Multiplayer VR Asset Documentation

Thanks again for downloading the asset.
As **IRONHEAD Games**, we will do our best to help you set up the asset and provide the support needed in the future.

This asset uses **PUN 2- FREE** and **Photon Voice 2** assets for Multiplayer and uses **Unity XR Interaction Toolkit** package for VR functionality.

Also, this asset is compatible with at least Unity 2020.3 LTS. So, make sure not to use alower version than Unity 2020.3 LTS!

The suggested Unity version is Unity 2020.3 LTS.

You are welcome to ask your questions to email address:
tevfikufuk@ironheadgamestudios.com

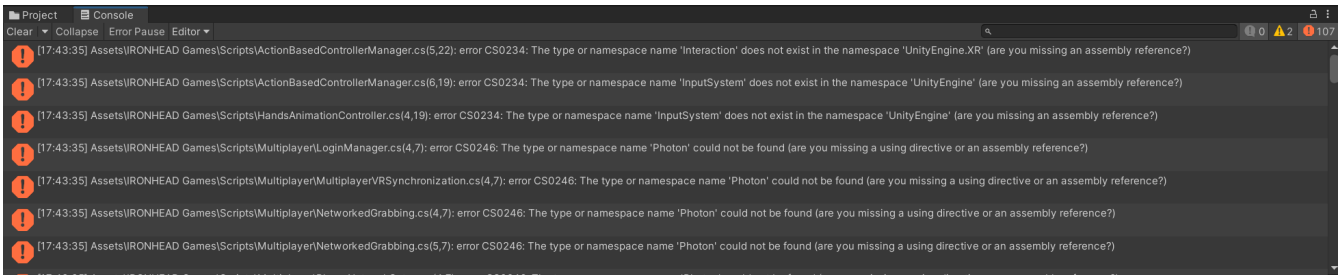
Also, here is the Asset's Discord server to get in touch better:
https://discord.gg/9KtmrcfpBw

You can follow the below steps to set up the asset. Good luck!

How to Set Up the Asset

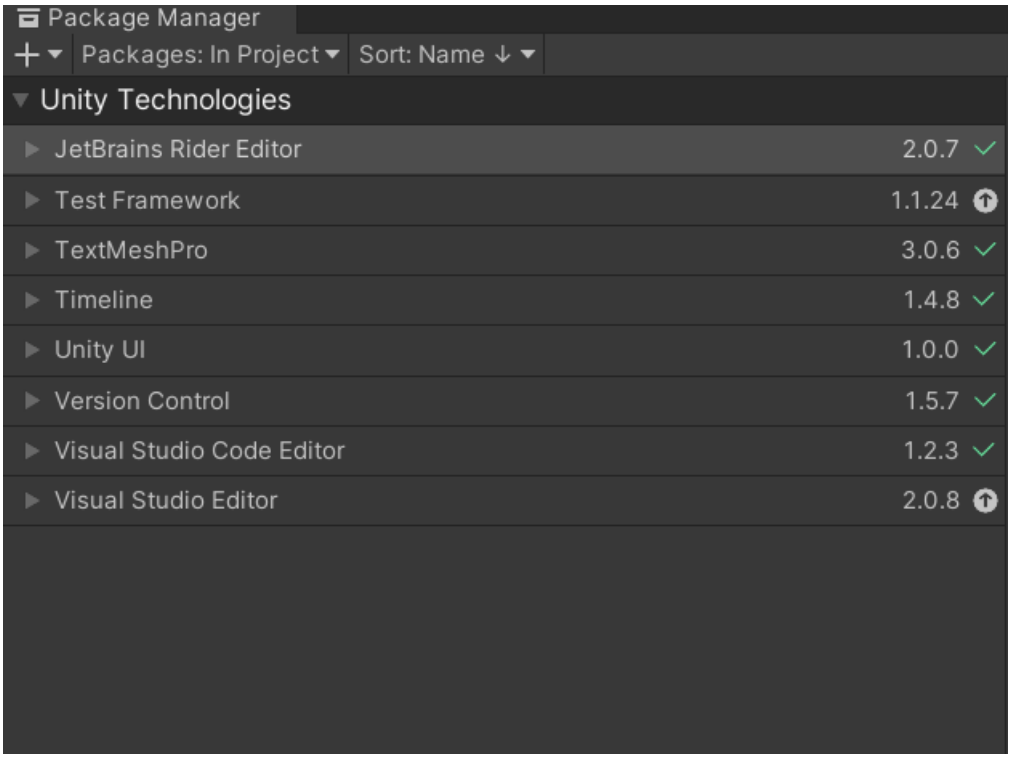
Importing the Asset and Configuring Photon Settings

1. Firstly, create a new fresh Unity project. Then, import the asset via Package Manager. After importing the asset, you may see lots of errors on Unity Console.

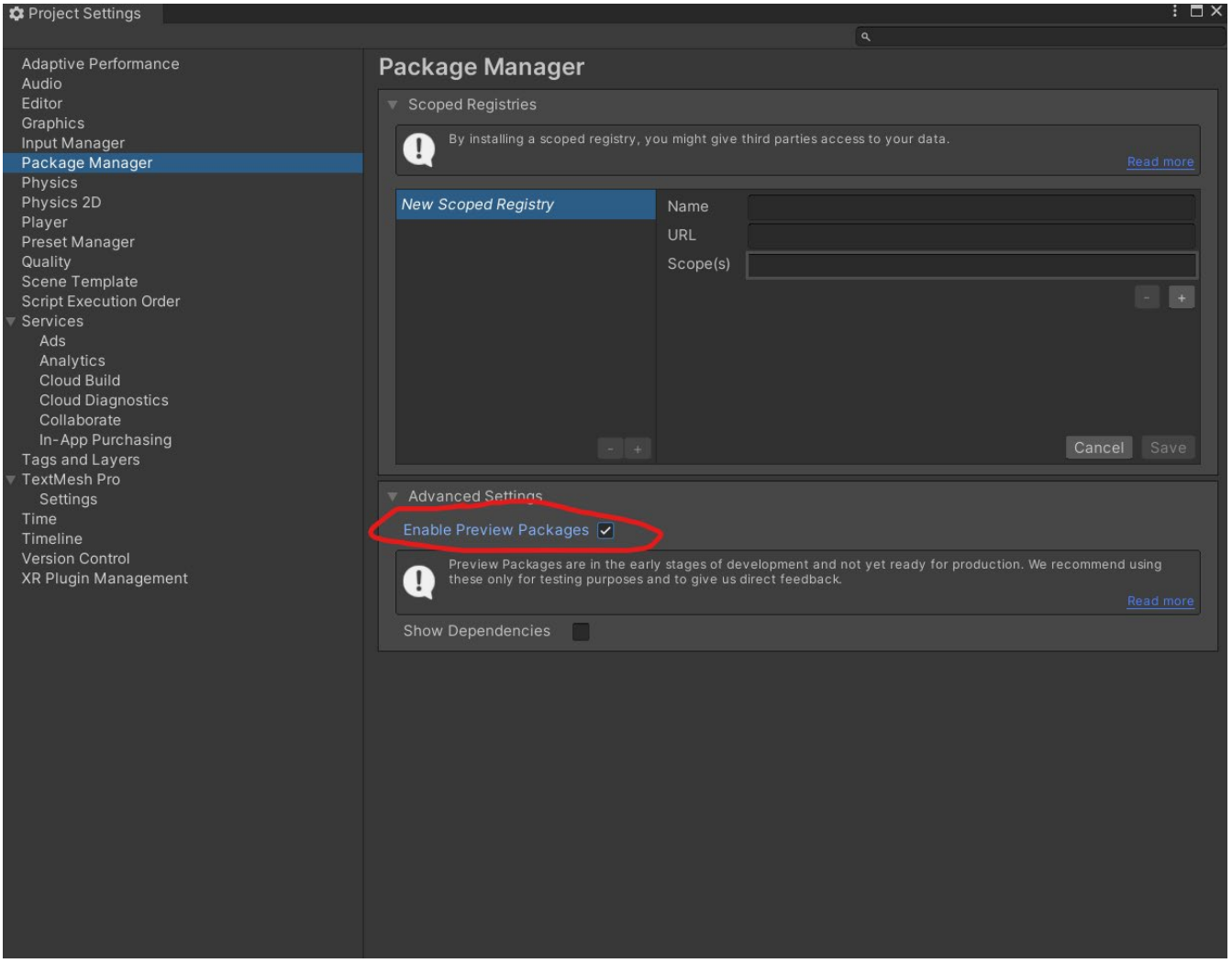


This is very normal, and we will fix it soon.

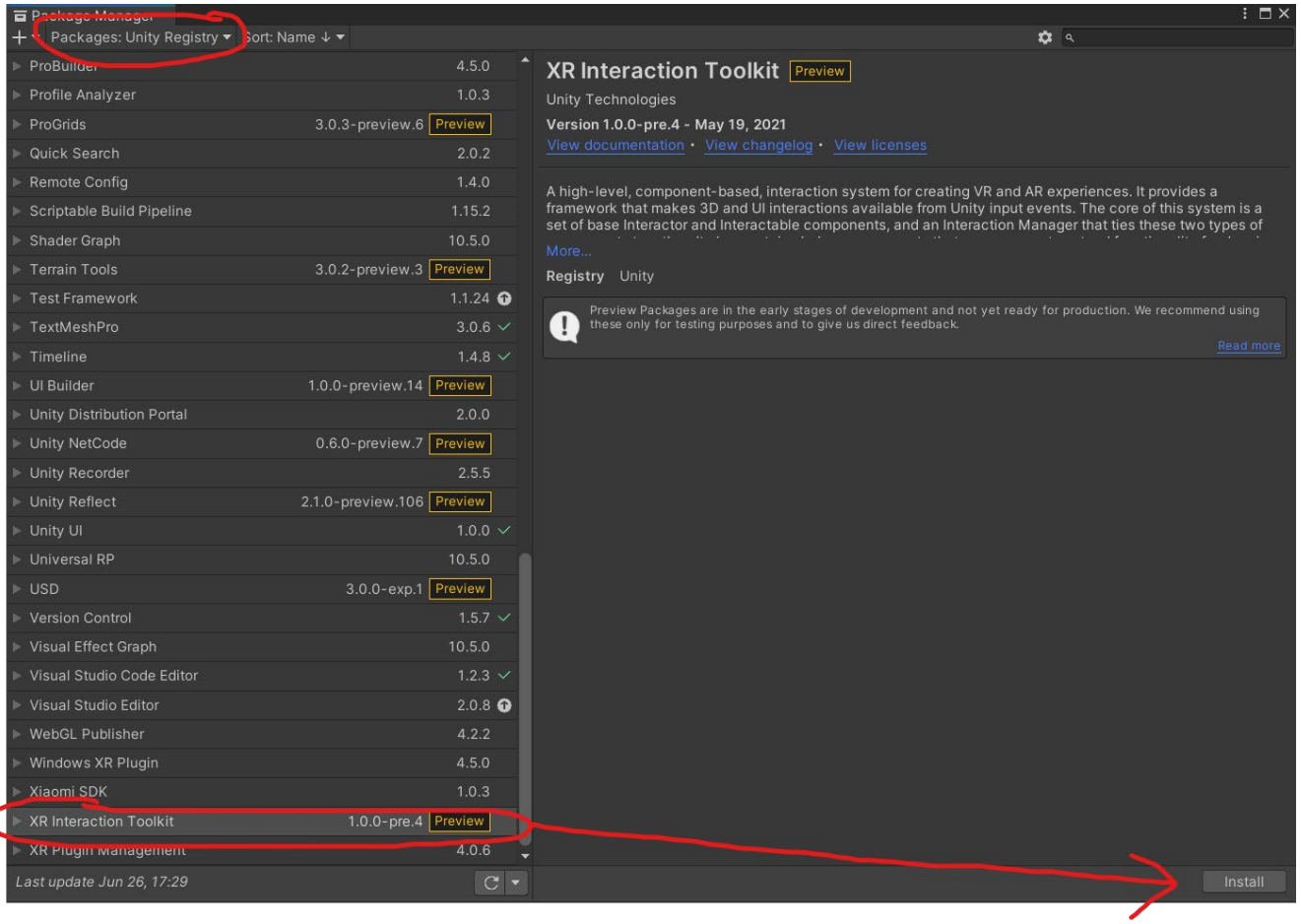
First, we will import Unity's XR Interaction Toolkit if it is not imported automatically when you imported the asset. To check this, you can open Window > Package Manager and see if XR Interaction Toolkit is installed or not.



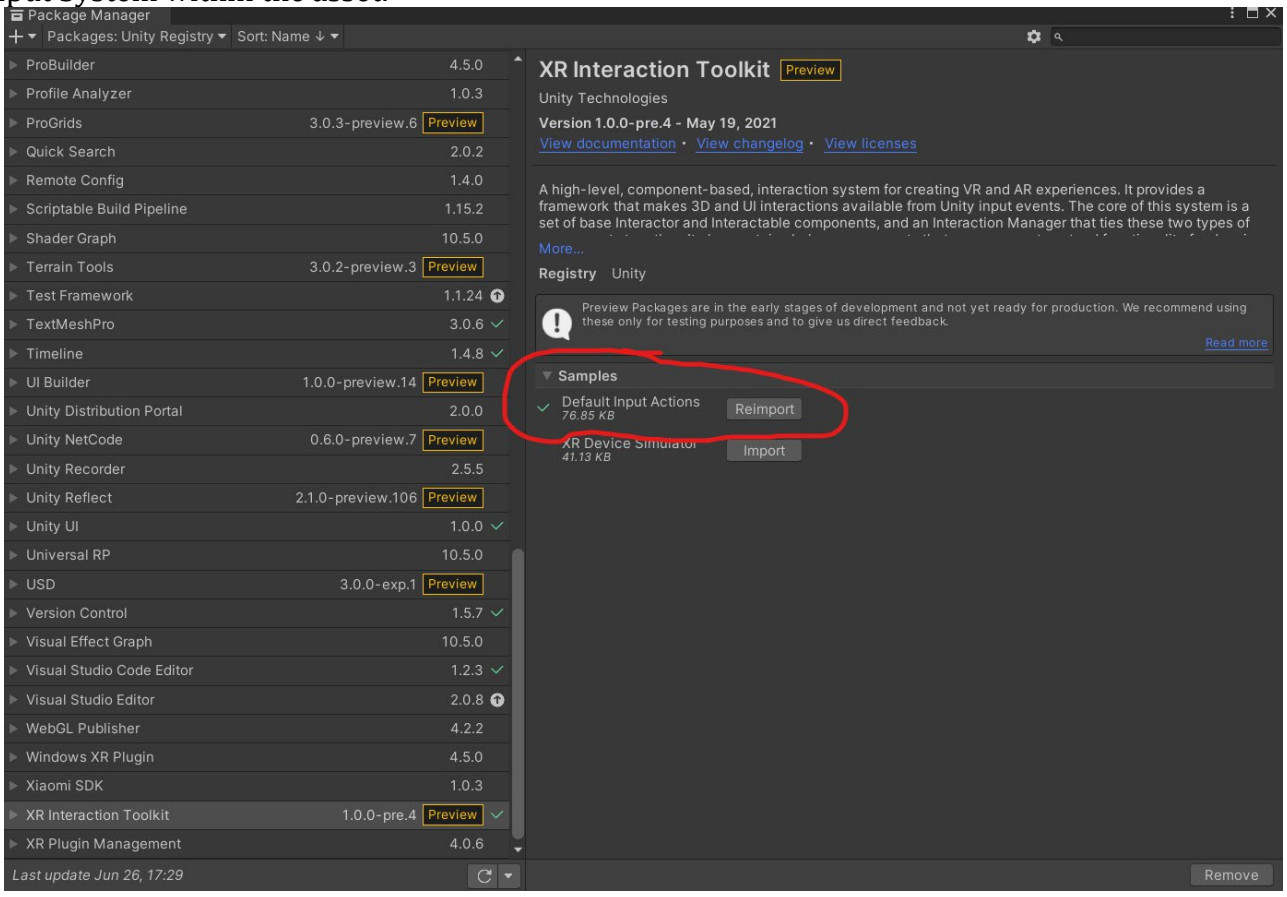
2. If not, we need to manually install XR Interaction Toolkit. Firstly, go to Edit > Project Settings > Project Settings > Package Manager. Then, check *Enable Preview Packages* toggle so that we can access the preview versions of the XR Interaction Toolkit.



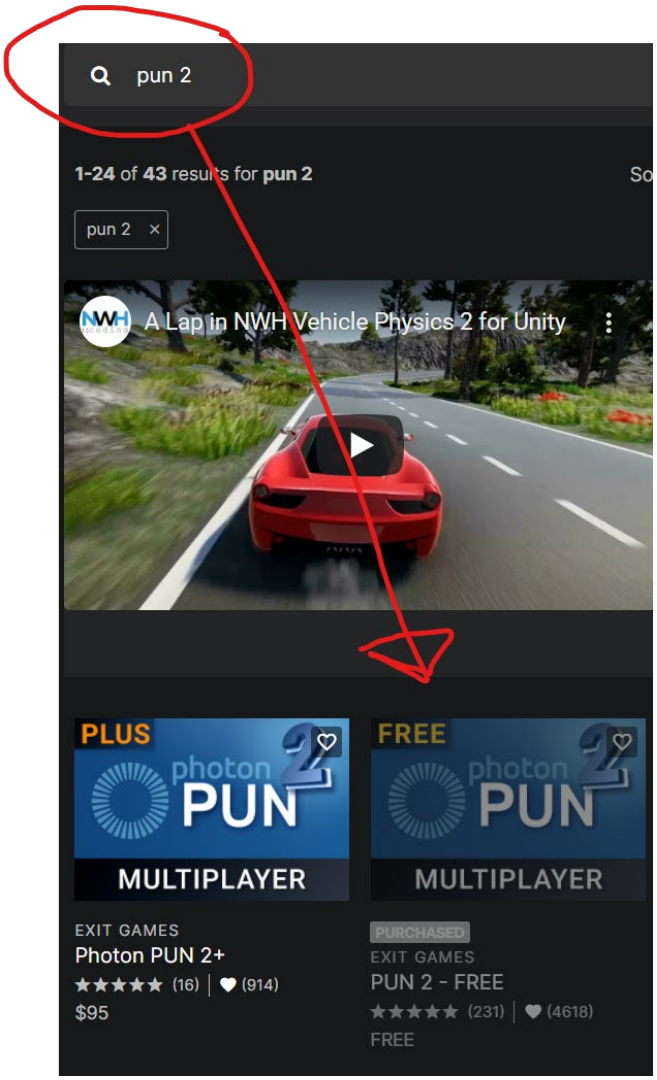
3. Then, under Unity Registry, locate and install **XR Interaction Toolkit** under All packages.



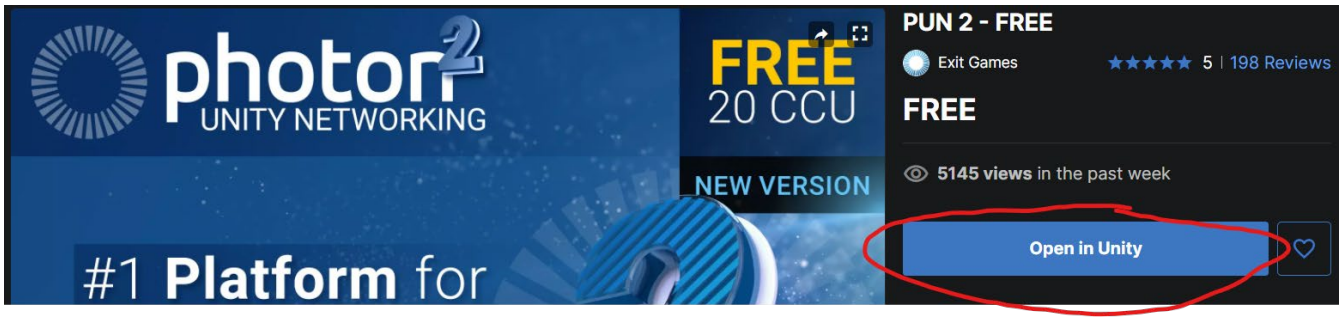
Also, import the Default Input Actions under Samples. This will help us setting up Unity's New Input System within the asset.



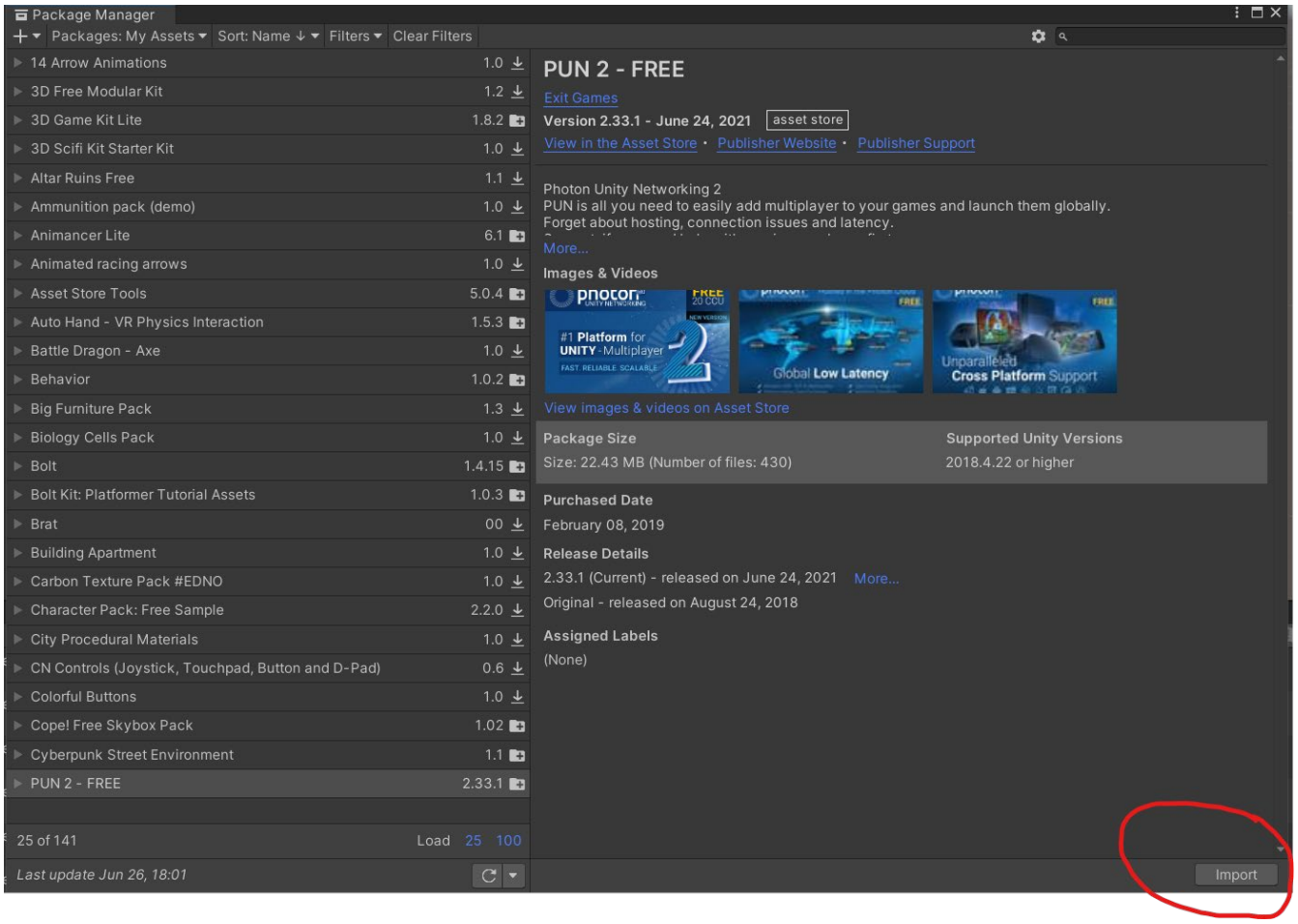
4. Now, we need to install the Multiplayer solution used in this asset, which is Photon PUN 2-Free asset. Open the Asset Store and search for PUN 2.

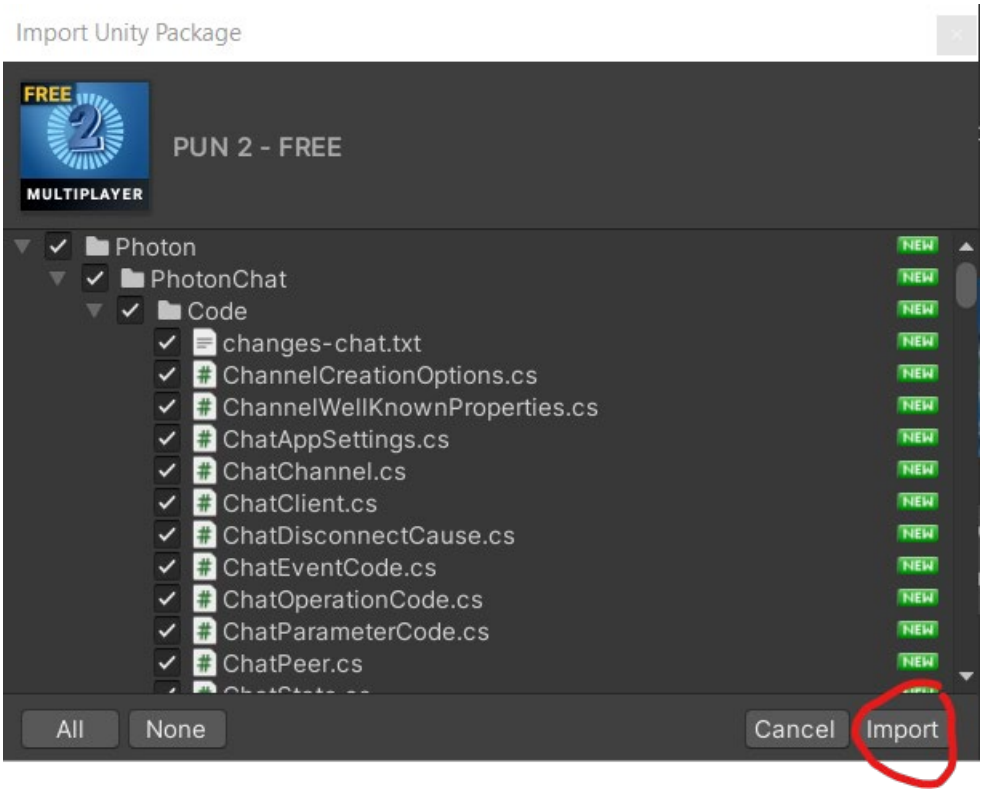


Click on Add to My Assets and then, click on Open in Unity.

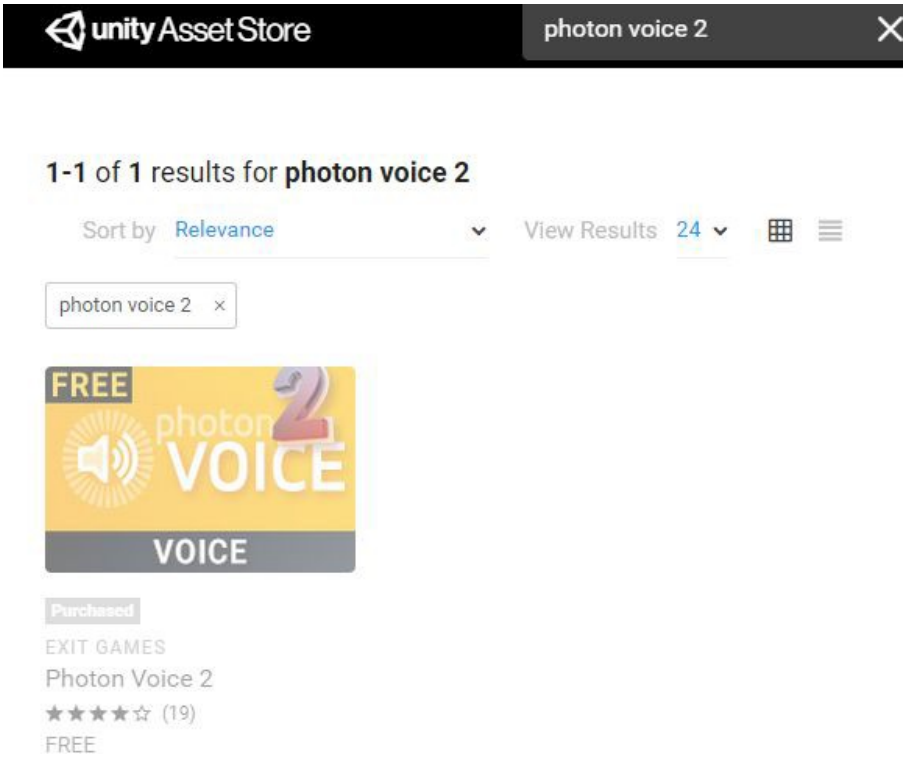


This will make Package Manager open the Asset page within Unity editor. Then, import it.

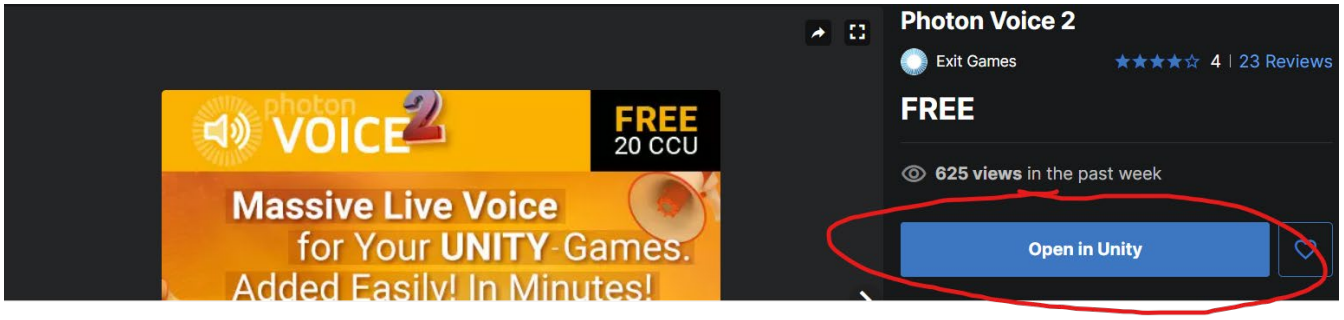




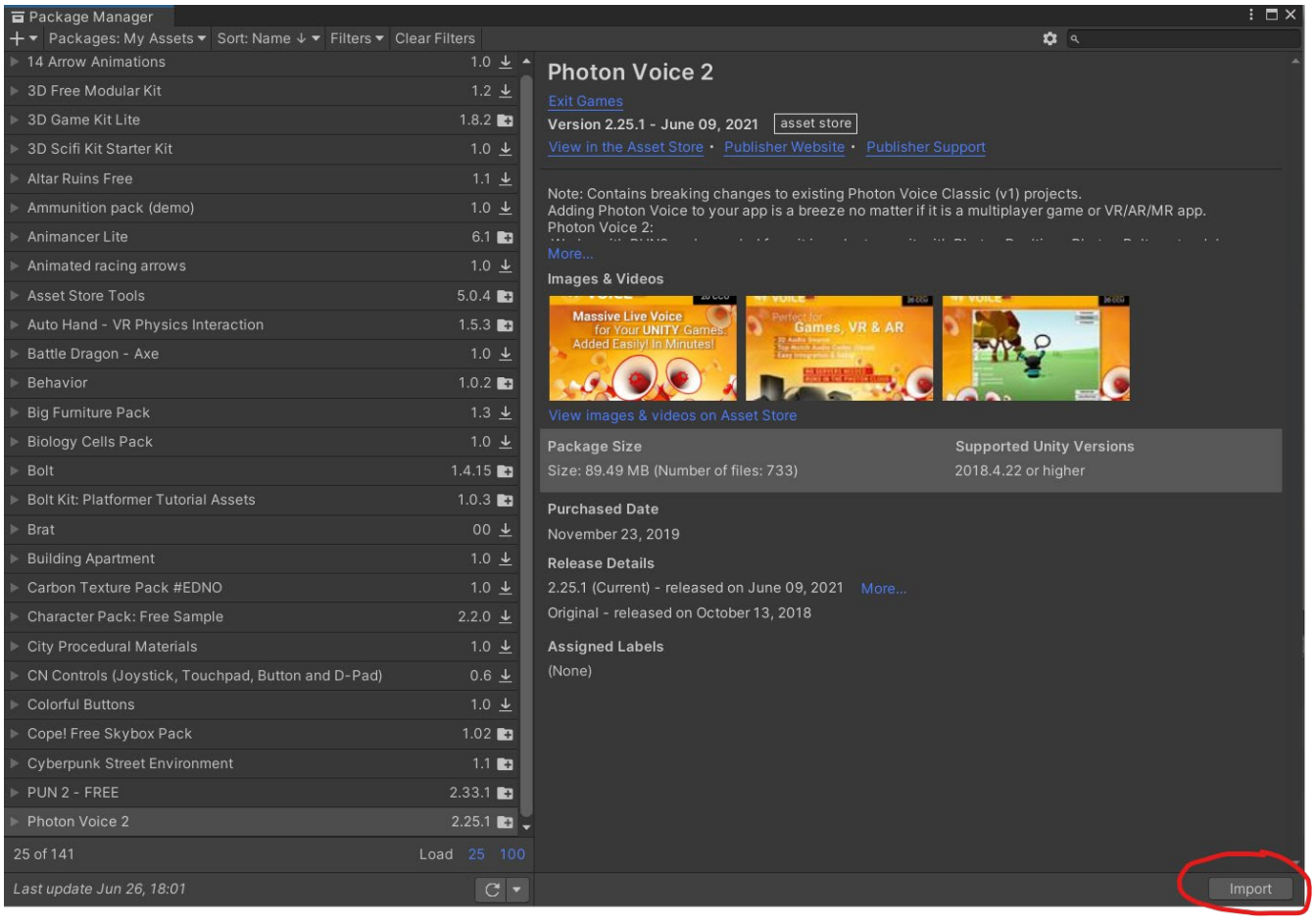
5. Next, search for **Photon Voice 2**. This is the asset we will use for Voice Chat feature.

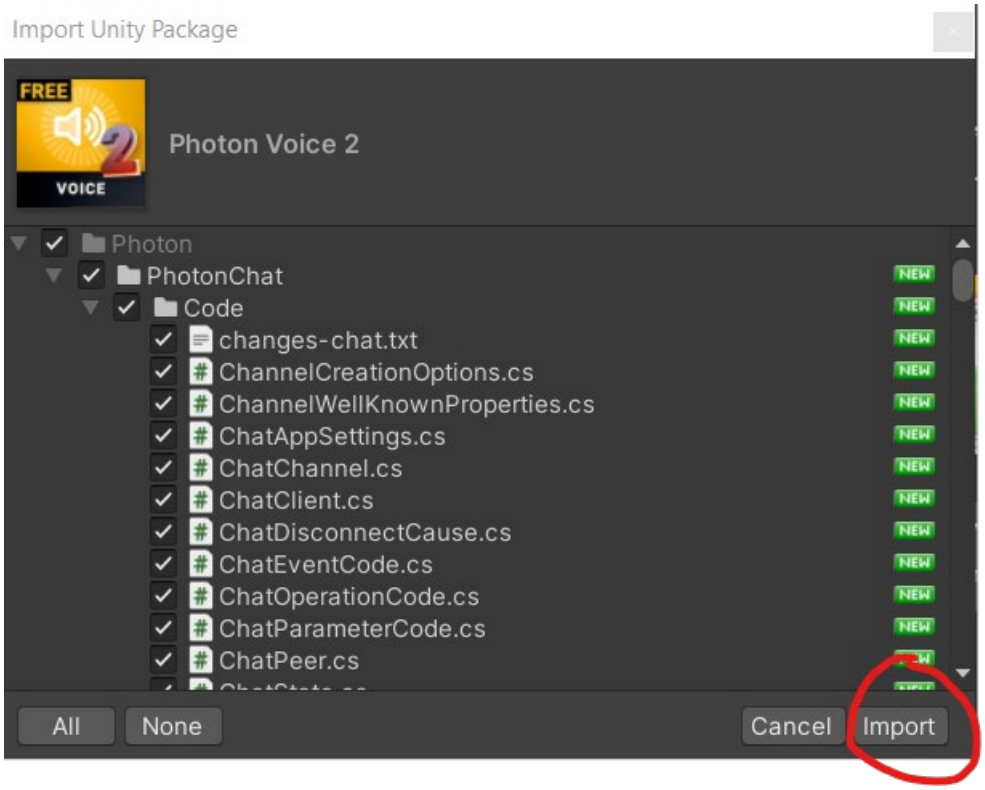


Click on Add to My Assets and then, click on Open in Unity.

