SteamVR Unity Plugin - Input System

Actions as inputs

The heart of the SteamVR Unity Plugin is actions. It's time to abstract our thinking about input from the low level button presses to what we actually care about: user action. Instead of "pulling the trigger button down 75% will grab the block", take the first part out, just "grab the block". Let SteamVR and the user's binding (their input preferences) sort out what physical action that means.

This style of abstracting input is part of Valve's implementation of OpenXR. Most major VR platforms are contributing to the OpenXR standards. For more information in participating in this organization and contributing to industry standards like this see: https://www.khronos.org/openxr

We separate actions out into 6 different types of input:

Boolean
Single
Vector2
Vector3
Pose
Skeleton

Boolean actions are things that are either true or false. For example, Grab is a common action that is either true or false. You're either intending to hold something or you aren't, there is no in between. With a Vive Wand this may mean pulling the trigger 75%, with an Oculus Touch this may mean pulling the grip trigger 25%, with Knuckles this may be some combination of capacitive sensing and force sensing. But in the end it breaks down to a

true or false value and as a developer that's all you really have to worry about.

Single actions are analog values from 0 to 1. These are scenarios where you actually care about the in between values. There are fewer of these than you'd expect. If previously you were reading a 0 to 1 value and then waiting for it to get to a certain point, a threshold, then you can accomplish the same thing with a boolean action. And probably have your input be more customizable for your users.
A good example of a Single action is the Throttle for the RC car in the SteamVR Interaction System. The action your taking as a user can vary depending on how fast you want the car to travel.

Vector2 actions are a combination of two analog values. An X and a Y. Generally in VR so far these sort of actions are best represented by the radial menus or 2D positioning. In the SteamVR Interaciton System we have an example of an RC car as well as a platformer type character. On the vive wand this is mapped to the touchpad but on Oculus Touch and Knuckles this is mapped to the joystick. With the RC car we're using the y axis to determine forward or backward direction and the x axis to determine turning. With the platformer we're just mapping the x/y input onto the direction of the controller and moving them around through that orientation.

Vector3 actions are pretty uncommon. In SteamVR Home this is used for scrolling, x, y, and z is the number of pages to scroll.

Pose actions are a representation of a position and rotation in 3d space. This is used to track your VR controllers. The user can customize these bindings by setting the point on the controller that the pose represents. Some users may find a slightly different tracked position or rotation feels better for them.

Skeleton actions use SteamVR Skeleton Input to get our best estimation of the orientation of your fingers while holding your VR controllers. This provides one standard to get joint positions and rotations for every controller regardless of tracking fidelity. Controllers with the capacity for higher fidelity finger tracking will be more accurate than those without.

In addition to input actions there is currently one type of output action as well. The ***vibration*** action. This is to trigger haptic feedback on the vr controllers.

