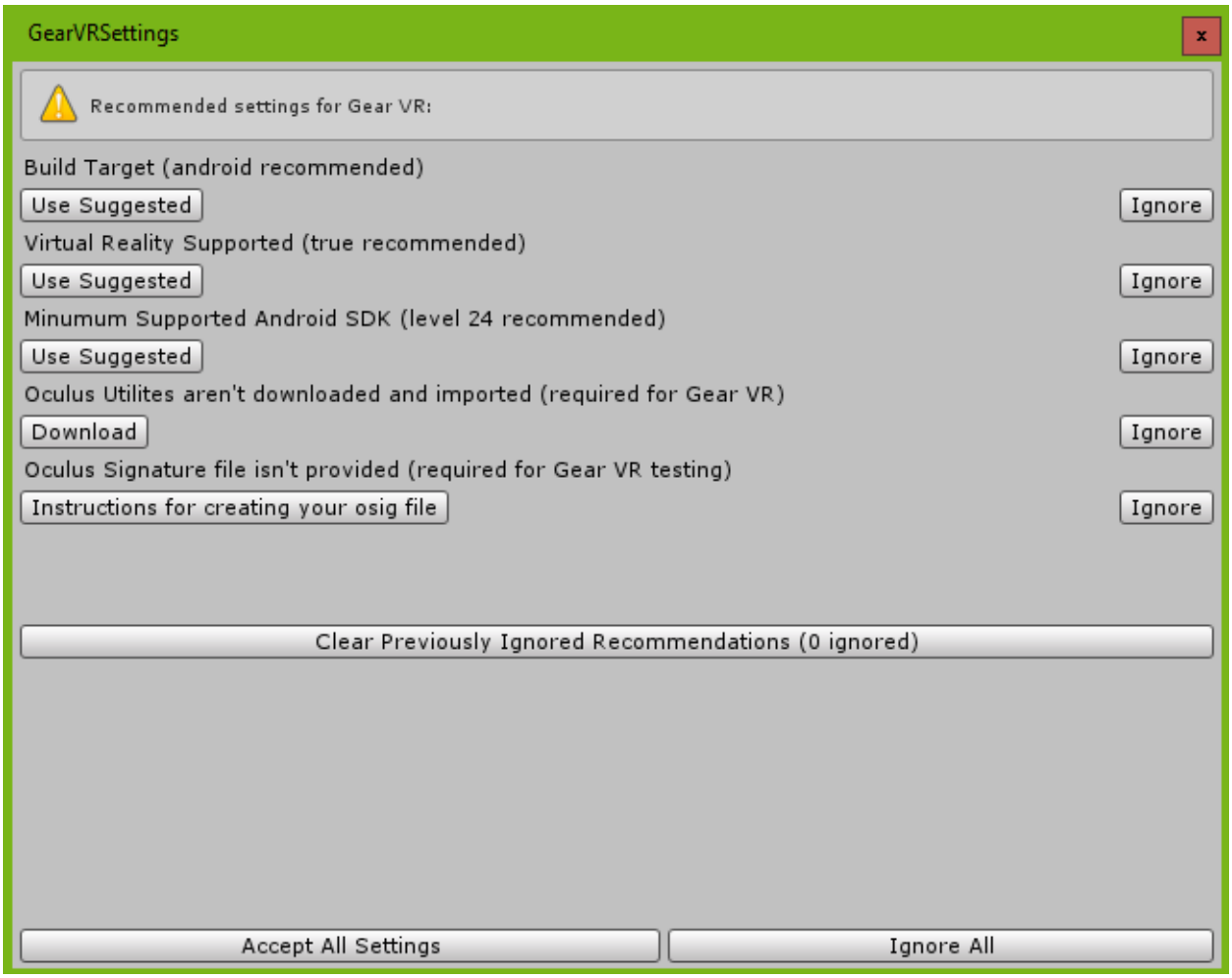
Easy Input For Gear VR Documentation

First Things to do

When you first import Easy Input into your project you will be greeted with an editor window that contains recommended settings as well as links to the Oculus Utilities which are required. All of these are highly recommended or required for developing for the Gear VR. Click the appropriate buttons to change settings automatically or to open a web page for required downloads. If you want to ingore the recommendations you can simply click the ignore button and proceed like normal.



Introduction

Easy Input for Gear VR makes supporting input for the Gear VR and the new Gear VR Controller a breeze. Whether you want support for the headset, Bluetooth controllers or the motion controller, everything that is unique about the platform is accessible in one easy to use API. Many common tasks even expose high level components to attach to your objects so you don't need to write a single line of code. The versatility is left in though so if you do want to write custom code it is easy to do so.

