Pigeon Coop Toolkit
Touch Input Manager

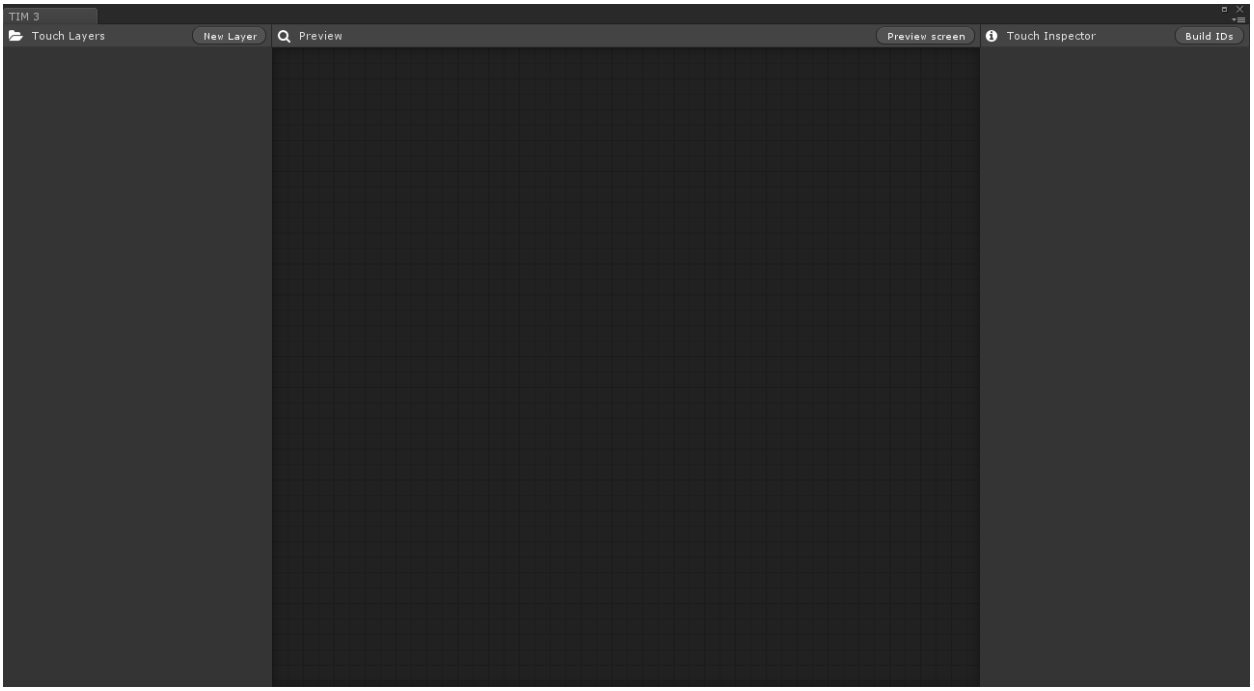
Unity has always lacked a built in method of controlling touch input through the editor. Most solutions that currently exist are cumbersome in their editor integrations. They feel like a tacked on extra, not something that's designed to work in sync with your editor. With Touch Input Manager, there is no setup, no scripts to attach to your objects or any prefabs to place in and messy up your scenes with. Touch Input Manager feels and performs like an integrated touch solution for your games providing you with easy to use touch joysticks and buttons.

If you are familiar with Unity3D's built in "Input" class, you'll be right at home with Touch Input Managers "TouchInput" class which brings you functions like TouchInput.GetButton and TouchInput.JoystickValue.

