Easy Input Helper Documentation

Introduction

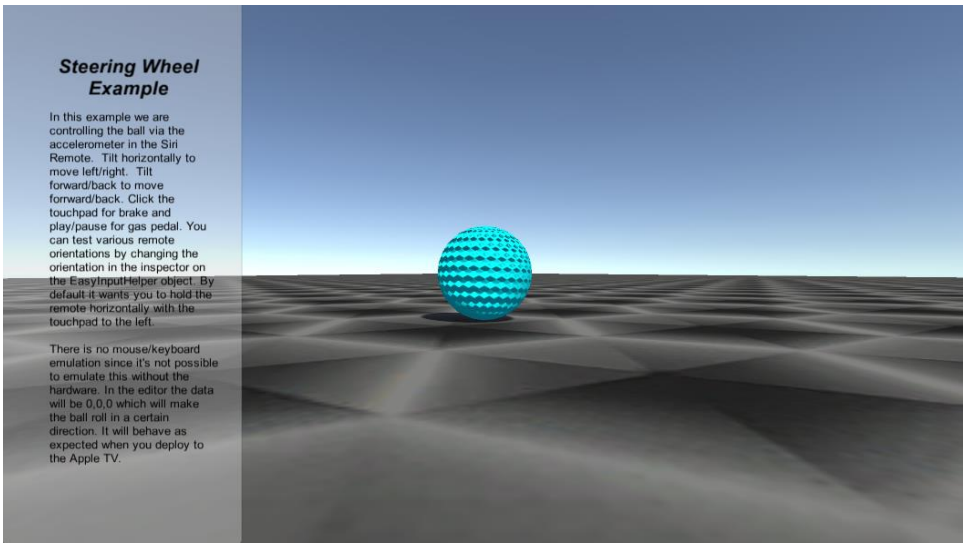
Easy Input Helper makes supporting input for the new Apple TV a breeze. Whether you want support for the siri remote or mFi controllers, everything that is unique about the platform is accessible in one easy to use API. Many common tasks even expose high level components to attach to you objects so you 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 siri remote support
- Full mFi controller support
- Standard controls for common things so no coding required
- 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
- Accelerometer/gyro support
- Touch and controller input module for Unity GUI support with no coding required
- 8 example scenes
- Easy callbacks to use if you want to do something custom

