THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

ENGINEERING

GETTING STARTED WITH THE SDL UNITY OPENXR UX BASE FOR QUEST, RIFT AND VIVE.

A Smart Digital Lab Guide

Purpose

This guide details how to build an interactive VR environment using OpenXR which can be used for any engineering (or other XR) project.

Contact
sdl@auckland.ac.nz

Getting started with the SDL OpenXR UX base

Summary

This document has been prepared as a guide to staff and students at the Civil and Environmental Engineering Smart Digital Lab at the University of Auckland, New Zealand. However, the procedures and tips contained herein are generally applicable to anyone interested in getting started in programming Virtual Environments. Note that with these technologies, any such document is time-limited given the way the technology progresses so quickly. Therefore, please check the date of the latest revision, and feel free to send me ideas, updates, notification of errors and feedback to sdl@auckland.ac.nz.

Version

1.5	211019	Code version 0.2.5 – Desktop mode added
1.4	211014	Code version 0.2.4 – Dynamic quality added
1.3	211006	Tweaked to reflect code tidy-ups and changes – and 0.2.3.
1.2	210929	Version 0.2.2 on GITHub
1.1	210916	Matches version 0.2.1 of the code on GITHub.
1.0	210901	Document created by Dr. Roy C. Davies (Senior Technician at SDL).

How to use this document

This document is intended to help you set up an interactive Virtual Environment that makes it easy to include interaction, movement and UX without having to worry too much about the low level complexities of how all these work. Further, the UX has been designed to be as intuitive as possible. It is assumed that you have a working knowledge of Unity and have completed the other SDL guides on getting started with Unity and GIThub (at the very least). The OpenXR UX Base works with Oculus Quest 1 and 2, Oculus Rift (and Quest in Link mode), and VR devices supported by Steam VR.

Table of Contents

Table of Figures

Introduction

Unity is a powerful tool for creating interactive Virtual Environments for use with all manner of devices, including VR headsets. However, for a beginner to Unity, the number of choices and possibilities can be overwhelming, with oftentimes many ways off achieving the same outcomes, though some may be more future-proof than others, and some may be more intuitive to users than others. Further, developing a Virtual Environment that will work on (almost) any VR headset, and many AR headsets as well, is a daunting task.

This guide is intended to leap-frog you into getting stuff working in XR in Unity with many of the core elements you will need already designed, which you can then put together and reuse as required. It is not just a base project, but also contains an extensible framework for building new elements.

To get started, there is a Unity Project associated with this document that you will first need to get from GIThub.

Getting the OpenXR UX Base for Unity on GIThub

The OpenXR UX Base is available on GIThub at:

https://github.com/smart-digital-lab/openxr_ux_unity_base

You can either copy this whole project, modify the example scene and rename to create your own new project; or use the prefabs and packages added to your own fresh project. Below are step by step instructions to start a fresh project and add the additional OpenXR UX elements.

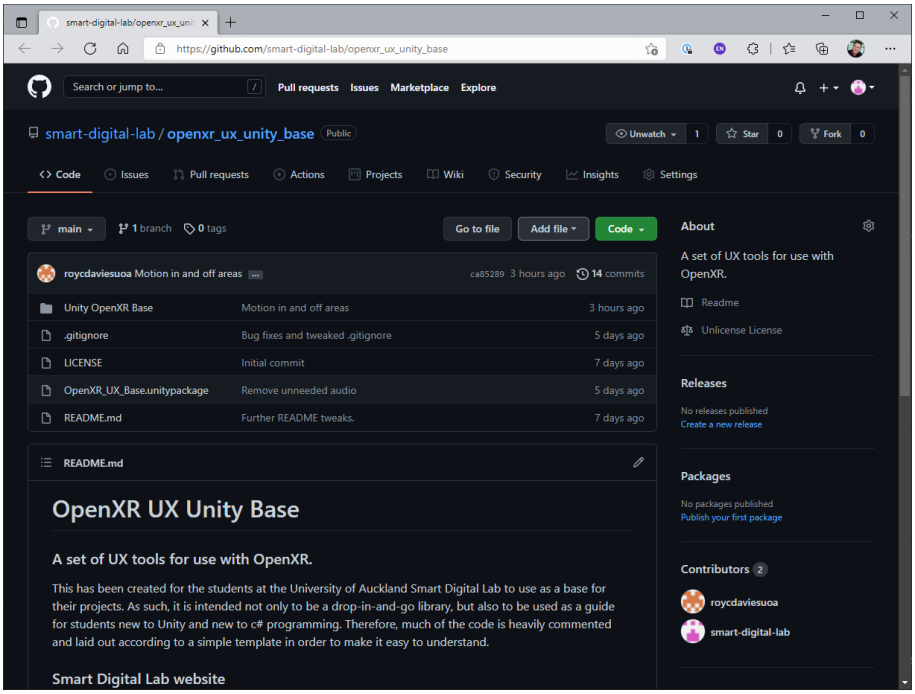


Figure 1 : Open XR UX Unity base on GITHub

Setting up your project and prerequisites for Oculus Quest

The OpenXR UX Base was built using Unity 2021.1.24f1. You should use the same or later version for your project, otherwise it may not work as intended.

Using UnityHub 3.0 or later, you can create a project specifically for VR which will include the OpenXR packages, though it will also work fine using the standard 3D setup as you can add the packages yourself. You may need to download the template first.

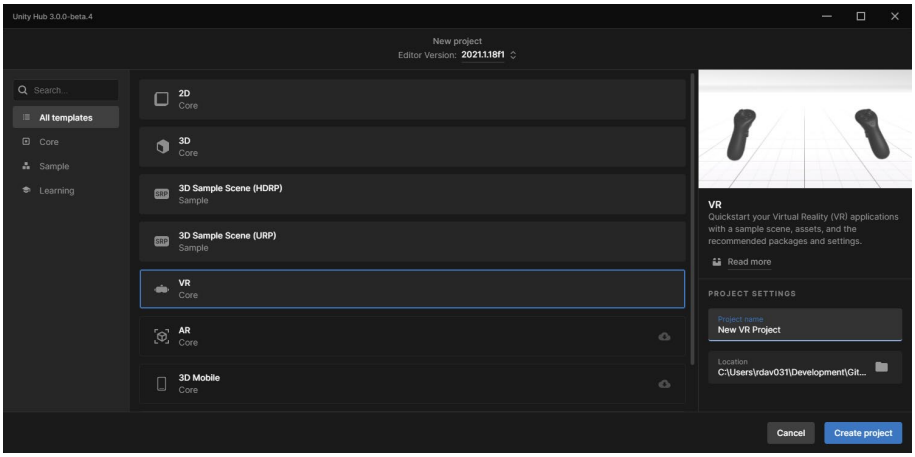


Figure 2 : Creating a new VR project

When the project starts, you have a clean project ideal for XR applications, however, there are some settings required before it will run on a VR headset which will depend on which platform you are building for. Whilst the OpenXR UX Libraries are common, you do need to set up Unity differently for each platform. To start with, we will work with the Oculus Quest (1 or 2). As per the getting started in Unity for VR on the Oculus Quest guide, and numerous other places around the internet, you must:

1) In Build Settings, Switch Platform to Android, and 'Add Open Scene' to make sure the right one will be built after saving the scene with some appropriate name.
2) Set the correct 'Run Device' to choose your VR headset.

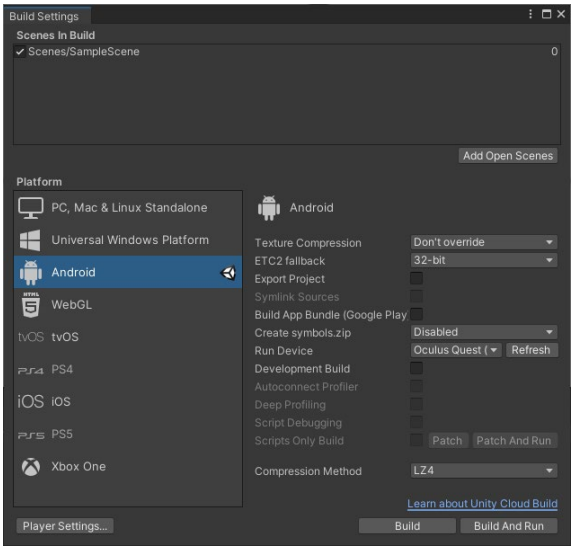


Figure 3 : Project Settings for Oculus Quest

3) In Package Manager, install the 'OpenXR Plugin' and uninstall the 'Oculus XR Plugin'. (Often, it will restart Unity at this stage). Make sure you are listing all the packages in the Unity Registry, not just the project. The OculusXR plugin is in the process of being phased out in preference to the OpenXR plugin and is only really needed if you want to do hand tracking. It may also ask you about the TextMeshPro package – install that too – see below.
4) In Project Manager, under 'XR Plug-in Management', first ensure 'OpenXR Plug-in Providers' is enabled, and then at the OpenXR level, make sure that you have OpenXR feature group 'Oculus Quest Support' ticked if you are using the Oculus Quest. It may have some issues that need fixing – choosing 'fix all' should clear that message.

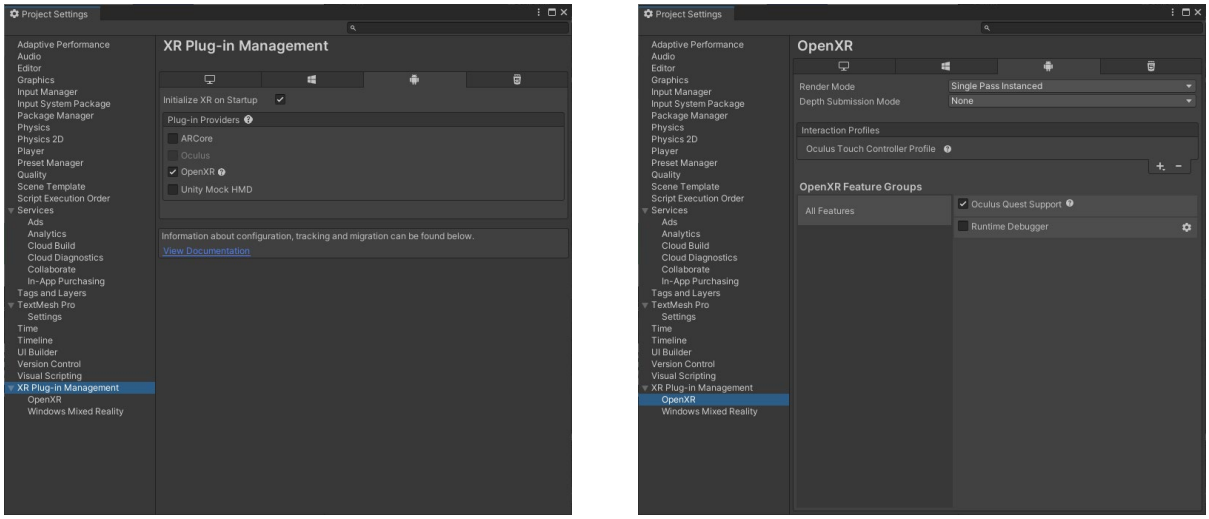


Figure 4 : The XR Plugin and OpenXR settings

5) This is also a good time to set the Company Name, the Product Name, the Version Number and the Identification/Package Name under the Player options in the Project Settings.

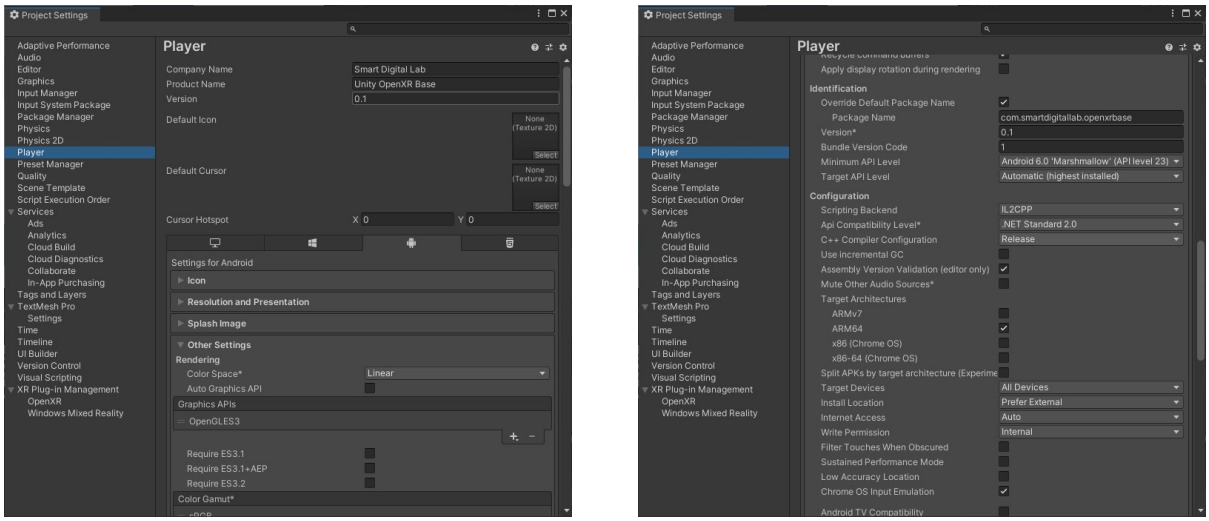


Figure 5 : Project Settings

Importing the SDL Unity OpenXR UX Base package

If you are building a fresh, new project and not using the pre-built scene and project you need to install the SDL Unity Open XR Base. To do this, use Import Package/Custom Package from the Assets menu, and choose the **OpenXR_UX_Base.unitypackage** file in the main folder of the project you downloaded from GITHub.

This will import all the assets required for getting started with an interactive XR Virtual Environment.

To test all is well, choose “Convert Main Camera To XR Rig With UX” under the GameObjects/OpenXR UX/ menu. Now, if you build and run the project, you should see an interactive Virtual Environment with some basic UI elements attached to the view and controllers. If you started with just a normal 3D project (not a VR one), the ground and background colours will be different.

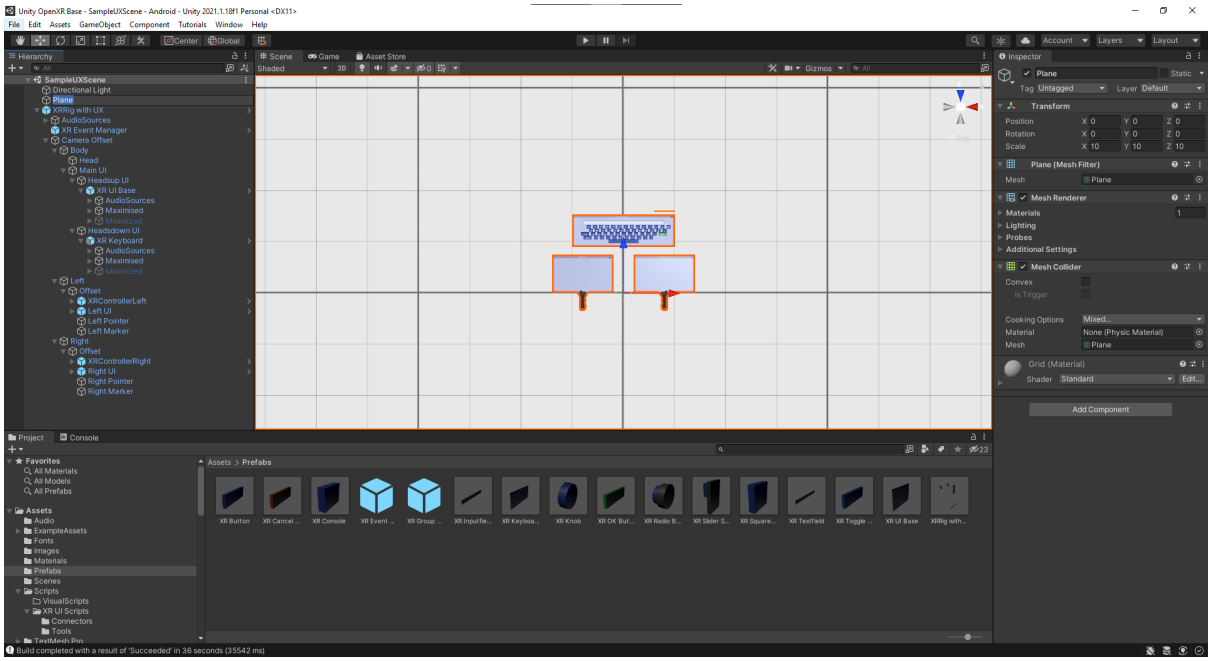


Figure 6 : An example OpenXR with UX project

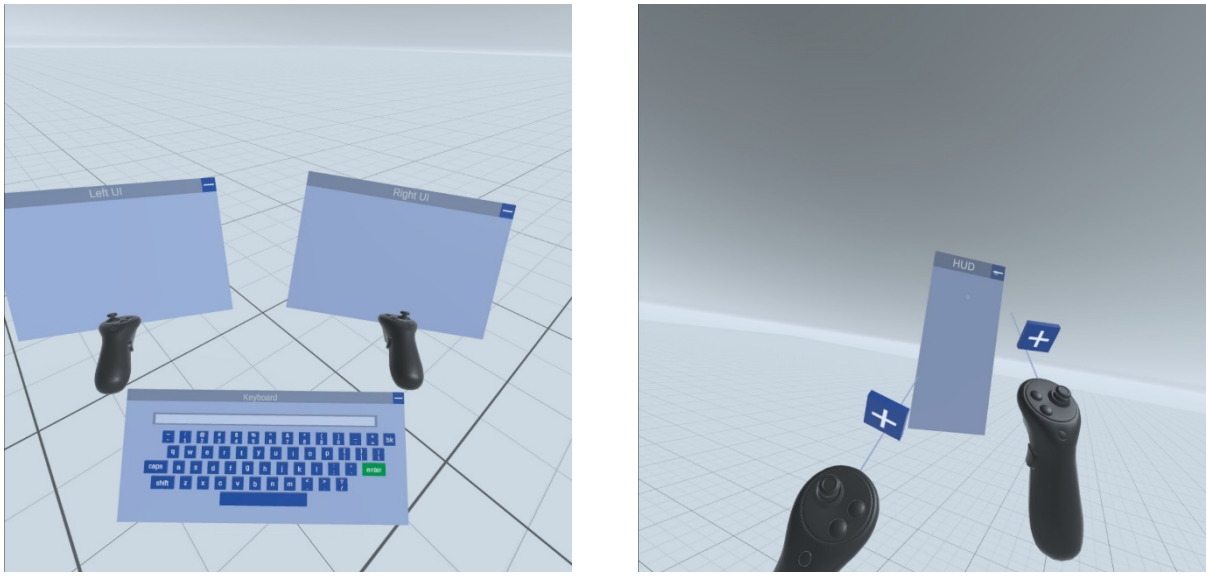


Figure 7 : Views from inside the Sample UX Scene

Problem solving

- Your Oculus Quest is not showing up in the list of devices in the Build Settings.
    - This is most likely due to either the device not being set to Developer Mode (which all devices in the SDL are by default), or that you have not accepted the connection to the computer inside the Quest 2 device. You can try unplugging and plugging it in again to force it to show the request again.
- When building, it pushes the program to the Quest, but it turns up as a window rather than an immersive experience.
    - Check the settings for OpenXR under the 'XR Plug-in Management' tab in Project Settings – you've probably missed checking the box for Oculus Quest Support.