Contents

Touch Input Editor

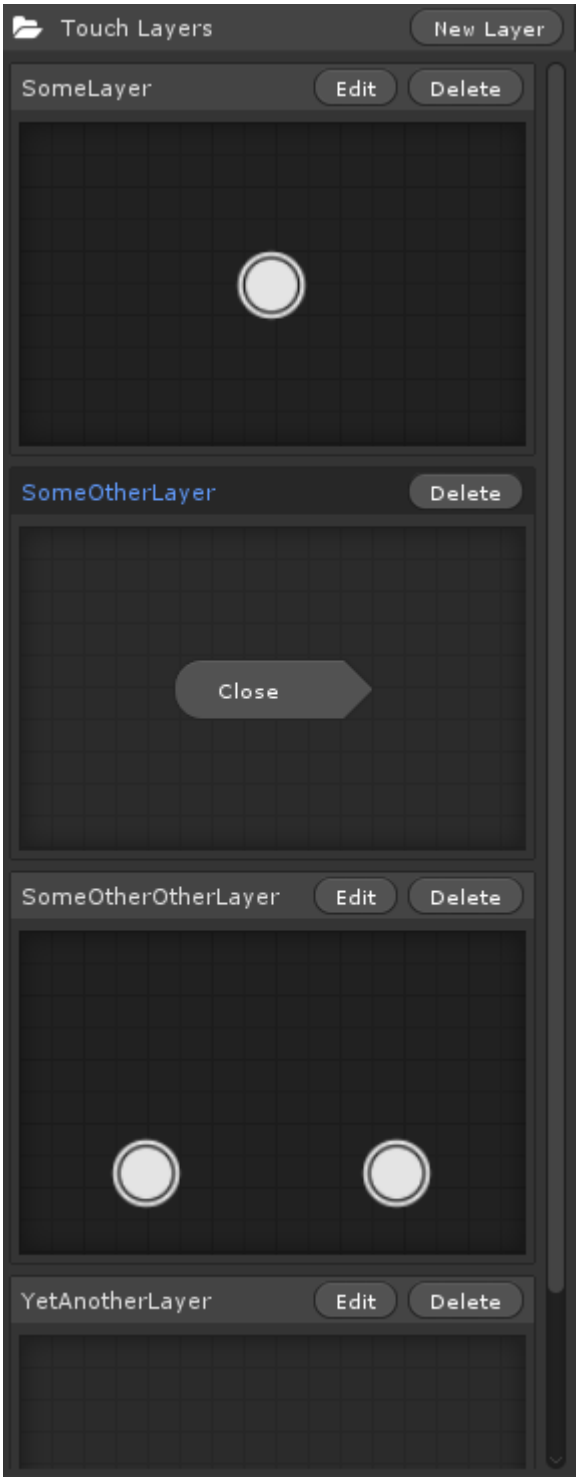
Find it under Window → Pigeon Coop Toolkit → Touch Input Manager → ***Editor***



The Touch Input Editor is where all the actions happens. From here you can create Touch Layers and populate them with Touch Devices (Joysticks + Buttons). The above image is what you will be greeted with when you open Touch Input Manager for the first time.

The skin supports both dark and light styles and will match your editor's style – for dark skin, you will need Unity Pro.

Touch Layers Column



Layers are just a way of organizing your devices. Layers can individually be toggled on and off at run time – you could for example have a layer for when your character in your game is 'driving' with a steering joystick and acceleration/break buttons and another layer for 'walking around' with 2 joysticks for looking around and walking. You would toggle between the layers when your character gets in and out of cars in your game.

In the Touch Layers column of the editor, you can see and preview all of your existing layers. You can interact with them too, so feel free to click away at the preview! (e.g.: *SomeLayer*)

The layer you are currently have selected will be highlighted blue and its preview will be disabled until you deselect the layer. (e.g.: *SomeOtherLayer*)

Layers that aren't completely inside the preview are will not show previews. (e.g.: *YetAnotherLayer*)

You can click **New Layer** at the top to add a new layer.

You can click **Edit** on an unselected layer to start editing it.

You can click **Delete** on any layer to delete it permanently. (Warning: You'll lose everything in that layer!)

You can click **Close** on a selected layer to deselect it.

Preview Column

In the preview column, you can edit and preview your Layer. The preview shown here is exactly what you'll see on your devices screen. For this reason, we've made it so you can interact with the joysticks here and see how they'll look when they animation just like on your live device. We've also provided you with many aspect ratio preview options so you can check out your layer configuration on any size screen in both portrait and landscape modes. This preview area will immediately reflect any changes you make in to the joysticks and buttons. When you've selected a layer to edit, you'll be able to add new joysticks and buttons.

