**Running the examples**

**Follow these steps to run the included sample scenes**

**Create a new Unity project (The demo was written against version 5.6.4p2)**

**Download and import Oculus Utilities For Unity (The demo was written against version 1.21)**

**The OVRTrackedRemote script found in 1.21 has different public fields than in previous versions**

**Download and import the package provided with this blog post**

**Some of the scripts such as OVRInputModule provided in this package also exist in Utilities 1.21.**

**Add the following scenes to build settings**

**OVRInputSelection/Scenes/main.unity**

**OVRInputSelection/Scenes/selection\_all.unity**

**OVRInputSelection/Scenes/selection\_physics.unity**

**OVRInputSelection/Scenes/selection\_raw.unity**

**OVRInputSelection/Scenes/selection\_ui.unity**

**Load the main scene (OVRInputSelection/Scenes/main.unity)**

**Launch the samples**

**What's in the package**

**All the scripts needed to interact with UI and objects are located in OVRInputSelection/InputSystem/\*, every script lives in the ControllerSelection namespace. Some scripts in here are duplicates of scripts that come with Oculus Utilities for Unity 5, make sure to use the ones in the correct namespace. All other resources located in the provided package are only used for the sample scenes. The OVRInputSelection/InputSystem/ folder contains the following scripts:**

**OVRInputModule.cs**

**This script provides a VR specific replacement for the Standalone Input Module. Learn more about how it works in the following blog posts: Unity's UI System in VR and Adding Gear VR Controller support to Unity's UI.**

**The script contains a public field for a “tracking space” transform. This transform should be set to the tracking space of the OVRCameraRig prefab. Setting the transform in editor is recommended, but not required. If the transform is not set, the OVRInputHelper script will try to find it in your scene.**

**OVRRayPointerEventData.cs**

**This script contains a VR specific subclass of Pointer Event Data. The world space Ray used for raycasting is stored in the worldSpaceRay member variable.**

**OVRRaycaster.cs**

**This script contains a VR specific subclass of Graphics Raycaster. An instance of this script should be attached to any canvas which will receive pointer input in VR. This script replaces GraphicsRaycaster. Having a GraphicsRaycaster attached to a canvas will break the functionality of OVRRaycaster.**

**Any world space canvas should have an event camera set up. If no event camera is set up, the OVRRaycaster will try to find an appropriate camera to use.**

**OVRPhysicsRaycaster.cs**

**This script contains a VR specific replacement for the Physics Raycaster. OVRPhysicsRaycaster needs to be attached to the same game object as the OVRCameraRig script. This script is used to interact with 3D obects in a scene using Unity's event system. Any object that is interactable needs to have some kind of a Collider, as well as an Event Trigger controller. All pointer events are supported. The signature for a callback function looks like this: public void SomeCallback(BaseEventData data). The BaseEventData argument can be cast to a OVRRayPointerEventData object to get the world space ray being cast.**

**OVRPointerVisualizer.cs**

**This script is responsible for controlling the pointer visualization. When a controller (either Gear VR Controller or Touch controller) is present a selection ray is drawn. If no controller is present, the input system falls back to using a gaze controller, and a gaze reticle is used. The LineRenderer used for the selection ray and a Transform to be used for the gaze reticle need to be set.**

**The script contains a public field for a “tracking space” transform. This transform should be set to the tracking space of the OVRCameraRig prefab. Setting the transform in editor is optional, if it is not set the OVRInputHelper script will try to find it in your scene.**

**OVRRawRaycaster.cs**

**This script detects 3D objects in the scene using Physics.Raycast. This allows the script to interact with 3D objects without having to use a Physics Raycaster component or the Unity Event System.**

**The script contains a public field for a “tracking space” transform. This transform should be set to the tracking space of the OVRCameraRig prefab. Setting the transform in editor is optional, if it is not set the OVRInputHelper script will try to find it in your scene.**

**OVRInputHelpers.cs**

**This file contains several static helper functions.**

**Setting up the visualizer**

**he selection visualizer script is responsible for positioning the selection ray or gaze pointer in world space. Attach this script to any game object, follow these steps to configure it:**