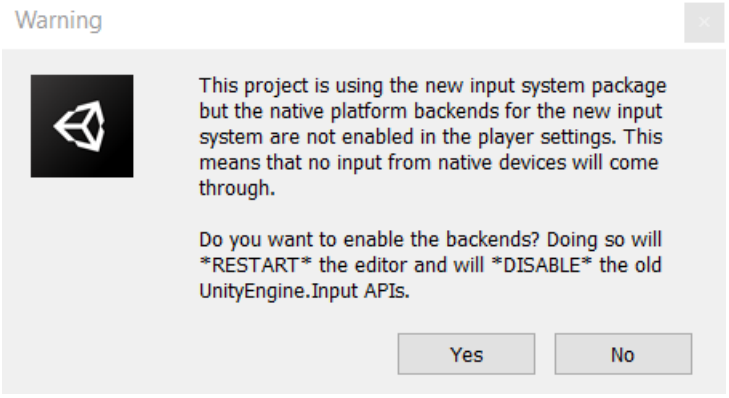


This will make Package Manager open the Asset page within Unity editor. Then, import it.



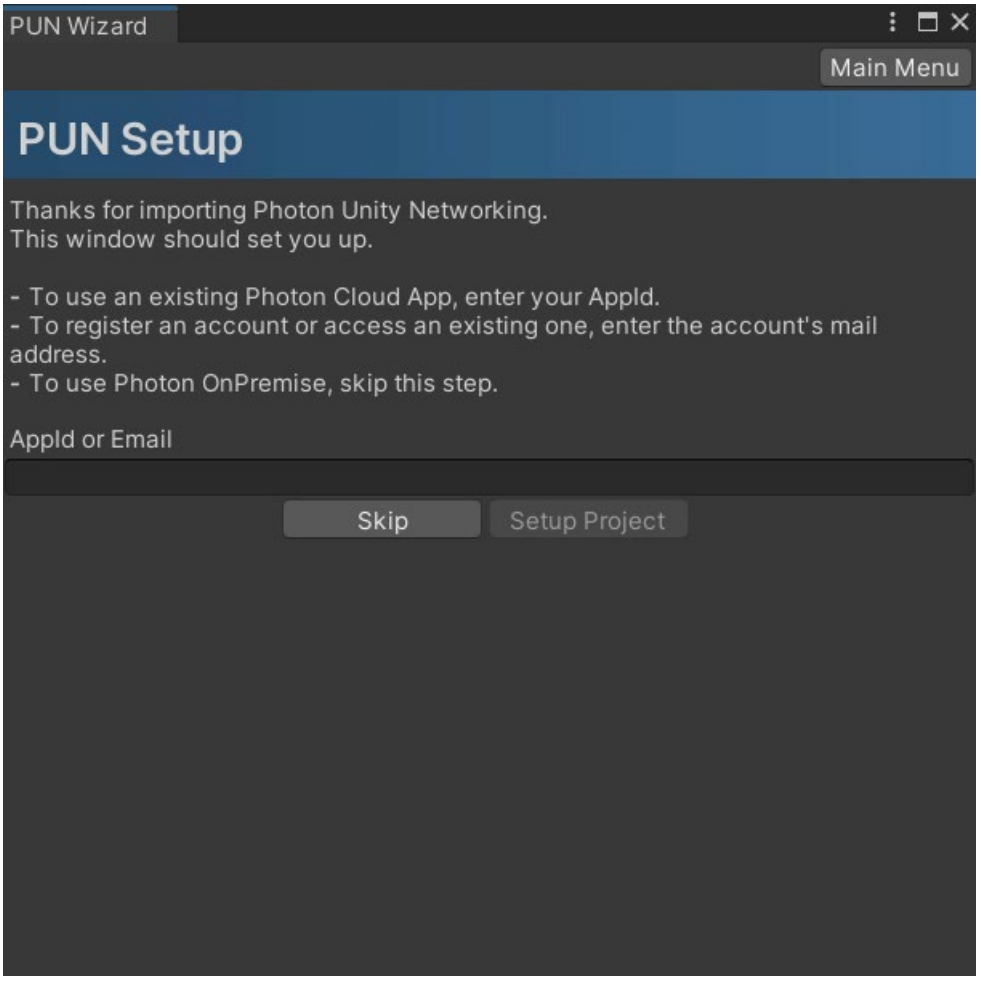


After doing that, you may have this warning:



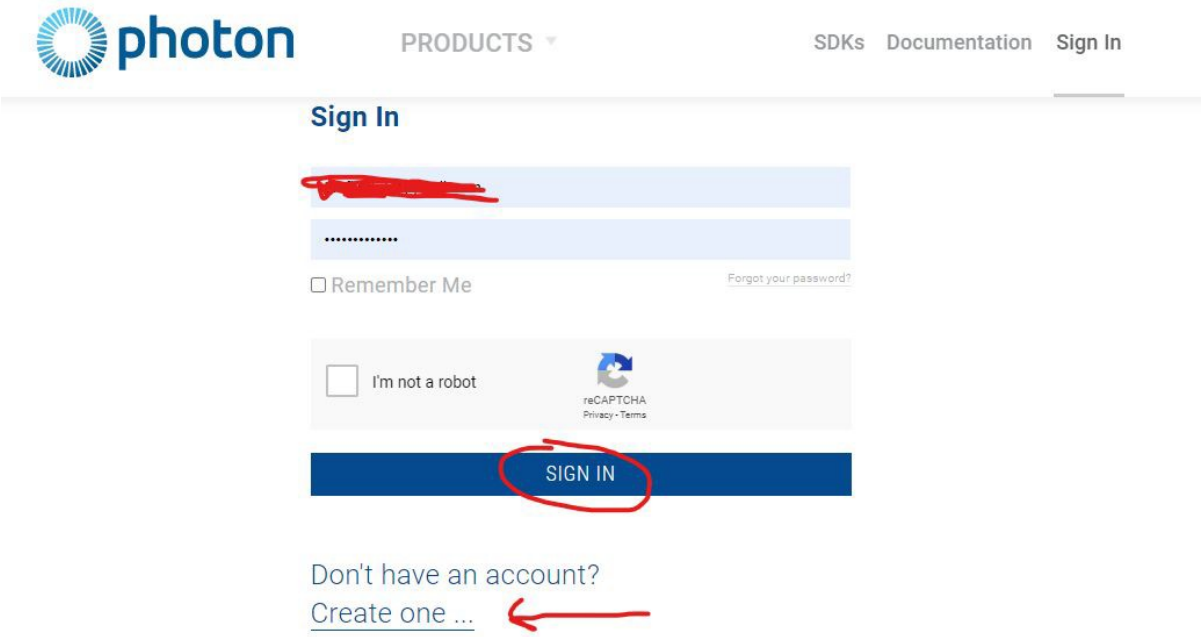
This is related to the new input system in Unity. Click on Yes to enable the new input system and it will Restart the project.

6. Next, all the errors in the console should be gone now and PUN Wizard should be visible.

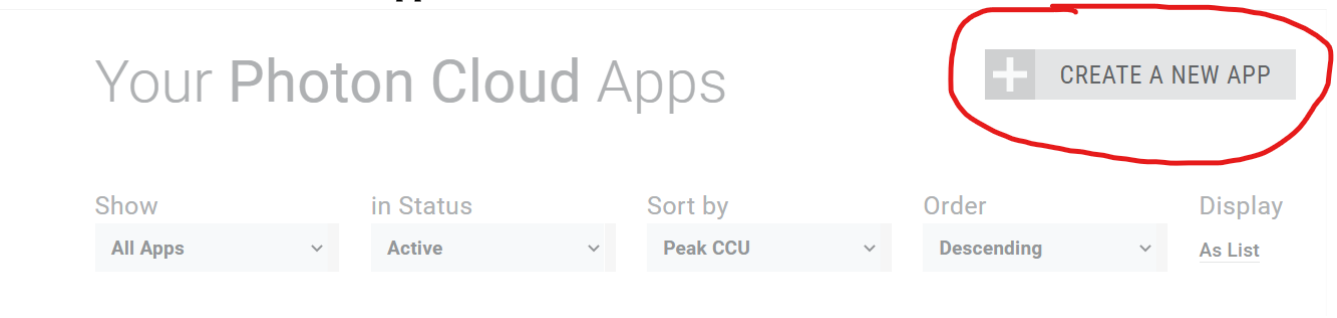


7. Now, we need to enter the appID from Photon cloud to set up Photon.

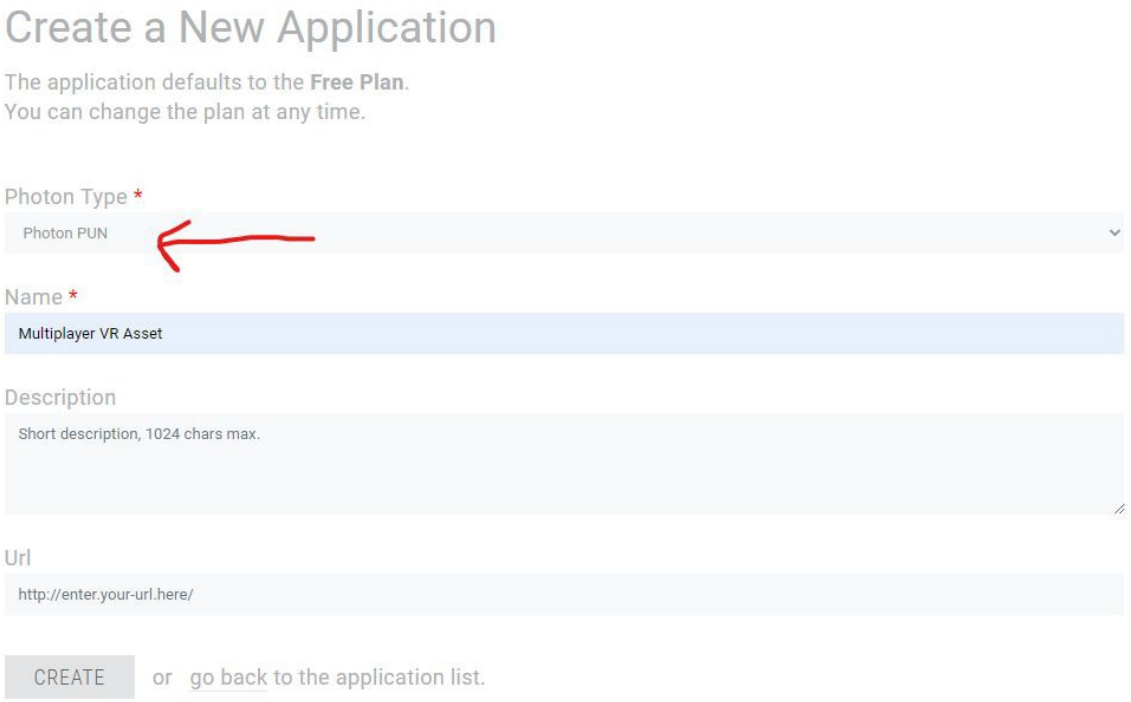
First, go to https://www.photonengine.com/ ,Then, Sign in **OR** Sign up if you don't have an account.



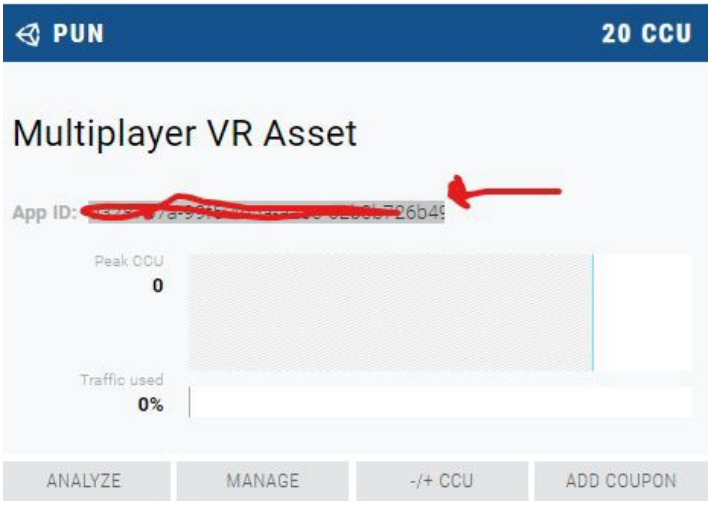
8. Click on ***Create a New App*** button.



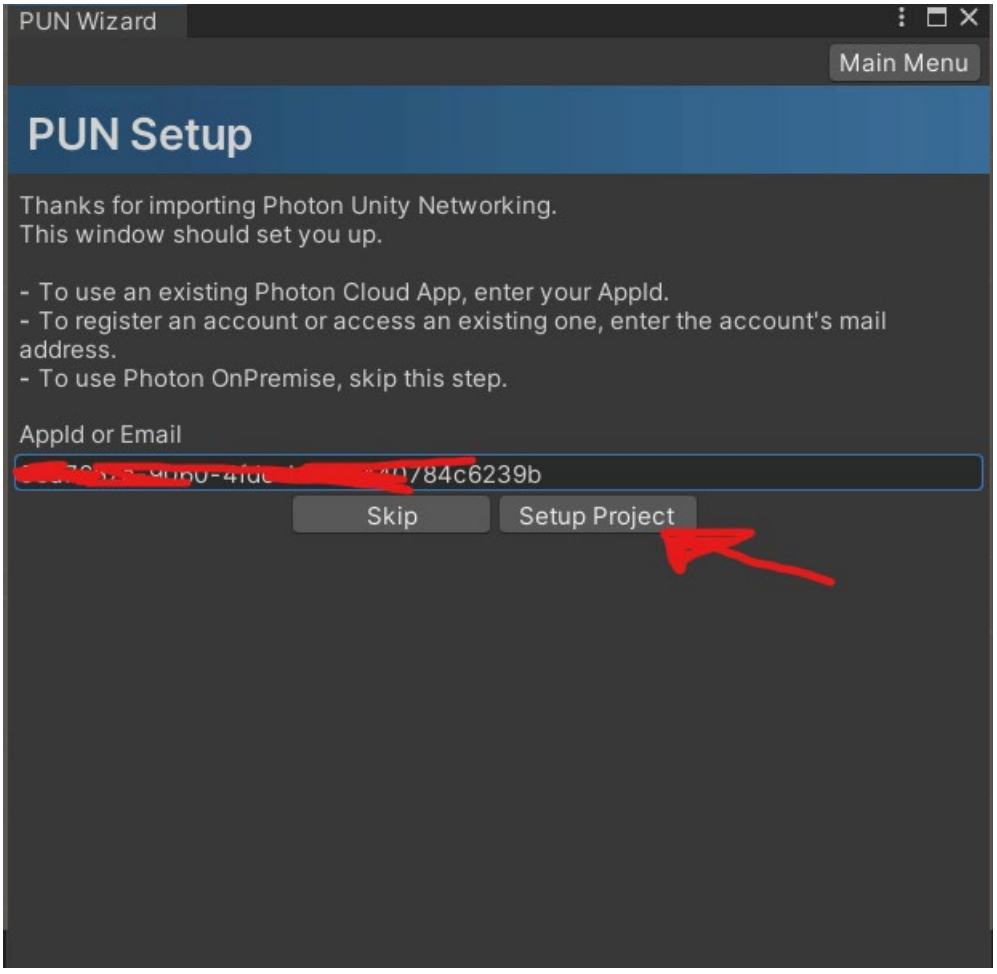
9. Create a new cloud application with Photon Pun type.



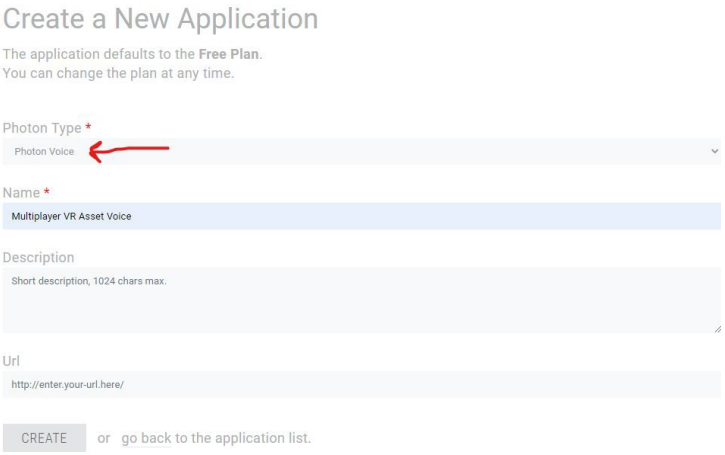
10. Copy its appID.



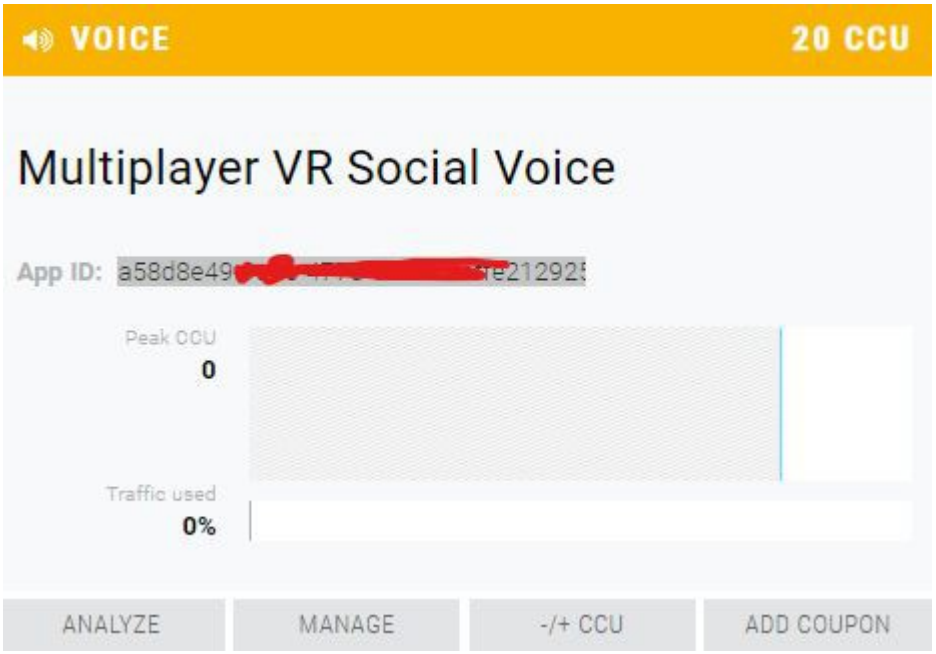
11. Then, paste it here and click on Setup.



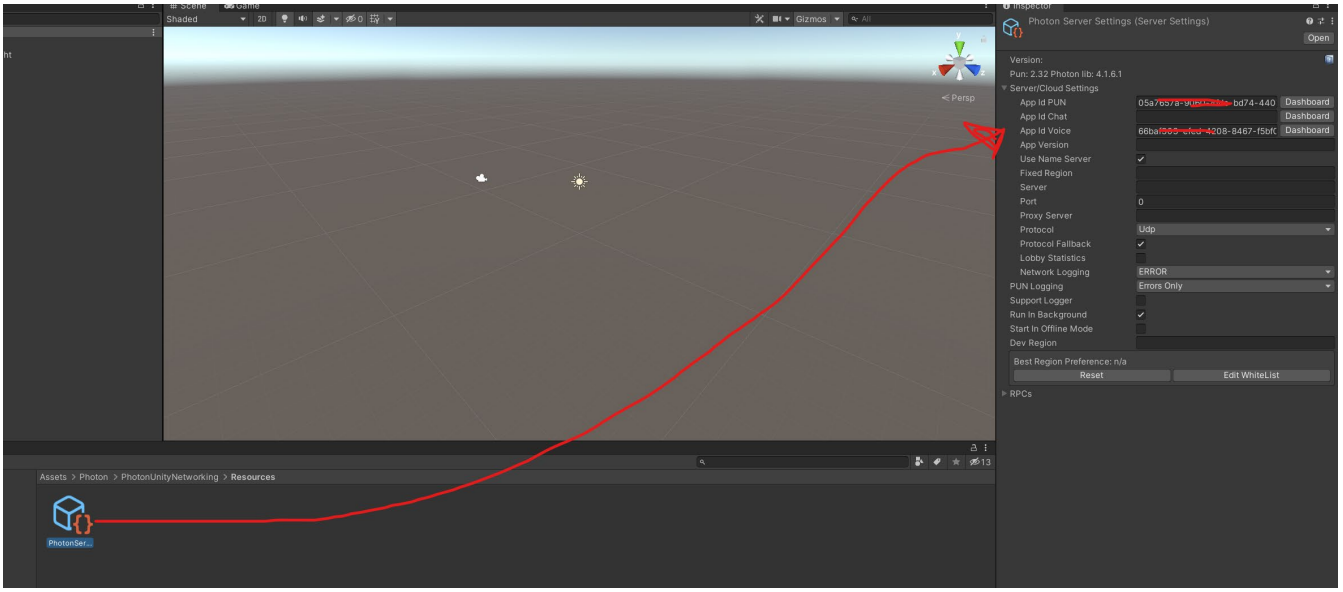
12. Go back to the Photon website and create another cloud application with Photon Voice type.



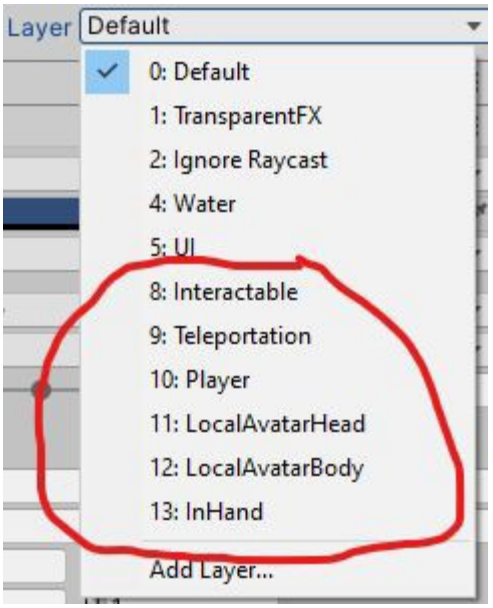
13. Copy its app Id.



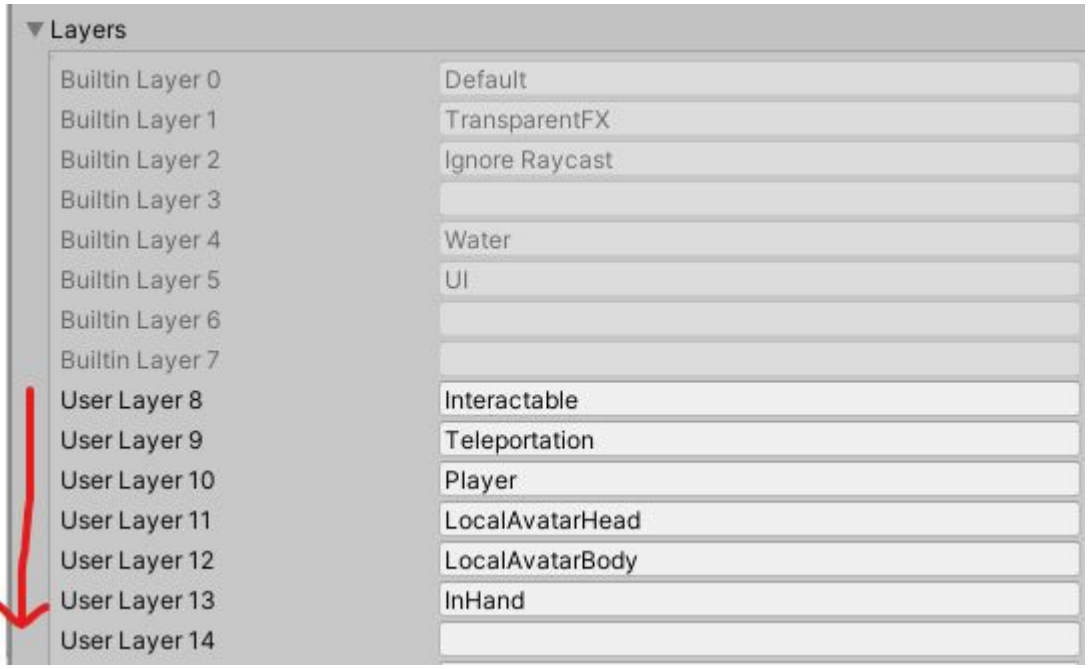
14. Back to Unity, go to Assets> Photon > Photon Unity Networking > Resources and open Photon Server Settings. And paste the app Id to App Id Voice =>



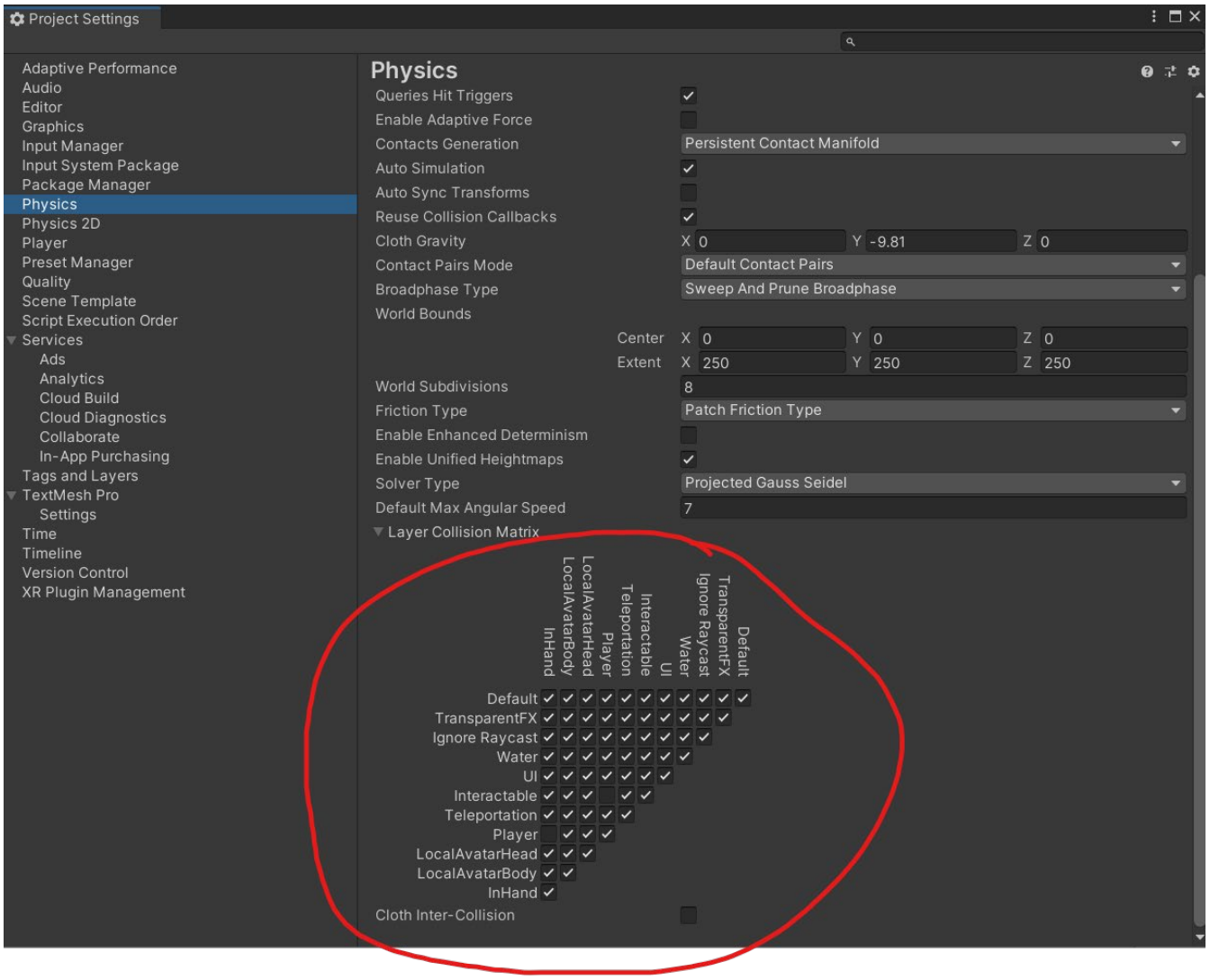
15. Next, add the following Layers to the project.



Note that the order is important.



16. Lastly, open Edit > Project Settings > Physics. Disable the collision between the Player/InHand, and Player/Interactable layer.



That is it! Congrats!

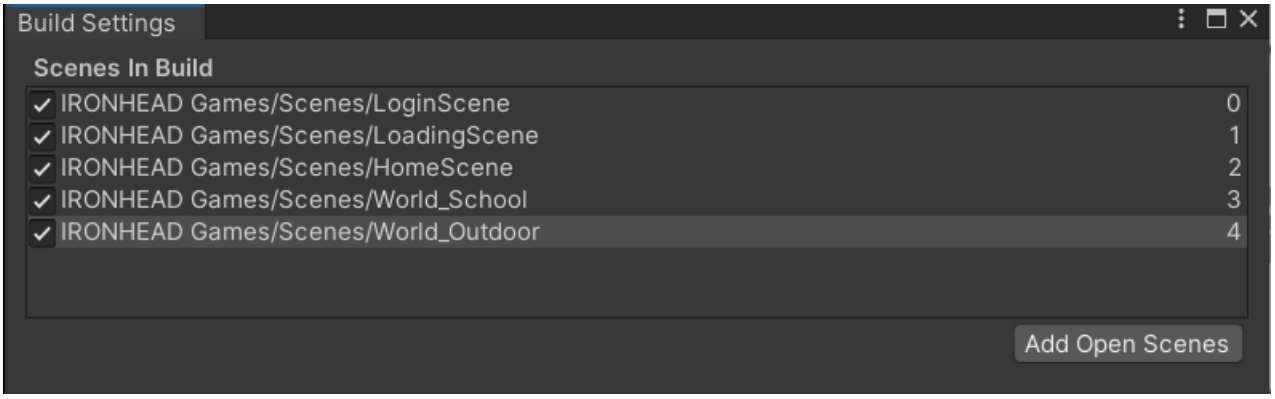
The next step is to configure the project for Oculus Quest and Quest 2. Let me remind you that this project is tested only with Oculus Quest and Quest 2. However, Unity XR Interaction Toolkit also supports SteamVR headsets via the OpenXR. So, keep reading and I will show you how to setup the project for OpenXR to support multiple VR headsets at the same time.

Configuring the Project for Oculus Quest/2 Build

Now, we will prepare the project for Oculus Quest and Quest 2 build. And we will mainly follow the settings from here:

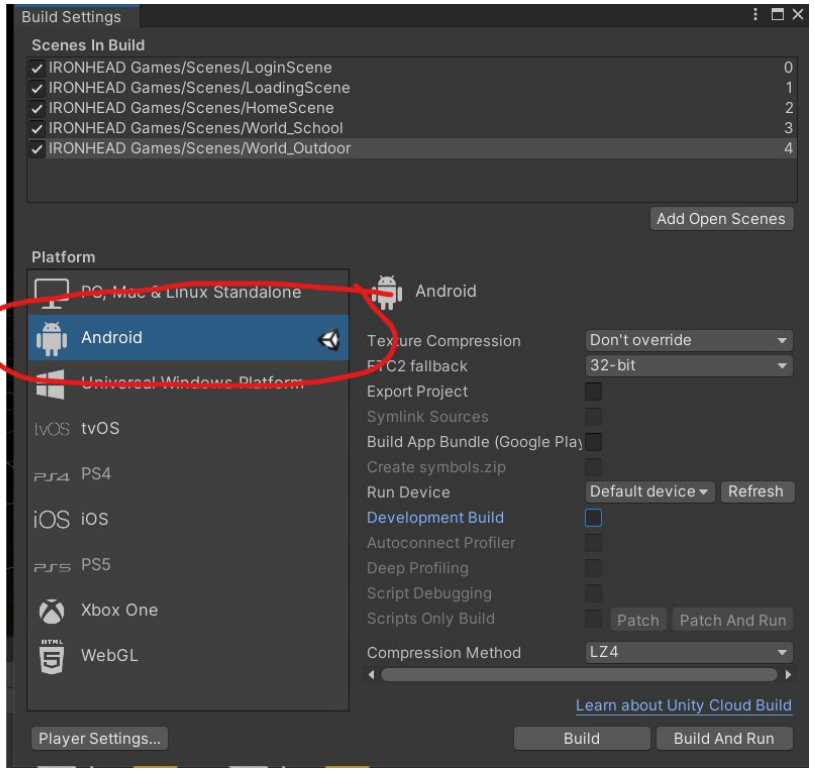
https://developer.oculus.com/documentation/unity/unity-conf-settings/

1. First, open File> Build Settings. Add the following scenes to Scenes in Build from IRONHEAD Games > Scenes folder.

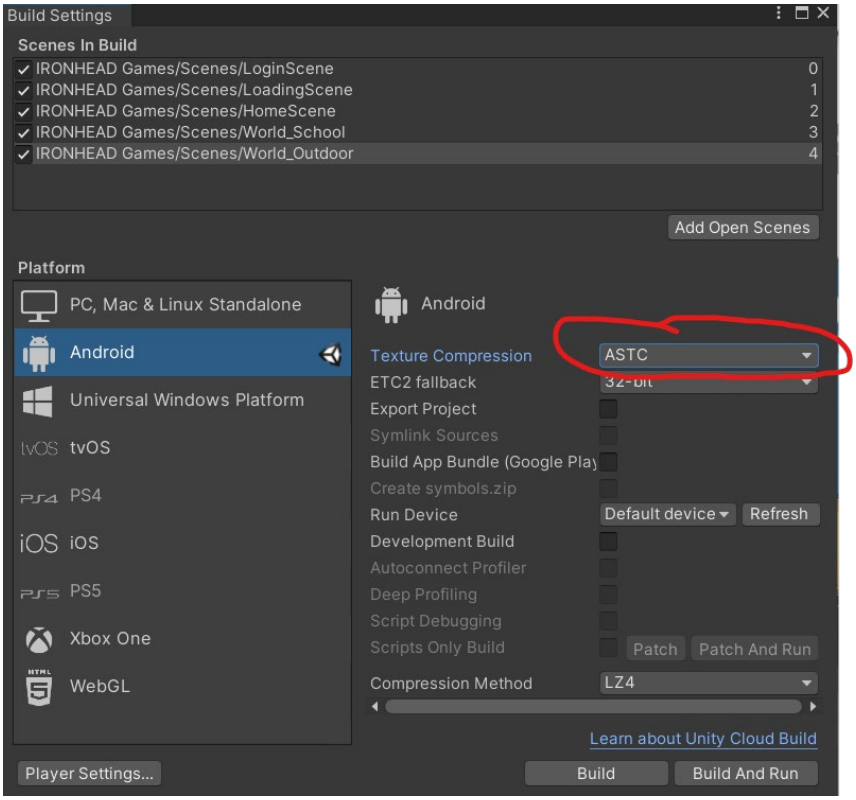


Make sure to follow the order such that the LoginScene must be the first scene. These scenes will be explained in detail in the following section.