Click **New Joystick** to add a new joystick to your selected layer. Click **New Button** to add a new button to your selected layer.\

Click **Preview** to select a preview aspect ratio.

To clear the preview aspect ratio, click **Previewing (x:y)** and select **Clear**.

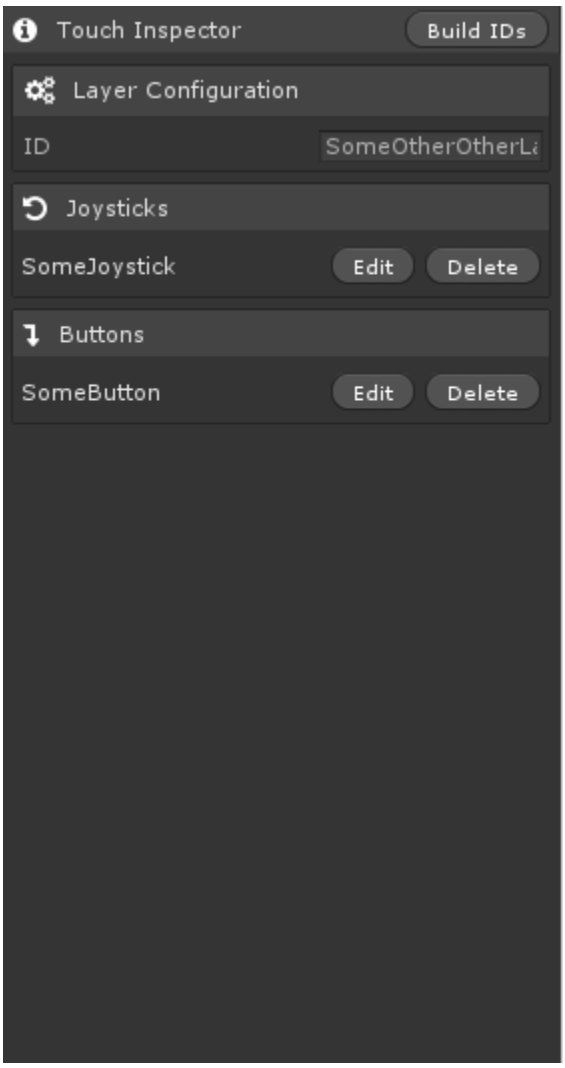
You can click and drag the **3 horizontal lines** to move around a touch device.

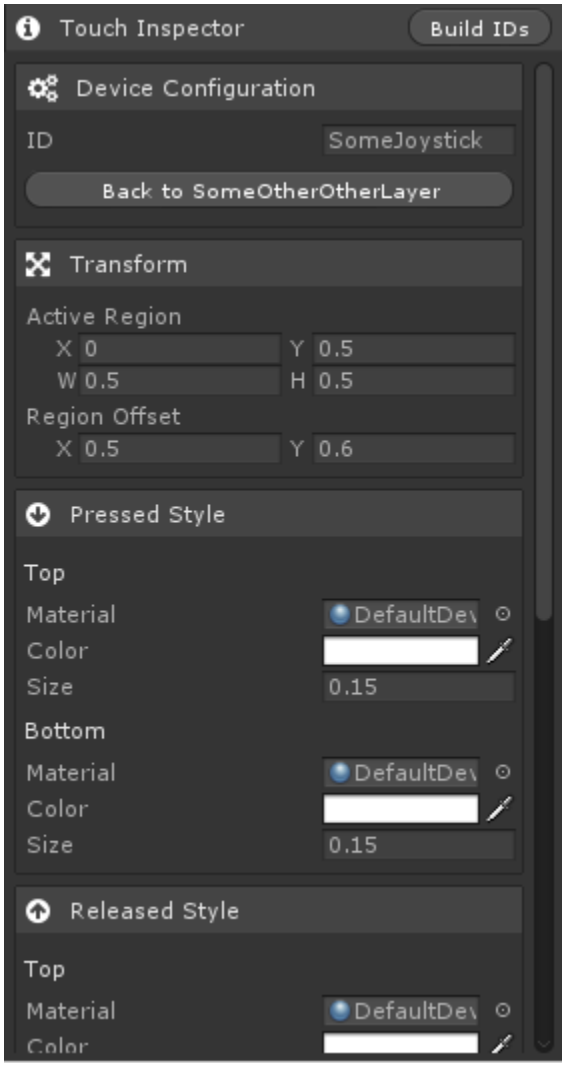
You can click and drag the **Triangle Icon** to resize the devices response region.

