XR Custom Framework

Contents

XRCustomFramework contains following parts:

Introduction

XRCustomFramework uses Unity XR API and other SDK which does not support XR API for most of the VR and MR devices.

Requires the Unity version 2019.1.1f1 (or above)

Setting up the project

- Create a new project in the Unity software version 2019.1.1f1 (or above) using 3D Template or open an existing project.
- ~~Create new project and download XRCustomFramework project and copy/paste in empty project. Or copy/paste in your existing project.~~
- Ensure Virtual Reality Supported is checked:
- In the Unity software select Main Menu -> Edit -> Project Settings to open the ProjectSettings window.
- Select Player from the left hand menu in the Project Settings window.
- In the Player settings panel expand XR Settings.
- In XR Settings ensure the Virtual Reality Supported option is checked.
- Ensure the project Scripting Runtime Version is set to .NET 4.x Equivalent:
- In the Unity software select Main Menu -> Edit -> Project Settings to open the Project Settingsinspector.
- Select Player from the left hand menu in the Project Settings window.
- In the Player settings panel expand Other Settings.

Note: Unity 2019.1 requires additional project setup before importing VRTK.

- Download and install the XR Legacy Input Helpers from the Unity Package Manager.
- In the Unity software select Main Menu -> Window -> Package Manager to open the Package Managerwindow.
- Select XR Legacy Input Helpers from the Packages tab in the Package Manager window.
- Click the Install button located in the bottom right of the Package Manager window.
- The XR Legacy Input Helpers package will now download and install into the project.

Importing XRCustomFramework

- Download the XRCustomFramework project from Git and Paste inside the project Assets/ directory.
- Or clone XRCustomFramework sub module using ***git submodule update --init --recursive***

Build Settings

We have customised the build setting by using build pre prosser script for building SDK which is not supported in Unity XR and some user interface settings before taking build.

To open Build Settings Window In the Unity software select XR Framework -> Build Settings

- Selecting SDK for build
    - None : Use this option for normal building without using Virtual Reality (XR) support.
    - XR : Use this option for building Virtual Reality (XR) project.
    - Wave VR : Use this option for taking build only for Wave VR Android build (Note: Only Android build is supported for Wave VR).
- Selecting VR SDK which to be included in build. [***Not yet implemented***]
- Build button to take build according to option selected. [***Not yet implemented***]

More SDK option will be added in future according to requirement.

