Udon SDK Simple Examples

This directory contains some sample Udon programs in a variety of scenes to get you started.

SpinningCubeSeries
This is the recommended starting point. You can watch the 5 video series made by Tupper on the Udon Forum:
https://ask.vrchat.com/t/spinning-cube-example-series/81

The 5 scenes match the videos so you can play with a working version if you'd like to start that way.

Prefabs
This scene has a few prefabs you might find useful.

- **VRCWorld** has the typical components needed to upload your world, and it has a special UdonBehaviour on it with three public variables: **jumpImpulse**, **walkSpeed**, and **runSpeed**. Take a look at the graph to see how these variables are set for the local player on **Start**. That means you can set them in the inspector and they will be set for each player in your world when they join.

- The **MirrorSystem** is a group of a few assets set up to easily toggle a **VRCMirror** on and off. It has two child objects – **MirrorCanvas**, which is a World Space UI Canvas with a **UIToggle**, and the **VRCMirror** prefab, with no special changes made. Take a look at the UdonBehaviour on the **MirrorSystem** object. This is program is called **ToggleGameObject,** and it can be used to turn any GameObject on and off. It has a public variable for a **targetGameObject** – this is set to **VRCMirror** for the example, but you could easily drop another GameObject in its place. When this program received a **CustomEvent** called "Toggle", it will check whether the **targetGameObject** is currently active and sends this value through '**Op Unary Negation**', which just flips the value to its opposite. So a value of true becomes false and vice versa. Then this new flipped value is used to set the new value of the GameObject, flipping its Active state each time the Toggle event is received. How does the UIToggle fire this 'Toggle' event? Take a look at the UIToggle under MirrorSystem>MirrorCanvas>MirrorToggle. This is a standard UIToggle component, with an event wired up – under **OnValueChanged**, we've added an Action to target the MirrorSystem's UdonBehaviour and fire its SendCustomEvent method with the string 'Toggle'. The final important detail is that the MirrorCanvas object has a **VRCUIShape** component attached. This tells VRChat to enable Interaction with all UI items on this canvas.

- The **AvatarPedestal** is a simple working avatar pedestal. You'll have to do a 'Build and Test' in the VRChat SDK window under 'Builder' in order to see it working. The prefab itself is a cube with a **VRC Avatar Pedestal** component with a public Blueprint Id set, and 'Change Avatars On Use' turned on. There is an UdonBehaviour on this object, open it up to see how the behaviour listens for an 'Interact' event, then uses **GetComponent** to fire the **SetAvatarUse** command for the **Local Player**.

- To activate the **Station,** you'll need to Build and test, and then walk your avatar to the chair object and **Interact** with it (typically your **Trigger** or **Left Mouse** button). This has a very simple **UseStationOnInteract** program that gets the local player object and calls Use Attached Station.

MultiExample
This scene contains working versions of a number of core concepts. Run the scene and you'll be able to explore the following programs:

- The **On Mouse Down** cube will switch between 3 materials when you click on it in the editor. Take a look at its two attached UdonBehaviours: **SendEventOnMouseDown** and **ChangeMaterialOnEvent.**

- The **Timer** cube automatically changes between its 3 materials based on a **duration** variable you can change in the inspector before you hit play. It does this with two UdonBehaviours: **SendEventOnTimer** and **ChangeMaterialOnEvent** (the same exact script as on the On Mouse Down cube).

- The Click for Loops cube will change its text to read something like 'loops:012345678'. It does this by running a loop X number of times and adding to the UI Text Field. It's got a **SendEventOnMouseDown** UdonBehaviour just like the first cube, but it points to another component. Click on the **target** public variable on this UdonBehaviour to highlight the Text field that is being changed. Click on this text field and you'll see a **SimpleForLoop** UdonBehaviour. You can change the **numberOfLoops** variable before running the scene to change the text it creates.

You'll need to Build & Test a local version of the scene so it can run in the VRChat Client in order to test the next group:

- To swap the materials on the **Interact Cube**, walk your avatar to it and **Interact**. You may have guessed – a **ChangeMaterialOnEvent** for the effect, and a **SendEventOnInteract** as a trigger.

- You can also walk over to the **On Pickup Cube** and press your pickup button (typically your **Grab** or **Left Mouse** button). Once your avatar is holding it, you can **Interact** with

it to change its color. Take a look at the **PickupAndUse** program on the cube. It changes the color of the material instead of swapping it out entirely.