The device you've currently got selected will have a lighter background and its name will appear in the breadcrumbs up top. (e.g.: *SomeButtons)* Click on a devices name to select it.

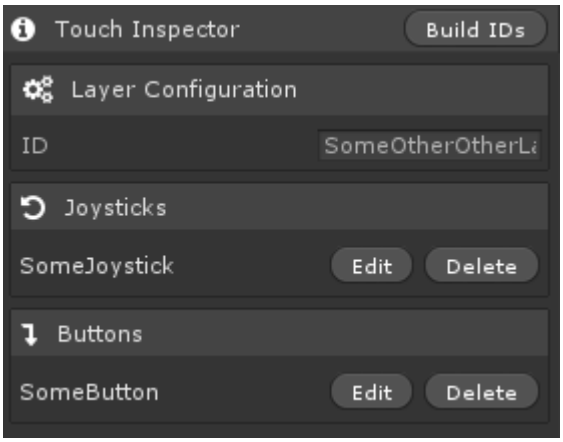
Touch Inspector Column

The Touch Inspector Column works similarly to Unity3D's inspector. It'll show you properties for whatever object you currently have selected. On the left is the inspector for a layer, and the right is the inspector for a button in that layer. We'll breakdown what everything does in the following pages!





Layer Inspector



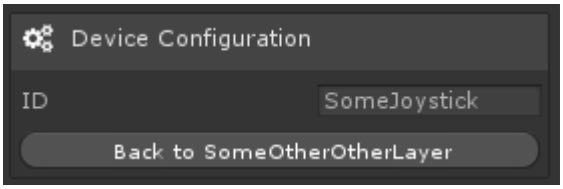
The layer inspector is really straight forward.

Layer Configuration panel allows you to set the unique ID for this layer. You can use this ID to access this layer through your scripts.

Joysticks and **Buttons** show you the devices in this layer. From here you can choose to edit or delete any of the devices. You can **add** devices in the *Preview Column (See above)*.

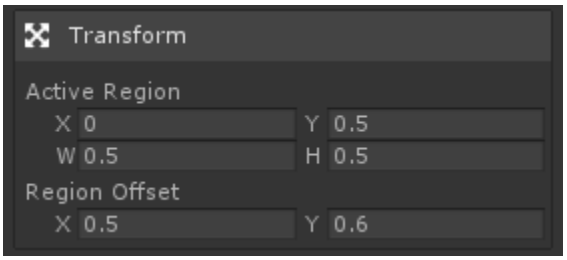
Device Inspector

We'll break the inspector down into smaller chunks and talk about each. All of this is fairly straight forward and you could easily figure it all out by pushing the buttons and seeing what they do in the editor, so this is for those of you who like to read everything about their new toy!