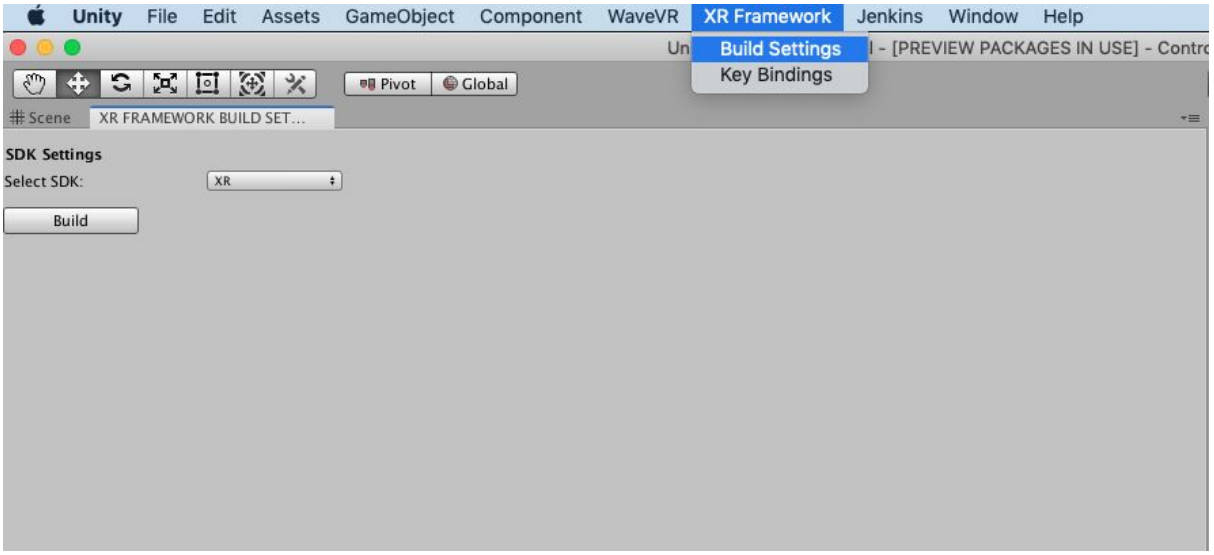

Figure 1. Build Settings Window

Building Settings API Reference

- BuildSettings : BuildSettings class is used to create Editor Window. Create own custom editor window that can float free or be docked as a tab, just like the native windows in the Unity interface.
- BuildPreprocessor : BuildPreprocessor class include IPreprocessBuildWithReport interface which are used to receive callback before the build is started. BuildPreprocessor helps to configure the SDK before creating build according to the SDK option selected.
- BuildSettingAssetsHolder : This class is Scriptable Object which keeps the information of XRCustomFramework build settings.

XR Framework

For XR Framework we are using Unity XR. [XR is an Umbrella term, encompassing Virtual Reality (VR), Augmented Reality (AR) and Mixed Reality (MR) applications].

XR Framework API Reference

- SDK Setup : SDKSetup class helps to initialize the Scene objects according to device platform and Virtual Reality support.
  SDKSetup class check the SDK type from Build Settings, for now we are using:
  - ❖ XR : For initializing Unity XR supported devices.
  - ❖ NonVR : For normal device without VR setup.
  - ❖ WaveVR : For Focus device which doesn't support XR.
- InputManager :
  This class is used for getting all Input for XR input devices and other SDK input Devices. We can either use Event to get input or directly use method to get particular input, for example see below Scripting.

Scripting

Static Methods:

IsHanded : Checks Left or Right controller is active. Return bool value
GetHandController : Returns Object. [*GetHandController(Hand)*];
GetPreferedHand : Returns Left/Right Hand default Right Hand
GetButton : Check Button is pressed, return true if pressed.

bool isPressed = InputManager.GetButton(FeatureUsageButton, Hand);

GetAxis : Get float value of button squeezed.

float value = InputManager.GetAxis(FeatureUsageAxis, Hand);

Get2DAxis : Get float value of button squeezed.

vector2 vector = InputManager.Get2DAxis(FeatureUsage2DAxis, Hand);

Events:

OnButtonPressed : Called when controller button is pressed/touched.
bool : True/False
Hand : Left/Right
FeatureUsageButton : Button Type
OnAxis : Called when controller button is squeezed.
float : Button squeezed value.
Hand : Left/Right
FeatureUsageAxis : Axis Type
On2DAxis : Called when controller Joystick is used.
vector2 : Joystick value.
Hand : Left/Right
FeatureUsage2DAxis : 2DAxis Type

- XR_Manager :
  - This class will represent the type of physical space available for XR.
  - Automatically set the Unity Physics Fixed Timestep value based on the headset render frequency.
  - Holds the reference for Play Area, Headset, Headset Camera, Left and Right Hand game object.
- XR_InputHandler :
  - This class keeps track of all XR api related Input devices connection and disconnection.
  - Handles left and right hand controller [XR_Controller].

