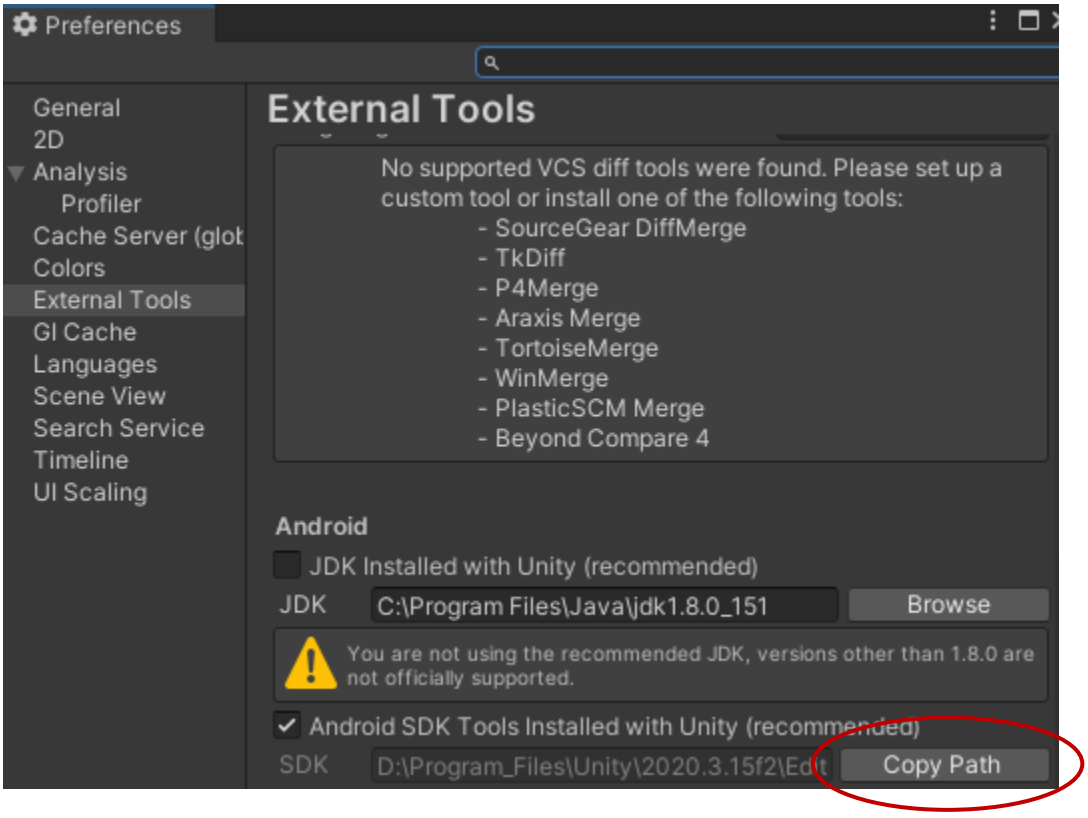


6. Set up Android SDK

This chapter is targeted for developers who never experience in Android development. You could skip it if you already set up Android environment.