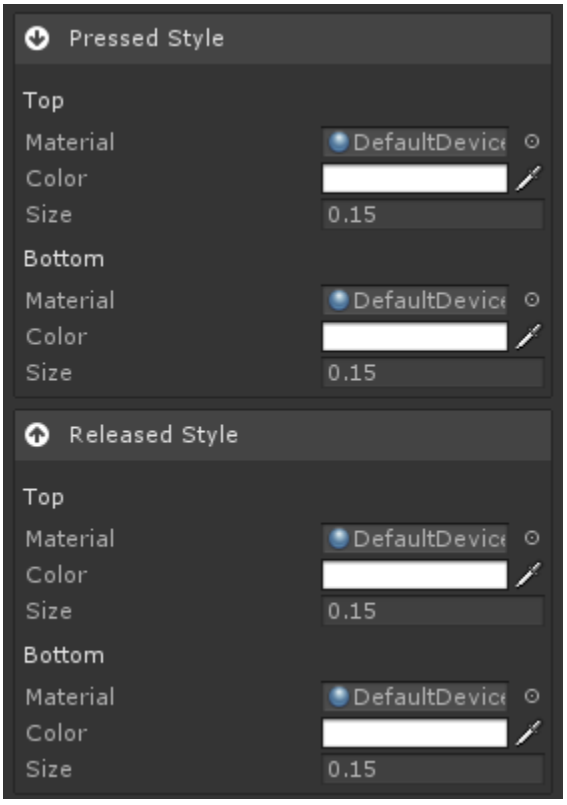
This is identical to the Layer Inspectors Configuration panel above.

You can press **Back to ******* to deselect the device and go back to the layer inspector.



The **Active Region** is the region in which the device will respond to input. The interactive tools are usually better for this job.

The **Region Offset** defines where within the **Active Region** the joystick should sit idle, (0.5, 0.5) being the center of the **Active Region**.



The Pressed Style and the Released style panels define how the device will look in either of those states.

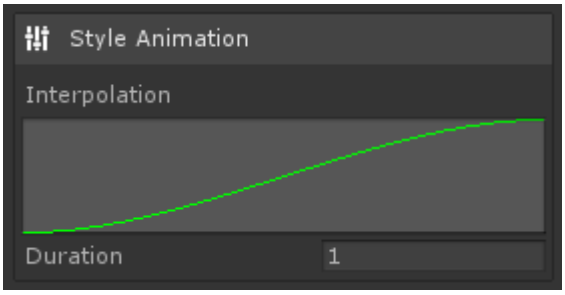
The joysticks and buttons are composed of 2 parts – a top and a bottom. These animate independently and so they can have different animations. When animated together, you can create some cool effects!

Material is the material that will be used to render the joystick onto the screen. For best results use the Transparent Unlit Texture shader built into unity! This gives you the most control. However, any shader will work.

Color is the Tint color of the texture.

Size is the screen size of the texture. (Where 1 is 'as wide the entire screen', 0.5 is 'half screen width', etc.)

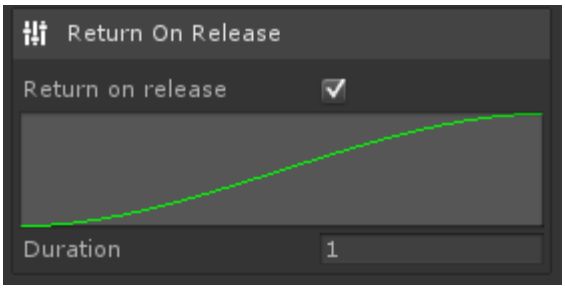
When the user presses down or released the device, the top and bottom components of the device will animate to the values defined under 'Pressed Style'/'Released Style'. You can play around with these values and see what it looks like in the preview area.



The Style Animation panel defines how the style will interpolate from ‘Pressed’ to ‘Released’ and vice versa.

The curve should start at **x = 0, y = 0** and end at **x = 1, y = 1**.

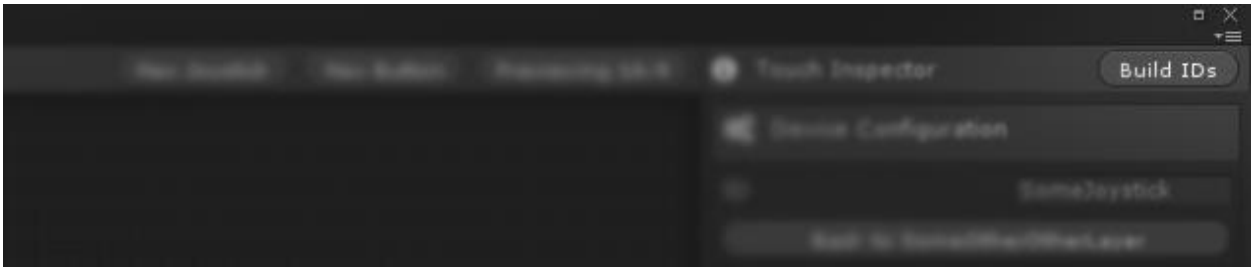
The **Duration** is how long it will take to interpolate from one state to another.