Action Sets

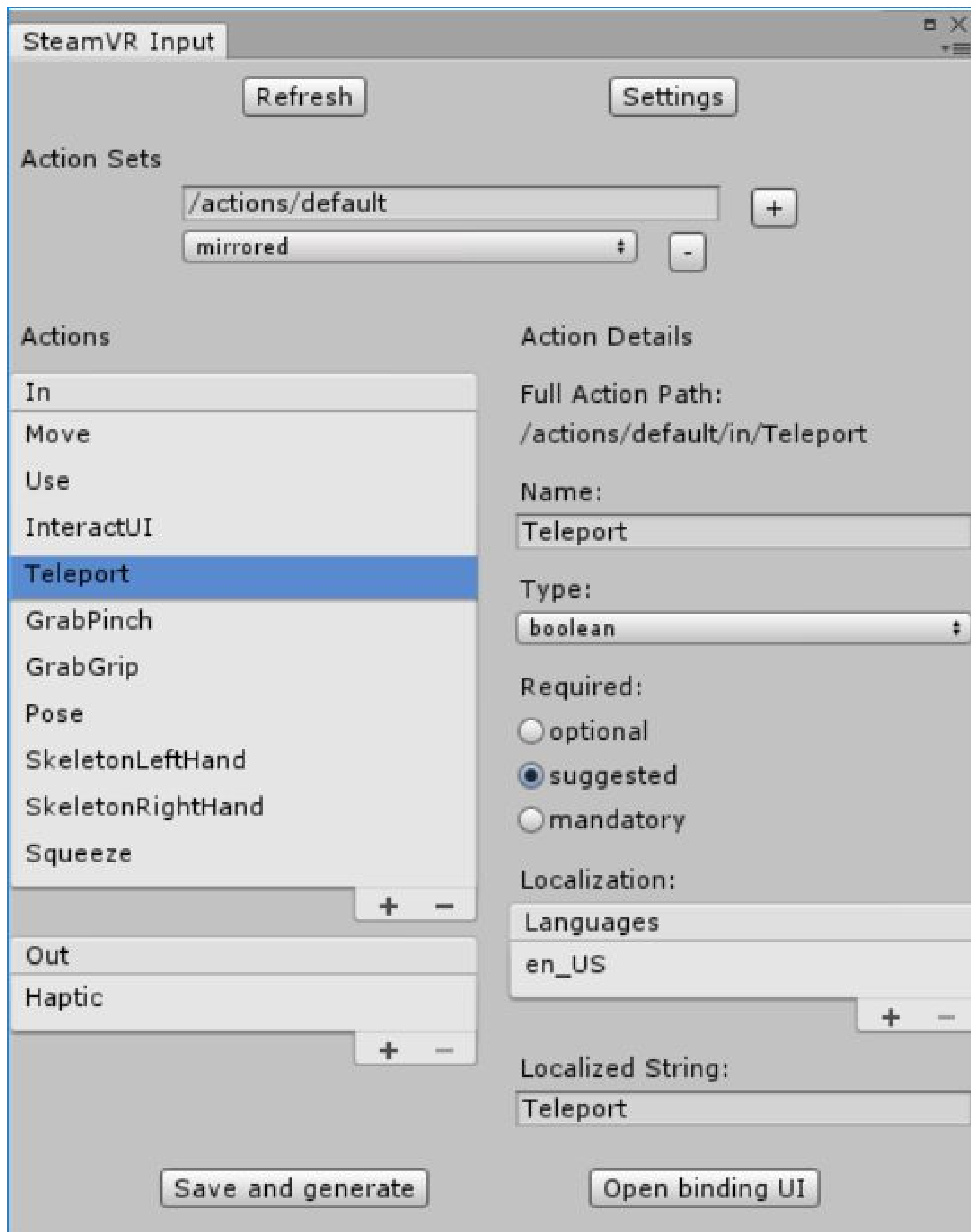
Actions are separated into logical groups by Action Sets. For example, if you have one scene of your game on earth picking up and throwing objects, and one in space flying a ship, those should probably be separate action sets. The main purpose of these is organization. To ease the discovery of actions for users who are looking to rebind them.

We have a component SteamVR_ActivateSetOnLoad that will automate activation and deactivation of action sets on a per scene basis. It activates the set on Start() and deactivates on OnDestroy().

SteamVR Input Window

Now that you have an understanding of the different types of actions lets see how you define them. In the Window menu you'll find a new option near the top labeled SteamVR Input. Selecting this will bring up a list of

actions found in your actions.json file. If you don’t have this file in the root of your project the plugin will recommend that you copy our example file over.

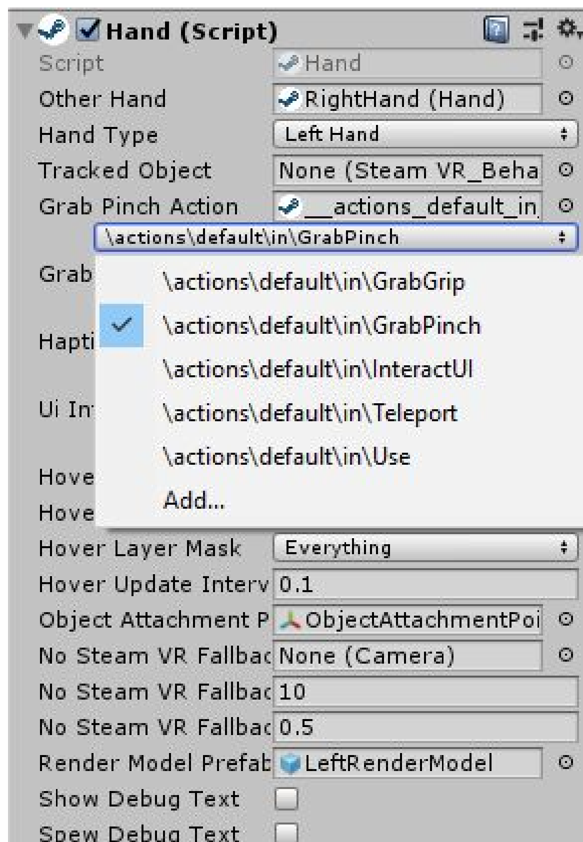


You'll see our example actions.json has three action sets. One for the general actions called default and two that are specific to a devices in the scene. The default set is active all the time and the device specific sets are only active while you're holding that device.

Actions have a name, type, how required they are, and set of localization strings. The localization strings are what will be shown to users who are looking to rebind them. So try to fill out as much as you can.

You can use the underlying SteamVR API calls to access actions and action sets directly, or you can use the wrapper layer we've built into the Unity Plugin. Then when you click Save and generate a few things will happen:

We generate scriptable objects for each action and each action set. This allows you to select actions from a dropdown in component inspectors. It also gives you persistent and accessible objects to refer to when using actions.