Android Logcat

If you are compiling for the Oculus Quest or other android-based OpenXR headset, you can more easily see what is happening on the device if you install the Android Logcat package from the package manager. As per the instructions on the package: *Android Logcat package adds support for displaying log messages coming from Android devices in Unity Editor. The window can be accessed in Unity Editor via 'Window > Analysis > Android Logcat', or simply by pressing 'Alt+6' on Windows or 'Option+6' on macOS. Make sure to have Android module loaded and switch to Android build target in 'Build Settings' window if the menu doesn't exist.*

Once installed, whenever you run the App on your device, the Android Logcat window will show as per the next figure. Note, you can find here the output printed with the Debug.Log command that would normally go to the Console Window. However, there are hundreds of debug messages coming through, so it pays to add something you can search for in your Debug.Log outputs – in the example below, all debug messages are preceded by the text "OPENXRUX" which has then been entered in the filter field to show only those messages that match – in this case from the XRKnob being turned.

Note, each time you build and run the App, you'll need to choose the correct App on the device to show the messages from (as per the figure below). This is because each time the App runs, it is given a different ID number.

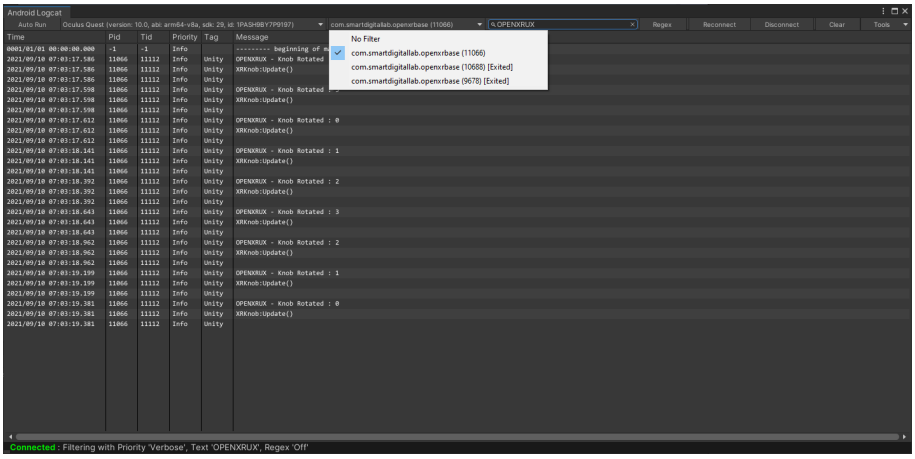


Figure 8 : Android Logcat

Working in Unity play mode

If your graphics card is capable enough according to the Oculus App, it is possible, rather than having to 'Build and Run' every time you make a change you want to test; you can simply press the play button in Unity (Windows only). First, you need to have the Oculus App running on your computer, and for it to be able to activate 'Link mode'. Then, you need to activate link mode on the Oculus Quest (if using a Rift, this is the only mode). Finally, in the Project Settings under XR Plug-In Management / OpenXR, the 'Play Mode OpenXR Runtime' needs to be set to 'Oculus', as below.

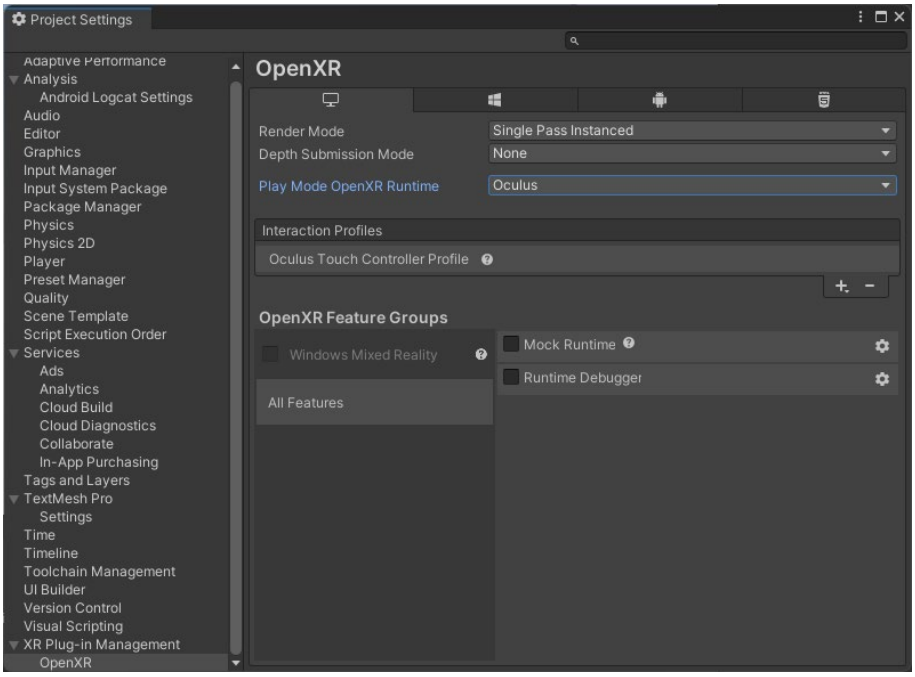


Figure 9 : Setting the play mode in Unity to Oculus

Visual and Performance Quality

To get the best performance from your XR Headset, you can tinker with the Quality under Project Settings to improve the overall look and feel, balancing with performance speed and frames per second. Generally, it is beneficial to allow some level of Antialiasing and Shadows, tweaking them to levels that are visually acceptable, but not too performance impacting. You can add your own Quality levels as per below. Make sure you set which quality level to use for which platform (eg PC or Android) under the Default menu or the effects won't be seen…

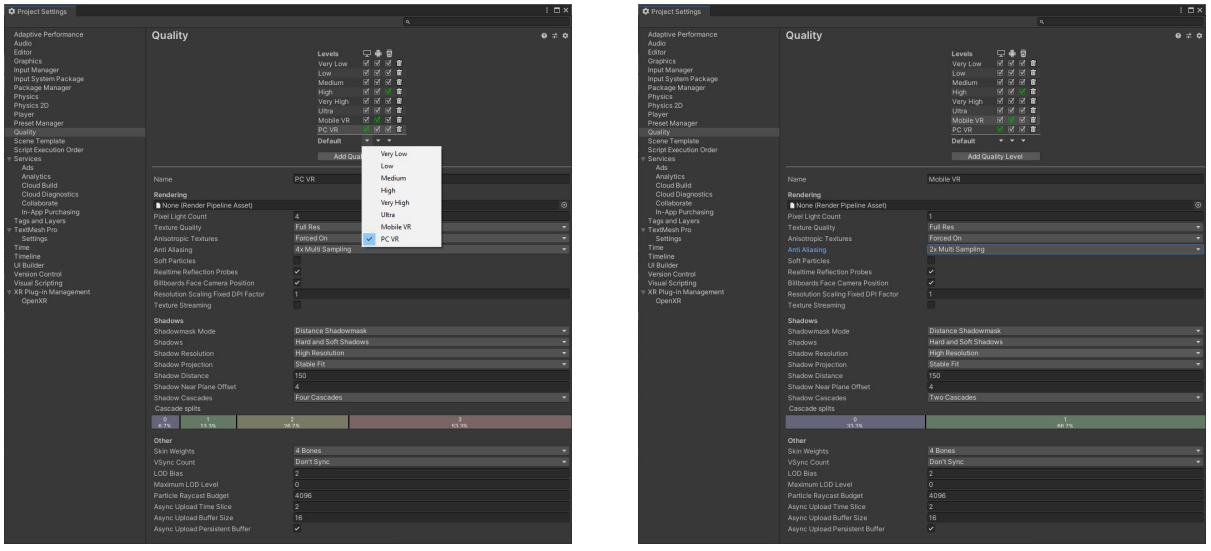


Figure 10 : Quality settings for PC and Mobile VR

In addition, you can set the Scene to use Dynamic Visual Quality which changes the visual quality when the user is moving around to be of lower quality, and then increases the visual quality when the user is standing still. This means that movement can be smooth when the user is not really able to focus on details anyway, and then the visual quality can be high when the user is looking around whilst not

moving. These settings are found on the XRRig under the section 'Dynamic Quality' of the 'XRRig Camera Mover' component as per the figure below. Make sure the moving quality settings are lower than the standing quality settings or it won't have the desired effect.

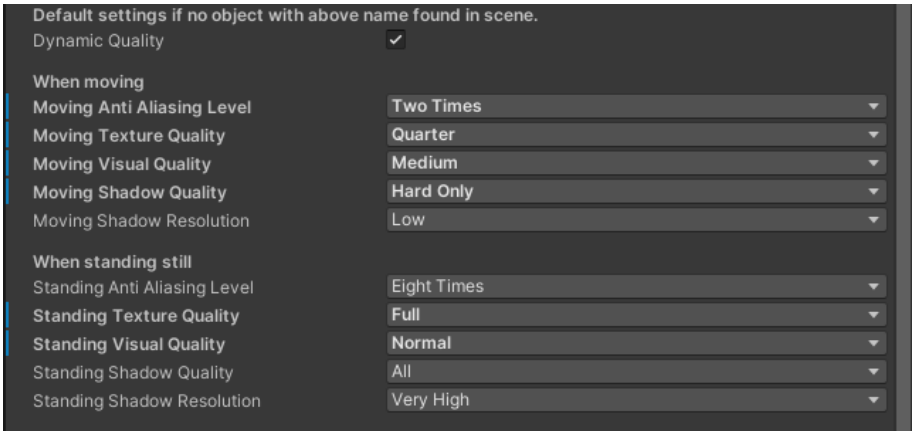


Figure 11 : Visual quality settings on the XRRig

In addition to the global quality settings on the XRRig, you can also set the quality settings for each Scene individually, which will override the settings on the XRRig. To do this, include the object from the prefab folder called 'XR ENTRY', or create one using the OpenXR UX/XR Modules GameObjects menu. It has the following settings:

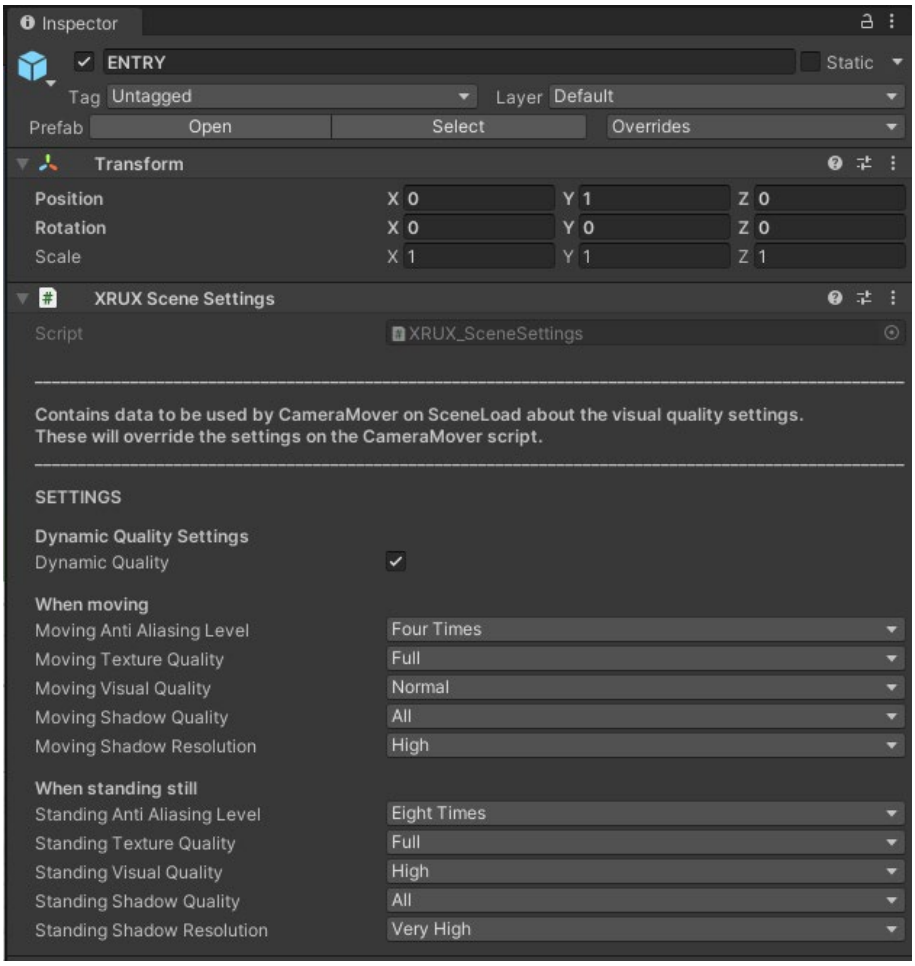


Figure 12 : Visual quality settings on scene ENTRY

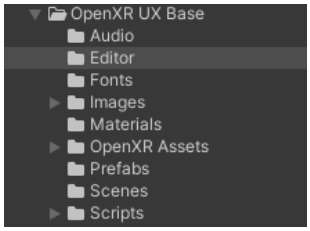
Levels of Detail

Another trick often used to improve the framerate of a Virtual Experience is to make it so that objects far away from view are represented by much simpler 3D objects. Simpler in this context means having fewer polygons and fewer polygons is easier for the graphics card to render – but because the object if far away, the user doesn't see the loss of detail. There are tools in many of the 3D modelling packages to produce these different levels of detail, and the SDL has some resources about how to do this automatically using Blender. You can read more about this is the Unity manual page:

https://docs.unity3d.com/Manual/LevelOfDetail.html

Project Directories

Within the Package, there are several important directories containing useful assets, as detailed below. Using these in this way will keep your project tidy.



Audio

A few audio clips used in the basic UX.

Editor

Some scripts to integrate the OpenXR UX Base into the Unity menus.

Fonts

Some fonts that you can use with TextMeshPro. To add more fonts, check out the TextMeshPro documentation.

Images

A few images used in the basic UX.

Materials

Contains the materials used for the basic UX. Changing these causes the materials across all the assets to change. This is a good way to change the look and feel of your project.

OpenXR Assets

The example assets that come with the OpenXR package, for example the 3D controllers models.

Prefabs

Contains all the prefabs for the OpenXR UX experience. Future updates will likely add new prefabs.

Scripts

The core scripts to manage the OpenXR UX experience, and that drive the objects in the prefabs. In this directory is the directory 'XR UI Scripts' with sub directories 'Tools' and 'Connectors' – explained later. You can add your own scripts here as well.

TextMeshPro

The package uses TextMeshPro – the first time you try to import something from the package, it will ask you to "Import TMP Essentials". Make sure you do that. You can get the Examples and Extras too if you like.

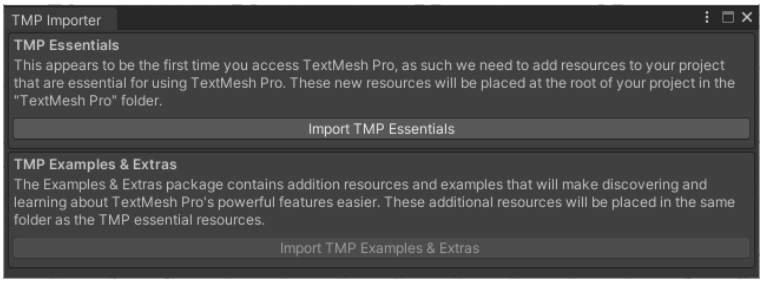


Figure 13 : TextMeshPro import settings

OpenXR Supported platforms

OpenXR has been designed to work across almost any VR device, from stand-alone VR headsets such as the Oculus Quest and Vive Focus through to PC-linked devices such as the Oculus Rift (or Quest in Link mode) and various SteamVR headsets such as the HTC Vive. It should also work on AR headsets such as the Magic Leap and Hololens, though it has not been tested. This is accomplished partly through the cross platform VR development tool, Unity, and partly through various software linkages on the PC. The Oculus Quest, for example, can work in Standalone, PC Linked and SteamVR modes. The OpenXR UX will work in all these setups, but for this guide, we are concentrating on the Standalone VR setup.