SyncUI

This scene shows Sync working within VRChat in a few different ways. You'll need to Publish this scene as a private world in order to see it in action, and have a friend join you.

- If you create the world, you will be the master, and you will have control over the **UIButton Master** on the left. Every time you Interact with this button, its counter will increase for everyone in the room. Only you can push this button, unless you leave the room – which turns someone else into the master. Take a look at the button in the hierarchy under Canvas/Panel/ButtonSyncMaster. Its OnClick event has been wired to the attached UdonBehaviour to fire a custom event also called OnClick. This doesn't happen automatically, you have to wire it up yourself. Note that you don't **have** to call the CustomEvent this name, as long as you use the same string in the UI Event as you do for the CustomEvent. Next, take a look at the Udon Graph to see how the **clickCount** is stored on the object and set from the event **OnDeserialization**. This event is called on the other people in the room when the variable is updated. The event is not called for the player who set the variable, so we wire the flow that comes out of **SetVariable** into the **Text.SetText** node so that changing the variable changes the Text for the Owner of the button as well.

- The **UIButton Anyone** on the right can be pushed by anyone in the room! All over the other UI items will only update when the Master user presses them. Upon interaction, that user becomes the owner of the button, which lets their instance become the source of truth for how many button clicks should be displayed. Take a look at the Udon Graph on this object to see how everything works. This graph does three things in a specific order:
  - When the **Interact** event fires, it sets the Owner of the Button to the local player – the one who triggered this event, and then updates the Text for this new Owner.
  - After the owner is set, it can set the **clickCount** variable. There is a known bug where this doesn't work right away, so a new Owner will only have their clicks counted starting on the second click. This will be fixed soon.
  - Finally, with the **OnDeserialization** event, the Button's **uiText** label is updated with the new count. This will happen for each player who is not the owner of the object.

- The **UISlider** can be controlled by master – just aim and interact to change the value and it will sync its value to the other players in the room. This Udon Graph is very similar to the **UIButton Master** and the rest of the UI examples in the scene. It

uses public variables to wire up the UI and listens to the slider's **OnValueChanged** to fire a Custom Event also called **OnValueChanged**. This prompts the graph to save the current value of the slider to a synced variable, which is picked up by the other players **OnDeserialization**. It also updates its own text readout using this value.

- The **UIToggle** is one of the simplest examples, following our familiar formula – fire a custom **OnValueChanged** event from a UI Element's **OnValueChanged** event, update a synced variable, and update its own state from the synced variable.

- The **UIDropdown** works the same way as the above UI elements.

- The **UITextField** works very similar to the above elements. Note that you have the choice between subscribing to **OnValueChanged** or **OnEndEdit**. This example uses **OnValueChanged** to send updates more frequently, the other option would wait until an 'enter' command is made.

- The **PickupCube** on the left can be picked up by anyone in the room. Once it is picked up, it will change its color, and that new color will be synced to everyone else in the room. Take a look inside the attached UdonProgram, and notice that the Color data is synced using **smooth** interpolation. This helps smooth out the data over the network. Try the other modes and see what changes. This program uses an **Update** event to run on every frame – but the first thing it does in the **Block** node is check whether this player is both the **Owner** of the object and whether the object **Is Held**. If either of these are false, the **Branch** after the **Op Conditional And** node will be false, and that will end this flow, skipping to the second flow of the **Block** statement, which sets the color of the material for every player.

- The **PickupSphere** on the right can also be picked up by anyone. Its UdonBehaviour is empty! Instead of containing a program, it provides sync and ownership abilities just by using the checkboxes on the UdonBehaviour.

SyncValueTypes

This scene serves as a testing area for a variety of basic types, with a barebones visualization of each one syncing in the scene, and how each type responds to the different sync styles of 'Linear' and 'Smooth'.

The Gameobjects **ValueStore**, **ValueStoreLinear** and **ValueStoreSmooth** contain almost identical UdonGraphs, with their sync types for each variable the only difference. They all have a public variable for the **UITextField** that they use to show their values, which are randomly generated, synced, and then turned into a single

string for display. Note that there is a known issue where integers don’t sync correctly using the Linear or Smooth syncTypes, and floats don’t sync right using Smooth. Once those issues are fixed, you’ll be able to see them work just like the others.