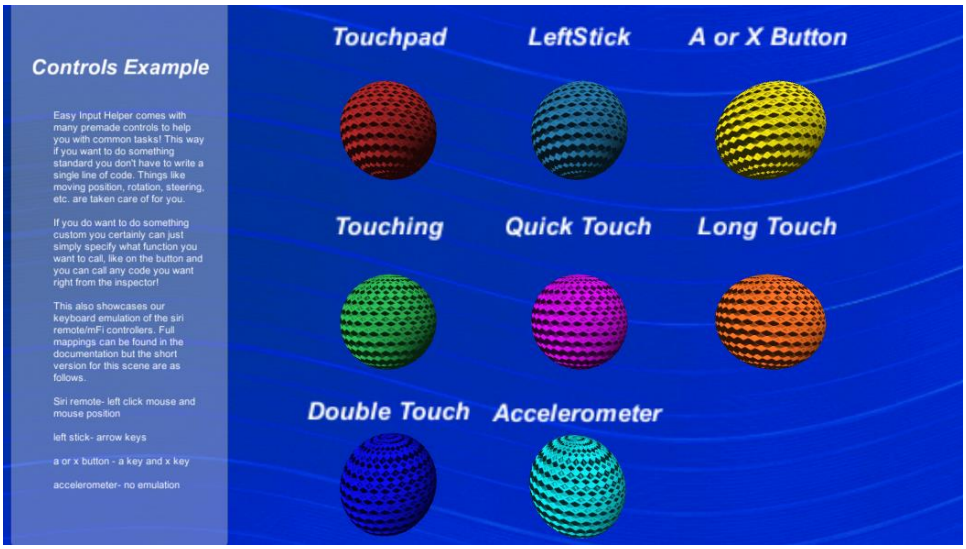
Specific Example scenes

Accelerometer Siri Remote example-



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

Controls example-

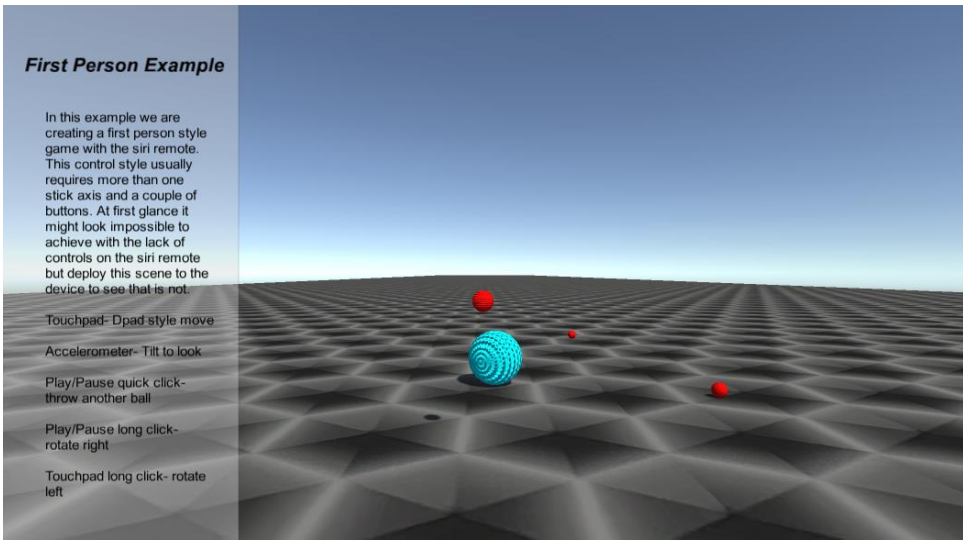


In this scene you have a wide variety of standard controls that showcase what you can do with no coding required. Use the remote as a touchpad, use the sticks on a mFi controller, call functions on button presses, or long touch, or double touch, and many more! This also shows that the product contains 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 siri remote, 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-

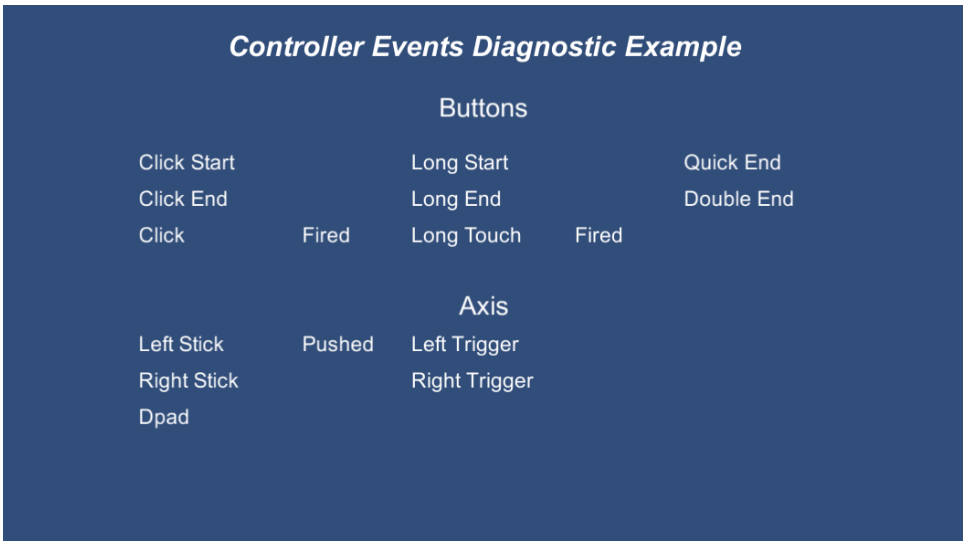
In this scene you are presented with a typical Unity UI with a grid of buttons (similar layout to the native TVOS screen). This example showcases our Easy Input Module which provides a natural way to navigate Unity UI's regardless of whether you're using the siri remote or a mFi controller. You'll notice the nice swipe style navigation is included without having to code anything. Furthermore, in addition to this the same input module also supports mFi controllers automatically at the same time, which is above and beyond what the unity standard input module provides.