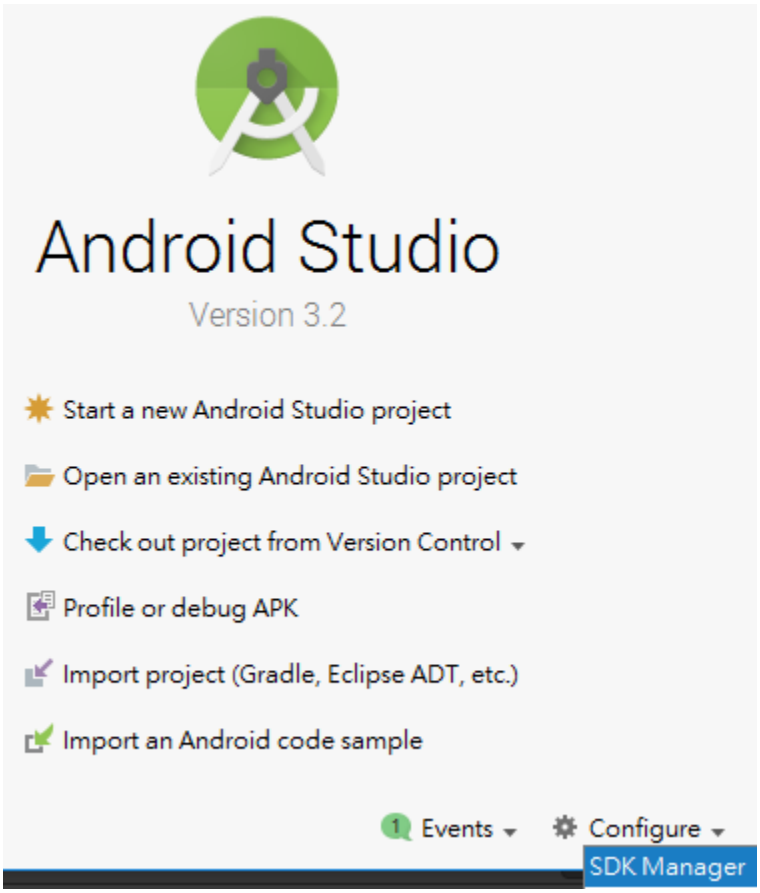
6.1. SDK Path

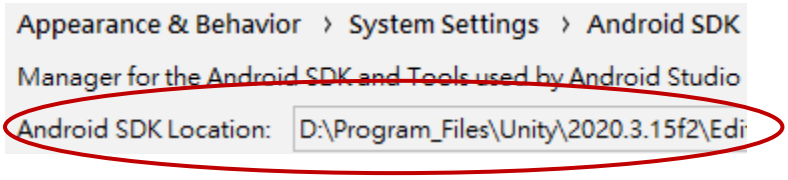
From Edit > Preference > External Tools, check the "Android SDK Tools Installed with Unity (recommended)" and copy the SDK path.



6.2. Android Build-Tools 29.0.2

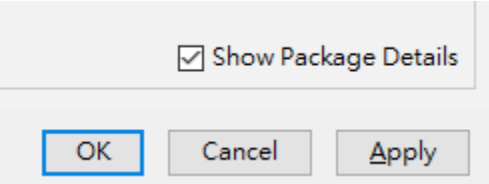
Install the "Android SDK Build-Tools 29.0.2" to the Android SDK by using the Android Studio.



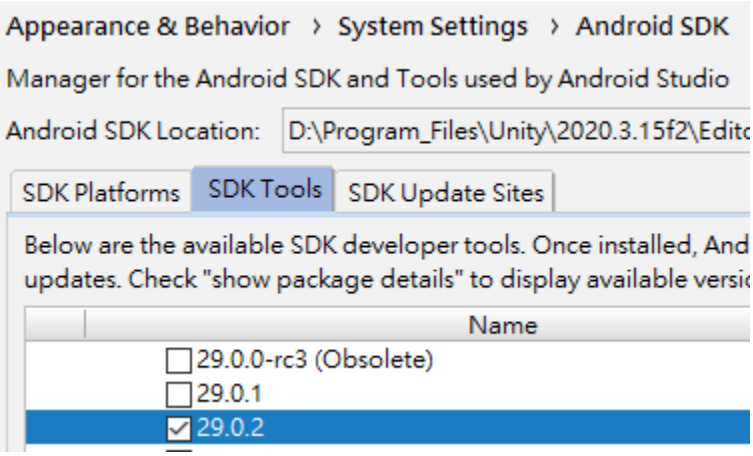


Specify the Unity used SDK path.

Select “Show Package Details” then you will see the list of build tools.



Select the Built-Tools “29.0.2” and click “Apply”.



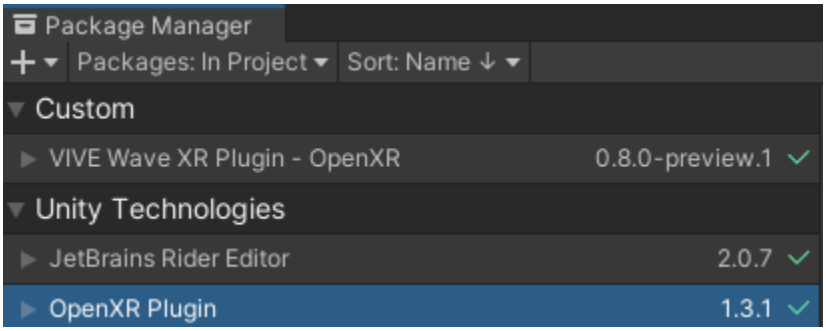
6.3. Configure Android Built-Tools 31.0.0 (Optional)

If you used the Android build-tools 31.0.0, we noticed some problems. The version is too high, and Unity didn't adapt it well. Therefore you would need to make some changes to the tools.