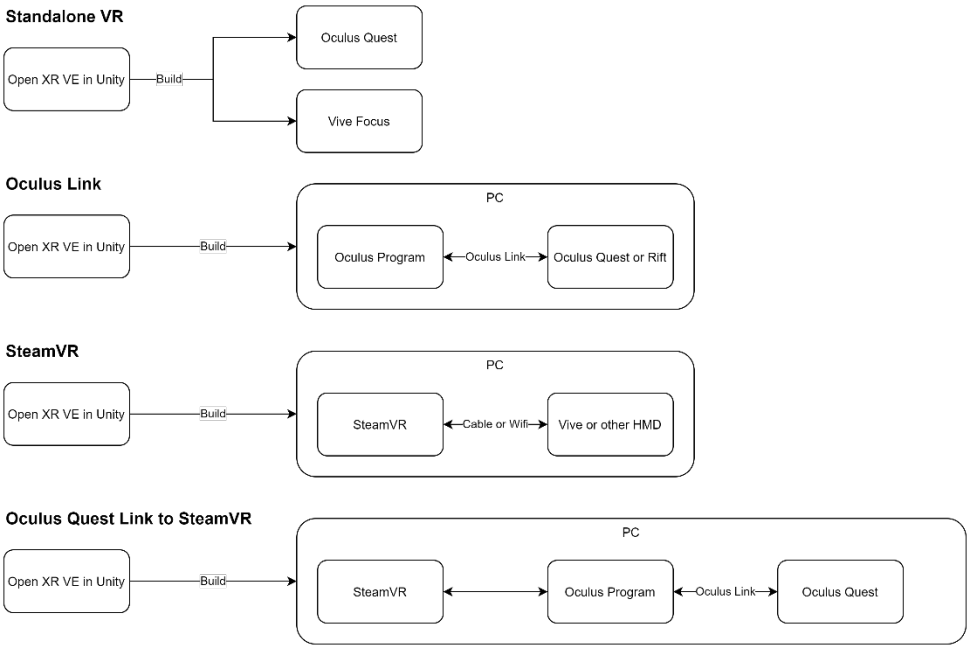


Figure 14 : Quest, Link and SteamVR setups

Building for Oculus PC Link mode

If you want to run your Oculus Quest in Link Mode (or are using a Rift) so you can benefit from the PC graphics card, then the Build Settings are a little different, as below. Mainly, the platform must be set as 'PC, Mac & Linux Standalone'.

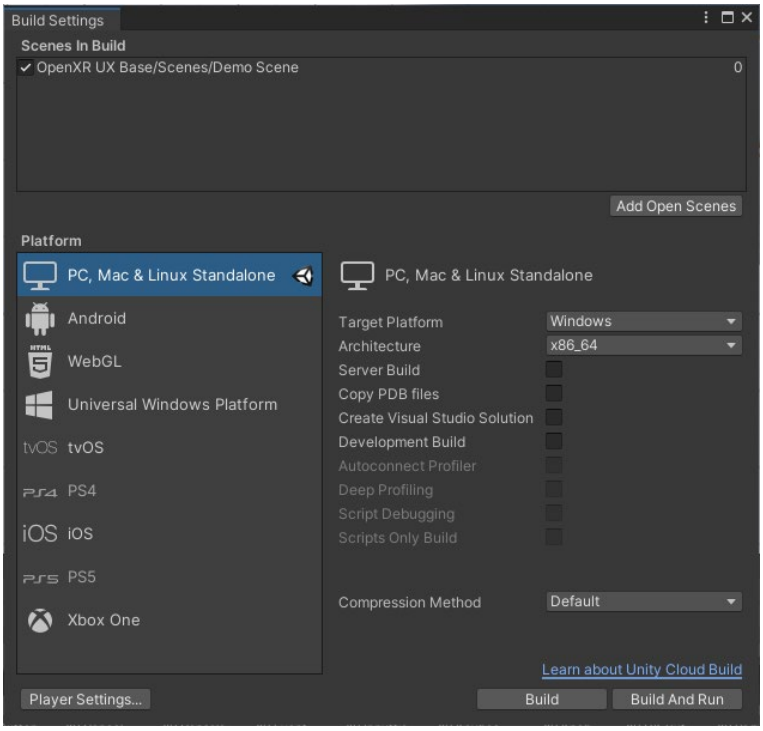


Figure 15 : Build settings for PC Link mode

Further, some tweaks are required in Project Settings – make sure OpenXR (and not Oculus) is selected in the Plug-in Providers.

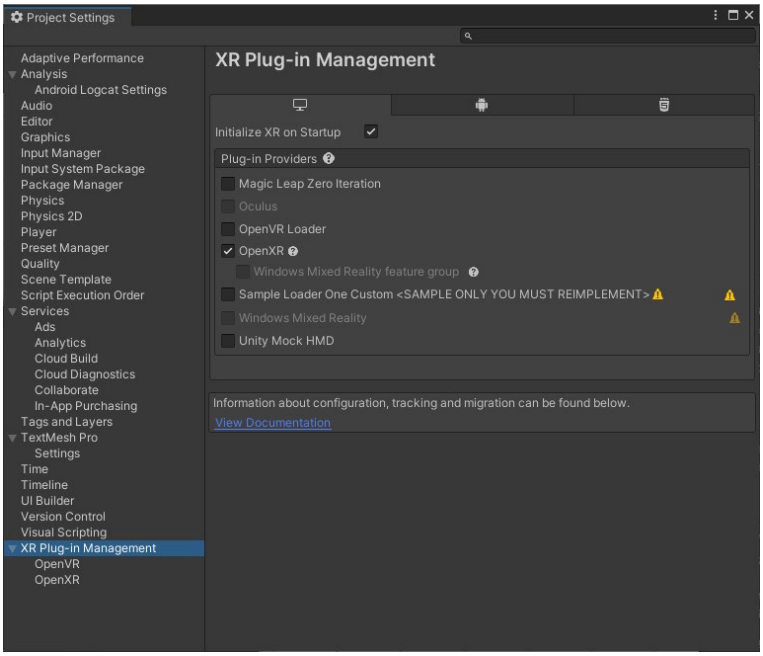


Figure 16 : Project Settings for PC Link mode

Building for SteamVR

Building to run on a SteamVR supported device such as the Vive Pro, is a little more complicated and requires some additional packages and settings.

Step 1 – Install Steam and SteamVR

Before any of this will work, you have to have installed Steam and SteamVR on your PC. These can be

found at the Steam main website (https://store.steampowered.com).

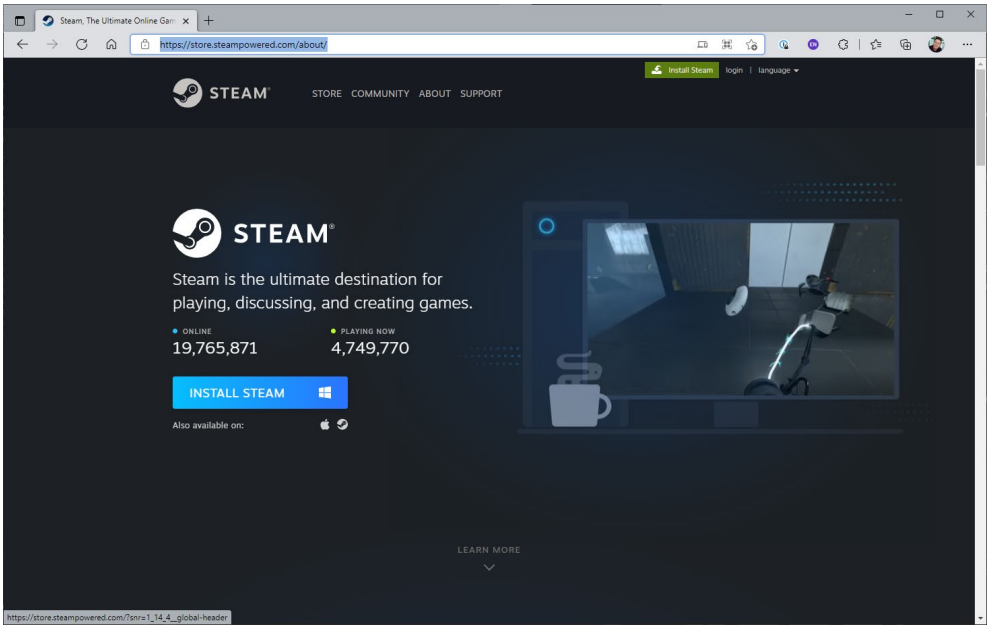


Figure 17 : Steam website

Once Steam is installed, and you have activated your account, SteamVR can be installed from inside the Steam App.

Step 2 – Install the SteamVR Plugin in Unity

This is available in the unity Asset Store. There is also a version included in the OpenXR UX codebase, but if you are starting from scratch, or want the latest version, then download and import this into your project.

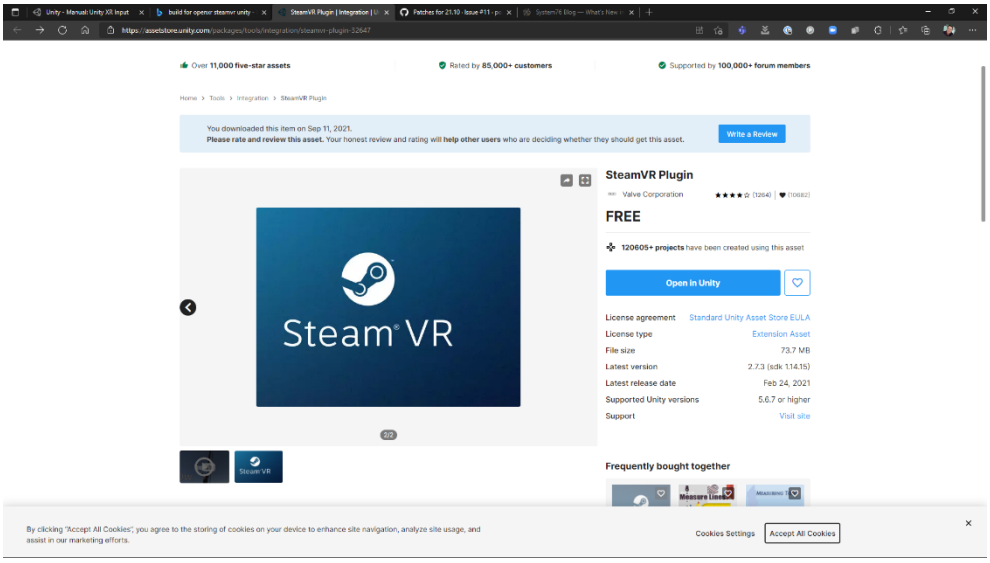


Figure 18 : SteamVR in the Unity Asset Store

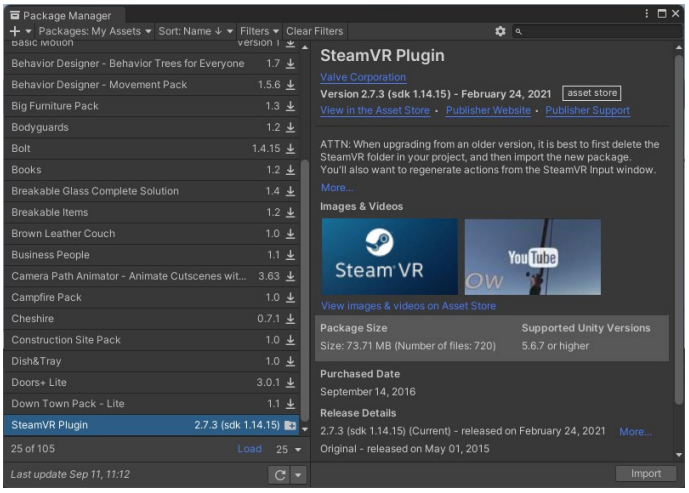


Figure 19 : SteamVR Plugin in Package Manager

Step 3 – Build Settings

Make sure the target platform is 'PC, Mac & Linux Standalone' in 'Build Settings' – same as for building for Link PC mode.

Step 4 – Activate the OpenVR Loader

In project Settings, under 'XRPlug-In Management', there is an additional setting for OpenVR Loader – this needs to be activated.

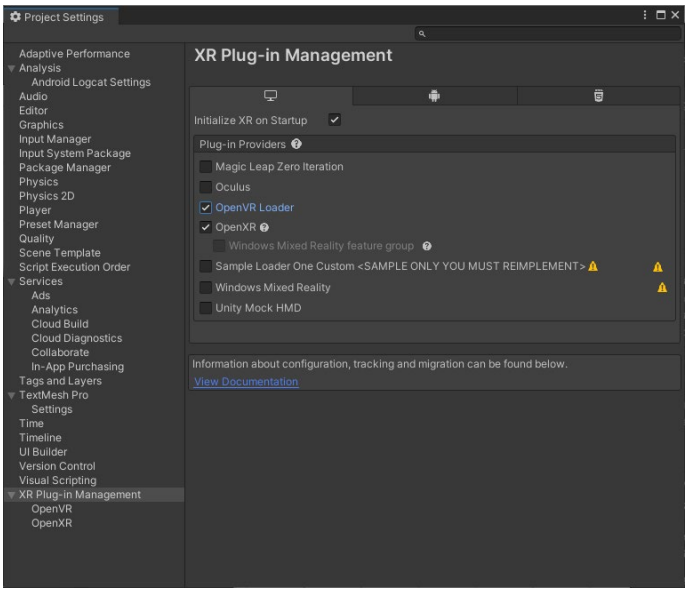


Figure 20 : OpenVR Loader Project Setting

Now, when pressing play in Unity, or doing a 'build and run', this should work on your connected headset.

Note, as of writing, this has not been fully tested with a Vive as I don't have one available, but does work fine with a Quest running in Link mode through SteamVR – so should work for others too 😊.

Building for Vive Focus

Coming soon (once I can get my hands on one…)

Building for Desktop VR

Sometimes you want to be able to build a Virtual Reality Experience for use on the desktop PC, Mac or Linux. The OpenXR UX seamlessly supports this with the following caveats:

- You still need to include the OpenXR libraries;
- It should automatically detect whether a standalone VR headset is connected or not, and run in the appropriate mode – but this is not fully tested as yet; and
- Some user interactions are difficult to map from interactive VR to desktop VR, so your UX design may need to take that into the account.

Mapping the keyboard and mouse

In Desktop VR mode, the game controllers are not available, so keyboard and mouse actions are used instead. The following mappings are applied:

Key / Mouse	Action
w	Left Controller Thumbstick Forward
s	Left Controller Thumbstick Back
a	Left Controller Thumbstick Left
d	Left Controller Thumbstick Right
↑	Right Controller Thumbstick Forward
↓	Right Controller Thumbstick Back
←	Right Controller Thumbstick Left
→	Right Controller Thumbstick Right
Left mouse button	Right Controller Trigger Click
Right mouse button	Right Controller Grip Click
Mouse scroll wheel	A DesktopVR unique mapping that can be useful for interaction with objects that require controller movement, such as the XRUX_Knob, which turns as you twist the controller after clicking. This is the only instance where code at the XRUX Object level may need some tweaks.
Mouse movement	Head movement – allows the user to look around by moving the mouse.

The Player settings need to be modified to allow both the old and new forms of input handling (at this stage), otherwise the key presses aren't registered.

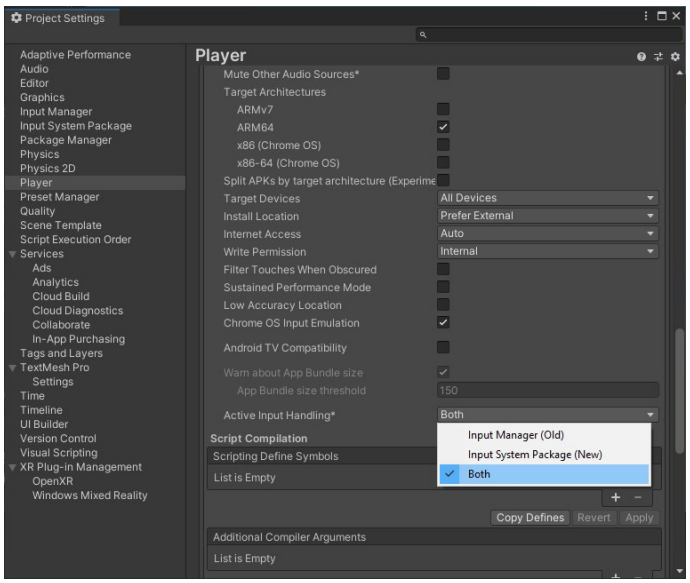


Figure 21 : Active Input Handling Settings for DesktopVR

The mapping occurs at the level of the OpenXR Core Manager, so all subsequent interactions will occur as normal, regardless of whether in Desktop or Interactive mode, with events propagating through the Event Queue as normal, but interactions with the Virtual Environment occurring via the mouse rather than through the pointers on the game controllers.

Figure 22 : Running in Desktop VR

Playing inside Unity

One advantage of built-in Desktop VR mode is that when you play the Virtual Environment inside Unity, it assumes you are in Desktop VR mode, and adjusts accordingly – making it much easier to test many aspects of the experience without always having to build and run on a standalone headset. However, when you do build and run to the headset, it will reconfigure and work as Immersive VR.

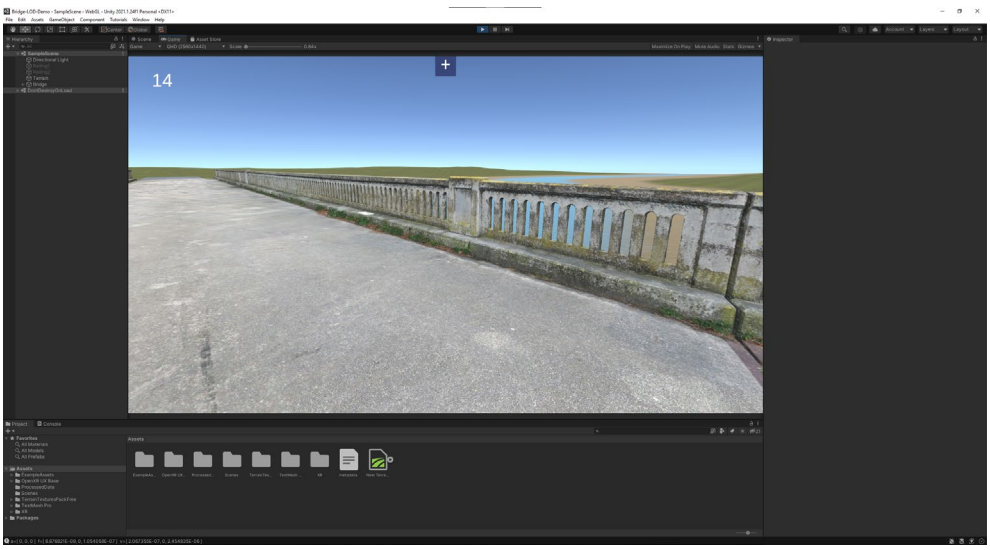


Figure 23 : Unity play mode with DesktopVR

Building for Windows, Mac or Linux

To build for Desktop VR, set the Platform to 'PC, Mac and Linux Standalone', and make sure that OpenXR is not enabled. If you leave OpenXR enabled, it should run in DesktopVR mode when a headset is not connected, and Immersive VR mode when one is connected, but this is yet to be fully tested and confirmed.