Scripting

Public Methods:

IsHanded : Checks Left or Right controller is active. Return bool value
GetHandController : Returns XR_Controller. [*GetHandController(Hand)*];
GetPreferedHand : Returns Hand default Right Hand

- XR_Controller :
    - This class is the base of all Input controller which is used for getting input from the hardware controller and trigger Input Manager.
    - Instantiate the controller model [Model Controller] according to device type.
- Model Controller :
    - Handles model button animation.
- AssetsReference : This class is Scriptable Object which holds the reference of Controllers prefab.
- XR_Utilities : Utilities helper class.
- XR_Enum : Enumerations
    - Hand
        - Left : Left hand side controller
        - Right : Right hand side controller
    - InputType
        - None
        - Bool : bool button pressed/touched
        - Axis1D : float button axis
        - Axis2D : vector2 joystick axis
    - FeaturesUsageButton
        - None,
        - PrimaryButton
        - PrimaryTouch
        - SecondaryButton
        - SecondaryTouch
        - GripButton
        - TriggerButton
        - MenuButton
        - Primary2DAxisClick
        - Primary2DAxisTouch
        - Thumbrest
    - FeaturesUsageAxis
        - None
        - Trigger
        - Grip
        - IndexTouch
        - ThumbTouch
        - IndexFinger
        - MiddleFinger
        - RingFinger
        - PinkyFinger
        - CombinedTrigger
    - FeaturesUsage2DAxis
        - None
        - Primary2DAxis

        - Secondary2DAxis
    - Device
        - None
        - Vive
        - Oculus
        - GearVR
        - WMR
        - OpenVR_Full
        - OpenVR_Oculus
        - OpenVR_MWR
- ConstantVar : Is static class which holds all constant variables.

Key Binding

Created a Unity editor window for binding controllers buttons and joysticks which helps user to easily bind the key from drop down list according to Device.

Key Binding API Reference

- KeyBindingWindow : This class is used to create Editor Window for user interface.
    - Controller Tab : Can add XR default controller for Left and Right controller. [Work in progress for other SDK like Wave VR]
    - Controller Button Tab : This window shows the details of buttons and joysticks of each Controller added in [Controller Tab]. In this window user can bind the key of controller.
- KeyBindingData : This class is Scriptable Object which keeps the information of controllers and their Key assigned.

To open Build Settings Window In the Unity software select XR Framework -> Key Bindings

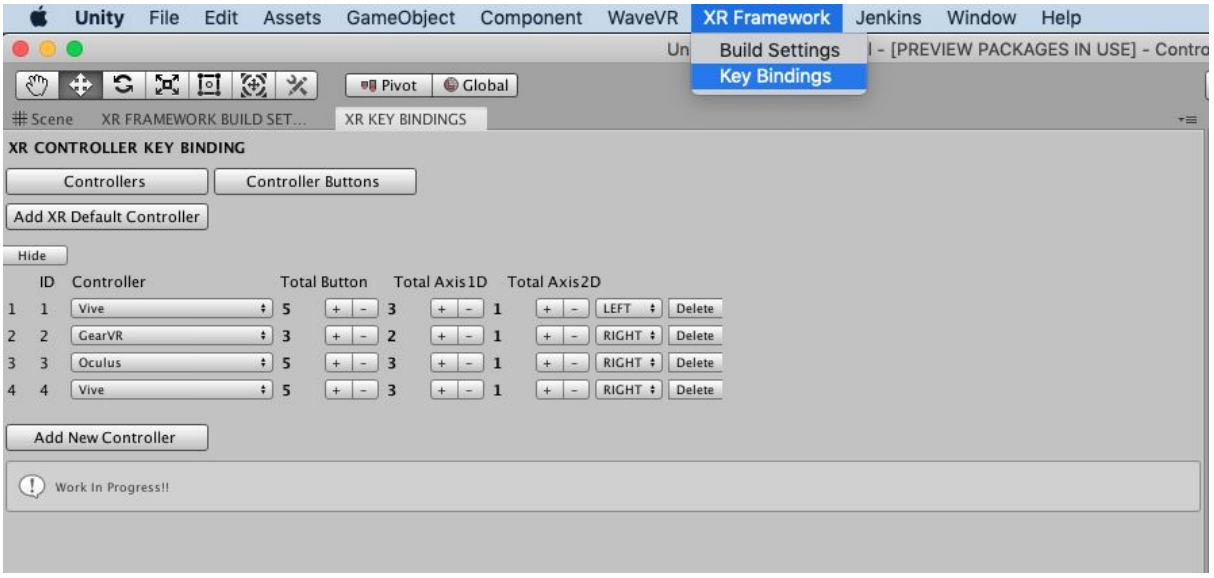

Figure 2. Key Bindings Window

Key Binding Window

Figure 2 shows the key binding window, which contains the following features:
[Note: Editor Window is not designed properly later we can redesign with proper UI]
For Key Binding we are following default **Unity XR Input mapping**.