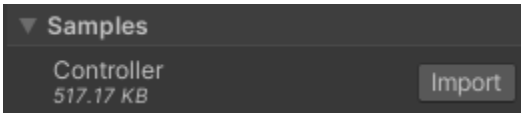
- In the <SDK path>\build-tools\31.0.0, rename "d8.bat" to dx.bat".
- In the <SDK path>\build-tools\31.0.0\lib, rename "d8.jar" to "dx.jar".

7. Build and Run Unity OpenXR Sample

After Step 3.2, you can see the **“OpenXR Plugin”** package from Window > Package Manager > Packages: In Project

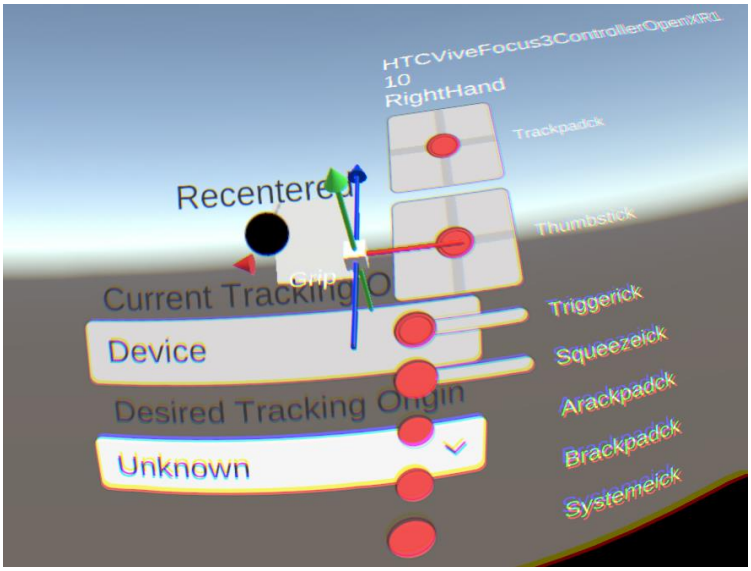


Import the Controller sample.



You can find the Controller sample in Assets > Samples > OpenXR Plugin > 1.3.1 > Controller. Open the sample scene and go to File > Build Settings to build the Android APK.

After installing this sample APK to Vive Focus3, you can see the controller indicator and panels.