Quite possibly, you may wish to change the settings for how the DesktopVR is to be displayed by adjusting the player settings – for example, to run in Windowed mode.

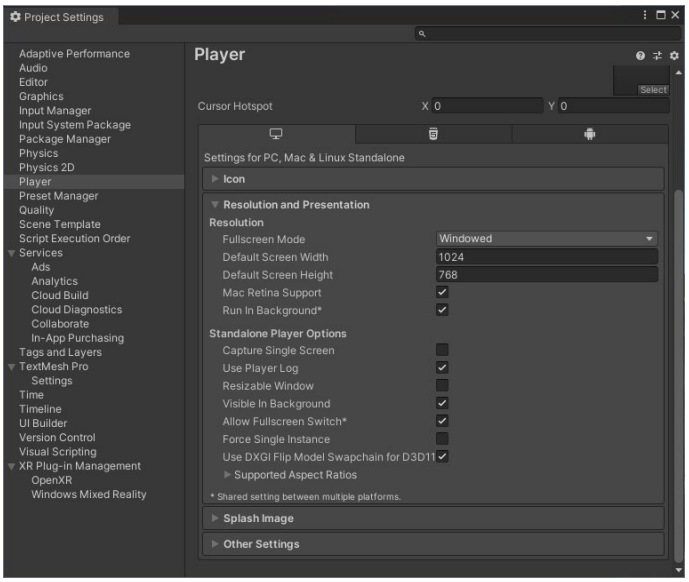


Figure 24 : Resolution and Presentation for PC Standalone

Building for WebGL

In a similar way as building for DesktopVR, you can also use build for WebGL so that you can embed the Virtual Environment into a webpage. There are many limitations to this, so check out the relevant pages in the Unity documentation (https://docs.unity3d.com/Manual/webgl.html).

WebGL mode uses the same mouse and keyboard mappings as for Desktop VR, and doesn't support WebVR or WebXR directly, although there are some AssetStore libraries that might work.

WebVR is a promising looking standard that will allow Interactive Virtual Environments to be published via webpages that will then work on whatever VR headset is attached to the computer, or from inside a Standalone VR headset. It's not quite ready for mainstream yet.

To export a WebGL version, make sure you have installed the WebGL module in UnityHub, and set the Platform to WebGL. There may be some tweaks to various settings required, but Unity is pretty good at letting you know what you need to do.

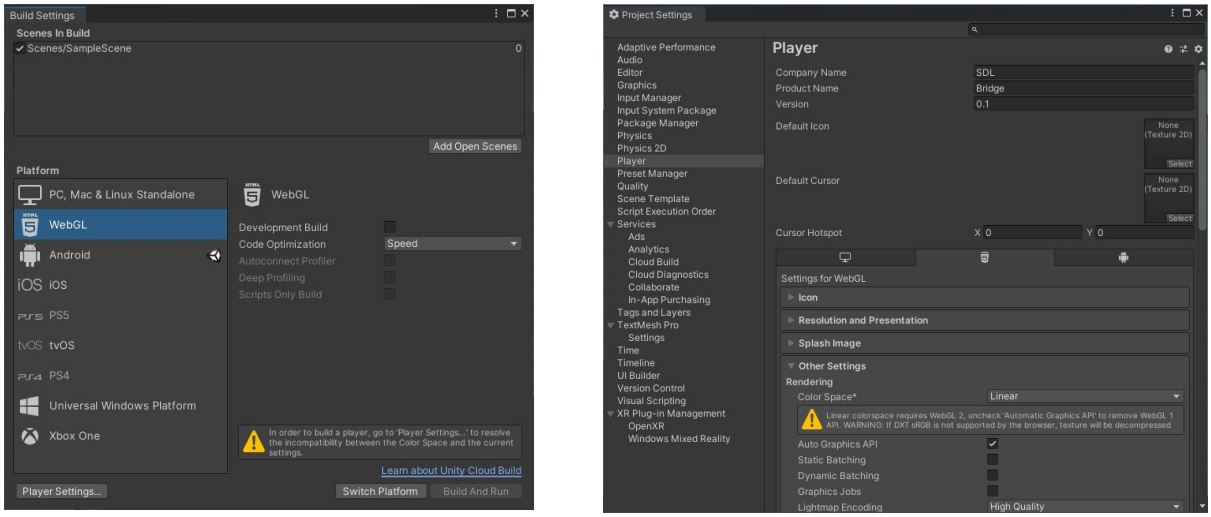


Figure 25 : WebGL Project Settings Tweaks

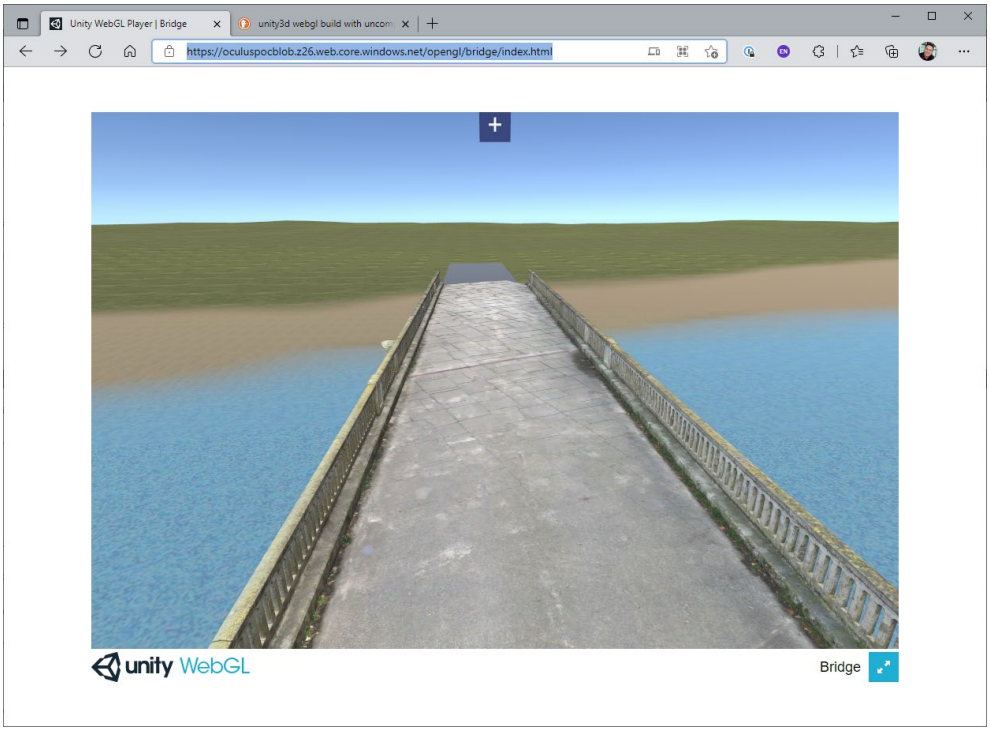


Figure 26 : Running WebGL inside a browser

Core Concepts

OpenXR

OpenXR is an open, royalty-free standard for access to virtual reality and augmented reality platforms and devices. It is developed by a working group managed by the Khronos Group consortium. Unity supports the OpenXR standard, with a variety of plugins becoming available for different headsets and devices. Using OpenXR, you can use the same project for any similar device without having to rewrite the programs for the different requirements. For example, running the program on Oculus Quest or HTC Vive should require very few changes.

You can read more about OpenXR on its Wikipedia page: https://en.wikipedia.org/wiki/OpenXR.

UnityEvents

Unity makes it easy to send information from one GameObject to another using a concept called 'Events'. In OpenXR, this is used primarily to transmit information from the controllers to any GameObjects that you want to react to the buttons, thumbsticks and triggers, and to communicate between the different OpenXR UX GameObjects.

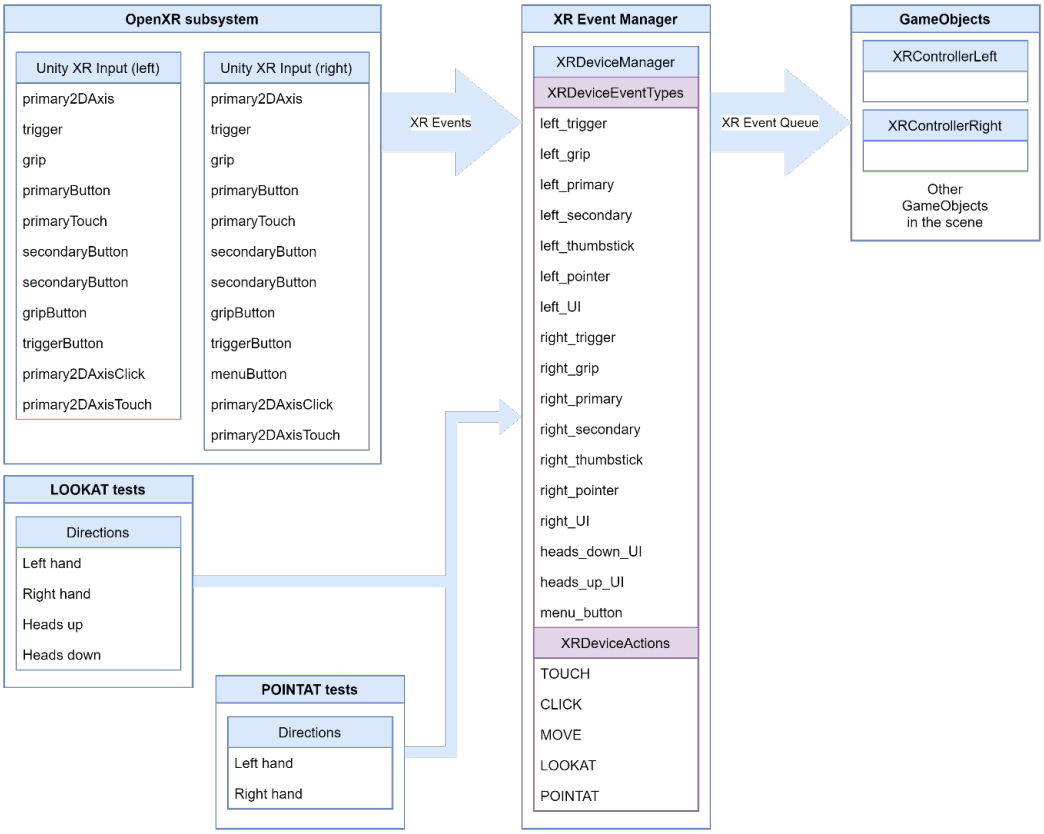


Figure 27 : Unity Events from OpenXR

The core of this is implemented in the XRRig_Manager.cs and XRRIg_SendEventOnAngle.cs scripts.

Tags and Layers

Unity provides ways of classifying objects into groups that can then be used to simplify calculations and reduce unnecessary computations by only including the GameObjects in groups that are required for that functionality. The OpenXR UX uses the following tags and layers:

Tag: XREvents

- For the XR Event Manager – there should be only one of these in any scene that contains the XRDeviceManager.cs script and is tagged XREvents.

Tag: XRLeft, XRRight

- Any GameObjects that are intended to be used to activate items from the left or right hand. Presently the Pointer attached to the Controller, and the Marker that moves around in the Virtual Environment as you move the Controller around.

Layer: 6

- This can be named anything you like, for example 'XR UX'. Any objects you want to be interacted with by the Pointers and Controllers need to be on Layer 6, otherwise they are ignored by the pointing system.

Layer: 7

- Objects that can be teleported and moved onto, particularly the ground, but also potentially other structures. Using this technique, you can create surfaces where the player is allowed to go, making it easy to restrict the player from no-go areas.

Layer: 8

- Areas where the player cannot teleport to or move onto. One way to use this is to create invisible planes under objects that you want the player to avoid, and put them in layer 8. The OpenXR UX movement system does the rest. These can even sit above layer 7 objects to exclude movement to part of a larger area.

Layer: 9

- This is used for the current user's body objects, which are not rendered for that user so they don't obstruct the viewport.

Layer: 10

- Other participant's bodies – these will therefore be visible in multi-user mode (yet to be implemented).

When you set up a fresh project and create a new XRRig_with_UX, the tags and layers are set correctly.

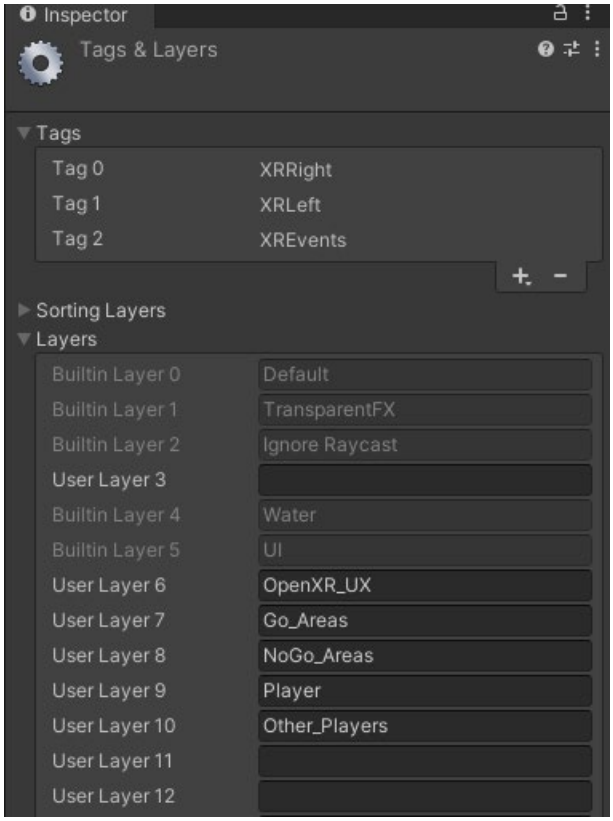


Figure 28 : Tags and Layers for OpenXR UX

XRData

In order to facilitate the connection of XRModules, a common data type is used to pass between modules that can hold boolean, integer, float or string data. To set an XRData variable, construct a new XRData object and pass in the data that you wish it to hold; for example, the following sends a new XRData Event with a float value:

```
onChange.Invoke(new XRData(3.0f));
```

When XRData is set, the type is defined by the value being sent in, and it is converted to the other types so the internal representation is always consistent. Conversion happens as follows:

- Floats are rounded to integers (so below X.5 goes down to X, otherwise up to X+1).
- Booleans are false if the float or integer is 0, or if the string is ‘false’, true otherwise, or if the string is ‘true’.
- Strings take on the string equivalent of the number or Boolean.
- Strings inputs are converted to numbers or Boolean as appropriate, but if not successful the result is 0 or false (ie not an error).

XR UX Rig, GameObjects and Components

The OpenXR UX library comes with some shortcuts to make it easy to integrate into your project. You can create a UX Rig, add new XR UX GameObjects and add functionality using Components to existing GameObjects.

Converting your camera to an OpenXR Rig with UX

As above, any scene can be converted to an OpenXR compatible one using the GameObject menu ‘Convert Main Camera to XR Rig With UX’. This not only adds some default XR elements for you to

populate (though you can remove these of you want something else), but also adds the movement algorithms using the controllers.

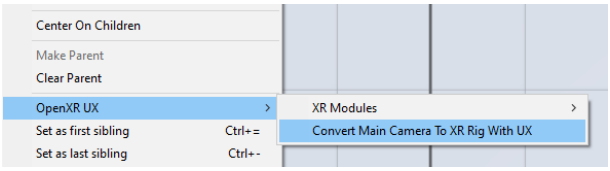


Figure 29 : Convert Main Camera to XR Rig With UX

If this doesn't work (for example if the camera you are trying to replace is inside a complex hierarchy of GameObjects), delete the GameObject heirarchy that includes the Camera, and try again.

Adding an XR UX Module as a GameObject

XR UX Modules can be added to your scene through the GameObject menu. Select first the parent object in the Hierarchy, then right click or use the GameObject menu find the OpenXR GameObjects as below.