We also generate a bunch of code if you prefer to work that way. A SteamVR_Input partial class is created that has static references to each action set, and each action. Inside each action set you'll find references to the actions it contains. This means you can easily access your actions from anywhere.

```
public SteamVR_Input_Sources thisHand;

private void Update()
{
    if (SteamVR_Input._default.inActions.Teleport.GetStateUp(thisHand))
    {
        Teleport();
    }
}
```

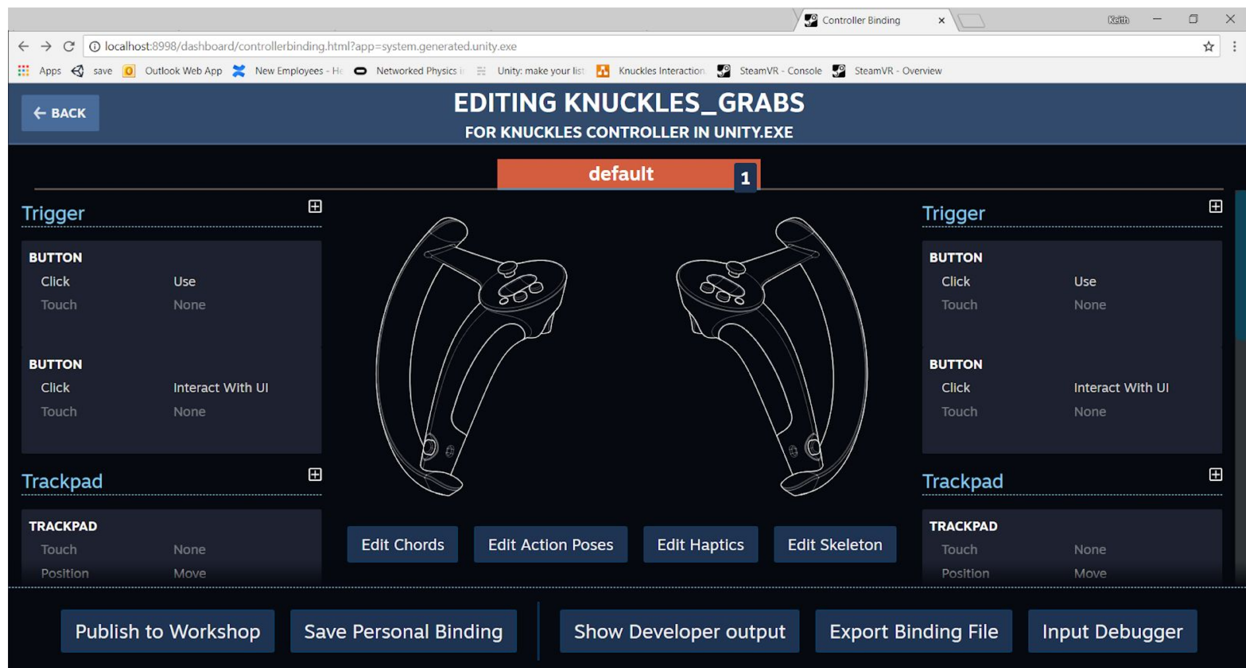
Binding UI

Creating actions is half of the process, the other half is setting up default bindings. When your users load your application for the first time they will load whatever default binding you've created for their controller type. If there isn't one they'll be prompted to create a binding themselves or pick from an existing community binding.

The SteamVR Unity Plugin has a quick link to this UI. If you open the SteamVR Input window at the bottom you'll see "Open binding UI". This will open the SteamVR Binding UI in your default browser and select the unity application you're currently editing.

Once you've created your actions and generated them you'll need to setup default bindings for each controller. Turn on your controllers and in the SteamVR Input window select "Open binding UI". This will open SteamVR's binding UI in your default browser.

From here you can select the type of controller you'd like to configure bindings for. If there's an existing binding you can click Edit to add your new bindings to the existing one.



In the SteamVR Binding UI you can hit the plus icon next to a button and it will ask you what mode you would like to bind an action as. For Single actions you'll generally want "Trigger". For Boolean actions you'll generally want "Button". Then you can click where it says "None" and select what action you want to bind to.

When you're done click the "Replace Default Binding" button in the lower right hand corner to save the binding to the same file in your project root. You can make changes to this binding while your application is running (and users can too!) and it will be reflected live in the game.

You can have one action bound multiple times. Specifically we bind most actions to the same button on each hand. By default this is done for you automatically with the "Mirror" checkbox near the bottom of the window. If you'd like to make actions that are different per controller just uncheck this box.

We have a built in Input Debugger on this screen as well. You can click on it to display the states of your actions, what they're currently bound to, and the live data from the controller. This is a great troubleshooting tool to use while testing your application when you think an action is firing but your expected result isn't happening.