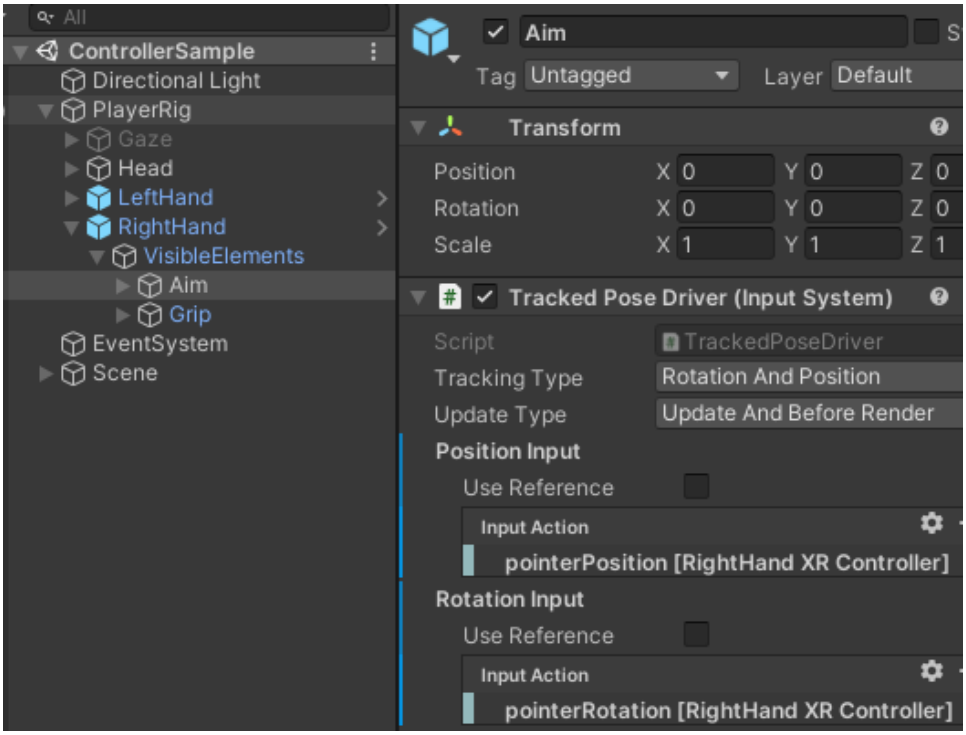
8. Known Issue

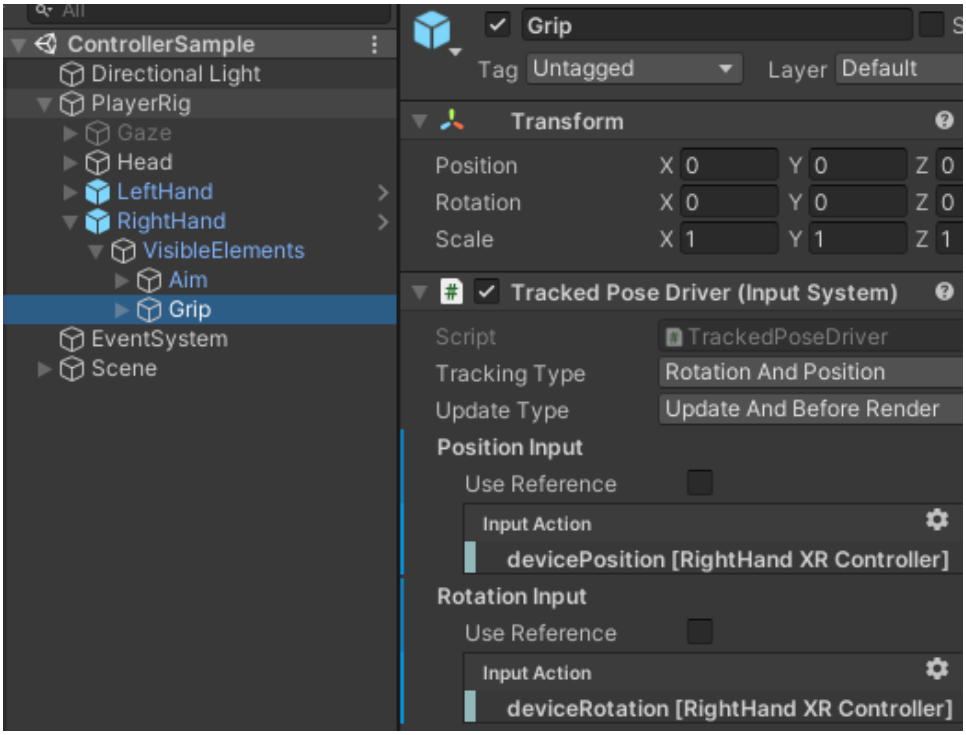
1. The imported Controller sample cannot switch "Desired Tracking Origin".

[How to fix] Modify the **TrackingModeOrigin.cs** line.81 from & to &&.

```
!= m_DesiredTrackingOriginMode && m_DesiredTrackingOriginMode !=
```

2. The OpenXR 1.3.1 Controller sample uses Left controller's pose on Right Aim/Grip GUIs. You have to configure the the path as below.