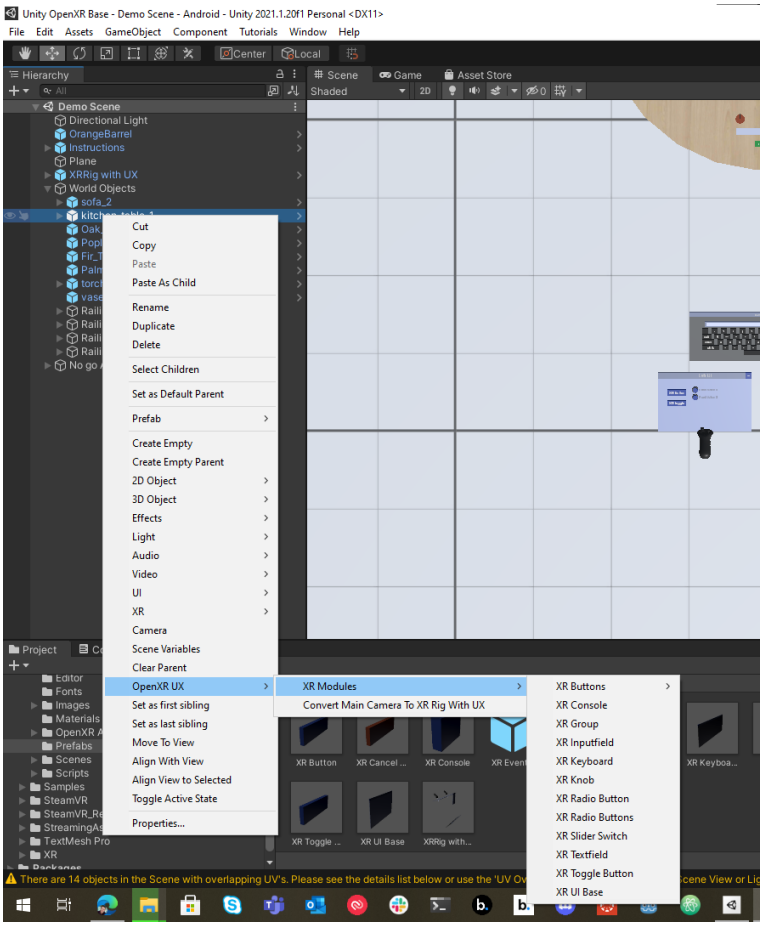


Figure 30 : Adding a XR UX GameObject

Adding an XR UX Module as a Component

Similarly, an existing GameObject can have a XR UX Module added as a component. Usually, this would be to add a Tool to a GameObject so it can be affected by the OpenXR UX, however, if you want to design your own UX modules, for example, for buttons, you can use those script components as well.

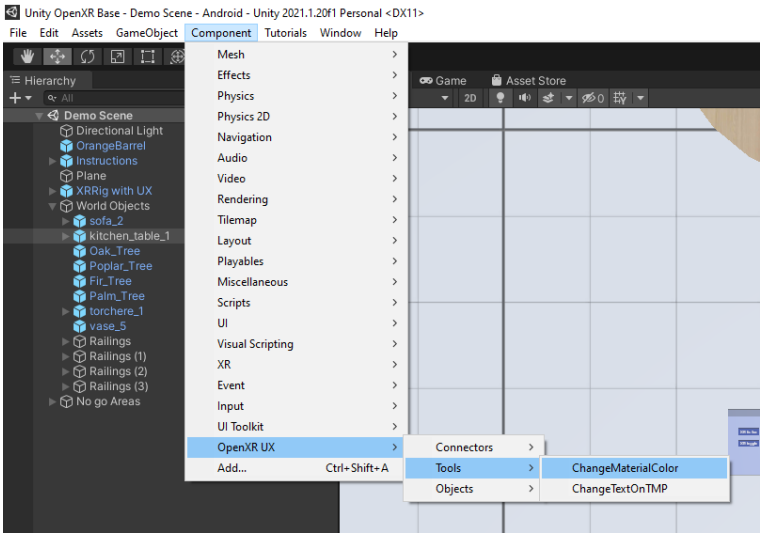


Figure 31 : Adding a XR UX Module component

Maintaining the XRRig across multiple scenes

Oftentimes, you will want to set up a UI on the XRRig controllers, headsdown and headsup areas that you will want to carry over between different scenes. However, in Unity when loading a new scene, all objects are usually replaced, including the current XRRig. To avoid this problem, you can choose to set the XRRig to 'Persist across Scenes' in the XRUX Set Scene Component on the main XRRig.

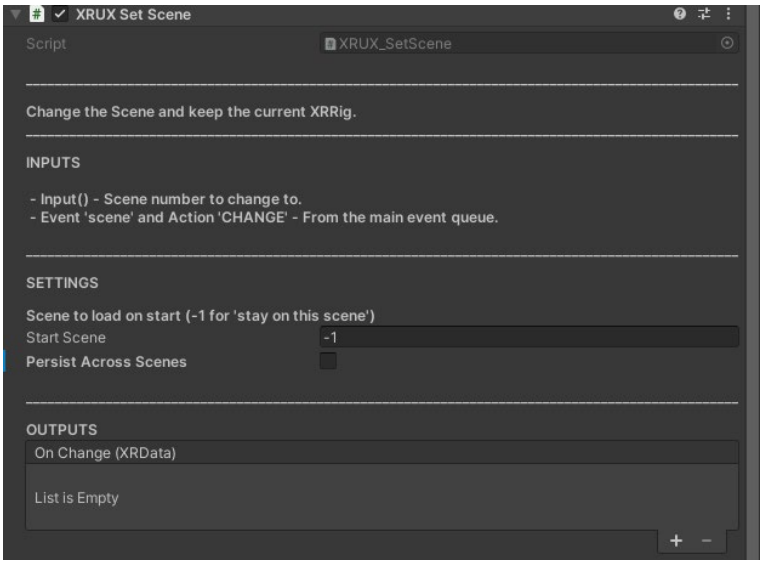


Figure 32 : Persisting the XRRig across scenes

One problem with this approach, however, is that the viewpoint of the user also carries over between scenes, so you end up in the same location from the previous scene, but in the new scene. Sometimes, this is not the desired effect. To get around that, you can include in the destination scene an XR ENTRY object, placed at the location where you want the user to enter the scene. This prefab also contains visual quality settings for the destination scene.

When persisting the XRRig across scenes, any additional XRRig or camera system in the destination scene is removed, which allows you to test the scene by itself, without having to build and run all the scenes, and without having to remember to delete the XRRig from the destination scene before building and running all the scenes together.

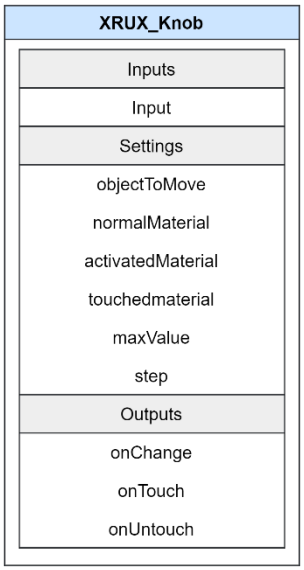
Note that if you want different UIs for different scenes, just put an XRRig in each one and don't tick the 'Persist Across Scenes' box. Note that if there is no XR ENTRY object, the entry position will be wherever the XRRig is placed, and the visual quality settings will be those on that scene's XRRig (or the XRRig persisting across scenes).

XR UX Module types

There are three types of XR UX Modules: Objects – which are visible, interactable elements in the Virtual Environment; Tools – which interact with other elements on the SceneGraph hierarchy; and Connectors – which sit in the event chain and add or modify XRData as it comes through.

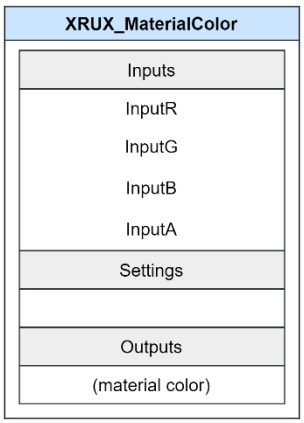
Objects

The XR Module Objects are designed to be easily linked up using the Inspector in Unity. Each module consists of a number of inputs and outputs, for example the XRKnob takes one input – a value that can be used to move the knob from another XR Object, for example in the response to other events; and three outputs that are activated when the knob is touched, or untouched (ie when the user moves the pointer off the knob), and when the knob is turned and the value changes.
Modules are defined as GameObjects that can be interacted with using the GameControllers.



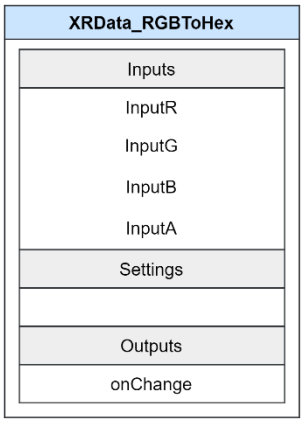
Tools

In a similar way, XR Module Tools which are defined with inputs that can be connected via the Inspector, and generally are used to affect other GameObjects but provide no further outputs. For example, XRUX_MaterialColor Tool takes 4 inputs of RGBA and changes the material color of the GameObject it is placed on.



Connectors

XR Module Connectors that take some inputs and produce an output in response, designed to interface between Modules to change data from one format to another that can be used by the receiving module. For example, XRData_RGBToHex turns the inputted values into a string in Hexadecimal that can then be used to print to the console or some text output.



Dynamic and Static parameters

When connecting inputs to outputs in the Inspector, the input parameters can be either dynamic or static. Dynamic parameters are taken from the GameObject outputting the value and change during runtime, whereas Static parameters are set in the Inspector and don't change. You can tell the difference in the Inspector by whether there is a field to enter a value (static) or not (dynamic). If you find that items are not changing value as expected, you probably have chosen the static rather than the dynamic parameter option.

Figure 33 : Dynamic vs Static Parameters in the Inspector

In the following figure, the parameter being sent to the XRUX_Textfield is a dynamic input. You can see the two possibilities for Input as both Dynamic and Static.

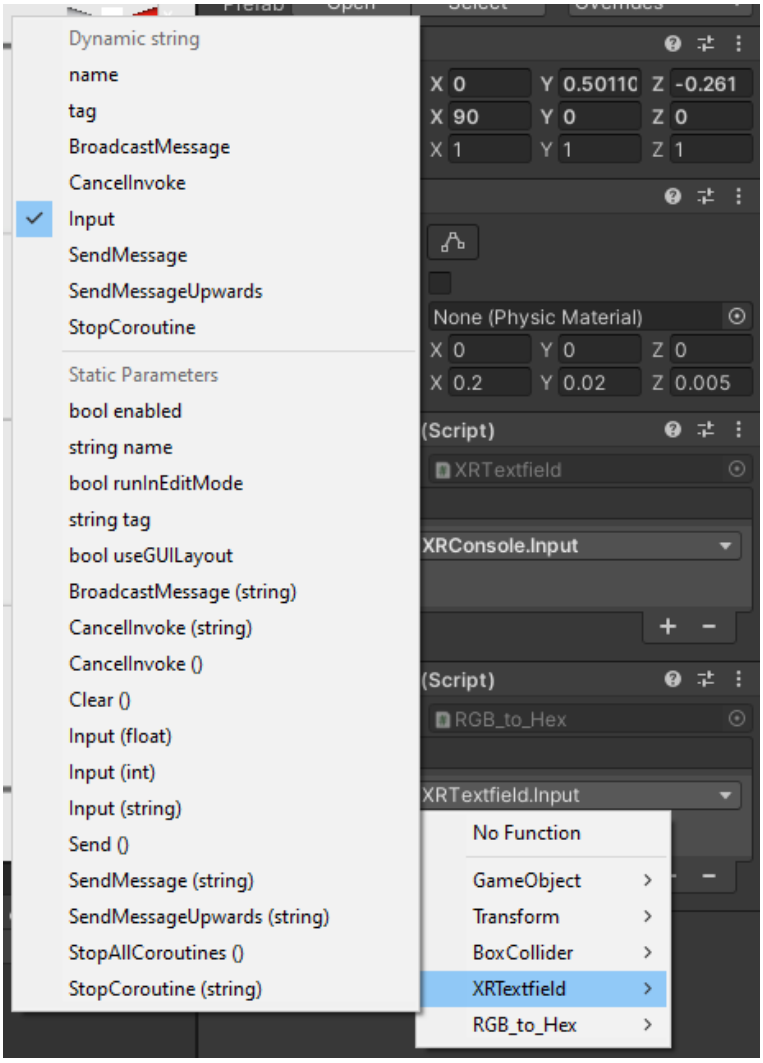


Figure 34 : Selecting a Dynamic Parameter in the Inspector

XRModules

In this section, we detail the current set of XRModules, which are Unity Components programmed to perform certain tasks so they can be easily included in your project. The code has been written to make them easily modified further or used as templates for new XRModules. XRModules form a chain that connects user input (usually) to some required effect. For example, the figure below shows a chain of XRModules that connects the user twisting three XR Knobs (red, green and blue) to changing the colour of an object in the Virtual Environment and show the resulting Hexadecimal representation of that colour on a display.

The XRUX_Knobs are taken directly from the prefabs; the Readout likewise, but has had another XRModule Component added to convert the separate RGBA values into something more understandable; and the vase is a standard GameObject with a XRUX_MaterialColor XRModule Component added.

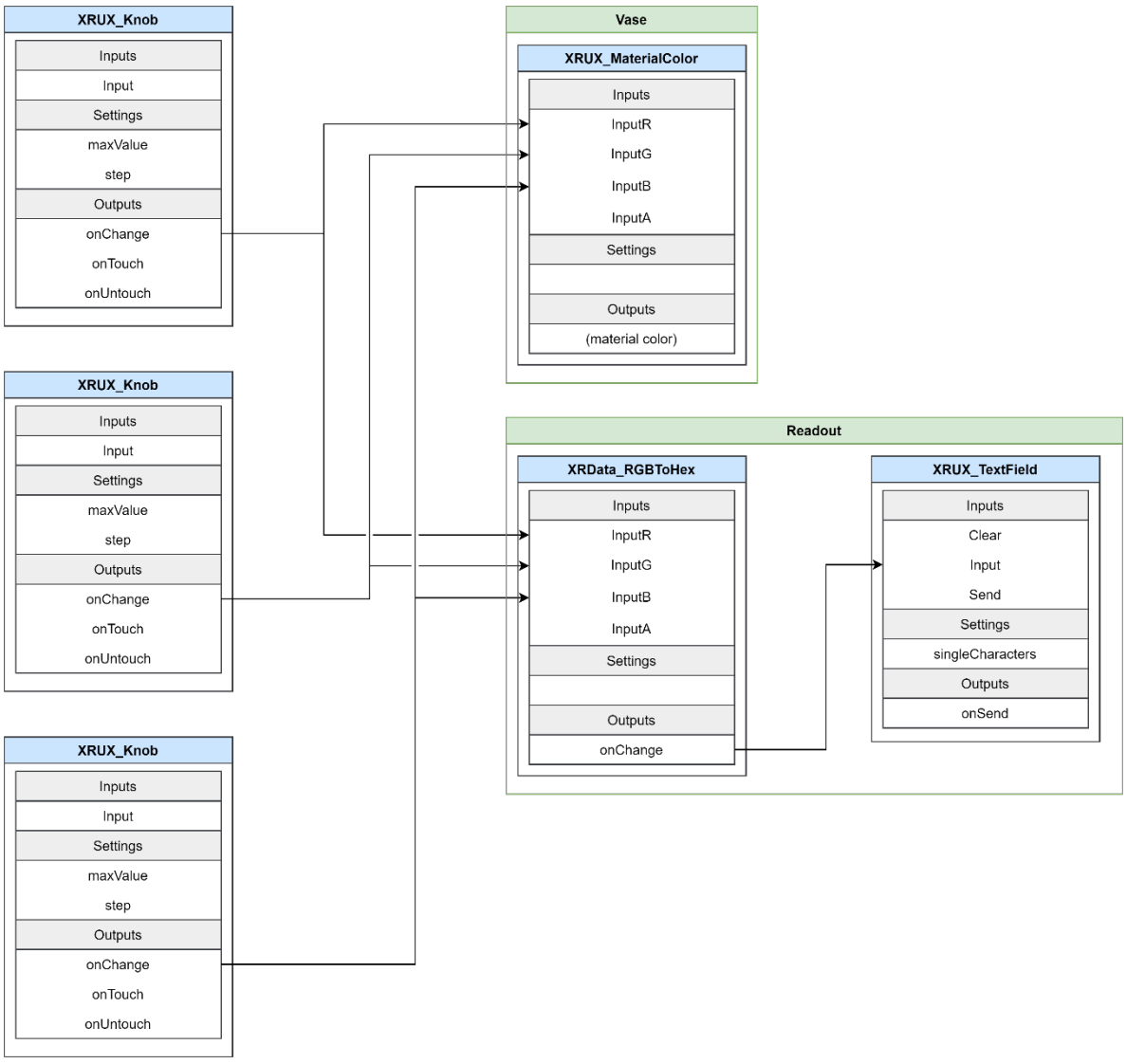
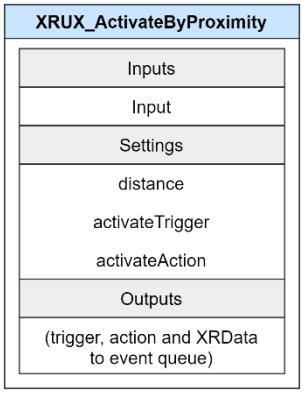


Figure 35 : Example XRModules connected in a chain.

Tools

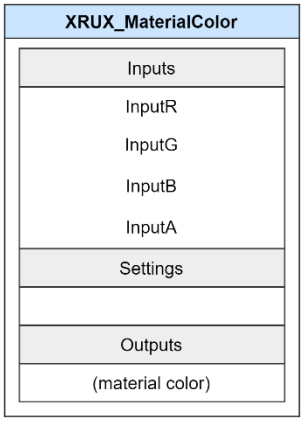
The following XR Tools are presently defined.

XRUX_ActivateByProximity



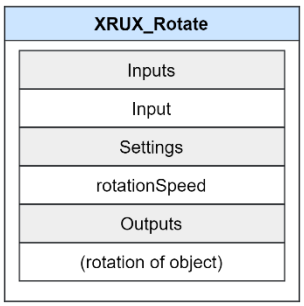
A tool that sends an event, action and some XRData when the user (ie the camera object) gets close. To set the XRData, send it in from some other module (eg XRData_Integer if just a simple number).

XRUX_MaterialColor



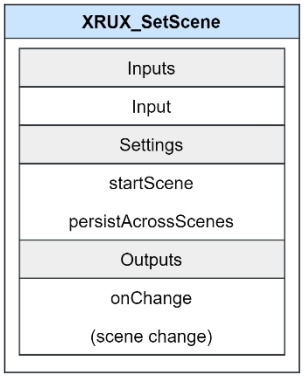
A tool that changes the colour of the material of the GameObject the script is placed on. The value are between 0 and 255, and start at 255 (ie opaque white).