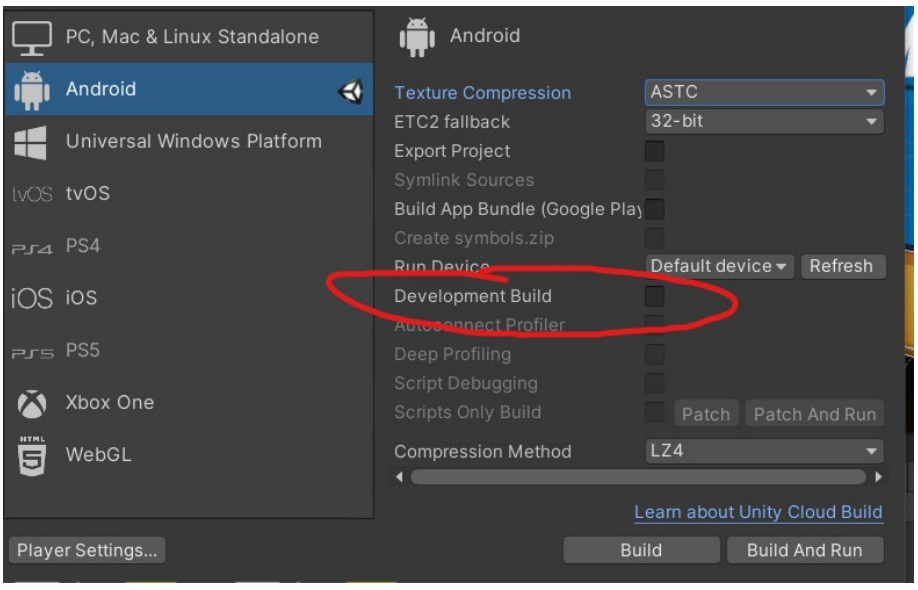
2. Next, switch the platform to Android.
We do this since Oculus Quest/2 is actually an Android device with lenses and VR controllers. So, the output should be apk files.



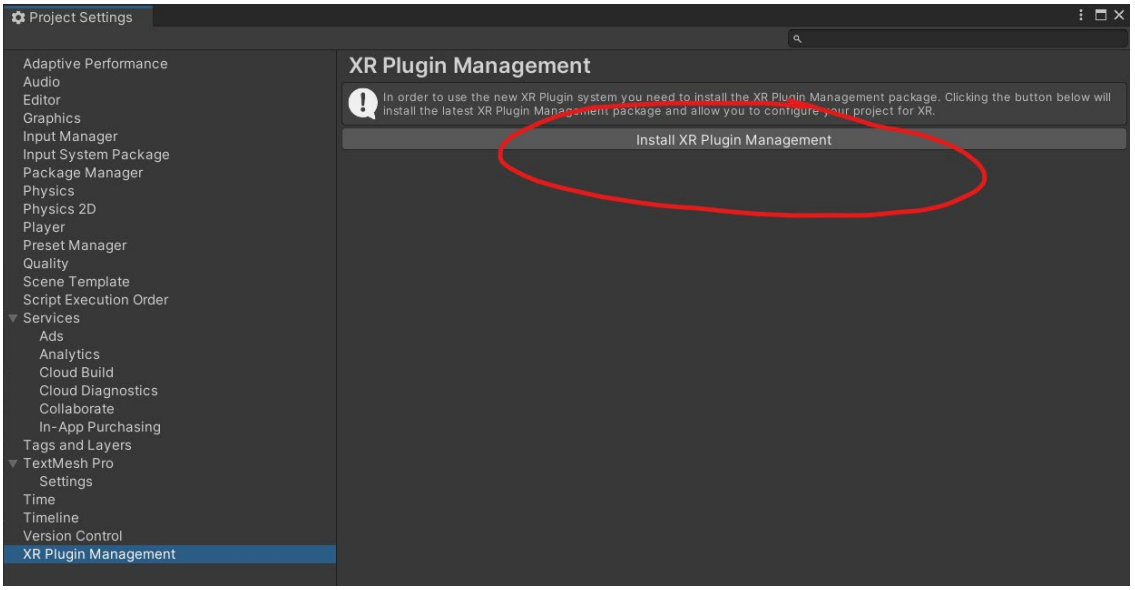
3. Change the Texture Compression to ASTC.



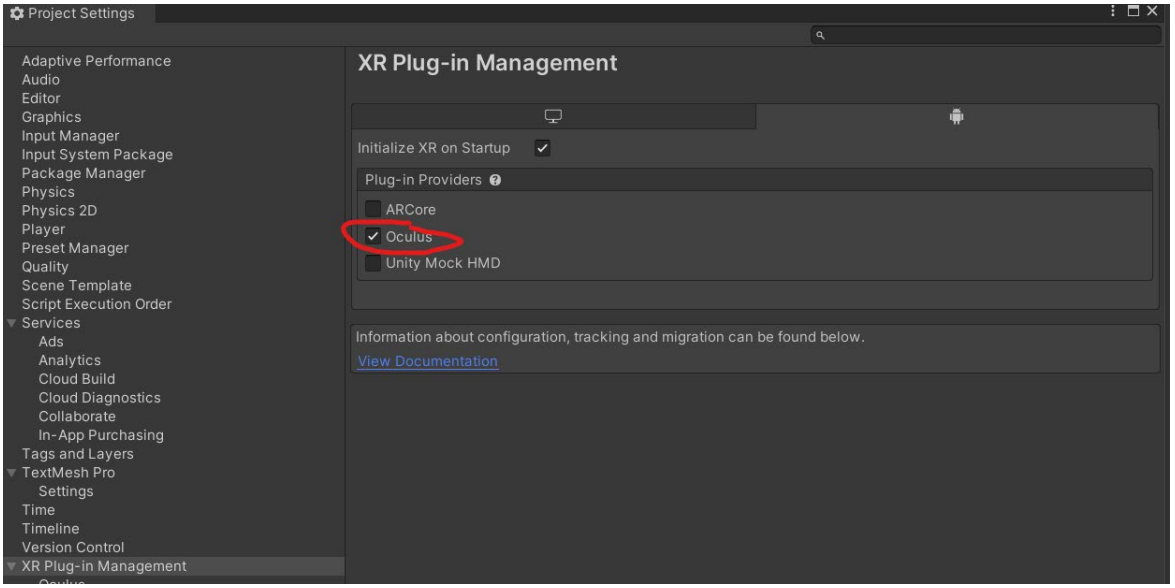
Also, clear the Development Build selection since it may impact the performance.



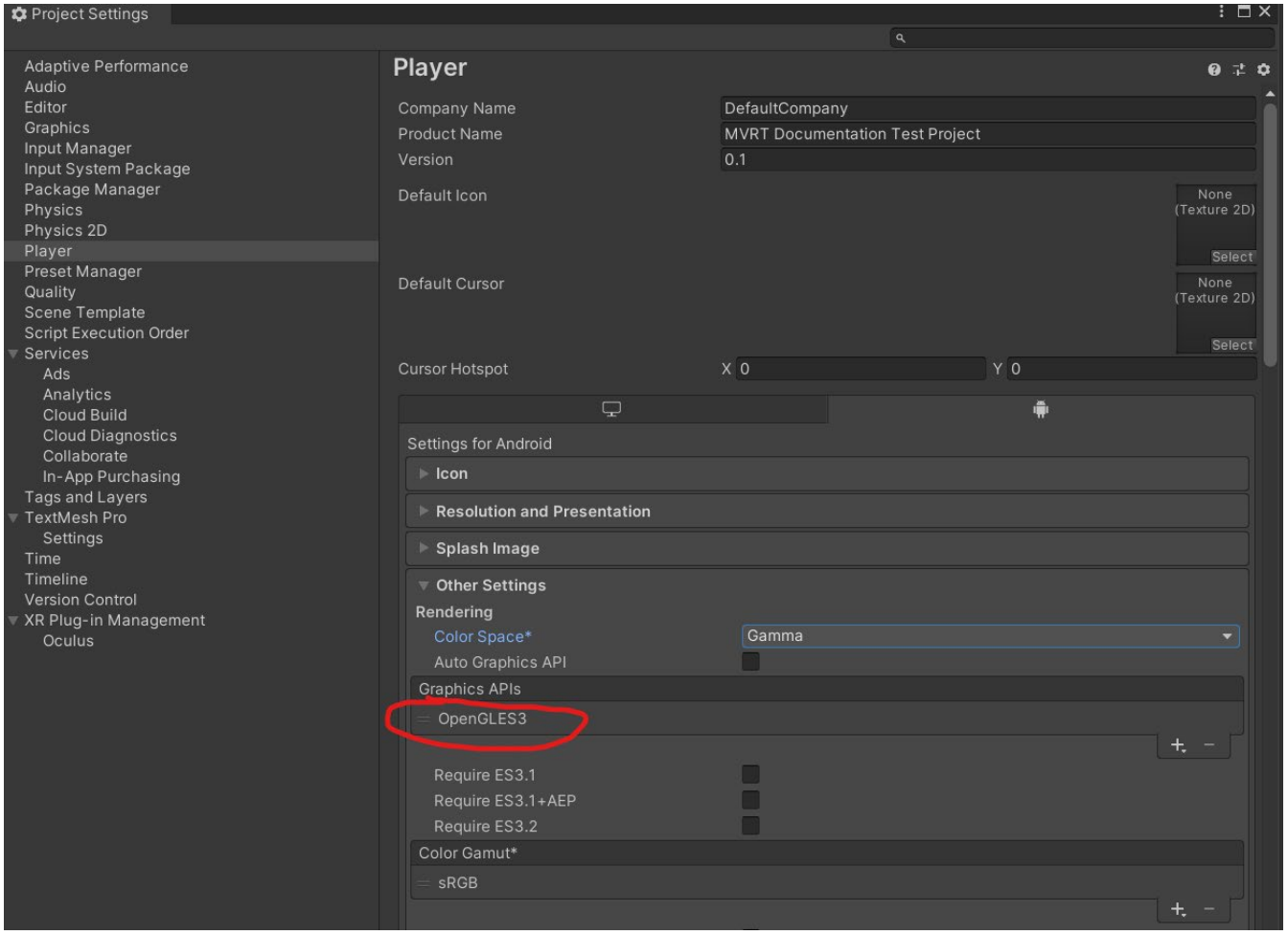
4. Then, open Edit > Project Settings. Click on XR Plugin Management. And install it.



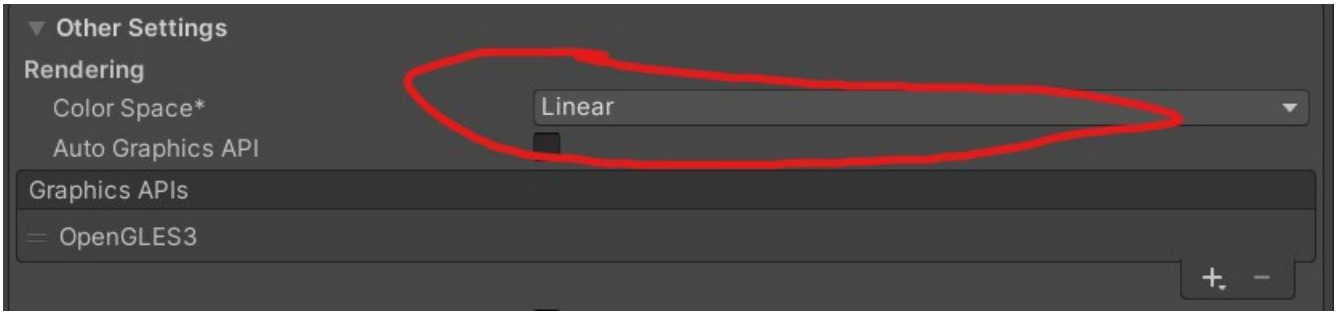
5. Under Android tab and under Plug-in Providers, check Oculus toggle. It could take a while to install the Oculus XR plugin.



6. Open the Player Settings. Under Other Settings, make sure to have only OpenGLES3 for the Graphics API. You may even need to remote Vulkan Graphics API since it is still experimentally supportedfor Oculus Quest.

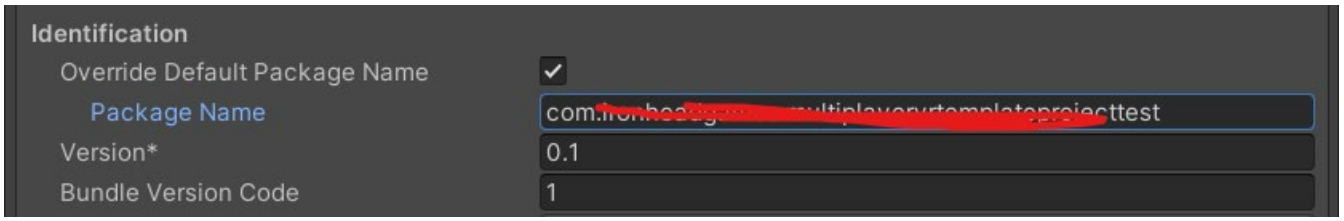


7. Then, set the Color Space as **Linear**. It may take significant amount of time.

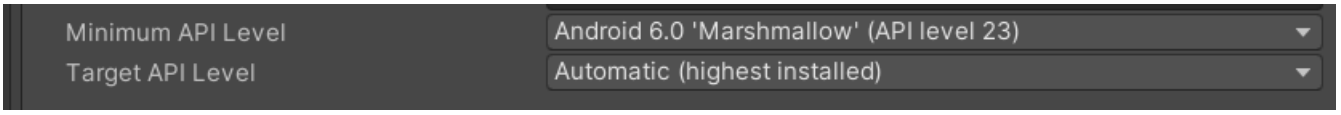


As stated before, these settings are suggested from Oculus. See more info: https://developer.oculus.com/documentation/unity/unity-conf-settings/

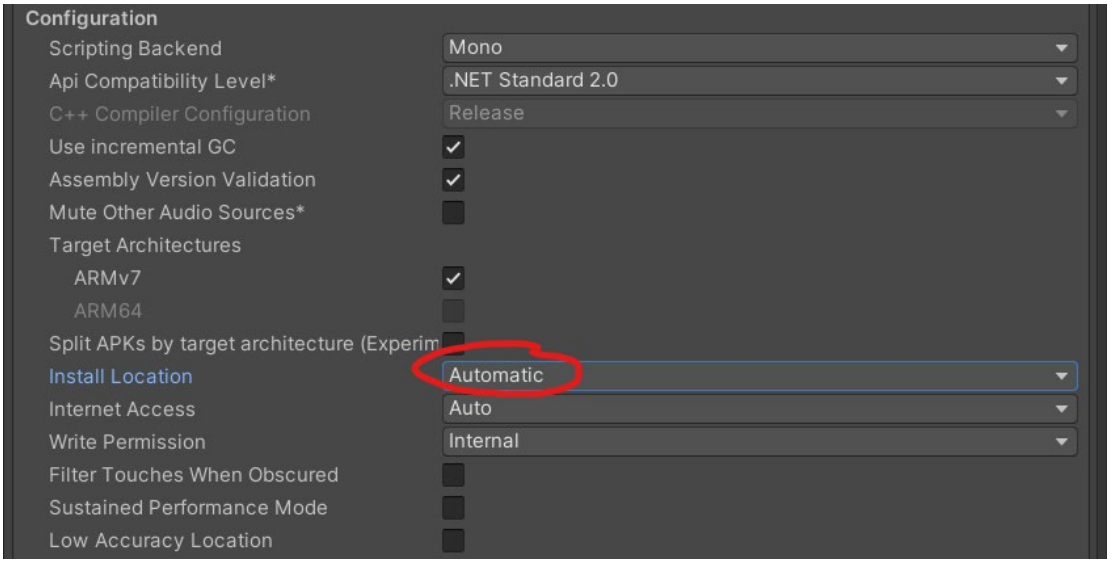
8. Under Identification, check *Override Default Package Name* and enter your unique package name.



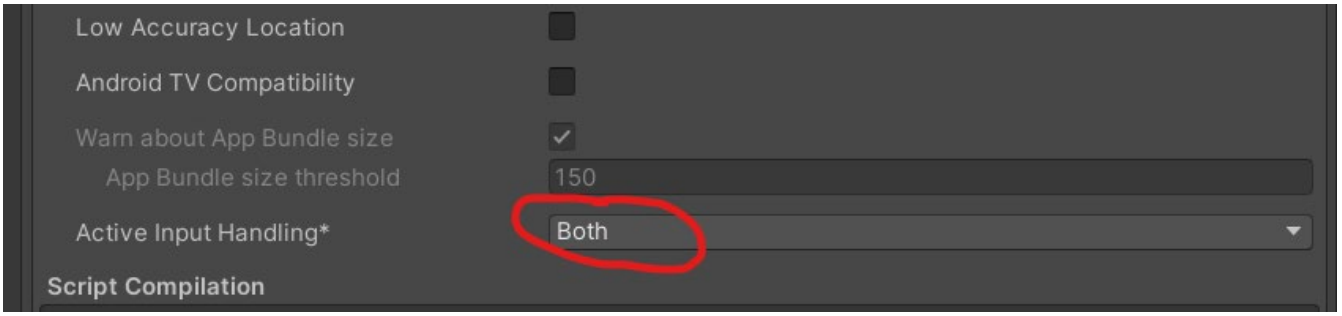
9. Set the minimum API level as 23 and Target API Level as Automatic (highest installed).



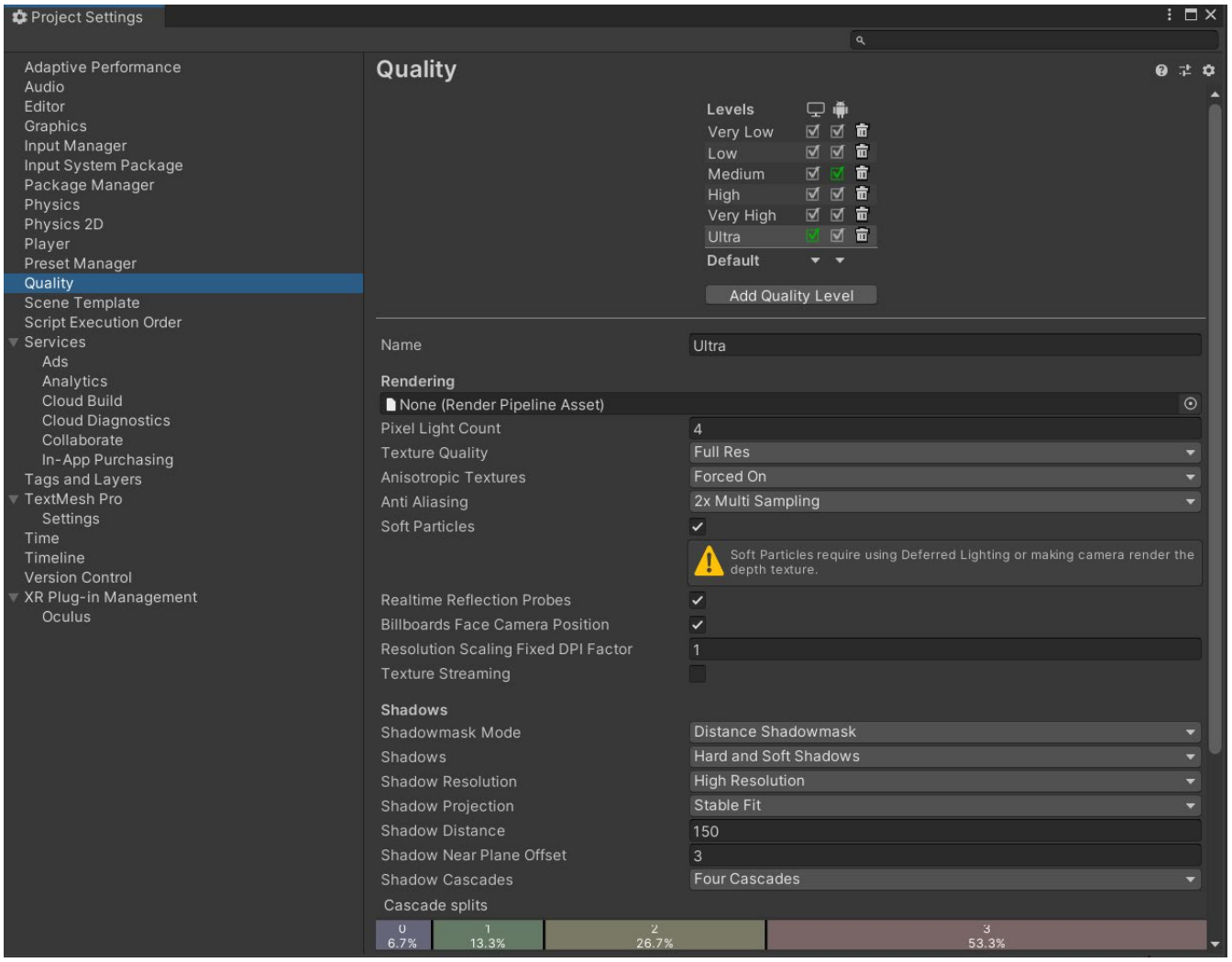
10. Select Install Location as Automatic.



11. Set the Active Input Handling to Both. Because some assets did not switch to the new input system yet and this setting will help prevent issues with the assets using the old input system.

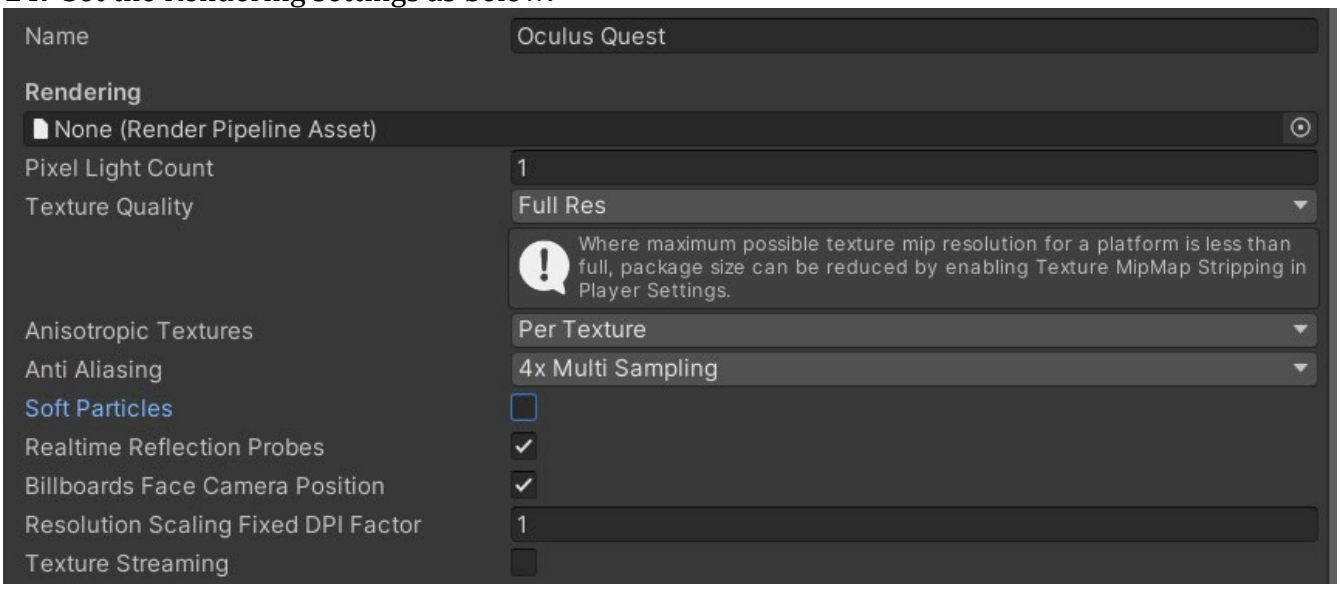


12. Next, under Project Settings, select Quality tab.

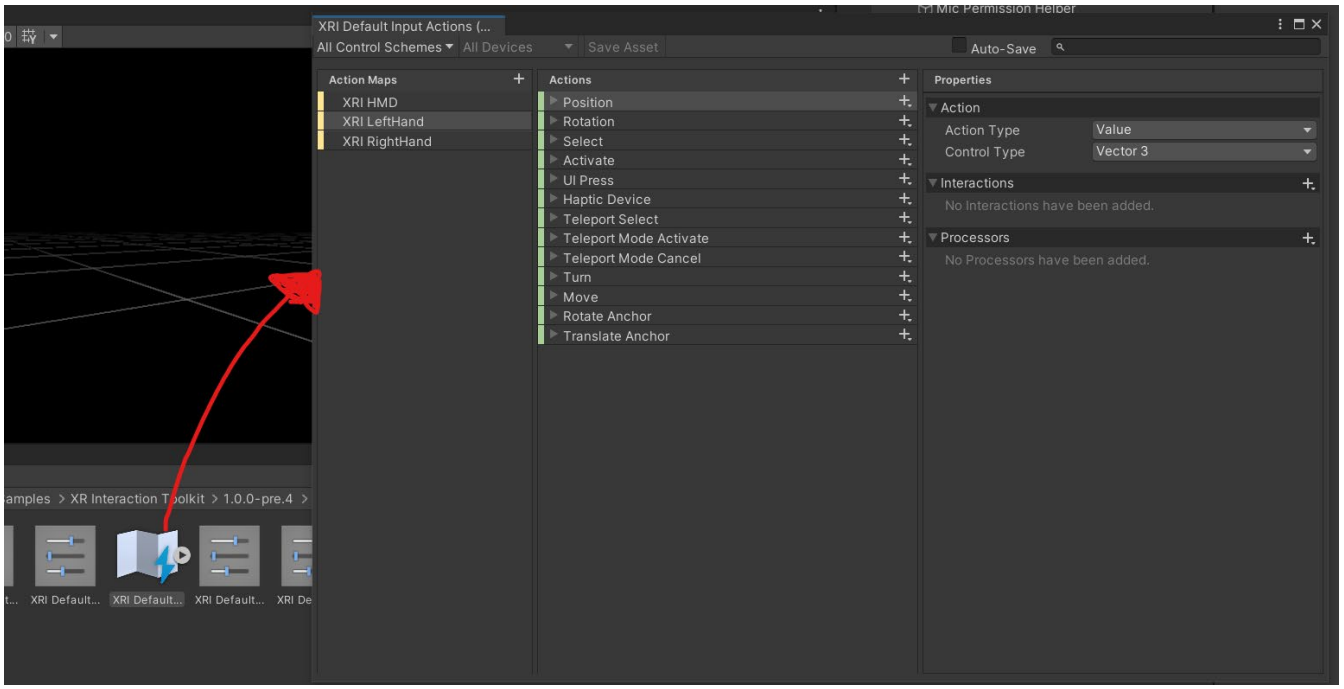


13. Uncheck all the settings and add a new Quality Level called Oculus Quest. Then, enable it for Android and check Ultra for the PC platform.

14. Set the Rendering settings as below.



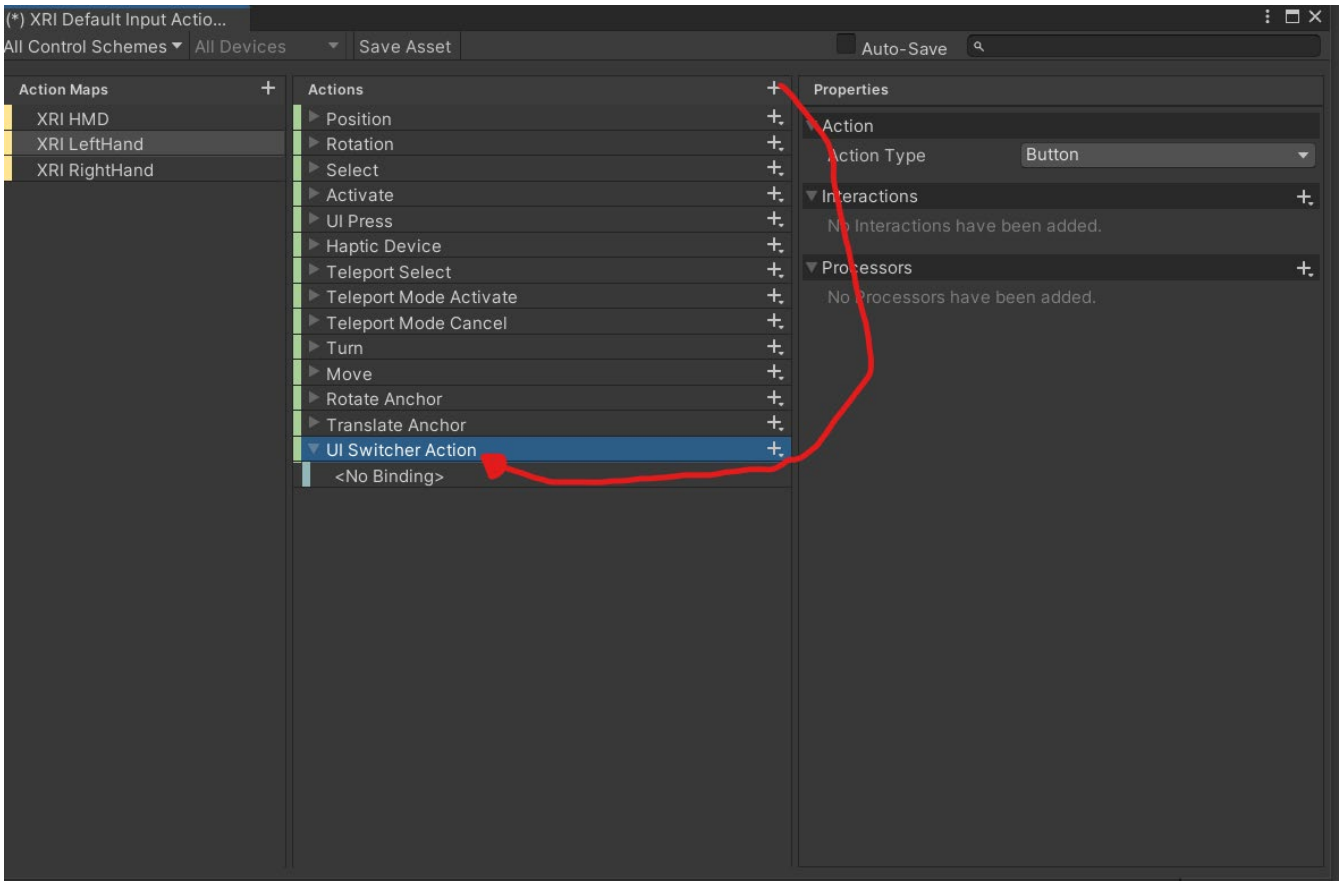
15. Now, we need to setup the project for the new input system.
With the new input system, we need to have Input Actions Asset that contains the general input data. We already have this asset under **Samples > XR Interaction Toolkit > 1.0.0-pre.4 > Default Input Actions.**



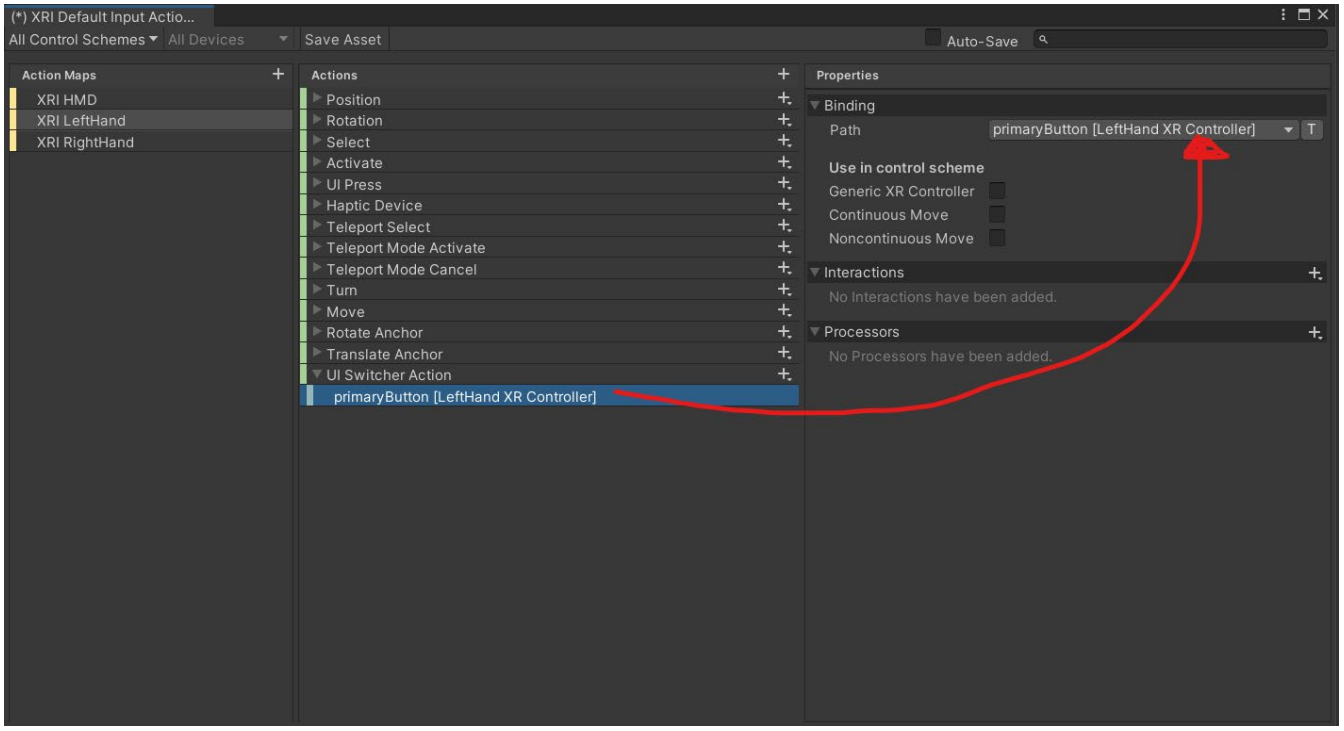
This Input Actions Asset is provided from XR Interaction Toolkit and helps us use the new input system with XR projects. But we need to add a new input action and adjust project settings to be able to setup this asset with the new input system. More info about the new input system can be found in this official documentation:
https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/QuickStartGuide.html

Learning the new input system can really help you further customizing this asset.