3. The imported Controller sample dropdown list cannot be clicked.

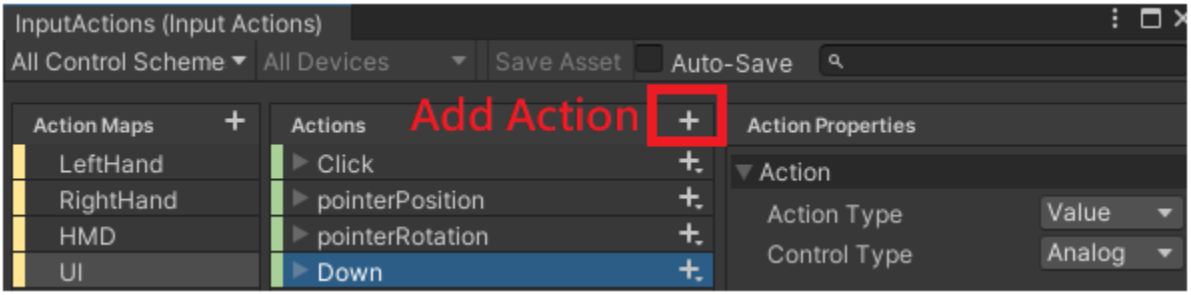
[How to fix] The root cause is that the **Input System (1.2.0) UI Input Module** does NOT send the **Click** event. To click the dropdown list, we can send the **Submit** event instead of **Click** event.

However, the **Submit** event is sent only when the target object already received a **Down** event. We need to send a **Down** event before sending a **Submit** event.

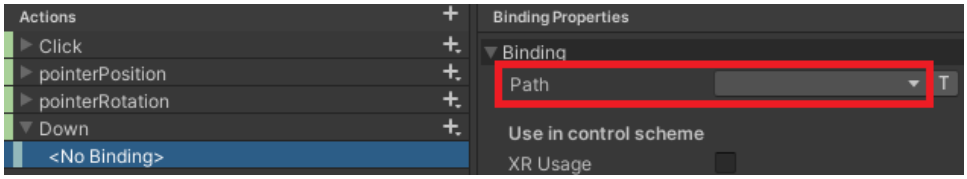
To accomplish the event flow, we need to send a **Down** event when the trigger axis is bigger than 0.5f and send a **Submit** event then the trigger key is pressed.

Follow the steps below:

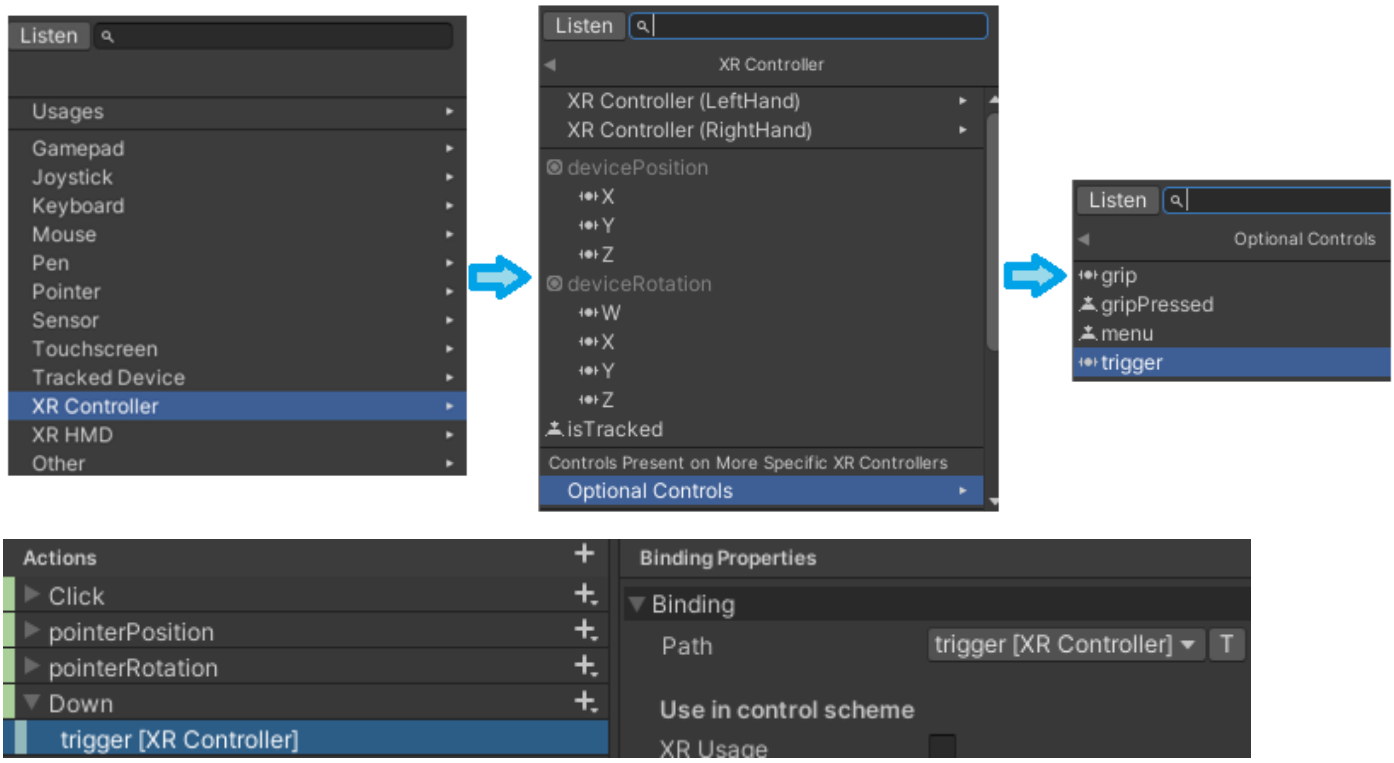
a. Open the <path_to_sample>/ControllerSampleActions.inputactions.
b. Select the **Actions Maps** “UI” and add an action named “Down” with the **Action Type** “Value” as well as **Control Type** “Analog”.