XRUX_Rotate



Set the rotation in degrees on the y axis of the object it is on, and move to that rotation at the given rotationSpeed.

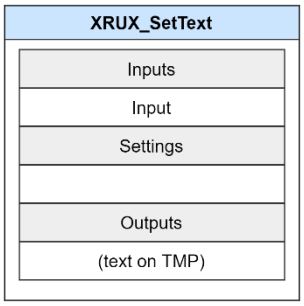
XRUX_SetScene



Change the scene to the one given in the XRData integer attached to the event. When the scene is changed, activates the onChange event. The initial camera position is set to the position of an empty GameObject called 'ENTRY'.

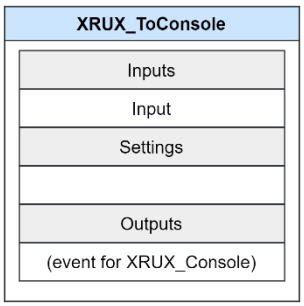
This is typically placed on the root of the XRRig by default, and makes sure that the XRRig is carried across (persisted) across the scenes loaded if persistAcrossScenes is true. In that case, it also removes any competing camera objects and the hierarchy they are on upon entry into a scene so that there are not two main cameras – which can cause some odd effects.

XRUX_SetText



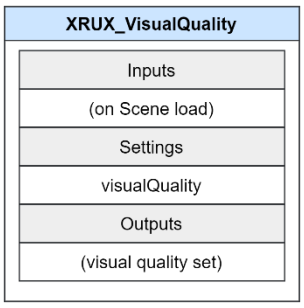
A tool that changes the text field of the TextMeshPro object the script is placed on. Used mostly internally for changing the titles on other XR Objects, but also used in the setting of text on the keyboard.

XRUX_ToConsole



Sends XRData to every XRUX_Console in the scene without requiring a direct link. This makes it easier to use the console to see what is happening in the scene.

XRUX_VisualQuality

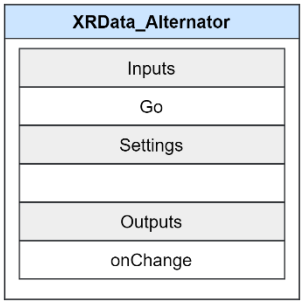


Sets the visual quality of the scene when loaded. The values are low, medium, normal, high and extra, where low is half resolution, normal is 1 to 1 resolution, and extra becomes one and a half times as fine.
Visual quality impacts the performance, so if the scene is particularly complex, using a lower resolution can help keep the framerate higher, but at the expense of visual quality.
Other settings can also be used in Unity to improve performance or visual quality such as the level of antialiasing, and the style of shadows.

Connectors

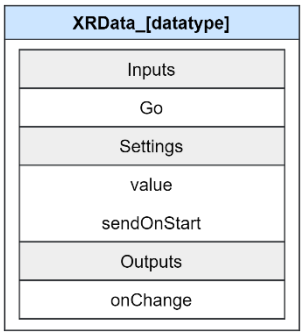
The following XR Connectors are presently defined. These typically work on XRData.

XRData_Alternator



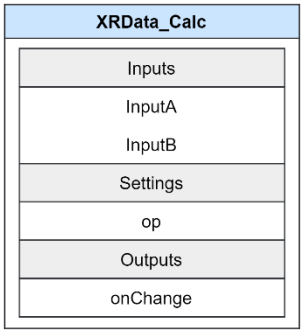
Toggles the output between true and false.

XRData_Boolean, XRData_Float, XRData_Integer, XRData_String



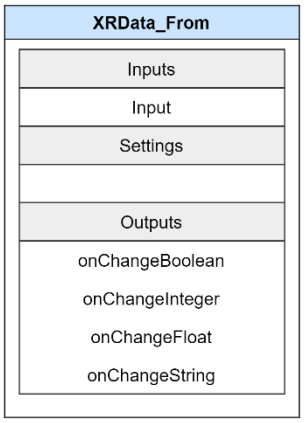
Generates a variable of the given type to send further. If sendOnStart is true, send the value initially when the scene is loaded, as well as when it receives the Go signal.

XRData_Calc



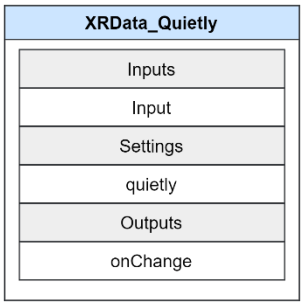
Calculates and sends on a simple mathematical calculation of the XRData coming through InputA and InputB. The operation to be performed is defined in 'op'. The calculations are InputA op InputB, eg InputA + InputB. The onChange is only activated once both Inputs have been set at least once.

XRData_From



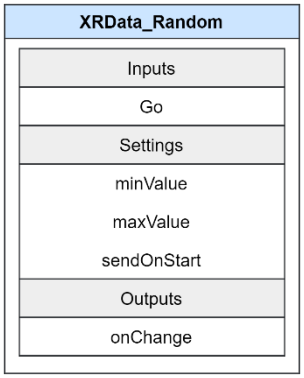
Converts XRData to an ordinary Boolean, Float, Integer or String. By connecting to a specific output, the connector also acts as a means to convert from one type to another. See the section on XRData above about how conversion occurs.

XRData_Quietly



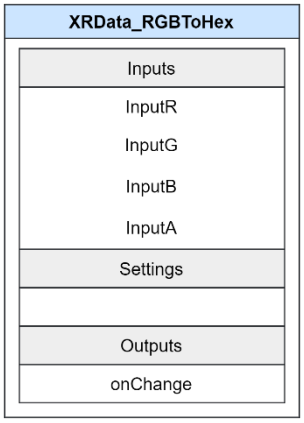
Sometimes if you have two XR UX modules interlinked, you can get a never-ending loop of activations that will crash Unity as each triggers the other's events. To break this cycle, add a 'quietly' parameter to the XRData which tells the receiving module to not send any further events (for example to its onChange output).

XRData_Random



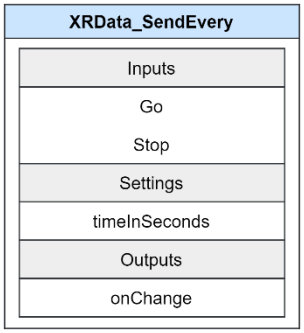
Generate a random number in the given range. If sendOnStart is true, sends this on when the scene is loaded as well as when the Go signal comes in.

XRData_RGBToHex



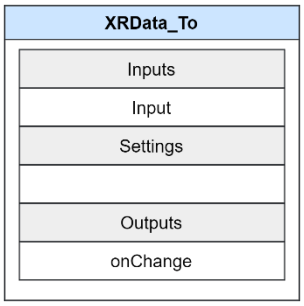
A connector that converts Red, Green, Blue and Alpha values between 0 and 255 into a Hexadecimal string.

XRData_SendEvery



Creates an XRData signal at a regular interval time-period in seconds.

XRData_To



Takes an ordinary Boolean, Float, Integer or String as Input and converts to XRData. Useful as an interface between non-OpenXRUX components and OpenXRUX components.

Playing a sound

Playing a sound in response to, say, a button being pressed, or a knob being twisted doesn't require a special Module as you can just directly access the AudioSource Play function, for example as below:

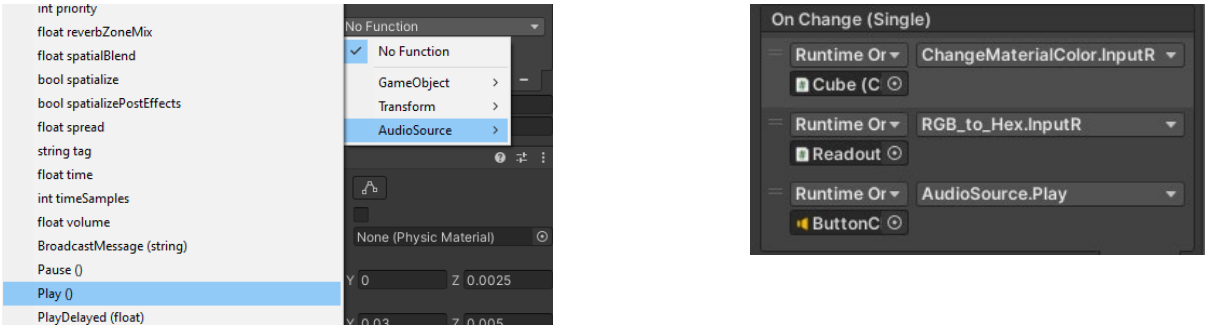
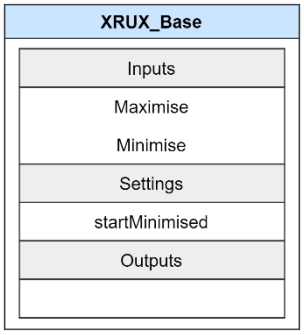


Figure 36 : Playing a sound when an XRKnob is turned by choosing Play() on an AudioSource

Objects

The following XR Objects are presently defined.

XRUX_Base



XRUX_Base is a convenient flat surface that can be minimised and maximised for you to put other XRObjects on. However, you are not limited to this, and can use any GameObject to hold XR Objects.

This is used with the prefab 'XRUX Base' with pre-defined interactions with the minimise and maximise buttons.

Figure 37 : XRUX_Base Inspector and Scene views

XRUX_Button

XRUX_Button
Inputs
Title
Input
Settings
objectToColor
objectToMove
normalMaterial
activatedMaterial
touchedMaterial
movementAxis
movementAmount
movementStyle
Outputs
onChange
onTouch
onClick
onUnclick

A generic interactable button that takes an input to activate or is activated by being clicked on, and produces XRData outputs when it starts and ends being touched, when it is changed (ie turned on or off) and Boolean events when it is clicked and when the click finishes.

The material of the button can be set to change when activated or touched, reverting back to the normal material otherwise.

The button can move when pressed along the specified axis by the specified amount.

The button can be either momentary or toggle on and off.

The text on the button can be changed dynamically by sending a string to 'Title'.

This class is used to derive many other types of button, switch and toggle which are available as GameObject prefabs as shown below.

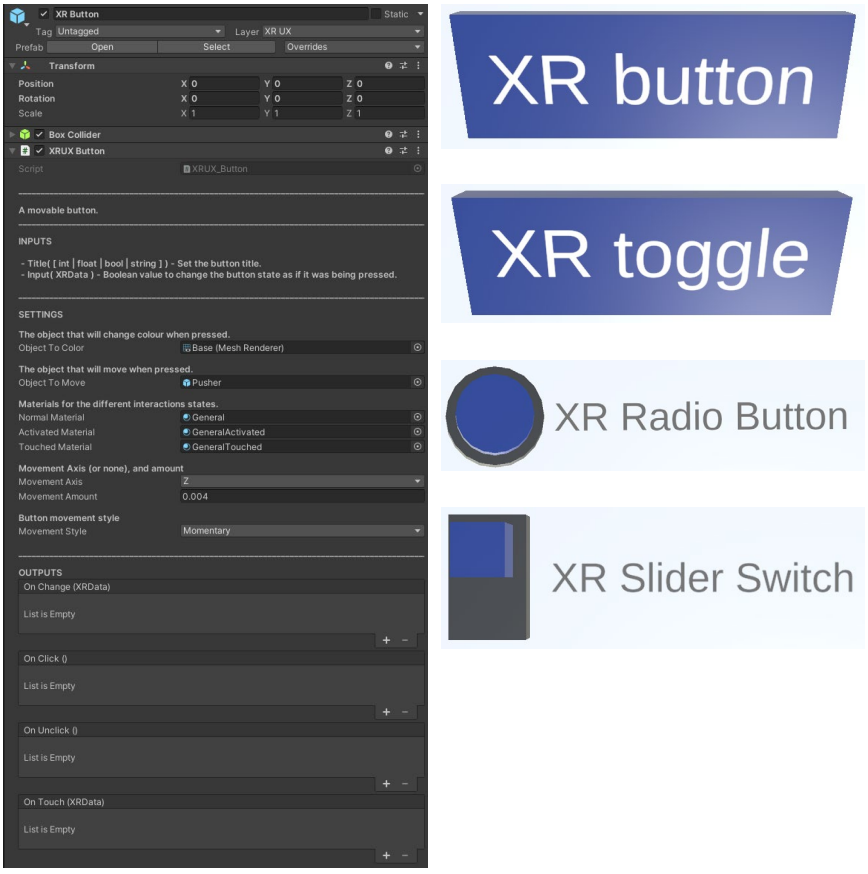


Figure 38 : XRUX_Button Inspector and Scene view for different styles of button

XRUX_ButtonGroup

XRUX_ButtonGroup
Inputs
Input
Settings
buttonPrefab
dynamicButtons
dynamicButtonSpacing
Outputs
onChange

A generic XRModule that makes it easy to create a set of XRUX_buttons.

To use this, first create an XRUX_ButtonGroup GameObject in the scene from the prefabs, then either add some strings to the dynamicButtons array in the Inspector, or drag some XRUX_Buttons from prefabs onto the XRUX_ButtonGroup GameObject so they become children objects in the Scenegraph (you'll need to space them out in the vertical direction as well). DynamicButtonSpacing defines the vertical spacing of the dynamically created buttons, taken from the bottom of the last button in the children radio buttons.

Input takes an integer and toggles the specified button allowing the set of buttons to be controlled from other XRModules.

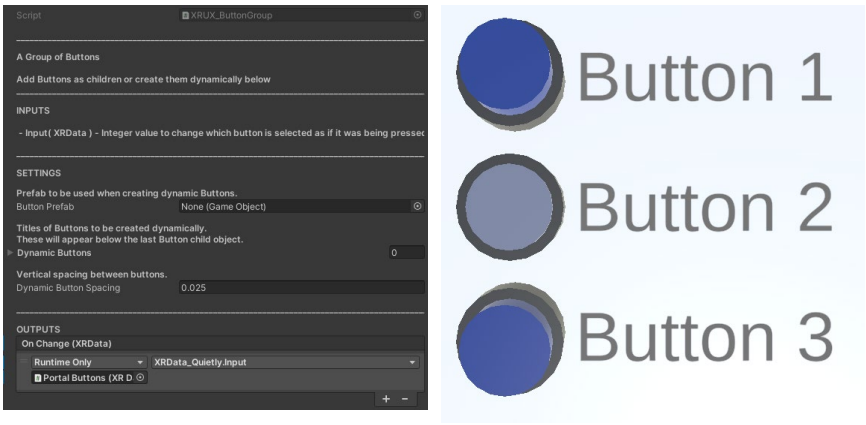


Figure 39 : XRUX_ButtonGroup Inspector and Scene view for Radio Buttons

XRUX_Console

XRUX_Console
Inputs
Clear
Input
Settings
numLines
acceptGlobal
Outputs
(text on the console object)

A column of text that can be useful for sending values to when you need to know what is happening inside your code whilst running the Virtual Environment (since the Unity console is not available when inside the VE). The prefab Includes a pre-linked-up button to clear the console. This is useful when put in front on a surface, for example in the Heads Up Display (HUD).

The parameter numLines defines the number of lines in the console, and the Clear and Input commands clear the console, or add a line of input respectively. The XRUX_Console in the prefabs has 20 lines, but you can make a console whatever size you like and formatted however you like. If not used in the prefab, use this script as a component on a TextMeshPro GameObject.

The object can also accept input from the XR eventQueue as sent by the XRModule XRData_ToConsole if acceptGlobal is true.

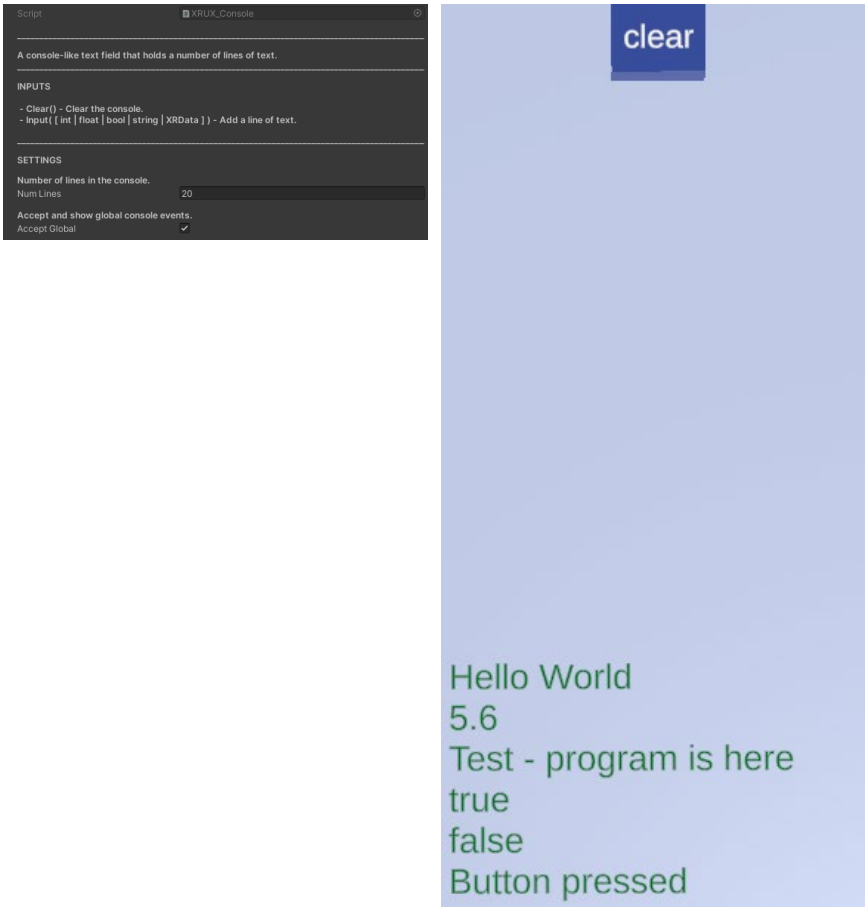


Figure 40 : XRUX_Console Inspector and Scene views

XRUX_Inputfield

XRUX_Inputfield
Inputs
Backspace
CapsLock
ShiftLock
Clear
Input
Send
Settings
singleCharacters
Outputs
onSend

An input field that collects text sent via 'Input' and behaves like a computer keyboard when Backspace, Capslock or Shiftlock are called. Clear erases the text held, and Send causes the collected text to be sent on to the next Module on the onSend output.