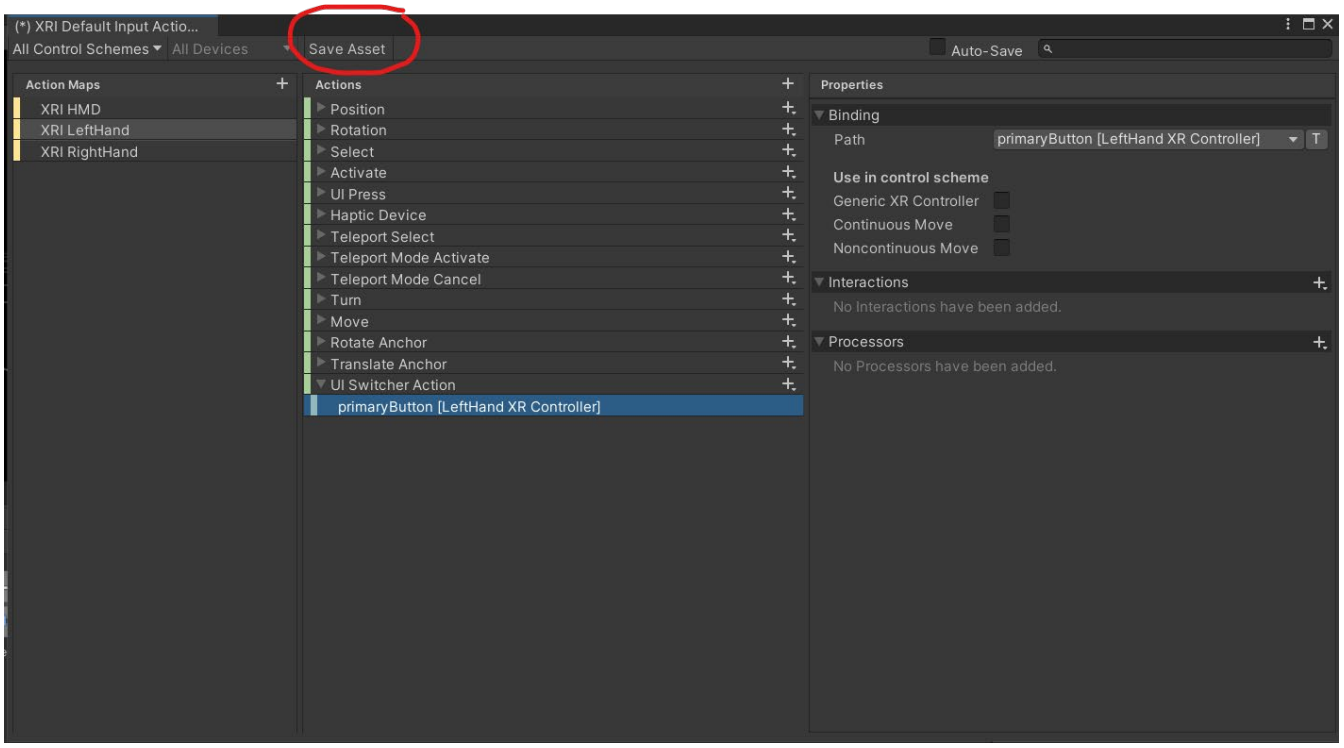
Firstly, click on XRI LeftHand Action Map, and add a new Action called UI Switcher Action:



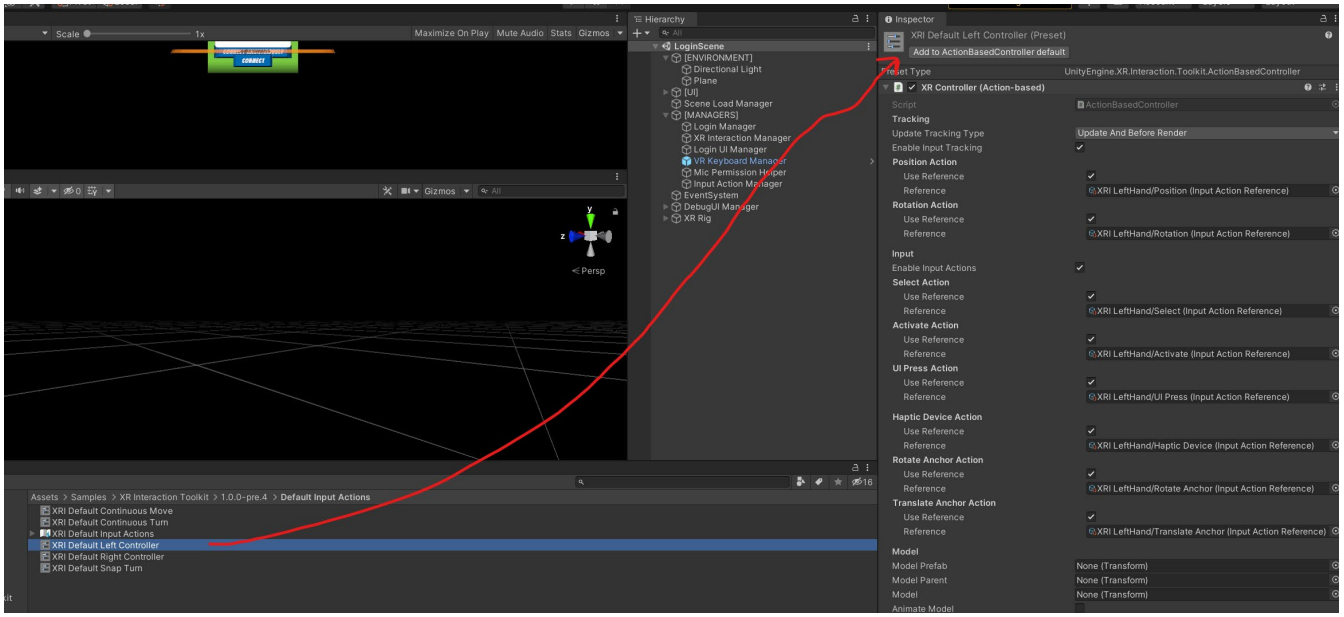
Then, add a binding and set its Path as *primaryButton [LeftHand XR Controller]:*

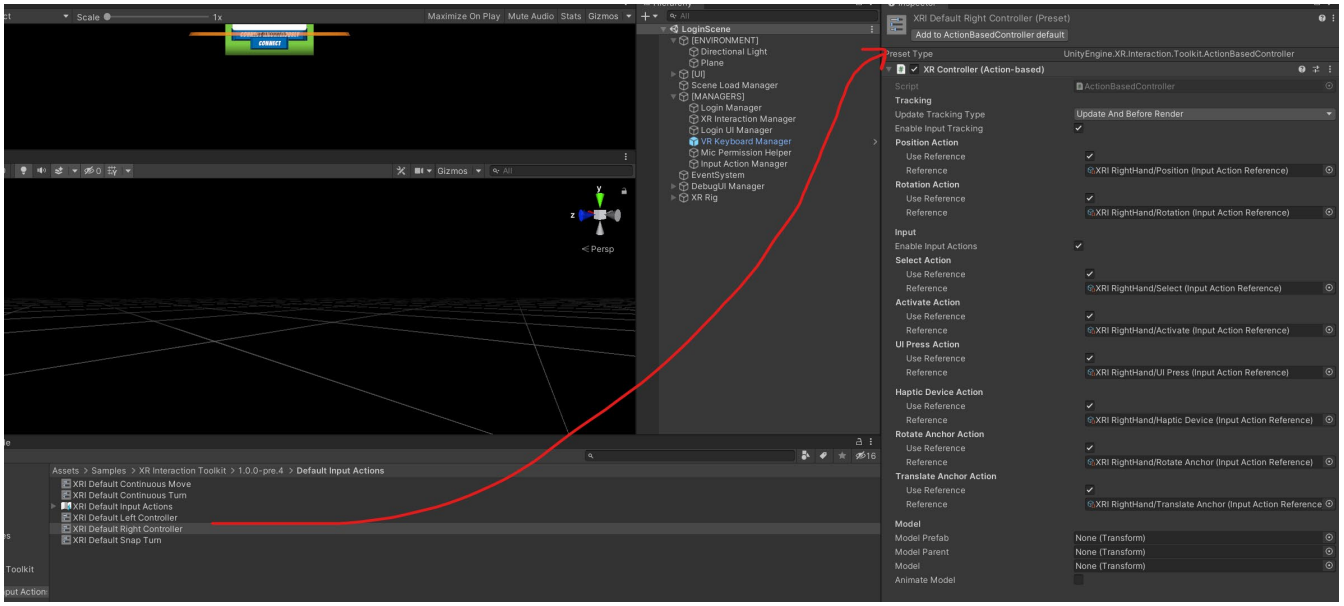


You can find this Path by selecting XR Controller > Optional Controls. Then, click on Save Asset button and close it.



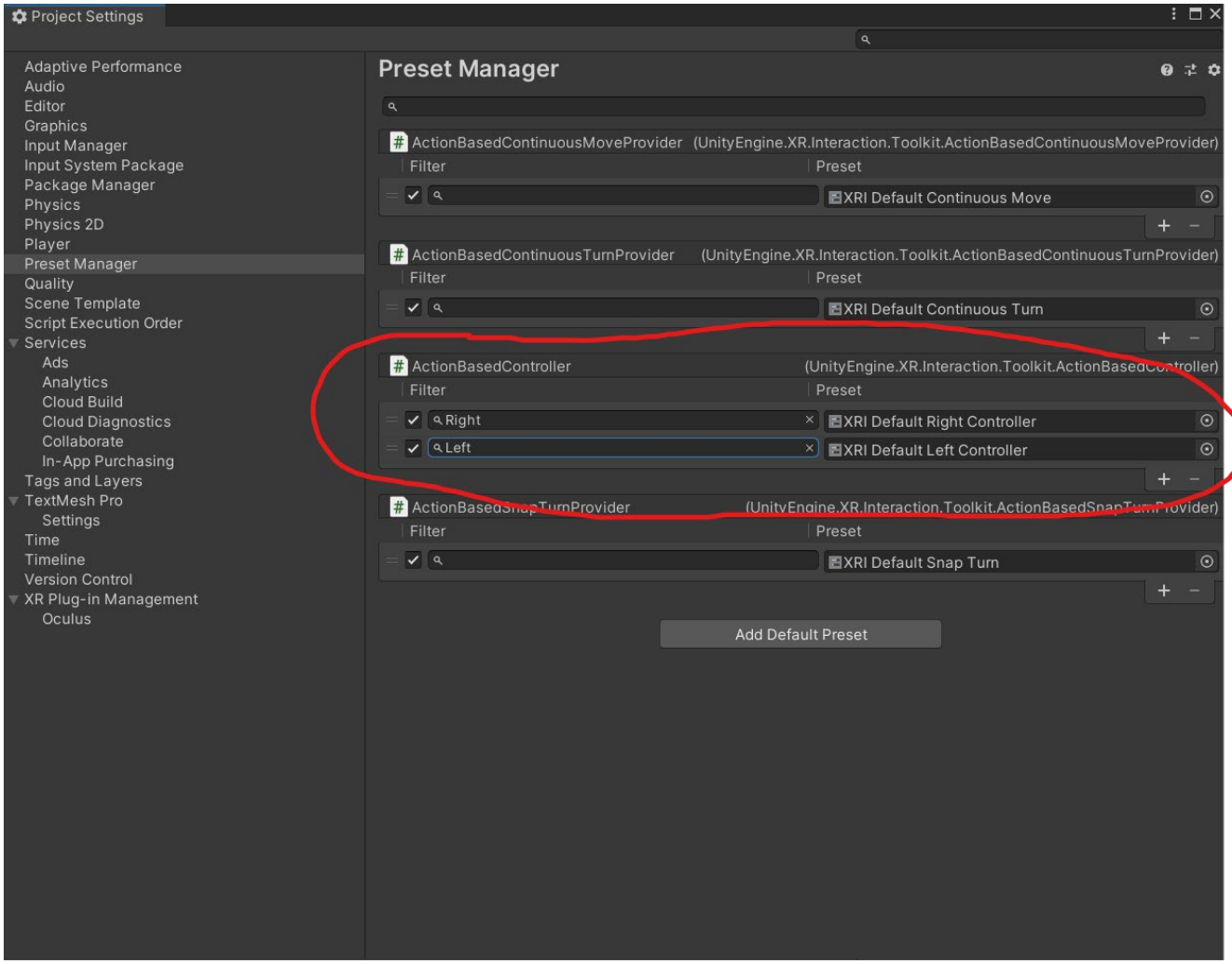
Next, click on XRI Default Left and Right Controller and click on *Add to ActionBasedController default.*



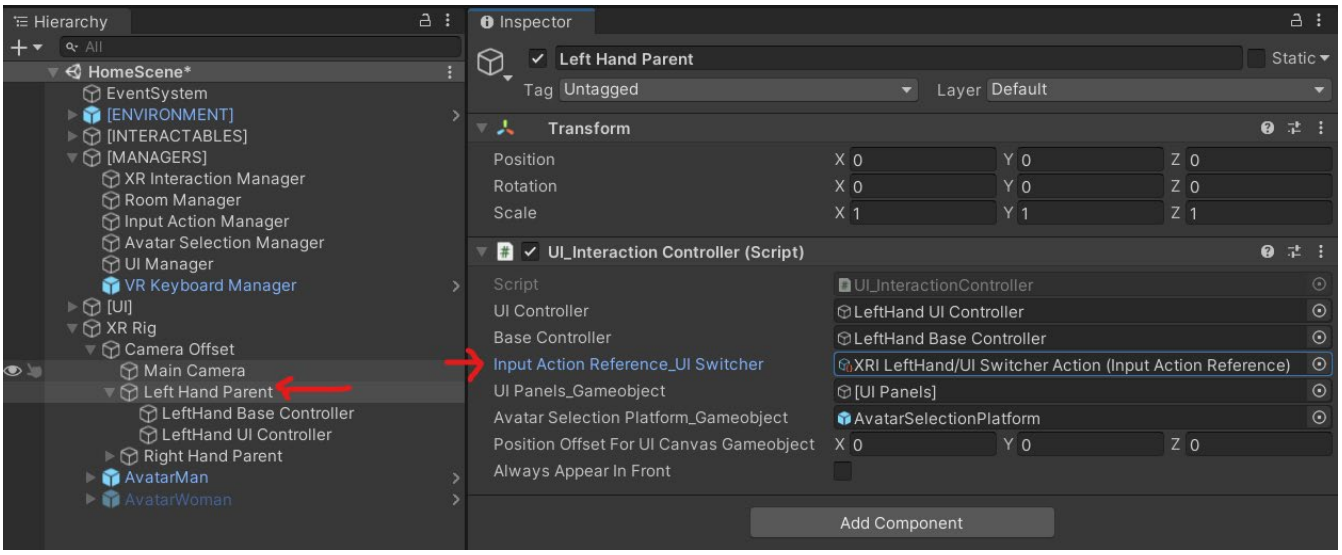


Do the same for the XRI Default Continuous Turn, Move, Snap Turn presets.

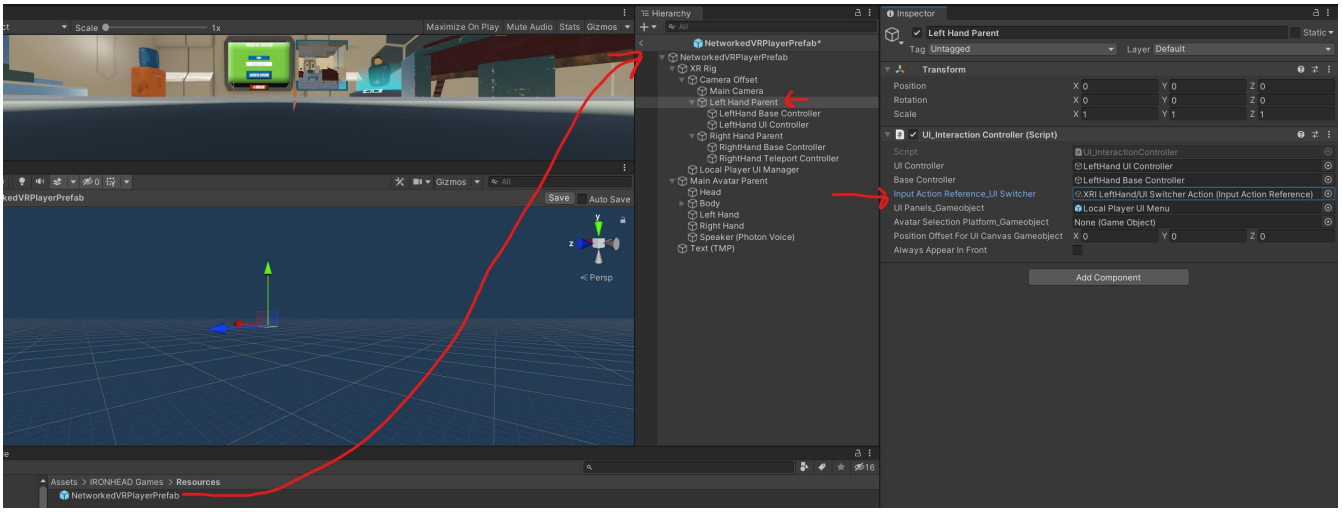
Lastly, open Edit > Project Settings > Preset Manager and fill Filters for XRI Default Left and Right Controllers as Right and Left respectively.



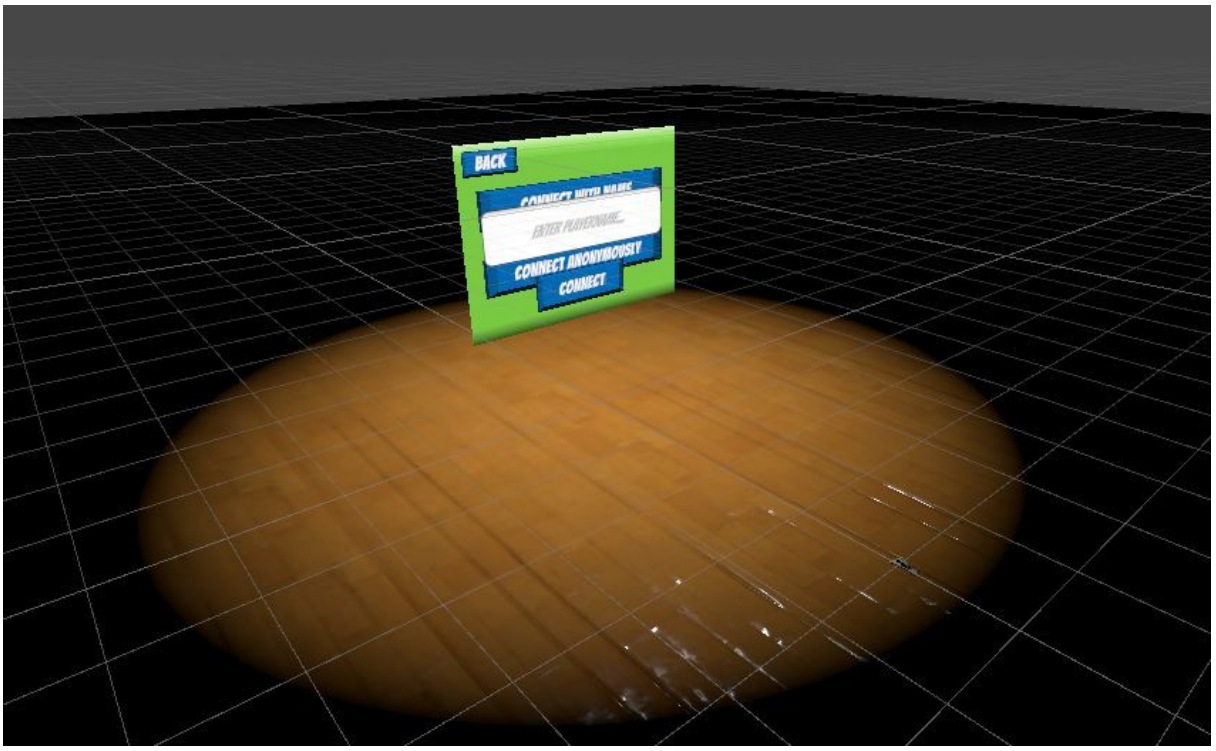
16. Now, open IRONHEAD Games > Scenes > HomeScene. And click on Left Hand Parent gameobject under XR Rig. Set the Input Action Reference as the LeftHand UI Switcher Action. And Save the scene.



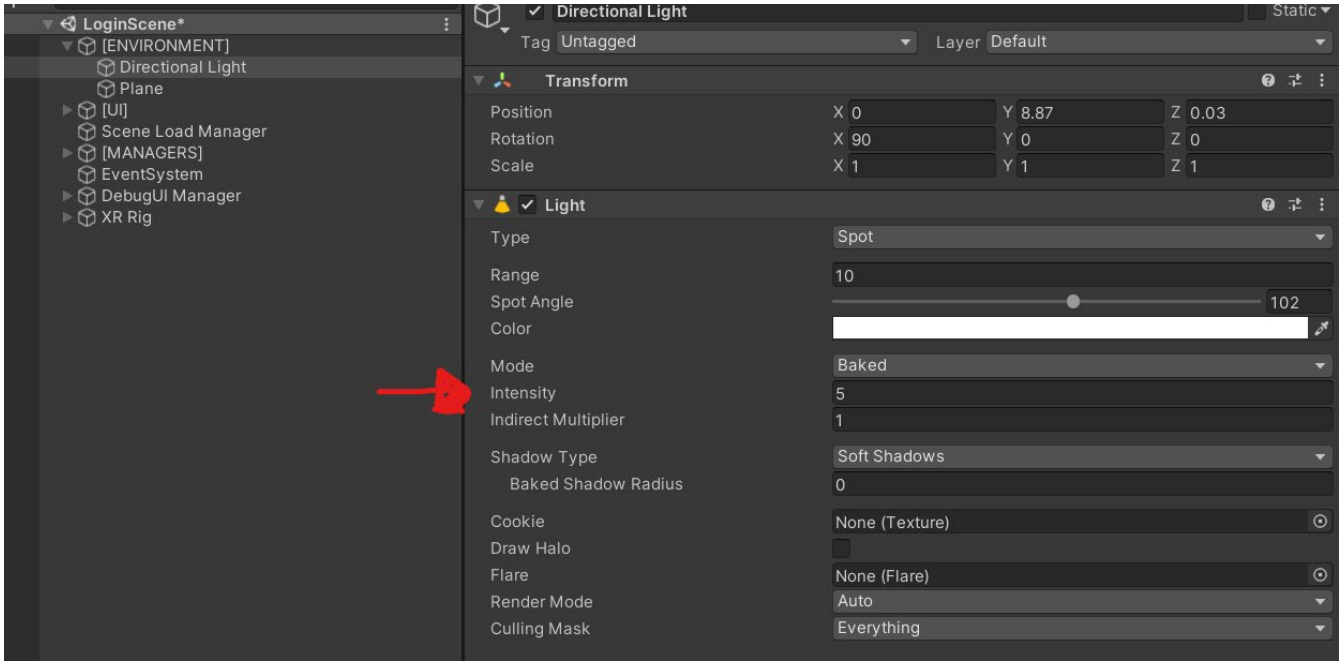
17. Also, do the same for the NetworkedVRPlayerPrefab gameobject under IRONHEAD Games Resources folder and save the prefab.



18. Lastly, open the LoginScene from IRONHEAD Games > Scenes.



Adjust the light settings so that the Login panel is visible enough.



Okay, now it is time to build this project to Oculus Quest and have fun with the basics of Multiplayer Virtual Reality.

To build this project to your Oculus Quest, you need to configure your Oculus Quest and do some other things to unlock the development.

To do this, you can follow the setup page from SideQuest:
https://sidequestvr.com/setup-howto

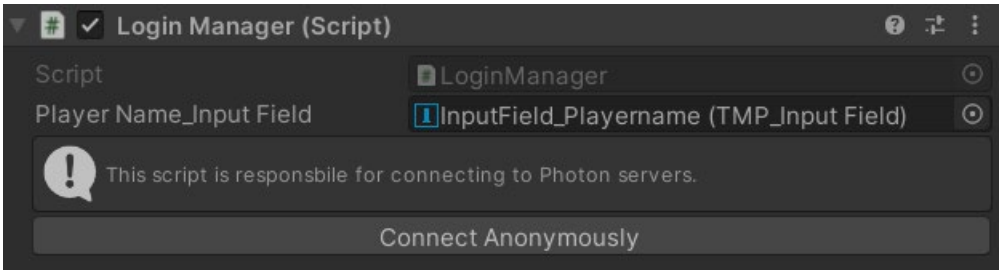
You can always ask your questions to me using this email address:
tevfikufuk@ironheadgamestudios.com

How to Test the Asset

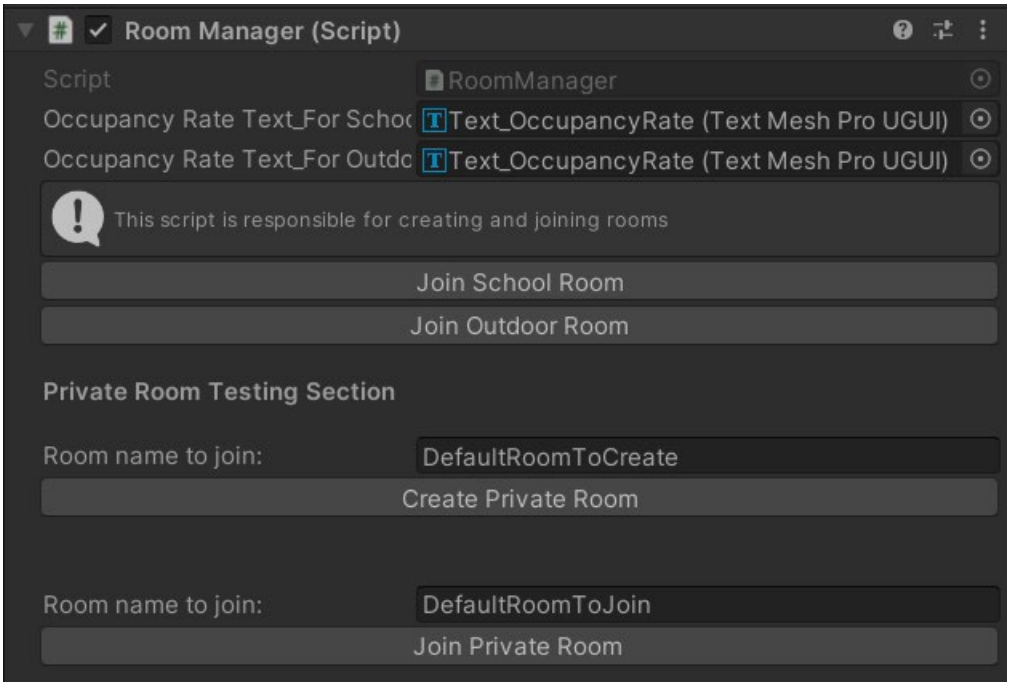
The main reason why this asset exists is to help developers quickly implement **Multiplayer VR** into their projects. Multiplayer VR Development can be tricky, especially in testing. Here is how I conduct the tests when preparing this asset and my other personal projects. Note that I will explain the testing with only one VR Headset and Unity Editor. More than one headset is better. But to get the debug messages, it is safe to include Unity Editor as one of the test platforms. Also, I assume that you can build and install your app/game to your Quest without any issue. If you do not know how to do that, this Youtube tutorial can help you:
https://youtu.be/JeFHgAblEAk

1. First, open Unity editor and your VR headset which is Oculus Quest 2 for me.
2. Connect your headset to your PC. Then, build and run the project to your Oculus Quest.
3. Next, wear the headset and open the app, connect to server and join a networked Room.

4. Then, in Unity Editor, hit play.
5. Click on LoginManager script and press Connect Anonymously editor button. It will load the HomeScene.

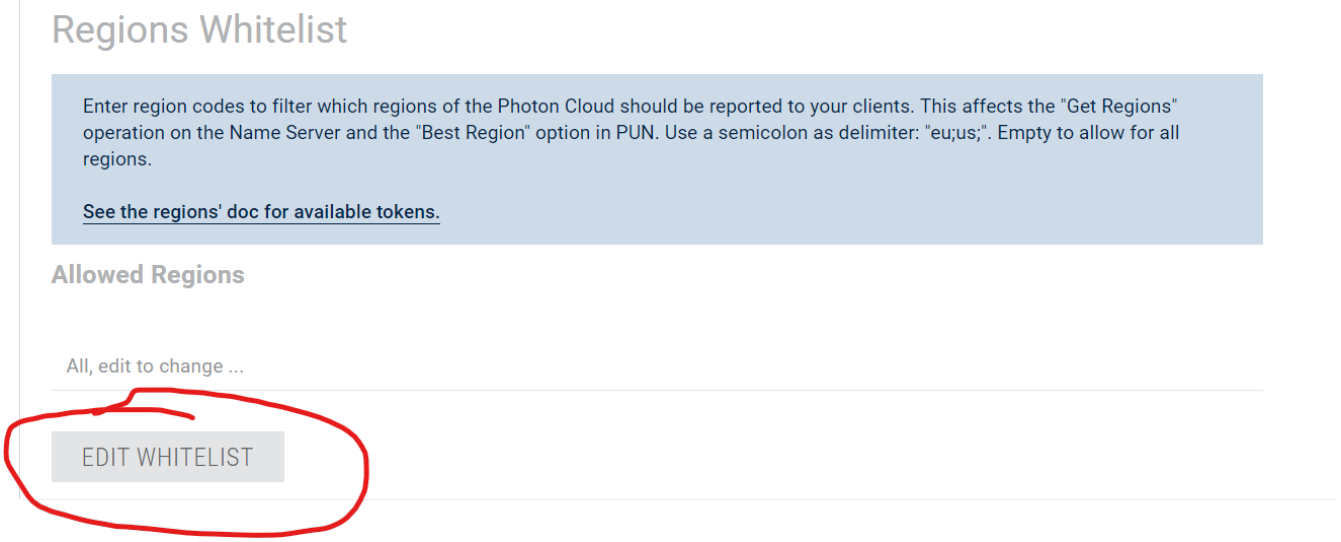


6. Next, click on RoomManager and using Editor buttons, join to the same room with your headset.

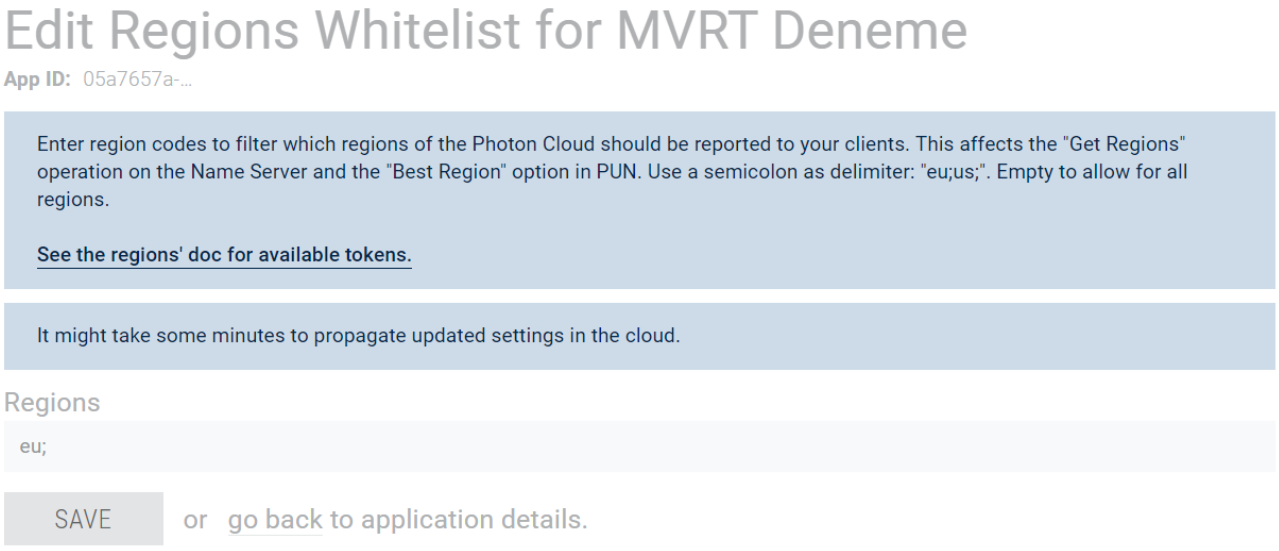


Note that these Editor buttons are coded to make the testing easier.

7. If you connect to the same room, you should see 2 VR players in the networked scene.
8. If you cannot see the other VR player, there is a chance that your test players connected to the different server regions due to server traffic. This is a common issue in Photon. To fix this issue, go to Photon Engine cloud dashboard and locate the cloud app that you get appID from. Then, here Edit Whitelist:



And enter a server region code that is nearest to you:



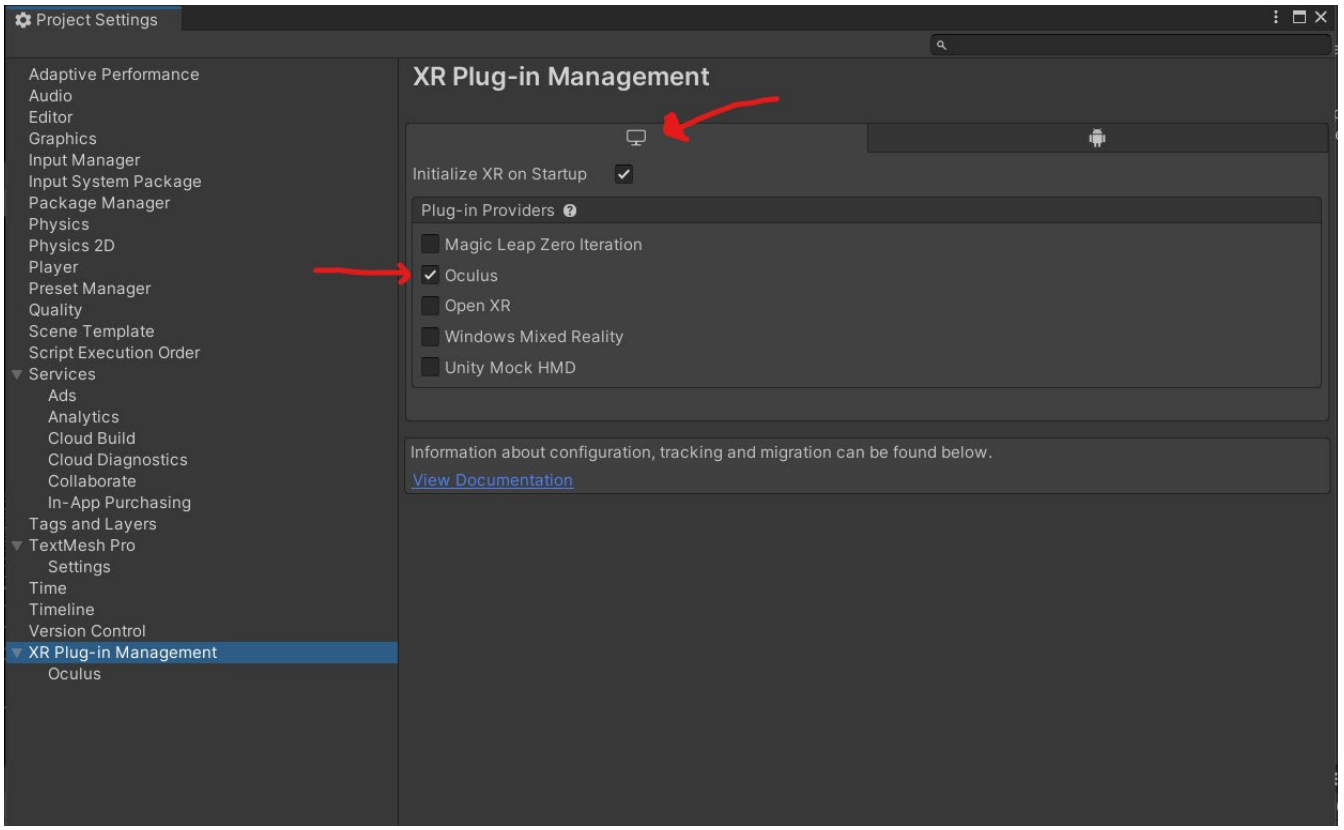
Done! You can use this method to test your Multiplayer VR projects with only one VR headset. Using Editor buttons, it is possible to test even without a VR headset. But you eventually need to make tests with a VR headset.

Oculus Link Setup

This asset is mainly built for Oculus Quest and Quest 2. However, you can test and build for PC VR thanks to Oculus Link which allows us play PC VR titles with Oculus Quest/2. More info for Oculus Link can be found here:
https://support.oculus.com/394778968099974/

To be able to use the asset with PC, first, we need to configure the Unity project. Here are the steps:

1. Open Edit> Project Settings. Click on XR Plug-in Management.
2. Under PC platform tab, check Oculus toggle.



3. That is it! Now, when you enable Oculus Link and hit play Unity Editor, you will see that you can test the asset in runtime.
4. Also, you can switch platform to PC and run the PC build.