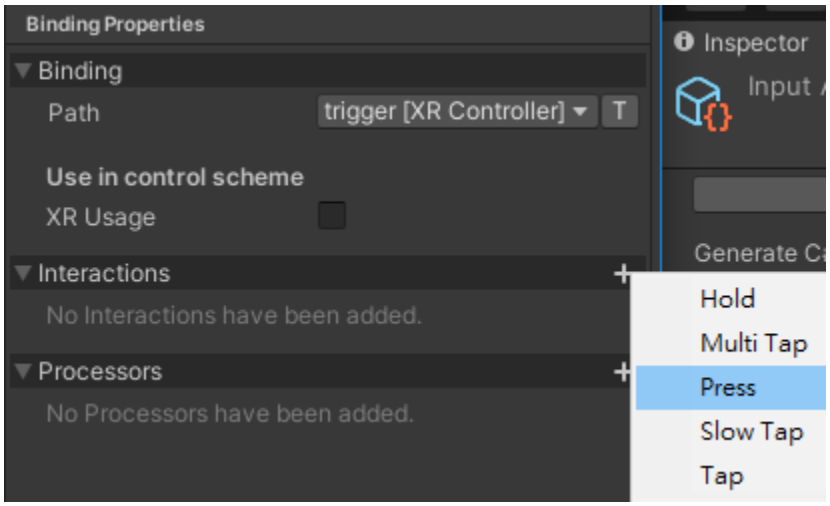
c. Select the **Path**“ to add a binding.

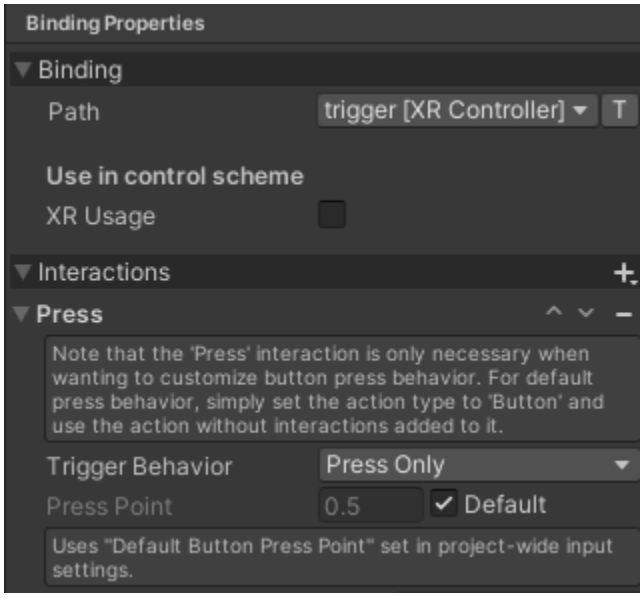


d. Set the binding path to “trigger [XR Controller]” by specifying the **Path** to XR Controller > Optional Controls > trigger.