On setting up in the SceneGraph, you can also set a parameter 'Single Characters' to true or false. When true, you can send two characters to the Input (for example as the output from a button press), and then depending on whether Shift or Caps are on or not, it will take either the first or second character. This makes it easy to build a keyboard with lower and upper case characters, and symbols – as with the XRUX_Keyboard module.

Have a look at the XRUX_Keyboard module to see how this works.

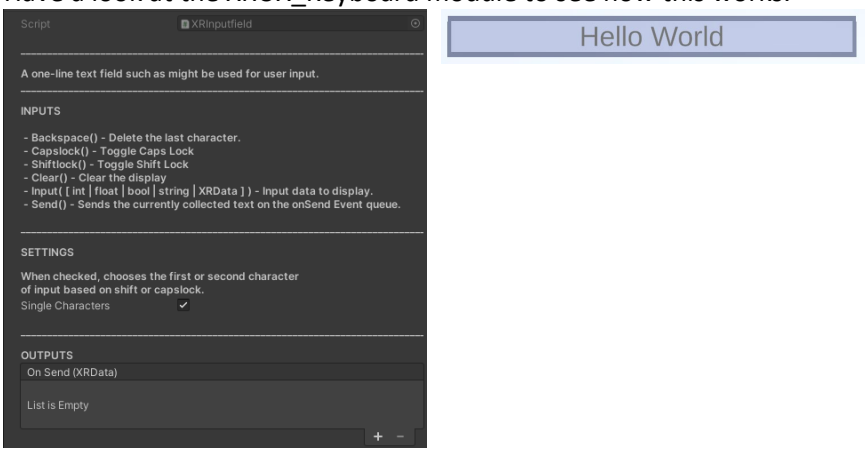
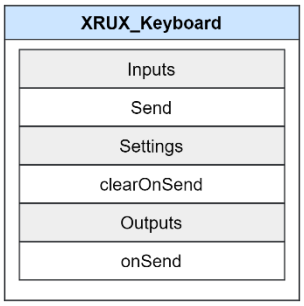


Figure 41 : XRUX_InputField Inspector and Scene views

XRUX_Keyboard



The XRUX_Keyboard is a meta object made up of other XR Objects to represent a keyboard that the user can 'type' on with the controllers. Text is collected in the XRUX_Inputfield display, and can be sent wherever required when the enter button is pressed (you will need to link the onSend output from the . Note that the Shift key is more like a shift-lock to make it easier to use with controllers. Keys click when pressed. The keyboard can be minimised and maximised.

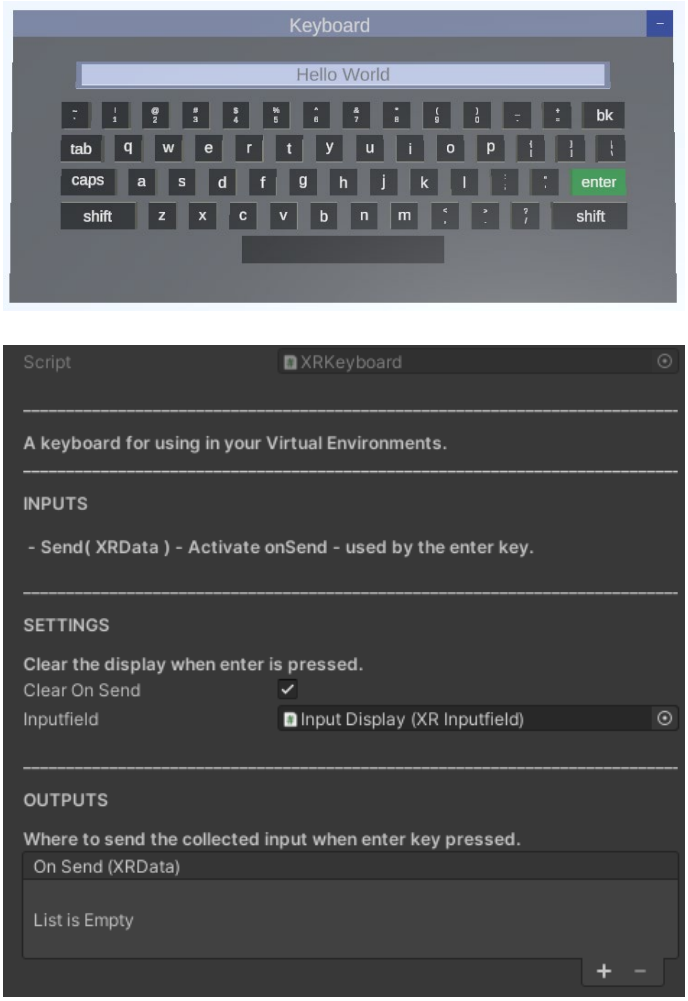


Figure 42 : XRUX_Keyboard Inspector and Scene views

XRUX_Knob

XRUX_Knob
Inputs
Input
Settings
objectToColor
objectToMove
normalMaterial
activatedMaterial
touchedmaterial
maxValue
step
Outputs
onChange
onTouch
onUntouch

As the name suggests, this is a knob that can be clicked on and turned (by twisting the hand controller in the roll direction relative to the knob).

When setting up the knob, you set the maximum value and the step at which the knob 'clicks' around, and as with the XRButton, you can set the materials. As with XRUX_Button, the materials for normal, activated and touched can be set as well.

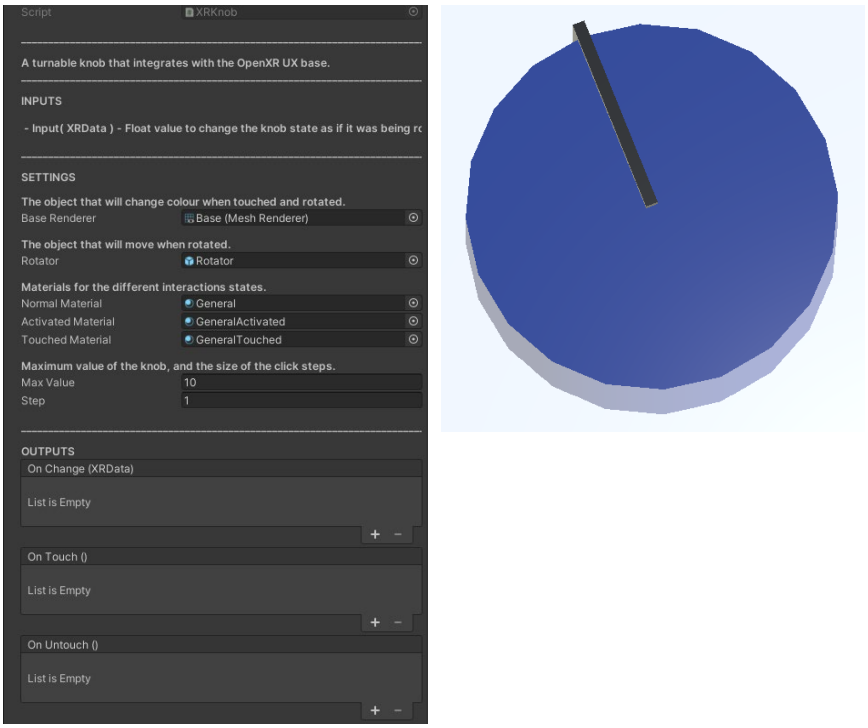


Figure 43 : XRUX_Knob Inspector and Scene views

XRUX_Textfield

XRUX_Textfield
Inputs
Clear
Input
Send
Settings
singleCharacters
Outputs
onSend

The XRUX_Textfield is a simpler version of the XRUX_Inputfield. This takes in text via Input, has a commend to clear the display, and another to send on the text to another module.

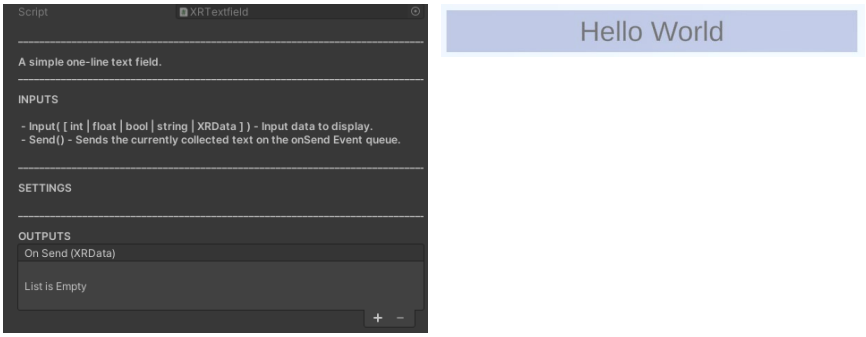


Figure 44 : XRUX_Textfield Inspector and Scene views

Prefabs and XRUX GameObjects

Many convenient prefabs have been included to make it easy to create a variety of VR user experiences. These are also available through the GameObjects/XR Modules menu.

Note that these prefabs have been deliberately made using very simple objects. You can create your own sets of GameObjects and put the appropriate scripts on them to create your own set of prefabs to use in your project.

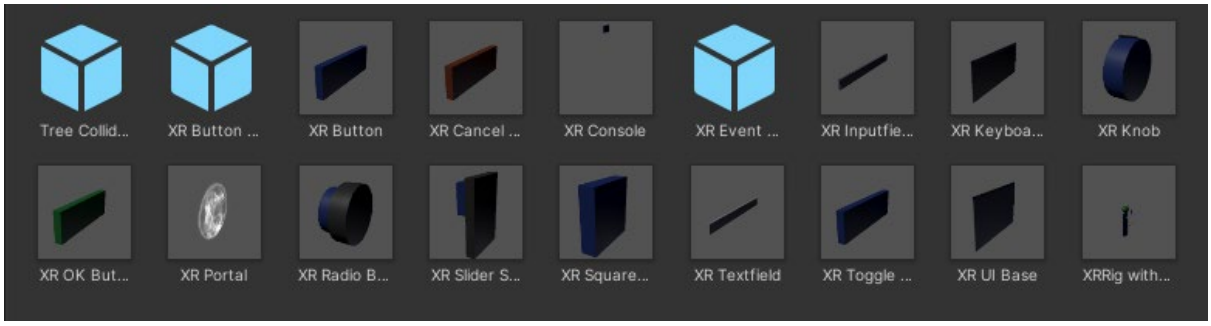


Figure 45 : XRUX prefabs in Unity

XR Button Group

A group of buttons, ready to be used with any button that derives from XRUX_Button.

XR Button

A single button coloured the normal (blue) colour.

XR Cancel Button

A single button coloured the cancel (red) colour.

XR Console

A preset console object with a clear button.

XR EventManager

The main Event Manager – must be one of these in the scene somewhere. Is included by default on the XRRig.

XR Inputfield

An input field.

XR Keyboard

A keyboard, such as the one used on the heads down display.

XR Knob

An turnable knob.

XR OK Button

A single button coloured the OK (green) colour.

XR Portal

A meta object that allows you to change scene by ‘going through a portal’.

XR Radio Button

A single radio button that toggles on or off.

XR Slider Switch

A single slider, two-position switch.

XR Square Button

A single button, in a square format. Good for keyboards.

XR Textfield

A textfield.

XR Toggle Button

A single button that toggles on or off.

XR UI Base

A flat surface with minimise and maximise buttons useful for putting XR Objects on.

XRRig with UX

The main rig with hand controllers, heads up and down interfaces, and left and right controller interfaces, with scene loading and character movement built in. This is the XRRig that is used in the menu command 'Convert Main Camera to XR Rig with UX'.

Navigation and Interaction using the Controllers

Moving around your Virtual Environment is generally achieved by simply walking around if you are in room-scale mode. However, for environments bigger than your physical space, or for stationary mode when sitting down, you need some other means to move.

This library includes a simple navigation script attached by default to the XRRig to get you going, called the XR Camera Mover. It has many features:

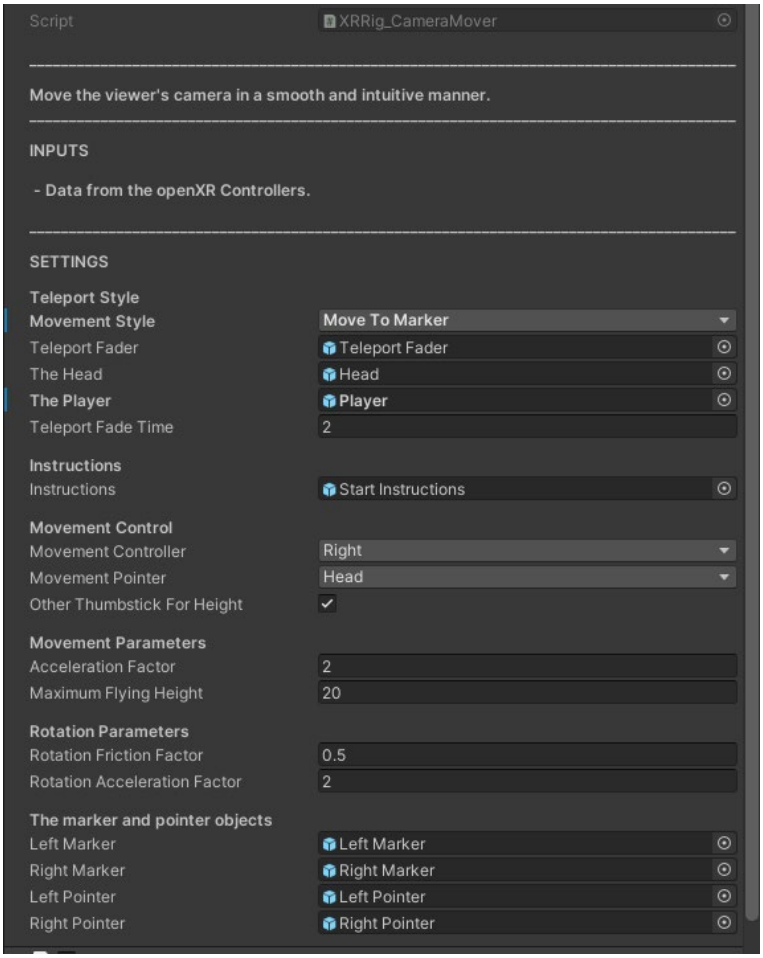


Figure 46 : The XR Camera Mover

In addition to moving around, the process of getting the input data from the OpenXR framework and making that available to you; the programmer, for your users; is encapsulated in the user experience of the Pointers and Markers and can be found in the many Helper scripts in the XR UI Scripts folder.

Teleport vs Move to Marker

You can teleport the user to the Marker (the big blue ball on the ground at the end of your Controller Pointer), or you can move the user swiftly to that spot. To activate, the user presses the Trigger to make the marker and pointer visible, points to where they want to go, and squeezes the Grip.

If teleporting, the scene fades out and back, and they find themselves at the new location. If moving, the viewpoint will move towards the new location, though may be stopped by obstacles or no-go areas.

Moving using the thumbsticks

For fine tuning your location, or moving smaller distances, the thumbsticks can be used – push forward

or back to move forward or back, and push left or right to rotate left or right. Either thumbstick may be used depending on the setting. The angular and linear accelerations can be adjusted as desired.

Using the head or controller to direct movement

Either the controller being used for moving can be pointed to direct movement, or the movement can move in the direction the user is looking. The most intuitive seems to be to move where the user is looking.

Flying around

Sometimes, you want to see things from above. Either the other thumbstick can be used to adjust the height, or the view can be moved up or down by pointing the controller up or down when moving the thumbstick, or looking up or down if head-directed movement is chosen. The maximum height determines how high the user can move the viewpoint.

Go and No-go areas.

Parts of your environment can be designated as places one can navigate to, and others can be designated as no-go areas. This is similar to the built-in Navigation and Wayfinding tools in Unity, but simpler to make it less taxing for mobile VR and AR headsets. This is achieved using Layers (as described in the section on Tags and Layers). Places one can move or teleport to will show the navigation marker, whereas in no-go areas, the navigation ball is absent.

If an object is not designated as go or no-go, you can still move over it with the thumbsticks, but won't be able to teleport or move on that surface using the marker. You can fly above no go areas, but can't move onto them whilst on the ground (or at the level of the no-go area).

A typical scenario is to make paths one can move on, and ensure doors to buildings can be entered, but not be able to walk through walls. A common technique is to use a large surface to move around on and put this in Layer 7, then create small flat Colliders to put underneath objects you don't want to go through, slightly above the main ground object, and put these on Layer 8. An easy way to achieve this is to create Plane GameObject, remove the Mesh Renderer (unless you want to be able to see it), and adjust the Collider using the 'Edit Collider' button in the Inspector. The little points on the Collider lines can be interacted with using the mouse.