**The “Tracking Space” field should be set to the TrackingSpace transform found in OVRCameraRig**

**The “Line Pointer” field should be set to a line renderer, below is the configuration of the line renderer used for the examples. Any fields which are not listed should keep their default value:**

**Cast Shadows: Off**

**Receive Shadows: Disabled (Unchecked)**

**Materials: The material used is Unlit/Color**

**Position: 2 (The individual elements don't matter)**

**Use World Space: Enabled**

**Width: 0.02**

**The “Gaze Pointer” field should be set to the transform of the object which will be used as the fallback gaze reticle.**

**In the examples, the gaze fallback is a sphere which has been scaled down to 0.05 uniformly**

**The “Ray Draw Distance” field is how long the visual ray being drawn should be**

**The “Gaze Draw Distance” is how many units from the camera the gaze pointer should be placed**

**Setting up UI Interaction**

**The Unity UI system needs an EventSystem and a Canvas to interact with UI elements. The EventSystem has a StandaloneInputModule component attached to it, which handles mouse input. The Canvas has a GraphicsRaycaster component attached to it, which performs a raycast using the mouse data against the UI elements. To interact with UI, the StandaloneInputModule and GraphicsRaycaster need to be replaced with their VR equivalents. Follow these steps to set up VR UI interaction:**

**1) Remove the StandaloneInputModule component from the EventSystem, replace it with the OVRInputModule provided with this post. If possible, set the tracking space variable that is exposed to the editor! It's expected to be set to the tracking space game object which is a part of the OVRCameraRig prefab. Setting this tracking space is optional.**

**2) To have world space UI, the Canvas has to be a world space canvas. A world space canvas needs an event camera. Set the event camera to be the CenterEyeAnchor camera located in the OVRCameraRig prefab. This is optional, if the event camera is not set, the OVRRaycaster script will try to set it.**

**3) Remove the GraphicsRaycaster component from the Canvas, replace it with a OVRRaycaster component.**

**4) Optionally, create a OVRPointerVisualizer to see the pointer being used to interact with elements in the scene**

**Setting up Event System Interaction**

**The included OVRInputModule can be used to interact with more than just the UI. It can also work with Unity's event system. This makes it easy to react to objects being clicked or hovered over. To work with the built in event system:**

**1) Add a new EventSystem to the scene. Remove the StandaloneInputModule component and replace it with a OVRInputModule component. If possible, set the tracking space variable that is exposed to the editor! It's expected to be set to the tracking space game object which is a part of the OVRCameraRig prefab. Setting this tracking space is optional.**

**2) Add a OVRPhysicsRaycaster component to the OVRCameraRig prefab instance. The OVRPhysicsRaycaster script expects to be attached to a game object with a OVRCameraRig component.**

**3) Whatever object is going to be interacted with needs to have a Collider component. The object should also have an EventTrigger component attached. All of the pointer events on the EventTrigger should behave as expected.**

**4) Optionally, create a OVRPointerVisualizer to see the pointer being used to interact with elements in the scene.**

**Setting up Raycast Interaction**

**Sometimes it can be useful to just interact with objects in a scene without using an EventSystem. The OVRRawRaycaster component was written for this purpose. Internally, it uses a Physics.Raycast to interact with the world. All of the event callbacks for selecting and hovering objects are exposed as events on the OVRRawRaycaster component. To use this component:**

**1) Attach the OVRRawRaycaster component to any game object in the scene. If possible, set the tracking space variable that is exposed to the editor! It's expected to be set to the tracking space game object which is a part of the OVRCameraRig prefab. Setting this tracking space is optional.**

**2) Configure the callbacks exposed in the editor.**

**3) Optionally, create a OVRPointerVisualizer to see the pointer being used to interact with elements in the scene.**

**Final notes**

**You may find multiple instances of OVRInputModule, OVRRaycaster, OVRPhysicsRaycaster and OVRRayPointerData in your project. This happens because Oculus Utilities 1.21 includes a version of Andy Borrell's input handling code, which is the same code that this package is based on. Be sure to use the version of these scripts located in OVRInputSelection/InputSystem, all of these scripts live in the ControllerSelection namespace.**