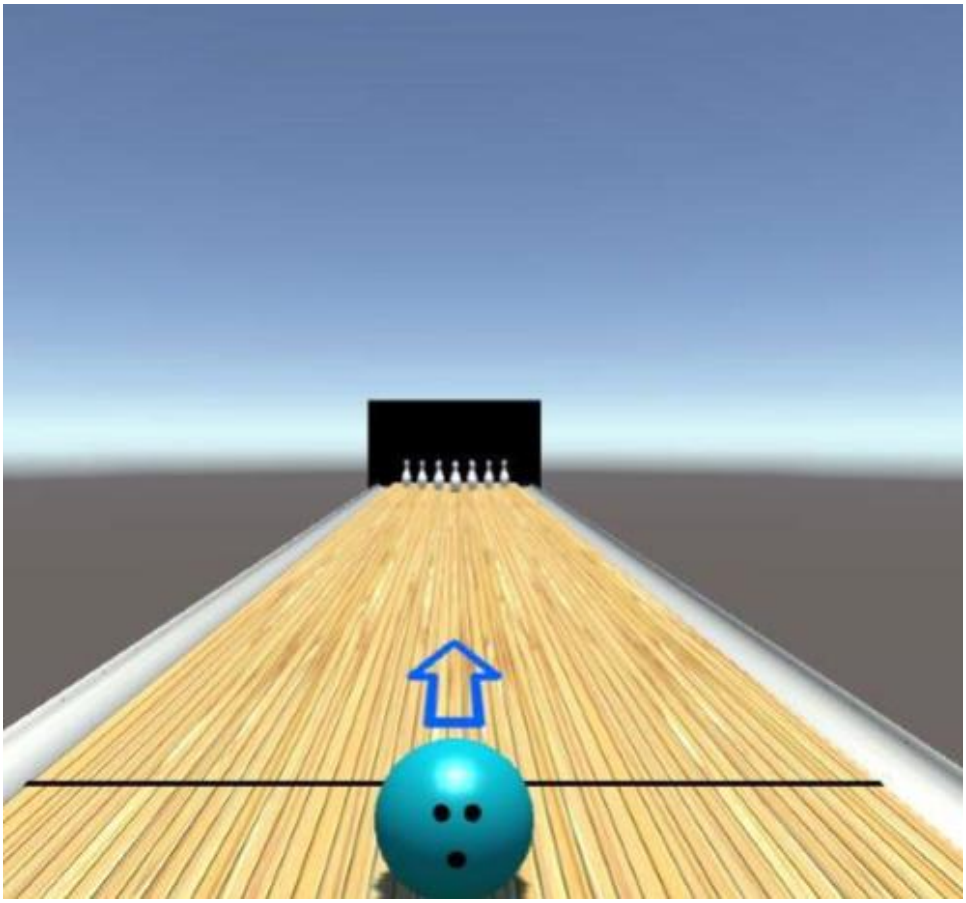
Features

- Full Gear VR headset support
- Full Gear VR controller support
- Full Bluetooth or USB gamepad support
- Standard controls for common things so no coding required
- Grabbing, moving, and many more actions made easy
- Nice high level API (quick click, long click, double click, touch, etc.)
- Emulates input in editor so no more wasting time doing a build on every change

- Motion support
- Laser Pointer input module for Unity GUI support with no coding required
- 8 example scenes
- Easy callbacks to use if you want to do something custom

Specific Example scenes

Bowling example-



In this scene you are presented with a functional bowling example. This uses our motion API to allow you to throw the ball like you might in real life. Left/Right swipes changes the launch location, prior to throw tilting changes the aim. After click the pad to start the throw and let go to throw the ball. While the pad is clicked do a normal bowling throw and hardness and spin will be determined from you motion.

Tilt Gear VR Controller Example-

In this scene you are presented with a simple marble style game. Simply tilt the Gear VR controller to steer the marble. This example showcases motion support and how are standard controls take care of the coding for you. This showcases a style of game that fits the new Gear VR motion controller well!