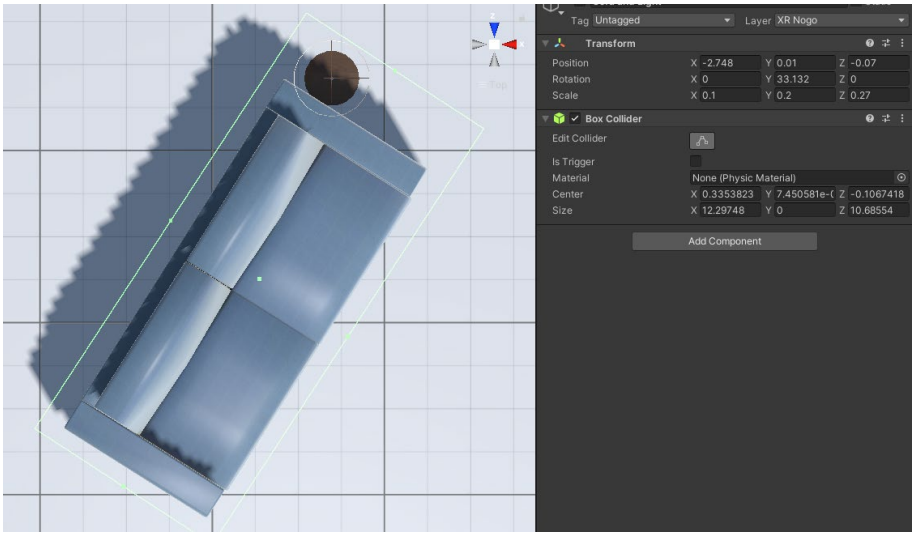


Figure 47 : Adjusting the collider of a no-go area under an object

You can use mesh colliders of larger objects as no-go colliders as well, but be warned that too many

mesh colliders could slow down the Virtual Environment, particularly on mobile headsets.

It is also possible to have a large no-go area, with a few designated places to move to on top. Whatever is the top layer looking down from the user's head height will determine the go/no-go capability. In the environment below, the user can not move onto the blue (no-go) area, but can hop over onto the green (go) cubes. The whole structure is on a larger plane which is a designated go-to area.

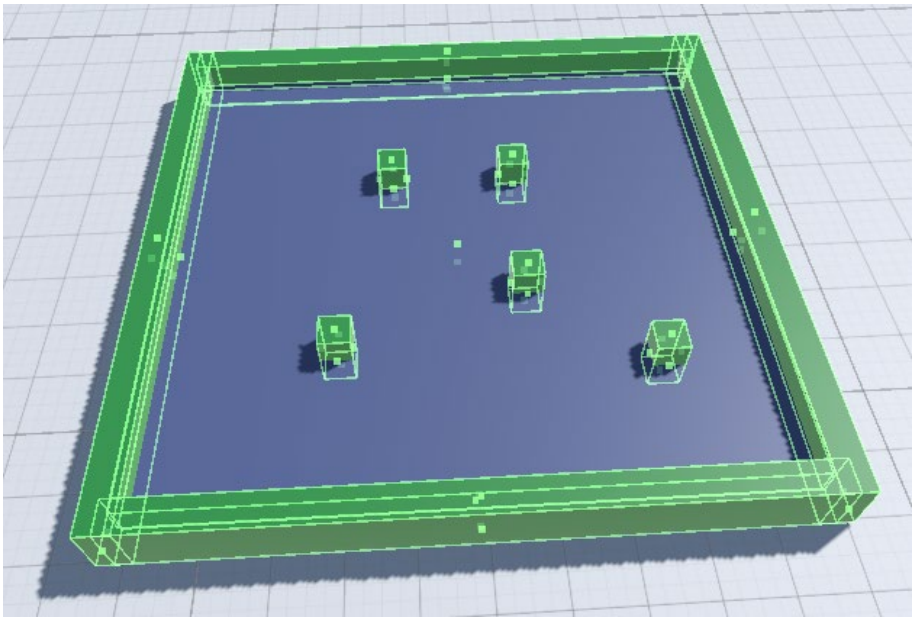


Figure 48 : Go (green) and No-go (blue) areas with the colliders highlighted

The movement using thumbsticks or move-to-marker is such that the view will decelerate as it approaches a no-go area so the stop is not too jarring.

Climbing and Falling

The user can be allowed to climb stairs or move onto higher (or lower) surfaces by creating Layer 7 objects higher up. It is recommended for stairs to add an invisible pathway angled up the steps rather than using the stair GameObject itself as the collider as the latter will result in jerky up and down motion.

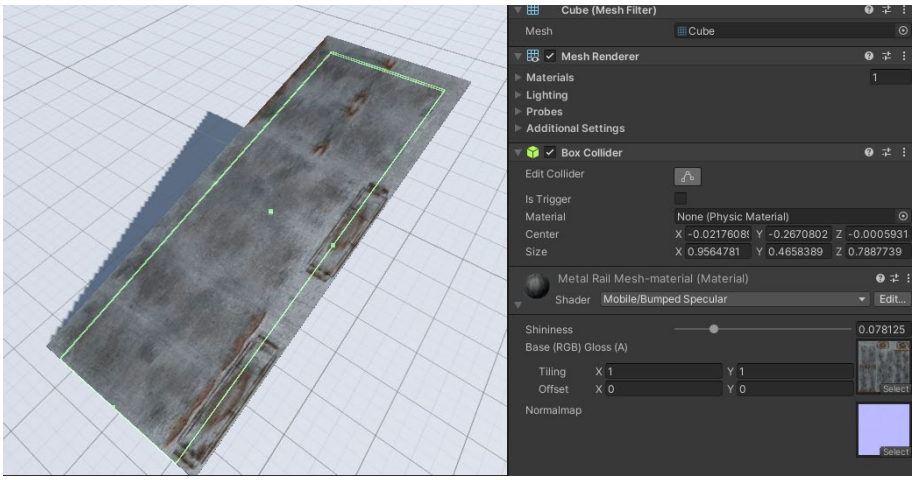


Figure 49 : A ramp the user can climb showing the box collider.

OpenXR UX Module Code Template

Each of the OpenXR UX Modules follows the same format:

1) Title and short description in the comments at the top.
2) All the various namespaces required.
3) Any public enumerations that are needed (for example, that might be required by other modules, or to be used in the inspector.
4) Class Interface with all the public functions – conveniently near the top.
5) The main class, consisting of:
    a. Public variables and useful information for the Inspector view.
    b. Private variables
    c. Various functions as required, including Start, FixedUpdate, Update and callbacks for being triggered by colliders.
    d. Private functions internal to this class.
    e. Public functions to implement the class interface.

Program for the XRUX_Button

```
/************************************************************************************************************************************************************************************
 * XRUX_Button
 * -----------
 *
 * 2021-08-25
 *
 * A generic button class inherited by other button classes that provide the core functionality
 *
 * Roy Davies, Smart Digital Lab, University of Auckland.
 ************************************************************************************************************************************************************************************/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.Events;

public enum XRGenericButtonAxis { X, Y, Z, None };
public enum XRGenericButtonMovement { Toggle, Momentary };

// ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
// Public functions
// ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
public interface _XRUX_Button
{
    void Title(string newTitle);            // Change the text on the button
    void Title(int newTitle);               // Change the text on the button
    void Title(float newTitle);             // Change the text on the button
    void Title(bool newTitle);              // Change the text on the button
```

```
    void Input(XRData newdata);             // Set the state of the button.  If quietly is set to true, doesn't invoke the callbacks.

    string Title();                         // Return the title of the button
}
// --------------------------------------------------------------------------------------------------------------------------------------------------------
```

The interface above defines two functions, Title (which is overloaded with several datatype parameters) and Input, which takes an XRData parameter).

```
// --------------------------------------------------------------------------------------------------------------------------------------------------------
// Main class
// --------------------------------------------------------------------------------------------------------------------------------------------------------
[AddComponentMenu("OpenXR UX/Objects/XRUX Button")]
public class XRUX_Button : MonoBehaviour, _XRUX_Button
{
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    // Public variables
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    [Header("_____________________________________________________________________________________________")]
    [Header("A movable button.\n_____________________________________________________________________________________________")]
    [Header("INPUTS\n\n - Title( [ int | float | bool | string ] ) - Set the button title.\n - Input( XRData ) - Boolean value to change the button state as if it
was being pressed.")]

    [Header("_____________________________________________________________________________________________")]
    [Header("SETTINGS")]
    [Header("The object that will change colour when pressed.")]
    public Renderer objectToColor;          // The object that needs to change colour when activated
    [Header("The object that will move when pressed.")]
    public GameObject objectToMove;         // The GameObject that will move when activated

    [Header("Materials for the different interactions states.")]
    public Material normalMaterial;         // The material for when not pressed
    public Material activatedMaterial;      // The material for when pressed
    public Material touchedMaterial;        // The material for when touched

    [Header("Movement Axis (or none), and amount")]
    public XRGenericButtonAxis movementAxis = XRGenericButtonAxis.Z;
    public float movementAmount = 0.004f;

    [Header("Button movement style")]
    public XRGenericButtonMovement movementStyle = XRGenericButtonMovement.Toggle;

    [Header("_____________________________________________________________________________________________")]
    [Header("OUTPUTS")]
    public UnityXRDataEvent onChange;       // Changes on click or unclick, with boolean
    public UnityEvent onClick;              // Functions to call when click-down
    public UnityEvent onUnclick;            // Functions to call when click-up
    public UnityXRDataEvent onTouch;        // Functions to call when first touched
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
```

The above section shows all the public variables and inspector comments and information.

```
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    // Private variables
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    private bool buttonState = false;       // Current button state
    private float touchTime;                // Time when last touched - used to make sure the button resets if touch-up doesn't occur - can happen occasionally.
```

```
    private Vector3 startPosition;          // Stores the start position at startup so it can be used for the 'off / out' position.
    private bool isLeft = false;            // Keeps tracks of whether the left or right controller has touched the button for when clicking occurs.
    private bool isRight = false;
    private bool touched = false;
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
```

After that comes the private variables that are only to be used in this class.

```
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    // Change the text on the button
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    public void Title(float newTitle) { Title(newTitle.ToString()); }
    public void Title(int newTitle) { Title(newTitle.ToString()); }
    public void Title(bool newTitle) { Title(newTitle.ToString()); }
    public void Title(string newTitle)
    {
        XRUX_SetText textToChange = GetComponentInChildren<XRUX_SetText>();
        if (textToChange != null) textToChange.Input(newTitle);
    }
    // ----------------------------------------------------------------------------------------------------------------------------------------------------



    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    // Change the state of the button
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    public void Input(XRData newData)
    {
        Set(newData.ToBool(), newData.quietly);
    }
    // ----------------------------------------------------------------------------------------------------------------------------------------------------



    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    // Return the title of the XR Radio Button
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    public string Title()
    {
        XRUX_SetText textToChange = GetComponentInChildren<XRUX_SetText>();
        return ((textToChange == null) ? "" : textToChange.Text());
    }
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
```

Next comes the implementation of the public functions named in the interface at the top.

```
        // ----------------------------------------------------------------------------------------------------------------------------------------------------
    // Save start position and material
    // ----------------------------------------------------------------------------------------------------------------------------------------------------
    void Awake()
    {
        if (objectToColor != null) objectToColor.material = normalMaterial;
        if (objectToMove != null) startPosition = objectToMove.transform.localPosition;
    }
```

```
// ---------------------------------------------------------------------------------------------------------------------------------------------------



// ---------------------------------------------------------------------------------------------------------------------------------------------------
// Set up the link to the event manager
// ---------------------------------------------------------------------------------------------------------------------------------------------------
void Start()
{
    // Listen for events coming from the XR Controllers and other devices
    if (XRRig.EventQueue != null) XRRig.EventQueue.AddListener(onDeviceEvent);
}
// ---------------------------------------------------------------------------------------------------------------------------------------------------



// ---------------------------------------------------------------------------------------------------------------------------------------------------
// If the touching object disappears before it stops touching, there is no OnTriggerExit event
// ---------------------------------------------------------------------------------------------------------------------------------------------------
void Update()
{
    if (((Time.time - touchTime) > 0.1) && touched)
    {
        if (movementStyle == XRGenericButtonMovement.Momentary)
        {
            touched = false;
            Set(false);
        }
        else
        {
            DoTouchExit();
        }
    }
}
// ---------------------------------------------------------------------------------------------------------------------------------------------------
```

Following are the main unity GameObject functions. Others such as FixedUpdate may be used as required. Note in this module, since it requires direct user interaction, a link to the main EventQueue which interprets the GameController signals into something more generic, is set up in the Start function. This can be copied into any other class that requires it.

```
// ---------------------------------------------------------------------------------------------------------------------------------------------------
// Set the button to the correct state when being activated in case it was not active and missed being moved or coloured
// ---------------------------------------------------------------------------------------------------------------------------------------------------
void OnDisable()
{
    touched = false;
}
void OnEnable()
{
    touched = false;
    if (movementStyle == XRGenericButtonMovement.Momentary)
    {
        Set(false, true);
```

```
        }
        else
        {
            Set(buttonState, true);
        }
    }
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
```

Some functions that define what happens when the button is enabled or disabled – not all classes require these.

```
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    // What to do when the button collider is triggered or untriggered (usually by the pointers).
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    void OnTriggerEnter(Collider other)
    {
        touched = true;
        if (other.gameObject.tag == "XRLeft") isLeft = true;
        if (other.gameObject.tag == "XRRight") isRight = true;
        if (objectToColor != null) objectToColor.material = touchedMaterial;
        if (onTouch != null) onTouch.Invoke(new XRData(true));
    }
    void OnTriggerStay(Collider other)
    {
        touched = true;
        if (other.gameObject.tag == "XRLeft") isLeft = true;
        if (other.gameObject.tag == "XRRight") isRight = true;
        touchTime = Time.time;
    }
    void OnTriggerExit(Collider other)
    {
        DoTouchExit();
    }
    private void DoTouchExit()
    {
        touched = false;
        if (movementStyle == XRGenericButtonMovement.Momentary)
        {
            Set(false);
        }
        else
        {
            if (buttonState)
            {
                if (objectToColor != null) objectToColor.material = activatedMaterial;
            }
            else
            {
                if (objectToColor != null) objectToColor.material = normalMaterial;
            }
        }
        isLeft = isRight = false;
        if (onTouch != null) onTouch.Invoke(new XRData(false));
    }
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
```

The set of three functions that are required to manager triggers (ie when one of the pointers touches the button). In this case, they define what happens when the user first touches the button, continues to touch the button, and stops touching the button. It also tracks which controller touched the button so later when a click event comes through, we can be sure it was the same controller.

```
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
// Once the button is touched, what to do when one of the triggers is pressed.
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
private void onDeviceEvent(XREvent theEvent)
{
    if ((((theEvent.eventType == XRDeviceEventTypes.left_trigger) && isLeft) || ((theEvent.eventType == XRDeviceEventTypes.right_trigger) && isRight)) &&
    (theEvent.eventAction == XRDeviceActions.CLICK) && touched)
    {
        if (movementStyle == XRGenericButtonMovement.Momentary)
        {
            Set(theEvent.eventBool);
        }
        else
        {
            if (theEvent.eventBool)
            {
                Set(!buttonState);
            }
        }
    }
}
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

The callback function as set up in the Start function to be get Unity events from the GameControllers.

```
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
// Set the button.  Can also be called from other functions via the Input function.
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
private void Set(bool newButtonState, bool quietly = false)
{
    buttonState = newButtonState;

    if (buttonState)
    {
        if (objectToColor != null) objectToColor.material = activatedMaterial;
        if (objectToMove != null)
        {
            switch (movementAxis)
            {
                case XRGenericButtonAxis.X:
                    objectToMove.transform.localPosition = new Vector3(startPosition.x + movementAmount, startPosition.y, startPosition.z);
                    break;
                case XRGenericButtonAxis.Y:
                    objectToMove.transform.localPosition = new Vector3(startPosition.x, startPosition.y + movementAmount, startPosition.z);
                    break;
                case XRGenericButtonAxis.Z:
                    objectToMove.transform.localPosition = new Vector3(startPosition.x, startPosition.y, startPosition.z + movementAmount);
                    break;
```

```
                    default:
                        break;
                }

            }
            if ((onClick != null) && !quietly) onClick.Invoke();
        }
        else
        {
            if (objectToColor != null) objectToColor.material = normalMaterial;
            if (objectToMove != null) objectToMove.transform.localPosition = startPosition;
            if ((onUnclick != null) && !quietly) onUnclick.Invoke();
        }

        if ((onChange != null) && !quietly) onChange.Invoke(new XRData(buttonState));
    }
    // ------------------------------------------------------------------------------------------------------------------------------------------------------------------------ }
```

Finally, the private function that does all the work of moving the button, changing colour and calling the click and unclick events.

Program for the XRData_Alternator Script

```
/*****************************************************************************************************************************************************************************
 * XRData_Alternator
 * -----------------
 *
 * 2021-09-05
 *
 * Alternates between true and false
 *
 * Roy Davies, Smart Digital Lab, University of Auckland.
 *****************************************************************************************************************************************************************************/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
// Public functions
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
public interface _XRData_Alternator
{
    void Go();
}
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Public interface

```

// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
// Main class
// ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
[AddComponentMenu("OpenXR UX/Connectors/XRData Alternator")]
public class XRData_Alternator : MonoBehaviour, _XRData_Alternator
{
    // ------------------------------------------------------------------------------------------------------------------------------------------------------------------------
    // Public variables
```

```
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    [Header("_____________________________________________________________________________________")]
    [Header("Toggles output between true and false.\n_____________________________________________________________________________________")]
    [Header("INPUTS\n\n - Go() - Trigger to activate function.")]

    [Header("_____________________________________________________________________________________")]
    [Header("SETTINGS")]

    [Header("_____________________________________________________________________________________")]
    [Header("OUTPUTS")]
     public UnityXRDataEvent onChange;
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
```

Public variables – not much in this case – just the XRData Event to pass on the new value.

```
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    // Private variables
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    private bool currentValue = false;
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
```

Private variables – in this case to hold the current Boolean value.

```
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    // Send the alternating true or false value
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
    public void Go ()
    {
        currentValue = !currentValue;
        if (onChange != null) onChange.Invoke(new XRData(currentValue));
    }
    // ------------------------------------------------------------------------------------------------------------------------------------------------------
}
```

And finally, the main public function that is called whenever you want to alternate between true and false, and send an event to another XRModule. IN this module, there are no private functions.

Conclusion

Congratulations, you have reached the end of this guide. The following sections contain some tutorials.

Tutorials

TODO…

Color cube Demo

In this tutorial, we are going to create an experience whereby you can change the colour of an object in the Virtual Environment by turning some knobs on its surface.