First Person example-



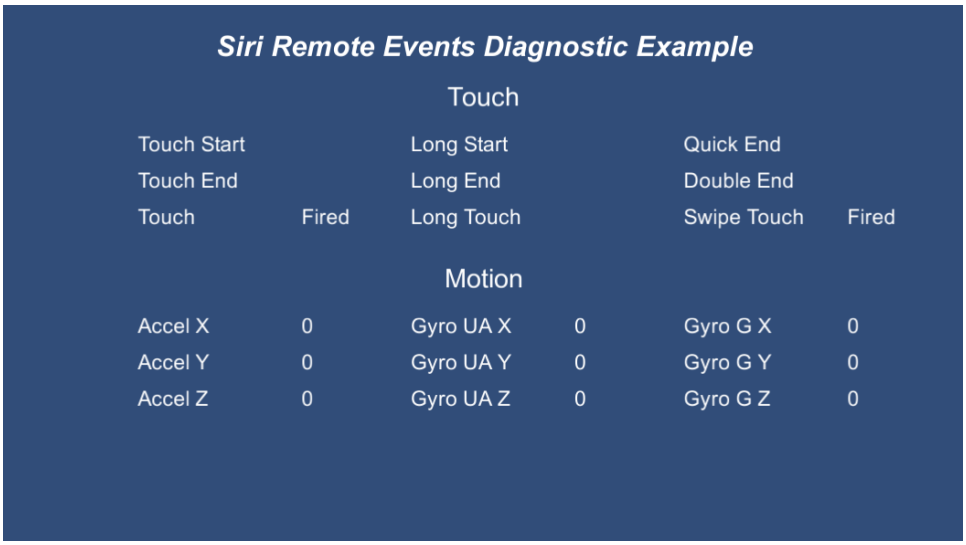
In this scene you are presented with a typical first person controller scheme adapted to the siri remote. The orientation is with the touchpad facing to the left. Use the touchpad as a dpad to move the character around. Tilt the siri remote to look around. Long click the touchpad to rotate left, long click the play/pause button to rotate right, and quick click the play/pause button to shoot.

MFi Diagnostic example-



In this scene you are presented with a real time view into our MFi 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.

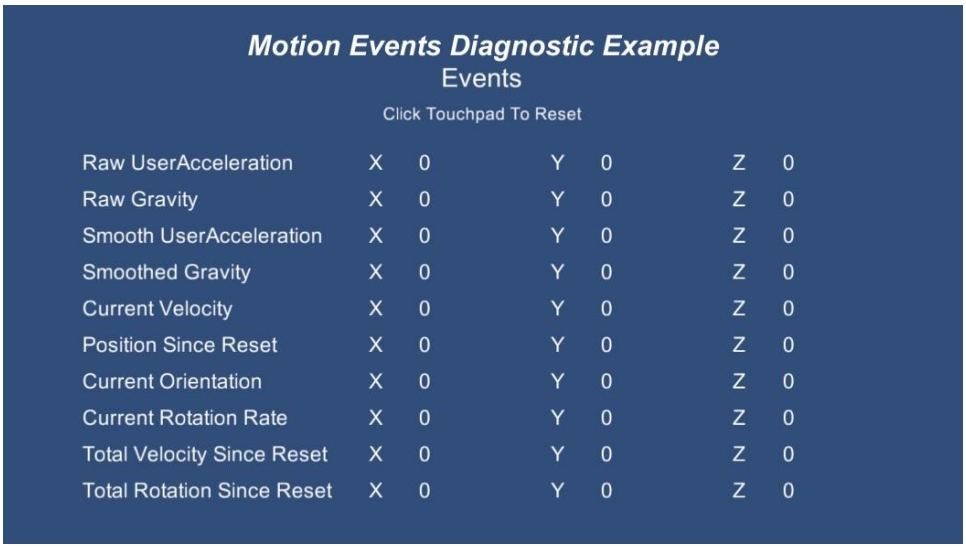
Siri Remote Diagnositc example-



In this scene you are presented with a real time view into our Siri Remote 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.