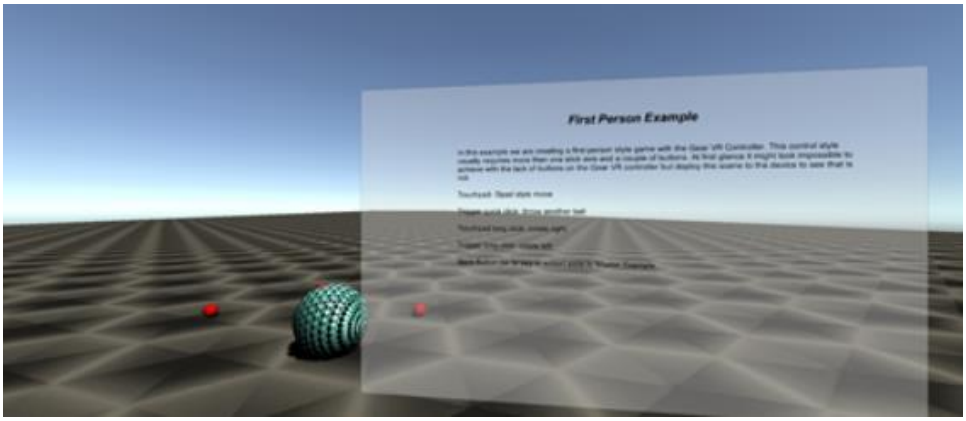
Controls example-

In this scene you have a wide variety of standard controls that showcase what you can do with no coding required. Use the HMD or Gear VR Controller touch surfaces as a touchpad, use the sticks on a bluetooth or USB controller, call functions on button presses, or long touch, or double touch, and many more! This also shows that the product contains high level functionality like quick, long, and double presses (for either the touchpad or button clicks). We also provide this functionality for swiping. This allows you to overcome the lack of usable buttons on the Gear VR controller, simply map one action to a single press, another to a long press, and another to a double press all on the same button!

GUI Navigation example-

In this scene you are presented with a typical Unity UI with a grid of buttons, checkboxes, scrollbars, and other common Unity GUI controls. This example showcases our Easy Input Module which provides a natural way to navigate Unity UI's regardless of whether you're using the HMD, motion controller, or conventional controller. You'll notice the nice laser pointer navigation and swipe style navigation is included without having to code anything. Furthermore, in addition to this the same input module also supports bluetooth controllers automatically at the same time.

First Person example-



In this scene you are presented with a typical first person controller scheme adapted to the Gear VR controller. Use the touchpad as a dpad to move the character around. Long click the trigger to rotate left, long click the touchpad button to rotate right, and quick click the trigger button to shoot.

Gamepad Controller diagnostic example-

In this scene you are presented with a real time view into our gampad controller API. This demonstrates when the events are fired off to give you a better idea what happens when you click a button or move a stick. This also showcases that not just the touchpad has the ability for long presses or double clicks. This will help you wrap your head around when the events are fired to the callbacks if your trying to do something advanced.

GVR controller diagnostic example-

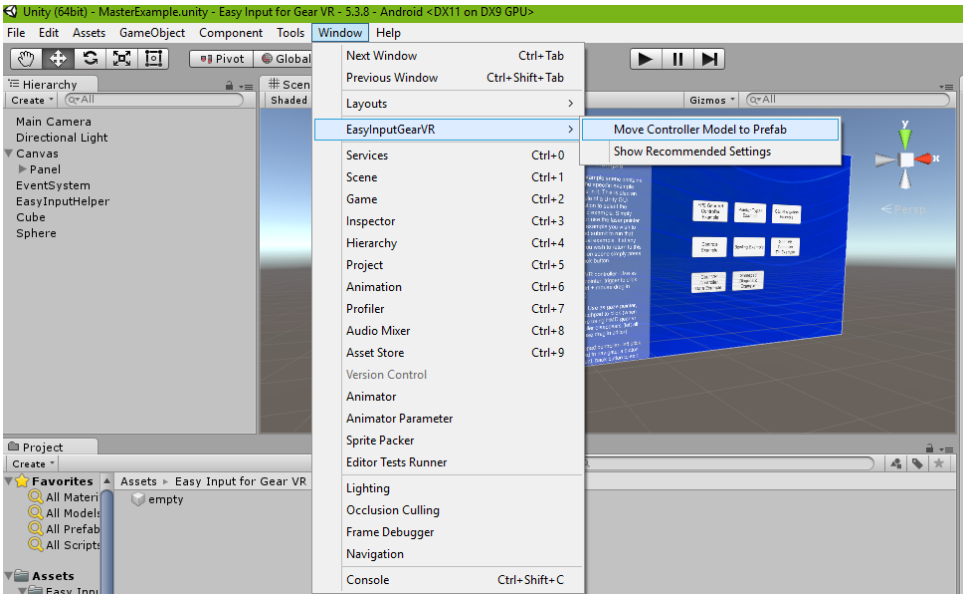
In this scene you are presented with a real time view into our Gear VR controller and HMD API. This demonstrates when the events are fired off to give you a better idea what happens when you touch the pad or click a button. This also showcases the motion telemetry to help you visualize the data that is coming in. This is useful if you want to model a specific motion and look at key points that you can base the input of your game off of. In addition to the raw orientation that is provided by the motion controller, Easy Input calculates useful derivative information like velocity and position. It's certainly fine for a quick motion but not any longer than a few seconds (more sensors like on the Rift or the Vive are required for full positional tracking). If you want to do motion controls its always helpful to practice your motion and look at the telemetry to see if there is something useful you can base it on.