Controllers Tab

In Controller Tab we can add new Device controller which is also divided in two parts. Add XR Default Controller and Add New Controller.

Add XR Default Controller

Is for adding Unity XR supported SDK controller only. Supported device controller are. Link

Add New Controller [***Work in progress***]

Is for adding other SDK controller like WiveVR.

Controller contains System generated ID, default device name, total keys and controller hand side Left/Right.

Controllers Buttons

In Controller Buttons we can Bind the key according to controller. We can select the controller by pressing the Left/Right button.

The button name shown in left side is Feature usage name used in XR and right side name is Controller input name according to controller type, check Unity XR Input Mapping.

Unity XR Input Mapping

Feature Usage	Feature Type	Legacy Input Index [L/R]	WMR	Oculus	GearVR	Daydream	OpenVR (Full)	Vive	OpenVR (Oculus)	OpenVR (WMR)
Primary 2DAxis	2D Axis	[(1,2)/(4,5)]	Joystick	Joystick	Joystick	Touchpad	[Trackpad/Joystick]	Trackpad	Joystick	Joystick
Trigger	Axis	[9/10]	Trigger	Trigger	Trigger	Trigger	Trigger	Trigger	Trigger	Trigger
Grip	Axis	[11/12]	Grip	Grip		Grip	Grip	Grip	Grip	Grip
IndexTo	Axis	[13/14]		Index -						

uch				Near Touch						
Thumb Touch	Axis	[15/16]		Thumb - Near Touch						
Secondary2DAxis	2DAxis	[(17,18)/(19,20)]	Touchpad							Touchpad
IndexFinger	Axis	[21/22]					Index			
MiddleFinger	Axis	[23/24]					Middle			
RingFinger	Axis	[25/26]					Ring			
PinkyFinger	Axis	[27/28]					Pinky			
CombinedTrigger	Axis	[3/3]	CombinedTrigger	Combined Trigger	Combined Trigger		Combined Trigger	Combined Trigger	Combined Trigger	
Primary Button	Button	[2/0]		[X/A]		App	Primary	Primary	Primary [Y/B]	Menu
Primary Touch	Button	[12/10]		[X/A] - Touch						
SecondaryButton	Button	[3/1]		[Y/B]			Alternate		Alternate [B/A]	
SecondaryTouch	Button	[13/11]		[Y/B] - Touch						
GripButton	Button	[4/5]	Grip - Press	Grip - Press		Grip - Press	Grip - Press	Grip - Press	Grip - Press	Grip
Trigger Button	Button	[14/15]	Trigger - Press	Index - Touch	Trigger - Press	Trigger - Press	Trigger - Press	Trigger - Press	Trigger - Touch	Trigger-Press
MenuButton	Button	[6/7]	Menu	Start (6)						
Primary 2DAxis Click	Button	[8/9]	Touchpad - Click	Thumbstick - Click	Touchpad - Click	Touchpad - Click	StickOrPad - Press	StickOrPad - Press	StickOrPad - Press	Touchpad - Click
Primary 2DAxis Touch	Button	[16/17]	Touchpad - Touch	Thumbstick - Touch	Touchpad - Touch	Touchpad - Touch	StickOrPad - Touch	StickOrPad - Touch	StickOrPad - Touch	Touchpad - Touch
Thumbrest	Button	[18/19]	Joystick - Click	Thumb Rest - Touch						

Unity Supported Device

Oculus

Google Cardboard

HTC Vive

Sony PlayStation

Samsung Gear VR

Microsoft Mixed Reality

Google Daydream