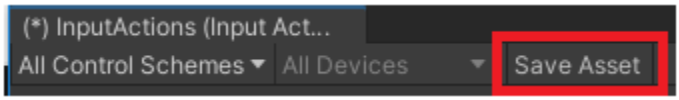


e. Click the **Interactions** “+” to add a “Press” interaction.

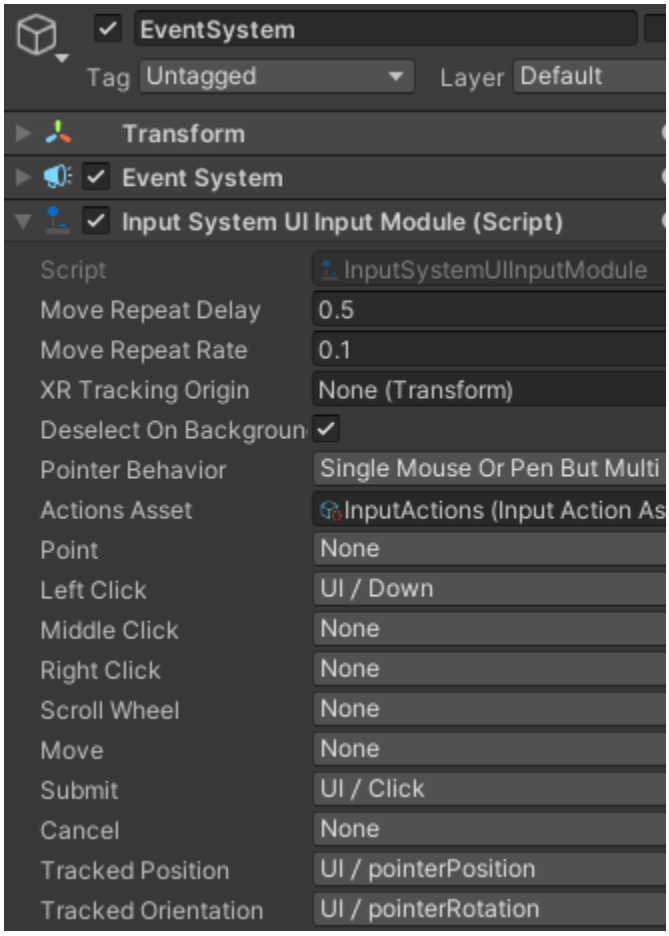




Click “Save Asset” to finish.



f. In ControllerSample > EventSystem, set the **Left Click** to “UI / Down” and **Submit** to “UI / Click”.



9. Troubleshooting

1. The application does NOT run in VR mode.

   Please check the following factors in your Unity project:

   - Ensure at least one interaction profile
   - Ensure OPENXR permission added in AndroidManifest

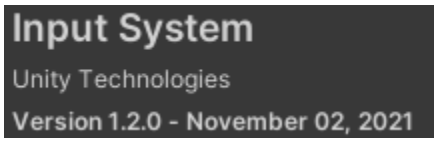
     ```
     <uses-permission android:name="org.khronos.openxr.permission.OPENXR" />

     <uses-permission android:name="org.khronos.openxr.permission.OPENXR_SYSTEM" />
     ```

   - Ensure libopenxr_loader.so imported in Plugins/Android

2. Cannot click the dropdown list in Controller sample.

   Please check the version of "**Input System**" package is 1.2.0 from Window > Package Manager" and follow step 8.2 to solve the problem.

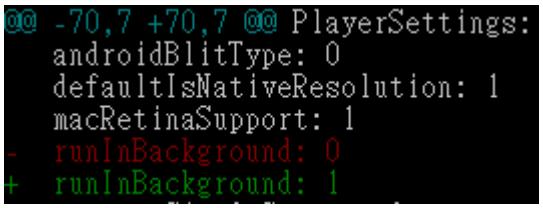
3. Cannot get the headset and controller pose in Controller sample.

   Please check your Unity version. DO NOT use 2020.3.18f1.