Pointer Types example-

In this scene you are presented with a basic introduction into all our pointers and receivers. You can grab objects, teleport, call custom methods, etc. When being interacted with the pointers. You can pick from a laser pointer, gaze pointer, or curved laser pointer by clicking on the buttons on a provided panel in realtime.

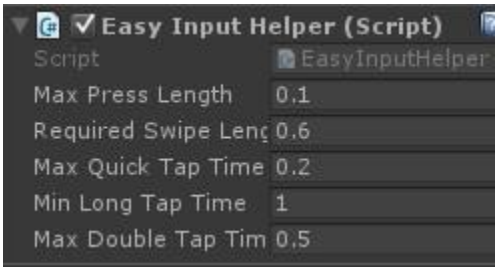
Using Model from Oculus SDK

Easy Input allows you to supply any model you choose as the hand avatar for the controller that is displayed to you in VR or the editor. If you wish the example scenes to use the high poly controller model provided by Oculus then simply select Window -> EasyInputGearVR -> Move Controller Model to Prefab. It will then be included in the scenes. The examples without the model just use a cube as a placeholder but you can certainly provide your own graphics without issue for whatever suits your game.



Easy Input Helper

The Easy Input Helper is a singleton class that needs to be placed into your scene in order to use our product. It contains global settings that dictate how you want your players to hold the remote, the timings for double clicks and other settings. Generally speaking you also want to place OVRManager on this object as well from the Gear VR SDK unless you already have it somewhere else in your scene.



If you want to use Easy Input Helper simply place one in your scene by the Gameobject -> Easy Input Helper -> Add Easy Input menu. As you can see this allows you to tune how you want your game to fire off the appropriate events. It's basically a single place to tune all of the settings to your liking so that it best matches the game your trying to make.

Max Press Length- The farthest you can move on the touchpad and fire off a press event (quick press, long press, double press). If you move farther than this you are on your way to a swipe event instead of your typical press.

Required Swipe Length- The distance you need to move to fire off the swipe event. You can swipe left, right, up, or down

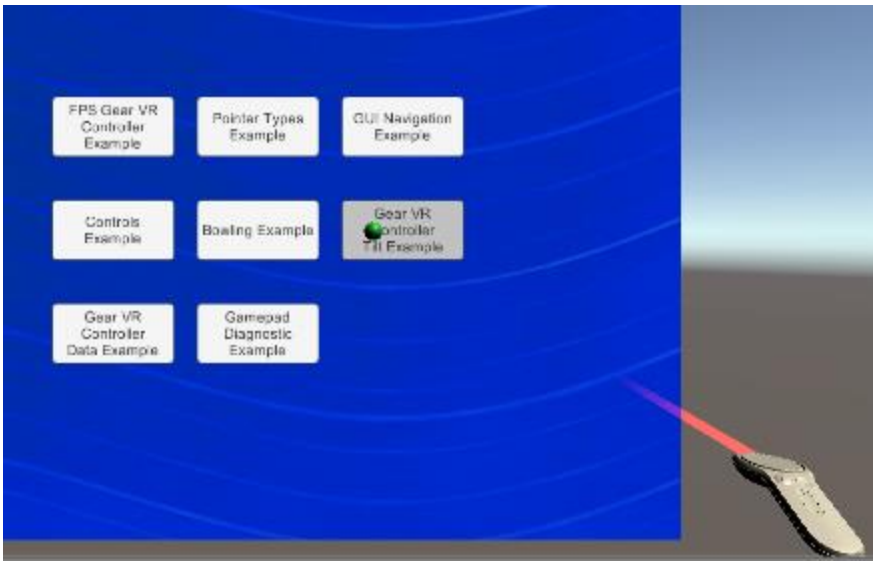
Max Quick Tap Time- The longest amount of time you can touch the pad and have it register as a quick press.

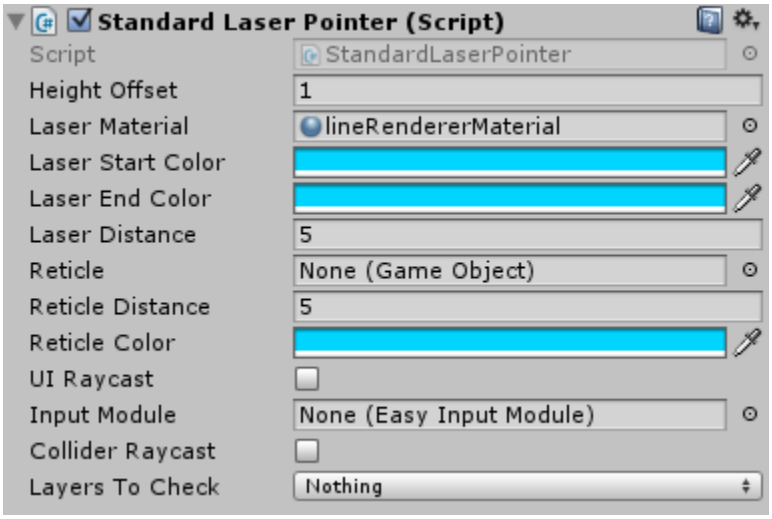
Min Long Tap Time- The amount of time you have to touch the pad and have it fire off a long press event