The Return On Release panel allows you to configure how and if the device will return to its idle position when the user releases it.

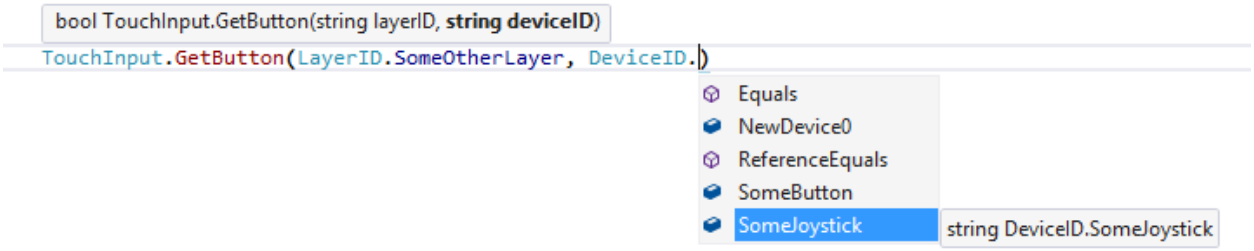
The values work the same as above.

Build IDs

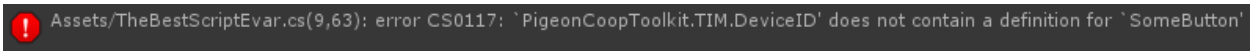


The Build ID's button can be found at the top of the Inspector column. When you press the Build ID's button, Touch Input Manager will generate a C# file with static string constants that you can use to access your layers and devices. It is recommend that you use these static string constants in your code rather than standard strings when invoking TouchInput.****** functions. You gain some awesome advantages using these static string constants.

Your IDE will autocomplete/suggest devices and layers.



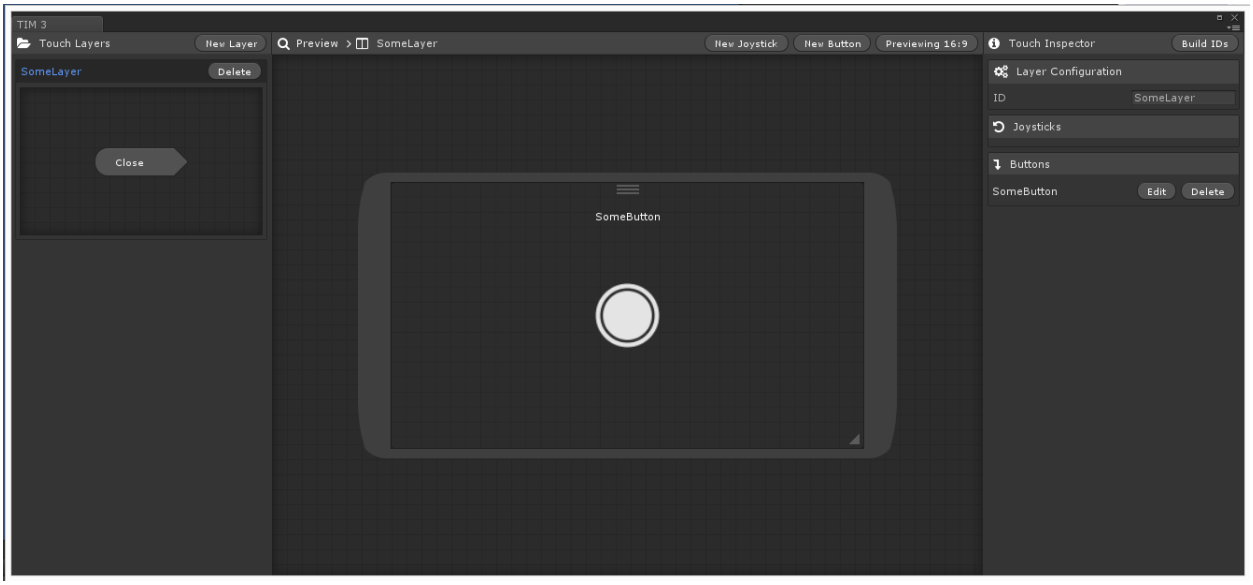
You will receive compiler errors if you delete a layer or device which us currently being used in your code. This way, you aren't left wondering why the hell your character isn't moving because you accidently deleted something or maybe you renamed a devices ID.