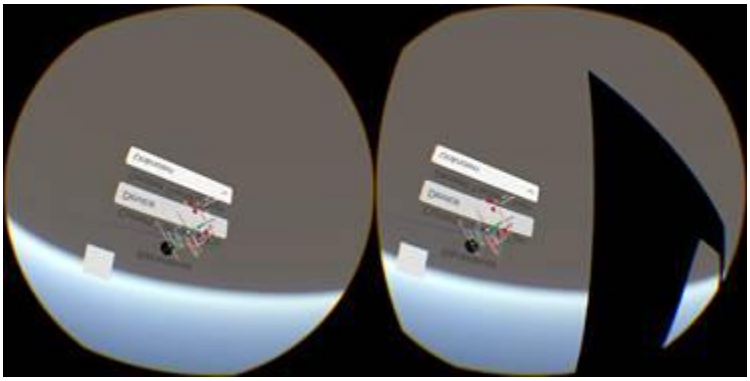
   2020.3.15f2 and **2020.3.30f1** are our recommended version for this early access trial.

4. **The scene is frozen.**

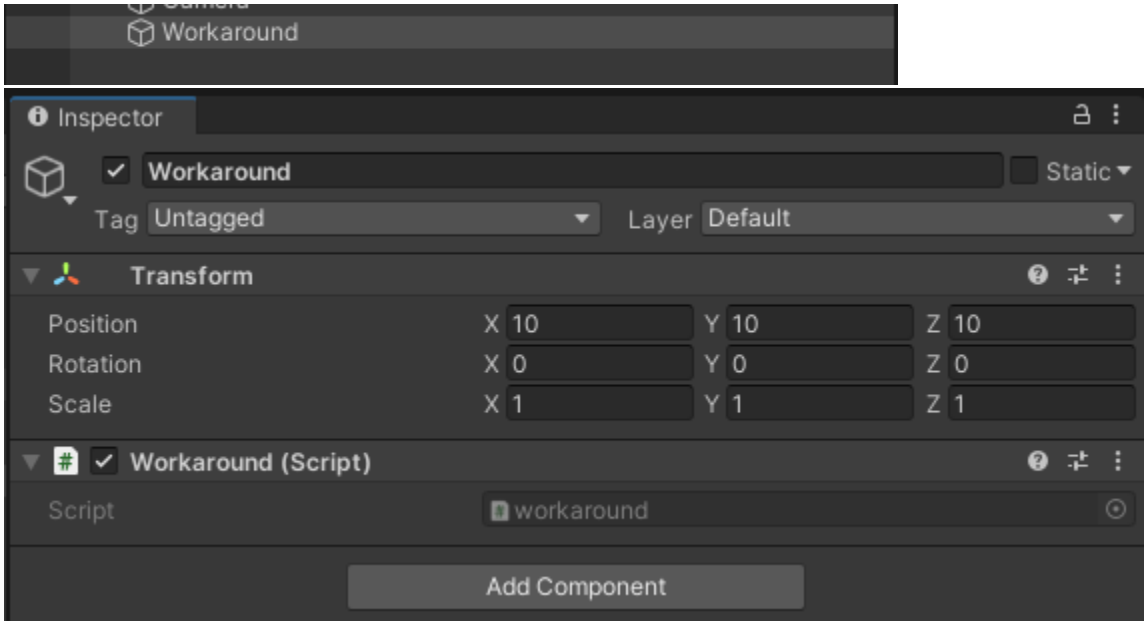
   Modify the file *ProjectSettings > ProjectSettings.asset* file. Change the value of "runInBackground" from 0 to 1.

5. For Unity OpenXR single-pass Vulkan case, there will be a Visibility Mask on the right eye. (We are clarifying this issue.)



(Workaround) We can add an object with a script component to hide the visibility mask. We need to **set XRSettings.occlusionMaskScale to 0.0f** at Start and after Resume (need skip some frames).



The script is as below.

workaround.cs

Assembly-CSharp

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.XR;
5
6   public class workaround : MonoBehaviour
7   {
8       bool isResume = false;
9       int counter = 0;
10      // Start is called before the first frame update
11      void Start()
12      {
13          XRSettings.occlusionMaskScale = 0.0f;
14      }
15
16      void OnApplicationPause(bool pause)
17      {
18          if (pause == false)
19          {
20              counter = 0;
21              isResume = true;
22          }
23      }
24
25      // Update is called once per frame
26      void Update()
27      {
28          if (isResume)
29          {
30              counter++;
31              if (counter >= 10)
32              {
33                  XRSettings.occlusionMaskScale = 0.0f;
34                  isResume = false;
35              }
36          }
37      }
38  }
39
```