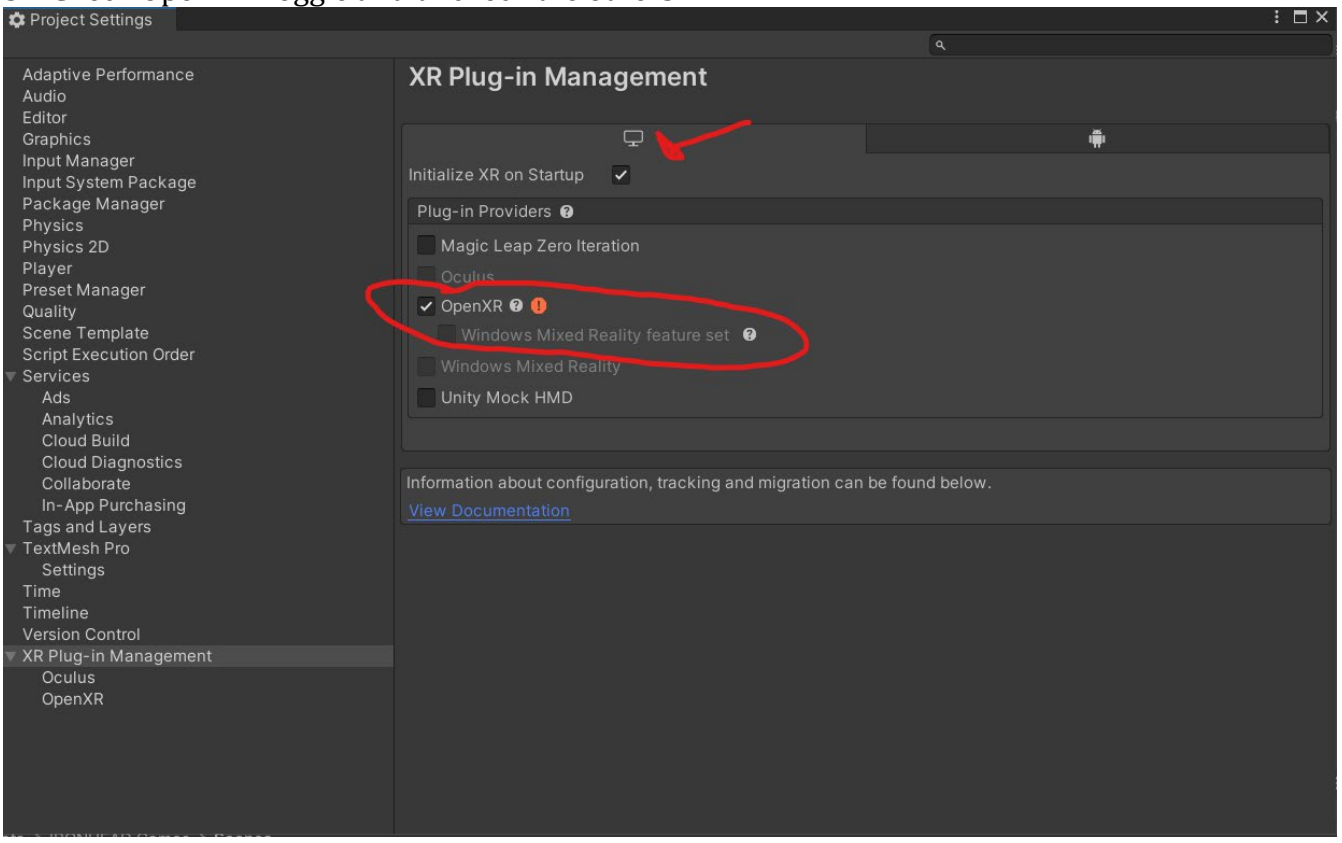
OpenXR Setup (For SteamVR)

OpenXR is a standard from Khoros group that provides cross-platform, high quality solution to target multiple platforms without developing separately for each platform. For example, thanks to OpenXR, we can make our VR projects target multiple VR headsets/platforms at the same time.

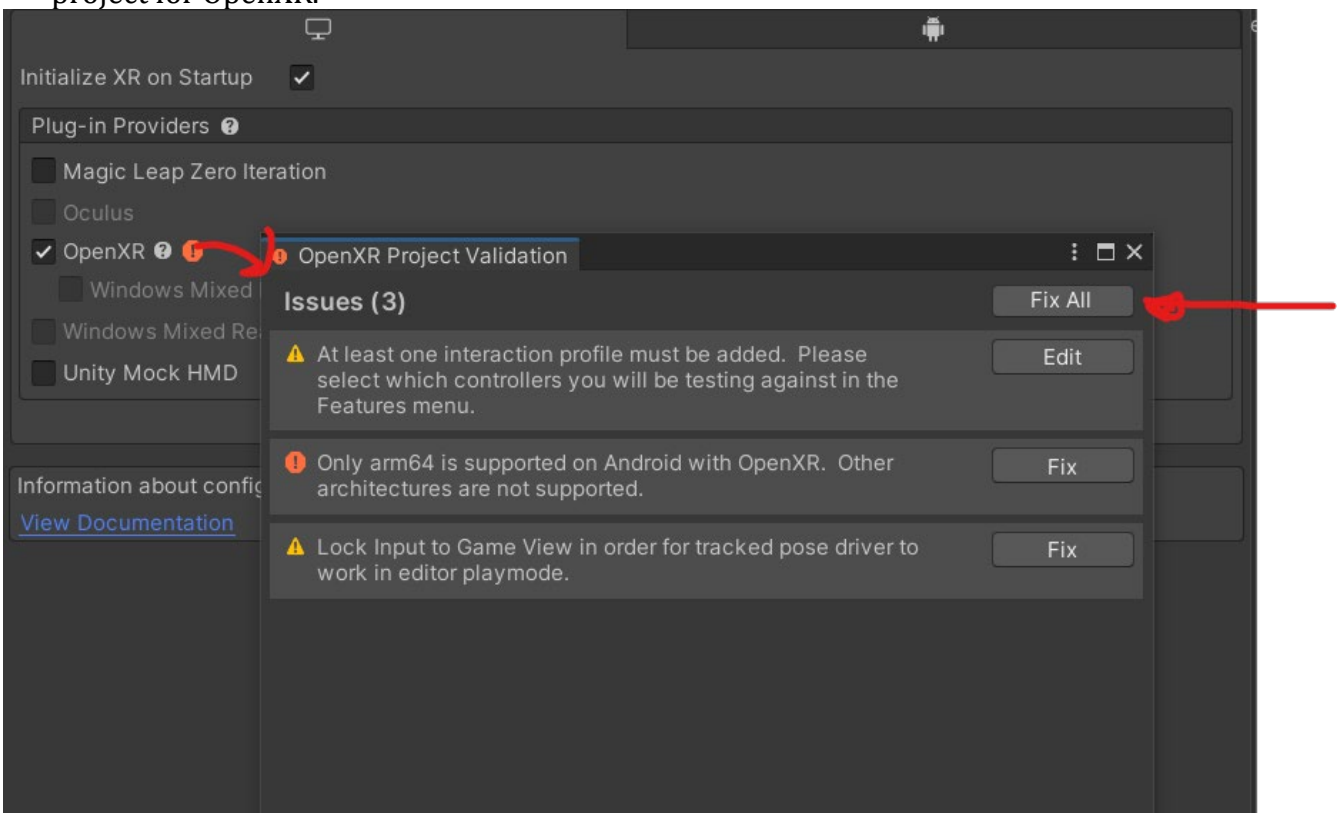
So, we can setup the project for OpenXR and run on both SteamVR and Oculus devices. Note that I do not have any SteamVR headset, so I did not test it yet. Let me know if that works for you. Note that OpenXR does not support Oculus Quest and Quest 2 yet. You can target only PC VR platform.

Here how you can enable OpenXR:

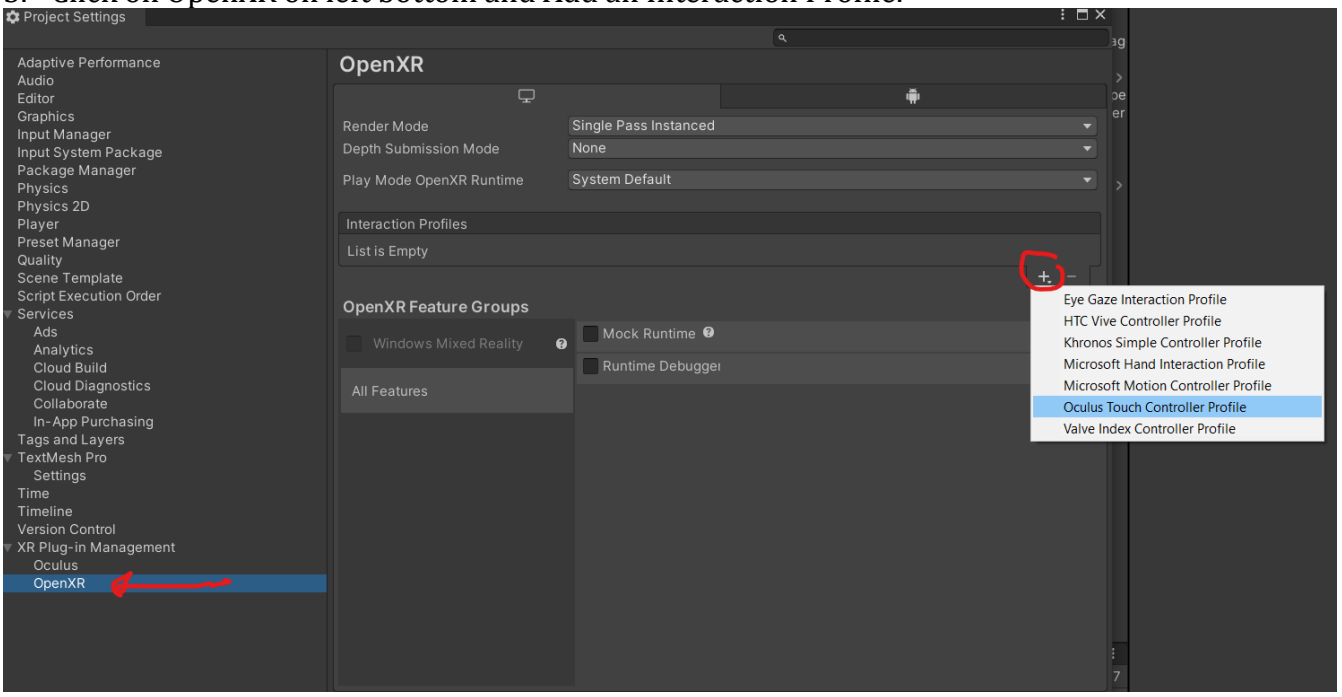
1. Open File > Project Settings > Player.
2. Click on XR Plug-in Management and enable PC tab.
3. Check OpenXR Toggle and uncheck the others.



4. Click on red error notification and press Fix All button. This will automatically configure the project for OpenXR.



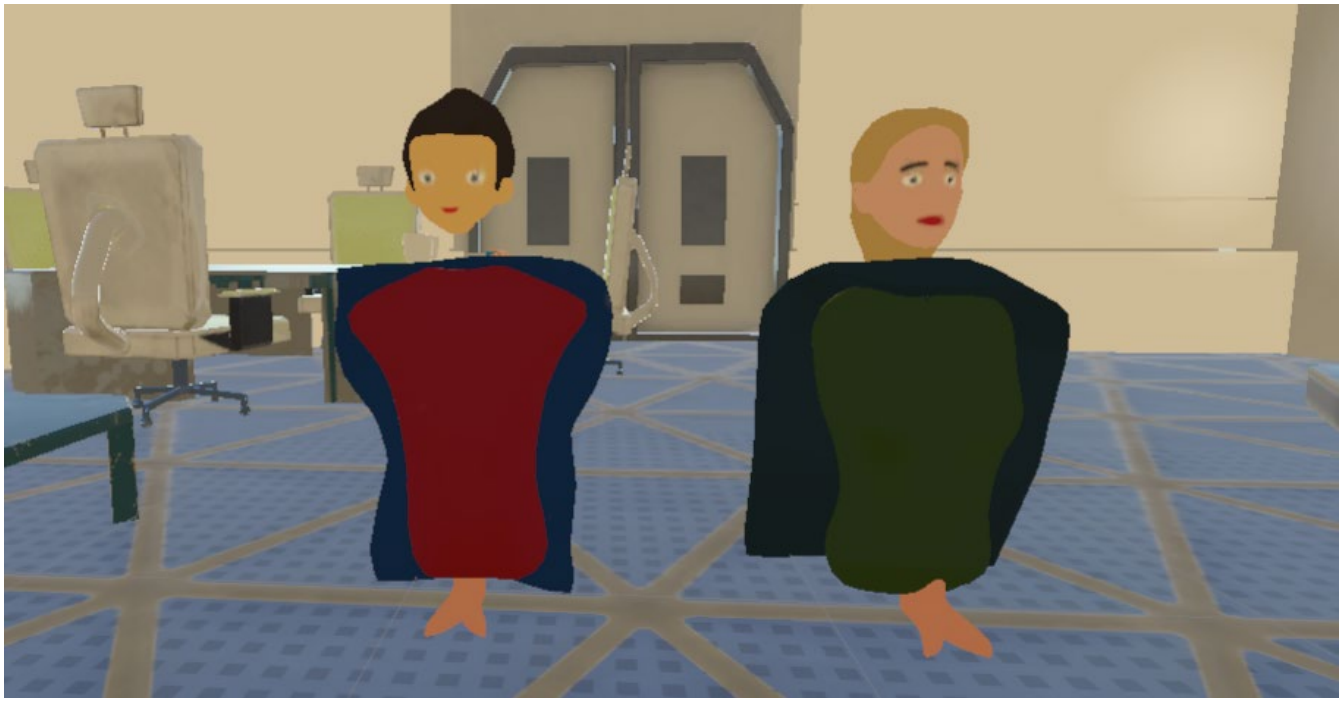
5. Click on OpenXR on left bottom and Add an Interaction Profile.



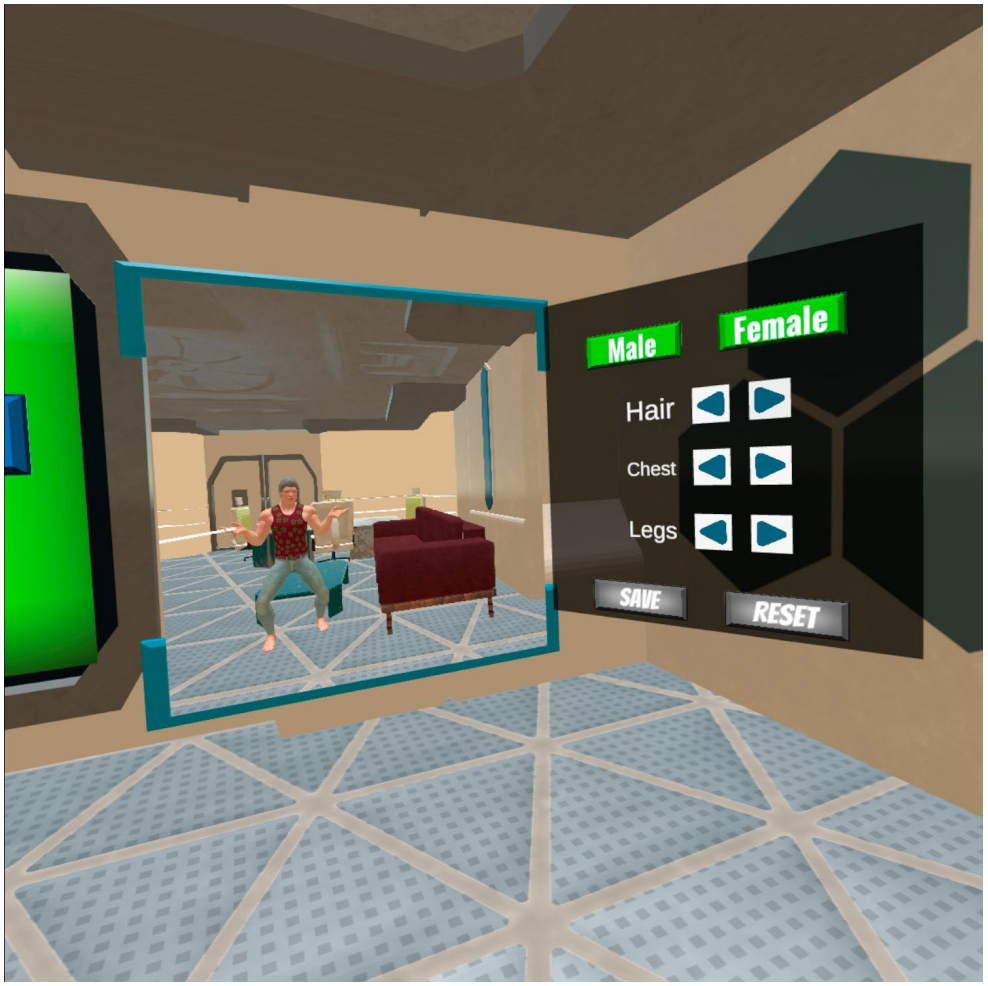
6. I will add Oculus Touch Controller Profile. If you have SteamVR headset, you can add the corresponding profile.
7. That is it! The project should now work with SteamVR or Oculus for PC VR Platform.
8. You can get more info here: https://www.khronos.org/openxr/

Project Documentation

Note that in this asset, we have simple VR Avatar Setup that contains separate Head, Body and Hands.



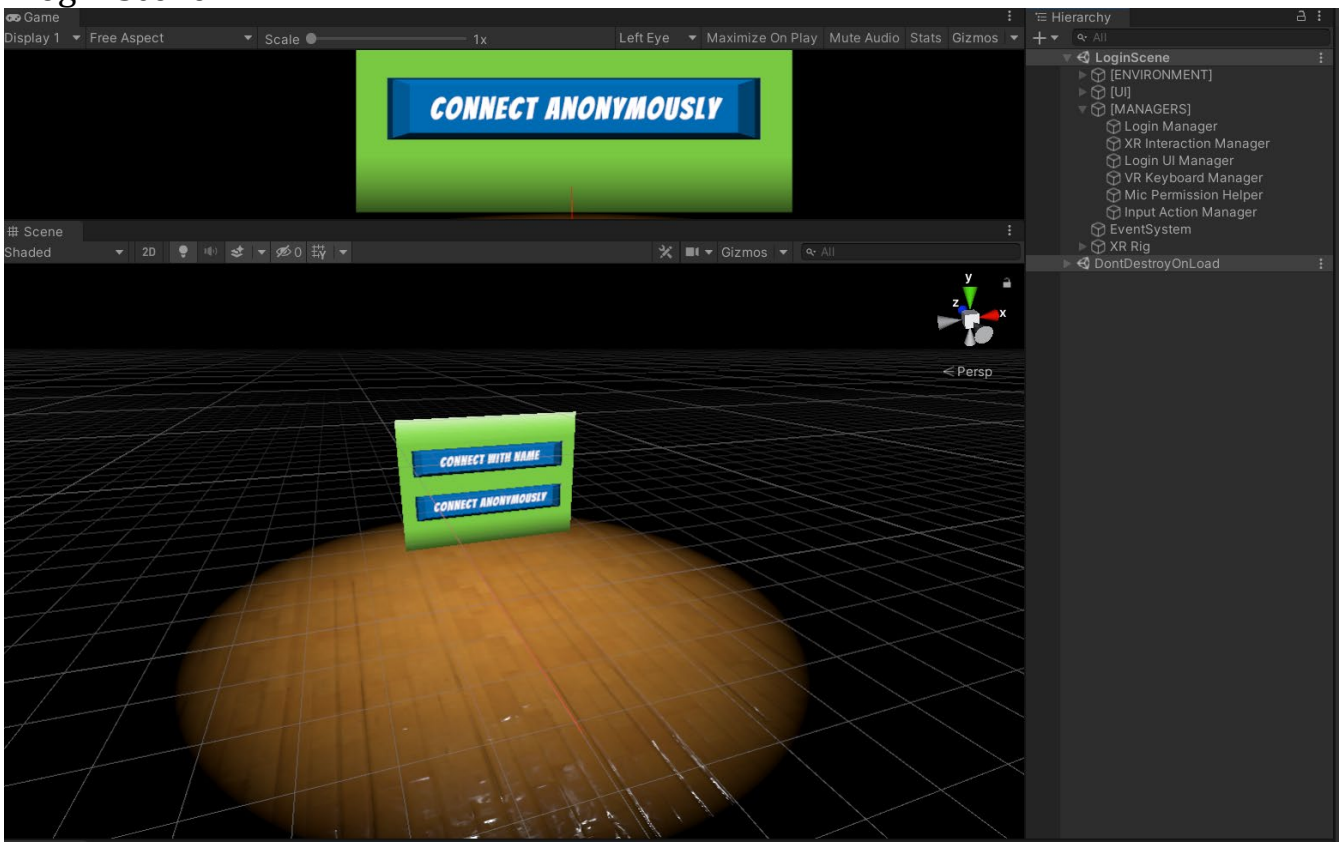
But with Final IK and UMA 2 Integrations, we have full body VR Avatar system.



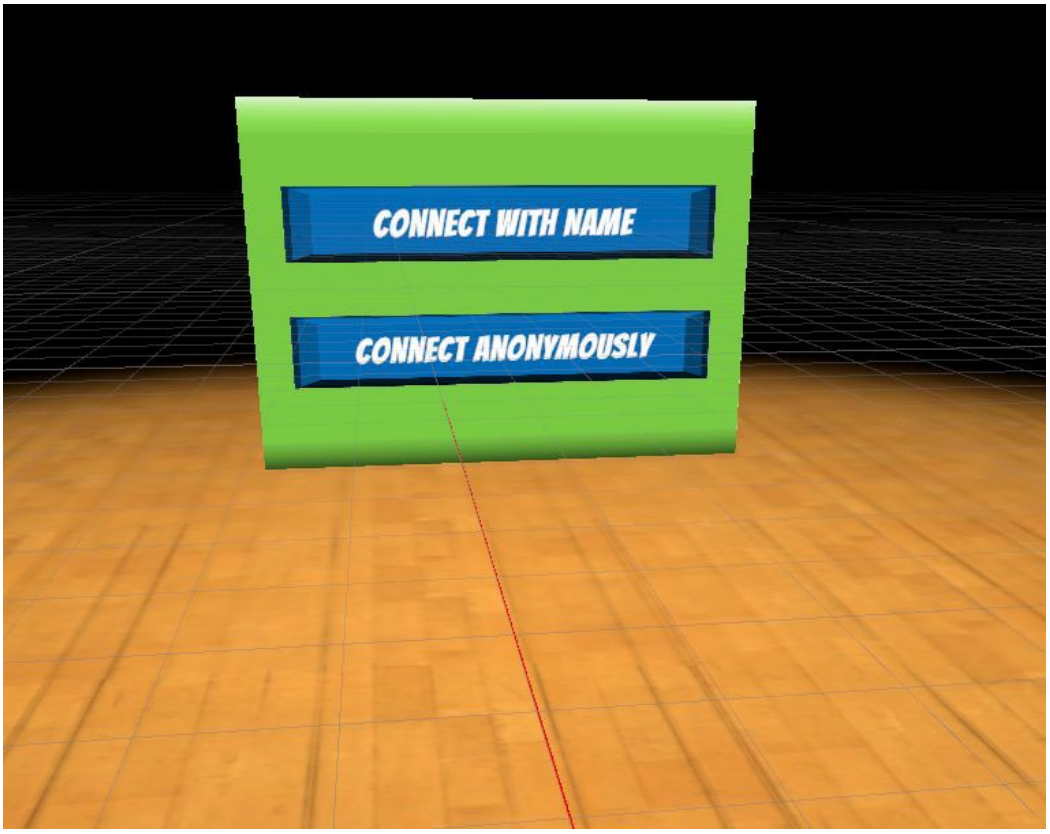
Available Scenes

There are 4 main scenes in the project: **Login Scene, Home Scene, World_School and,World_Outdoor Scene**.

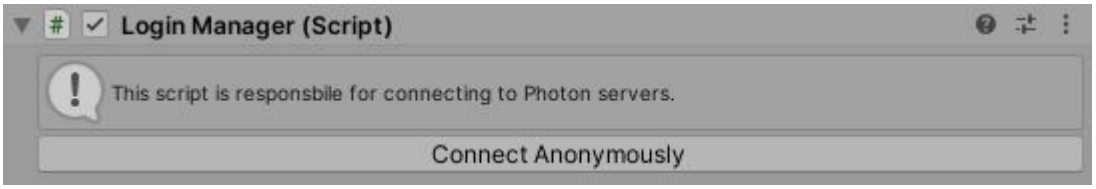
Login Scene



This scene is the first scene that is loaded when you open the app. It is a small, dark environment with a UI panel for login operations.



The most important part of this scene is the Login Manager.



Thanks to it, we can connect to Photon servers. Inside this script, there is a method called **ConnectToPhotonServer.**

```
#region UI Callback Methods
1 reference
public void ConnectToPhotonServer()
{
    if (PlayerName_InputField != null)
    {
        PhotonNetwork.NickName = PlayerName_InputField.text;
        PhotonNetwork.ConnectUsingSettings();
    }
}
#endregion
```

So, we get the player name string from VR Keyboard and call ***PhotonNetwork.ConnectUsingSettings()*** to connect to Photon servers. When we connect,there are some Photon callback methods that are called.

```
#region Photon Callback Methods
8 references
public override void OnConnected()
{
    Debug.Log("OnConnected is called. The server is available.");
}

14 references
public override void OnConnectedToMaster()
{
    Debug.Log("Connected to the Master Server with player name: "+PhotonNetwork.NickName);
    PhotonNetwork.LoadLevel("HomeScene");
}


#endregion
```

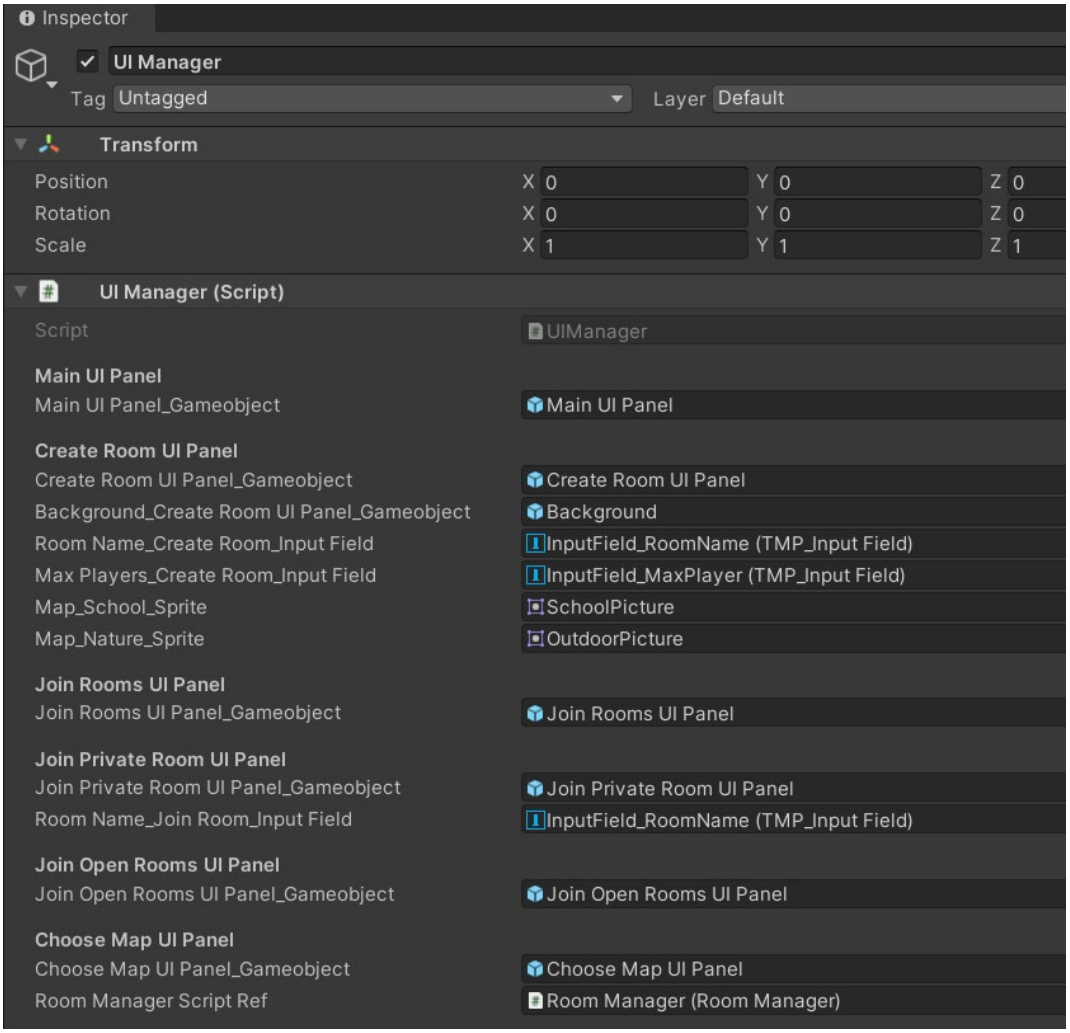
So, with these methods, we ensure that the connection is established. Then, we load theHome Scene using ***PhotonNetwork.LoadLevel()*** method.

Home Scene

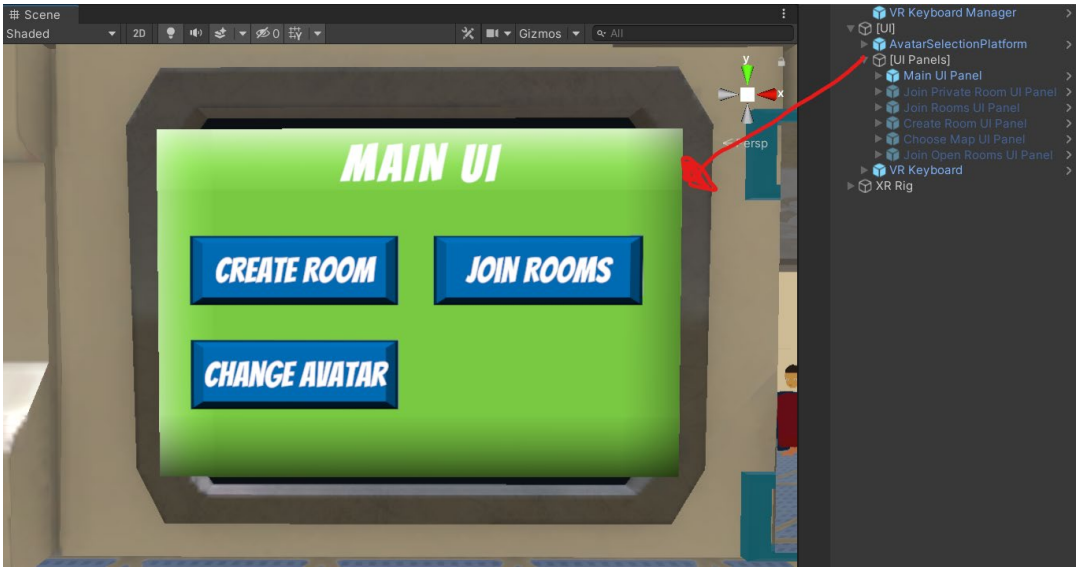
This scene is the main scene where the players do most of the operations such as joining virtual worlds and changing avatars.

Inside this scene, there are three main managers that operate this scene such that UI Manager,Room Manager, and Avatar Selection Manager. XR Interaction Manager and Input Action Manager come from the XR Interaction Toolkit and Unity's new input system.

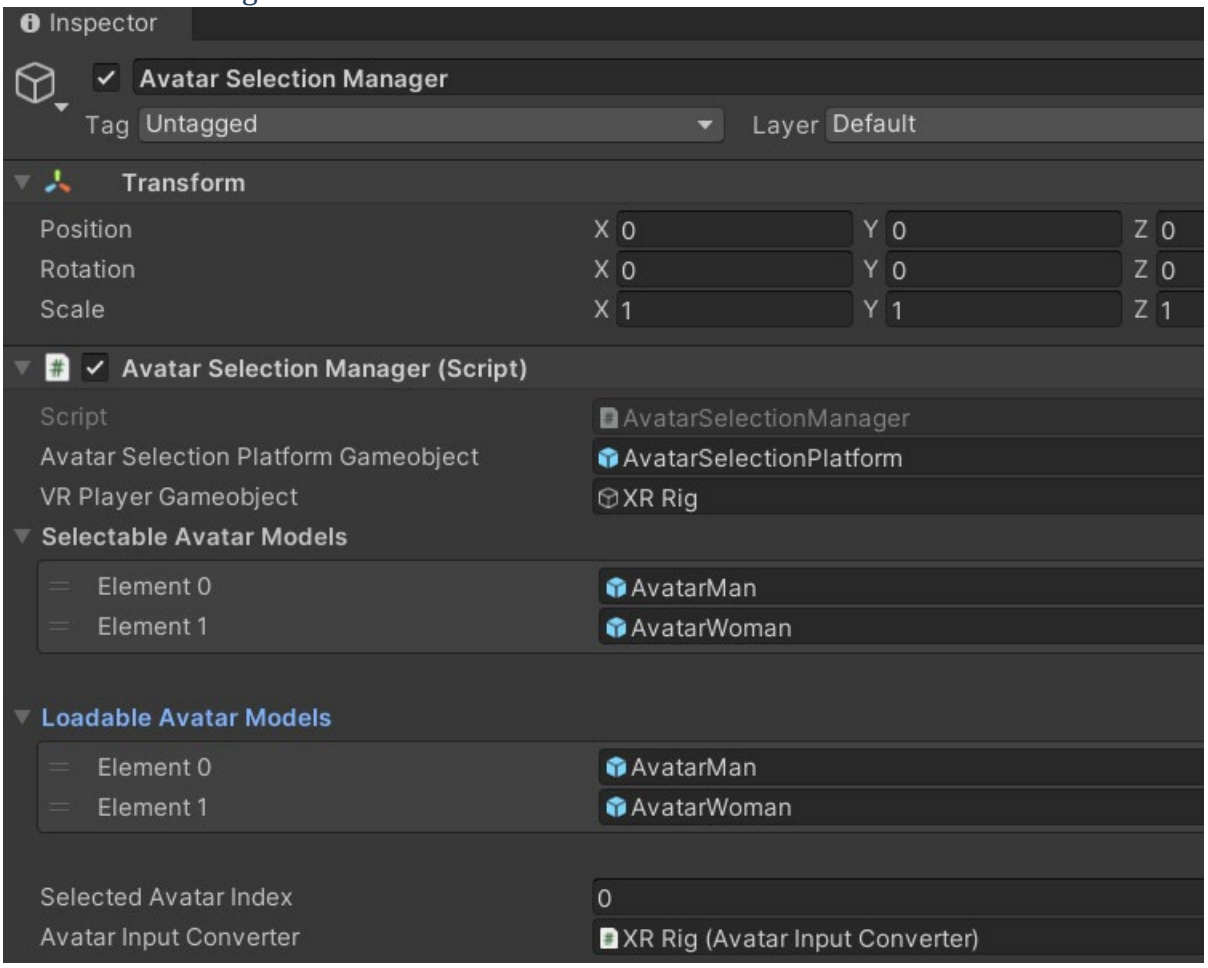
UI Manager



UI Manager allows us manage UI Panels and call methods based on player's UI inputs.

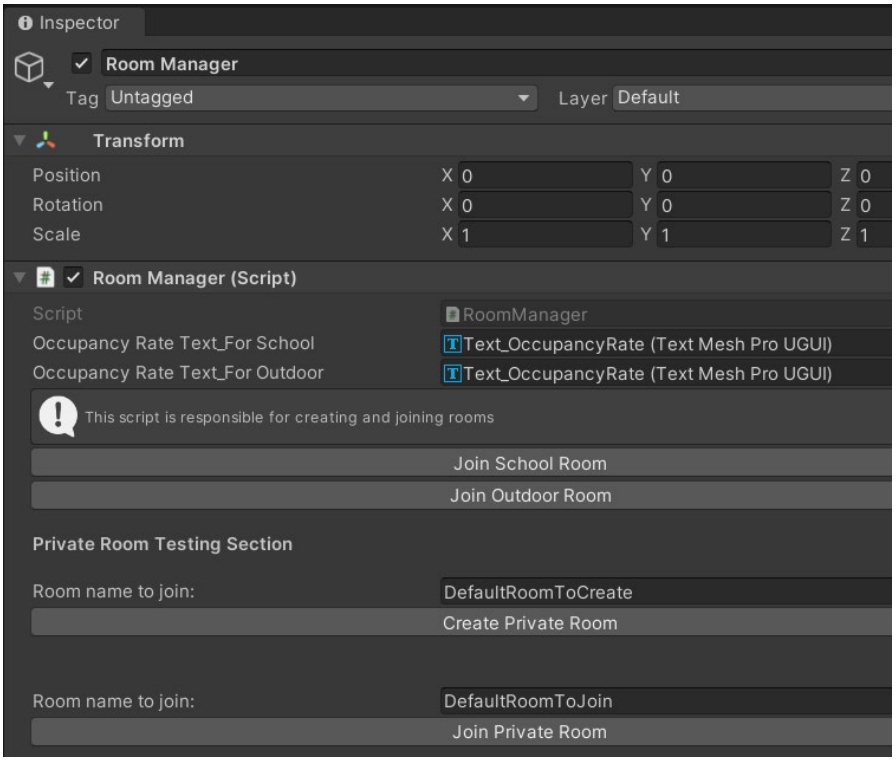


Avatar Selection Manager



It is the main script that manages Avatar selection. It holds the selectable avatar models andsaves the selection data across the scenes.