Oops! Better fix that!

Using Touch Input Manager in your scripts

It is very easy to start using Touch Input Manager in your own scripts! First, make sure you've setup your layer, devices and that you've hit "Built ID's"! In this Example, we'll use a relatively simple configuration and check to see if a button was pressed. Bellow is our layer (ID: *SomeLayer* and a button in that layer ID: *SomeButton*)



Firstly, we need to include the Touch Input Manager namespace in our code. This will make the Touch Input Manager code visible to our script! We've done this so that our code does not conflict with any of your other code! Nice and neat!

Let's start! Add this to you're under using statements:

```
using PigeonCoopToolkit.TIM;
```

Now we can access the static TouchInput class! Awesome!

We're going to want to reveal our Layer and pass input to it first. Otherwise the user can't see our button and they certainly can't interact with it! In your Start function, add this:

```
void Start ()
{
	TouchInput.SetVisible(LayerID.SomeLayer);
	TouchInput.SetActive(LayerID.SomeLayer);
}
```

Now our button is visible and active (will receive input!). Lastly, let's log our some text to see if its working in Update:

```
void Update()
{
	if (TouchInput.GetButtonDown(LayerID.SomeLayer, DeviceID.SomeButton))
	{
		Debug.Log("Button down!");
	}

	if(TouchInput.GetButton(LayerID.SomeLayer,DeviceID.SomeButton))
	{
		Debug.Log("Button!");
	}

	if (TouchInput.GetButtonUp(LayerID.SomeLayer, DeviceID.SomeButton))
	{
		Debug.Log("Button up!");
	}
}
```

Hit play and observe our log! That's how easy and clean it is to use touch input manager. We wanted to bring you a product that 'just works' without all the hassle of setting up GameObjects and attaching lots of different components to them.

Code Reference

```
public static void SetAllInvisible()
```
Sets all layers to be invisible

```
public static void SetAllInactive()
```
Sets all layers to be inactive (Won't receive input)

```
public static void SetAllVisible()
```
Sets all layers to be visible

```
public static void SetAllActive()
```
Sets all layers to be active (Will receive input)

```
public static void SetVisible(string layerID)
```
Sets a specific layer to be visible

```
public static void SetInvisible(string layerID)
```
Sets a specific layer to be invisible

```
public static void SetActive(string layerID)
```
Sets a specific layer to be active (Will receive input)

```
public static void SetInactive(string layerID)
```
Sets a specific layers to be inactive (Won't receive input)

```
public static bool GetButtonUp(string layerID, string deviceID)
```
Checks to see if a specific button on a specific layer was just released this frame.

```
public static bool GetButton(string layerID, string deviceID)
```
Checks to see if a specific button on a specific layer is pressed down.

```
public static bool GetButtonDown(string layerID, string deviceID)
```
Checks to see if a specific button on a specific layer was just pressed down this frame.

```
public static Vector2 JoystickValue(string layerID, string deviceID, bool normalized =
false)
```
Retrieves the value of a specific joystick in a specific layer. By default, the value is not normalized but you can pass in 'true' for 'normalized' to normalize it. It is cheaper to normalize the input this way than to use Vector2.Normalize.

```
public static bool GetJoystickUp(string layerID, string deviceID)
```
See `GetButtonUp`.

```
public static bool GetJoystick(string layerID, string deviceID)
```
See `GetButton`.

```
public static bool GetJoystickDown(string layerID, string deviceID)
```
See `GetButtonDown`.