Max Double Tap Time- The longest amount of time where you can press twice quickly and have it register as a double press

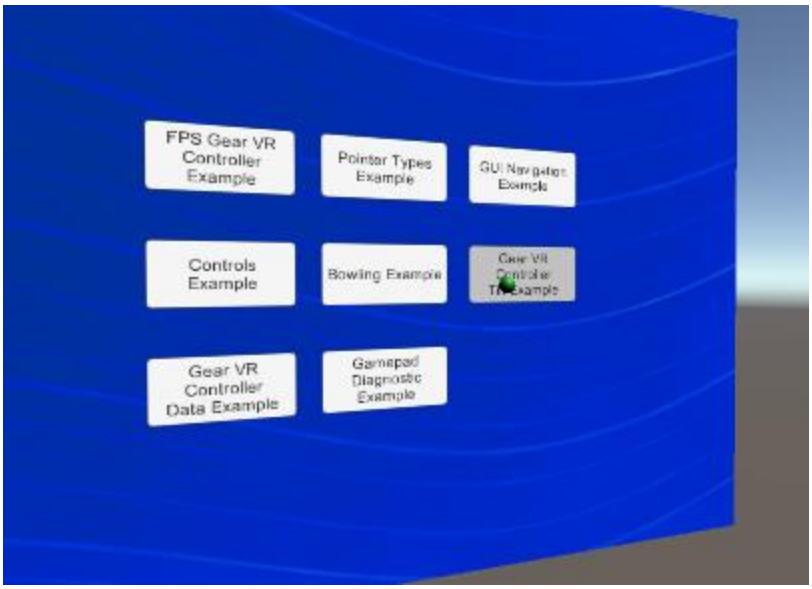
Pointers

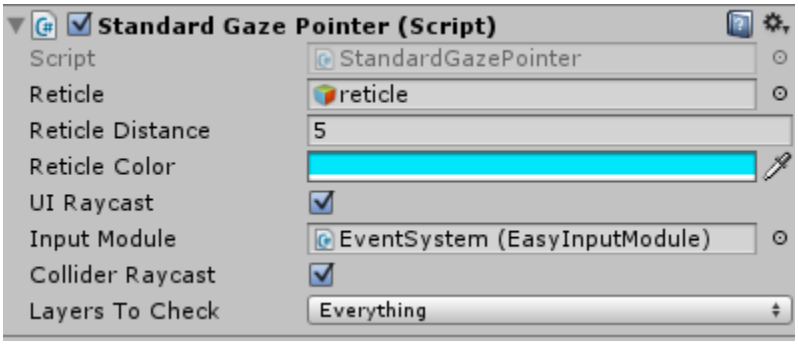
Standard Laser Pointer- Your typical pointer for the new Gear VR controller.





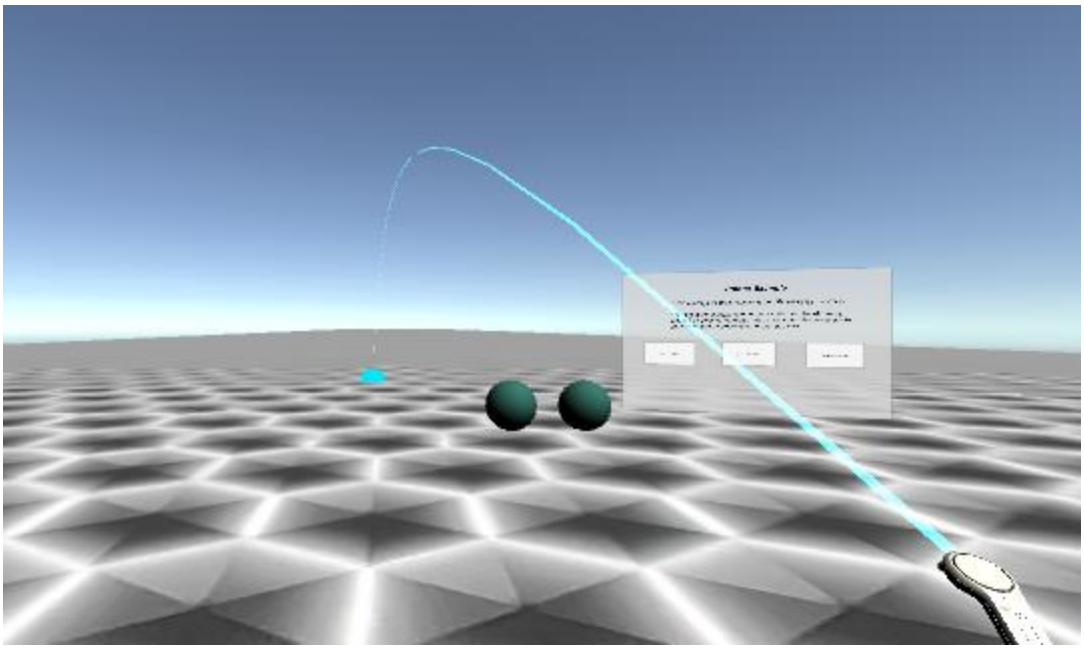
Standard Gaze Pointer- Used if you want a pointer based on your vision from the HMD.