Room Manager

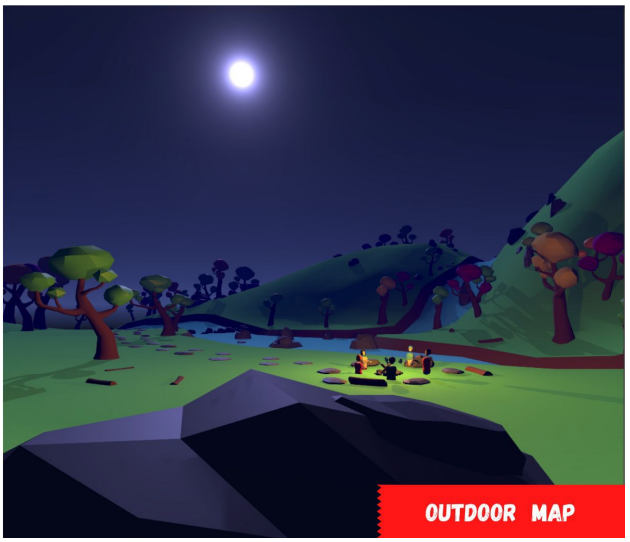


Room Manager script manages room operations such that joining rooms. It uses Photonclasses to join and list the existing rooms as well as creating rooms. When we joined the room, the **OnJoinedRoom** photon callback method is called. Then, we call PhotonNetwork.LoadLevel to load the scene according to the room map type such as World School or World Outdoor scenes.

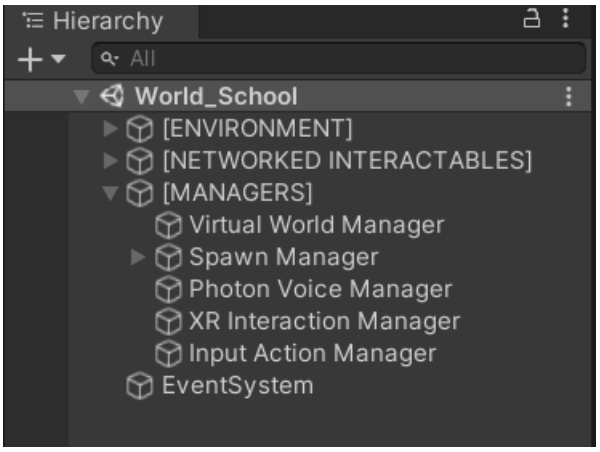
This script has also Editor buttons to make the testing easier.

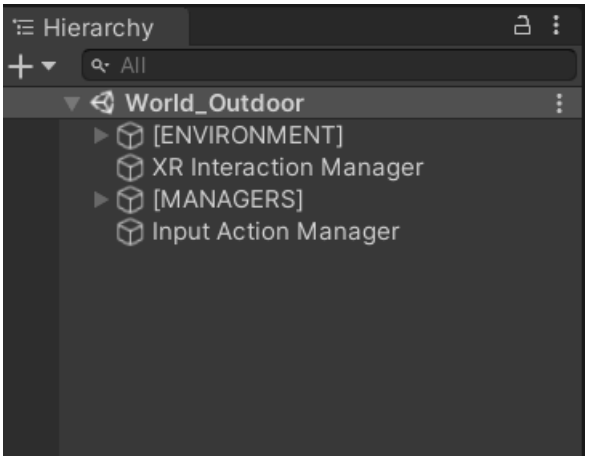
School and Outdoor Scenes- Networked Scenes

In this asset, there are 2 different, built-in virtual worlds; School and Outdoor environment.



These scenes are loaded when the players join the networked rooms. You can think of them like virtual worlds where VR Players join and interact with each other.

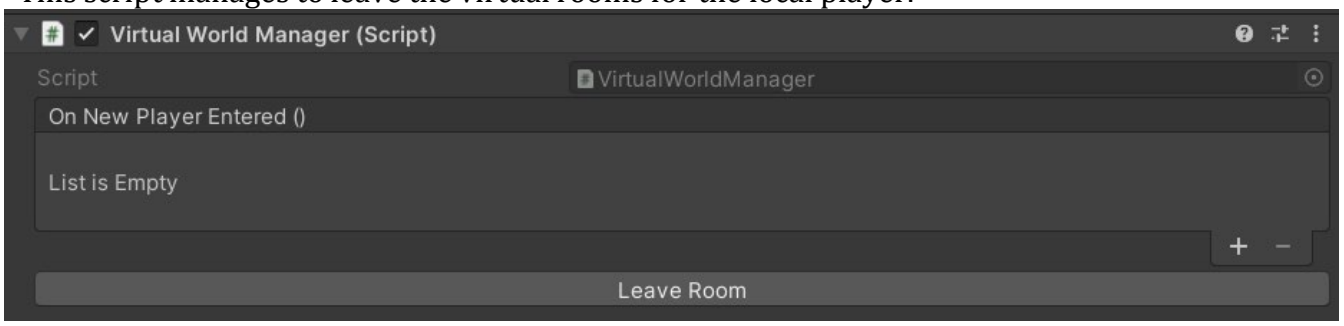




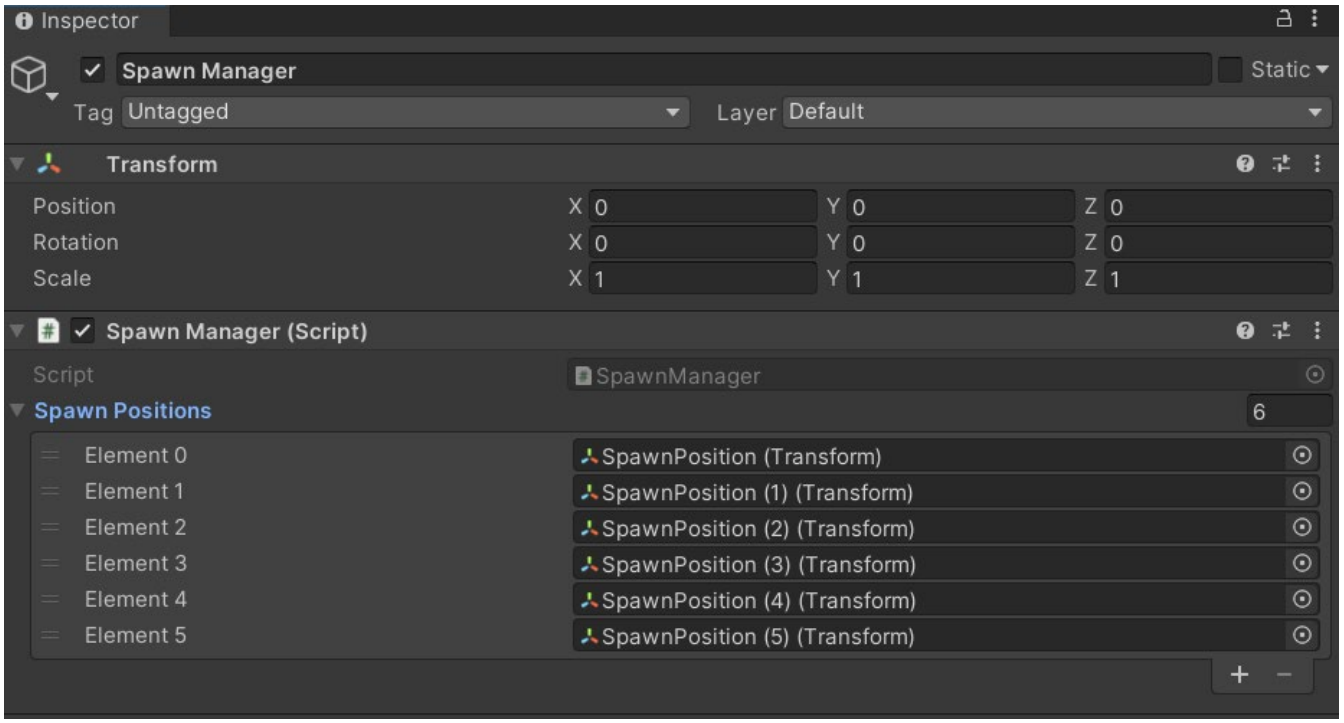
In these networked scenes, there are three main managers such that: **Virtual World Manager, Spawn Manager,Photon Voice Manager.**

Virtual World Manager

This script manages to leave the virtual rooms for the local player.

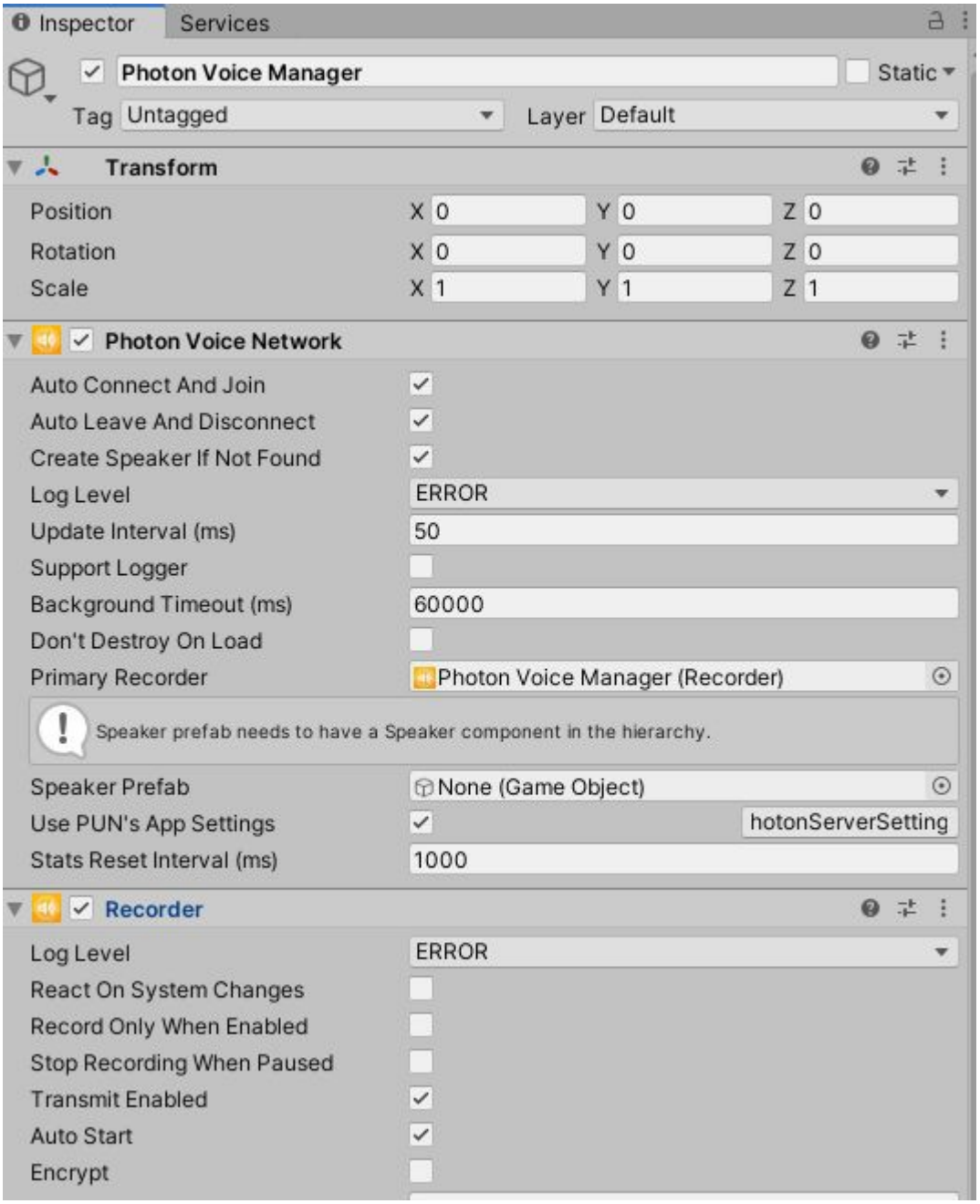


Spawn Manager



This script spawns the Networked VR Player Prefab across the network.

Photon Voice Manager



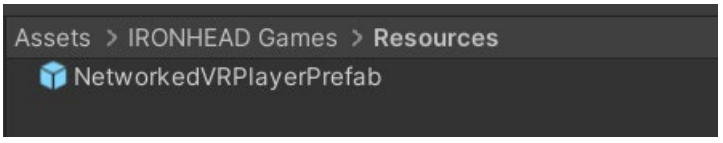
It manages Voice Chat operations. For more info, check this official Photon Voicedocumentation: https://doc.photonengine.com/en-US/voice/current/getting-started/voice-for-pun

VR Player Synchronization

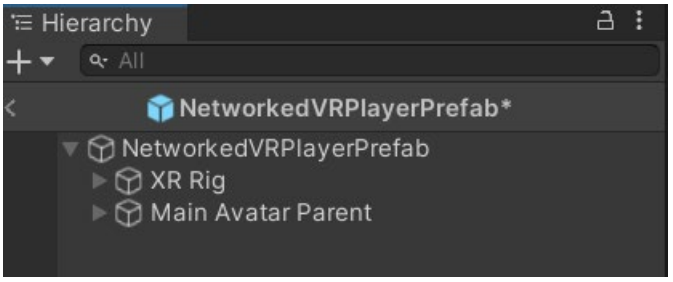
First, we have one generic VR Player prefab that is instantiated across the network using ***PhotonNetwork.Instantiate()*** method.

```
public class SpawnManager : MonoBehaviour
{
    [SerializeField]
    GameObject GenericVRPlayerPrefab;

    public Vector3 spawnPosition;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        if (PhotonNetwork.IsConnectedAndReady)
        {
            PhotonNetwork.Instantiate(GenericVRPlayerPrefab.name,spawnPosition,Quaternion.identity);
        }
    }
```

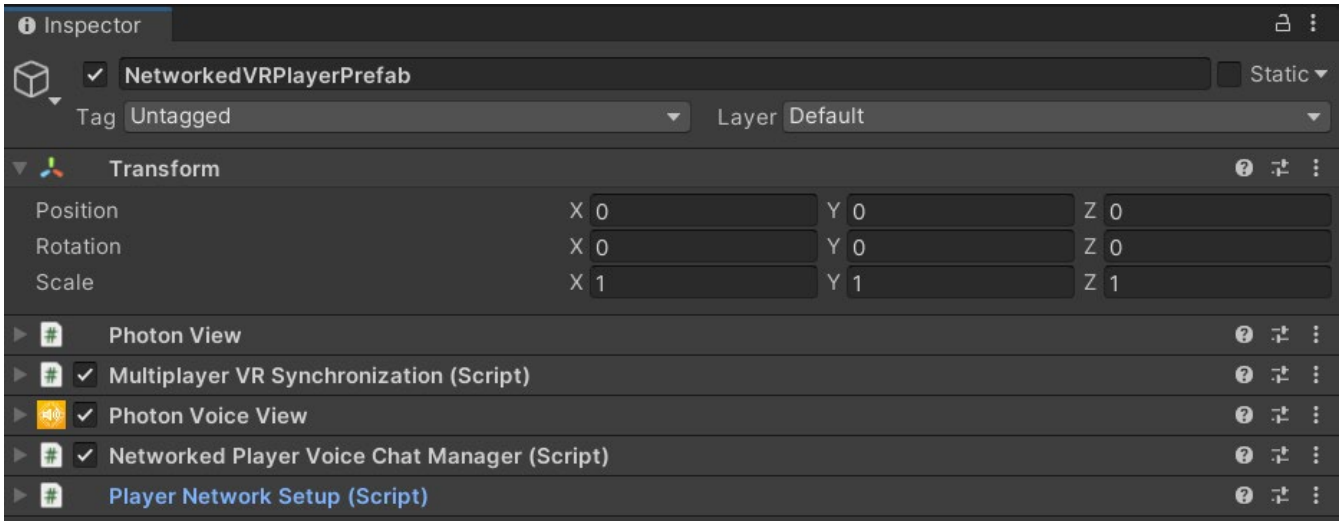
This prefab must be located under the **Resources folder** according to Photon.



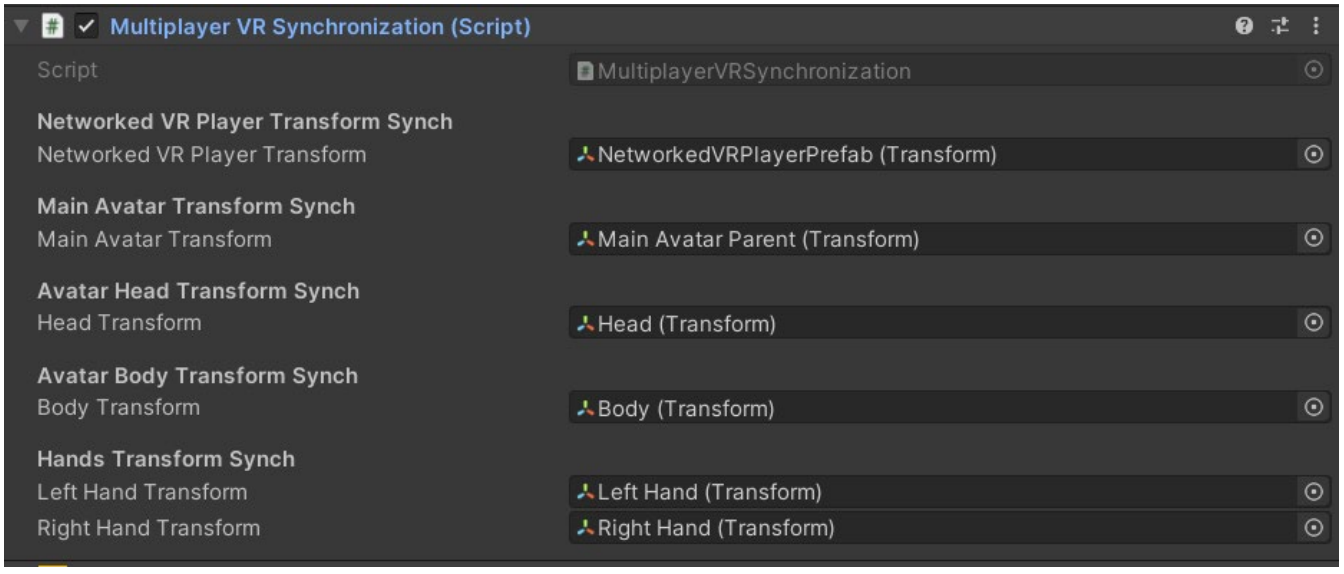
It contains 2 main game objects such that **XR Rig** and **Main Avatar**:



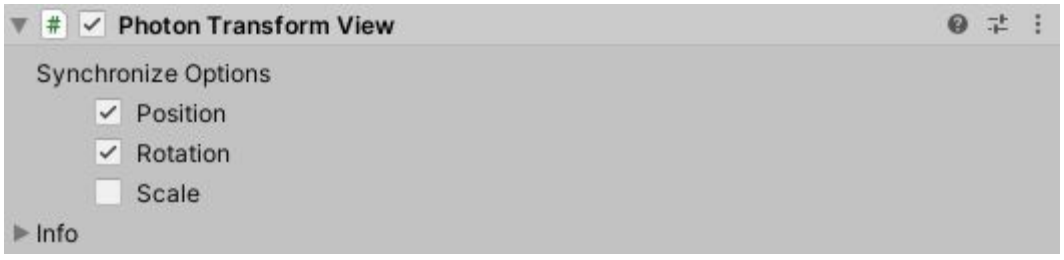
Here is the inspector of the Networked VR Player prefab:



The most important script for VR Player Synch is the **Multiplayer VR Synchronization** script.



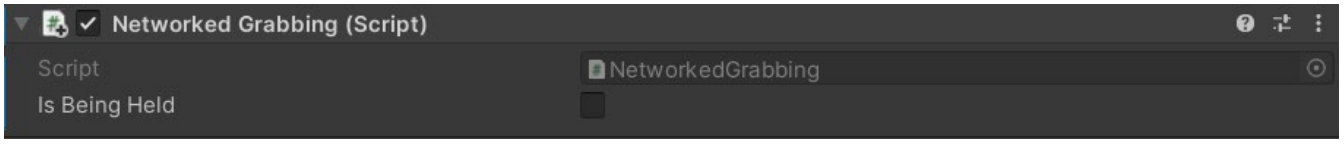
This script is written as customizable as possible for full VR synchs such as VR Head andcontrollers. The concept is similar to **PhotonTransformView**.



Networked Grabbing

Multiplayer VR Social asset contains the Networked Grabbing feature, too. This means that when a VR Player grabs an object, this will be synchronized for the remote players. This is an important feature for Multiplayer VR experiences. This feature is inside the World_School scene.

For Networked Grabbing, we have a main script called **NetworkedGrabbing**.



This script uses **Photon's Ownership Transfer** and **Remote Procedure Calls (RPCs)** tosynchronize the grabbing.

These are the main RPCs that make the networked grabbing possible.

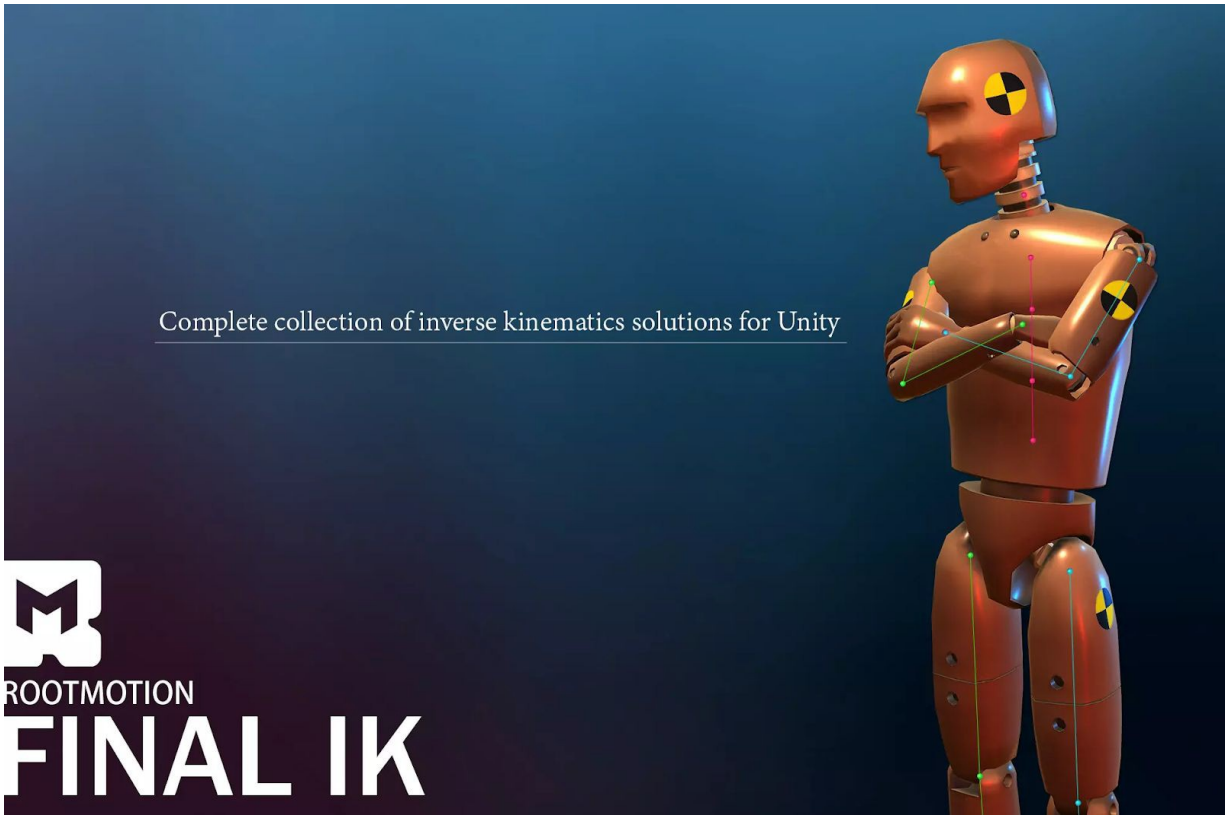
```
[PunRPC]
0 references
public void StartNetworkGrabbing(string whatIsGrabbed) // When grabbing begins change the bool to represent that the object is being held
{
    Debug.Log("StartNetworkGrabbing()");
    if (whatIsGrabbed == objectName)
    {
        isBeingHeld = true;
    }
}



[PunRPC]
0 references
public void StopNetworkGrabbing(string whatIsGrabbed) // When the object is released change the bool to represent that the object is not being held
{
    Debug.Log("StopNetworkGrabbing()");
    if (whatIsGrabbed == objectName)
    {
        Debug.Log("theGrabbedObjectView.gameObject.name == gameObject.name");
        isBeingHeld = false;
    }
}
```

Integrations

Currently, Multiplayer VR Template (MVRT) supports two 3rd party assets which are Final IK and UMA 2- Unity Multipurpose Avatar assets. Thanks to these integrations, MVRT has the basic, customizable multiplayer Full body VR Avatar system. Note that UMA 2 Integration comes withing the Final IK integration. More detail on how to setup the integrations can be found below.

Final IK Integration

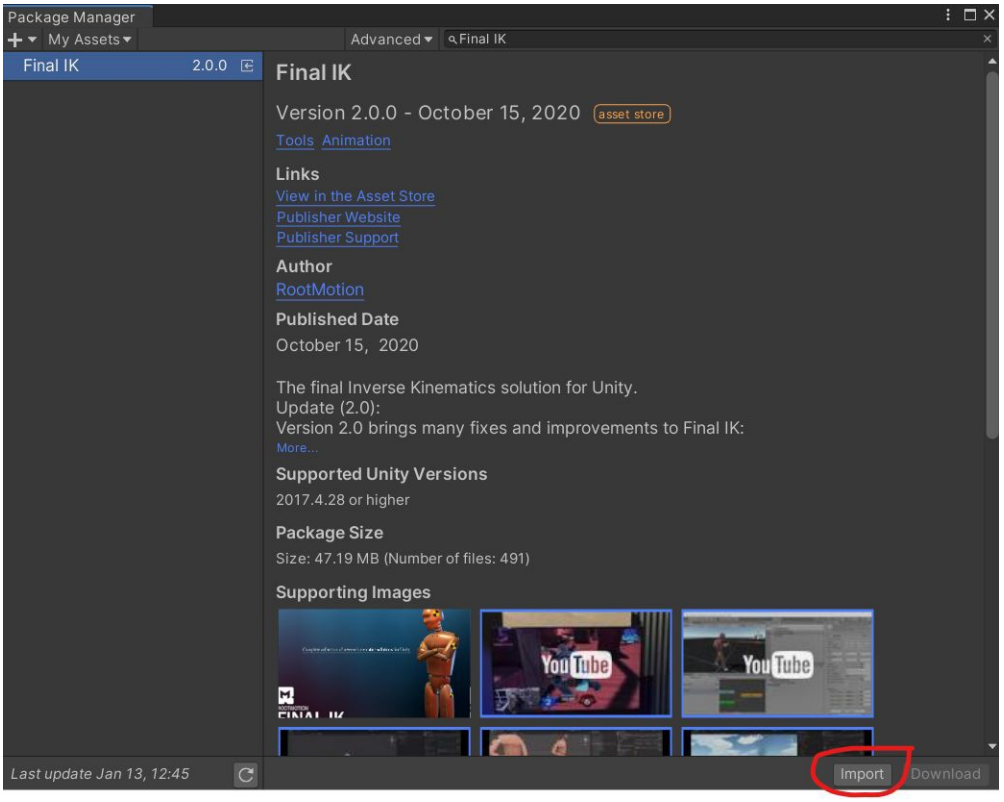
Final IK is an advanced inverse kinematics library for Unity. It allows developers to easily setup the full body IK systems for characters.



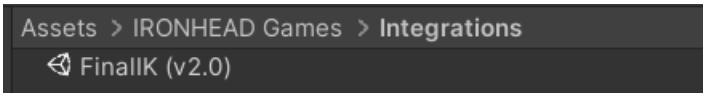
Final IK is really a great fit as a VR Full body solution. So, this integration will provide a verysolid Full Body VR system in Multiplayer VR Template.

How to set up Final IK Integration

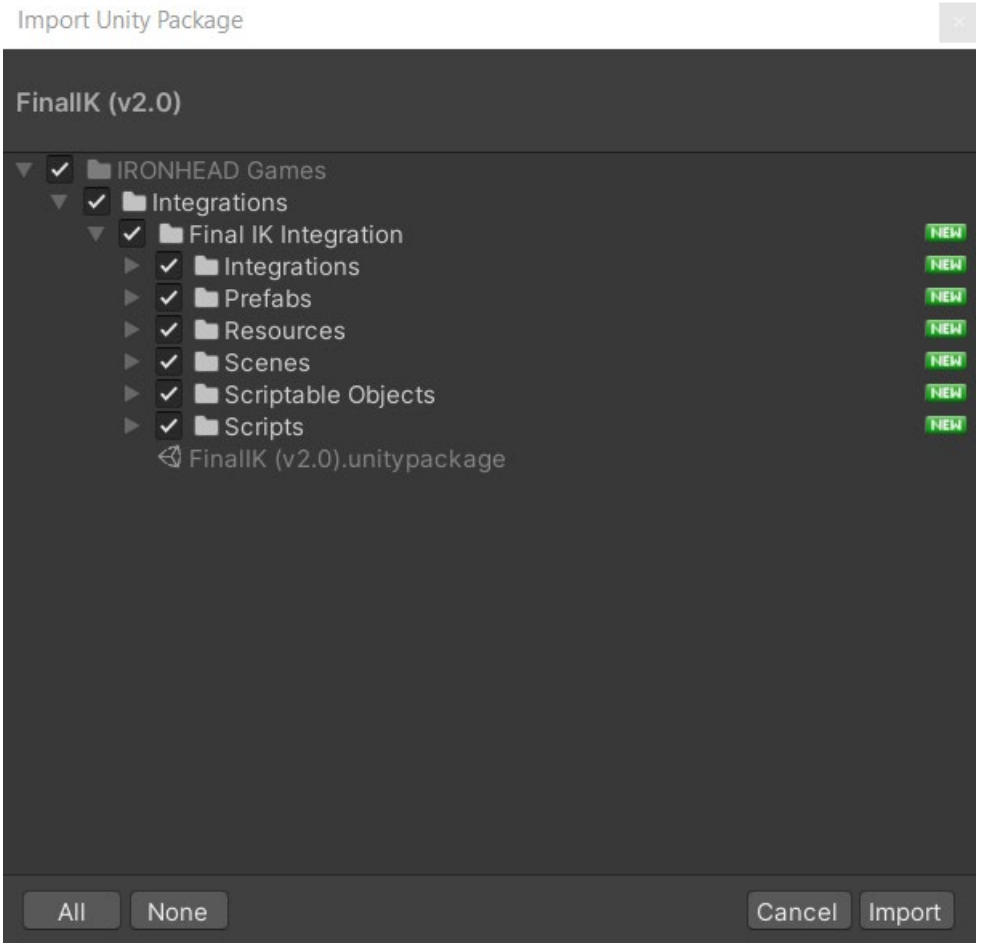
1. First of all, to be able to use this integration, you need to own the Final IK asset onUnity Asset Store.
2. If you already have it, download, and import Final IK into your project via Package Manager.



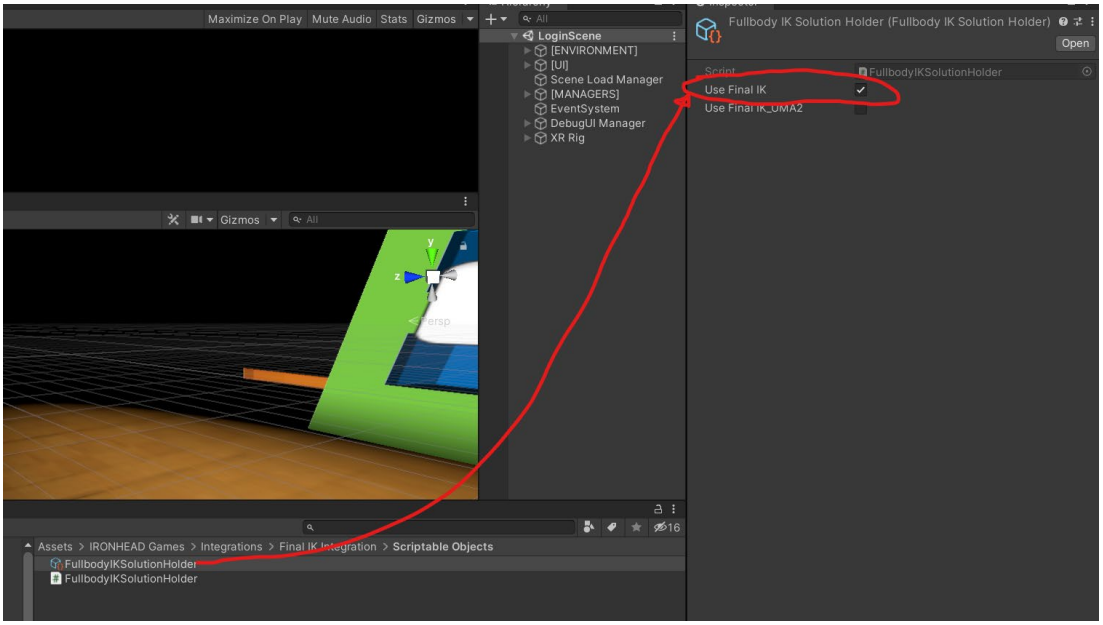
3. Then, go to IRONHEAD Games > Integrations > Final IK and import *the Final IK Integration Package* into your project.



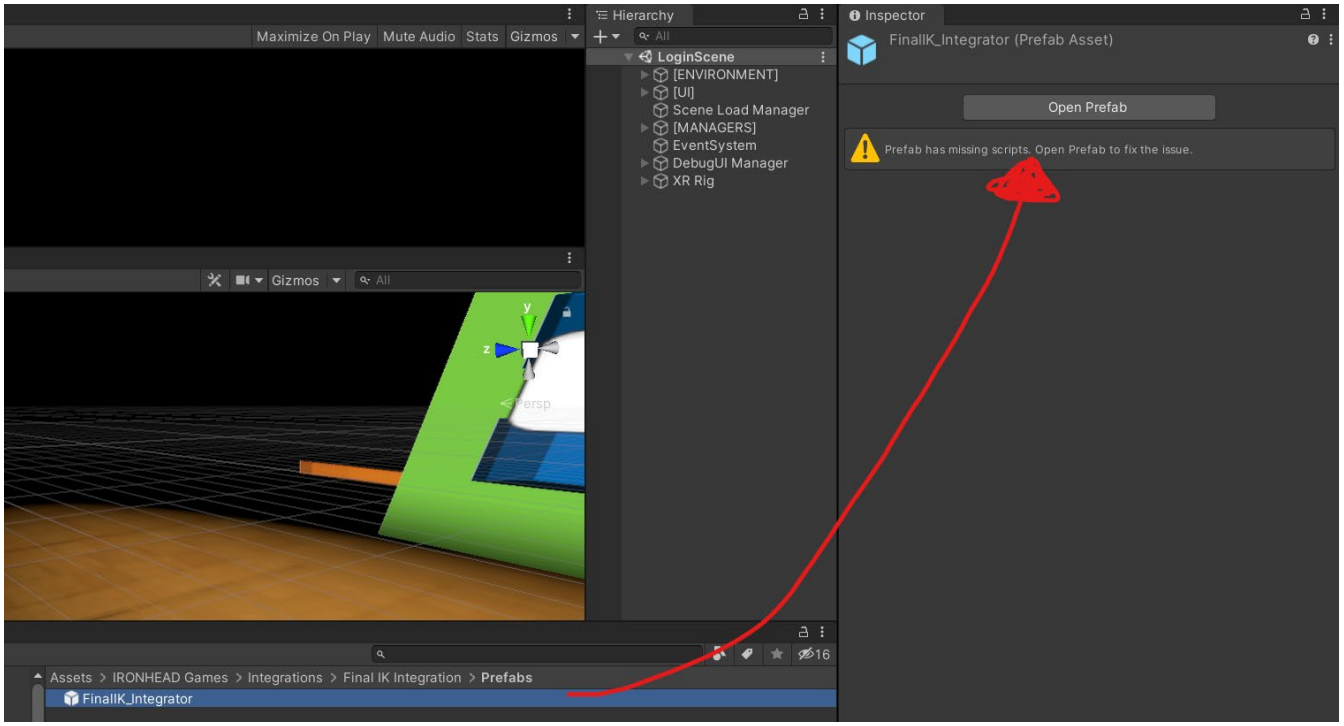
4. This package contains the necessary scripts and scenes for Final IK setup.