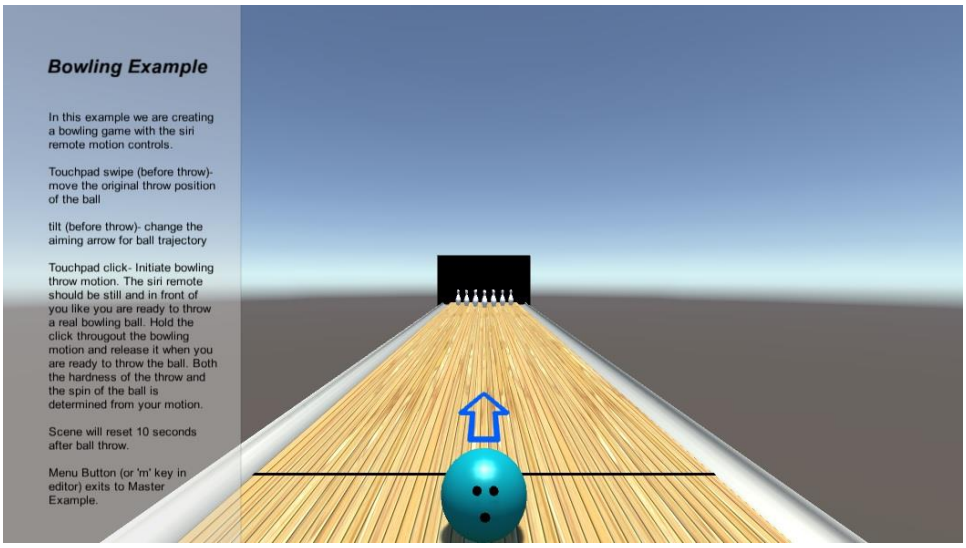
This also showcases the accelerometer and gyro telemetry to help you visualize the data that is coming in depending on how you are holding the controller.

Motion Events example-



In this scene you are presented with a real time view into our motion API. This demonstrates the motion telemetry and 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 user acceleration and gravity that is provided by the remote, Easy Input Helper calculates useful derivative information like smoothed out user acceleration and gravity, current velocity, position, orientation, rotation rate, etc. It's important to note that because the Siri Remote does not have a magnometer the axis parallel with gravity orientation and rotation rate can't be determined. The other 2 axes can though and so we include those in our API. It might just be the difference between being able to do a tennis shot, bowling throw, etc. Velocity and position are derived from the user acceleration by an integral so do not expect it to be accurate for very long. It's certainly fine for a quick motion but not any longer than a few seconds. 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.

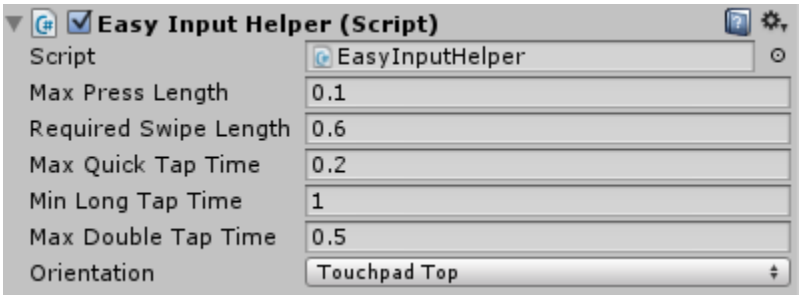
Bowling example-



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.

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.



If you want to use Easy Input Helper simply place one in your scene by the Gameobject -> Easy Input Helper -> Add Easy Input Helper 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

Title Screen- Check this box if it is the root menu of your game. When you set this to true the Apple TV API will set AllowExitToHome to true. This allows the menu button press to pause your game and return to the TVOS home screen. According to the Apple TV design documents you should make your game have a logical flow for menu presses. If in game, it should pause, then return to the main menu, then exit to the home screen. When this setting is unchecked handle the menu button presses as appropriate for your game, when it is checked AllowExitToHome makes it so a menu button press will return to the TVOS home screen.