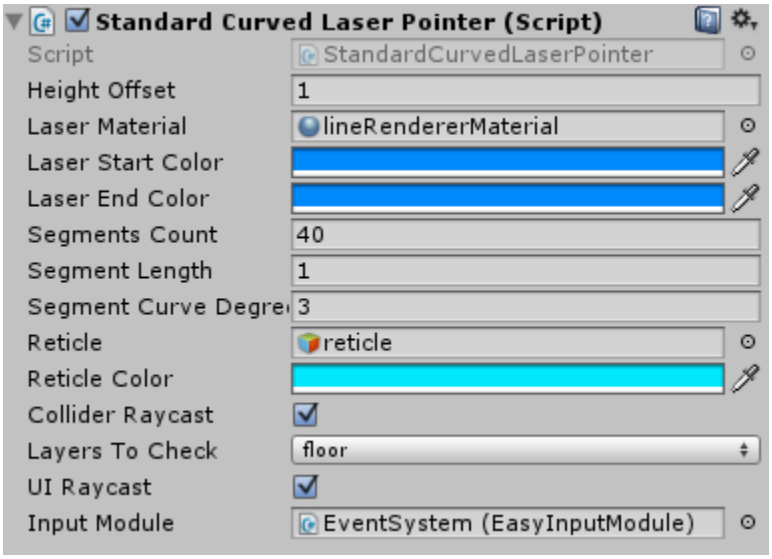




Standard Combo Pointer- Simulates the pointer for Oculus Home. It is a combination of the two above and will be a gaze pointer if you last touched the HMD touchpad or if you have no controller connected and will be a laser pointer if you've last touched the controllers touchpad.

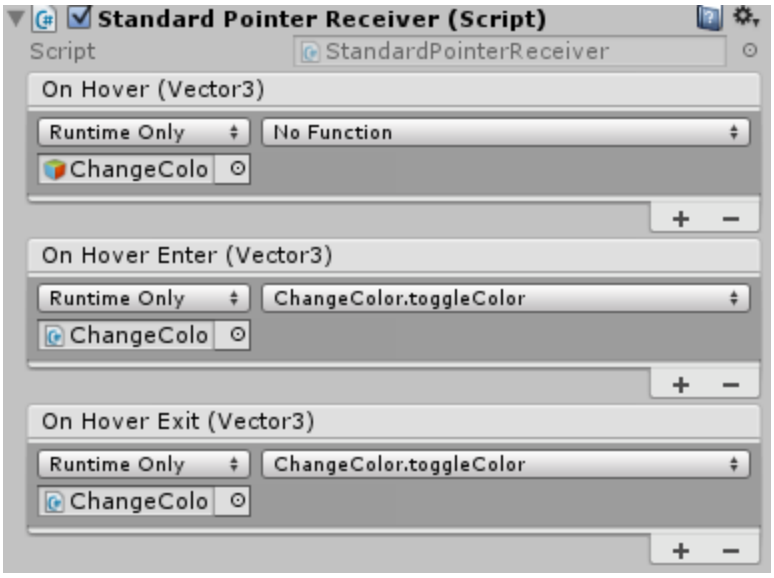
Standard Curved Laser Pointer- Similar to the laser pointer but is best for teleporting due to options to curve the laser.





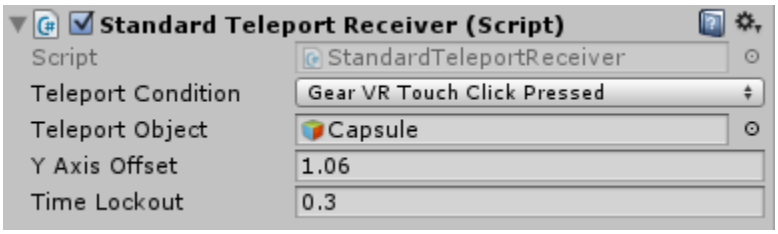
Receivers

Standard Pointer Receiver- Your basic receiver. Allows you to specify a function to be called when you Hover over an object with a pointer. Simply place this on any object you want to do something with the pointer.