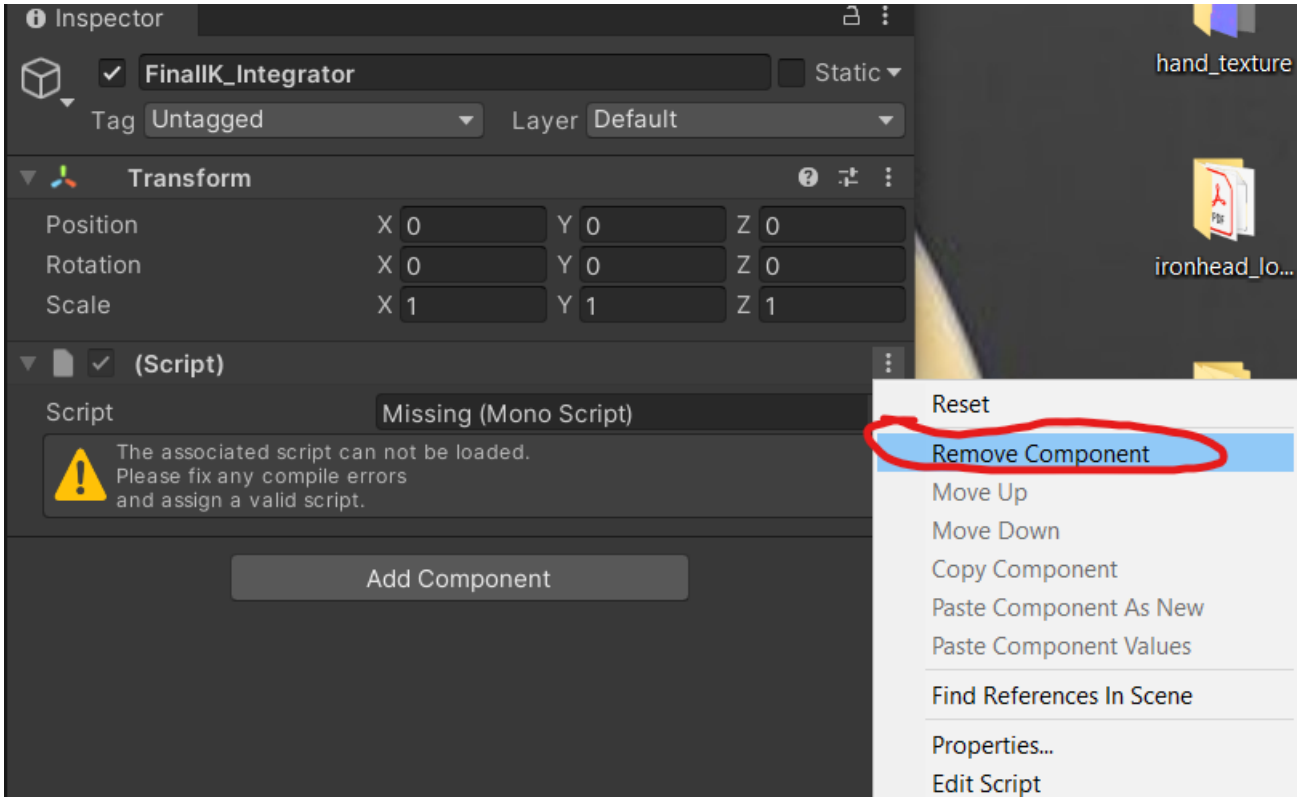
5. Then, go to IRONHEAD Games > Integrations > Final IK Integration > Scriptable Objects > FinalIKSolutionHolder and make sure that *Enable FinalIK* toggle is checked.



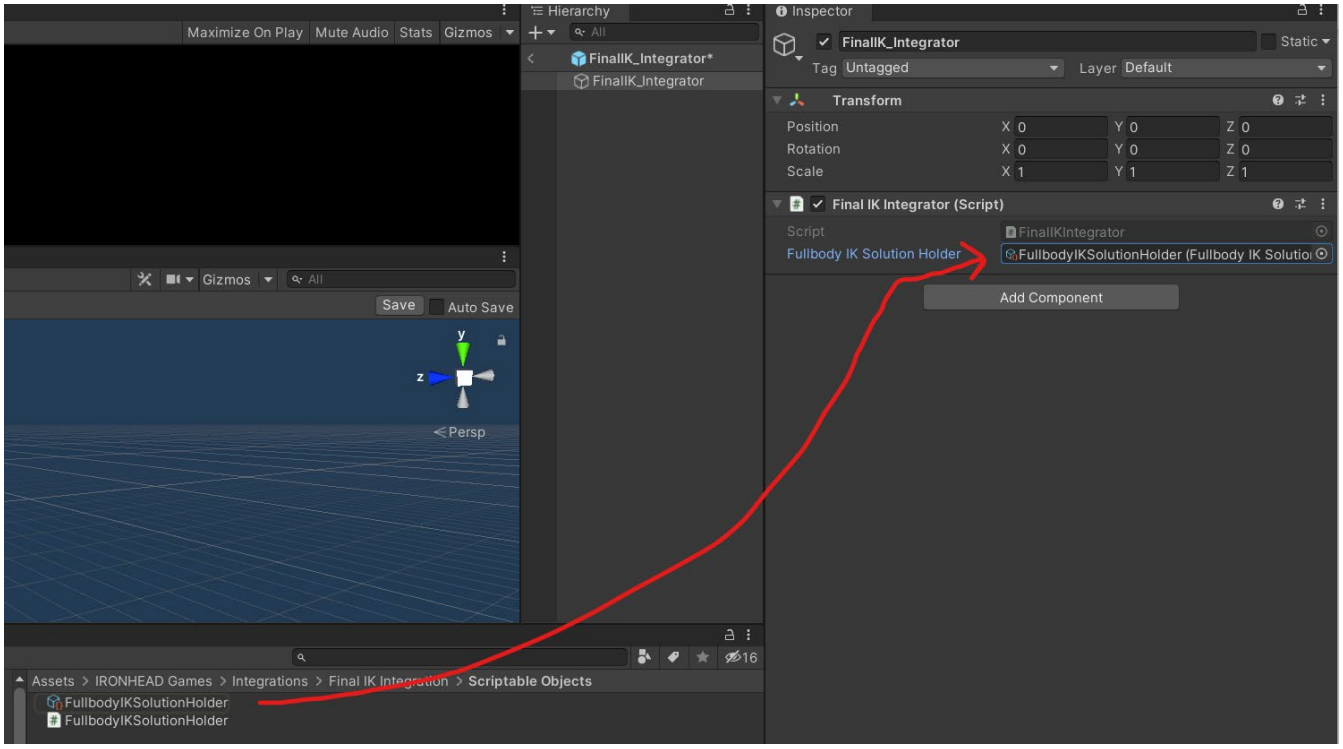
6. Next, open FinalIK_Integrator prefab from IRONHEAD Games> Integrations > Final IK > Final IK Integration > Prefabs. You may see this error=>



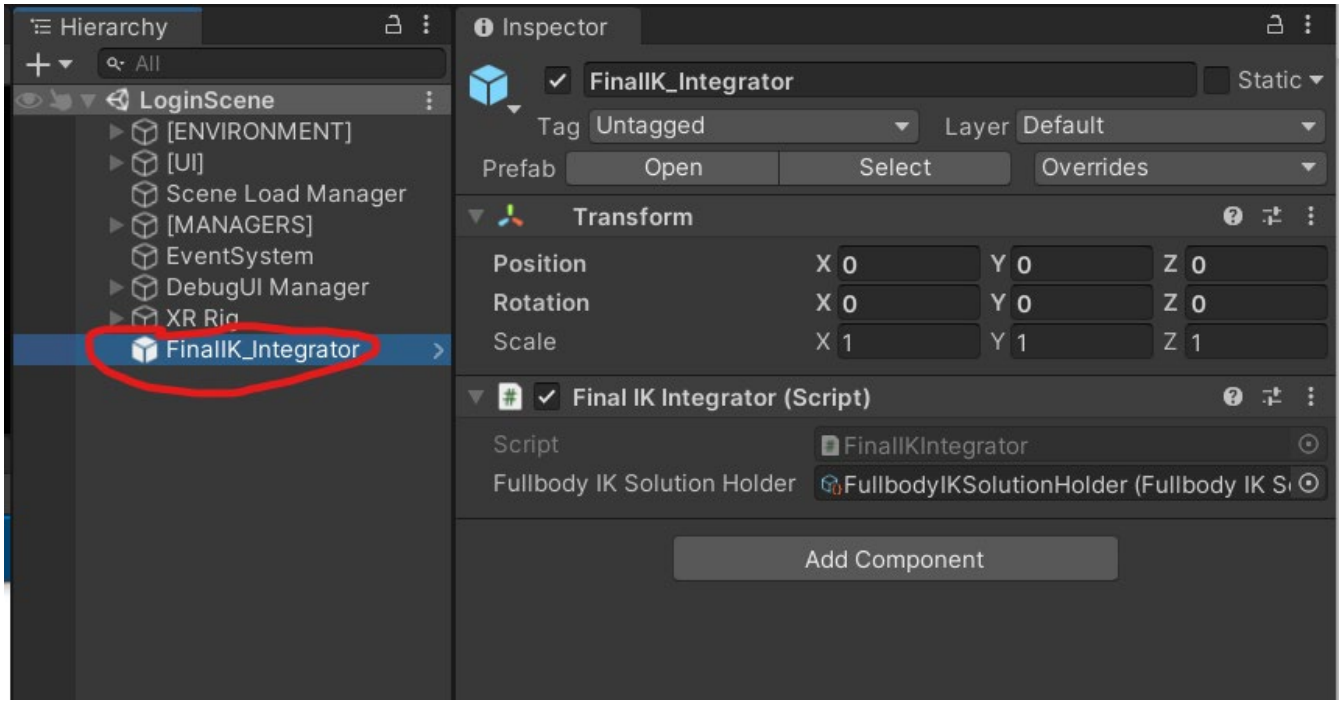
This is normal and we will fix it. First, open the prefab and remove the existing script.



Then, add Final IK Integrator script and drag&drop FinalIKSolutionHolder scriptableobject into it. And save the Prefab.

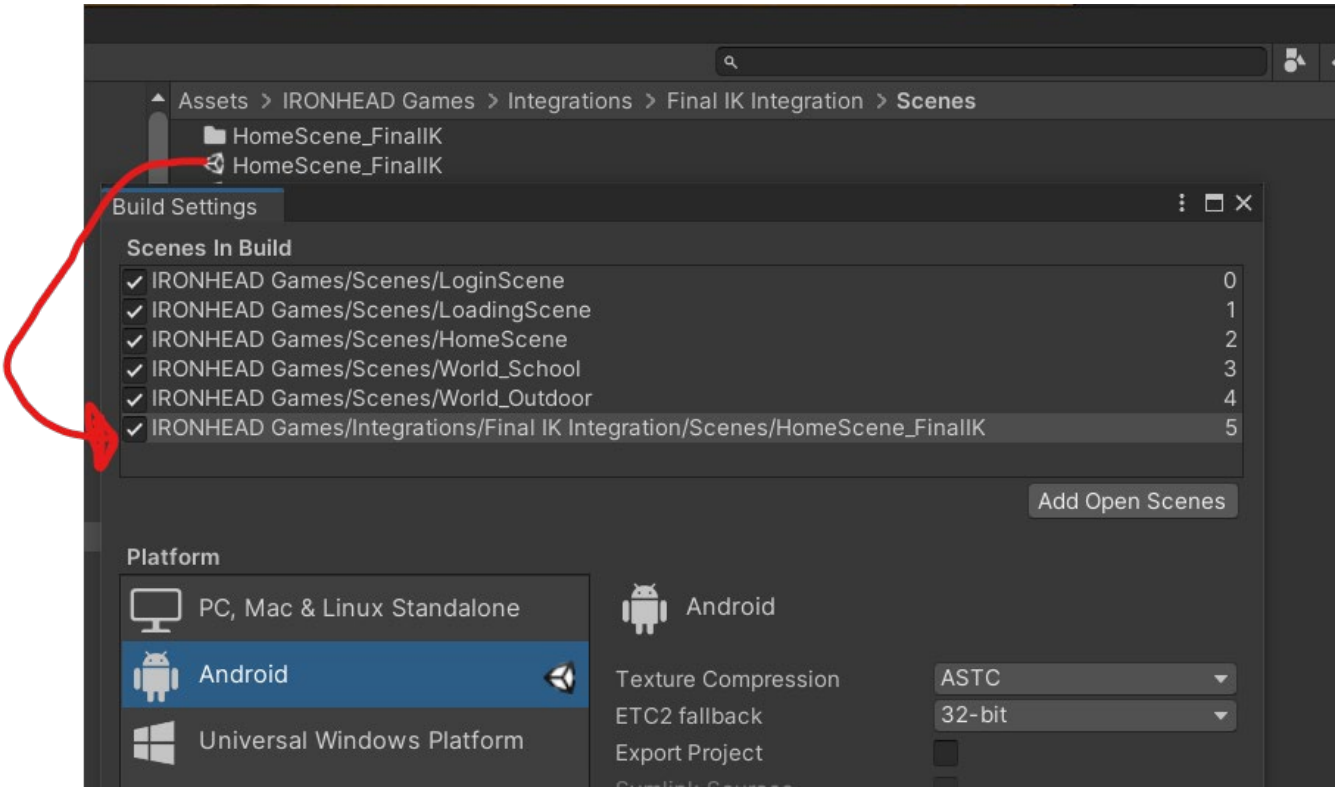


7. Now, open Login Scene. Drag and drop *FinalIK_Integrator* prefab from IRONHEAD Games> Integrations > Final IK > Final IK Integration > Prefabs. Also, save the scene.

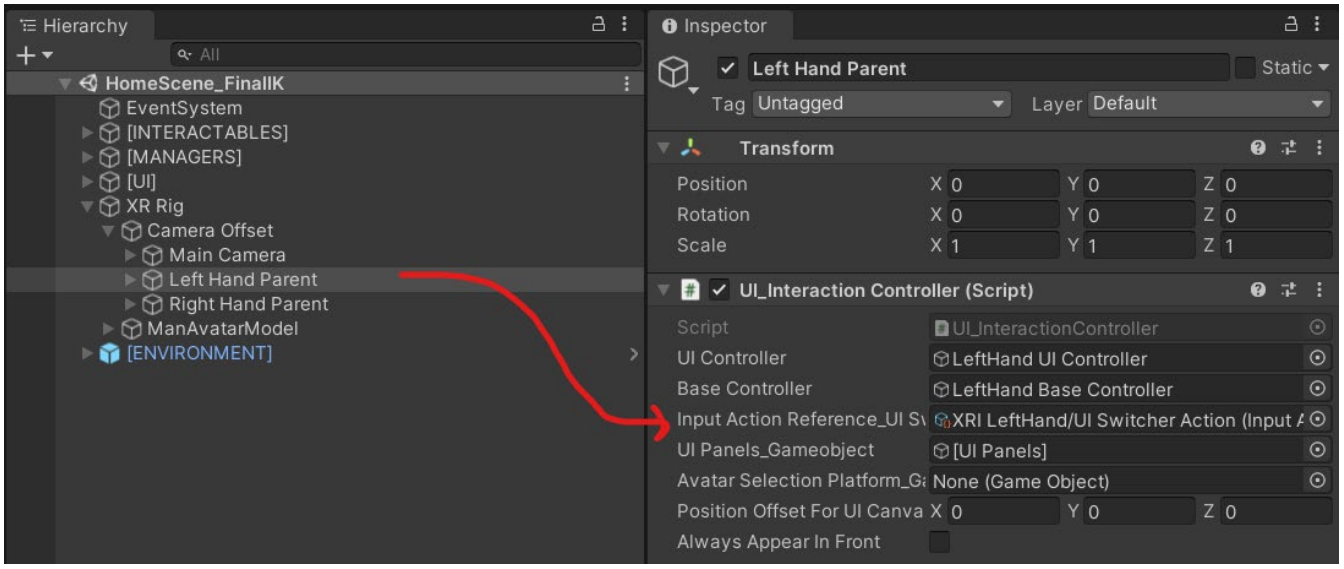


This will make sure that Final IK will be used for Full Body VR System across the scenes.

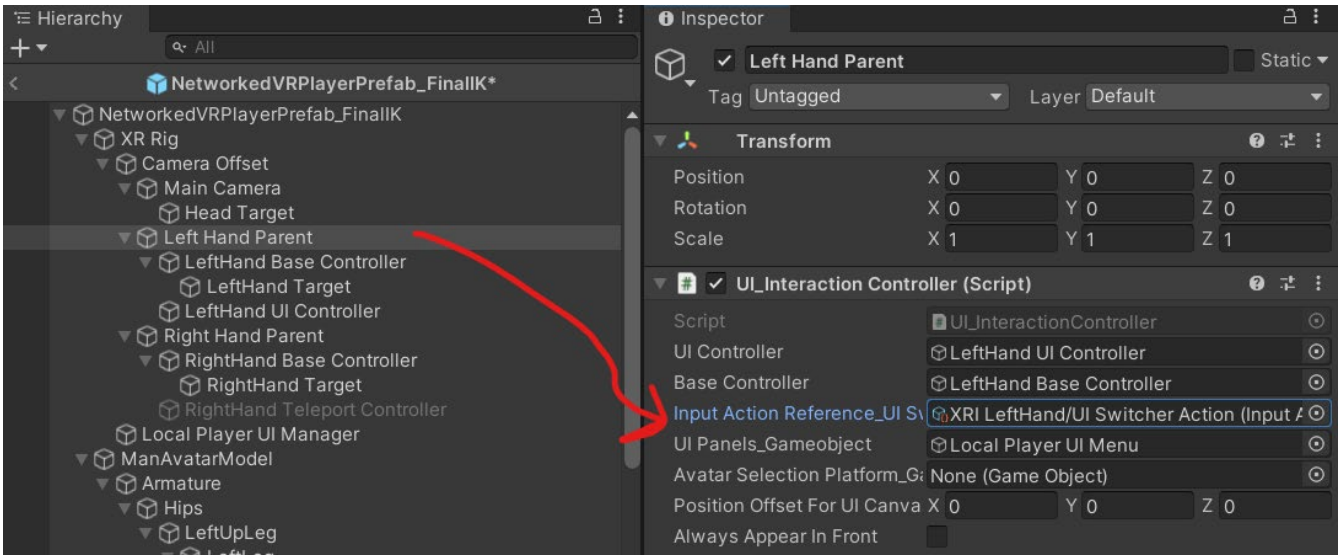
8. Next, click on File > Build Settings. Add HomeScene_FinalIK to the Scenes in Build.



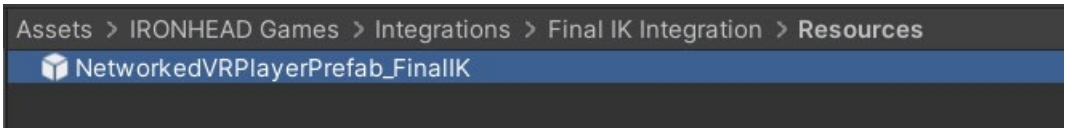
9. Also, open FinalIK Integration > Scenes > HomeScene_FinalIK. And click on Left Hand Parent gameobject under XR Rig. Set the Input Action Reference as the LeftHand UI Switcher Action. And save the scene.



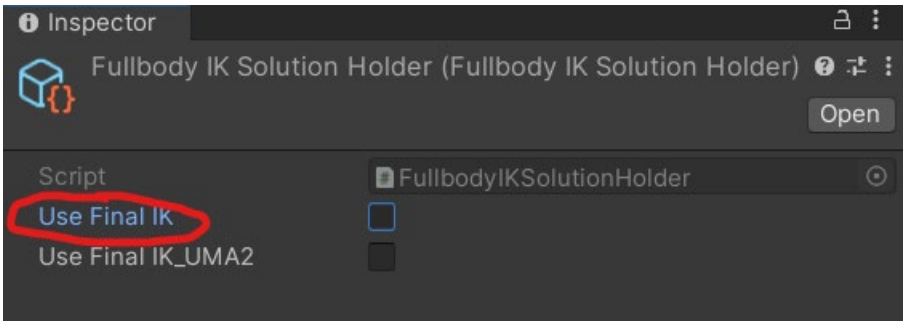
Next, do the same for the NetworkedVRPlayerPrefab_FinalIK gameobject under Final IK Integration > Resources folder and save the prefab.



10. That is it. Now, when you build the project, HomeScene_FinalIK scene will be loaded, and this scene includes the Final IK implementation for the VR Player. Also, NetworkedVRPlayerPrefab_FinalIK prefab will be instantiated when joining the networked scenes. For more info, you can check Project Documentation section.

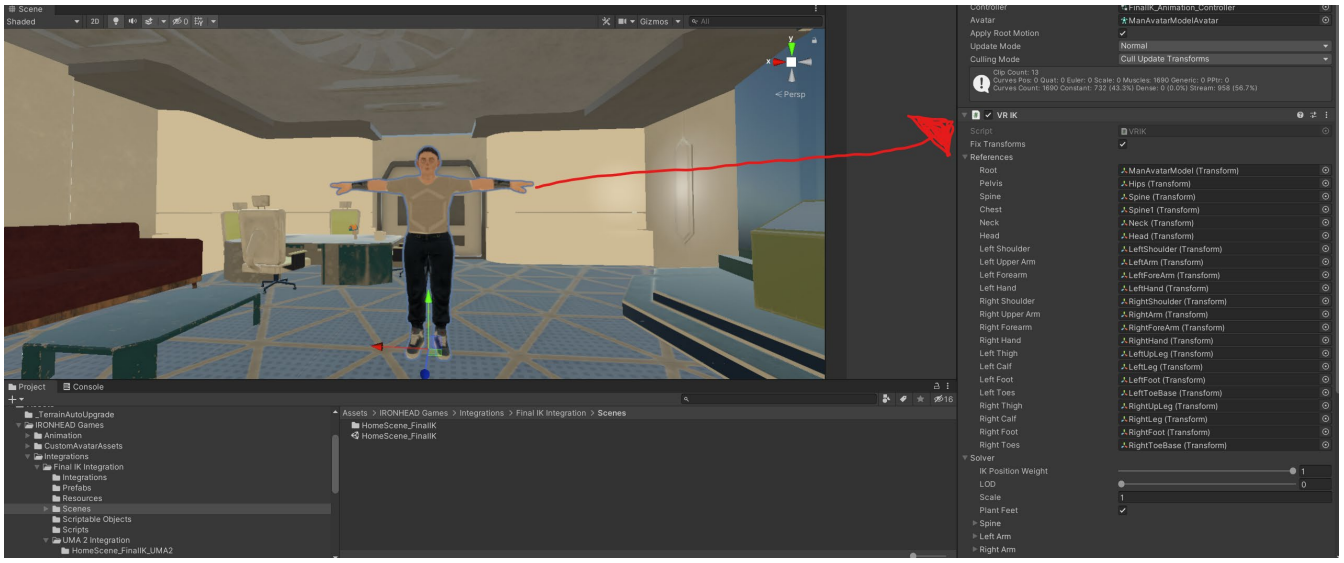


11. If you want to disable Final IK and continue to use the built-in VR Avatars (not fullbody), uncheck Use Final IK toggle inside IRONHEAD Games> Integrations > FinalIK Integration > Scriptable Objects > FinalIKSolutionHolder.



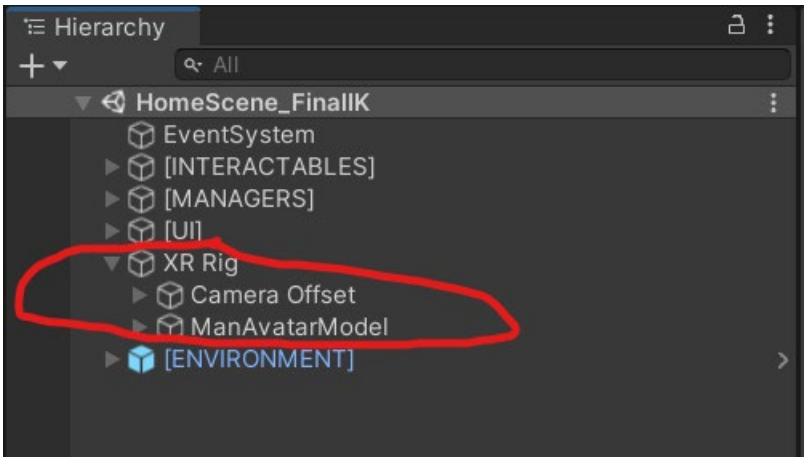
Final IK Integration Documentation

Thanks to Final IK, we have an exceptionally good Fullbody VR system out-of-box. If you open the HomeScene_FinalIK, you can see that we have VRIK script attached to the main Avatar gameobject.

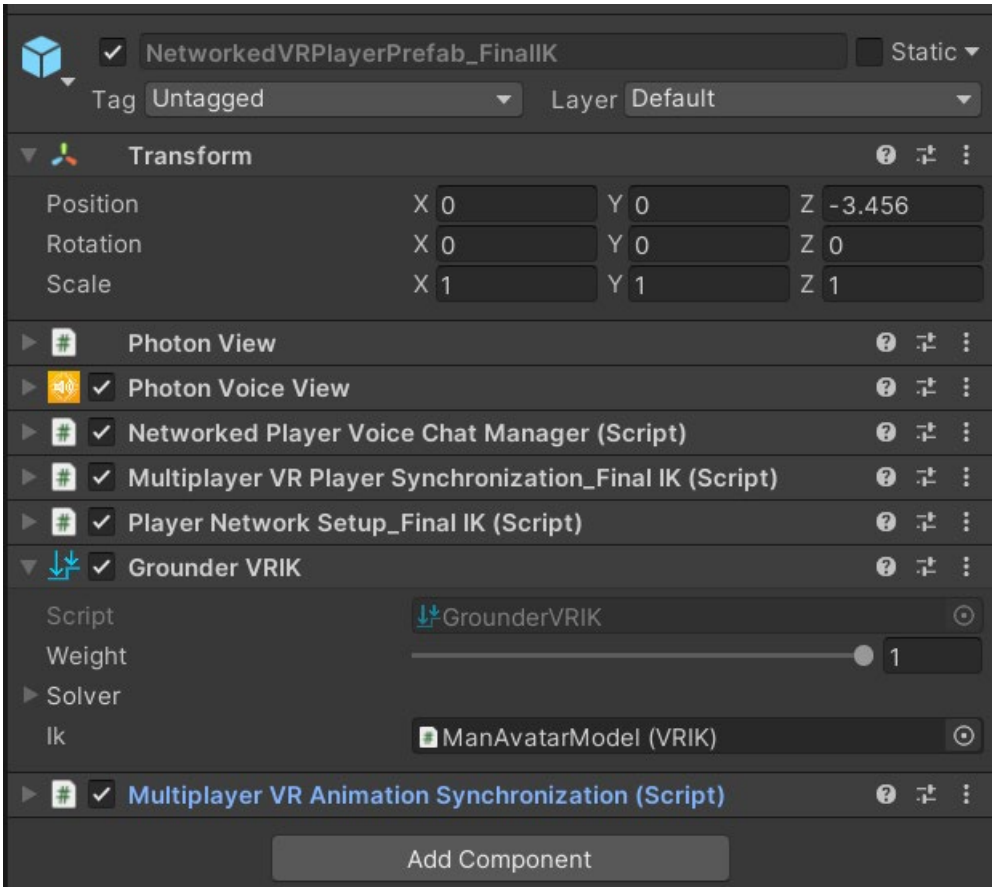


This is the main script that drives the fullbody VR system. If you want to add your own Avatar, simply add this script to it and you are almost there.

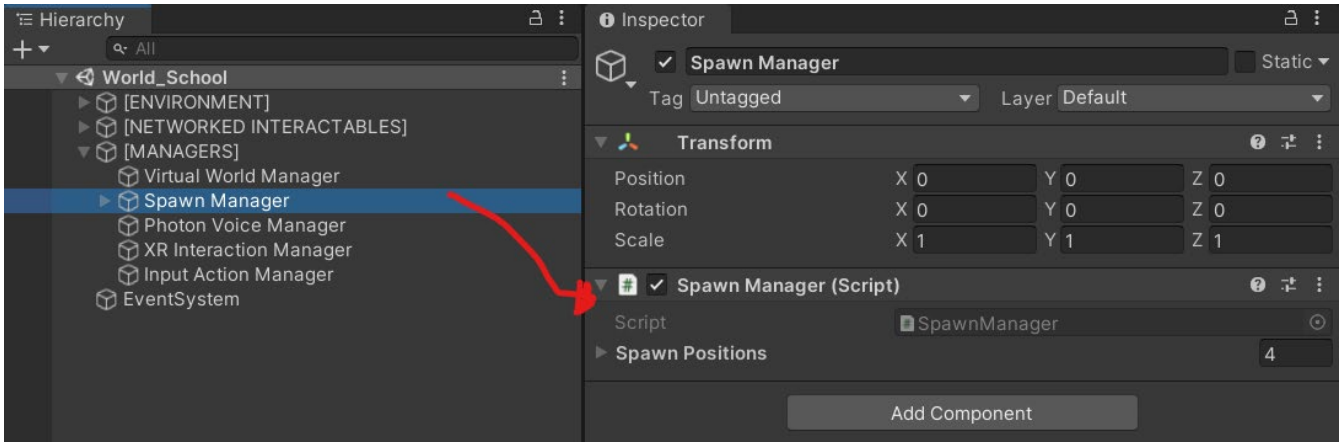
In the HomeScene_FinalIK, we have this Local XR Rig that has the fullbody VR system.

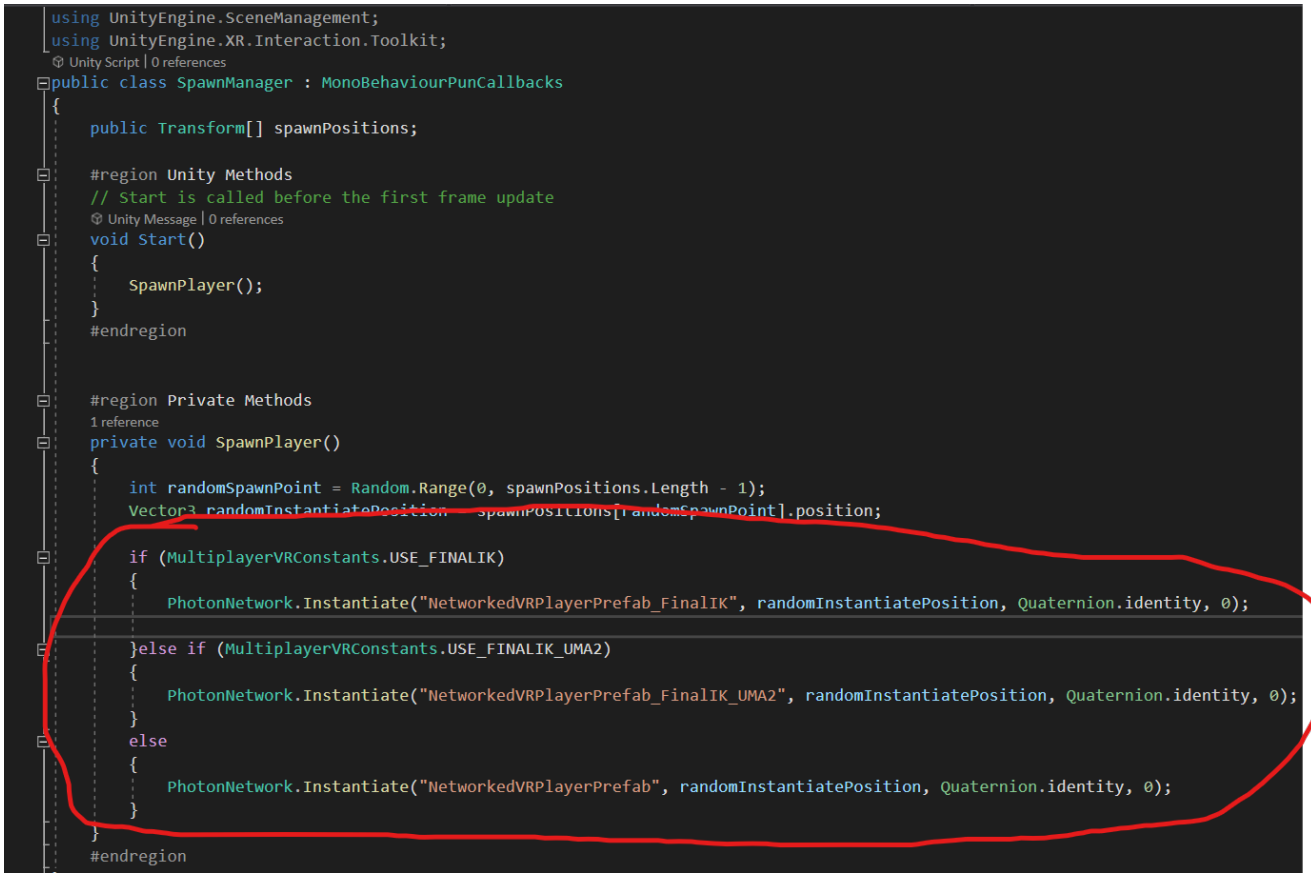


But in the networked scenes, we have the NetworkedVRPlayerPrefab_FinalIK prefab in the Resources folder. This prefab is similar to the local XR Rig but has other scripts for the networking.



If you open one of the Networked scenes such as World_School from IRONHEAD Games > Scenes, you will see that we have the Spawn Manager script. This script instantiates the corresponding networked player prefab based on the integration you choose.