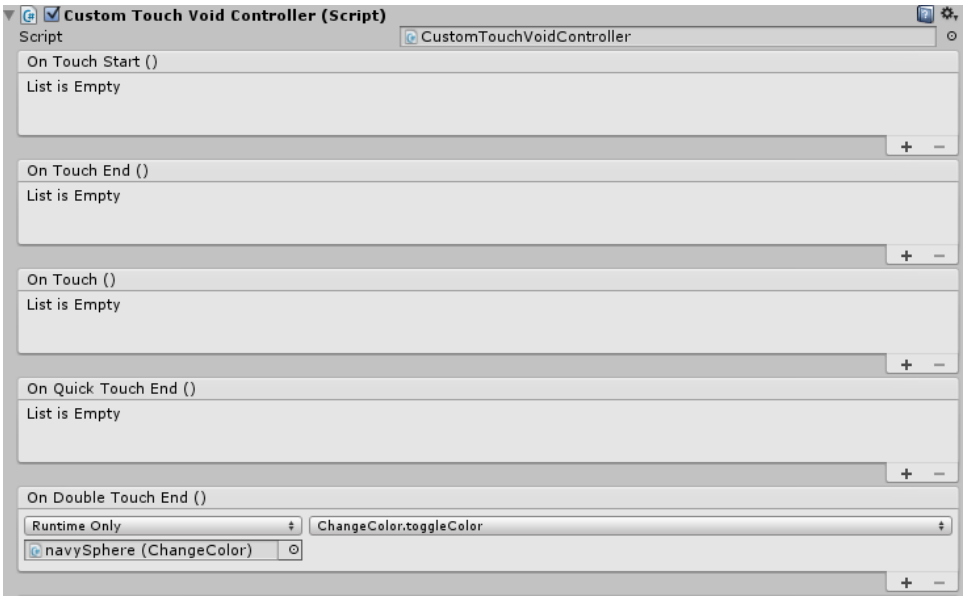
Orientation- How you want your users to hold the remote in your game. Some games want the touchpad up, some horizontal touchpad to the left, and some horizontal touchpad to the right. Easy Input helper will also auto adjust the touchpad coordinates, accelerometer axes, and gyro axes for you so that you only need to code once regardless of orientation. Simply select how you want to use it for your game. This can also be changed at runtime if you want to change at different parts of your game.

Standard Controls

Easy Input helper comes with 4 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 accelerometer. 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!

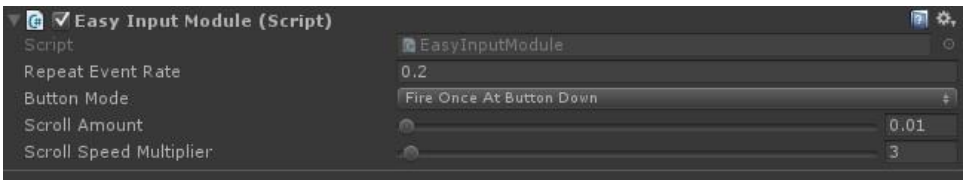
Custom Controls



Easy Input Helper 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 Siri Remote 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 the below section on manually subscribing to callbacks is for you.

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 -> EasyInputHelper -> 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- 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- 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 3 diagnostic examples. The touch events pass an InputTouch object, button events pass a ButtonClick object, axis pass a ControllerAxis object, Accelerometer passes 1 Vector3, Gyro passes 2 Vector3's, 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 Mac or PC without needing to do time consuming builds each time to the physical Apple TV. Below is the list of controls. Because most mac laptops have an abbreviated keyboard, you may want to change your mac's settings to rebind these, or change the bindings in code. If you are using a full mac keyboard with a numpad you won't have to do any of this.

Touchpad- Simulated with the mouse. 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

MFI Controller (including buttons on siri remote)-

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

Menu- m key

Left stick- arrow keys

Right stick- numpad arrow keys

Dpad- home/end/delete/pagedown keys

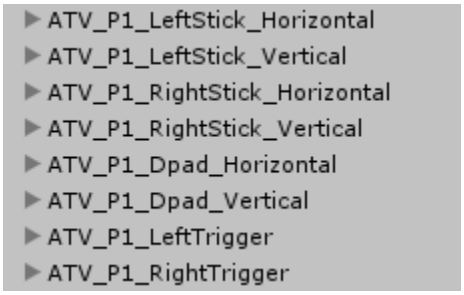
Left Trigger- numpad 0

Right Trigger- numpad .

Accelerometer/Gyro- There is no way to emulate these without the siri remote

Axis Generation

You might be wondering how the mFi controller code is being handled for you automatically if you've ever dealt with the Unity Input API before. When you import Easy Input Helper into your project it automatically creates Apple TV axes for all 3 players 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 mFi 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 Apple TV's uniqueness in one simple to use product! Enjoy!