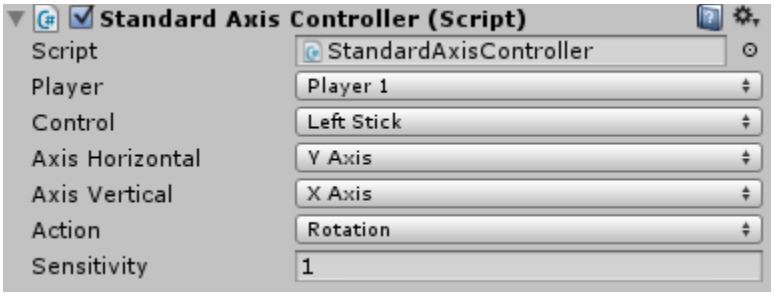


Standard Grab Receiver- This receiver allows an easy way to grab objects.

Standard Teleport Receiver- This receiver allows an easy way to teleport.

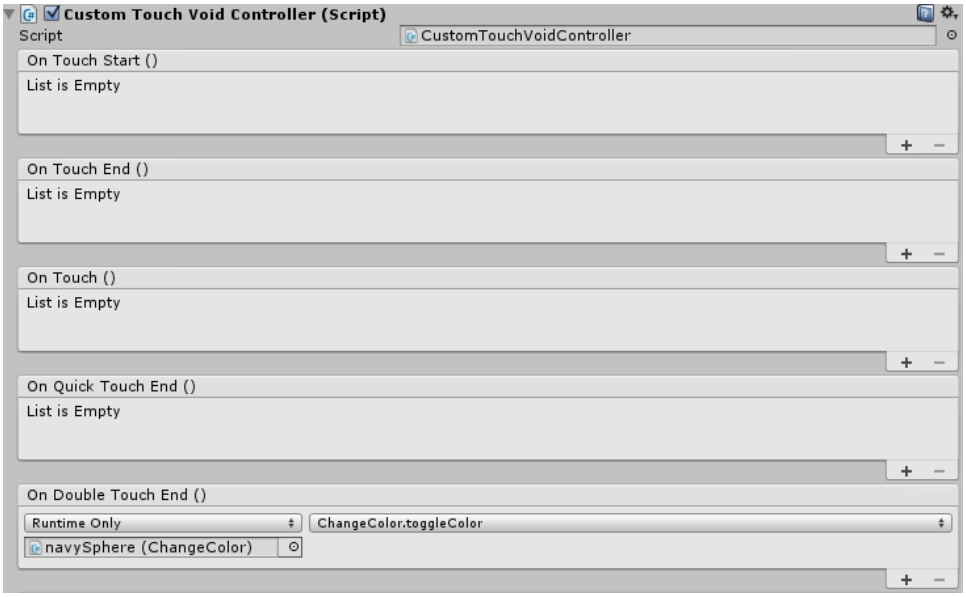


Standard Controls



Easy Input helper comes with many other standard controllers that makes it dead simple to do common things like move position, rotate, scale, etc. Above is the standard axis controller but there are also standard controllers for the touchpad and tilt. In each standard controller you have simple dropdowns that will dictate how it will affect the object it's attached to. In the above example when you hit the left stick on player 1 it will rotate on the global y axis (spin horizontally) when you move horizontally. Also, it will spin on the global X axis (spin forward/backward) when you move the left stick up and down. On each axis you can choose to affect local to the object, globally, or select none if you only want one axis affected. With these combinations you can do much of what you want without having to write any code!

Custom Controls



Easy Input comes with many custom controls. When you attach these to an object you are presented with the list of event subscriptions for the type of object it is (touchpad, axis, button, motion, etc.). Basically this GUI allows you to call any method you want straight from the inspector. Want the player to jump when you first touch the pad? No problem just hit the '+' button for On Touch Start and select your jump method and your done. Jump will now be called when you touch the Pad. This way of subscribing to events is much more user friendly to non programmers and yet you can still call custom code. If you are a programmer and want full control manually subscribing to callbacks is for you which is covered later in this document.

Easy Input Module

Easy Input module is very straightforward to use. Simply add the Easy Input module component to the EventSystem Object of your GUI via Add Component -> EasyInputforGearVR -> Input Modules -> Easy Input Module. That's all there is to it and your GUI will now be a breeze to use!

Repeat Event Rate- If you hold a button or direction (like the dpad) it's how quickly the event is repeated (ex. Navigating left)

Button Mode- When you hit a button whether it fires at button down, button up, or repeats at the repeat rate

Scroll Amount (Only used touchpad not laser pointer)- The number of divisions a scrollbar, slider, or scrollview should be divided into. .01 (100 divisions) is default and is good for most scrollbars to have smooth scrolling (versus the step scrolling from default unity). If you want more or less divisions adjust this number

Scroll Speed Multiplier (Only used touchpad not laser pointer)- Default is usually good, but if want scrolling to be faster (less swiping required) then increase this multiplier

Subscribing to callbacks manually

If you are a programmer and want to subscribe to the callbacks manually this is also an option. The list of callbacks are below.

```
//events
//touch
public static event onTouchStartHandler On_TouchStart;
public static event onTouchHandler On_Touch;
public static event onTouchEndHandler On_TouchEnd;
public static event quickTouchEndHandler On_QuickTouchEnd;
public static event longTouchStartHandler On_LongTouchStart;
public static event longTouchHandler On_LongTouch;
public static event longTouchEndHandler On_LongTouchEnd;
public static event doubleTouchEndHandler On_DoubleTouchEnd;
public static event swipeDistanceHandler On_SwipeDetected;

//buttons
public static event onClickStartHandler On_ClickStart;
public static event onClickHandler On_Click;
public static event onClickEndHandler On_ClickEnd;
public static event quickClickEndHandler On_QuickClickEnd;
public static event longClickStartHandler On_LongClickStart;
public static event longClickHandler On_LongClick;
public static event longClickEndHandler On_LongClickEnd;
public static event doubleClickEndHandler On_DoubleClickEnd;

//axis
public static event onAxisHandler On_LeftStick;
public static event onAxisHandler On_RightStick;
public static event onAxisHandler On_Dpad;
public static event onAxisHandler On_LeftTrigger;
public static event onAxisHandler On_RightTrigger;

//motion
public static event AccelerometerHandler On_Accelerometer;
public static event GyroHandler On_Gyro;
```

public static event MotionHandler On_Motion;

Each of these callback are pretty self explanatory and are fired off when appropriate if you want to know the exact timings simply run the 2 diagnostic examples. The touch events pass an InputTouch object, button events pass a ButtonClick object, axis pass a ControllerAxis object, and Motion passes a Motion object. You can easily subscribe to the events in any monobehaviour you've created as follows.

```
void OnEnable()
{
    EasyInputHelper.On_Accelerometer += localAccelerometer;
    EasyInputHelper.On_ClickStart += localClickStart;
    EasyInputHelper.On_ClickEnd += localClickEnd;

}

void OnDestroy()
{
    EasyInputHelper.On_Accelerometer -= localAccelerometer;
    EasyInputHelper.On_Click -= localClickStart;
    EasyInputHelper.On_Click -= localClickEnd;
}
```

As you can see just subscribe to the event in OnEnable and unsubscribe in OnDestroy. Just make sure your local method listed matches the signature and you can do whatever custom tasks you want with the data provided. Essentially the standard controls do this already for you so you don't need to code anything for common tasks. Anything special though you have a choice to call your method via a custom control or just subscribe manually in your code. Whichever style you choose the events will fire off appropriately

In Editor Keyboard/Mouse Mappings

One of the nice features of Easy Input Helper is that it makes it possible to test things in editor on a PC without needing to do time consuming builds each time to the physical Gear VR for non motion tasks. Below is the list of controls.

HMD- You can look around in editor by holding the left alt key and click dragging the mouse

*most games want to simulate with the gear vr controller buy if you want to in editor specify HMD mode without a controller simply hold the 'h' key

Motion- You can reorient the Gear VR Controller in editor by holding the left ctrl key and click dragging the mouse

Touchpad- Simulated with the mouse with no other keys pressed. A "touch" is when you left click the mouse and the position will be placed into a -1 to 1 range like on the device based on the screen width height

Gamepad Controller-

A button- a key or enter

B button- b key

X button- x key

Y button- y key

Left bumper- l key

Right bumper- r key

Start- s key

Back- b key

Left Stick Push- c key

Right Stick Push- v key

Left stick- arrow keys

Right stick- numpad arrow keys

Dpad- home/end/delete/pagedown keys

Left Trigger- numpad 0

Right Trigger- numpad .

Gear VR Touchpad click- t key

Gear VR Trigger- u key

Gear VR HMD tap- q key

Axis Generation

You might be wondering how the bluetooth or USB controller code is being handled for you automatically if you've ever dealt with the Unity Input API before. When you import Easy Input for Gear VR into your project it automatically creates Gamepad axes and keyboard/mouse support. If you look at your input manager you will notice added entries below. These entries are normal and are how we are able to detect any gamepad controllers axis. If you delete these make sure that they are put back for it to function properly. This happens automatically though so it shouldn't be an issue unless you manually delete them.

Summary

That's all there is to it! Finally it's possible to support the Gear VR's uniqueness in one simple to use product! Enjoy!