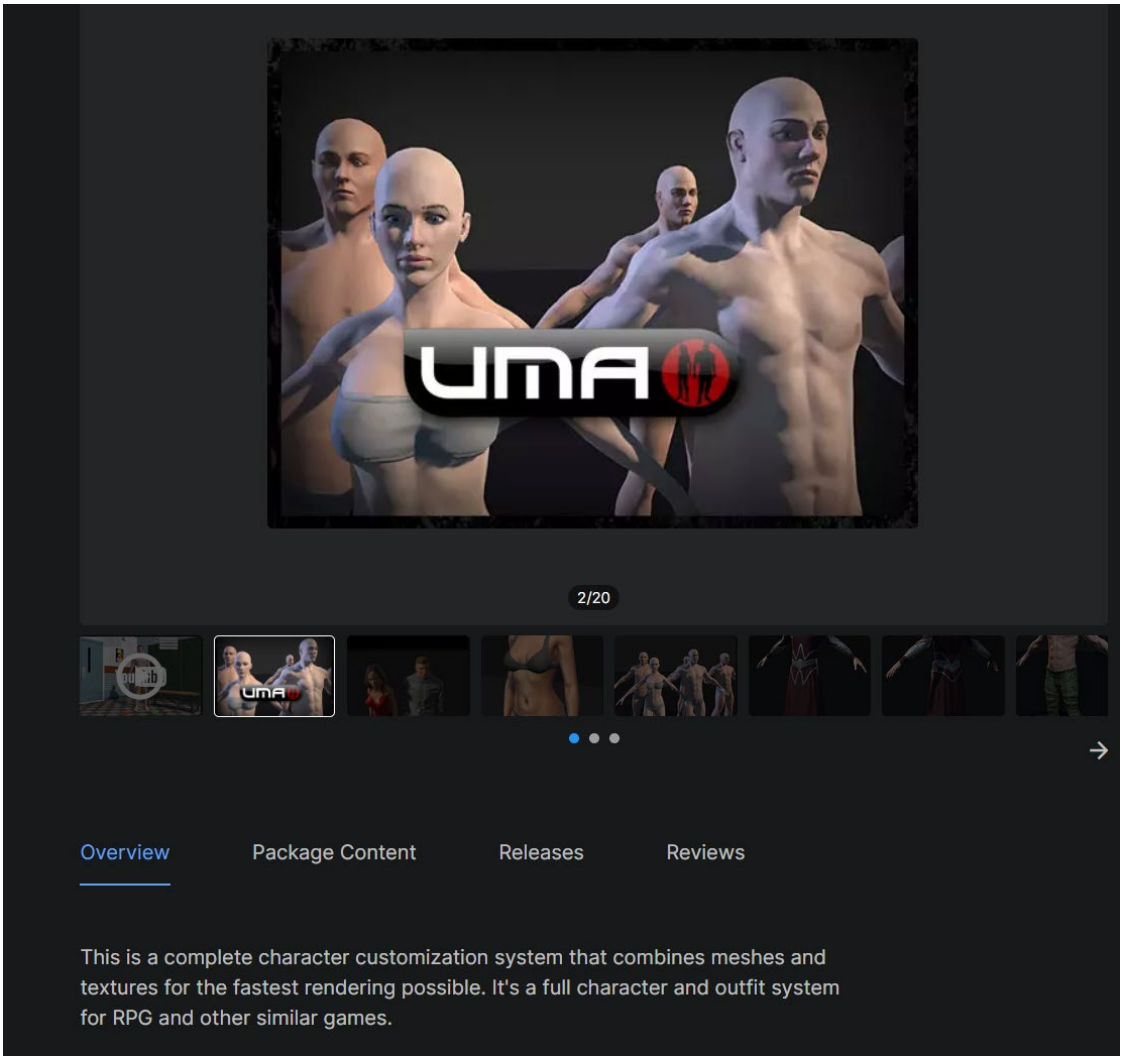


```
using UnityEngine.SceneManagement;
using UnityEngine.XR.Interaction.Toolkit;
Unity Script | 0 references
public class SpawnManager : MonoBehaviourPunCallbacks
{
    public Transform[] spawnPositions;

    #region Unity Methods
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        SpawnPlayer();
    }
    #endregion


    #region Private Methods
    1 reference
    private void SpawnPlayer()
    {
        int randomSpawnPoint = Random.Range(0, spawnPositions.Length - 1);
        Vector3 randomInstantiatePosition = spawnPositions[randomSpawnPoint].position;

        if (MultiplayerVRConstants.USE_FINALIK)
        {
            PhotonNetwork.Instantiate("NetworkedVRPlayerPrefab_FinalIK", randomInstantiatePosition, Quaternion.identity, 0);

        }else if (MultiplayerVRConstants.USE_FINALIK_UMA2)
        {
            PhotonNetwork.Instantiate("NetworkedVRPlayerPrefab_FinalIK_UMA2", randomInstantiatePosition, Quaternion.identity, 0);
        }
        else
        {
            PhotonNetwork.Instantiate("NetworkedVRPlayerPrefab", randomInstantiatePosition, Quaternion.identity, 0);
        }
    }
    #endregion
```

So, when customizing the asset and avatars, make sure to change the NetworkedVRPlayerPrefab_FinalIK prefab.

UMA 2- Unity Multipurpose Avatar Integration

UMA 2 is a 3rd party avatar system that allow developers creates high quality characters without having to design in 3D.

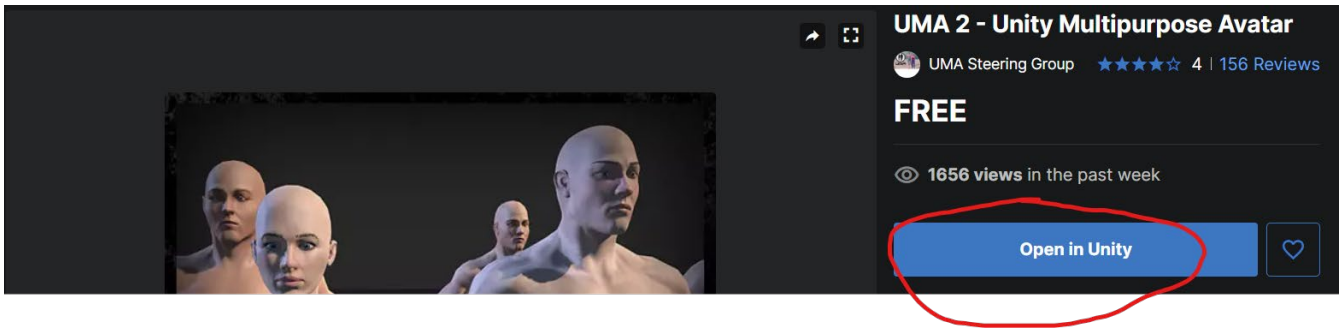


It offers character customization with a small learning curve withing the Unity Editor. So, UMA 2 is a perfect asset to create custom VR Avatars.

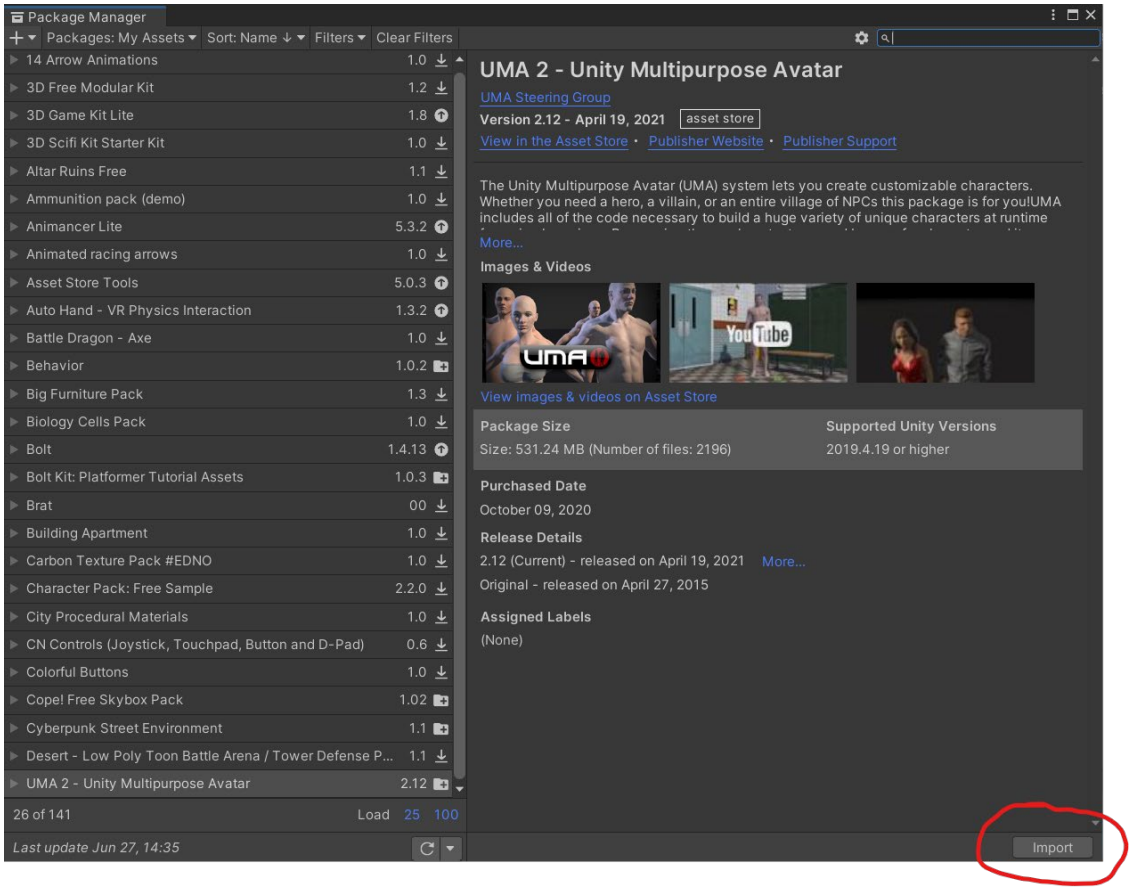
UMA 2 Integration is built on top of Final IK integration. So, first make sure that you have the Final IK Integration setup and ready. Below, you can find the steps to setup the UMA 2 integration.

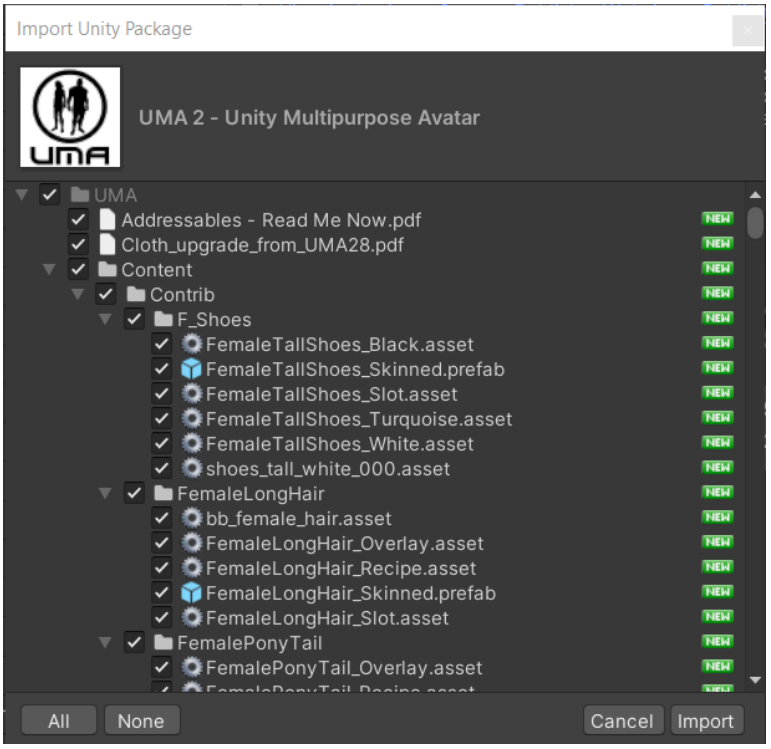
How to set up UMA 2 Integration

1. First, open Unity Asset Store and search for UMA 2. It is a Free asset so Add to your Assets and click on Open in Unity.

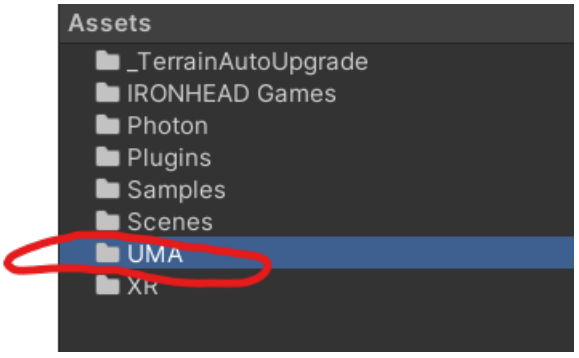


2. It will open the Package Manager in Unity Editor. Download and import the Asset. It may take while to finish.

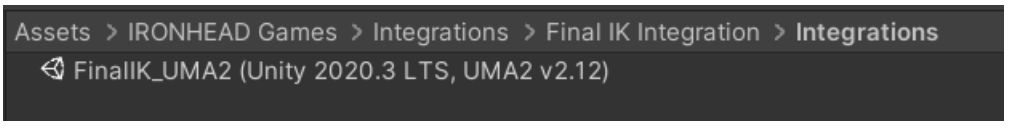




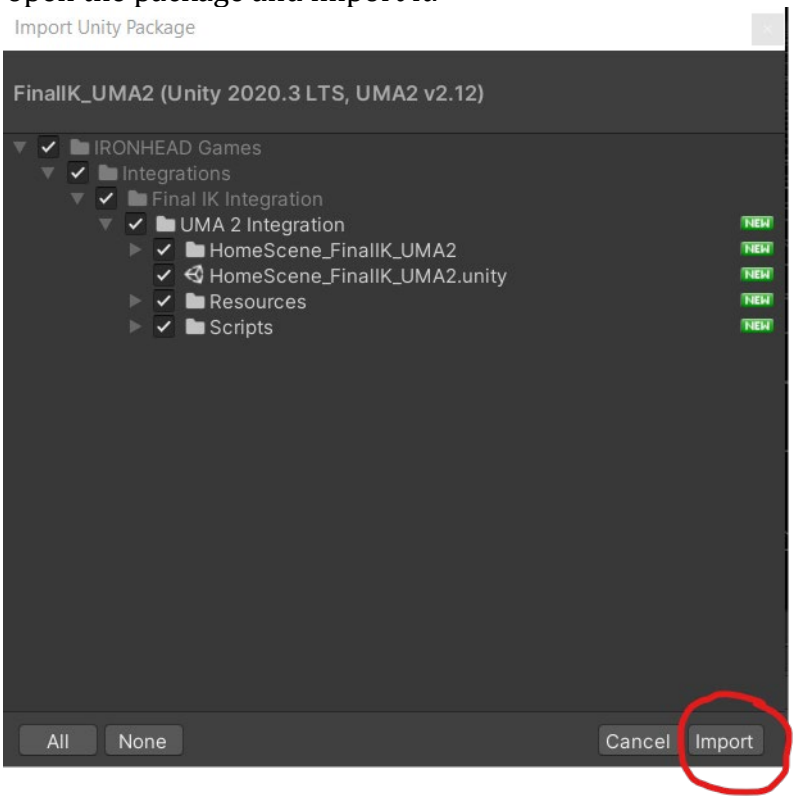
3. After that, you should a new folder called UMA.



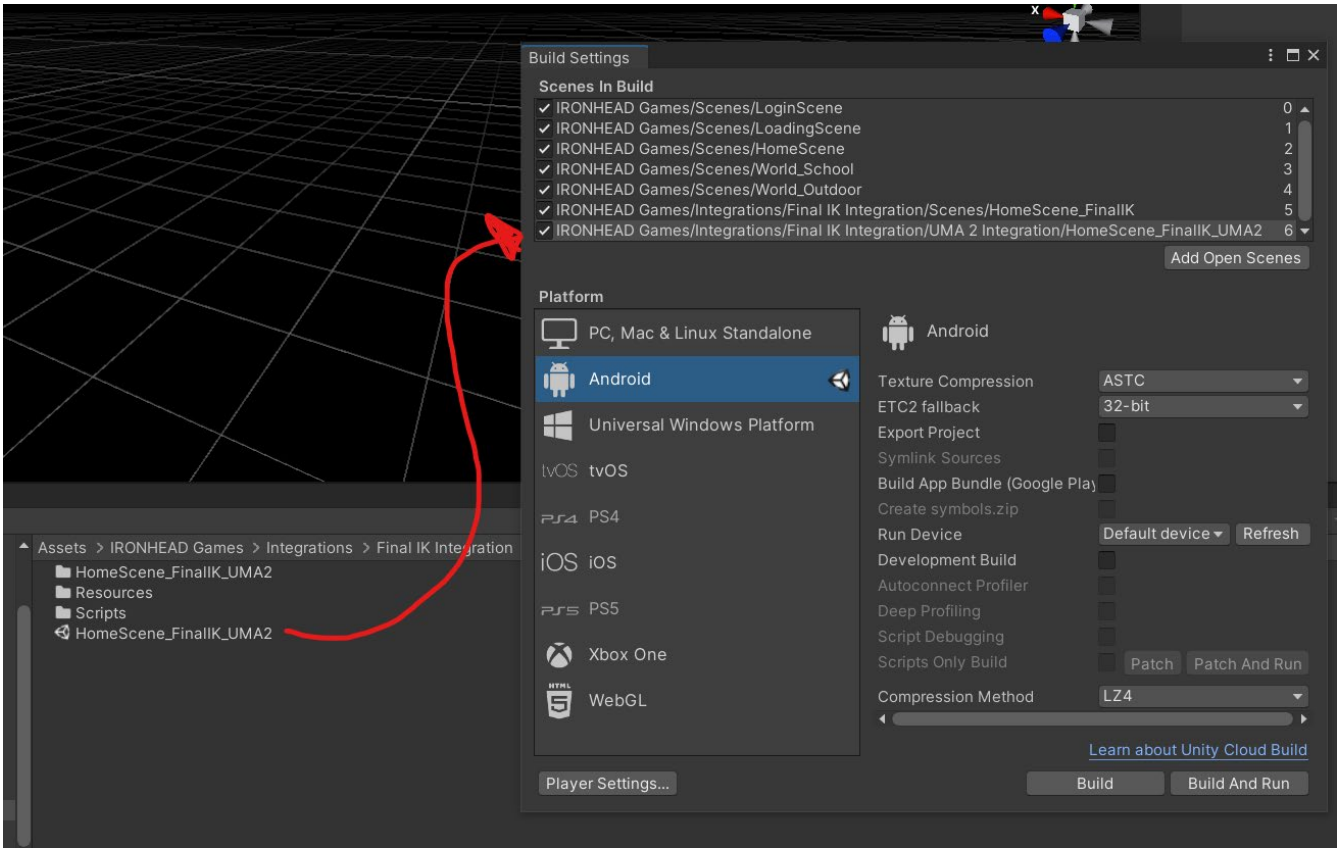
4. Next, go to IRONHEAD Games > Integrations > Final IK Integration > Integrations folder. You should the UMA 2 integration package.



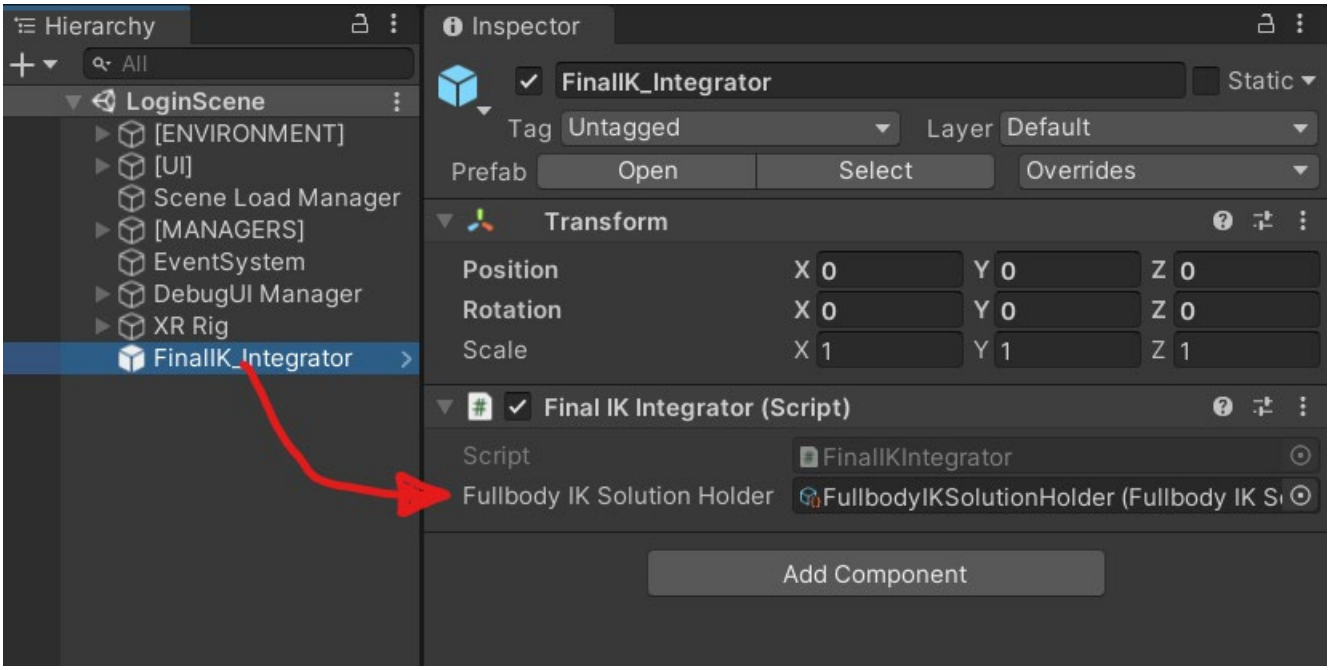
Double-click to open the package and import it.



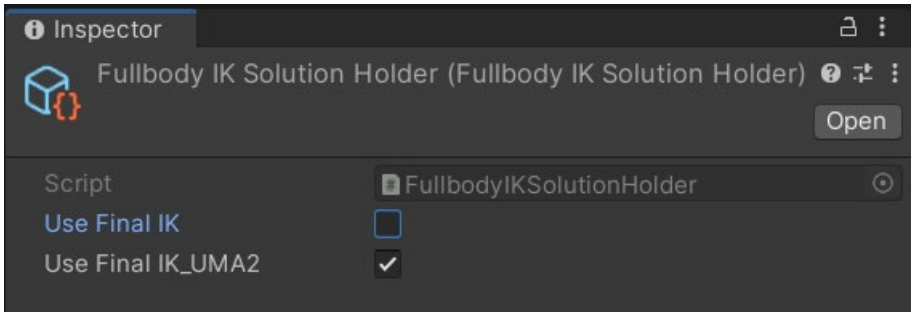
5. Next, open File > Build Settings and add HomeScene_FinalIK_UMA2 scene to the Scenes in Build.



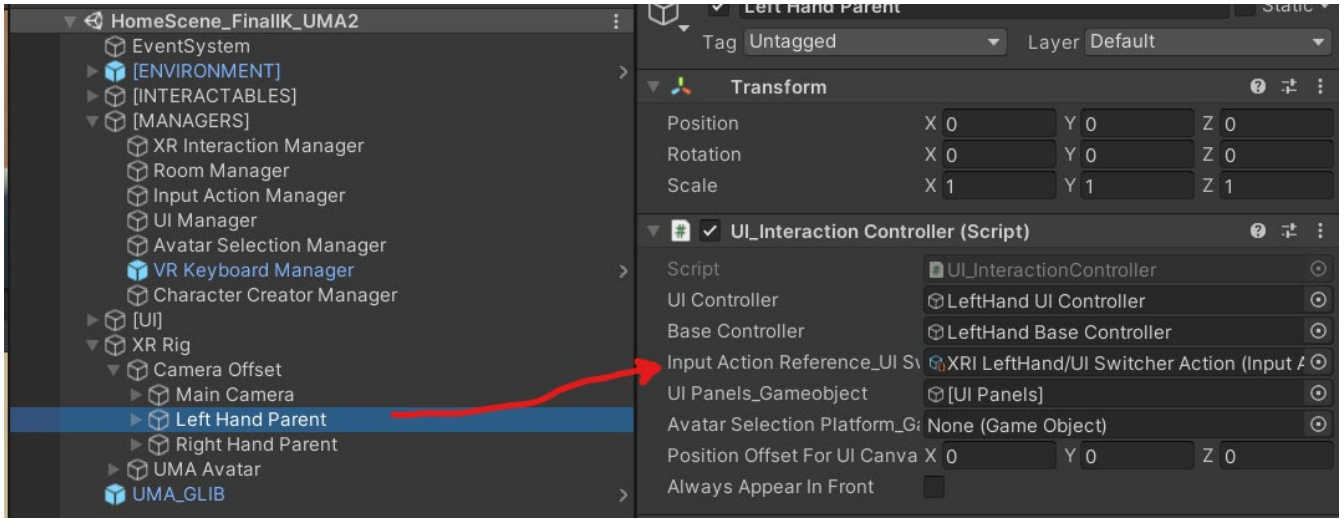
6. Then, open LoginScene and click on FullbodyIKSolutionHolder scriptable object on FinalIK Integrator.



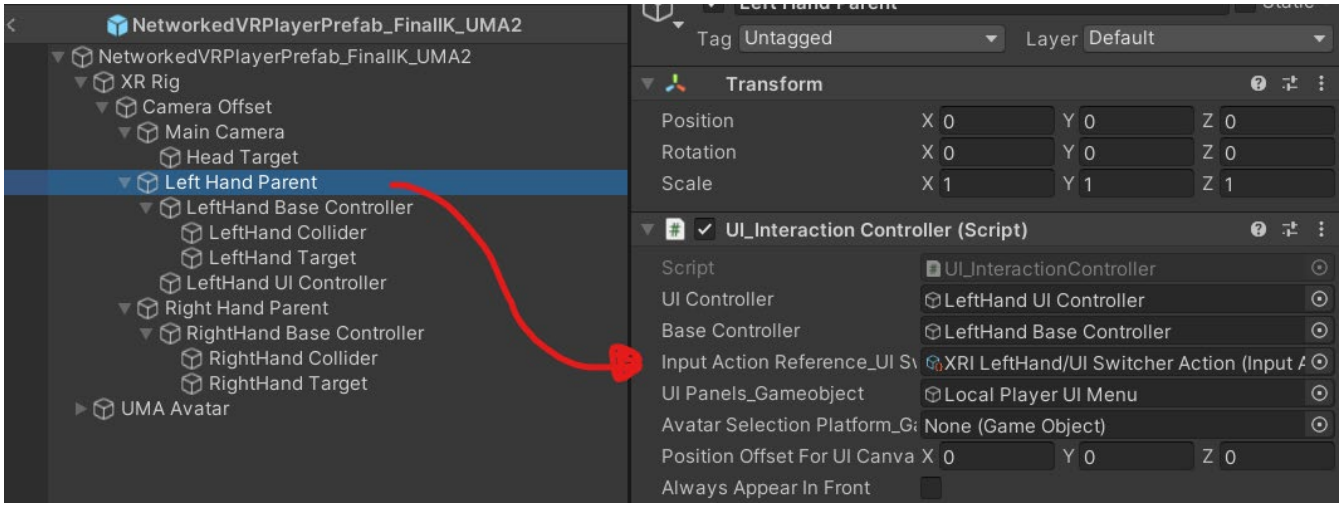
7. Enable Use *FinalIK_UMA2* toggle and disable Use *Final IK* toggle.



8. Also, open FinalIK Integration > UMA2 Integration > HomeScene_FinalIK_UMA2. And click on Left Hand Parent gameobject under XR Rig. Set the Input Action Reference as the LeftHand UI Switcher Action. And save the scene.



Next, do the same for the NetworkedVRPlayerPrefab_FinalIK_UMA2 gameobject under Final IK Integration > Resources folder and save the prefab.

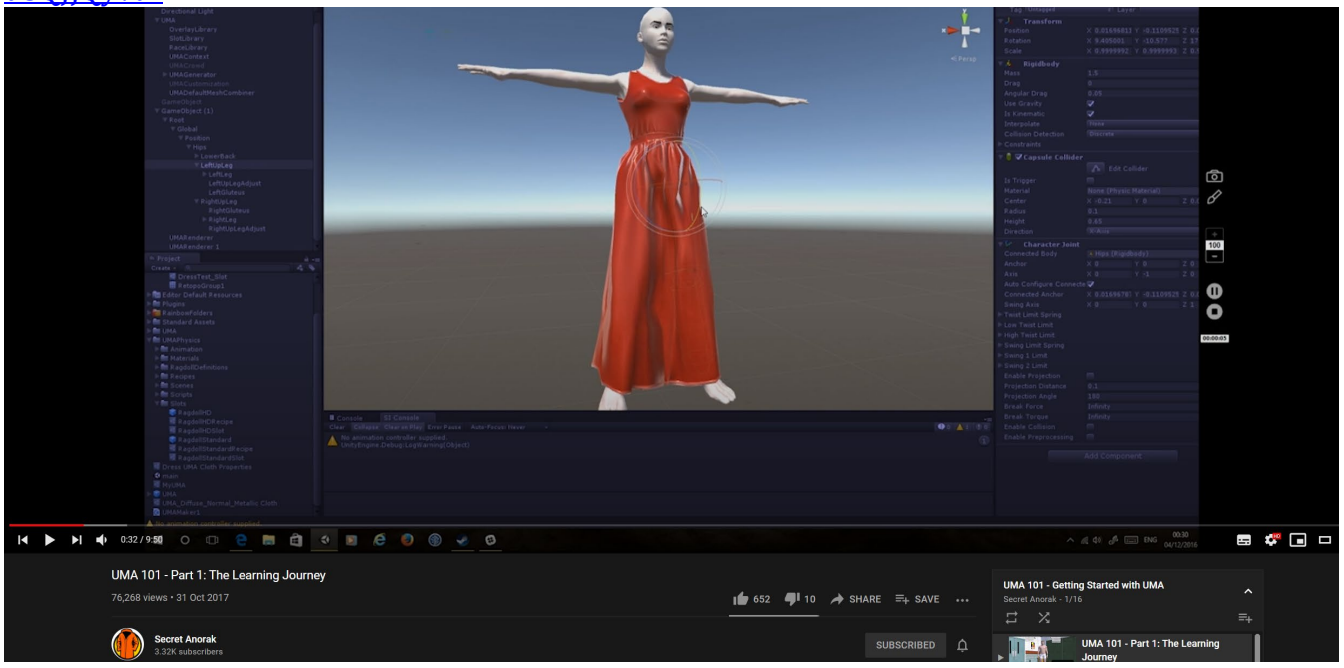


9. Done! You can open LoginScene, build the project to your Quest/2 and enjoy UMA 2 integration.

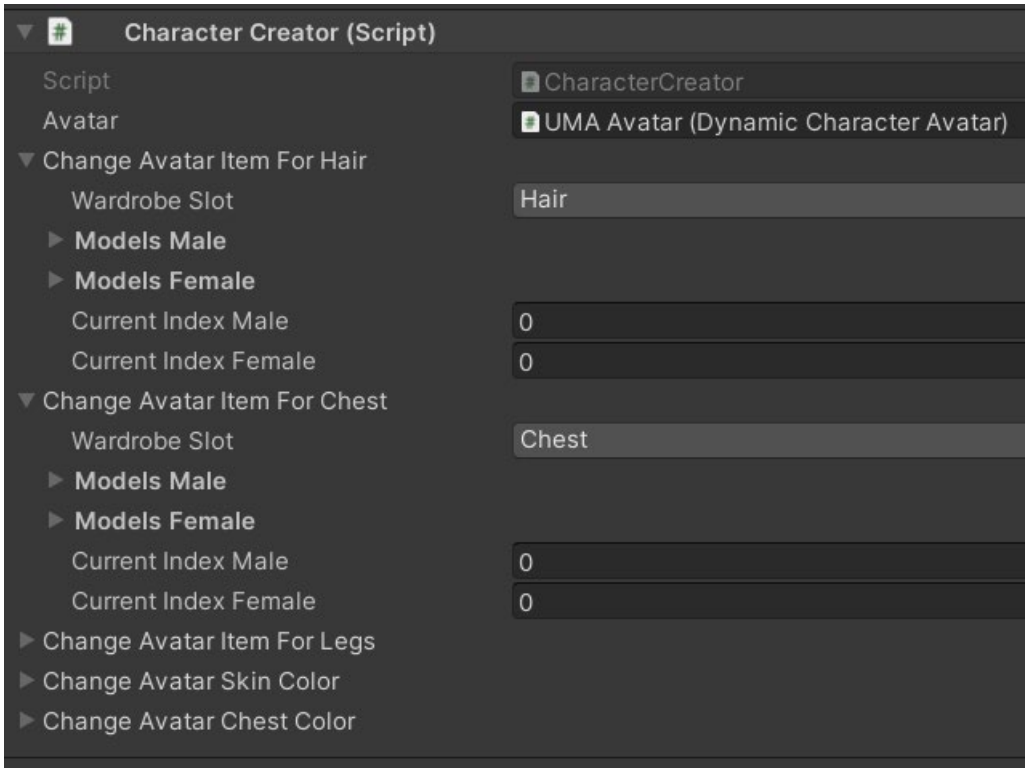
UMA 2 Integration Documentation

UMA 2 allows indie developers/studios to easily create high quality, customizable 3D Avatars. Using UMA 2 in Unity projects requires you to learn the basics. At first, it may seem hard to understand but once you get the fundamentals, you will see that it will save you/ your team a lot of time. Even more, using UMA 2, you can create your own Character Creator for Unity your game.

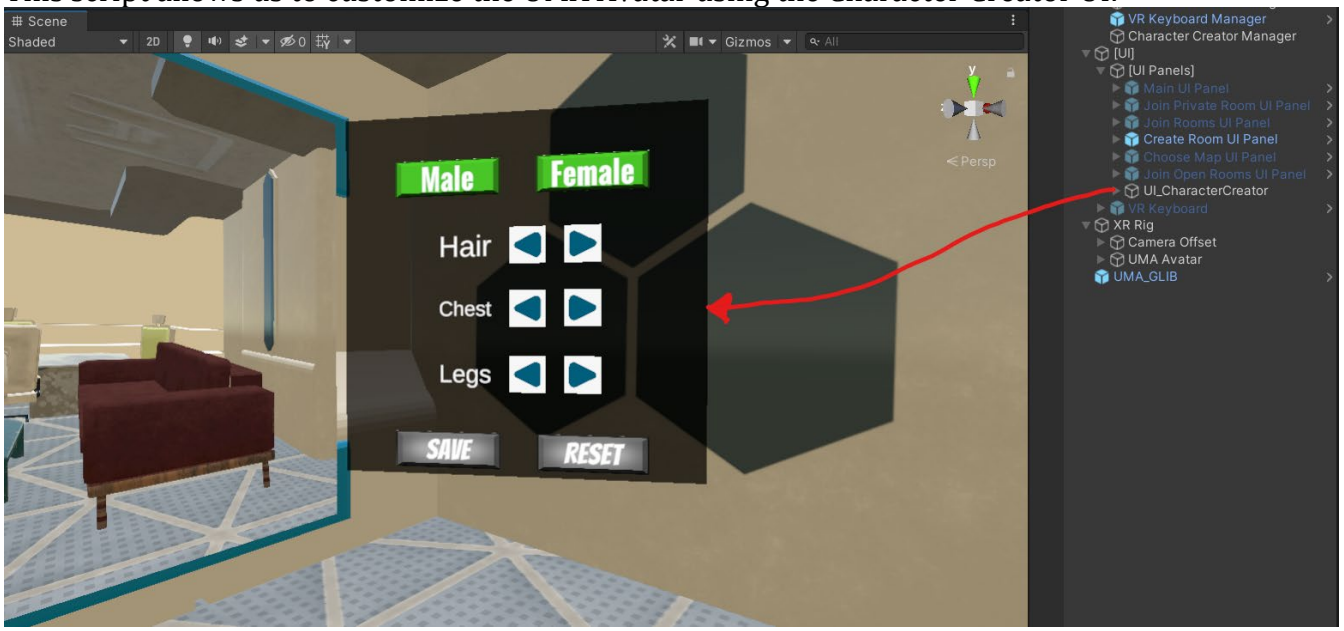
So, before adapting the asset's UMA 2 integration to your needs, I strongly advise you to watch this YouTube tutorial playlist:
https://www.youtube.com/watch?v=qzJeMWNEhWM&list=PLkDHFObfS19zFVfbrfB14P-u5QJJQyvtP



Character Creator Manager

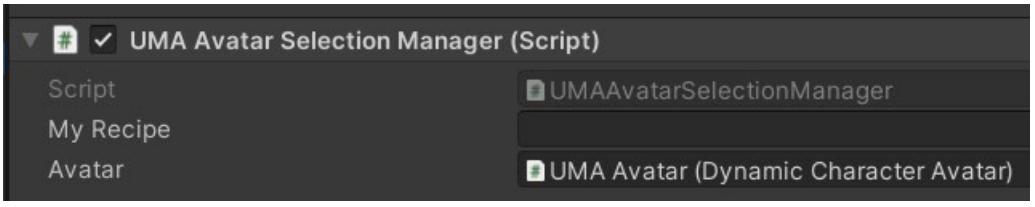


This script allows us to customize the UMA Avatar using the Character Creator UI:



Currently, we have only a few customization options such as Gender, Hair, Chest and Legs. But more can be added by further working with UMA 2.

Avatar Selection Manager



This script saves and loads the avatar selection data in Json format in the local device. So, when you create your own UMA Avatar, its DNA data is saved to the device and sent to the networked players when you join to the networked rooms.

UMA_GLIB

This gameobject is a Prefab from UMA 2 asset and we need to have it in any scene we have UMA Characters. It allows us to render and customize UMA Avatars.

After watching the basic UMA 2 tutorials, open the HomeScene_FinalIK_UMA2 scene. Here, these are game objects related to UMA 2 integration: