

Introduction

This file provides documentation on how to use the included prefabs and scripts.

- [Prefabs](#)
 - [Pointers](#)
 - [Pointer Renderers](#)
 - [Locomotion](#)
 - [Object Control Actions](#)
 - [Interactions](#)
 - [Highlighters](#)
 - [Grab Attach Mechanics](#)
 - [Secondary Controller Grab Actions](#)
 - [Presence](#)
 - [UI](#)
 - [3D Controls](#)
 - [Utilities](#)
 - [Base SDK](#)
 - [Fallback SDK](#)
 - [Simulator SDK](#)
 - [SteamVR SDK](#)
 - [Oculus SDK](#)
 - [Daydream SDK](#)
 - [Ximmerse SDK](#)
-

Prefabs (VRTK/Prefabs)

A collection of pre-defined usable prefabs have been included to allow for each drag-and-drop set up of common elements.

- [VR Simulator](#)
- [Frames Per Second Canvas](#)
- [Object Tooltip](#)
- [Controller Tooltips](#)
- [Controller Rigidbody Activator](#)
- [Snap Drop Zone](#)
- [Radial Menu](#)
- [Radial Menu Controller](#)
- [Independent Radial Menu Controller](#)
- [Destination Point](#)
- [Pointer Direction Indicator](#)

- [Console Viewer Canvas](#)
 - [Panel Menu Controller](#)
 - [Panel Menu Item Controller](#)
-

VR Simulator (SDK_InputSimulator)

Overview

The `VRSimulatorCameraRig` prefab is a mock Camera Rig set up that can be used to develop with VRTK without the need for VR Hardware.

Use the mouse and keyboard to move around both play area and hands and interacting with objects without the need of a hmd or VR controls.

Inspector Parameters

- **Show Control Hints:** Show control information in the upper left corner of the screen.
- **Hide Hands At Switch:** Hide hands when disabling them.
- **Reset Hands At Switch:** Reset hand position and rotation when enabling them.
- **Mouse Movement Input:** Whether mouse movement always acts as input or requires a button press.
- **Lock Mouse To View:** Lock the mouse cursor to the game window when the mouse movement key is pressed.
- **Hand Move Multiplier:** Adjust hand movement speed.
- **Hand Rotation Multiplier:** Adjust hand rotation speed.
- **Player Move Multiplier:** Adjust player movement speed.
- **Player Rotation Multiplier:** Adjust player rotation speed.
- **Player Sprint Multiplier:** Adjust player sprint speed.
- **Mouse Movement Key:** Key used to enable mouse input if a button press is required.
- **Toggle Control Hints:** Key used to toggle control hints on/off.
- **Change Hands:** Key used to switch between left and right hand.
- **Hands On Off:** Key used to switch hands On/Off.
- **Rotation Position:** Key used to switch between positional and rotational movement.
- **Change Axis:** Key used to switch between X/Y and X/Z axis.
- **Distance Pickup Left:** Key used to distance pickup with left hand.
- **Distance Pickup Right:** Key used to distance pickup with right hand.
- **Distance Pickup Modifier:** Key used to enable distance pickup.
- **Move Forward:** Key used to move forward.
- **Move Left:** Key used to move to the left.
- **Move Backward:** Key used to move backwards.
- **Move Right:** Key used to move to the right.
- **Sprint:** Key used to sprint.
- **Trigger Alias:** Key used to simulate trigger button.

- **Grip Alias:** Key used to simulate grip button.
- **Touchpad Alias:** Key used to simulate touchpad button.
- **Button One Alias:** Key used to simulate button one.
- **Button Two Alias:** Key used to simulate button two.
- **Start Menu Alias:** Key used to simulate start menu button.
- **Touch Modifier:** Key used to switch between button touch and button press mode.
- **Hair Touch Modifier:** Key used to switch between hair touch mode.

Class Variables

- `public enum MouseInputMode` - Mouse input mode types
 - `Always` - Mouse movement is always treated as mouse input.
 - `RequiresButtonPress` - Mouse movement is only treated as movement when a button is pressed.

Class Methods

FindInScene/0

```
public static GameObject FindInScene()
```

- Parameters
 - *none*
- Returns
 - `GameObject` - Returns the found `VRSimulatorCameraRig` `GameObject` if it is found. If it is not found then it prints a debug log error.

The `FindInScene` method is used to find the `VRSimulatorCameraRig` `GameObject` within the current scene.

Frames Per Second Canvas (VRTK_FramesPerSecondViewer)

Overview

This canvas adds a frames per second text element to the headset. To use the prefab it must be placed into the scene then the headset camera needs attaching to the canvas:

- Select `FramesPerSecondCanvas` object from the scene objects
- Find the `Canvas` component
- Set the `Render Camera` parameter to the camera used by the VR Headset (e.g. SteamVR: [CameraRig]-> Camera(Head) -> Camera(eye))

This script is pretty much a copy and paste from the script at:

<http://talesfromtherift.com/vr-fps-counter/> So all credit to Peter Koch for his work. Twitter: @peterept

Inspector Parameters

- **Display FPS:** Toggles whether the FPS text is visible.
- **Target FPS:** The frames per second deemed acceptable that is used as the benchmark to change the FPS text colour.
- **Font Size:** The size of the font the FPS is displayed in.
- **Position:** The position of the FPS text within the headset view.
- **Good Color:** The colour of the FPS text when the frames per second are within reasonable limits of the Target FPS.
- **Warn Color:** The colour of the FPS text when the frames per second are falling short of reasonable limits of the Target FPS.
- **Bad Color:** The colour of the FPS text when the frames per second are at an unreasonable level of the Target FPS.

Example

`VRTK/Examples/018_CameraRig_FramesPerSecondCounter` displays the frames per second in the centre of the headset view. Pressing the trigger generates a new sphere and pressing the touchpad generates ten new spheres. Eventually when lots of spheres are present the FPS will drop and demonstrate the prefab.

Object Tooltip (VRTK_ObjectTooltip)

Overview

This adds a UI element into the World Space that can be used to provide additional information about an object by providing a piece of text with a line drawn to a destination point.

There are a number of parameters that can be set on the Prefab which are provided by the `VRTK_ObjectTooltip` script which is applied to the prefab.

Inspector Parameters

- **Display Text:** The text that is displayed on the tooltip.
- **Font Size:** The size of the text that is displayed.
- **Container Size:** The size of the tooltip container where `x = width` and `y = height`.
- **Draw Line From:** An optional transform of where to start drawing the line from. If one is not provided the centre of the tooltip is used for the initial line position.
- **Draw Line To:** A transform of another object in the scene that a line will be drawn from the tooltip to, this helps denote what the tooltip is in relation to. If no transform is provided and the tooltip is a child of another object, then the parent object's transform will be used as this destination

position.

- **Line Width:** The width of the line drawn between the tooltip and the destination transform.
- **Font Color:** The colour to use for the text on the tooltip.
- **Container Color:** The colour to use for the background container of the tooltip.
- **Line Color:** The colour to use for the line drawn between the tooltip and the destination transform.
- **Always Face Headset:** If this is checked then the tooltip will be rotated so it always face the headset.

Class Events

- `ObjectTooltipReset` - Emitted when the object tooltip is reset.
- `ObjectTooltipTextUpdated` - Emitted when the object tooltip text is updated.

Unity Events

Adding the `VRTK_ObjectTooltip_UnityEvents` component to `VRTK_ObjectTooltip` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `string newText` - The optional new text that is given to the tooltip.

Class Methods

ResetTooltip/0

```
public virtual void ResetTooltip()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ResetTooltip` method resets the tooltip back to its initial state.

UpdateText/1

```
public virtual void UpdateText(string newText)
```

- Parameters
 - `string newText` - A string containing the text to update the tooltip to display.
- Returns

- *none*

The `UpdateText` method allows the tooltip text to be updated at runtime.

Example

`VRTK/Examples/029_Controller_Tooltips` displays two cubes that have an object tooltip added to them along with tooltips that have been added to the controllers.

Controller Tooltips (`VRTK_ControllerTooltips`)

Overview

This adds a collection of Object Tooltips to the Controller that give information on what the main controller buttons may do. To add the prefab, it just needs to be added as a child of the relevant alias controller `GameObject`.

If the transforms for the buttons are not provided, then the script will attempt to find the attach transforms on the default controller model.

If no text is provided for one of the elements then the tooltip for that element will be set to disabled.

There are a number of parameters that can be set on the Prefab which are provided by the `VRTK_ControllerTooltips` script which is applied to the prefab.

Inspector Parameters

- **Trigger Text:** The text to display for the trigger button action.
- **Grip Text:** The text to display for the grip button action.
- **Touchpad Text:** The text to display for the touchpad action.
- **Button One Text:** The text to display for button one action.
- **Button Two Text:** The text to display for button two action.
- **Start Menu Text:** The text to display for the start menu action.
- **Tip Background Color:** The colour to use for the tooltip background container.
- **Tip Text Color:** The colour to use for the text within the tooltip.
- **Tip Line Color:** The colour to use for the line between the tooltip and the relevant controller button.
- **Trigger:** The transform for the position of the trigger button on the controller.
- **Grip:** The transform for the position of the grip button on the controller.
- **Touchpad:** The transform for the position of the touchpad button on the controller.
- **Button One:** The transform for the position of button one on the controller.
- **Button Two:** The transform for the position of button two on the controller.
- **Start Menu:** The transform for the position of the start menu on the controller.

- **Controller Events:** The controller to read the controller events from. If this is blank then it will attempt to get a controller events script from the same or parent GameObject.
- **Headset Controller Aware:** The headset controller aware script to use to see if the headset is looking at the controller. If this is blank then it will attempt to get a controller events script from the same or parent GameObject.
- **Hide When Not In View:** If this is checked then the tooltips will be hidden when the headset is not looking at the controller.
- **Retry Init Max Tries:** The total number of initialisation attempts to make when waiting for the button transforms to initialise.
- **Retry Init Counter:** The amount of seconds to wait before re-attempting to initialise the controller tooltips if the button transforms have not been initialised yet.

Class Events

- `ControllerTooltipOn` - Emitted when the controller tooltip is turned on.
- `ControllerTooltipOff` - Emitted when the controller tooltip is turned off.

Unity Events

Adding the `VRTK_ControllerTooltips_UnityEvents` component to `VRTK_ControllerTooltips` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_ControllerTooltips.TooltipButtons` element - The tooltip element being affected.

Class Methods

ResetTooltip/0

```
public virtual void ResetTooltip()
```

- Parameters
 - *none*
- Returns
 - *none*

The Reset method reinitialises the tooltips on all of the controller elements.

UpdateText/2

```
public virtual void UpdateText(TipButtons element, string newText)
```

- Parameters

- `TooltipButtons element` - The specific controller element to change the tooltip text on.
- `string newText` - A string containing the text to update the tooltip to display.

- Returns

- *none*

The `UpdateText` method allows the tooltip text on a specific controller element to be updated at runtime.

ToggleTips/2

```
public virtual void ToggleTips(bool state, TooltipButtons element =  
    TooltipButtons.None)
```

- Parameters

- `bool state` - The state of whether to display or hide the controller tooltips, `true` will display and `false` will hide.
- `TooltipButtons element` - The specific element to hide the tooltip on, if it is `TooltipButtons.None` then it will hide all tooltips. Optional parameter defaults to `TooltipButtons.None`

- Returns

- *none*

The `ToggleTips` method will display the controller tooltips if the state is `true` and will hide the controller tooltips if the state is `false`. An optional `element` can be passed to target a specific controller tooltip to toggle otherwise all tooltips are toggled.

Example

`VRTK/Examples/029_Controller_Tooltips` displays two cubes that have an object tooltip added to them along with tooltips that have been added to the controllers.

Controller Rigidbody Activator (VRTK_ControllerRigidbodyActivator)

Overview

This adds a simple trigger collider volume that when a controller enters will enable the rigidbody on the controller.

The prefab game object should be placed in the scene where another interactable game object (such as a button control) is located to turn the controller rigidbody on at the appropriate time for interaction with the control without needing to manually activate by pressing the grab.

If the prefab is placed as a child of the target interactable game object then the collider volume on the prefab will trigger collisions on the interactable object.

The sphere collider on the prefab can have the radius adjusted to determine how close the controller needs to be to the object before the rigidbody is activated.

It's also possible to replace the sphere trigger collider with an alternative trigger collider for customised collision detection.

Inspector Parameters

- **Is Enabled:** If this is checked then the collider will have it's rigidbody toggled on and off during a collision.

Class Events

- `ControllerRigidbodyOn` - Emitted when the controller rigidbody is turned on.
- `ControllerRigidbodyOff` - Emitted when the controller rigidbody is turned off.

Unity Events

Adding the `VRTK_ControllerRigidbodyActivator_UnityEvents` component to `VRTK_ControllerRigidbodyActivator` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `interactingObject` - The object that touching the activator.
-

Snap Drop Zone (`VRTK_SnapDropZone`)

Overview

This sets up a predefined zone where an existing interactable object can be dropped and upon dropping it snaps to the set snap drop zone transform position, rotation and scale.

The position, rotation and scale of the `SnapDropZone` Game Object will be used to determine the final position of the dropped interactable object if it is dropped within the drop zone collider volume.

The provided Highlight Object Prefab is used to create the highlighting object (also within the Editor for easy placement) and by default the standard Material Color Swap highlighter is used.

An alternative highlighter can also be added to the `SnapDropZone` Game Object and this new highlighter component will be used to show the interactable object position on release.

The prefab is a pre-built game object that contains a default trigger collider (Sphere Collider) and a kinematic rigidbody (to ensure collisions occur).

If an alternative collider is required, then the default Sphere Collider can be removed and another collider added.

If the `Use Joint Snap Type` is selected then a custom Joint component is required to be added to the `SnapDropZone` Game Object and upon release the interactable object's rigidbody will be linked to this joint as the `Connected Body`.

Inspector Parameters

- **Highlight Object Prefab:** A game object that is used to draw the highlighted destination for within the drop zone. This object will also be created in the Editor for easy placement.
- **Snap Type:** The Snap Type to apply when a valid interactable object is dropped within the snap zone.
- **Snap Duration:** The amount of time it takes for the object being snapped to move into the new snapped position, rotation and scale.
- **Apply Scaling On Snap:** If this is checked then the scaled size of the snap drop zone will be applied to the object that is snapped to it.
- **Clone New On Unsnap:** If this is checked then when the snapped object is unsnapped from the drop zone, a clone of the unsnapped object will be snapped back into the drop zone.
- **Highlight Color:** The colour to use when showing the snap zone is active.
- **Highlight Always Active:** The highlight object will always be displayed when the snap drop zone is available even if a valid item isn't being hovered over.
- **Valid Object List Policy:** A specified `VRTK_PolicyList` to use to determine which interactable objects will be snapped to the snap drop zone on release.
- **Display Drop Zone In Editor:** If this is checked then the drop zone highlight section will be displayed in the scene editor window.
- **Default Snapped Object:** The game object to snap into the dropzone when the drop zone is enabled. The game object must be valid in any given policy list to snap.

Class Variables

- `public enum SnapTypes` - The types of snap on release available.
 - `UseKinematic` - Will set the interactable object rigidbody to `isKinematic = true`.
 - `UseJoint` - Will attach the interactable object's rigidbody to the provided joint as it's `Connected Body`.
 - `UseParenting` - Will set the `SnapDropZone` as the interactable object's parent and set it's rigidbody to `isKinematic = true`.

Class Events

- `ObjectEnteredSnapDropZone` - Emitted when a valid interactable object enters the snap drop zone trigger collider.
- `ObjectExitedSnapDropZone` - Emitted when a valid interactable object exists the snap drop zone

trigger collider.

- `ObjectSnappedToDropZone` - Emitted when an interactable object is successfully snapped into a drop zone.
- `ObjectUnsnappedFromDropZone` - Emitted when an interactable object is removed from a snapped drop zone.

Unity Events

Adding the `VRTK_SnapDropZone_UnityEvents` component to `VRTK_SnapDropZone` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `GameObject snappedObject` - The interactable object that is dealing with the snap drop zone.

Class Methods

InitialiseHighlightObject/1

```
public virtual void InitialiseHighlightObject(bool removeOldObject = false)
```

- Parameters
 - `bool removeOldObject` - If this is set to true then it attempts to delete the old highlight object if it exists. Defaults to `false`
- Returns
 - *none*

The `InitialiseHighlightObject` method sets up the highlight object based on the given Highlight Object Prefab.

ForceSnap/1

```
public virtual void ForceSnap(GameObject objectToSnap)
```

- Parameters
 - `GameObject objectToSnap` - The `GameObject` to attempt to snap.
- Returns
 - *none*

the `ForceSnap` method attempts to automatically attach a valid game object to the snap drop zone.

ForceUnsnap/0

```
public virtual void ForceUnsnap()
```

- Parameters
 - *none*
- Returns
 - *none*

The ForceUnsnap method attempts to automatically remove the current snapped game object from the snap drop zone.

ValidSnappableObjectIsHovering/0

```
public virtual bool ValidSnappableObjectIsHovering()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if a valid object is currently in the snap drop zone area.

The ValidSnappableObjectIsHovering method determines if any valid objects are currently hovering in the snap drop zone area.

IsObjectHovering/1

```
public virtual bool IsObjectHovering(GameObject checkObject)
```

- Parameters
 - `GameObject checkObject` - The `GameObject` to check to see if it's hovering in the snap drop zone area.
- Returns
 - `bool` - Returns true if the given `GameObject` is hovering (but not snapped) in the snap drop zone area.

The IsObjectHovering method determines if the given `GameObject` is currently hovering (but not snapped) in the snap drop zone area.

GetHoveringObjects/0

```
public virtual List<GameObject> GetHoveringObjects()
```

- Parameters
 - *none*

- Returns

- `List<GameObject>` - The List of valid GameObjects that are hovering (but not snapped) in the snap drop zone area.

The `GetHoveringObjects` method returns a List of valid GameObjects that are currently hovering (but not snapped) in the snap drop zone area.

GetCurrentSnappedObject/0

```
public virtual GameObject GetCurrentSnappedObject()
```

- Parameters

- *none*

- Returns

- `GameObject` - The GameObject that is currently snapped in the snap drop zone area.

The `GetCurrentSnappedObject` method returns the GameObject that is currently snapped in the snap drop zone area.

Example

`VRTK/Examples/041_Controller_ObjectSnappingToDropZones` uses the `VRTK_SnapDropZone` prefab to set up pre-determined snap zones for a range of objects and demonstrates how only objects of certain types can be snapped into certain areas.

Radial Menu (VRTK_RadialMenu)

Overview

This adds a UI element into the world space that can be dropped into a Controller object and used to create and use Radial Menus from the touchpad.

If the RadialMenu is placed inside a controller, it will automatically find a `VRTK_ControllerEvents` in its parent to use at the input. However, a `VRTK_ControllerEvents` can be defined explicitly by setting the `Events` parameter of the `Radial Menu Controller` script also attached to the prefab.

The RadialMenu can also be placed inside a `VRTK_InteractableObject` for the RadialMenu to be anchored to a world object instead of the controller. The `Events Manager` parameter will automatically be set if the RadialMenu is a child of an `InteractableObject`, but it can also be set manually in the inspector. Additionally, for the RadialMenu to be anchored in the world, the `RadialMenuController` script in the prefab must be replaced with `VRTK_IndependentRadialMenuController`. See the script information for further details on making the RadialMenu independent of the controllers.

Inspector Parameters

- **Buttons:** An array of Buttons that define the interactive buttons required to be displayed as part of the radial menu.
- **Button Prefab:** The base for each button in the menu, by default set to a dynamic circle arc that will fill up a portion of the menu.
- **Generate On Awake:** If checked, then the buttons will be auto generated on awake.
- **Button Thickness:** Percentage of the menu the buttons should fill, 1.0 is a pie slice, 0.1 is a thin ring.
- **Button Color:** The background colour of the buttons, default is white.
- **Offset Distance:** The distance the buttons should move away from the centre. This creates space between the individual buttons.
- **Offset Rotation:** The additional rotation of the Radial Menu.
- **Rotate Icons:** Whether button icons should rotate according to their arc or be vertical compared to the controller.
- **Icon Margin:** The margin in pixels that the icon should keep within the button.
- **Is Shown:** Whether the buttons are shown
- **Hide On Release:** Whether the buttons should be visible when not in use.
- **Execute On Unclick:** Whether the button action should happen when the button is released, as opposed to happening immediately when the button is pressed.
- **Base Haptic Strength:** The base strength of the haptic pulses when the selected button is changed, or a button is pressed. Set to zero to disable.
- **Menu Buttons:** The actual GameObjects that make up the radial menu.

Class Methods

HoverButton/1

```
public virtual void HoverButton(float angle)
```

- Parameters
 - `float angle` - The angle on the radial menu.
- Returns
 - *none*

The HoverButton method is used to set the button hover at a given angle.

ClickButton/1

```
public virtual void ClickButton(float angle)
```

- Parameters
 - `float angle` - The angle on the radial menu.

- Returns
 - *none*

The ClickButton method is used to set the button click at a given angle.

UnClickButton/1

```
public virtual void UnClickButton(float angle)
```

- Parameters
 - float angle - The angle on the radial menu.
- Returns
 - *none*

The UnClickButton method is used to set the button unclick at a given angle.

ToggleMenu/0

```
public virtual void ToggleMenu()
```

- Parameters
 - *none*
- Returns
 - *none*

The ToggleMenu method is used to show or hide the radial menu.

StopTouching/0

```
public virtual void StopTouching()
```

- Parameters
 - *none*
- Returns
 - *none*

The StopTouching method is used to stop touching the menu.

ShowMenu/0

```
public virtual void ShowMenu()
```

- Parameters

- *none*
- Returns
 - *none*

The ShowMenu method is used to show the menu.

GetButton/1

```
public virtual RadialMenuButton GetButton(int id)
```

- Parameters
 - `int id` - The id of the button to retrieve.
- Returns
 - `RadialMenuButton` - The found radial menu button.

The GetButton method is used to get a button from the menu.

HideMenu/1

```
public virtual void HideMenu(bool force)
```

- Parameters
 - `bool force` - If true then the menu is always hidden.
- Returns
 - *none*

The HideMenu method is used to hide the menu.

RegenerateButtons/0

```
public void RegenerateButtons()
```

- Parameters
 - *none*
- Returns
 - *none*

The RegenerateButtons method creates all the button arcs and populates them with desired icons.

AddButton/1

```
public void AddButton(RadialMenuButton newButton)
```


- Parameters
 - `RadialMenuButton newButton` - The button to add.
- Returns
 - *none*

The `AddButton` method is used to add a new button to the menu.

Example

`VRTK/Examples/030_Controls_RadialTouchpadMenu` displays a radial menu for each controller. The left controller uses the `Hide On Release` variable, so it will only be visible if the left touchpad is being touched. It also uses the `Execute On Unclick` variable to delay execution until the touchpad button is unclicked. The example scene also contains a demonstration of anchoring the `RadialMenu` to an interactable cube instead of a controller.

Radial Menu Controller (VRTK_RadialMenuController)

Overview

This adds a UI element into the world space that can be dropped into a Controller object and used to create and use Radial Menus from the touchpad.

If the `RadialMenu` is placed inside a controller, it will automatically find a `VRTK_ControllerEvents` in its parent to use at the input. However, a `VRTK_ControllerEvents` can be defined explicitly by setting the `Events` parameter of the `Radial Menu Controller` script also attached to the prefab.

The `RadialMenu` can also be placed inside a `VRTK_InteractiveObject` for the `RadialMenu` to be anchored to a world object instead of the controller. The `Events Manager` parameter will automatically be set if the `RadialMenu` is a child of an `InteractiveObject`, but it can also be set manually in the inspector. Additionally, for the `RadialMenu` to be anchored in the world, the `RadialMenuController` script in the prefab must be replaced with `VRTK_IndependentRadialMenuController`. See the script information for further details on making the `RadialMenu` independent of the controllers.

Inspector Parameters

- **Events:** The controller to listen to the controller events on.

Example

`VRTK/Examples/030_Controls_RadialTouchpadMenu` displays a radial menu for each controller. The left controller uses the `Hide On Release` variable, so it will only be visible if the left touchpad is being touched. It also uses the `Execute On Unclick` variable to delay execution until the touchpad button is unclicked. The example scene also contains a demonstration of anchoring the `RadialMenu` to an interactable cube instead of a controller.

Independent Radial Menu Controller (VRTK_IndependentRadialMenuController)

extends VRTK_RadialMenuController

Overview

This script inherited from `RadialMenuController` and therefore can be used instead of `RadialMenuController` to allow the RadialMenu to be anchored to any object, not just a controller. The RadialMenu will show when a controller is near the object and the buttons can be clicked with the `Use Alias` button. The menu also automatically rotates towards the user.

To convert the default `RadialMenu` prefab to be independent of the controllers:

- Make the `RadialMenu` a child of an object other than a controller.
- Position and scale the menu by adjusting the transform of the `RadialMenu` empty.
- Replace `RadialMenuController` with `VRTK_IndependentRadialMenuController`.
- Ensure the parent object has the `VRTK_InteractableObject` script.
- Verify that `Is Usable` and `Hold Button to Use` are both checked.
- Attach `VRTK_InteractTouch` and `VRTK_InteractUse` scripts to the controllers.

Inspector Parameters

- **Events Manager:** If the `RadialMenu` is the child of an object with `VRTK_InteractableObject` attached, this will be automatically obtained. It can also be manually set.
- **Add Menu Collider:** Whether or not the script should dynamically add a `SphereCollider` to surround the menu.
- **Collider Radius Multiplier:** This times the size of the `RadialMenu` is the size of the collider.
- **Hide After Execution:** If true, after a button is clicked, the `RadialMenu` will hide.
- **Offset Multiplier:** How far away from the object the menu should be placed, relative to the size of the `RadialMenu`.
- **Rotate Towards:** The object the `RadialMenu` should face towards. If left empty, it will automatically try to find the Headset Camera.

Class Methods

UpdateEventManager/0

```
public virtual void UpdateEventManager()
```

- Parameters

- *none*
- Returns
 - *none*

The `UpdateEventManager` method is used to update the events within the menu controller.

Destination Point (VRTK_DestinationPoint)

extends `VRTK_DestinationMarker`

Overview

The Destination Point allows for a specific scene marker that can be teleported to.

The destination points can provide a useful way of having specific teleport locations in a scene.

The destination points can also have a locked state if the `Enable Teleport` flag is disabled.

Inspector Parameters

- **Default Cursor Object:** The `GameObject` to use to represent the default cursor state.
- **Hover Cursor Object:** The `GameObject` to use to represent the hover cursor state.
- **Locked Cursor Object:** The `GameObject` to use to represent the locked cursor state.
- **Destination Location:** An optional transform to determine the destination location for the destination marker. This can be useful to offset the destination location from the destination point. If this is left empty then the destination point transform will be used.
- **Snap To Point:** If this is checked then after teleporting, the play area will be snapped to the origin of the destination point. If this is false then it's possible to teleport to anywhere within the destination point collider.
- **Hide Pointer Cursor On Hover:** If this is checked, then the pointer cursor will be hidden when a valid destination point is hovered over.
- **Hide Direction Indicator On Hover:** If this is checked, then the pointer direction indicator will be hidden when a valid destination point is hovered over. A pointer direction indicator will always be hidden if snap to rotation is set.
- **Snap To Rotation:** Determines if the play area will be rotated to the rotation of the destination point upon the destination marker being set.

Class Variables

- `public enum RotationTypes` - Allowed snap to rotation types.
 - `NoRotation` - No rotation information will be emitted in the destination set payload.
 - `RotateWithNoHeadsetOffset` - The destination point's rotation will be emitted without taking into consideration the current headset rotation.

- `RotateWithHeadsetOffset` - The destination point's rotation will be emitted and will take into consideration the current headset rotation.

Class Events

- `DestinationPointEnabled` - Emitted when the destination point is enabled.
- `DestinationPointDisabled` - Emitted when the destination point is disabled.
- `DestinationPointLocked` - Emitted when the destination point is locked.
- `DestinationPointUnlocked` - Emitted when the destination point is unlocked.
- `DestinationPointReset` - Emitted when the destination point is reset.

Unity Events

Adding the `VRTK_DestinationPoint_UnityEvents` component to `VRTK_DestinationPoint` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

ResetDestinationPoint/0

```
public virtual void ResetDestinationPoint()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ResetDestinationPoint` resets the destination point back to the default state.

Example

`044_CameraRig_RestrictedTeleportZones` uses the `VRTK_DestinationPoint` prefab to set up a collection of pre-defined teleport locations.

Pointer Direction Indicator (VRTK_PointerDirectionIndicator)

Overview

The Pointer Direction Indicator is used to determine a given world rotation that can be used by a Destination Marker.

The Pointer Direction Indicator can be attached to a `VRTK_Pointer` in the `Direction Indicator` parameter and will be used to send rotation data when the destination marker events are

emitted.

This can be useful for rotating the play area upon teleporting to face the user in a new direction without expecting them to physically turn in the play space.

Inspector Parameters

- **Include Headset Offset:** If this is checked then the reported rotation will include the offset of the headset rotation in relation to the play area.
- **Display On Invalid Location:** If this is checked then the direction indicator will be displayed when the location is invalid.
- **Use Pointer Color:** If this is checked then the pointer valid/invalid colours will also be used to change the colour of the direction indicator.

Class Events

- `PointerDirectionIndicatorPositionSet` - Emitted when the object tooltip is reset.

Unity Events

Adding the `VRTK_PointerDirectionIndicator_UnityEvents` component to `VRTK_PointerDirectionIndicator` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

Initialize/1

```
public virtual void Initialize(VRTK_ControllerEvents events)
```

- Parameters
 - `VRTK_ControllerEvents events` - The Controller Events script that is used to control the direction indicator's rotation.
- Returns
 - *none*

The Initialize method is used to set up the direction indicator.

SetPosition/2

```
public virtual void SetPosition(bool active, Vector3 position)
```

- Parameters
 - `bool active` - Determines if the direction indicator GameObject should be active or not.

- `Vector3 position` - The position to set the direction indicator to.
- Returns
 - *none*

The `SetPosition` method is used to set the world position of the direction indicator.

GetRotation/0

```
public virtual Quaternion GetRotation()
```

- Parameters
 - *none*
- Returns
 - `Quaternion` - The reported rotation of the direction indicator.

The `GetRotation` method returns the current reported rotation of the direction indicator.

SetMaterialColor/2

```
public virtual void SetMaterialColor(Color color, bool validity)
```

- Parameters
 - `Color color` - The colour to update the direction indicator material to.
 - `bool validity` - Determines if the colour being set is based from a valid location or invalid location.
- Returns
 - *none*

The `SetMaterialColor` method sets the current material colour on the direction indicator.

Console Viewer Canvas (VRTK_ConsoleViewer)

Overview

This canvas adds the unity console log to a world game object. To use the prefab, it simply needs to be placed into the scene and it will be visible in world space. It's also possible to child it to other objects such as the controller so it can track where the user is.

It's also recommended to use the Simple Pointer and UI Pointer on a controller to interact with the Console Viewer Canvas as it has a scrollable text area, a button to clear the log and a checkbox to toggle whether the log messages are collapsed.

Inspector Parameters

- **Font Size:** The size of the font the log text is displayed in.
- **Info Message:** The colour of the text for an info log message.
- **Assert Message:** The colour of the text for an assertion log message.
- **Warning Message:** The colour of the text for a warning log message.
- **Error Message:** The colour of the text for an error log message.
- **Exception Message:** The colour of the text for an exception log message.

Class Methods

SetCollapse/1

```
public void SetCollapse(bool state)
```

- Parameters
 - `bool state` - The state of whether to collapse the output messages, `true` will collapse and `false` will not collapse.
- Returns
 - *none*

The `SetCollapse` method determines whether the console will collapse same message output into the same line. A state of `true` will collapse messages and `false` will print the same message for each line.

ClearLog/0

```
public void ClearLog()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ClearLog` method clears the current log view of all messages

Panel Menu Controller (VRTK_PanelMenuController)

Overview

Purpose: top-level controller class to handle the display of up to four child `PanelMenuItemController` items which are displayed as a canvas UI panel.

This script should be attached to a `VRTK_InteractableObject` > first child `GameObject` [`PanelMenuController`]. The [`PanelMenuController`] must have a child `GameObject` [`panel items container`]. The [`panel items container`] must have a `Canvas` component. A [`panel items container`] can have up to four child `GameObject`, each of these contains the UI for a panel that can be displayed by [`PanelMenuController`]. They also have the [`PanelMenuItemController`] script attached to them. The [`PanelMenuItemController`] script intercepts the controller events sent from this [`PanelMenuController`] and passes them onto additional custom event subscriber scripts, which then carry out the required custom UI actions. To show / hide a UI panel, you must first pick up the `VRTK_InteractableObject` and then by pressing the touchpad top/bottom/left/right you can open/close the child UI panel that has been assigned via the Unity Editor panel. Button type UI actions are handled by a trigger press when the panel is open.

Inspector Parameters

- **Rotate Towards:** The `GameObject` the panel should rotate towards, which is the Camera (eye) by default.
- **Zoom Scale Multiplier:** The scale multiplier, which relates to the scale of parent interactable object.
- **Top Panel Menu Item Controller:** The top `PanelMenuItemController`, which is triggered by pressing up on the controller touchpad.
- **Bottom Panel Menu Item Controller:** The bottom `PanelMenuItemController`, which is triggered by pressing down on the controller touchpad.
- **Left Panel Menu Item Controller:** The left `PanelMenuItemController`, which is triggered by pressing left on the controller touchpad.
- **Right Panel Menu Item Controller:** The right `PanelMenuItemController`, which is triggered by pressing right on the controller touchpad.

Class Methods

ToggleMenu/0

```
public virtual void ToggleMenu()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ToggleMenu` method is used to show or hide the menu.

ShowMenu/0

```
public virtual void ShowMenu()
```


- Parameters
 - *none*
- Returns
 - *none*

The ShowMenu method is used to show the menu.

HideMenu/1

```
public virtual void HideMenu(bool force)
```

- Parameters
 - `bool force` - If true then the menu is always hidden.
- Returns
 - *none*

The HideMenu method is used to hide the menu.

HideMenuImmediate/0

```
public virtual void HideMenuImmediate()
```

- Parameters
 - *none*
- Returns
 - *none*

The HideMenuImmediate method is used to immediately hide the menu.

Example

040_Controls_Panel_Menu contains three basic interactive object examples of the PanelMenu in use.

Panel Menu Item Controller (VRTK_PanelMenuItemController)

Overview

Purpose: panel item controller class that intercepts the controller events sent from a [PanelMenuController] and passes them onto additional custom event subscriber scripts, which then carry out the required custom UI actions.

This script should be attached to a VRTK_InteractiveObject > [PanelMenuController] > [panel items

container] > child GameObject (See the [PanelMenuController] class for more details on setup structure.). To show / hide a UI panel, you must first pick up the VRTK_InteractableObject and then by pressing the touchpad top/bottom/left/right you can open/close the child UI panel that has been assigned via the Unity Editor panel.

Class Events

- `PanelMenuItemShowing` - Emitted when the panel menu item is showing.
- `PanelMenuItemHiding` - Emitted when the panel menu item is hiding.
- `PanelMenuItemSwipeLeft` - Emitted when the panel menu item is open and the user swipes left on the controller touchpad.
- `PanelMenuItemSwipeRight` - Emitted when the panel menu item is open and the user swipes right on the controller touchpad.
- `PanelMenuItemSwipeTop` - Emitted when the panel menu item is open and the user swipes top on the controller touchpad.
- `PanelMenuItemSwipeBottom` - Emitted when the panel menu item is open and the user swipes bottom on the controller touchpad.
- `PanelMenuItemTriggerPressed` - Emitted when the panel menu item is open and the user presses the trigger of the controller holding the interactable object.

Event Payload

- `GameObject interactableObject` - The GameObject for the interactable object the PanelMenu is attached to.

Class Methods

SetPanelMenuItemEvent/1

```
public virtual PanelMenuItemControllerEventArgs  
SetPanelMenuItemEvent(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - `PanelMenuItemControllerEventArgs` - The payload for the event.

The `SetPanelMenuItemEvent` is used to build up the event payload.

Show/1

```
public virtual void Show(GameObject interactableObject)
```

- Parameters

- `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The Show method is used to show the menu.

Hide/1

```
public virtual void Hide(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The Hide method is used to show the menu.

SwipeLeft/1

```
public virtual void SwipeLeft(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The SwipeLeft method is used when the control is swiped left.

SwipeRight/1

```
public virtual void SwipeRight(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The SwipeRight method is used when the control is swiped right.

SwipeTop/1

```
public virtual void SwipeTop(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The `SwipeTop` method is used when the control is swiped up.

SwipeBottom/1

```
public virtual void SwipeBottom(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The `SwipeBottom` method is used when the control is swiped down.

TriggerPressed/1

```
public virtual void TriggerPressed(GameObject interactableObject)
```

- Parameters
 - `GameObject interactableObject` - The object the menu is attached to.
- Returns
 - *none*

The `TriggerPressed` method is used when the control action button is pressed.

Example

`040_Controls_Panel_Menu` contains three basic interactive object examples of the `PanelMenu` in use.

Pointers (VRTK/Scripts/Pointers)

A collection of scripts that provide the ability to create pointers and set destination markers in the scene.

- [Destination Marker](#)
- [Pointer](#)
- [Play Area Cursor](#)

Destination Marker (VRTK_DestinationMarker)

Overview

This abstract class provides the ability to emit events of destination markers within the game world. It can be useful for tagging locations for specific purposes such as teleporting.

It is utilised by the `VRTK_BasePointer` for dealing with pointer events when the pointer cursor touches areas within the game world.

Inspector Parameters

- **Enable Teleport:** If this is checked then the teleport flag is set to true in the Destination Set event so teleport scripts will know whether to action the new destination.
- **Target List Policy:** A specified `VRTK_PolicyList` to use to determine whether destination targets will be considered valid or invalid.

Class Events

- `DestinationMarkerEnter` - Emitted when a collision with another collider has first occurred.
- `DestinationMarkerExit` - Emitted when the collision with the other collider ends.
- `DestinationMarkerSet` - Emitted when the destination marker is active in the scene to determine the last destination position (useful for selecting and teleporting).

Unity Events

Adding the `VRTK_DestinationMarker_UnityEvents` component to `VRTK_DestinationMarker` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `float distance` - The distance between the origin and the collided destination.
- `Transform target` - The Transform of the collided destination object.
- `RaycastHit raycastHit` - The optional RaycastHit generated from when the ray collided.
- `Vector3 destinationPosition` - The world position of the destination marker.
- `Quaternion? destinationRotation` - The world rotation of the destination marker.
- `bool forceDestinationPosition` - If true then the given destination position should not be altered by anything consuming the payload.
- `bool enableTeleport` - Whether the destination set event should trigger teleport.
- `VRTK_ControllerReference controllerReference` - The optional reference to the controller controlling the destination marker.

Class Methods

SetNavMeshCheckDistance/1

```
public virtual void SetNavMeshCheckDistance(float distance)
```

- Parameters

- `float distance` - The max distance the nav mesh can be from the sample point to be valid.

- Returns

- *none*

The `SetNavMeshCheckDistance` method sets the max distance the destination marker position can be from the edge of a nav mesh to be considered a valid destination.

SetHeadsetPositionCompensation/1

```
public virtual void SetHeadsetPositionCompensation(bool state)
```

- Parameters

- `bool state` - The state of whether to take the position of the headset within the play area into account when setting the destination marker.

- Returns

- *none*

The `SetHeadsetPositionCompensation` method determines whether the offset position of the headset from the centre of the play area should be taken into consideration when setting the destination marker. If `true` then it will take the offset position into consideration.

SetForceHoverOnRepeatedEnter/1

```
public virtual void SetForceHoverOnRepeatedEnter(bool state)
```

- Parameters

- `bool state` - The state of whether to force the hover on or off.

- Returns

- *none*

The `SetForceHoverOnRepeatedEnter` method is used to set whether the Enter event will forcibly call the Hover event if the existing colliding object is the same as it was the previous enter call.

Pointer (VRTK_Pointer)

extends VRTK_DestinationMarker

Overview

The VRTK Pointer class forms the basis of being able to emit a pointer from a game object (e.g. controller).

The concept of the pointer is it can be activated and deactivated and used to select elements utilising different button combinations if required.

The Pointer requires a Pointer Renderer which is the visualisation of the pointer in the scene.

A Pointer can also be used to extend the interactions of an interacting object such as a controller. This enables pointers to touch (and highlight), grab and use interactable objects.

The Pointer script does not need to go on a controller game object, but if it's placed on another object then a controller must be provided to determine what activates the pointer.

It extends the VRTK_DestinationMarker to allow for destination events to be emitted when the pointer cursor collides with objects.

Inspector Parameters

- **Pointer Renderer:** The specific renderer to use when the pointer is activated. The renderer also determines how the pointer reaches it's destination (e.g. straight line, bezier curve).
- **Activation Button:** The button used to activate/deactivate the pointer.
- **Hold Button To Activate:** If this is checked then the Activation Button needs to be continuously held down to keep the pointer active. If this is unchecked then the Activation Button works as a toggle, the first press/release enables the pointer and the second press/release disables the pointer.
- **Activate On Enable:** If this is checked then the pointer will be toggled on when the script is enabled.
- **Activation Delay:** The time in seconds to delay the pointer being able to be active again.
- **Selection Button:** The button used to execute the select action at the pointer's target position.
- **Select On Press:** If this is checked then the pointer selection action is executed when the Selection Button is pressed down. If this is unchecked then the selection action is executed when the Selection Button is released.
- **Selection Delay:** The time in seconds to delay the pointer being able to execute the select action again.
- **Select After Hover Duration:** The amount of time the pointer can be over the same collider before it automatically attempts to select it. 0f means no selection attempt will be made.
- **Interact With Objects:** If this is checked then the pointer will be an extension of the controller and

able to interact with Interactable Objects.

- **Grab To Pointer Tip:** If `Interact With Objects` is checked and this is checked then when an object is grabbed with the pointer touching it, the object will attach to the pointer tip and not snap to the controller.
- **Controller:** An optional controller that will be used to toggle the pointer. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Interact Use:** An optional `InteractUse` script that will be used when using interactable objects with pointer. If this is left blank then it will attempt to get the `InteractUse` script from the same `GameObject` and if it cannot find one then it will attempt to get it from the attached controller.
- **Custom Origin:** A custom transform to use as the origin of the pointer. If no pointer origin transform is provided then the transform the script is attached to is used.

Class Events

- `ActivationButtonPressed` - Emitted when the pointer activation button is pressed.
- `ActivationButtonReleased` - Emitted when the pointer activation button is released.
- `SelectionButtonPressed` - Emitted when the pointer selection button is pressed.
- `SelectionButtonReleased` - Emitted when the pointer selection button is released.
- `PointerStateValid` - Emitted when the pointer is in a valid state.
- `PointerStateInvalid` - Emitted when the pointer is in an invalid state.

Unity Events

Adding the `VRTK_Pointer_UnityEvents` component to `VRTK_Pointer` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

`IsActivationButtonPressed/0`

```
public virtual bool IsActivationButtonPressed()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the activationButton is being pressed.

The `IsActivationButtonPressed` method returns whether the configured activation button is being pressed.

`IsSelectionButtonPressed/0`


```
public virtual bool IsSelectionButtonPressed()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the selectionButton is being pressed.

The `IsSelectionButtonPressed` method returns whether the configured activation button is being pressed.

PointerEnter/1

```
public virtual void PointerEnter(RaycastHit givenHit)
```

- Parameters

- `RaycastHit givenHit` - The valid collision.

- Returns

- *none*

The `PointerEnter` method emits a `DestinationMarkerEnter` event when the pointer first enters a valid object, it emits a `DestinationMarkerHover` for every following frame that the pointer stays over the valid object.

PointerExit/1

```
public virtual void PointerExit(RaycastHit givenHit)
```

- Parameters

- `RaycastHit givenHit` - The previous valid collision.

- Returns

- *none*

The `PointerExit` method emits a `DestinationMarkerExit` event when the pointer leaves a previously entered object.

CanActivate/0

```
public virtual bool CanActivate()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the pointer can be activated.

The `CanActivate` method is used to determine if the pointer has passed the activation time limit.

CanSelect/0

```
public virtual bool CanSelect()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the pointer can execute the select action.

The `CanSelect` method is used to determine if the pointer has passed the selection time limit.

IsPointerActive/0

```
public virtual bool IsPointerActive()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the pointer is currently active.

The `IsPointerActive` method is used to determine if the pointer's current state is active or not.

ResetActivationTimer/1

```
public virtual void ResetActivationTimer(bool forceZero = false)
```

- Parameters
 - `bool forceZero` - If this is true then the next activation time will be 0.
- Returns
 - *none*

The `ResetActivationTimer` method is used to reset the pointer activation timer to the next valid activation time.

ResetSelectionTimer/1

```
public virtual void ResetSelectionTimer(bool forceZero = false)
```

- Parameters

- `bool forceZero` - If this is true then the next activation time will be 0.
- Returns
 - *none*

The `ResetSelectionTimer` method is used to reset the pointer selection timer to the next valid activation time.

Toggle/1

```
public virtual void Toggle(bool state)
```

- Parameters
 - `bool state` - If true the pointer will be enabled if possible, if false the pointer will be disabled if possible.
- Returns
 - *none*

The `Toggle` method is used to enable or disable the pointer.

IsStateValid/0

```
public virtual bool IsStateValid()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the pointer is in the valid state (showing the valid colour), returns false if the pointer is in the invalid state (showing the invalid colour).

The `IsStateValid` method is used to determine if the pointer is currently in a valid state (i.e. on it's valid colour).

Play Area Cursor (VRTK_PlayAreaCursor)

Overview

The Play Area Cursor is used in conjunction with a Pointer script and displays a representation of the play area where the pointer cursor hits.

Inspector Parameters

- **Use Pointer Color:** If this is checked then the pointer valid/invalid colours will also be used to change the colour of the play area cursor when colliding/not colliding.

- **Play Area Cursor Dimensions:** Determines the size of the play area cursor and collider. If the values are left as zero then the Play Area Cursor will be sized to the calibrated Play Area space.
- **Handle Play Area Cursor Collisions:** If this is checked then if the play area cursor is colliding with any other object then the pointer colour will change to the `Pointer Miss Color` and the `DestinationMarkerSet` event will not be triggered, which will prevent teleporting into areas where the play area will collide.
- **Headset Out Of Bounds Is Collision:** If this is checked then if the user's headset is outside of the play area cursor bounds then it is considered a collision even if the play area isn't colliding with anything.
- **Display On Invalid Location:** If this is checked then the play area cursor will be displayed when the location is invalid.
- **Target List Policy:** A specified `VRTK_PolicyList` to use to determine whether the play area cursor collisions will be acted upon.
- **Valid Location Object:** A custom `GameObject` to use for the play area cursor representation for when the location is valid.
- **Invalid Location Object:** A custom `GameObject` to use for the play area cursor representation for when the location is invalid.

Class Events

- `PlayAreaCursorStartCollision` - Emitted when the play area collides with another object.
- `PlayAreaCursorEndCollision` - Emitted when the play area stops colliding with another object.

Unity Events

Adding the `VRTK_PlayAreaCursor_UnityEvents` component to `VRTK_PlayAreaCursor` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `collidedWith` - The collider that is/was being collided with.

Class Methods

HasCollided/0

```
public virtual bool HasCollided()
```

- Parameters
 - *none*
- Returns
 - `bool` - A `bool` to determine the state of collision. `true` if the play area is colliding with a valid object and `false` if not.

The HasCollided method returns the state of whether the play area cursor has currently collided with another valid object.

SetHeadsetPositionCompensation/1

```
public virtual void SetHeadsetPositionCompensation(bool state)
```

- Parameters

- `bool state` - The state of whether to take the position of the headset within the play area into account when setting the destination marker.

- Returns

- *none*

The SetHeadsetPositionCompensation method determines whether the offset position of the headset from the centre of the play area should be taken into consideration when setting the destination marker. If `true` then it will take the offset position into consideration.

SetPlayAreaCursorCollision/2

```
public virtual void SetPlayAreaCursorCollision(bool state, Collider collider  
= null)
```

- Parameters

- `bool state` - The state of whether to check for play area collisions.
- `Collider collider` - The state of whether to check for play area collisions.

- Returns

- *none*

The SetPlayAreaCursorCollision method determines whether play area collisions should be taken into consideration with the play area cursor.

SetMaterialColor/2

```
public virtual void SetMaterialColor(Color color, bool validity)
```

- Parameters

- `Color color` - The colour to update the play area cursor material to.
- `bool validity` - Determines if the colour being set is based from a valid location or invalid location.

- Returns

- *none*

The SetMaterialColor method sets the current material colour on the play area cursor.

SetPlayAreaCursorTransform/1

```
public virtual void SetPlayAreaCursorTransform(Vector3 location)
```

- Parameters

- `Vector3 location` - The location where to draw the play area cursor.

- Returns

- *none*

The `SetPlayAreaCursorTransform` method is used to update the position of the play area cursor in world space to the given location.

ToggleState/1

```
public virtual void ToggleState(bool state)
```

- Parameters

- `bool state` - The state of whether to show or hide the play area cursor.

- Returns

- *none*

The `ToggleState` method enables or disables the visibility of the play area cursor.

IsActive/0

```
public virtual bool IsActive()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the play area cursor `GameObject` is active.

The `IsActive` method returns whether the play area cursor game object is active or not.

GetPlayAreaContainer/0

```
public virtual GameObject GetPlayAreaContainer()
```

- Parameters

- *none*

- Returns

- `GameObject` - The `GameObject` that is the container of the play area cursor.

The `GetPlayAreaContainer` method returns the created game object that holds the play area cursor representation.

ToggleVisibility/1

```
public virtual void ToggleVisibility(bool state)
```

- Parameters

- `bool state` - The state of the cursor visibility. True will show the renderers and false will hide the renderers.

- Returns

- *none*

The `ToggleVisibility` method enables or disables the play area cursor renderers to allow the cursor to be seen or hidden.

Example

`VRTK/Examples/012_Controller_PointerWithAreaCollision` shows how a Bezier Pointer with the Play Area Cursor and Collision Detection enabled can be used to traverse a game area but not allow teleporting into areas where the walls or other objects would fall into the play area space enabling the user to enter walls.

Pointer Renderers (VRTK/Scripts/Pointers/PointerRenderers)

This directory contains scripts that are used to provide different renderers for the `VRTK_Pointer`.

- [Base Pointer Renderer](#)
 - [Straight Pointer Renderer](#)
 - [Bezier Pointer Renderer](#)
-

Base Pointer Renderer (PointerOriginSmoothingSettings)

Overview

Specifies the smoothing to be applied to the pointer.

Inspector Parameters

- **Smooths Position:** Whether or not to smooth the position of the pointer origin when positioning

the pointer tip.

- **Max Allowed Per Frame Distance Difference:** The maximum allowed distance between the unsmoothed pointer origin and the smoothed pointer origin per frame to use for smoothing.
- **Smooths Rotation:** Whether or not to smooth the rotation of the pointer origin when positioning the pointer tip.
- **Max Allowed Per Frame Angle Difference:** The maximum allowed angle between the unsmoothed pointer origin and the smoothed pointer origin per frame to use for smoothing.
- **Playarea Cursor:** An optional Play Area Cursor generator to add to the destination position of the pointer tip.
- **Direction Indicator:** A custom `VRTK_PointerDirectionIndicator` to use to determine the rotation given to the destination set event.
- **Custom Raycast:** A custom raycaster to use for the pointer's raycasts to ignore.
- **Pointer Origin Smoothing Settings:** Specifies the smoothing to be applied to the pointer origin when positioning the pointer tip.
- **Valid Collision Color:** The colour to change the pointer materials when the pointer collides with a valid object. Set to `Color.clear` to bypass changing material colour on valid collision.
- **Invalid Collision Color:** The colour to change the pointer materials when the pointer is not colliding with anything or with an invalid object. Set to `Color.clear` to bypass changing material colour on invalid collision.
- **Tracer Visibility:** Determines when the main tracer of the pointer renderer will be visible.
- **Cursor Visibility:** Determines when the cursor/tip of the pointer renderer will be visible.

Class Variables

- `public enum VisibilityStates` - States of Pointer Visibility.
 - `OnWhenActive` - Only shows the object when the pointer is active.
 - `AlwaysOn` - Ensures the object is always.
 - `AlwaysOff` - Ensures the object beam is never visible.

Class Methods

GetPointerObjects/0

```
public abstract GameObject[] GetPointerObjects();
```

- Parameters
 - *none*
- Returns
 - `GameObject[]` - An array of pointer auto generated GameObjects.

The `GetPointerObjects` returns an array of the auto generated GameObjects associated with the pointer.

InitializePointer/4


```
public virtual void InitializePointer(VRTK_Pointer givenPointer,
VRTK_PolicyList givenInvalidListPolicy, float givenNavMeshCheckDistance,
bool givenHeadsetPositionCompensation)
```

- Parameters

- VRTK_Pointer givenPointer - The VRTK_Pointer that is controlling the pointer renderer.
- VRTK_PolicyList givenInvalidListPolicy - The VRTK_PolicyList for managing valid and invalid pointer locations.
- float givenNavMeshCheckDistance - The given distance from a nav mesh that the pointer can be to be valid.
- bool givenHeadsetPositionCompensation - Determines whether the play area cursor will take the headset position within the play area into account when being displayed.

- Returns

- *none*

The InitializePointer method is used to set up the state of the pointer renderer.

ResetPointerObjects/0

```
public virtual void ResetPointerObjects()
```

- Parameters

- *none*

- Returns

- *none*

The ResetPointerObjects method is used to destroy any existing pointer objects and recreate them at runtime.

Toggle/2

```
public virtual void Toggle(bool pointerState, bool actualState)
```

- Parameters

- bool pointerState - The activation state of the pointer.
- bool actualState - The actual state of the activation button press.

- Returns

- *none*

The Toggle Method is used to enable or disable the pointer renderer.

ToggleInteraction/1

```
public virtual void ToggleInteraction(bool state)
```

- Parameters
 - `bool state` - If true then the object interactor will be enabled.
- Returns
 - *none*

The `ToggleInteraction` method is used to enable or disable the controller extension interactions.

UpdateRenderer/0

```
public virtual void UpdateRenderer()
```

- Parameters
 - *none*
- Returns
 - *none*

The `UpdateRenderer` method is used to run an Update routine on the pointer.

GetDestinationHit/0

```
public virtual RaycastHit GetDestinationHit()
```

- Parameters
 - *none*
- Returns
 - `RaycastHit` - The `RaycastHit` containing the information where the pointer is hitting.

The `GetDestinationHit` method is used to get the `RaycastHit` of the pointer destination.

ValidPlayArea/0

```
public virtual bool ValidPlayArea()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if there is a valid play area and no collisions. Returns false if there is no valid play area or there is but with a collision detected.

The `ValidPlayArea` method is used to determine if there is a valid play area and if it has had any collisions.

IsVisible/0

```
public virtual bool IsVisible()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if either the tracer or cursor renderers are visible. Returns false if none are visible.

The `IsVisible` method determines if the pointer renderer is at all visible by checking the state of the tracer and the cursor.

IsTracerVisible/0

```
public virtual bool IsTracerVisible()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the tracer renderers are visible.

The `IsTracerVisible` method determines if the pointer tracer renderer is visible.

IsCursorVisible/0

```
public virtual bool IsCursorVisible()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the cursor renderers are visible.

The `IsCursorVisible` method determines if the pointer cursor renderer is visible.

IsValidCollision/0

```
public virtual bool IsValidCollision()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the pointer is in a valid collision, returns false if the pointer is in an invalid collision state.

The `IsValidCollision` method determines if the pointer is currently in it's valid collision state.

GetObjectInteractor/0

```
public virtual GameObject GetObjectInteractor()
```

- Parameters

- *none*

- Returns

- `GameObject` - The auto generated object interactor `GameObject`.
- `GameObject` -

The `GetObjectInteractor` method returns the auto generated `GameObject` that acts as the controller extension for interacting with objects.

Straight Pointer Renderer (VRTK_StraightPointerRenderer)

extends [VRTK_BasePointerRenderer](#)

Overview

The Straight Pointer Renderer emits a coloured beam from the end of the object it is attached to and simulates a laser beam.

It can be useful for pointing to objects within a scene and it can also determine the object it is pointing at and the distance the object is from the controller the beam is being emitted from.

Inspector Parameters

- **Maximum Length:** The maximum length the pointer tracer can reach.
- **Scale Factor:** The scale factor to scale the pointer tracer object by.
- **Cursor Scale Multiplier:** The scale multiplier to scale the pointer cursor object by in relation to the `Scale Factor`.
- **Cursor Match Target Rotation:** The cursor will be rotated to match the angle of the target surface if this is true, if it is false then the pointer cursor will always be horizontal.
- **Cursor Distance Rescale:** Rescale the cursor proportionally to the distance from the tracer origin.
- **Maximum Cursor Scale:** The maximum scale the cursor is allowed to reach. This is only used when rescaling the cursor proportionally to the distance from the tracer origin.
- **Custom Tracer:** A custom game object to use as the appearance for the pointer tracer. If this is

empty then a Box primitive will be created and used.

- **Custom Cursor:** A custom game object to use as the appearance for the pointer cursor. If this is empty then a Sphere primitive will be created and used.

Class Methods

UpdateRenderer/0

```
public override void UpdateRenderer()
```

- Parameters
 - *none*
- Returns
 - *none*

The UpdateRenderer method is used to run an Update routine on the pointer.

GetPointerObjects/0

```
public override GameObject[] GetPointerObjects()
```

- Parameters
 - *none*
- Returns
 - `GameObject[]` - An array of pointer auto generated GameObjects.

The GetPointerObjects returns an array of the auto generated GameObjects associated with the pointer.

Example

VRTK/Examples/003_Controller_SimplePointer shows the simple pointer in action and code examples of how the events are utilised and listened to can be viewed in the script

VRTK/Examples/Resources/Scripts/VRTK_ControllerPointerEvents_ListenerExample.cs

Bezier Pointer Renderer (VRTK_BezierPointerRenderer)

extends VRTK_BasePointerRenderer

Overview

The Bezier Pointer Renderer emits a curved line (made out of game objects) from the end of the attached object to a point on a ground surface (at any height).

It is more useful than the Simple Pointer Renderer for traversing objects of various heights as the end point can be curved on top of objects that are not visible to the user.

The bezier curve generation code is in another script located at `VRTK/Scripts/Internal/VRTK_CurveGenerator.cs` and was heavily inspired by the tutorial and code from [Catlike Coding](#).

Inspector Parameters

- **Maximum Length:** The maximum length of the projected beam. The x value is the length of the forward beam, the y value is the length of the downward beam.
- **Tracer Density:** The number of items to render in the bezier curve tracer beam. A high number here will most likely have a negative impact of game performance due to large number of rendered objects.
- **Cursor Radius:** The size of the ground cursor. This number also affects the size of the objects in the bezier curve tracer beam. The larger the radius, the larger the objects will be.
- **Height Limit Angle:** The maximum angle in degrees of the origin before the beam curve height is restricted. A lower angle setting will prevent the beam being projected high into the sky and curving back down.
- **Curve Offset:** The amount of height offset to apply to the projected beam to generate a smoother curve even when the beam is pointing straight.
- **Rescale Tracer:** Rescale each tracer element according to the length of the Bezier curve.
- **Cursor Match Target Rotation:** The cursor will be rotated to match the angle of the target surface if this is true, if it is false then the pointer cursor will always be horizontal.
- **Collision Check Frequency:** The number of points along the bezier curve to check for an early beam collision. Useful if the bezier curve is appearing to clip through teleport locations. 0 won't make any checks and it will be capped at `Pointer Density`. The higher the number, the more CPU intensive the checks become.
- **Custom Tracer:** A custom game object to use as the appearance for the pointer tracer. If this is empty then a collection of Sphere primitives will be created and used.
- **Custom Cursor:** A custom game object to use as the appearance for the pointer cursor. If this is empty then a Cylinder primitive will be created and used.
- **Valid Location Object:** A custom game object can be applied here to appear only if the location is valid.
- **Invalid Location Object:** A custom game object can be applied here to appear only if the location is invalid.

Class Methods

UpdateRenderer/0

```
public override void UpdateRenderer()
```

- Parameters
 - *none*
- Returns
 - *none*

The UpdateRenderer method is used to run an Update routine on the pointer.

GetPointerObjects/0

```
public override GameObject[] GetPointerObjects()
```

- Parameters
 - *none*
- Returns
 - `GameObject[]` - An array of pointer auto generated GameObjects.

The GetPointerObjects returns an array of the auto generated GameObjects associated with the pointer.

Example

`VRTK/Examples/009_Controller_BezierPointer` is used in conjunction with the Height Adjust Teleporter shows how it is possible to traverse different height objects using the curved pointer without needing to see the top of the object.

``VRTK/Examples/036_Controller_CustomCompoundPointer'` shows how to display an object (a teleport beam) only if the teleport location is valid, and can create an animated trail along the tracer curve.

Locomotion (VRTK/Scripts/Locomotion)

A collection of scripts that provide varying methods of moving the user around the scene.

- [Basic Teleport](#)
- [Height Adjust Teleport](#)
- [Dash Teleport](#)
- [Teleport Disable On Headset Collision](#)
- [Teleport Disable On Controller Obscured](#)
- [Object Control](#)
- [Touchpad Control](#)

- [Button Control](#)
 - [Move In Place](#)
 - [Player Climb](#)
 - [Room Extender](#)
-

Basic Teleport (VRTK_BasicTeleport)

Overview

The basic teleporter updates the user's x/z position in the game world to the position of a Base Pointer's tip location which is set via the `DestinationMarkerSet` event.

The y position is never altered so the basic teleporter cannot be used to move up and down game objects as it only allows for travel across a flat plane.

Inspector Parameters

- **Blink To Color:** The colour to fade to when blinking on teleport.
- **Blink Transition Speed:** The fade blink speed can be changed on the basic teleport script to provide a customised teleport experience. Setting the speed to 0 will mean no fade blink effect is present.
- **Distance Blink Delay:** A range between 0 and 32 that determines how long the blink transition will stay blacked out depending on the distance being teleported. A value of 0 will not delay the teleport blink effect over any distance, a value of 32 will delay the teleport blink fade in even when the distance teleported is very close to the original position. This can be used to simulate time taking longer to pass the further a user teleports. A value of 16 provides a decent basis to simulate this to the user.
- **Headset Position Compensation:** If this is checked then the teleported location will be the position of the headset within the play area. If it is unchecked then the teleported location will always be the centre of the play area even if the headset position is not in the centre of the play area.
- **Target List Policy:** A specified `VRTK_PolicyList` to use to determine whether destination targets will be acted upon by the Teleporter.
- **Nav Mesh Limit Distance:** The max distance the teleport destination can be outside the nav mesh to be considered valid. If a value of 0 is given then the nav mesh restrictions will be ignored.

Class Events

- `Teleporting` - Emitted when the teleport process has begun.
- `Teleported` - Emitted when the teleport process has successfully completed.

Unity Events

Adding the `VRTK_BasicTeleport_UnityEvents` component to `VRTK_BasicTeleport` object allows

access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `float distance` - The distance between the origin and the collided destination.
- `Transform target` - The Transform of the collided destination object.
- `RaycastHit raycastHit` - The optional `RaycastHit` generated from when the ray collided.
- `Vector3 destinationPosition` - The world position of the destination marker.
- `Quaternion? destinationRotation` - The world rotation of the destination marker.
- `bool forceDestinationPosition` - If true then the given destination position should not be altered by anything consuming the payload.
- `bool enableTeleport` - Whether the destination set event should trigger teleport.
- `VRTK_ControllerReference controllerReference` - The optional reference to the controller controlling the destination marker.

Class Methods

InitDestinationSetListener/2

```
public virtual void InitDestinationSetListener(GameObject markerMaker, bool register)
```

- Parameters
 - `GameObject markerMaker` - The game object that is used to generate destination marker events, such as a controller.
 - `bool register` - Determines whether to register or unregister the listeners.
- Returns
 - *none*

The `InitDestinationSetListener` method is used to register the teleport script to listen to events from the given game object that is used to generate destination markers. Any destination set event emitted by a registered game object will initiate the teleport to the given destination location.

ToggleTeleportEnabled/1

```
public virtual void ToggleTeleportEnabled(bool state)
```

- Parameters
 - `bool state` - Toggles whether the teleporter is enabled or disabled.
- Returns
 - *none*

The `ToggleTeleportEnabled` method is used to determine whether the teleporter will initiate a teleport on a destination set event, if the state is true then the teleporter will work as normal, if the state is false then the teleporter will not be operational.

ValidLocation/2

```
public virtual bool ValidLocation(Transform target, Vector3 destinationPosition)
```

- Parameters

- `Transform target` - The Transform that the destination marker is touching.
- `Vector3 destinationPosition` - The position in world space that is the destination.

- Returns

- `bool` - Returns true if the target is a valid location.

The `ValidLocation` method determines if the given target is a location that can be teleported to

Teleport/1

```
public virtual void Teleport(DestinationMarkerEventArgs teleportArgs)
```

- Parameters

- `DestinationMarkerEventArgs teleportArgs` - The pseudo Destination Marker event for the teleport action.

- Returns

- *none*

The `Teleport/1` method calls the teleport to update position without needing to listen for a Destination Marker event.

Teleport/4

```
public virtual void Teleport(Transform target, Vector3 destinationPosition, Quaternion? destinationRotation = null, bool forceDestinationPosition = false)
```

- Parameters

- `Transform target` - The Transform of the destination object.
- `Vector3 destinationPosition` - The world position to teleport to.
- `Quaternion? destinationRotation` - The world rotation to teleport to.
- `bool forceDestinationPosition` - If true then the given destination position should not be altered by anything consuming the payload.

- Returns
 - *none*

The Teleport/4 method calls the teleport to update position without needing to listen for a Destination Marker event. It will build a destination marker out of the provided parameters.

ForceTeleport/2

```
public virtual void ForceTeleport(Vector3 destinationPosition, Quaternion? destinationRotation = null)
```

- Parameters
 - `Vector3 destinationPosition` - The world position to teleport to.
 - `Quaternion? destinationRotation` - The world rotation to teleport to.
- Returns
 - *none*

The ForceTeleport method forces the position to update to a given destination and ignores any target checking or floor adjustment.

Example

`VRTK/Examples/004_CameraRig_BasicTeleport` uses the `VRTK_SimplePointer` script on the Controllers to initiate a laser pointer by pressing the `Touchpad` on the controller and when the laser pointer is deactivated (release the `Touchpad`) then the user is teleported to the location of the laser pointer tip as this is where the pointer destination marker position is set to.

Height Adjust Teleport (VRTK_HeightAdjustTeleport)

extends `VRTK_BasicTeleport`

Overview

The height adjust teleporter extends the basic teleporter and allows for the y position of the user's position to be altered based on whether the teleport location is on top of another object.

Inspector Parameters

- **Snap To Nearest Floor:** If this is checked, then the teleported Y position will snap to the nearest available below floor. If it is unchecked, then the teleported Y position will be where ever the destination Y position is.
- **Custom Raycast:** A custom raycaster to use when raycasting to find floors.

Example

`VRTK/Examples/007_CameraRig_HeightAdjustTeleport` has a collection of varying height objects that the user can either walk up and down or use the laser pointer to climb on top of them.

`VRTK/Examples/010_CameraRig_TerrainTeleporting` shows how the teleportation of a user can also traverse terrain colliders.

`VRTK/Examples/020_CameraRig_MeshTeleporting` shows how the teleportation of a user can also traverse mesh colliders.

Dash Teleport (VRTK_DashTeleport)

extends `VRTK_HeightAdjustTeleport`

Overview

The dash teleporter extends the height adjust teleporter and allows to have the user's position dashing to a new teleport location.

The basic principle is to dash for a very short amount of time, to avoid sim sickness. The default value is 100 milliseconds. This value is fixed for all normal and longer distances. When the distances get very short the minimum speed is clamped to 50 mps, so the dash time becomes even shorter.

The minimum distance for the fixed time dash is determined by the `minSpeed` and `normalLerpTime` values, if you want to always lerp with a fixed mps speed instead, set the `normalLerpTime` to a high value. Right before the teleport a capsule is cast towards the target and registers all colliders blocking the way. These obstacles are then broadcast in an event so that for example their gameobjects or renderers can be turned off while the dash is in progress.

Inspector Parameters

- **Normal Lerp Time:** The fixed time it takes to dash to a new position.
- **Min Speed Mps:** The minimum speed for dashing in meters per second.
- **Capsule Top Offset:** The Offset of the CapsuleCast above the camera.
- **Capsule Bottom Offset:** The Offset of the CapsuleCast below the camera.
- **Capsule Radius:** The radius of the CapsuleCast.

Class Events

- `WillDashThruObjects` - Emitted when the CapsuleCast towards the target has found that obstacles are in the way.
- `DashedThruObjects` - Emitted when obstacles have been crossed and the dash has ended.

Unity Events

Adding the `VRTK_DashTeleport_UnityEvents` component to `VRTK_DashTeleport` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `RaycastHit[] hits` - An array of objects that the CapsuleCast has collided with.

Example

`SteamVR_Unity_Toolkit/Examples/038_CameraRig_DashTeleport` shows how to turn off the mesh renderers of objects that are in the way during the dash.

Teleport Disable On Headset Collision (VRTK_TeleportDisableOnHeadsetCollision)

Overview

The purpose of the Teleport Disable On Headset Collision script is to detect when the headset is colliding with a valid object and prevent teleportation from working. This is to ensure that if a user is clipping their head into a wall then they cannot teleport to an area beyond the wall.

Teleport Disable On Controller Obscured (VRTK_TeleportDisableOnControllerObscured)

Overview

The purpose of the Teleport Disable On Controller Obscured script is to detect when the headset does not have a line of sight to the controllers and prevent teleportation from working. This is to ensure that if a user is clipping their controllers through a wall then they cannot teleport to an area beyond the wall.

Object Control (VRTK_ObjectControl)

Overview

An abstract class to provide a mechanism to control an object based on controller input.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Inspector Parameters

- **Controller:** The controller to read the controller events from. If this is blank then it will attempt to get a controller events script from the same GameObject.
- **Device For Direction:** The direction that will be moved in is the direction of this device.
- **Disable Other Controls On Active:** If this is checked then whenever the axis on the attached controller is being changed, all other object control scripts of the same type on other controllers will be disabled.
- **Affect On Falling:** If a `VRTK_BodyPhysics` script is present and this is checked, then the object control will affect the play area whilst it is falling.
- **Control Override Object:** An optional game object to apply the object control to. If this is blank then the PlayArea will be controlled.

Class Variables

- `public enum DirectionDevices` - Devices for providing direction.
 - `Headset` - The headset device.
 - `LeftController` - The left controller device.
 - `RightController` - The right controller device.
 - `ControlledObject` - The controlled object.

Class Events

- `XAxisChanged` - Emitted when the X Axis Changes.
- `YAxisChanged` - Emitted when the Y Axis Changes.

Unity Events

Adding the `VRTK_ObjectControl_UnityEvents` component to `VRTK_ObjectControl` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `GameObject controlledGameObject` - The GameObject that is going to be affected.
 - `Transform directionDevice` - The device that is used for the direction.
 - `Vector3 axisDirection` - The axis that is being affected.
 - `Vector3 axis` - The value of the current touchpad touch point based across the axis direction.
 - `float deadzone` - The value of the deadzone based across the axis direction.
 - `bool currentlyFalling` - Whether the controlled GameObject is currently falling.
 - `bool modifierActive` - Whether the modifier button is pressed.
-

Touchpad Control (VRTK_TouchpadControl)

extends [VRTK_ObjectControl](#)

Overview

The ability to control an object with the touchpad based on the position of the finger on the touchpad axis.

The Touchpad Control script forms the stub to allow for pre-defined actions to execute when the touchpad axis changes.

This is enabled by the Touchpad Control script emitting an event each time the X axis and Y Axis on the touchpad change and the corresponding Object Control Action registers with the appropriate axis event. This means that multiple Object Control Actions can be triggered per axis change.

This script is placed on the Script Alias of the Controller that is required to be affected by changes in the touchpad.

If the controlled object is the play area and `VRTK_BodyPhysics` is also available, then additional logic is processed when the user is falling such as preventing the touchpad control from affecting a falling user.

Inspector Parameters

- **Primary Activation Button:** An optional button that has to be engaged to allow the touchpad control to activate.
- **Action Modifier Button:** An optional button that when engaged will activate the modifier on the touchpad control action.
- **Axis Deadzone:** Any input on the axis will be ignored if it is within this deadzone threshold. Between 0f and 1f.

Example

`VRTK/Examples/017_CameraRig_TouchpadWalking` has a collection of walls and slopes that can be traversed by the user with the touchpad. There is also an area that can only be traversed if the user is crouching.

Button Control (VRTK_ButtonControl)

extends [VRTK_ObjectControl](#)

Overview

The ability to control an object with a button press on a given button to control a specified direction.

The Button Control script forms the stub to allow for pre-defined actions to execute when a button press affects a direction axis.

This is enabled by the Button Control script emitting an event each time the pseudo X axis and pseudo Y Axis are changed by a button press and the corresponding Object Control Action registers with the appropriate axis event. This means that multiple Object Control Actions can be triggered per axis change.

This script is placed on the Script Alias of the Controller that is required to be affected by button presses.

If the controlled object is the play area and `VRTK_BodyPhysics` is also available, then additional logic is processed when the user is falling such as preventing the button control from affecting a falling user.

Inspector Parameters

- **Forward Button:** The button to set the y axis to +1.
 - **Backward Button:** The button to set the y axis to -1.
 - **Left Button:** The button to set the x axis to -1.
 - **Right Button:** The button to set the x axis to +1.
-

Move In Place (VRTK_MoveInPlace)

Overview

Move In Place allows the user to move the play area by calculating the y-movement of the user's headset and/or controllers. The user is propelled forward the more they are moving. This simulates moving in game by moving in real life.

This locomotion method is based on Immersive Movement, originally created by Highsight. Thanks to KJack (author of Arm Swinger) for additional work.

Inspector Parameters

- **Left Controller:** If this is checked then the left controller touchpad will be enabled to move the play area.
- **Right Controller:** If this is checked then the right controller touchpad will be enabled to move the play area.
- **Engage Button:** Select which button to hold to engage Move In Place.

- **Control Options:** Select which trackables are used to determine movement.
- **Direction Method:** How the user's movement direction will be determined. The Gaze method tends to lead to the least motion sickness. Smart decoupling is still a Work In Progress.
- **Speed Scale:** Lower to decrease speed, raise to increase.
- **Max Speed:** The max speed the user can move in game units. (If 0 or less, max speed is uncapped)
- **Deceleration:** The speed in which the play area slows down to a complete stop when the user is no longer pressing the engage button. This deceleration effect can ease any motion sickness that may be suffered.
- **Falling Deceleration:** The speed in which the play area slows down to a complete stop when the user is falling.
- **Smart Decouple Threshold:** The degree threshold that all tracked objects (controllers, headset) must be within to change direction when using the Smart Decoupling Direction Method.
- **Sensitivity:** The maximum amount of movement required to register in the virtual world. Decreasing this will increase acceleration, and vice versa.
- **Body Physics:** An optional Body Physics script to check for potential collisions in the moving direction. If any potential collision is found then the move will not take place. This can help reduce collision tunnelling.

Class Variables

- `public enum ControlOptions` - Options for testing if a play space fall is valid.
 - `HeadsetAndControllers` - Track both headset and controllers for movement calculations.
 - `ControllersOnly` - Track only the controllers for movement calculations.
 - `HeadsetOnly` - Track only headset for movement calucations.
- `public enum DirectionalMethod` - Options for which method is used to determine player direction while moving.
 - `Gaze` - Player will always move in the direction they are currently looking.
 - `ControllerRotation` - Player will move in the direction that the controllers are pointing (averaged).
 - `DumbDecoupling` - Player will move in the direction they were first looking when they engaged Move In Place.
 - `SmartDecoupling` - Player will move in the direction they are looking only if their headset point the same direction as their controllers.
 - `EngageControllerRotationOnly` - Player will move in the direction that the controller with the engage button pressed is pointing.
 - `LeftControllerRotationOnly` - Player will move in the direction that the left controller is pointing.
 - `RightControllerRotationOnly` - Player will move in the direction that the right controller is pointing.

Class Methods

SetControlOptions/1

```
public virtual void SetControlOptions(ControlOptions givenControlOptions)
```

- Parameters

- `ControlOptions givenControlOptions` - The control options to set the current control options to.

- Returns

- *none*

Set the control options and modify the trackables to match.

GetMovementDirection/0

```
public virtual Vector3 GetMovementDirection()
```

- Parameters

- *none*

- Returns

- `Vector3` - Returns a vector representing the player's current movement direction.

The `GetMovementDirection` method will return the direction the player is moving.

GetSpeed/0

```
public virtual float GetSpeed()
```

- Parameters

- *none*

- Returns

- `float` - Returns a float representing the player's current movement speed.

The `GetSpeed` method will return the current speed the player is moving at.

Example

`VRTK/Examples/042_CameraRig_MoveInPlace` demonstrates how the user can move and traverse colliders by either swinging the controllers in a walking fashion or by running on the spot utilising the head bob for movement.

Player Climb (VRTK_PlayerClimb)

Overview

The Player Climb allows player movement based on grabbing of `VRTK_InteractableObject` objects that have a `Climbable` grab attach script. Because it works by grabbing, each controller should have a `VRTK_InteractGrab` and `VRTK_InteractTouch` component attached.

Inspector Parameters

- **Use Player Scale:** Will scale movement up and down based on the player transform's scale.
- **Body Physics:** The VRTK Body Physics script to use for dealing with climbing and falling. If this is left blank then the script will need to be applied to the same `GameObject`.
- **Teleporter:** The VRTK Teleport script to use when snapping to nearest floor on release. If this is left blank then a Teleport script will need to be applied to the same `GameObject`.
- **Headset Collision:** The VRTK Headset Collision script to use for determining if the user is climbing inside a collidable object. If this is left blank then the script will need to be applied to the same `GameObject`.
- **Position Rewind:** The VRTK Position Rewind script to use for dealing resetting invalid positions. If this is left blank then the script will need to be applied to the same `GameObject`.

Class Events

- `PlayerClimbStarted` - Emitted when player climbing has started.
- `PlayerClimbEnded` - Emitted when player climbing has ended.

Unity Events

Adding the `VRTK_PlayerClimb_UnityEvents` component to `VRTK_PlayerClimb` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_ControllerReference controllerReference` - The reference to the controller doing the interaction.
- `GameObject target` - The `GameObject` of the interactable object that is being interacted with by the controller.

Example

`VRTK/Examples/037_CameraRig_ClimbingFalling` shows how to set up a scene with player climbing. There are many different examples showing how the same system can be used in unique ways.

Room Extender (VRTK_RoomExtender)

Overview

This script allows the `playArea` to move with the user. The play area is only moved when at the edge

of a defined circle.

There is an additional script `VRTK_RoomExtender_PlayAreaGizmo` which can be attached alongside to visualize the extended playArea within the Editor.

Inspector Parameters

- **Movement Function:** This determines the type of movement used by the extender.
- **Additional Movement Enabled:** This is the a public variable to enable the additional movement. This can be used in other scripts to toggle the play area movement.
- **Additional Movement Enabled On Button Press:** This configures the controls of the RoomExtender. If this is true then the touchpad needs to be pressed to enable it. If this is false then it is disabled by pressing the touchpad.
- **Additional Movement Multiplier:** This is the factor by which movement at the edge of the circle is amplified. 0 is no movement of the play area. Higher values simulate a bigger play area but may be too uncomfortable.
- **Head Zone Radius:** This is the size of the circle in which the playArea is not moved and everything is normal. If it is too low it becomes uncomfortable when crouching.
- **Debug Transform:** This transform visualises the circle around the user where the play area is not moved. In the demo scene this is a cylinder at floor level. Remember to turn off collisions.

Class Variables

- `public enum MovementFunction` - Movement methods.
 - `Nonlinear` - Moves the head with a non-linear drift movement.
 - `LinearDirect` - Moves the headset in a direct linear movement.

Example

`VRTK/Examples/028_CameraRig_RoomExtender` shows how the RoomExtender script is controlled by a `VRTK_RoomExtender_Controller Example` script located at both controllers. Pressing the `Touchpad` on the controller activates the Room Extender. The Additional Movement Multiplier is changed based on the touch distance to the centre of the touchpad.

Object Control Actions (VRTK/Scripts/Locomotion/ObjectControlActions)

This directory contains scripts that are used to provide different actions when using Object Control.

- [Base Object Control Action](#)
- [Slide Object Control Action](#)

- [Rotate Object Control Action](#)
 - [Snap Rotate Object Control Action](#)
 - [Warp Object Control Action](#)
-

Base Object Control Action (VRTK_BaseObjectControlAction)

Overview

The Base Object Control Action script is an abstract class that all object control action scripts inherit.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Inspector Parameters

- **Object Control Script:** The Object Control script to receive axis change events from.
 - **Listen On Axis Change:** Determines which Object Control Axis event to listen to.
-

Slide Object Control Action (VRTK_SlideObjectControlAction)

extends [VRTK_BaseObjectControlAction](#)

Overview

The Slide Object Control Action script is used to slide the controlled GameObject around the scene when changing the axis.

The effect is a smooth sliding motion in forward and sideways directions to simulate walking.

Inspector Parameters

- **Maximum Speed:** The maximum speed the controlled object can be moved in based on the position of the axis.
- **Deceleration:** The rate of speed deceleration when the axis is no longer being changed.
- **Falling Deceleration:** The rate of speed deceleration when the axis is no longer being changed and the object is falling.
- **Speed Multiplier:** The speed multiplier to be applied when the modifier button is pressed.
- **Body Physics:** An optional Body Physics script to check for potential collisions in the moving direction. If any potential collision is found then the move will not take place. This can help reduce collision tunnelling.

Example

VRTK/Examples/017_CameraRig_TouchpadWalking has a collection of walls and slopes that can be traversed by the user with the touchpad. There is also an area that can only be traversed if the user is crouching.

To enable the Slide Object Control Action, ensure one of the `TouchpadControlOptions` children (located under the Controller script alias) has the `Slide` control script active.

Rotate Object Control Action (VRTK_RotateObjectControlAction)

extends VRTK_BaseObjectControlAction

Overview

The Rotate Object Control Action script is used to rotate the controlled GameObject around the up vector when changing the axis.

The effect is a smooth rotation to simulate turning.

Inspector Parameters

- **Maximum Rotation Speed:** The maximum speed the controlled object can be rotated based on the position of the axis.
- **Rotation Multiplier:** The rotation multiplier to be applied when the modifier button is pressed.

Example

VRTK/Examples/017_CameraRig_TouchpadWalking has a collection of walls and slopes that can be traversed by the user with the touchpad. There is also an area that can only be traversed if the user is crouching.

To enable the Rotate Object Control Action, ensure one of the `TouchpadControlOptions` children (located under the Controller script alias) has the `Rotate` control script active.

Snap Rotate Object Control Action (VRTK_SnapRotateObjectControlAction)

extends VRTK_BaseObjectControlAction

Overview

The Snap Rotate Object Control Action script is used to snap rotate the controlled GameObject around the up vector when changing the axis.

The effect is a immediate snap rotation to quickly face in a new direction.

Inspector Parameters

- **Angle Per Snap:** The angle to rotate for each snap.
- **Angle Multiplier:** The snap angle multiplier to be applied when the modifier button is pressed.
- **Snap Delay:** The amount of time required to pass before another snap rotation can be carried out.
- **Blink Transition Speed:** The speed for the headset to fade out and back in. Having a blink between rotations can reduce nausea.
- **Axis Threshold:** The threshold the listened axis needs to exceed before the action occurs. This can be used to limit the snap rotate to a single axis direction (e.g. pull down to flip rotate). The threshold is ignored if it is 0.

Example

`VRTK/Examples/017_CameraRig_TouchpadWalking` has a collection of walls and slopes that can be traversed by the user with the touchpad. There is also an area that can only be traversed if the user is crouching.

To enable the Snap Rotate Object Control Action, ensure one of the `TouchpadControlOptions` children (located under the Controller script alias) has the `Snap Rotate` control script active.

Warp Object Control Action (`VRTK_WarpObjectControlAction`)

extends `VRTK_BaseObjectControlAction`

Overview

The Warp Object Control Action script is used to warp the controlled GameObject a given distance when changing the axis.

The effect is a immediate snap to a new position in the given direction.

Inspector Parameters

- **Warp Distance:** The distance to warp in the facing direction.
- **Warp Multiplier:** The multiplier to be applied to the warp when the modifier button is pressed.
- **Warp Delay:** The amount of time required to pass before another warp can be carried out.
- **Floor Height Tolerance:** The height different in floor allowed to be a valid warp.
- **Blink Transition Speed:** The speed for the headset to fade out and back in. Having a blink between warps can reduce nausea.

- **Body Physics:** An optional Body Physics script to check for potential collisions in the moving direction. If any potential collision is found then the move will not take place. This can help reduce collision tunnelling.

Example

`VRTK/Examples/017_CameraRig_TouchpadWalking` has a collection of walls and slopes that can be traversed by the user with the touchpad. There is also an area that can only be traversed if the user is crouching.

To enable the Warp Object Control Action, ensure one of the `TouchpadControlOptions` children (located under the Controller script alias) has the `Warp` control script active.

Interactions (VRTK/Scripts/Interactions)

A collection of scripts that provide the ability to interact with game objects with the controllers.

- [Controller Events](#)
 - [Controller Highlighter](#)
 - [Controller Haptics](#)
 - [Interact Controller Appearance](#)
 - [Interact Touch](#)
 - [Interact Grab](#)
 - [Interact Use](#)
 - [Interactable Object](#)
 - [Object Appearance](#)
 - [Interact Haptics](#)
 - [Object Auto Grab](#)
 - [Object Touch Auto Interact](#)
-

Controller Events (VRTK_ControllerEvents)

Overview

The Controller Events script deals with events that the game controller is sending out.

The Controller Events script requires the Controller Mapper script on the same `GameObject` and provides event listeners for every button press on the controller (excluding the System Menu button as this cannot be overridden and is always used by Steam).

When a controller button is pressed, the script emits an event to denote that the button has been pressed which allows other scripts to listen for this event without needing to implement any

controller logic. When a controller button is released, the script also emits an event denoting that the button has been released.

The script also has a public boolean pressed state for the buttons to allow the script to be queried by other scripts to check if a button is being held down.

Inspector Parameters

- **Axis Fidelity:** The amount of fidelity in the changes on the axis, which is defaulted to 1. Any number higher than 2 will probably give too sensitive results.
- **Trigger Click Threshold:** The level on the trigger axis to reach before a click is registered.
- **Trigger Force Zero Threshold:** The level on the trigger axis to reach before the axis is forced to 0f.
- **Trigger Axis Zero On Untouch:** If this is checked then the trigger axis will be forced to 0f when the trigger button reports an untouch event.
- **Grip Click Threshold:** The level on the grip axis to reach before a click is registered.
- **Grip Force Zero Threshold:** The level on the grip axis to reach before the axis is forced to 0f.
- **Grip Axis Zero On Untouch:** If this is checked then the grip axis will be forced to 0f when the grip button reports an untouch event.

Class Variables

- `public enum ButtonAlias` - Button types
 - `Undefined` - No button specified
 - `TriggerHairline` - The trigger is squeezed past the current hairline threshold.
 - `TriggerTouch` - The trigger is squeezed a small amount.
 - `TriggerPress` - The trigger is squeezed about half way in.
 - `TriggerClick` - The trigger is squeezed all the way down.
 - `GripHairline` - The grip is squeezed past the current hairline threshold.
 - `GripTouch` - The grip button is touched.
 - `GripPress` - The grip button is pressed.
 - `GripClick` - The grip button is pressed all the way down.
 - `TouchpadTouch` - The touchpad is touched (without pressing down to click).
 - `TouchpadPress` - The touchpad is pressed (to the point of hearing a click).
 - `ButtonOneTouch` - The button one is touched.
 - `ButtonOnePress` - The button one is pressed.
 - `ButtonTwoTouch` - The button one is touched.
 - `ButtonTwoPress` - The button one is pressed.
 - `StartMenuPress` - The button one is pressed.
- `public bool triggerPressed` - This will be true if the trigger is squeezed about half way in.
Default: `false`
- `public bool triggerTouched` - This will be true if the trigger is squeezed a small amount. Default: `false`
- `public bool triggerHairlinePressed` - This will be true if the trigger is squeezed a small amount more from any previous squeeze on the trigger. Default: `false`
- `public bool triggerClicked` - This will be true if the trigger is squeezed all the way down.

Default: false

- `public bool triggerAxisChanged` - This will be true if the trigger has been squeezed more or less.

Default: false

- `public bool gripPressed` - This will be true if the grip is squeezed about half way in. Default: false
- `public bool gripTouched` - This will be true if the grip is touched. Default: false
- `public bool gripHairlinePressed` - This will be true if the grip is squeezed a small amount more from any previous squeeze on the grip. Default: false
- `public bool gripClicked` - This will be true if the grip is squeezed all the way down. Default: false
- `public bool gripAxisChanged` - This will be true if the grip has been squeezed more or less. Default: false
- `public bool touchpadPressed` - This will be true if the touchpad is held down. Default: false
- `public bool touchpadTouched` - This will be true if the touchpad is being touched. Default: false
- `public bool touchpadAxisChanged` - This will be true if the touchpad touch position has changed. Default: false
- `public bool buttonOnePressed` - This will be true if button one is held down. Default: false
- `public bool buttonOneTouched` - This will be true if button one is being touched. Default: false
- `public bool buttonTwoPressed` - This will be true if button two is held down. Default: false
- `public bool buttonTwoTouched` - This will be true if button two is being touched. Default: false
- `public bool startMenuPressed` - This will be true if start menu is held down. Default: false
- `public bool pointerPressed` - This will be true if the button aliased to the pointer is held down. Default: false
- `public bool grabPressed` - This will be true if the button aliased to the grab is held down. Default: false
- `public bool usePressed` - This will be true if the button aliased to the use is held down. Default: false
- `public bool uiClickPressed` - This will be true if the button aliased to the UI click is held down. Default: false
- `public bool menuPressed` - This will be true if the button aliased to the menu is held down. Default: false
- `public bool controllerVisible` - This will be true if the controller model alias renderers are visible. Default: true

Class Events

- `TriggerPressed` - Emitted when the trigger is squeezed about half way in.
- `TriggerReleased` - Emitted when the trigger is released under half way.
- `TriggerTouchStart` - Emitted when the trigger is squeezed a small amount.
- `TriggerTouchEnd` - Emitted when the trigger is no longer being squeezed at all.
- `TriggerHairlineStart` - Emitted when the trigger is squeezed past the current hairline threshold.
- `TriggerHairlineEnd` - Emitted when the trigger is released past the current hairline threshold.
- `TriggerClicked` - Emitted when the trigger is squeezed all the way down.
- `TriggerUnclicked` - Emitted when the trigger is no longer being held all the way down.

- `TriggerAxisChanged` - Emitted when the amount of squeeze on the trigger changes.
- `GripPressed` - Emitted when the grip is squeezed about half way in.
- `GripReleased` - Emitted when the grip is released under half way.
- `GripTouchStart` - Emitted when the grip is squeezed a small amount.
- `GripTouchEnd` - Emitted when the grip is no longer being squeezed at all.
- `GripHairlineStart` - Emitted when the grip is squeezed past the current hairline threshold.
- `GripHairlineEnd` - Emitted when the grip is released past the current hairline threshold.
- `GripClicked` - Emitted when the grip is squeezed all the way down.
- `GripUnclicked` - Emitted when the grip is no longer being held all the way down.
- `GripAxisChanged` - Emitted when the amount of squeeze on the grip changes.
- `TouchpadPressed` - Emitted when the touchpad is pressed (to the point of hearing a click).
- `TouchpadReleased` - Emitted when the touchpad has been released after a pressed state.
- `TouchpadTouchStart` - Emitted when the touchpad is touched (without pressing down to click).
- `TouchpadTouchEnd` - Emitted when the touchpad is no longer being touched.
- `TouchpadAxisChanged` - Emitted when the touchpad is being touched in a different location.
- `ButtonOneTouchStart` - Emitted when button one is touched.
- `ButtonOneTouchEnd` - Emitted when button one is no longer being touched.
- `ButtonOnePressed` - Emitted when button one is pressed.
- `ButtonOneReleased` - Emitted when button one is released.
- `ButtonTwoTouchStart` - Emitted when button two is touched.
- `ButtonTwoTouchEnd` - Emitted when button two is no longer being touched.
- `ButtonTwoPressed` - Emitted when button two is pressed.
- `ButtonTwoReleased` - Emitted when button two is released.
- `StartMenuPressed` - Emitted when start menu is pressed.
- `StartMenuReleased` - Emitted when start menu is released.
- `AliasPointerOn` - Emitted when the pointer toggle alias button is pressed.
- `AliasPointerOff` - Emitted when the pointer toggle alias button is released.
- `AliasPointerSet` - Emitted when the pointer set alias button is released.
- `AliasGrabOn` - Emitted when the grab toggle alias button is pressed.
- `AliasGrabOff` - Emitted when the grab toggle alias button is released.
- `AliasUseOn` - Emitted when the use toggle alias button is pressed.
- `AliasUseOff` - Emitted when the use toggle alias button is released.
- `AliasMenuOn` - Emitted when the menu toggle alias button is pressed.
- `AliasMenuOff` - Emitted when the menu toggle alias button is released.
- `AliasUIClickOn` - Emitted when the UI click alias button is pressed.
- `AliasUIClickOff` - Emitted when the UI click alias button is released.
- `ControllerEnabled` - Emitted when the controller is enabled.
- `ControllerDisabled` - Emitted when the controller is disabled.
- `ControllerIndexChanged` - Emitted when the controller index changed.
- `ControllerVisible` - Emitted when the controller is set to visible.
- `ControllerHidden` - Emitted when the controller is set to hidden.

Unity Events

Adding the `VRTK_ControllerEvents_UnityEvents` component to `VRTK_ControllerEvents` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_ControllerReference controllerReference` - The reference for the controller that was used.
- `float buttonPressure` - The amount of pressure being applied to the button pressed. `0f` to `1f`.
- `Vector2 touchpadAxis` - The position the touchpad is touched at. `(0,0)` to `(1,1)`.
- `float touchpadAngle` - The rotational position the touchpad is being touched at, `0` being top, `180` being bottom and all other angles accordingly. `0f` to `360f`.

Class Methods

SetControllerEvent/0

```
public virtual ControllerInteractionEventArgs SetControllerEvent()
```

- Parameters
 - *none*
- Returns
 - `ControllerInteractionEventArgs` - The payload for a Controller Event.

The `SetControllerEvent/0` method is used to set the Controller Event payload.

SetControllerEvent/3

```
public virtual ControllerInteractionEventArgs SetControllerEvent(ref bool  
buttonBool, bool value = false, float buttonPressure = 0f)
```

- Parameters
 - `ref bool buttonBool` - The state of the pressed button if required.
 - `bool value` - The value to set the `buttonBool` reference to.
 - `float buttonPressure` - The pressure of the button pressed if required.
- Returns
 - `ControllerInteractionEventArgs` - The payload for a Controller Event.

The `SetControllerEvent/3` method is used to set the Controller Event payload.

GetTouchpadAxis/0

```
public virtual Vector2 GetTouchpadAxis()
```

- Parameters
 - *none*
- Returns
 - `Vector2` - A 2 dimensional vector containing the x and y position of where the touchpad is being touched. (0,0) to (1,1).

The `GetTouchpadAxis` method returns the coordinates of where the touchpad is being touched and can be used for directional input via the touchpad. The `x` value is the horizontal touch plane and the `y` value is the vertical touch plane.

GetTouchpadAxisAngle/0

```
public virtual float GetTouchpadAxisAngle()
```

- Parameters
 - *none*
- Returns
 - `float` - A float representing the angle of where the touchpad is being touched. 0f to 360f.

The `GetTouchpadAxisAngle` method returns the angle of where the touchpad is currently being touched with the top of the touchpad being 0 degrees and the bottom of the touchpad being 180 degrees.

GetTriggerAxis/0

```
public virtual float GetTriggerAxis()
```

- Parameters
 - *none*
- Returns
 - `float` - A float representing the amount of squeeze that is being applied to the trigger. 0f to 1f.

The `GetTriggerAxis` method returns a float that represents how much the trigger is being squeezed. This can be useful for using the trigger axis to perform high fidelity tasks or only activating the trigger press once it has exceeded a given press threshold.

GetGripAxis/0

```
public virtual float GetGripAxis()
```

- Parameters
 - *none*
- Returns

- `float` - A float representing the amount of squeeze that is being applied to the grip. 0f to 1f.

The `GetGripAxis` method returns a float that represents how much the grip is being squeezed. This can be useful for using the grip axis to perform high fidelity tasks or only activating the grip press once it has exceeded a given press threshold.

GetHairTriggerDelta/0

```
public virtual float GetHairTriggerDelta()
```

- Parameters

- *none*

- Returns

- `float` - A float representing the difference in the trigger pressure from the hairline threshold start to current position.

The `GetHairTriggerDelta` method returns a float representing the difference in how much the trigger is being pressed in relation to the hairline threshold start.

GetHairGripDelta/0

```
public virtual float GetHairGripDelta()
```

- Parameters

- *none*

- Returns

- `float` - A float representing the difference in the trigger pressure from the hairline threshold start to current position.

The `GetHairTriggerDelta` method returns a float representing the difference in how much the trigger is being pressed in relation to the hairline threshold start.

AnyButtonPressed/0

```
public virtual bool AnyButtonPressed()
```

- Parameters

- *none*

- Returns

- `bool` - Is true if any of the controller buttons are currently being pressed.

The `AnyButtonPressed` method returns true if any of the controller buttons are being pressed and this can be useful to determine if an action can be taken whilst the user is using the controller.

IsButtonPressed/1

```
public virtual bool IsButtonPressed(ButtonAlias button)
```

- Parameters

- `ButtonAlias button` - The button to check if it's being pressed.

- Returns

- `bool` - Is true if the button is being pressed.

The `IsButtonPressed` method takes a given button alias and returns a boolean whether that given button is currently being pressed or not.

SubscribeToButtonAliasEvent/3

```
public virtual void SubscribeToButtonAliasEvent(ButtonAlias givenButton,  
bool startEvent, ControllerInteractionEventHandler callbackMethod)
```

- Parameters

- `ButtonAlias givenButton` - The `ButtonAlias` to register the event on.
- `bool startEvent` - If this is `true` then the start event related to the button is used (e.g. `OnPress`). If this is `false` then the end event related to the button is used (e.g. `OnRelease`).
- `ControllerInteractionEventHandler callbackMethod` - The method to subscribe to the event.

- Returns

- *none*

The `SubscribeToButtonAliasEvent` method makes it easier to subscribe to a button event on either the start or end action. Upon the event firing, the given callback method is executed.

UnsubscribeToButtonAliasEvent/3

```
public virtual void UnsubscribeToButtonAliasEvent(ButtonAlias givenButton,  
bool startEvent, ControllerInteractionEventHandler callbackMethod)
```

- Parameters

- `ButtonAlias givenButton` - The `ButtonAlias` to unregister the event on.
- `bool startEvent` - If this is `true` then the start event related to the button is used (e.g. `OnPress`). If this is `false` then the end event related to the button is used (e.g. `OnRelease`).
- `ControllerInteractionEventHandler callbackMethod` - The method to unsubscribe from the event.

- Returns

- *none*

The `UnsubscribeToButtonAliasEvent` method makes it easier to unsubscribe to from button event on either the start or end action.

Example

`VRTK/Examples/002_Controller_Events` shows how the events are utilised and listened to. The accompanying example script can be viewed in

`VRTK/Examples/Resources/Scripts/VRTK_ControllerEvents_ListenerExample.cs`.

Controller Highlighter (VRTK_ControllerHighlighter)

Overview

The Controller Highlighter script provides methods to deal with highlighting controller elements.

The highlighting of the controller is defaulted to use the `VRTK_MaterialColorSwapHighlighter` if no other highlighter is applied to the Object.

Inspector Parameters

- **Transition Duration:** The amount of time to take to transition to the set highlight colour.
- **Highlight Controller:** The colour to set the entire controller highlight colour to.
- **Highlight Body:** The colour to set the body highlight colour to.
- **Highlight Trigger:** The colour to set the trigger highlight colour to.
- **Highlight Grip:** The colour to set the grip highlight colour to.
- **Highlight Touchpad:** The colour to set the touchpad highlight colour to.
- **Highlight Button One:** The colour to set the button one highlight colour to.
- **Highlight Button Two:** The colour to set the button two highlight colour to.
- **Highlight System Menu:** The colour to set the system menu highlight colour to.
- **Highlight Start Menu:** The colour to set the start menu highlight colour to.
- **Model Element Paths:** A collection of strings that determine the path to the controller model sub elements for identifying the model parts at runtime. If the paths are left empty they will default to the model element paths of the selected SDK Bridge.
- **Element Highlighter Overrides:** A collection of highlighter overrides for each controller model sub element. If no highlighter override is given then highlighter on the Controller game object is used.
- **Controller Alias:** An optional GameObject to specify which controller to apply the script methods to. If this is left blank then this script is required to be placed on a Controller Alias GameObject.
- **Model Container:** An optional GameObject to specify where the controller models are. If this is left blank then the Model Alias object will be used.

Class Methods

`ConfigureControllerPaths/0`


```
public virtual void ConfigureControllerPaths()
```

- Parameters
 - *none*
- Returns
 - *none*

The ConfigureControllerPaths method is used to set up the model element paths.

PopulateHighlighters/0

```
public virtual void PopulateHighlighters()
```

- Parameters
 - *none*
- Returns
 - *none*

The PopulateHighlighters method sets up the highlighters on the controller model.

HighlightController/2

```
public virtual void HighlightController(Color color, float fadeDuration = 0f)
```

- Parameters
 - `Color color` - The colour to highlight the controller to.
 - `float fadeDuration` - The duration in time to fade from the initial colour to the target colour.
- Returns
 - *none*

The HighlightController method attempts to highlight all sub models of the controller.

UnhighlightController/0

```
public virtual void UnhighlightController()
```

- Parameters
 - *none*
- Returns
 - *none*

The UnhighlightController method attempts to remove the highlight from all sub models of the controller.

HighlightElement/3

```
public virtual void HighlightElement(SDK_BaseController.ControllerElements
elementType, Color color, float fadeDuration = 0f)
```

- Parameters

- `SDK_BaseController.ControllerElements elementType` - The element type on the controller.
- `Color color` - The colour to highlight the controller element to.
- `float fadeDuration` - The duration in time to fade from the initial colour to the target colour.

- Returns

- *none*

The HighlightElement method attempts to highlight a specific controller element.

UnhighlightElement/1

```
public virtual void UnhighlightElement(SDK_BaseController.ControllerElements
elementType)
```

- Parameters

- `SDK_BaseController.ControllerElements elementType` - The element type on the controller.

- Returns

- *none*

The UnhighlightElement method attempts to remove the highlight from the specific controller element.

Example

`VRTK/Examples/035_Controller_OpacityAndHighlighting` demonstrates the ability to change the opacity of a controller model and to highlight specific elements of a controller such as the buttons or even the entire controller model.

Controller Haptics (VRTK_ControllerHaptics)

Overview

The Controller Haptics scripts are a collection of static methods for calling haptic functions on a given controller.

Class Methods

TriggerHapticPulse/2

```
public static void TriggerHapticPulse(VRTK_ControllerReference  
controllerReference, float strength)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to activate the haptic feedback on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.

- Returns

- *none*

The `TriggerHapticPulse/2` method calls a single haptic pulse call on the controller for a single tick.

TriggerHapticPulse/4

```
public static void TriggerHapticPulse(VRTK_ControllerReference  
controllerReference, float strength, float duration, float pulseInterval)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to activate the haptic feedback on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.
- `float duration` - The length of time the rumble should continue for.
- `float pulseInterval` - The interval to wait between each haptic pulse.

- Returns

- *none*

The `TriggerHapticPulse/4` method calls a haptic pulse for a specified amount of time rather than just a single tick. Each pulse can be separated by providing a `pulseInterval` to pause between each haptic pulse.

TriggerHapticPulse/2

```
public static void TriggerHapticPulse(VRTK_ControllerReference  
controllerReference, AudioClip clip)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to activate the haptic feedback on.
- `AudioClip clip` - The audio clip to use for the haptic pattern.

- Returns

- *none*

The `TriggerHapticPulse/2` method calls a haptic pulse based on a given audio clip.

CancelHapticPulse/1

```
public static void CancelHapticPulse(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to cancel the haptic feedback on.

- Returns

- *none*

The `CancelHapticPulse` method cancels the existing running haptic pulse on the given controller index.

Interact Controller Appearance (VRTK_InteractControllerAppearance)

Overview

The `Interact Controller Appearance` script is attached on the same `GameObject` as an `Interactable Object` script and is used to determine whether the controller model should be visible or hidden on touch, grab or use.

Inspector Parameters

- **Hide Controller On Touch:** Hides the controller model when a valid touch occurs.
- **Hide Delay On Touch:** The amount of seconds to wait before hiding the controller on touch.
- **Hide Controller On Grab:** Hides the controller model when a valid grab occurs.
- **Hide Delay On Grab:** The amount of seconds to wait before hiding the controller on grab.
- **Hide Controller On Use:** Hides the controller model when a valid use occurs.
- **Hide Delay On Use:** The amount of seconds to wait before hiding the controller on use.

Class Events

- `ControllerHidden` - Emitted when the interacting object is hidden.
- `ControllerVisible` - Emitted when the interacting object is shown.
- `HiddenOnTouch` - Emitted when the interacting object is hidden on touch.
- `VisibleOnTouch` - Emitted when the interacting object is shown on untouch.
- `HiddenOnGrab` - Emitted when the interacting object is hidden on grab.

- `VisibleOnGrab` - Emitted when the interacting object is shown on ungrab.
- `HiddenOnUse` - Emitted when the interacting object is hidden on use.
- `VisibleOnUse` - Emitted when the interacting object is shown on unuse.

Unity Events

Adding the `VRTK_InteractControllerAppearance_UnityEvents` component to `VRTK_InteractControllerAppearance` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `GameObject interactingObject` - The object that is interacting.
- `GameObject ignoredObject` - The object that is being ignored.

Class Methods

ToggleControllerOnTouch/3

```
public virtual void ToggleControllerOnTouch(bool showController, GameObject touchingObject, GameObject ignoredObject)
```

- Parameters
 - `bool showController` - If true then the controller will attempt to be made visible when no longer touching, if false then the controller will be hidden on touch.
 - `GameObject touchingObject` - The touching object to apply the visibility state to.
 - `GameObject ignoredObject` - The object that is currently being interacted with by the touching object which is passed through to the visibility to prevent the object from being hidden as well.
- Returns
 - *none*

The `ToggleControllerOnTouch` method determines whether the controller should be shown or hidden when touching an interactable object.

ToggleControllerOnGrab/3

```
public virtual void ToggleControllerOnGrab(bool showController, GameObject grabbingObject, GameObject ignoredObject)
```

- Parameters
 - `bool showController` - If true then the controller will attempt to be made visible when no longer grabbing, if false then the controller will be hidden on grab.
 - `GameObject grabbingObject` - The grabbing object to apply the visibility state to.

- `GameObject ignoredObject` - The object that is currently being interacted with by the grabbing object which is passed through to the visibility to prevent the object from being hidden as well.
- Returns
 - *none*

The `ToggleControllerOnGrab` method determines whether the controller should be shown or hidden when grabbing an interactable object.

ToggleControllerOnUse/3

```
public virtual void ToggleControllerOnUse(bool showController, GameObject usingObject, GameObject ignoredObject)
```

- Parameters
 - `bool showController` - If true then the controller will attempt to be made visible when no longer using, if false then the controller will be hidden on use.
 - `GameObject usingObject` - The using object to apply the visibility state to.
 - `GameObject ignoredObject` - The object that is currently being interacted with by the using object which is passed through to the visibility to prevent the object from being hidden as well.
- Returns
 - *none*

The `ToggleControllerOnUse` method determines whether the controller should be shown or hidden when using an interactable object.

Example

`VRTK/Examples/008_Controller_UsingAGrabbedObject` shows that the controller can be hidden when touching, grabbing and using an object.

Interact Touch (VRTK_InteractTouch)

Overview

The Interact Touch script is usually applied to a Controller and provides a collider to know when the controller is touching something.

Colliders are created for the controller and by default the selected controller SDK will have a set of colliders for the given default controller of that SDK.

A custom collider can be provided by the Custom Rigidbody Object parameter.

Inspector Parameters

- **Custom Collider Container:** An optional `GameObject` that contains the compound colliders to represent the touching object. If this is empty then the collider will be auto generated at runtime to match the SDK default controller.

Class Events

- `ControllerStartTouchInteractableObject` - Emitted when the touch of a valid object has started.
- `ControllerTouchInteractableObject` - Emitted when a valid object is touched.
- `ControllerStartUntouchInteractableObject` - Emitted when the untouch of a valid object has started.
- `ControllerUntouchInteractableObject` - Emitted when a valid object is no longer being touched.
- `ControllerRigidbodyActivated` - Emitted when the controller rigidbody is activated.
- `ControllerRigidbodyDeactivated` - Emitted when the controller rigidbody is deactivated.

Unity Events

Adding the `VRTK_InteractTouch_UnityEvents` component to `VRTK_InteractTouch` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_ControllerReference controllerReference` - The reference to the controller doing the interaction.
- `GameObject target` - The `GameObject` of the interactable object that is being interacted with by the controller.

Class Methods

ForceTouch/1

```
public virtual void ForceTouch(GameObject obj)
```

- Parameters
 - `GameObject obj` - The game object to attempt to force touch.
- Returns
 - *none*

The `ForceTouch` method will attempt to force the controller to touch the given game object. This is useful if an object that isn't being touched is required to be grabbed or used as the controller doesn't physically have to be touching it to be forced to interact with it.

GetTouchedObject/0

```
public virtual GameObject GetTouchedObject()
```

- Parameters

- *none*

- Returns

- `GameObject` - The game object of what is currently being touched by this controller.

The `GetTouchedObject` method returns the current object being touched by the controller.

IsObjectInteractable/1

```
public virtual bool IsObjectInteractable(GameObject obj)
```

- Parameters

- `GameObject obj` - The game object to check to see if it's interactable.

- Returns

- `bool` - Is true if the given object is of type `VRTK_InteractableObject`.

The `IsObjectInteractable` method is used to check if a given game object is of type `VRTK_InteractableObject` and whether the object is enabled.

ToggleControllerRigidBody/2

```
public virtual void ToggleControllerRigidBody(bool state, bool forceToggle = false)
```

- Parameters

- `bool state` - The state of whether the rigidbody is on or off. `true` toggles the rigidbody on and `false` turns it off.
- `bool forceToggle` - Determines if the rigidbody has been forced into it's new state by another script. This can be used to override other non-force settings. Defaults to `false`

- Returns

- *none*

The `ToggleControllerRigidBody` method toggles the controller's rigidbody's ability to detect collisions. If it is true then the controller rigidbody will collide with other collidable game objects.

IsRigidBodyActive/0

```
public virtual bool IsRigidBodyActive()
```

- Parameters

- *none*
- Returns
 - `bool` - Is true if the rigidbody on the controller is currently active and able to affect other scene rigidbodies.

The `IsRigidBodyActive` method checks to see if the rigidbody on the controller object is active and can affect other rigidbodies in the scene.

IsRigidBodyForcedActive/0

```
public virtual bool IsRigidBodyForcedActive()
```

- Parameters
 - *none*
- Returns
 - `bool` - Is true if the rigidbody is active and has been forced into the active state.

The `IsRigidBodyForcedActive` method checks to see if the rigidbody on the controller object has been forced into the active state.

ForceStopTouching/0

```
public virtual void ForceStopTouching()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ForceStopTouching` method will stop the controller from touching an object even if the controller is physically touching the object still.

ControllerColliders/0

```
public virtual Collider[] ControllerColliders()
```

- Parameters
 - *none*
- Returns
 - `Collider[]` - An array of colliders that are associated with the controller.

The `ControllerColliders` method retrieves all of the associated colliders on the controller.

Example

`VRTK/Examples/005_Controller/BasicObjectGrabbing` demonstrates the highlighting of objects that have the `VRTK_InteractableObject` script added to them to show the ability to highlight interactable objects when they are touched by the controllers.

Interact Grab (VRTK_InteractGrab)

Overview

The Interact Grab script is attached to a Controller object and requires the `VRTK_ControllerEvents` script to be attached as it uses this for listening to the controller button events for grabbing and releasing interactable game objects.

It listens for the `AliasGrabOn` and `AliasGrabOff` events to determine when an object should be grabbed and should be released.

The Controller object also requires the `VRTK_InteractTouch` script to be attached to it as this is used to determine when an interactable object is being touched. Only valid touched objects can be grabbed.

An object can be grabbed if the Controller touches a game object which contains the `VRTK_InteractableObject` script and has the flag `isGrabbable` set to `true`.

If a valid interactable object is grabbable then pressing the set `Grab` button on the Controller (default is `Grip`) will grab and snap the object to the controller and will not release it until the `Grab` button is released.

When the Controller `Grab` button is released, if the interactable game object is grabbable then it will be propelled in the direction and at the velocity the controller was at, which can simulate object throwing.

The interactable objects require a collider to activate the trigger and a rigidbody to pick them up and move them around the game world.

Inspector Parameters

- **Grab Button:** The button used to grab/release a touched object.
- **Grab Precognition:** An amount of time between when the grab button is pressed to when the controller is touching something to grab it. For example, if an object is falling at a fast rate, then it is very hard to press the grab button in time to catch the object due to human reaction times. A higher number here will mean the grab button can be pressed before the controller touches the object and when the collision takes place, if the grab button is still being held down then the grab action will be successful.
- **Throw Multiplier:** An amount to multiply the velocity of any objects being thrown. This can be

useful when scaling up the play area to simulate being able to throw items further.

- **Create Rigid Body When Not Touching:** If this is checked and the controller is not touching an Interactable Object when the grab button is pressed then a rigid body is added to the controller to allow the controller to push other rigid body objects around.
- **Controller Attach Point:** The rigidbody point on the controller model to snap the grabbed object to. If blank it will be set to the SDK default.
- **Controller Events:** The controller to listen for the events on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Interact Touch:** The Interact Touch to listen for touches on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.

Class Events

- `GrabButtonPressed` - Emitted when the grab button is pressed.
- `GrabButtonReleased` - Emitted when the grab button is released.
- `ControllerStartGrabInteractableObject` - Emitted when a grab of a valid object is started.
- `ControllerGrabInteractableObject` - Emitted when a valid object is grabbed.
- `ControllerStartUngrabInteractableObject` - Emitted when a ungrab of a valid object is started.
- `ControllerUngrabInteractableObject` - Emitted when a valid object is released from being grabbed.

Unity Events

Adding the `VRTK_InteractGrab_UnityEvents` component to `VRTK_InteractGrab` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

IsGrabButtonPressed/0

```
public virtual bool IsGrabButtonPressed()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the grab alias button is being held down.

The `IsGrabButtonPressed` method determines whether the current grab alias button is being pressed down.

ForceRelease/1

```
public virtual void ForceRelease(bool applyGrabbingObjectVelocity = false)
```

- Parameters

- `bool applyGrabbingObjectVelocity` - If this is true then upon releasing the object any velocity on the grabbing object will be applied to the object to essentially throw it. Defaults to `false`.

- Returns

- *none*

The `ForceRelease` method will force the controller to stop grabbing the currently grabbed object.

AttemptGrab/0

```
public virtual void AttemptGrab()
```

- Parameters

- *none*

- Returns

- *none*

The `AttemptGrab` method will attempt to grab the currently touched object without needing to press the grab button on the controller.

GetGrabbedObject/0

```
public virtual GameObject GetGrabbedObject()
```

- Parameters

- *none*

- Returns

- `GameObject` - The game object of what is currently being grabbed by this controller.

The `GetGrabbedObject` method returns the current object being grabbed by the controller.

Example

`VRTK/Examples/005_Controller/BasicObjectGrabbing` demonstrates the grabbing of interactable objects that have the `VRTK_InteractableObject` script attached to them. The objects can be picked up and thrown around.

`VRTK/Examples/013_Controller_UsingAndGrabbingMultipleObjects` demonstrates that each controller can grab and use objects independently and objects can also be toggled to their use state simultaneously.

`VRTK/Examples/014_Controller_SnappingObjectsOnGrab` demonstrates the different mechanisms

for snapping a grabbed object to the controller.

Interact Use (VRTK_InteractUse)

Overview

The Interact Use script is attached to a Controller object and requires the `VRTK_ControllerEvents` script to be attached as it uses this for listening to the controller button events for using and stop using interactable game objects.

It listens for the `AliasUseOn` and `AliasUseOff` events to determine when an object should be used and should stop using.

The Controller object also requires the `VRTK_InteractTouch` script to be attached to it as this is used to determine when an interactable object is being touched. Only valid touched objects can be used.

An object can be used if the Controller touches a game object which contains the `VRTK_InteractableObject` script and has the flag `isUsable` set to `true`.

If a valid interactable object is usable then pressing the set `Use` button on the Controller (default is `Trigger`) will call the `StartUsing` method on the touched interactable object.

Inspector Parameters

- **Use Button:** The button used to use/unuse a touched object.
- **Controller Events:** The controller to listen for the events on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Interact Touch:** The Interact Touch to listen for touches on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Interact Grab:** The Interact Grab to listen for grab actions on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.

Class Events

- `UseButtonPressed` - Emitted when the use toggle alias button is pressed.
- `UseButtonReleased` - Emitted when the use toggle alias button is released.
- `ControllerStartUseInteractableObject` - Emitted when a use of a valid object is started.
- `ControllerUseInteractableObject` - Emitted when a valid object starts being used.
- `ControllerStartUnuseInteractableObject` - Emitted when a unuse of a valid object is started.
- `ControllerUnuseInteractableObject` - Emitted when a valid object stops being used.

Unity Events

Adding the `VRTK_InteractUse_UnityEvents` component to `VRTK_InteractUse` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

IsUseButtonPressed/0

```
public virtual bool IsUseButtonPressed()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the use alias button is being held down.

The `IsUsebuttonPressed` method determines whether the current use alias button is being pressed down.

GetUsingObject/0

```
public virtual GameObject GetUsingObject()
```

- Parameters
 - *none*
- Returns
 - `GameObject` - The game object of what is currently being used by this controller.

The `GetUsingObject` method returns the current object being used by the controller.

ForceStopUsing/0

```
public virtual void ForceStopUsing()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ForceStopUsing` method will force the controller to stop using the currently touched object and will also stop the object's using action.

ForceResetUsing/0

```
public virtual void ForceResetUsing()
```

- Parameters
 - *none*
- Returns
 - *none*

The ForceResetUsing will force the controller to stop using the currently touched object but the object will continue with it's existing using action.

AttemptUse/0

```
public virtual void AttemptUse()
```

- Parameters
 - *none*
- Returns
 - *none*

The AttemptUse method will attempt to use the currently touched object without needing to press the use button on the controller.

Example

VRTK/Examples/006_Controller_UsingADoor simulates using a door object to open and close it. It also has a cube on the floor that can be grabbed to show how interactable objects can be usable or grabbable.

VRTK/Examples/008_Controller_UsingAGrabbedObject shows that objects can be grabbed with one button and used with another (e.g. firing a gun).

Interactable Object (VRTK_InteractableObject)

Overview

The Interactable Object script is attached to any game object that is required to be interacted with (e.g. via the controllers).

The basis of this script is to provide a simple mechanism for identifying objects in the game world that can be grabbed or used but it is expected that this script is the base to be inherited into a

script with richer functionality.

The highlighting of an Interactable Object is defaulted to use the

`VRTK_MaterialColorSwapHighlighter` if no other highlighter is applied to the Object.

Inspector Parameters

- **Disable When Idle:** If this is checked then the interactable object script will be disabled when the object is not being interacted with. This will eliminate the potential number of calls the interactable objects make each frame.
- **Touch Highlight Color:** The colour to highlight the object when it is touched. This colour will override any globally set colour (for instance on the `VRTK_InteractTouch` script).
- **Allowed Touch Controllers:** Determines which controller can initiate a touch action.
- **Ignored Colliders:** An array of colliders on the object to ignore when being touched.
- **Is Grabbable:** Determines if the object can be grabbed.
- **Hold Button To Grab:** If this is checked then the grab button on the controller needs to be continually held down to keep grabbing. If this is unchecked the grab button toggles the grab action with one button press to grab and another to release.
- **Stay Grabbed On Teleport:** If this is checked then the object will stay grabbed to the controller when a teleport occurs. If it is unchecked then the object will be released when a teleport occurs.
- **Valid Drop:** Determines in what situation the object can be dropped by the controller grab button.
- **Grab Override Button:** If this is set to `Undefined` then the global grab alias button will grab the object, setting it to any other button will ensure the override button is used to grab this specific interactable object.
- **Allowed Grab Controllers:** Determines which controller can initiate a grab action.
- **Grab Attach Mechanic Script:** This determines how the grabbed item will be attached to the controller when it is grabbed. If one isn't provided then the first Grab Attach script on the `GameObject` will be used, if one is not found and the object is grabbable then a Fixed Joint Grab Attach script will be created at runtime.
- **Secondary Grab Action Script:** The script to utilise when processing the secondary controller action on a secondary grab attempt. If one isn't provided then the first Secondary Controller Grab Action script on the `GameObject` will be used, if one is not found then no action will be taken on secondary grab.
- **Is Usable:** Determines if the object can be used.
- **Hold Button To Use:** If this is checked then the use button on the controller needs to be continually held down to keep using. If this is unchecked the the use button toggles the use action with one button press to start using and another to stop using.
- **Use Only If Grabbed:** If this is checked the object can be used only if it is currently being grabbed.
- **Pointer Activates Use Action:** If this is checked then when a Base Pointer beam (projected from the controller) hits the interactable object, if the object has `Hold Button To Use` unchecked then whilst the pointer is over the object it will run it's `Using` method. If `Hold Button To Use` is unchecked then the `Using` method will be run when the pointer is deactivated. The world pointer will not throw the `Destination Set` event if it is affecting an interactable object with this setting checked as this prevents unwanted teleporting from happening when using an object with a

pointer.

- **Use Override Button:** If this is set to `Undefined` then the global use alias button will use the object, setting it to any other button will ensure the override button is used to use this specific interactable object.
- **Allowed Use Controllers:** Determines which controller can initiate a use action.

Class Variables

- `public enum AllowedController` - Allowed controller type.
 - `Both` - Both controllers are allowed to interact.
 - `LeftOnly` - Only the left controller is allowed to interact.
 - `RightOnly` - Only the right controller is allowed to interact.
- `public enum ValidDropTypes` - The types of valid situations that the object can be released from grab.
 - `NoDrop` - The object cannot be dropped via the controller
 - `DropAnywhere` - The object can be dropped anywhere in the scene via the controller.
 - `DropValidSnapDropZone` - The object can only be dropped when it is hovering over a valid snap drop zone.
- `public int usingState` - The current using state of the object. 0 not being used, 1 being used.
Default: 0
- `public bool isKinematic` - `isKinematic` is a pass through to the `isKinematic` getter/setter on the object's rigidbody component.

Class Events

- `InteractableObjectTouched` - Emitted when another object touches the current object.
- `InteractableObjectUntouched` - Emitted when the other object stops touching the current object.
- `InteractableObjectGrabbed` - Emitted when another object grabs the current object (e.g. a controller).
- `InteractableObjectUngrabbed` - Emitted when the other object stops grabbing the current object.
- `InteractableObjectUsed` - Emitted when another object uses the current object (e.g. a controller).
- `InteractableObjectUnused` - Emitted when the other object stops using the current object.
- `InteractableObjectEnteredSnapDropZone` - Emitted when the object enters a snap drop zone.
- `InteractableObjectExitedSnapDropZone` - Emitted when the object exists a snap drop zone.
- `InteractableObjectSnappedToDropZone` - Emitted when the object gets snapped to a drop zone.
- `InteractableObjectUnsnappedFromDropZone` - Emitted when the object gets unsnapped from a drop zone.

Unity Events

Adding the `VRTK_InteractableObject_UnityEvents` component to `VRTK_InteractableObject` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `GameObject interactingObject` - The object that is initiating the interaction (e.g. a controller).

Class Methods

IsTouched/0

```
public virtual bool IsTouched()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns `true` if the object is currently being touched.

The `IsTouched` method is used to determine if the object is currently being touched.

IsGrabbed/1

```
public virtual bool IsGrabbed(GameObject grabbedBy = null)
```

- Parameters
 - `GameObject grabbedBy` - An optional `GameObject` to check if the Interactable Object is grabbed by that specific `GameObject`. Defaults to `null`
- Returns
 - `bool` - Returns `true` if the object is currently being grabbed.

The `IsGrabbed` method is used to determine if the object is currently being grabbed.

IsUsing/1

```
public virtual bool IsUsing(GameObject usedBy = null)
```

- Parameters
 - `GameObject usedBy` - An optional `GameObject` to check if the Interactable Object is used by that specific `GameObject`. Defaults to `null`
- Returns
 - `bool` - Returns `true` if the object is currently being used.

The `IsUsing` method is used to determine if the object is currently being used.

StartTouching/1

```
public virtual void StartTouching(VRTK_InteractTouch currentTouchingObject = null)
```

- Parameters

- `VRTK_InteractTouch currentTouchingObject` - The object that is currently touching this object.

- Returns

- *none*

The `StartTouching` method is called automatically when the object is touched initially. It is also a virtual method to allow for overriding in inherited classes.

StopTouching/1

```
public virtual void StopTouching(VRTK_InteractTouch previousTouchingObject = null)
```

- Parameters

- `VRTK_InteractTouch previousTouchingObject` - The object that was previously touching this object.

- Returns

- *none*

The `StopTouching` method is called automatically when the object has stopped being touched. It is also a virtual method to allow for overriding in inherited classes.

Grabbed/1

```
public virtual void Grabbed(VRTK_InteractGrab currentGrabbingObject = null)
```

- Parameters

- `VRTK_InteractGrab currentGrabbingObject` - The object that is currently grabbing this object.

- Returns

- *none*

The `Grabbed` method is called automatically when the object is grabbed initially. It is also a virtual method to allow for overriding in inherited classes.

Ungrabbed/1

```
public virtual void Ungrabbed(VRTK_InteractGrab previousGrabbingObject = null)
```

- Parameters

- `VRTK_InteractGrab previousGrabbingObject` - The object that was previously grabbing this object.

- Returns

- *none*

The Ungrabbed method is called automatically when the object has stopped being grabbed. It is also a virtual method to allow for overriding in inherited classes.

StartUsing/1

```
public virtual void StartUsing(VRTK_InteractUse currentUsingObject = null)
```

- Parameters

- `VRTK_InteractUse currentUsingObject` - The object that is currently using this object.

- Returns

- *none*

The StartUsing method is called automatically when the object is used initially. It is also a virtual method to allow for overriding in inherited classes.

StopUsing/1

```
public virtual void StopUsing(VRTK_InteractUse previousUsingObject = null)
```

- Parameters

- `VRTK_InteractUse previousUsingObject` - The object that was previously using this object.

- Returns

- *none*

The StopUsing method is called automatically when the object has stopped being used. It is also a virtual method to allow for overriding in inherited classes.

ToggleHighlight/1

```
public virtual void ToggleHighlight(bool toggle)
```

- Parameters

- `bool toggle` - The state to determine whether to activate or deactivate the highlight. `true` will enable the highlight and `false` will remove the highlight.

- Returns

- *none*

The ToggleHighlight method is used to turn on or off the colour highlight of the object.

ResetHighlighter/0

```
public virtual void ResetHighlighter()
```

- Parameters
 - *none*
- Returns
 - *none*

The ResetHighlighter method is used to reset the currently attached highlighter.

PauseCollisions/1

```
public virtual void PauseCollisions(float delay)
```

- Parameters
 - `float delay` - The amount of time to pause the collisions for.
- Returns
 - *none*

The PauseCollisions method temporarily pauses all collisions on the object at grab time by removing the object's rigidbody's ability to detect collisions. This can be useful for preventing clipping when initially grabbing an item.

ZeroVelocity/0

```
public virtual void ZeroVelocity()
```

- Parameters
 - *none*
- Returns
 - *none*

The ZeroVelocity method resets the velocity and angular velocity to zero on the rigidbody attached to the object.

SaveCurrentState/0

```
public virtual void SaveCurrentState()
```

- Parameters
 - *none*
- Returns
 - *none*

The `SaveCurrentState` method stores the existing object parent and the object's rigidbody kinematic setting.

GetTouchingObjects/0

```
public virtual List<GameObject> GetTouchingObjects()
```

- Parameters
 - *none*
- Returns
 - `List<GameObject>` - A list of game object of that are currently touching the current object.

The `GetTouchingObjects` method is used to return the collection of valid game objects that are currently touching this object.

GetGrabbingObject/0

```
public virtual GameObject GetGrabbingObject()
```

- Parameters
 - *none*
- Returns
 - `GameObject` - The game object of what is grabbing the current object.

The `GetGrabbingObject` method is used to return the game object that is currently grabbing this object.

GetSecondaryGrabbingObject/0

```
public virtual GameObject GetSecondaryGrabbingObject()
```

- Parameters
 - *none*
- Returns
 - `GameObject` - The game object of the secondary controller influencing the current grabbed object.

The `GetSecondaryGrabbingObject` method is used to return the game object that is currently being

used to influence this object whilst it is being grabbed by a secondary controller.

GetUsingObject/0

```
public virtual GameObject GetUsingObject()
```

- Parameters
 - *none*
- Returns
 - `GameObject` - The `GameObject` of what is using the current object.

The `GetUsingObject` method is used to return the `GameObject` that is currently using this object.

GetUsingScript/0

```
public virtual VRTK_InteractUse GetUsingScript()
```

- Parameters
 - *none*
- Returns
 - `VRTK_InteractUse` - The `InteractUse` script of the object that is using the current object.

The `GetUsingScript` method is used to return the `InteractUse` script that is currently using this object.

IsValidInteractableController/2

```
public virtual bool IsValidInteractableController(GameObject  
actualController, AllowedController controllerCheck)
```

- Parameters
 - `GameObject actualController` - The game object of the controller that is being checked.
 - `AllowedController controllerCheck` - The value of which controller is allowed to interact with this object.
- Returns
 - `bool` - Is true if the interacting controller is allowed to grab the object.

The `IsValidInteractableController` method is used to check to see if a controller is allowed to perform an interaction with this object as sometimes controllers are prohibited from grabbing or using an object depending on the use case.

ForceStopInteracting/0

```
public virtual void ForceStopInteracting()
```

- Parameters
 - *none*
- Returns
 - *none*

The ForceStopInteracting method forces the object to no longer be interacted with and will cause a controller to drop the object and stop touching it. This is useful if the controller is required to auto interact with another object.

ForceStopSecondaryGrabInteraction/0

```
public virtual void ForceStopSecondaryGrabInteraction()
```

- Parameters
 - *none*
- Returns
 - *none*

The ForceStopSecondaryGrabInteraction method forces the object to no longer be influenced by the second controller grabbing it.

RegisterTeleporters/0

```
public virtual void RegisterTeleporters()
```

- Parameters
 - *none*
- Returns
 - *none*

The RegisterTeleporters method is used to find all objects that have a teleporter script and register the object on the `OnTeleported` event. This is used internally by the object for keeping Tracked objects positions updated after teleporting.

UnregisterTeleporters/0

```
public virtual void UnregisterTeleporters()
```

- Parameters
 - *none*

- Returns
 - *none*

The UnregisterTeleporters method is used to unregister all teleporter events that are active on this object.

StoreLocalScale/0

```
public virtual void StoreLocalScale()
```

- Parameters
 - *none*
- Returns
 - *none*

the StoreLocalScale method saves the current transform local scale values.

ToggleSnapDropZone/2

```
public virtual void ToggleSnapDropZone(VRTK_SnapDropZone snapDropZone, bool state)
```

- Parameters
 - VRTK_SnapDropZone snapDropZone - The Snap Drop Zone object that is being interacted with.
 - bool state - The state of whether the interactable object is fixed in or removed from the Snap Drop Zone. True denotes the interactable object is fixed to the Snap Drop Zone and false denotes it has been removed from the Snap Drop Zone.
- Returns
 - *none*

The ToggleSnapDropZone method is used to set the state of whether the interactable object is in a Snap Drop Zone or not.

IsInSnapDropZone/0

```
public virtual bool IsInSnapDropZone()
```

- Parameters
 - *none*
- Returns
 - bool - Returns true if the interactable object is currently snapped in a drop zone and returns false if it is not.

The `IsInSnapDropZone` method determines whether the interactable object is currently snapped to a drop zone.

SetSnapDropZoneHover/2

```
public virtual void SetSnapDropZoneHover(VRTK_SnapDropZone snapDropZone,  
    bool state)
```

- Parameters

- `VRTK_SnapDropZone snapDropZone` - The Snap Drop Zone object that is being interacted with.
- `bool state` - The state of whether the object is being hovered or not.

- Returns

- *none*

The `SetSnapDropZoneHover` method sets whether the interactable object is currently being hovered over a valid Snap Drop Zone.

GetStoredSnapDropZone/0

```
public virtual VRTK_SnapDropZone GetStoredSnapDropZone()
```

- Parameters

- *none*

- Returns

- `VRTK_SnapDropZone` - The `SnapDropZone` that the interactable object is currently snapped to.

The `GetStoredSnapDropZone` method returns the snap drop zone that the interactable object is currently snapped to.

IsDroppable/0

```
public virtual bool IsDroppable()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the object can currently be dropped and returns false if it is not currently possible to drop.

The `IsDroppable` method returns whether the object can be dropped or not in it's current situation.

IsSwappable/0

```
public virtual bool IsSwappable()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the object can be grabbed by a secondary controller whilst already being grabbed and the object will swap controllers. Returns false if the object cannot be swapped.

The `IsSwappable` method returns whether the object can be grabbed with one controller and then swapped to another controller by grabbing with the secondary controller.

PerformSecondaryAction/0

```
public virtual bool PerformSecondaryAction()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the object has a secondary action, returns false if it has no secondary action or is swappable.

The `PerformSecondaryAction` method returns whether the object has a secondary action that can be performed when grabbing the object with a secondary controller.

ResetIgnoredColliders/0

```
public virtual void ResetIgnoredColliders()
```

- Parameters

- *none*

- Returns

- *none*

The `ResetIgnoredColliders` method is used to clear any stored ignored colliders in case the `IgnoredColliders` array parameter is changed at runtime. This needs to be called manually if changes are made at runtime.

Example

`VRTK/Examples/005_Controller_BasicObjectGrabbing` uses the `VRTK_InteractTouch` and `VRTK_InteractGrab` scripts on the controllers to show how an interactable object can be grabbed and snapped to the controller and thrown around the game world.

`VRTK/Examples/013_Controller_UsingAndGrabbingMultipleObjects` shows multiple objects that

can be grabbed by holding the buttons or grabbed by toggling the button click and also has objects that can have their Using state toggled to show how multiple items can be turned on at the same time.

Object Appearance (VRTK_ObjectAppearance)

Overview

The Object Appearance script provides a collection of static methods for controlling the appearance of a GameObject.

The GameObject can have its opacity changed, or its renderers toggled, or highlighters toggled.

Class Methods

SetOpacity/3

```
public static void SetOpacity(GameObject model, float alpha, float
transitionDuration = 0f)
```

- Parameters
 - `GameObject model` - The GameObject to change the renderer opacity on.
 - `float alpha` - The alpha level to apply to opacity of the controller object. 0f to 1f.
 - `float transitionDuration` - The time to transition from the current opacity to the new opacity.
- Returns
 - *none*

The SetOpacity method allows the opacity of the given GameObject to be changed. A lower alpha value will make the object more transparent, such as 0.5f will make the controller partially transparent where as 0f will make the controller completely transparent.

SetRendererVisible/2

```
public static void SetRendererVisible(GameObject model, GameObject
ignoredModel = null)
```

- Parameters
 - `GameObject model` - The GameObject to show the renderers for.
 - `GameObject ignoredModel` - An optional GameObject to ignore the renderer toggle on.
- Returns
 - *none*

The `SetRendererVisible` method turns on renderers of a given `GameObject`. It can also be provided with an optional model to ignore the render toggle on.

SetRendererHidden/2

```
public static void SetRendererHidden(GameObject model, GameObject
ignoredModel = null)
```

- Parameters

- `GameObject model` - The `GameObject` to hide the renderers for.
- `GameObject ignoredModel` - An optional `GameObject` to ignore the renderer toggle on.

- Returns

- *none*

The `SetRendererHidden` method turns off renderers of a given `GameObject`. It can also be provided with an optional model to ignore the render toggle on.

ToggleRenderer/3

```
public static void ToggleRenderer(bool state, GameObject model, GameObject
ignoredModel = null)
```

- Parameters

- `bool state` - If true then the renderers will be enabled, if false the renderers will be disabled.
- `GameObject model` - The `GameObject` to toggle the renderer states of.
- `GameObject ignoredModel` - An optional `GameObject` to ignore the renderer toggle on.

- Returns

- *none*

The `ToggleRenderer` method turns on or off the renderers of a given `GameObject`. It can also be provided with an optional model to ignore the render toggle of.

HighlightObject/3

```
public static void HighlightObject(GameObject model, Color? highlightColor,
float fadeDuration = 0f)
```

- Parameters

- `GameObject model` - The `GameObject` to attempt to call the Highlight on.
- `Color? highlightColor` - The colour to highlight to.
- `float fadeDuration` - The duration in time to fade from the initial colour to the target colour.

- Returns

- *none*

The HighlightObject method calls the Highlight method on the highlighter attached to the given GameObject with the provided colour.

UnhighlightObject/1

```
public static void UnhighlightObject(GameObject model)
```

- Parameters

- `GameObject model` - The GameObject to attempt to call the Unhighlight on.

- Returns

- *none*

The UnhighlightObject method calls the Unhighlight method on the highlighter attached to the given GameObject.

Interact Haptics (VRTK_InteractHaptics)

Overview

The Interact Haptics script is attached on the same GameObject as an Interactable Object script and provides controller haptics on touch, grab and use of the object.

Inspector Parameters

- **Clip On Touch:** Denotes the audio clip to use to rumble the controller on touch.
- **Strength On Touch:** Denotes how strong the rumble in the controller will be on touch.
- **Duration On Touch:** Denotes how long the rumble in the controller will last on touch.
- **Interval On Touch:** Denotes interval between rumbles in the controller on touch.
- **Clip On Grab:** Denotes the audio clip to use to rumble the controller on grab.
- **Strength On Grab:** Denotes how strong the rumble in the controller will be on grab.
- **Duration On Grab:** Denotes how long the rumble in the controller will last on grab.
- **Interval On Grab:** Denotes interval between rumbles in the controller on grab.
- **Clip On Use:** Denotes the audio clip to use to rumble the controller on use.
- **Strength On Use:** Denotes how strong the rumble in the controller will be on use.
- **Duration On Use:** Denotes how long the rumble in the controller will last on use.
- **Interval On Use:** Denotes interval between rumbles in the controller on use.

Class Events

- `InteractHapticsTouched` - Emitted when the haptics are from a touch.
- `InteractHapticsGrabbed` - Emitted when the haptics are from a grab.

- `InteractHapticsUsed` - Emitted when the haptics are from a use.

Unity Events

Adding the `VRTK_InteractHaptics_UnityEvents` component to `VRTK_InteractHaptics` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_ControllerReference controllerReference` - The reference to the controller to perform haptics on.

Class Methods

HapticsOnTouch/1

```
public virtual void HapticsOnTouch(VRTK_ControllerReference  
controllerReference)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference to the controller to activate the haptic feedback on.
- Returns
 - *none*

The `HapticsOnTouch` method triggers the haptic feedback on the given controller for the settings associated with touch.

HapticsOnGrab/1

```
public virtual void HapticsOnGrab(VRTK_ControllerReference  
controllerReference)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference to the controller to activate the haptic feedback on.
- Returns
 - *none*

The `HapticsOnGrab` method triggers the haptic feedback on the given controller for the settings associated with grab.

HapticsOnUse/1

```
public virtual void HapticsOnUse(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to activate the haptic feedback on.

- Returns

- *none*

The HapticsOnUse method triggers the haptic feedback on the given controller for the settings associated with use.

Object Auto Grab (VRTK_ObjectAutoGrab)

Overview

It is possible to automatically grab an Interactable Object to a specific controller by applying the Object Auto Grab script to the controller that the object should be grabbed by default.

Inspector Parameters

- **Object To Grab:** A game object (either within the scene or a prefab) that will be grabbed by the controller on game start.
- **Object Is Prefab:** If the `Object To Grab` is a prefab then this needs to be checked, if the `Object To Grab` already exists in the scene then this needs to be unchecked.
- **Clone Grabbed Object:** If this is checked then the `Object To Grab` will be cloned into a new object and attached to the controller leaving the existing object in the scene. This is required if the same object is to be grabbed to both controllers as a single object cannot be grabbed by different controllers at the same time. It is also required to clone a grabbed object if it is a prefab as it needs to exist within the scene to be grabbed.
- **Always Clone On Enable:** If `Clone Grabbed Object` is checked and this is checked, then whenever this script is disabled and re-enabled, it will always create a new clone of the object to grab. If this is false then the original cloned object will attempt to be grabbed again. If the original cloned object no longer exists then a new clone will be created.
- **Interact Touch:** The Interact Touch to listen for touches on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Interact Grab:** The Interact Grab to listen for grab actions on. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.

Class Events

- `ObjectAutoGrabCompleted` - Emitted when the object auto grab has completed successfully.

Unity Events

Adding the `VRTK_ObjectAutoGrab_UnityEvents` component to `VRTK_ObjectAutoGrab` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

ClearPreviousClone/0

```
public virtual void ClearPreviousClone()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ClearPreviousClone` method resets the previous cloned object to null to ensure when the script is re-enabled that a new cloned object is created, rather than the original clone being grabbed again.

Example

`VRTK/Examples/026_Controller_ForceHoldObject` shows how to automatically grab a sword to each controller and also prevents the swords from being dropped so they are permanently attached to the user's controllers.

Object Touch Auto Interact (`VRTK_ObjectTouchAutoInteract`)

Overview

The `Object Touch Auto Interact` script allows grab or use interactions on an object to automatically happen upon touching the interactable object.

Inspector Parameters

- **Grab On Touch When:** Determines when a grab on touch should occur.
- **Regrab Delay:** After being ungrabbed, another auto grab on touch can only occur after this time.
- **Continuous Grab Check:** If this is checked then the grab on touch check will happen every frame and not only on the first touch of the object.

- **Use On Touch When:** Determines when a use on touch should occur.
- **Reuse Delay:** After being unused, another auto use on touch can only occur after this time.
- **Continuous Use Check:** If this is checked then the use on touch check will happen every frame and not only on the first touch of the object.
- **Interactable Object:** The interactable object that the auto interaction will occur on. If this is blank then the script must be on the same GameObject as the Interactable Object script.

Class Variables

- `public enum AutoInteractions` - Situation when auto interaction can occur.
 - `Never` - Auto interaction can never occur on touch.
 - `NoButtonHeld` - Auto interaction will occur on touch even if the specified interaction button is not held down.
 - `ButtonHeld` - Auto interaction will only occur on touch if the specified interaction button is held down.
-

Highlighters (VRTK/Scripts/Interactions/Highlighters)

This directory contains scripts that are used to provide different object highlighting.

- [Base Highlighter](#)
 - [Material Colour Swap](#)
 - [Material Property Block Colour Swap](#)
 - [Outline Object Copy](#)
-

Base Highlighter (VRTK_BaseHighlighter)

Overview

The Base Highlighter is an abstract class that all other highlighters inherit and are required to implement the public methods.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Inspector Parameters

- **Active:** Determines if this highlighter is the active highlighter for the object the component is attached to. Only 1 active highlighter can be applied to a game object.
- **Unhighlight On Disable:** Determines if the highlighted object should be unhighlighted when it is disabled.

Class Methods

Initialise/2

```
public abstract void Initialise(Color? color = null, Dictionary<string, object> options = null);
```

- Parameters

- `Color? color` - An optional colour may be passed through at point of initialisation in case the highlighter requires it.
- `Dictionary<string, object> options` - An optional dictionary of highlighter specific options that may be differ with highlighter implementations.

- Returns

- *none*

The Initialise method is used to set up the state of the highlighter.

ResetHighlighter/0

```
public abstract void ResetHighlighter();
```

- Parameters

- *none*

- Returns

- *none*

The ResetHighlighter method is used to reset the highlighter if anything on the object has changed. It should be called by any scripts changing object materials or colours.

Highlight/2

```
public abstract void Highlight(Color? color = null, float duration = 0f);
```

- Parameters

- `Color? color` - An optional colour to highlight the game object to. The highlight colour may already have been set in the `Initialise` method so may not be required here.
- `float duration` - An optional duration of how long before the highlight has occurred. It can be used by highlighters to fade the colour if possible.

- Returns

- *none*

The Highlight method is used to initiate the highlighting logic to apply to an object.

Unhighlight/2

```
public abstract void Unhighlight(Color? color = null, float duration = 0f);
```

- Parameters

- `Color? color` - An optional colour that could be used during the unhighlight phase. Usually will be left as null.
- `float duration` - An optional duration of how long before the unhighlight has occurred.

- Returns

- *none*

The Unhighlight method is used to initiate the logic that returns an object back to it's original appearance.

GetOption/2

```
public virtual T GetOption<T>(Dictionary<string, object> options, string key)
```

- Type Params

- `T` - The system type that is expected to be returned.

- Parameters

- `Dictionary<string, object> options` - The dictionary of options to check in.
- `string key` - The identifier key to look for.

- Returns

- `T` - The value in the options at the given key returned in the provided system type.

The GetOption method is used to return a value from the options array if the given key exists.

UsesClonedObject/0

```
public virtual bool UsesClonedObject()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the highlighter creates a cloned object to apply the highlighter on, returns false if no additional object is created.

The UsesClonedObject method is used to return whether the current highlighter creates a cloned object to do the highlighting with.

GetActiveHighlighter/1

```
public static VRTK_BaseHighlighter GetActiveHighlighter(GameObject obj)
```

- Parameters
 - `GameObject obj` - The game object to check for a highlighter on.
- Returns
 - `VRTK_BaseHighlighter` - A valid and active highlighter.

The `GetActiveHighlighter` method checks the given game object for a valid and active highlighter.

Material Colour Swap (VRTK_MaterialColorSwapHighlighter)

extends `VRTK_BaseHighlighter`

Overview

The Material Colour Swap Highlighter is a basic implementation that simply swaps the texture colour for the given highlight colour.

Due to the way the object material is interacted with, changing the material colour will break Draw Call Batching in Unity whilst the object is highlighted.

The Draw Call Batching will resume on the original material when the item is no longer highlighted.

This is the default highlighter that is applied to any script that requires a highlighting component (e.g. `VRTK_Interactable_Object`).

Inspector Parameters

- **Emission Darken:** The emission colour of the texture will be the highlight colour but this percent darker.
- **Custom Material:** A custom material to use on the highlighted object.

Class Methods

Initialise/2

```
public override void Initialise(Color? color = null, Dictionary<string, object> options = null)
```

- Parameters
 - `Color? color` - Not used.
 - `Dictionary<string, object> options` - A dictionary array containing the highlighter options:

- `<'resetMainTexture', bool>` - Determines if the default main texture should be cleared on highlight. `true` to reset the main default texture, `false` to not reset it.

- Returns
 - *none*

The Initialise method sets up the highlighter for use.

ResetHighlighter/0

```
public override void ResetHighlighter()
```

- Parameters
 - *none*
- Returns
 - *none*

The ResetHighlighter method stores the object's materials and shared materials prior to highlighting.

Highlight/2

```
public override void Highlight(Color? color, float duration = 0f)
```

- Parameters
 - `Color? color` - The colour to highlight to.
 - `float duration` - The time taken to fade to the highlighted colour.
- Returns
 - *none*

The Highlight method initiates the change of colour on the object and will fade to that colour (from a base white colour) for the given duration.

Unhighlight/2

```
public override void Unhighlight(Color? color = null, float duration = 0f)
```

- Parameters
 - `Color? color` - Not used.
 - `float duration` - Not used.
- Returns
 - *none*

The Unhighlight method returns the object back to its original colour.

Example

`VRTK/Examples/005_Controller_BasicObjectGrabbing` demonstrates the solid highlighting on the green cube, red cube and flying saucer when the controller touches it.

`VRTK/Examples/035_Controller_OpacityAndHighlighting` demonstrates the solid highlighting if the right controller collides with the green box or if any of the buttons are pressed.

Material Property Block Colour Swap (VRTK_MaterialPropertyBlockColorSwapHighlighter)

extends `VRTK_MaterialColorSwapHighlighter`

Overview

This highlighter swaps the texture colour for the given highlight colour using `MaterialPropertyBlocks`. The effect of this highlighter is the same as of the `VRTK_MaterialColorSwapHighlighter.cs` but this highlighter can additionally handle objects which make use material instances.

Due to the way the object material is interacted with, changing the material colour will break Draw Call Batching in Unity whilst the object is highlighted.

The Draw Call Batching will resume on the original material when the item is no longer highlighted.

Class Methods

Initialise/2

```
public override void Initialise(Color? color = null, Dictionary<string, object> options = null)
```

- Parameters

- `Color? color` - Not used.
- `Dictionary<string, object> options` - A dictionary array containing the highlighter options:
 - `<'resetMainTexture', bool>` - Determines if the default main texture should be cleared on highlight. `true` to reset the main default texture, `false` to not reset it.

- Returns

- *none*

The `Initialise` method sets up the highlighter for use.

Unhighlight/2

```
public override void Unhighlight(Color? color = null, float duration = 0f)
```

- Parameters

- Color? color - Not used.
- float duration - Not used.

- Returns

- *none*

The Unhighlight method returns the object back to it's original colour.

Outline Object Copy (VRTK_OutlineObjectCopyHighlighter)

extends VRTK_BaseHighlighter

Overview

The Outline Object Copy Highlighter works by making a copy of a mesh and adding an outline shader to it and toggling the appearance of the highlighted object.

Inspector Parameters

- **Thickness:** The thickness of the outline effect
- **Custom Outline Models:** The GameObjects to use as the model to outline. If one isn't provided then the first GameObject with a valid Renderer in the current GameObject hierarchy will be used.
- **Custom Outline Model Paths:** A path to a GameObject to find at runtime, if the GameObject doesn't exist at edit time.
- **Enable Submesh Highlight:** If the mesh has multiple sub-meshes to highlight then this should be checked, otherwise only the first mesh will be highlighted.

Class Methods

Initialise/2

```
public override void Initialise(Color? color = null, Dictionary<string,  
object> options = null)
```

- Parameters

- Color? color - Not used.
- Dictionary<string, object> options - A dictionary array containing the highlighter options:

- `<'thickness', float>` - Same as `thickness` inspector parameter.
- `<'customOutlineModels', GameObject[]>` - Same as `customOutlineModels` inspector parameter.
- `<'customOutlineModelPaths', string[]>` - Same as `customOutlineModelPaths` inspector parameter.
- Returns
 - *none*

The `Initialise` method sets up the highlighter for use.

ResetHighlighter/0

```
public override void ResetHighlighter()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ResetHighlighter` method creates the additional model to use as the outline highlighted object.

Highlight/2

```
public override void Highlight(Color? color, float duration = 0f)
```

- Parameters
 - `Color? color` - The colour to outline with.
 - `float duration` - Not used.
- Returns
 - *none*

The `Highlight` method initiates the outline object to be enabled and display the outline colour.

Unhighlight/2

```
public override void Unhighlight(Color? color = null, float duration = 0f)
```

- Parameters
 - `Color? color` - Not used.
 - `float duration` - Not used.
- Returns
 - *none*

The Unhighlight method hides the outline object and removes the outline colour.

Example

`VRTK/Examples/005_Controller_BasicObjectGrabbing` demonstrates the outline highlighting on the green sphere when the controller touches it.

`VRTK/Examples/035_Controller_OpacityAndHighlighting` demonstrates the outline highlighting if the left controller collides with the green box.

Grab Attach Mechanics (VRTK/Scripts/Interactions/GrabAttachMechanics)

This directory contains scripts that are used to provide different mechanics to apply when grabbing an interactable object.

- [Base Grab Attach](#)
 - [Base Joint Grab Attach](#)
 - [Fixed Joint Grab Attach](#)
 - [Spring Joint Grab Attach](#)
 - [Custom Joint Grab Attach](#)
 - [Child Of Controller Grab Attach](#)
 - [Track Object Grab Attach](#)
 - [Rotator Track Grab Attach](#)
 - [Climbable Grab Attach](#)
-

Base Grab Attach (VRTK_BaseGrabAttach)

Overview

The Base Grab Attach script is an abstract class that all grab attach script inherit.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Inspector Parameters

- **Precision Grab:** If this is checked then when the controller grabs the object, it will grab it with precision and pick it up at the particular point on the object the controller is touching.
- **Right Snap Handle:** A Transform provided as an empty game object which must be the child of the item being grabbed and serves as an orientation point to rotate and position the grabbed

item in relation to the right handed controller. If no Right Snap Handle is provided but a Left Snap Handle is provided, then the Left Snap Handle will be used in place. If no Snap Handle is provided then the object will be grabbed at its central point. Not required for `Precision Snap`.

- **Left Snap Handle:** A Transform provided as an empty game object which must be the child of the item being grabbed and serves as an orientation point to rotate and position the grabbed item in relation to the left handed controller. If no Left Snap Handle is provided but a Right Snap Handle is provided, then the Right Snap Handle will be used in place. If no Snap Handle is provided then the object will be grabbed at its central point. Not required for `Precision Snap`.
- **Throw Velocity With Attach Distance:** If checked then when the object is thrown, the distance between the object's attach point and the controller's attach point will be used to calculate a faster throwing velocity.
- **Throw Multiplier:** An amount to multiply the velocity of the given object when it is thrown. This can also be used in conjunction with the Interact Grab Throw Multiplier to have certain objects be thrown even further than normal (or thrown a shorter distance if a number below 1 is entered).
- **On Grab Collision Delay:** The amount of time to delay collisions affecting the object when it is first grabbed. This is useful if a game object may get stuck inside another object when it is being grabbed.

Class Methods

IsTracked/0

```
public virtual bool IsTracked()
```

- Parameters
 - *none*
- Returns
 - `bool` - Is true if the mechanic is of type tracked.

The `IsTracked` method determines if the grab attach mechanic is a track object type.

IsClimbable/0

```
public virtual bool IsClimbable()
```

- Parameters
 - *none*
- Returns
 - `bool` - Is true if the mechanic is of type climbable.

The `IsClimbable` method determines if the grab attach mechanic is a climbable object type.

IsKinematic/0

```
public virtual bool IsKinematic()
```

- Parameters
 - *none*
- Returns
 - `bool` - Is true if the mechanic is of type kinematic.

The `IsKinematic` method determines if the grab attach mechanic is a kinematic object type.

ValidGrab/1

```
public virtual bool ValidGrab(Rigidbody checkAttachPoint)
```

- Parameters
 - `Rigidbody checkAttachPoint` -
- Returns
 - `bool` - Always returns true for the base check.

The `ValidGrab` method determines if the grab attempt is valid.

SetTrackPoint/1

```
public virtual void SetTrackPoint(Transform givenTrackPoint)
```

- Parameters
 - `Transform givenTrackPoint` - The track point to set on the grabbed object.
- Returns
 - *none*

The `SetTrackPoint` method sets the point on the grabbed object where the grab is happening.

SetInitialAttachPoint/1

```
public virtual void SetInitialAttachPoint(Transform givenInitialAttachPoint)
```

- Parameters
 - `Transform givenInitialAttachPoint` - The point where the initial grab took place.
- Returns
 - *none*

The `SetInitialAttachPoint` method sets the point on the grabbed object where the initial grab happened.

StartGrab/3

```
public virtual bool StartGrab(GameObject grabbingObject, GameObject  
givenGrabbedObject, Rigidbody givenControllerAttachPoint)
```

- Parameters

- `GameObject grabbingObject` - The object that is doing the grabbing.
- `GameObject givenGrabbedObject` - The object that is being grabbed.
- `Rigidbody givenControllerAttachPoint` - The point on the grabbing object that the grabbed object should be attached to after grab occurs.

- Returns

- `bool` - Is true if the grab is successful, false if the grab is unsuccessful.

The `StartGrab` method sets up the grab attach mechanic as soon as an object is grabbed.

StopGrab/1

```
public virtual void StopGrab(bool applyGrabbingObjectVelocity)
```

- Parameters

- `bool applyGrabbingObjectVelocity` - If true will apply the current velocity of the grabbing object to the grabbed object on release.

- Returns

- *none*

The `StopGrab` method ends the grab of the current object and cleans up the state.

CreateTrackPoint/4

```
public virtual Transform CreateTrackPoint(Transform controllerPoint,  
GameObject currentGrabbedObject, GameObject currentGrabbingObject, ref bool  
customTrackPoint)
```

- Parameters

- `Transform controllerPoint` - The point on the controller where the grab was initiated.
- `GameObject currentGrabbedObject` - The object that is currently being grabbed.
- `GameObject currentGrabbingObject` - The object that is currently doing the grabbing.
- `ref bool customTrackPoint` - A reference to whether the created track point is an auto generated custom object.

- Returns

- `Transform` - The transform of the created track point.

The CreateTrackPoint method sets up the point of grab to track on the grabbed object.

ProcessUpdate/0

```
public virtual void ProcessUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessUpdate method is run in every Update method on the interactable object.

ProcessFixedUpdate/0

```
public virtual void ProcessFixedUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessFixedUpdate method is run in every FixedUpdate method on the interactable object.

Base Joint Grab Attach (VRTK_BaseJointGrabAttach)

extends [VRTK_BaseGrabAttach](#)

Overview

The Base Joint Grab Attach script is an abstract class that all joint grab attach types inherit.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Inspector Parameters

- **Destroy Immediately On Throw:** Determines whether the joint should be destroyed immediately on release or whether to wait till the end of the frame before being destroyed.

Class Methods

ValidGrab/1

```
public override bool ValidGrab(Rigidbody checkAttachPoint)
```

- Parameters

- `Rigidbody checkAttachPoint` -

- Returns

- `bool` - Returns true if there is no current grab happening, or the grab is initiated by another grabbing object.

The ValidGrab method determines if the grab attempt is valid.

StartGrab/3

```
public override bool StartGrab(GameObject grabbingObject, GameObject  
givenGrabbedObject, Rigidbody givenControllerAttachPoint)
```

- Parameters

- `GameObject grabbingObject` - The object that is doing the grabbing.
- `GameObject givenGrabbedObject` - The object that is being grabbed.
- `Rigidbody givenControllerAttachPoint` - The point on the grabbing object that the grabbed object should be attached to after grab occurs.

- Returns

- `bool` - Is true if the grab is successful, false if the grab is unsuccessful.

The StartGrab method sets up the grab attach mechanic as soon as an object is grabbed. It is also responsible for creating the joint on the grabbed object.

StopGrab/1

```
public override void StopGrab(bool applyGrabbingObjectVelocity)
```

- Parameters

- `bool applyGrabbingObjectVelocity` - If true will apply the current velocity of the grabbing object to the grabbed object on release.

- Returns

- *none*

The StopGrab method ends the grab of the current object and cleans up the state. It is also responsible for removing the joint from the grabbed object.

Fixed Joint Grab Attach (VRTK_FixedJointGrabAttach)

extends [VRTK_BaseJointGrabAttach](#)

Overview

The Fixed Joint Grab Attach script is used to create a simple Fixed Joint connection between the object and the grabbing object.

Inspector Parameters

- **Break Force:** Maximum force the joint can withstand before breaking. Infinity means unbreakable.

Example

[VRTK/Examples/005_Controller_BasicObjectGrabbing](#) demonstrates this grab attach mechanic all of the grabbable objects in the scene.

Spring Joint Grab Attach (VRTK_SpringJointGrabAttach)

extends [VRTK_BaseJointGrabAttach](#)

Overview

The Spring Joint Grab Attach script is used to create a simple Spring Joint connection between the object and the grabbing object.

Inspector Parameters

- **Break Force:** Maximum force the joint can withstand before breaking. Infinity means unbreakable.
- **Strength:** The strength of the spring.
- **Damper:** The amount of dampening to apply to the spring.

Example

[VRTK/Examples/021_Controller_GrabbingObjectsWithJoints](#) demonstrates this grab attach mechanic on the Drawer object in the scene.

Custom Joint Grab Attach (VRTK_CustomJointGrabAttach)

extends [VRTK_BaseJointGrabAttach](#)

Overview

The Custom Joint Grab Attach script allows a custom joint to be provided for the grab attach mechanic.

The custom joint is placed on the interactable object and at runtime the joint is copied into a `JointHolder` game object that becomes a child of the interactable object.

The custom joint is then copied from this `JointHolder` to the interactable object when a grab happens and is removed when a grab ends.

Inspector Parameters

- **Custom Joint:** The joint to use for the grab attach joint.

Example

`VRTK/Examples/021_Controller_GrabbingObjectsWithJoints` demonstrates this grab attach mechanic on the Lamp object in the scene.

Child Of Controller Grab Attach (VRTK_ChildOfControllerGrabAttach)

extends [VRTK_BaseGrabAttach](#)

Overview

The Child Of Controller Grab Attach script is used to make the grabbed object a child of the grabbing object upon grab.

The object upon grab will naturally track the position and rotation of the grabbing object as it is a child of the grabbing game object.

The rigidbody of the object will be set to kinematic upon grab and returned to it's original state on release.

Class Methods

StartGrab/3

```
public override bool StartGrab(GameObject grabbingObject, GameObject
```

```
givenGrabbedObject, Rigidbody givenControllerAttachPoint)
```

- Parameters

- `GameObject grabbingObject` - The object that is doing the grabbing.
- `GameObject givenGrabbedObject` - The object that is being grabbed.
- `Rigidbody givenControllerAttachPoint` - The point on the grabbing object that the grabbed object should be attached to after grab occurs.

- Returns

- `bool` - Is true if the grab is successful, false if the grab is unsuccessful.

The `StartGrab` method sets up the grab attach mechanic as soon as an object is grabbed. It is also responsible for creating the joint on the grabbed object.

StopGrab/1

```
public override void StopGrab(bool applyGrabbingObjectVelocity)
```

- Parameters

- `bool applyGrabbingObjectVelocity` - If true will apply the current velocity of the grabbing object to the grabbed object on release.

- Returns

- *none*

The `StopGrab` method ends the grab of the current object and cleans up the state.

Example

`VRTK/Examples/023_Controller_ChildOfControllerOnGrab` uses this grab attach mechanic for the bow and the arrow.

Track Object Grab Attach (`VRTK_TrackObjectGrabAttach`)

```
extends VRTK\_BaseGrabAttach
```

Overview

The Track Object Grab Attach script doesn't attach the object to the controller via a joint, instead it ensures the object tracks the direction of the controller.

This works well for items that are on hinged joints or objects that require to interact naturally with other scene rigidbodies.

Inspector Parameters

- **Detach Distance:** The maximum distance the grabbing controller is away from the object before it is automatically dropped.
- **Velocity Limit:** The maximum amount of velocity magnitude that can be applied to the object. Lowering this can prevent physics glitches if objects are moving too fast.
- **Angular Velocity Limit:** The maximum amount of angular velocity magnitude that can be applied to the object. Lowering this can prevent physics glitches if objects are moving too fast.

Class Methods

StopGrab/1

```
public override void StopGrab(bool applyGrabbingObjectVelocity)
```

- Parameters
 - `bool applyGrabbingObjectVelocity` - If true will apply the current velocity of the grabbing object to the grabbed object on release.
- Returns
 - *none*

The StopGrab method ends the grab of the current object and cleans up the state.

CreateTrackPoint/4

```
public override Transform CreateTrackPoint(Transform controllerPoint,  
GameObject currentGrabbedObject, GameObject currentGrabbingObject, ref bool  
customTrackPoint)
```

- Parameters
 - `Transform controllerPoint` - The point on the controller where the grab was initiated.
 - `GameObject currentGrabbedObject` - The object that is currently being grabbed.
 - `GameObject currentGrabbingObject` - The object that is currently doing the grabbing.
 - `ref bool customTrackPoint` - A reference to whether the created track point is an auto generated custom object.
- Returns
 - `Transform` - The transform of the created track point.

The CreateTrackPoint method sets up the point of grab to track on the grabbed object.

ProcessUpdate/0

```
public override void ProcessUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessUpdate method is run in every Update method on the interactable object. It is responsible for checking if the tracked object has exceeded it's detach distance.

ProcessFixedUpdate/0

```
public override void ProcessFixedUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessFixedUpdate method is run in every FixedUpdate method on the interactable object. It applies velocity to the object to ensure it is tracking the grabbing object.

Example

VRTK/Examples/021_Controller_GrabbingObjectsWithJoints demonstrates this grab attach mechanic on the Chest handle and Fire Extinguisher body.

Rotator Track Grab Attach (VRTK_RotatorTrackGrabAttach)

```
extends VRTK_TrackObjectGrabAttach
```

Overview

The Rotator Track Grab Attach script is used to track the object but instead of the object tracking the direction of the controller, a force is applied to the object to cause it to rotate.

This is ideal for hinged joints on items such as wheels or doors.

Class Methods

StopGrab/1

```
public override void StopGrab(bool applyGrabbingObjectVelocity)
```

- Parameters
 - `bool applyGrabbingObjectVelocity` - If true will apply the current velocity of the grabbing object to the grabbed object on release.
- Returns
 - *none*

The StopGrab method ends the grab of the current object and cleans up the state.

ProcessFixedUpdate/0

```
public override void ProcessFixedUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessFixedUpdate method is run in every FixedUpdate method on the interactable object. It applies a force to the grabbed object to move it in the direction of the grabbing object.

Example

`VRTK/Examples/021_Controller_GrabbingObjectsWithJoints` demonstrates this grab attach mechanic on the Wheel and Door objects in the scene.

Climbable Grab Attach (VRTK_ClimbableGrabAttach)

extends `VRTK_BaseGrabAttach`

Overview

The Climbable Grab Attach script is used to mark the object as a climbable interactable object.

Inspector Parameters

- **Use Object Rotation:** Will respect the grabbed climbing object's rotation if it changes dynamically

Example

`VRTK/Examples/037_CameraRig_ClimbingFalling` uses this grab attach mechanic for each item that is climbable in the scene.

Secondary Controller Grab Actions (VRTK/Scripts/Interactions/SecondaryControllerGrabActions)

This directory contains scripts that are used to provide different actions when a secondary controller grabs a grabbed object.

- [Base Grab Action](#)
 - [Swap Controller Grab Action](#)
 - [Axis Scale Grab Action](#)
 - [Control Direction Grab Action](#)
-

Base Grab Action (VRTK_BaseGrabAction)

Overview

The Base Grab Action is an abstract class that all other secondary controller actions inherit and are required to implement the public abstract methods.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Class Methods

Initialise/5

```
public virtual void Initialise(VRTK_InteractableObject currentGrabbdObject,
VRTK_InteractGrab currentPrimaryGrabbingObject, VRTK_InteractGrab
currentSecondaryGrabbingObject, Transform primaryGrabPoint, Transform
secondaryGrabPoint)
```

Parameters

- `VRTK_InteractableObject currentGrabbdObject` - The Interactable Object script for the object currently being grabbed by the primary controller.
- `VRTK_InteractGrab currentPrimaryGrabbingObject` - The Interact Grab script for the object that is associated with the primary controller.
- `VRTK_InteractGrab currentSecondaryGrabbingObject` - The Interact Grab script for the object that is associated with the secondary controller.
- `Transform primaryGrabPoint` - The point on the object where the primary controller initially grabbed the object.

- `Transform secondaryGrabPoint` - The point on the object where the secondary controller initially grabbed the object.
- Returns
 - *none*

The Initialise method is used to set up the state of the secondary action when the object is initially grabbed by a secondary controller.

ResetAction/0

```
public virtual void ResetAction()
```

- Parameters
 - *none*
- Returns
 - *none*

The ResetAction method is used to reset the secondary action when the object is no longer grabbed by a secondary controller.

IsActionable/0

```
public virtual bool IsActionable()
```

- Parameters
 - *none*
- Returns
 - `bool` - Is true if the secondary grab action does perform an action on secondary grab.

The IsActionable method is used to determine if the secondary grab action performs an action on grab.

IsSwappable/0

```
public virtual bool IsSwappable()
```

- Parameters
 - *none*
- Returns
 - `bool` - Is true if the grab action allows swapping to another grabbing object.

The IsSwappable method is used to determine if the secondary grab action allows to swab the grab state to another grabbing object.

ProcessUpdate/0

```
public virtual void ProcessUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessUpdate method runs in every Update on the Interactable Object whilst it is being grabbed by a secondary controller.

ProcessFixedUpdate/0

```
public virtual void ProcessFixedUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessFixedUpdate method runs in every FixedUpdate on the Interactable Object whilst it is being grabbed by a secondary controller.

OnDropAction/0

```
public virtual void OnDropAction()
```

- Parameters
 - *none*
- Returns
 - *none*

The OnDropAction method is executed when the current grabbed object is dropped and can be used up to clean up any secondary grab actions.

Swap Controller Grab Action (VRTK_SwapControllerGrabAction)

extends [VRTK_BaseGrabAction](#)

Overview

The Swap Controller Grab Action provides a mechanism to allow grabbed objects to be swapped between controllers.

Example

`VRTK/Examples/005_Controller_BasicObjectGrabbing` demonstrates the ability to swap objects between controllers on grab.

Axis Scale Grab Action (VRTK_AxisScaleGrabAction)

extends `VRTK_BaseGrabAction`

Overview

The Axis Scale Grab Action provides a mechanism to scale objects when they are grabbed with a secondary controller.

Inspector Parameters

- **Ungrab Distance:** The distance the secondary controller must move away from the original grab position before the secondary controller auto ungrabs the object.
- **Lock X Axis:** If checked the current X Axis of the object won't be scaled
- **Lock Y Axis:** If checked the current Y Axis of the object won't be scaled
- **Lock Z Axis:** If checked the current Z Axis of the object won't be scaled
- **Uniform Scaling:** If checked all the axes will be scaled together (unless locked)

Class Methods

Initialise/5

```
public override void Initialise(VRTK_InteractableObject currentGrabbdObject,
VRTK_InteractGrab currentPrimaryGrabbingObject, VRTK_InteractGrab
currentSecondaryGrabbingObject, Transform primaryGrabPoint, Transform
secondaryGrabPoint)
```

Parameters

- `VRTK_InteractableObject currentGrabbdObject` - The Interactable Object script for the object currently being grabbed by the primary controller.
- `VRTK_InteractGrab currentPrimaryGrabbingObject` - The Interact Grab script for the object that is associated with the primary controller.

- `VRTK_InteractGrab currentSecondaryGrabbingObject` - The Interact Grab script for the object that is associated with the secondary controller.
- `Transform primaryGrabPoint` - The point on the object where the primary controller initially grabbed the object.
- `Transform secondaryGrabPoint` - The point on the object where the secondary controller initially grabbed the object.
- Returns
 - *none*

The Initialise method is used to set up the state of the secondary action when the object is initially grabbed by a secondary controller.

ProcessUpdate/0

```
public override void ProcessUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessUpdate method runs in every Update on the Interactable Object whilst it is being grabbed by a secondary controller.

ProcessFixedUpdate/0

```
public override void ProcessFixedUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessFixedUpdate method runs in every FixedUpdate on the Interactable Object whilst it is being grabbed by a secondary controller and performs the scaling action.

Example

`VRTK/Examples/043_Controller_SecondaryControllerActions` demonstrates the ability to grab an object with one controller and scale it by grabbing and pulling with the second controller.

Control Direction Grab Action (VRTK_ControlDirectionGrabAction)

extends [VRTK_BaseGrabAction](#)

Overview

The Control Direction Grab Action provides a mechanism to control the facing direction of the object when they are grabbed with a secondary controller.

For an object to correctly be rotated it must be created with the front of the object pointing down the z-axis (forward) and the upwards of the object pointing up the y-axis (up).

It's not possible to control the direction of an interactable object with a `Fixed_Joint` as the joint fixes the rotation of the object.

Inspector Parameters

- **Ungrab Distance:** The distance the secondary controller must move away from the original grab position before the secondary controller auto ungrabs the object.
- **Release Snap Speed:** The speed in which the object will snap back to it's original rotation when the secondary controller stops grabbing it. 0 for instant snap, `infinity` for no snap back.
- **Lock Z Rotation:** Prevent the secondary controller rotating the grabbed object through it's z-axis.

Class Methods

Initialise/5

```
public override void Initialise(VRTK_InteractableObject currentGrabbdObject,
VRTK_InteractGrab currentPrimaryGrabbingObject, VRTK_InteractGrab
currentSecondaryGrabbingObject, Transform primaryGrabPoint, Transform
secondaryGrabPoint)
```

• Parameters

- `VRTK_InteractableObject currentGrabbdObject` - The Interactable Object script for the object currently being grabbed by the primary controller.
- `VRTK_InteractGrab currentPrimaryGrabbingObject` - The Interact Grab script for the object that is associated with the primary controller.
- `VRTK_InteractGrab currentSecondaryGrabbingObject` - The Interact Grab script for the object that is associated with the secondary controller.
- `Transform primaryGrabPoint` - The point on the object where the primary controller initially grabbed the object.
- `Transform secondaryGrabPoint` - The point on the object where the secondary controller

initially grabbed the object.

- Returns
 - *none*

The Initialise method is used to set up the state of the secondary action when the object is initially grabbed by a secondary controller.

ResetAction/0

```
public override void ResetAction()
```

- Parameters
 - *none*
- Returns
 - *none*

The ResetAction method is used to reset the secondary action when the object is no longer grabbed by a secondary controller.

OnDropAction/0

```
public override void OnDropAction()
```

- Parameters
 - *none*
- Returns
 - *none*

The OnDropAction method is executed when the current grabbed object is dropped and can be used up to clean up any secondary grab actions.

ProcessUpdate/0

```
public override void ProcessUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessUpdate method runs in every Update on the Interactable Object whilst it is being grabbed by a secondary controller.

ProcessFixedUpdate/0

```
public override void ProcessFixedUpdate()
```

- Parameters
 - *none*
- Returns
 - *none*

The ProcessFixedUpdate method runs in every FixedUpdate on the Interactable Object whilst it is being grabbed by a secondary controller and influences the rotation of the object.

Example

VRTK/Examples/043_Controller_SecondaryControllerActions demonstrates the ability to grab an object with one controller and control their direction with the second controller.

Presence (VRTK/Scripts/Presence)

A collection of scripts that provide the ability to deal with tracking the world around the user in the scene.

- [Headset Collision](#)
 - [Headset Fade](#)
 - [Headset Collision Fade](#)
 - [Headset Controller Aware](#)
 - [Hip Tracking](#)
 - [Body Physics](#)
 - [Position Rewind](#)
-

Headset Collision (VRTK_HeadsetCollision)

Overview

The purpose of the Headset Collision is to detect when the user's VR headset collides with another game object.

The Headset Collision script will automatically create a script on the headset to deal with the collision events.

Inspector Parameters

- **Ignore Trigger Colliders:** If this is checked then the headset collision will ignore colliders set to `IsTrigger = true`.
- **Collider Radius:** The radius of the auto generated sphere collider for detecting collisions on the headset.
- **Target List Policy:** A specified `VRTK_PolicyList` to use to determine whether any objects will be acted upon by the Headset Collision.

Class Variables

- `public bool headsetColliding` - Determines if the headset is currently colliding with another object. Default: `false`
- `public Collider collidingWith` - Stores the collider of what the headset is colliding with. Default: `null`

Class Events

- `HeadsetCollisionDetect` - Emitted when the user's headset collides with another game object.
- `HeadsetCollisionEnded` - Emitted when the user's headset stops colliding with a game object.

Unity Events

Adding the `VRTK_HeadsetCollision_UnityEvents` component to `VRTK_HeadsetCollision` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `Collider collider` - The Collider of the game object the headset has collided with.
- `Transform currentTransform` - The current Transform of the object that the Headset Collision Fade script is attached to (Camera).

Class Methods

IsColliding/0

```
public virtual bool IsColliding()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the headset is currently colliding with a valid game object.

The `IsColliding` method is used to determine if the headset is currently colliding with a valid game object and returns true if it is and false if it is not colliding with anything or an invalid game object.

GetHeadsetColliderContainer/0

```
public virtual GameObject GetHeadsetColliderContainer()
```

- Parameters
 - *none*
- Returns
 - `GameObject` - The auto generated headset collider `GameObject`.

The `GetHeadsetColliderContainer` method returns the auto generated `GameObject` that contains the headset collider.

Example

`VRTK/Examples/011_Camera_HeadSetCollisionFading` has collidable walls around the play area and if the user puts their head into any of the walls then the headset will fade to black.

Headset Fade (VRTK_HeadsetFade)

Overview

The purpose of the Headset Fade is to change the colour of the headset view to a specified colour over a given duration and to also unfade it back to being transparent.

The `Fade` and `Unfade` methods can only be called via another script and this Headset Fade script does not do anything on initialisation to fade or unfade the headset view.

Class Events

- `HeadsetFadeStart` - Emitted when the user's headset begins to fade to a given colour.
- `HeadsetFadeComplete` - Emitted when the user's headset has completed the fade and is now fully at the given colour.
- `HeadsetUnfadeStart` - Emitted when the user's headset begins to unfade back to a transparent colour.
- `HeadsetUnfadeComplete` - Emitted when the user's headset has completed unfading and is now fully transparent again.

Unity Events

Adding the `VRTK_HeadsetFade_UnityEvents` component to `VRTK_HeadsetFade` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `float timeTillComplete` - A float that is the duration for the fade/unfade process has remaining.
- `Transform currentTransform` - The current Transform of the object that the Headset Fade script is attached to (Camera).

Class Methods

IsFaded/0

```
public virtual bool IsFaded()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the headset is currently fading or faded.

The `IsFaded` method returns true if the headset is currently fading or has completely faded and returns false if it is completely unfaded.

IsTransitioning/0

```
public virtual bool IsTransitioning()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the headset is currently in the process of fading or unfading.

The `IsTransitioning` method returns true if the headset is currently fading or unfading and returns false if it is completely faded or unfaded.

Fade/2

```
public virtual void Fade(Color color, float duration)
```

- Parameters
 - `Color color` - The colour to fade the headset view to.
 - `float duration` - The time in seconds to take to complete the fade transition.
- Returns
 - *none*

The `Fade` method initiates a change in the colour of the headset view to the given colour over a

given duration.

Unfade/1

```
public virtual void Unfade(float duration)
```

- **Parameters**
 - `float duration` - The time in seconds to take to complete the unfade transition.
- **Returns**
 - *none*

The Unfade method initiates the headset to change colour back to a transparent colour over a given duration.

Example

`VRTK/Examples/011_Camera_HeadSetCollisionFading` has collidable walls around the play area and if the user puts their head into any of the walls then the headset will fade to black.

Headset Collision Fade (VRTK_HeadsetCollisionFade)

Overview

The purpose of the Headset Collision Fade is to detect when the user's VR headset collides with another game object and fades the screen to a solid colour.

This is to deal with a user putting their head into a game object and seeing the inside of the object clipping, which is an undesired effect. The reasoning behind this is if the user puts their head where it shouldn't be, then fading to a colour (e.g. black) will make the user realise they've done something wrong and they'll probably naturally step backwards.

The Headset Collision Fade uses a composition of the Headset Collision and Headset Fade scripts to derive the desired behaviour.

Inspector Parameters

- **Time Till Fade:** The amount of time to wait until a fade occurs.
- **Blink Transition Speed:** The fade blink speed on collision.
- **Fade Color:** The colour to fade the headset to on collision.
- **Headset Collision:** The VRTK Headset Collision script to use when determining headset collisions. If this is left blank then the script will need to be applied to the same GameObject.
- **Headset Fade:** The VRTK Headset Fade script to use when fading the headset. If this is left blank then the script will need to be applied to the same GameObject.

Example

`VRTK/Examples/011_Camera_HeadSetCollisionFading` has collidable walls around the play area and if the user puts their head into any of the walls then the headset will fade to black.

Headset Controller Aware (VRTK_HeadsetControllerAware)

Overview

The purpose of Headset Controller Aware is to allow the headset to know if something is blocking the path between the headset and controllers and to know if the headset is looking at a controller.

Inspector Parameters

- **Track Left Controller:** If this is checked then the left controller will be checked if items obscure it's path from the headset.
- **Track Right Controller:** If this is checked then the right controller will be checked if items obscure it's path from the headset.
- **Controller Glance Radius:** The radius of the accepted distance from the controller origin point to determine if the controller is being looked at.
- **Custom Right Controller Origin:** A custom transform to provide the world space position of the right controller.
- **Custom Left Controller Origin:** A custom transform to provide the world space position of the left controller.
- **Custom Raycast:** A custom raycaster to use when raycasting to find controllers.

Class Events

- `ControllerObscured` - Emitted when the controller is obscured by another object.
- `ControllerUnobscured` - Emitted when the controller is no longer obscured by an object.
- `ControllerGlanceEnter` - Emitted when the controller is seen by the headset view.
- `ControllerGlanceExit` - Emitted when the controller is no longer seen by the headset view.

Unity Events

Adding the `VRTK_HeadsetControllerAware_UnityEvents` component to

`VRTK_HeadsetControllerAware` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `RaycastHit raycastHit` - The RaycastHit struct of item that is obscuring the path to the controller.
- `VRTK_ControllerReference controllerReference` - The reference to the controller that is being

or has been obscured or being or has been glanced.

Class Methods

LeftControllerObscured/0

```
public virtual bool LeftControllerObscured()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the path between the headset and the controller is obscured.

The `LeftControllerObscured` method returns the state of if the left controller is being obscured from the path of the headset.

RightControllerObscured/0

```
public virtual bool RightControllerObscured()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the path between the headset and the controller is obscured.

The `RightControllerObscured` method returns the state of if the right controller is being obscured from the path of the headset.

LeftControllerGlanced/0

```
public virtual bool LeftControllerGlanced()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the headset can currently see the controller within the given radius threshold.

the `LeftControllerGlanced` method returns the state of if the headset is currently looking at the left controller or not.

RightControllerGlanced/0

```
public virtual bool RightControllerGlanced()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the headset can currently see the controller within the given radius threshold.

the `RightControllerGlanced` method returns the state of if the headset is currently looking at the right controller or not.

Example

`VRTK/Examples/029_Controller_Tooltips` displays tooltips that have been added to the controllers and are only visible when the controller is being looked at.

Hip Tracking (VRTK_HipTracking)

Overview

Hip Tracking attempts to reasonably track hip position in the absence of a hip position sensor.

The Hip Tracking script is placed on an empty `GameObject` which will be positioned at the estimated hip position.

Inspector Parameters

- **Head Offset:** Distance underneath Player Head for hips to reside.
 - **Head Override:** Optional Transform to use as the Head Object for calculating hip position. If none is given one will try to be found in the scene.
 - **Reference Up:** Optional Transform to use for calculating which way is 'Up' relative to the player for hip positioning.
-

Body Physics (VRTK_BodyPhysics)

extends `VRTK_DestinationMarker`

Overview

The body physics script deals with how a user's body in the scene reacts to world physics and how to handle drops.

The body physics creates a rigidbody and collider for where the user is standing to allow physics interactions and prevent walking through walls.

Upon actually moving in the play area, the rigidbody is set to kinematic to prevent the world from being pushed back in the user's view reducing sickness.

The body physics script also deals with snapping a user to the nearest floor if they look over a ledge or walk up stairs then it will move the play area to simulate movement in the scene.

To allow for peeking over a ledge and not falling, a fall restriction can happen by keeping a controller over the existing floor and the snap to the nearest floor will not happen until the controllers are also over the floor.

Inspector Parameters

- **Enable Body Collisions:** If checked then the body collider and rigidbody will be used to check for rigidbody collisions.
- **Ignore Grabbed Collisions:** If this is checked then any items that are grabbed with the controller will not collide with the body collider. This is very useful if the user is required to grab and wield objects because if the collider was active they would bounce off the collider.
- **Ignore Collisions With:** An array of GameObjects that will not collide with the body collider.
- **Headset Y Offset:** The collider which is created for the user is set at a height from the user's headset position. If the collider is required to be lower to allow for room between the play area collider and the headset then this offset value will shorten the height of the generated collider.
- **Movement Threshold:** The amount of movement of the headset between the headset's current position and the current standing position to determine if the user is walking in play space and to ignore the body physics collisions if the movement delta is above this threshold.
- **Play Area Movement Threshold:** The amount of movement of the play area between the play area's current position and the previous position to determine if the user is moving play space.
- **Standing History Samples:** The maximum number of samples to collect of headset position before determining if the current standing position within the play space has changed.
- **Lean Y Threshold:** The y distance between the headset and the object being leaned over, if object being leaned over is taller than this threshold then the current standing position won't be updated.
- **Step Up Y Offset:** The maximum height to consider when checking if an object can be stepped upon to.
- **Step Thickness Multiplier:** The width/depth of the foot collider in relation to the radius of the body collider.
- **Step Drop Threshold:** The distance between the current play area Y position and the new stepped up Y position to consider a valid step up. A higher number can help with juddering on slopes or small increases in collider heights.
- **Custom Raycast:** A custom raycaster to use when raycasting to find floors.
- **Fall Restriction:** A check to see if the drop to nearest floor should take place. If the selected restrictor is still over the current floor then the drop to nearest floor will not occur. Works well for being able to lean over ledges and look down. Only works for falling down not teleporting up.

- **Gravity Fall Y Threshold:** When the `y` distance between the floor and the headset exceeds this distance and `Enable Body Collisions` is true then the rigidbody gravity will be used instead of teleport to drop to nearest floor.
- **Blink Y Threshold:** The `y` distance between the floor and the headset that must change before a fade transition is initiated. If the new user location is at a higher distance than the threshold then the headset blink transition will activate on teleport. If the new user location is within the threshold then no blink transition will happen, which is useful for walking up slopes, meshes and terrains to prevent constant blinking.
- **Floor Height Tolerance:** The amount the `y` position needs to change by between the current floor `y` position and the previous floor `y` position before a change in floor height is considered to have occurred. A higher value here will mean that a `Drop To Floor` will be less likely to happen if the `y` of the floor beneath the user hasn't changed as much as the given threshold.
- **Fall Check Precision:** The amount of rounding on the play area `Y` position to be applied when checking if falling is occurring.
- **Teleporter:** The VRTK Teleport script to use when snapping to floor. If this is left blank then a Teleport script will need to be applied to the same `GameObject`.
- **Custom Body Collider Container:** A `GameObject` to represent a custom body collider container. It should contain a collider component that will be used for detecting body collisions. If one isn't provided then it will be auto generated.
- **Custom Foot Collider Container:** A `GameObject` to represent a custom foot collider container. It should contain a collider component that will be used for detecting step collisions. If one isn't provided then it will be auto generated.

Class Variables

- `public enum FallingRestrictors` - Options for testing if a play space fall is valid
 - `NoRestriction` - Always drop to nearest floor when the headset is no longer over the current standing object.
 - `LeftController` - Don't drop to nearest floor if the Left Controller is still over the current standing object even if the headset isn't.
 - `RightController` - Don't drop to nearest floor if the Right Controller is still over the current standing object even if the headset isn't.
 - `EitherController` - Don't drop to nearest floor if Either Controller is still over the current standing object even if the headset isn't.
 - `BothControllers` - Don't drop to nearest floor only if Both Controllers are still over the current standing object even if the headset isn't.

Class Events

- `StartFalling` - Emitted when a fall begins.
- `StopFalling` - Emitted when a fall ends.
- `StartMoving` - Emitted when movement in the play area begins.
- `StopMoving` - Emitted when movement in the play area ends.
- `StartColliding` - Emitted when the body collider starts colliding with another game object.
- `StopColliding` - Emitted when the body collider stops colliding with another game object.

- `StartLeaning` - Emitted when the body collider starts leaning over another game object.
- `StopLeaning` - Emitted when the body collider stops leaning over another game object.
- `StartTouchingGround` - Emitted when the body collider starts touching the ground.
- `StopTouchingGround` - Emitted when the body collider stops touching the ground.

Unity Events

Adding the `VRTK_BodyPhysics_UnityEvents` component to `VRTK_BodyPhysics` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `GameObject target` - The target the event is dealing with.
- `Collider collider` - An optional collider that the body physics is colliding with.

Class Methods

ArePhysicsEnabled/0

```
public virtual bool ArePhysicsEnabled()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the body physics will interact with other scene physics objects and false if the body physics will ignore other scene physics objects.

The `ArePhysicsEnabled` method determines whether the body physics are set to interact with other scene physics objects.

ApplyBodyVelocity/3

```
public virtual void ApplyBodyVelocity(Vector3 velocity, bool forcePhysicsOn = false, bool applyMomentum = false)
```

- Parameters
 - `Vector3 velocity` - The velocity to apply.
 - `bool forcePhysicsOn` - If true will toggle the body collision physics back on if enable body collisions is true.
 - `bool applyMomentum` - If true then the existing momentum of the play area will be applied as a force to the resulting velocity.
- Returns
 - *none*

The ApplyBodyVelocity method applies a given velocity to the rigidbody attached to the body physics.

ToggleOnGround/1

```
public virtual void ToggleOnGround(bool state)
```

- Parameters
 - `bool state` - If true then body physics are set to being on the ground.
- Returns
 - *none*

The ToggleOnGround method sets whether the body is considered on the ground or not.

TogglePreventSnapToFloor/1

```
public virtual void TogglePreventSnapToFloor(bool state)
```

- Parameters
 - `bool state` - If true the the snap to floor mechanic will not execute.
- Returns
 - *none*

The PreventSnapToFloor method sets whether the snap to floor mechanic should be used.

ForceSnapToFloor/0

```
public virtual void ForceSnapToFloor()
```

- Parameters
 - *none*
- Returns
 - *none*

The ForceSnapToFloor method disables the prevent snap to floor and forces the snap to nearest floor action.

IsFalling/0

```
public virtual bool IsFalling()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the body is currently falling via gravity or via teleport.

The `IsFalling` method returns the falling state of the body.

IsMoving/0

```
public virtual bool IsMoving()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the user is currently walking around their play area space.

The `IsMoving` method returns the moving within play area state of the body.

IsLeaning/0

```
public virtual bool IsLeaning()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the user is considered to be leaning over an object.

The `IsLeaning` method returns the leaning state of the user.

OnGround/0

```
public virtual bool OnGround()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the play area is on the ground and false if the play area is in the air.

The `OnGround` method returns whether the user is currently standing on the ground or not.

GetVelocity/0

```
public virtual Vector3 GetVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - The velocity of the body physics rigidbody.

The `GetVelocity` method returns the velocity of the body physics rigidbody.

GetAngularVelocity/0

```
public virtual Vector3 GetAngularVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - The angular velocity of the body physics rigidbody.

The `GetAngularVelocity` method returns the angular velocity of the body physics rigidbody.

ResetVelocities/0

```
public virtual void ResetVelocities()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ResetVelocities` method sets the rigidbody velocity and angular velocity to zero to stop the Play Area rigidbody from continuing to move if it has a velocity already.

ResetFalling/0

```
public virtual void ResetFalling()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ResetFalling` method force stops any falling states and conditions that might be set on this object.

GetBodyColliderContainer/0

```
public virtual GameObject GetBodyColliderContainer()
```

- Parameters

- *none*

- Returns

- `GameObject` - The auto generated body collider `GameObject`.
- `GameObject` -

The `GetBodyColliderContainer` method returns the auto generated `GameObject` that contains the body colliders.

GetFootColliderContainer/0

```
public virtual GameObject GetFootColliderContainer()
```

- Parameters

- *none*

- Returns

- `GameObject` - The auto generated foot collider `GameObject`.
- `GameObject` -

The `GetFootColliderContainer` method returns the auto generated `GameObject` that contains the foot colliders.

GetCurrentCollidingObject/0

```
public virtual GameObject GetCurrentCollidingObject()
```

- Parameters

- *none*

- Returns

- `GameObject` - The `GameObject` that is colliding with the body physics colliders.

The `GetCurrentCollidingObject` method returns the object that the body physics colliders are currently colliding with.

ResetIgnoredCollisions/0

```
public virtual void ResetIgnoredCollisions()
```

- Parameters

- *none*

- Returns
 - *none*

The `ResetIgnoredCollisions` method is used to clear any stored ignored colliders in case the `IgnoreCollisions On` array parameter is changed at runtime. This needs to be called manually if changes are made at runtime.

SweepCollision/2

```
public virtual bool SweepCollision(Vector3 direction, float maxDistance)
```

- Parameters
 - `Vector3 direction` - The direction to test for the potential collision.
 - `float maxDistance` - The maximum distance to check for a potential collision.
- Returns
 - `bool` - Returns true if a collision will occur on the given direction over the given maximum distance. Returns false if there is no collision about to happen.

The `SweepCollision` method tests to see if a collision will occur with the body collider in a given direction and distance.

Example

`VRTK/Examples/017_CameraRig_TouchpadWalking` has a collection of walls and slopes that can be traversed by the user with the touchpad but the user cannot pass through the objects as they are collidable and the rigidbody physics won't allow the intersection to occur.

Position Rewind (VRTK_PositionRewind)

Overview

The Position Rewind script is used to reset the user back to a good known standing position upon receiving a headset collision event.

Inspector Parameters

- **Collision Detector:** The colliders to determine if a collision has occurred for the rewind to be actioned.
- **Ignore Trigger Colliders:** If this is checked then the collision detector will ignore colliders set to `IsTrigger = true`.
- **Rewind Delay:** The amount of time from original headset collision until the rewind to the last good known position takes place.
- **Pushback Distance:** The additional distance to push the play area back upon rewind to prevent

being right next to the wall again.

- **Crouch Threshold:** The threshold to determine how low the headset has to be before it is considered the user is crouching. The last good position will only be recorded in a non-crouching position.
- **Crouch Rewind Threshold:** The threshold to determind how low the headset can be to perform a position rewind. If the headset Y position is lower than this threshold then a rewind won't occur.
- **Target List Policy:** A specified VRTK_PolicyList to use to determine whether any objects will be acted upon by the Position Rewind.
- **Body Physics:** The VRTK Body Physics script to use for the collisions and rigidbodies. If this is left blank then the first Body Physics script found in the scene will be used.
- **Headset Collision:** The VRTK Headset Collision script to use to determine if the headset is colliding. If this is left blank then the script will need to be applied to the same GameObject.

Class Variables

- `public enum CollisionDetectors` - Valid collision detectors.
 - `HeadsetOnly` - Listen for collisions on the headset collider only.
 - `BodyOnly` - Listen for collisions on the body physics collider only.
 - `HeadsetAndBody` - Listen for collisions on both the headset collider and body physics collider.

Class Events

- `PositionRewindToSafe` - Emitted when the draggable item is successfully dropped.

Unity Events

Adding the `VRTK_PositionRewind_UnityEvents` component to `VRTK_PositionRewind` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `Vector3 collidedPosition` - The position of the play area when it collded.
- `Vector3 resetPosition` - The position of the play area when it has been rewinded to a safe position.

Class Methods

SetLastGoodPosition/0

```
public virtual void SetLastGoodPosition()
```

- Parameters
 - *none*
- Returns

- *none*

The `SetLastGoodPosition` method stores the current valid play area and headset position.

RewindPosition/0

```
public virtual void RewindPosition()
```

- Parameters

- *none*

- Returns

- *none*

The `RewindPosition` method resets the play area position to the last known good position of the play area.

Example

`VRTK/Examples/017_CameraRig_TouchpadWalking` has the position rewind script to reset the user's position if they walk into objects.

UI (VRTK/Scripts/UI)

A collection of scripts that provide the ability to utilise and interact with Unity UI elements.

- [UI Canvas](#)
 - [UI Pointer](#)
 - [UI Draggable Item](#)
 - [UI Drop Zone](#)
-

UI Canvas (VRTK_UICanvas)

Overview

The UI Canvas is used to denote which World Canvases are interactable by a UI Pointer.

When the script is enabled it will disable the `Graphic Raycaster` on the canvas and create a custom `UI Graphics Raycaster` and the Blocking Objects and Blocking Mask settings are copied over from the `Graphic Raycaster`.

Inspector Parameters

- **Click On Pointer Collision:** Determines if a UI Click action should happen when a UI Pointer game object collides with this canvas.
- **Auto Activate Within Distance:** Determines if a UI Pointer will be auto activated if a UI Pointer game object comes within the given distance of this canvas. If a value of 0 is given then no auto activation will occur.

Example

`VRTK/Examples/034_Controls_InteractingWithUnityUI` uses the `VRTK_UICanvas` script on two of the canvases to show how the UI Pointer can interact with them.

UI Pointer (VRTK_UIPointer)

Overview

The UI Pointer provides a mechanism for interacting with Unity UI elements on a world canvas. The UI Pointer can be attached to any game object the same way in which a Base Pointer can be and the UI Pointer also requires a controller to initiate the pointer activation and pointer click states.

The simplest way to use the UI Pointer is to attach the script to a game controller along with a Simple Pointer as this provides visual feedback as to where the UI ray is pointing.

The UI pointer is activated via the `Pointer` alias on the `Controller Events` and the UI pointer click state is triggered via the `UI Click` alias on the `Controller Events`.

Inspector Parameters

- **Activation Button:** The button used to activate/deactivate the UI raycast for the pointer.
- **Activation Mode:** Determines when the UI pointer should be active.
- **Selection Button:** The button used to execute the select action at the pointer's target position.
- **Click Method:** Determines when the UI Click event action should happen.
- **Attempt Click On Deactivate:** Determines whether the UI click action should be triggered when the pointer is deactivated. If the pointer is hovering over a clickable element then it will invoke the click action on that element. Note: Only works with `Click Method = Click_On_Button_Up`
- **Click After Hover Duration:** The amount of time the pointer can be over the same UI element before it automatically attempts to click it. 0f means no click attempt will be made.
- **Controller:** The controller that will be used to toggle the pointer. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Pointer Origin Transform:** A custom transform to use as the origin of the pointer. If no pointer origin transform is provided then the transform the script is attached to is used.

Class Variables

- `public enum ActivationMethods` - Methods of activation.
 - `HoldButton` - Only activates the UI Pointer when the Pointer button on the controller is pressed and held down.
 - `ToggleButton` - Activates the UI Pointer on the first click of the Pointer button on the controller and it stays active until the Pointer button is clicked again.
 - `AlwaysOn` - The UI Pointer is always active regardless of whether the Pointer button on the controller is pressed or not.
- `public enum ClickMethods` - Methods of when to consider a UI Click action
 - `ClickOnButtonUp` - Consider a UI Click action has happened when the UI Click alias button is released.
 - `ClickOnButtonDown` - Consider a UI Click action has happened when the UI Click alias button is pressed.
- `public GameObject autoActivatingCanvas` - The GameObject of the front trigger activator of the canvas currently being activated by this pointer. Default: `null`
- `public bool collisionClick` - Determines if the UI Pointer has collided with a valid canvas that has collision click turned on. Default: `false`

Class Events

- `ActivationButtonPressed` - Emitted when the UI activation button is pressed.
- `ActivationButtonReleased` - Emitted when the UI activation button is released.
- `SelectionButtonPressed` - Emitted when the UI selection button is pressed.
- `SelectionButtonReleased` - Emitted when the UI selection button is released.
- `UIPointerElementEnter` - Emitted when the UI Pointer is colliding with a valid UI element.
- `UIPointerElementExit` - Emitted when the UI Pointer is no longer colliding with any valid UI elements.
- `UIPointerElementClick` - Emitted when the UI Pointer has clicked the currently collided UI element.
- `UIPointerElementDragStart` - Emitted when the UI Pointer begins dragging a valid UI element.
- `UIPointerElementDragEnd` - Emitted when the UI Pointer stops dragging a valid UI element.

Unity Events

Adding the `VRTK_UIPointer_UnityEvents` component to `VRTK_UIPointer` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_ControllerReference controllerReference` - The reference to the controller that was used.
- `bool isActive` - The state of whether the UI Pointer is currently active or not.
- `GameObject currentTarget` - The current UI element that the pointer is colliding with.
- `GameObject previousTarget` - The previous UI element that the pointer was colliding with.

- `RaycastResult raycastResult` - The raw raycast result of the UI ray collision.

Class Methods

SetEventSystem/1

```
public virtual VRTK_VRInputModule SetEventSystem(EventSystem eventSystem)
```

- Parameters
 - `EventSystem eventSystem` - The global Unity event system to be used by the UI pointers.
- Returns
 - `VRTK_VRInputModule` - A custom input module that is used to detect input from VR pointers.

The `SetEventSystem` method is used to set up the global Unity event system for the UI pointer. It also handles disabling the existing Standalone Input Module that exists on the `EventSystem` and adds a custom VRTK Event System VR Input component that is required for interacting with the UI with VR inputs.

RemoveEventSystem/0

```
public virtual void RemoveEventSystem()
```

- Parameters
 - *none*
- Returns
 - *none*

The `RemoveEventSystem` resets the Unity `EventSystem` back to the original state before the `VRTK_VRInputModule` was swapped for it.

PointerActive/0

```
public virtual bool PointerActive()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the ui pointer should be currently active.

The `PointerActive` method determines if the ui pointer beam should be active based on whether the pointer alias is being held and whether the Hold Button To Use parameter is checked.

IsActivationButtonPressed/0

```
public virtual bool IsActivationButtonPressed()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the activation button is active.

The `IsActivationButtonPressed` method is used to determine if the configured activation button is currently in the active state.

IsSelectionButtonPressed/0

```
public virtual bool IsSelectionButtonPressed()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the selection button is active.

The `IsSelectionButtonPressed` method is used to determine if the configured selection button is currently in the active state.

ValidClick/2

```
public virtual bool ValidClick(bool checkLastClick, bool lastClickState = false)
```

- Parameters
 - `bool checkLastClick` - If this is true then the last frame's state of the UI Click button is also checked to see if a valid click has happened.
 - `bool lastClickState` - This determines what the last frame's state of the UI Click button should be in for it to be a valid click.
- Returns
 - `bool` - Returns true if the UI Click button is in a valid state to action a click, returns false if it is not in a valid state.

The `ValidClick` method determines if the UI Click button is in a valid state to register a click action.

GetOriginPosition/0

```
public virtual Vector3 GetOriginPosition()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` of the pointer transform position

The `GetOriginPosition` method returns the relevant transform position for the pointer based on whether the `pointerOriginTransform` variable is valid.

GetOriginForward/0

```
public virtual Vector3 GetOriginForward()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` of the pointer transform forward

The `GetOriginPosition` method returns the relevant transform forward for the pointer based on whether the `pointerOriginTransform` variable is valid.

Example

`VRTK/Examples/034_Controls_InteractingWithUnityUI` uses the `VRTK_UIPointer` script on the right Controller to allow for the interaction with Unity UI elements using a Simple Pointer beam. The left Controller controls a Simple Pointer on the headset to demonstrate gaze interaction with Unity UI elements.

UI Draggable Item (VRTK_UIDraggableItem)

extends `MonoBehaviour`, `IBeginDragHandler`, `IDragHandler`, `IEndDragHandler`

Overview

The UI Draggable item will make any UI element draggable on the canvas.

If a UI Draggable item is set to `Restrict To Drop Zone = true` then the UI Draggable item must be a child of an element that has the `VRTK_UIDropZone` script applied to it to ensure it starts in a valid drop zone.

Inspector Parameters

- **Restrict To Drop Zone:** If checked then the UI element can only be dropped in valid a

VRTK_UIDropZone object and must start as a child of a VRTK_UIDropZone object. If unchecked then the UI element can be dropped anywhere on the canvas.

- **Restrict To Original Canvas:** If checked then the UI element can only be dropped on the original parent canvas. If unchecked the UI element can be dropped on any valid VRTK_UICanvas.
- **Forward Offset:** The offset to bring the UI element forward when it is being dragged.

Class Variables

- `public GameObject validDropZone` - The current valid drop zone the dragged element is hovering over.

Class Events

- `DraggableItemDropped` - Emitted when the draggable item is successfully dropped.
- `DraggableItemReset` - Emitted when the draggable item is reset.

Unity Events

Adding the `VRTK_UIDraggableItem_UnityEvents` component to `VRTK_UIDraggableItem` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `GameObject target` - The target the item is dragged onto.

Example

`VRTK/Examples/034_Controls_InteractingWithUnityUI` demonstrates a collection of UI elements that are draggable

UI Drop Zone (VRTK_UIDropZone)

extends `MonoBehaviour`, `IPointerEnterHandler`, `IPointerExitHandler`

Overview

A UI Drop Zone is applied to any UI element that is to be considered a valid parent for any UI Draggable element to be dropped into it.

It's usually appropriate to use a Panel UI element as a drop zone with a layout group applied so new children dropped into the drop zone automatically align.

Example

`VRTK/Examples/034_Controls_InteractingWithUnityUI` demonstrates a collection of UI Drop Zones.

3D Controls (VRTK/Scripts/Controls/3D)

In order to interact with the world beyond grabbing and throwing, controls can be used to mimic real-life objects.

A number of controls are available which partially support auto-configuration. So can a slider for example detect its min and max points or a button the distance until a push event should be triggered. In the editor gizmos will be drawn that show the current settings. A yellow gizmo signals a valid configuration. A red one shows that the position of the object should change or switch to manual configuration mode.

All 3D controls extend the `VRTK_Control` abstract class which provides common methods and events.

- [Control](#)
 - [Button](#)
 - [Chest](#)
 - [Door](#)
 - [Drawer](#)
 - [Knob](#)
 - [Wheel](#)
 - [Lever](#)
 - [Spring Lever](#)
 - [Slider](#)
 - [Content Handler](#)
-

Control (VRTK_Control)

Overview

All 3D controls extend the `VRTK_Control` abstract class which provides a default set of methods and events that all of the subsequent controls expose.

Inspector Parameters

- **Interact Without Grab:** If active the control will react to the controller without the need to push the grab button.

Class Variables

- `public ValueChangedEvent OnValueChanged` - Emitted when the control is interacted with.
- `public enum Direction` - 3D Control Directions
 - `autodetect` - Attempt to auto detect the axis
 - `x` - X axis
 - `y` - Y axis
 - `z` - Z axis

Class Events

- `ValueChanged` - Emitted when the 3D Control value has changed.

Unity Events

Adding the `VRTK_Control_UnityEvents` component to `VRTK_Control` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `float value` - The current value being reported by the control.
- `float normalizedValue` - The normalized value being reported by the control.

Class Methods

GetValue/0

```
public virtual float GetValue()
```

- Parameters
 - *none*
- Returns
 - `float` - The current value of the control.

The `GetValue` method returns the current value/position/setting of the control depending on the control that is extending this abstract class.

GetNormalizedValue/0

```
public virtual float GetNormalizedValue()
```

- Parameters
 - *none*
- Returns

- `float` - The current normalized value of the control.

The `GetNormalizedValue` method returns the current value mapped onto a range between 0 and 100.

SetContent/2

```
public virtual void SetContent(GameObject content, bool hideContent)
```

- Parameters

- `GameObject content` - The content to be considered within the control.
- `bool hideContent` - When true the content will be hidden in addition to being non-interactable in case the control is fully closed.

- Returns

- *none*

The `SetContent` method sets the given game object as the content of the control. This will then disable and optionally hide the content when a control is obscuring its view to prevent interacting with content within a control.

GetContent/0

```
public virtual GameObject GetContent()
```

- Parameters

- *none*

- Returns

- `GameObject` - The currently stored content for the control.

The `GetContent` method returns the current game object of the control's content.

Button (VRTK_Button)

extends [VRTK_Control](#)

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a push button. The direction into which the button should be pushable can be freely set and auto-detection is supported. Since this is physics-based there needs to be empty space in the push direction so that the button can move.

The script will instantiate the required Rigidbody and ConstantForce components automatically in case they do not exist yet.

Inspector Parameters

- **Connected To:** An optional game object to which the button will be connected. If the game object moves the button will follow along.
- **Direction:** The axis on which the button should move. All other axis will be frozen.
- **Activation Distance:** The local distance the button needs to be pushed until a push event is triggered.
- **Button Strength:** The amount of force needed to push the button down as well as the speed with which it will go back into its original position.

Class Variables

- `public enum ButtonDirection` - 3D Control Button Directions
 - `autodetect` - Attempt to auto detect the axis
 - `x` - X axis
 - `y` - Y axis
 - `z` - Z axis
 - `negX` - Negative X axis
 - `negY` - Negative Y axis
 - `negZ` - Negative Z axis

Class Events

- `Pushed` - Emitted when the 3D Button has reached its activation distance.
- `Released` - Emitted when the 3D Button's position has become less than activation distance after being pressed.

Unity Events

Adding the `VRTK_Button_UnityEvents` component to `VRTK_Button` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `float value` - The current value being reported by the control.
- `float normalizedValue` - The normalized value being reported by the control.

Example

`VRTK/Examples/025_Controls_Overview` shows a collection of pressable buttons that are interacted with by activating the rigidbody on the controller by pressing the grab button without grabbing an object.

Chest (VRTK_Chest)

extends [VRTK_Control](#)

Overview

Transforms a game object into a chest with a lid. The direction can be auto-detected with very high reliability or set manually.

The script will instantiate the required Rigidbody, Interactable and HingeJoint components automatically in case they do not exist yet. It will expect three distinct game objects: a body, a lid and a handle. These should be independent and not children of each other.

Inspector Parameters

- **Direction:** The axis on which the chest should open. All other axis will be frozen.
- **Lid:** The game object for the lid.
- **Body:** The game object for the body.
- **Handle:** The game object for the handle.
- **Content:** The parent game object for the chest content elements.
- **Hide Content:** Makes the content invisible while the chest is closed.
- **Max Angle:** The maximum opening angle of the chest.

Example

[VRTK/Examples/025_Controls_Overview](#) shows a chest that can be open and closed, it also displays the current opening angle of the chest.

Door (VRTK_Door)

extends [VRTK_Control](#)

Overview

Transforms a game object into a door with an optional handle attached to an optional frame. The direction can be freely set and also very reliably auto-detected.

There are situations when it can be very hard to automatically calculate the correct axis and anchor values for the hinge joint. If this situation is encountered then simply add the hinge joint manually and set these two values. All the rest will still be handled by the script.

The script will instantiate the required Rigidbodies, Interactable and HingeJoint components automatically in case they do not exist yet. Gizmos will indicate the direction.

Inspector Parameters

- **Direction:** The axis on which the door should open.
- **Door:** The game object for the door. Can also be an empty parent or left empty if the script is put onto the actual door mesh. If no colliders exist yet a collider will be automatically attached to all children that expose renderers.
- **Handles:** The game object for the handles. Can also be an empty parent or left empty. If empty the door can only be moved using the rigidbody mode of the controller. If no collider exists yet a compound collider made up of all children will try to be calculated but this will fail if the door is rotated. In that case a manual collider will need to be assigned.
- **Frame:** The game object for the frame to which the door is attached. Should only be set if the frame will move as well to ensure that the door moves along with the frame.
- **Content:** The parent game object for the door content elements.
- **Hide Content:** Makes the content invisible while the door is closed.
- **Max Angle:** The maximum opening angle of the door.
- **Open Inward:** Can the door be pulled to open.
- **Open Outward:** Can the door be pushed to open.
- **Min Snap Close:** The range at which the door must be to being closed before it snaps shut. Only works if either inward or outward is selected, not both.
- **Released Friction:** The amount of friction the door will have whilst swinging when it is not grabbed.
- **Grabbed Friction:** The amount of friction the door will have whilst swinging when it is grabbed.
- **Handle Interactable Only:** If this is checked then only the door handle is grabbable to operate the door.

Example

[VRTK/Examples/025_Controls_Overview](#) shows a selection of door types, from a normal door and trapdoor, to a door with a cat-flap in the middle.

Drawer (VRTK_Drawer)

extends [VRTK_Control](#)

Overview

Transforms a game object into a drawer. The direction can be freely set and also auto-detected with very high reliability.

The script will instantiate the required Rigidbody, Interactable and Joint components automatically in case they do not exist yet. There are situations when it can be very hard to automatically calculate the correct axis for the joint. If this situation is encountered simply add the configurable joint manually and set the axis. All the rest will still be handled by the script.

It will expect two distinct game objects: a body and a handle. These should be independent and not children of each other. The distance to which the drawer can be pulled out will automatically set depending on the length of it. If no body is specified the current object is assumed to be the body.

It is possible to supply a third game object which is the root of the contents inside the drawer. When this is specified the VRTK_InteractableObject components will be automatically deactivated in case the drawer is closed or not yet far enough open. This eliminates the issue that a user could grab an object inside a drawer although it is closed.

Inspector Parameters

- **Connected To:** An optional game object to which the drawer will be connected. If the game object moves the drawer will follow along.
- **Direction:** The axis on which the drawer should open. All other axis will be frozen.
- **Body:** The game object for the body.
- **Handle:** The game object for the handle.
- **Content:** The parent game object for the drawer content elements.
- **Hide Content:** Makes the content invisible while the drawer is closed.
- **Min Snap Close:** If the extension of the drawer is below this percentage then the drawer will snap shut.
- **Max Extend:** The maximum percentage of the drawer's total length that the drawer will open to.

Example

[VRTK/Examples/025_Controls_Overview](#) shows a drawer with contents that can be opened and closed freely and the contents can be removed from the drawer.

Knob (VRTK_Knob)

extends [VRTK_Control](#)

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a radial knob. The direction can be freely set.

The script will instantiate the required Rigidbody and Interactable components automatically in case they do not exist yet.

Inspector Parameters

- **Connected To:** An optional game object to which the knob will be connected. If the game object moves the knob will follow along.
- **Direction:** The axis on which the knob should rotate. All other axis will be frozen.
- **Min:** The minimum value of the knob.
- **Max:** The maximum value of the knob.
- **Step Size:** The increments in which knob values can change.

Example

`VRTK/Examples/025_Controls_Overview` has a couple of rotator knobs that can be rotated by grabbing with the controller and then rotating the controller in the desired direction.

Wheel (VRTK_Wheel)

extends [`VRTK_Control`](#)

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a spinnable wheel.

The script will instantiate the required Rigidbody and Interactable components automatically in case they do not exist yet.

Inspector Parameters

- **Connected To:** An optional game object to which the wheel will be connected. If the game object moves the wheel will follow along.
- **Grab Type:** The grab attach mechanic to use. Track Object allows for rotations of the controller, Rotator Track allows for grabbing the wheel and spinning it.
- **Detatch Distance:** The maximum distance the grabbing controller is away from the wheel before it is automatically released.
- **Minimum Value:** The minimum value the wheel can be set to.
- **Maximum Value:** The maximum value the wheel can be set to.
- **Step Size:** The increments in which values can change.
- **Snap To Step:** If this is checked then when the wheel is released, it will snap to the step rotation.
- **Grabbed Friction:** The amount of friction the wheel will have when it is grabbed.
- **Released Friction:** The amount of friction the wheel will have when it is released.
- **Max Angle:** The maximum angle the wheel has to be turned to reach it's maximum value.
- **Lock At Limits:** If this is checked then the wheel cannot be turned beyond the minimum and maximum value.

Example

VRTK/Examples/025_Controls_Overview has a collection of wheels that can be rotated by grabbing with the controller and then rotating the controller in the desired direction.

Lever (VRTK_Lever)

extends VRTK_Control

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a lever. The direction can be freely set.

The script will instantiate the required Rigidbody, Interactable and HingeJoint components automatically in case they do not exist yet. The joint is very tricky to setup automatically though and will only work in straight forward cases. If there are any issues, then create the HingeJoint component manually and configure it as needed.

Inspector Parameters

- **Connected To:** An optional game object to which the lever will be connected. If the game object moves the lever will follow along.
- **Direction:** The axis on which the lever should rotate. All other axis will be frozen.
- **Min Angle:** The minimum angle of the lever counted from its initial position.
- **Max Angle:** The maximum angle of the lever counted from its initial position.
- **Step Size:** The increments in which lever values can change.
- **Released Friction:** The amount of friction the lever will have whilst swinging when it is not grabbed.
- **Grabbed Friction:** The amount of friction the lever will have whilst swinging when it is grabbed.

Example

VRTK/Examples/025_Controls_Overview has a couple of levers that can be grabbed and moved. One lever is horizontal and the other is vertical.

Spring Lever (VRTK_SpringLever)

extends VRTK_Lever

Overview

This script extends `VRTK_Lever` to add spring force toward whichever end of the lever's range it is closest to.

The script will instantiate the required `Rigidbody`, `Interactable` and `HingeJoint` components automatically in case they do not exist yet. The joint is very tricky to setup automatically though and will only work in straight forward cases. If there are any issues, then create the `HingeJoint` component manually and configure it as needed.

Inspector Parameters

- **Spring Strength:** The strength of the spring force that will be applied upon the lever.
 - **Spring Damper:** The damper of the spring force that will be applied upon the lever.
 - **Snap To Nearest Limit:** If this is checked then the spring will snap the lever to the nearest end point (either min or max angle). If it is unchecked, the lever will always snap to the min angle position.
 - **Always Active:** If this is checked then the spring will always be active even when grabbing the lever.
-

Slider (VRTK_Slider)

extends [VRTK_Control](#)

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a horizontal or vertical slider. The direction can be freely set and auto-detection is supported.

The script will instantiate the required `Rigidbody` and `Interactable` components automatically in case they do not exist yet.

Inspector Parameters

- **Connected To:** An optional game object to which the wheel will be connected. If the game object moves the wheel will follow along.
- **Direction:** The axis on which the slider should move. All other axis will be frozen.
- **Minimum Limit:** The collider to specify the minimum limit of the slider.
- **Maximum Limit:** The collider to specify the maximum limit of the slider.
- **Minimum Value:** The minimum value of the slider.
- **Maximum Value:** The maximum value of the slider.
- **Step Size:** The increments in which slider values can change.
- **Snap To Step:** If this is checked then when the slider is released, it will snap to the nearest value

position.

- **Released Friction:** The amount of friction the slider will have when it is released.

Example

[VRTK/Examples/025_Controls_Overview](#) has a selection of sliders at various angles with different step values to demonstrate their usage.

Content Handler (VRTK_ContentHandler)

Overview

Manages objects defined as content. When taking out an object from a drawer and closing the drawer this object would otherwise disappear even if outside the drawer.

The script will use the boundaries of the control to determine if it is in or out and re-parent the object as necessary. It can be put onto individual objects or the parent of multiple objects. Using the latter way all interactable objects under that parent will become managed by the script.

Inspector Parameters

- **Control:** The 3D control responsible for the content.
- **Inside:** The transform containing the meshes or colliders that define the inside of the control.
- **Outside:** Any transform that will act as the parent while the object is not inside the control.

Example

[VRTK/Examples/025_Controls_Overview](#) has a drawer with a collection of items that adhere to this concept.

Utilities (VRTK/Scripts/Utilities)

A collection of scripts that provide useful functionality to aid the creation process.

- [SDK Manager](#)
- [SDK Setup Switcher](#)
- [SDK Setup](#)
- [SDK Info](#)
- [Device Finder](#)
- [Shared Methods](#)
- [Policy List](#)
- [Custom Raycast](#)

- [Adaptive Quality](#)
 - [Object Follow](#)
 - [Rigidbody Follow](#)
 - [Transform Follow](#)
 - [SDK Object Alias](#)
 - [SDK Transform Modify](#)
 - [Simulating Headset Movement](#)
-

SDK Manager (ScriptingDefineSymbolPredicateInfo)

Overview

A helper class that simply holds references to both the `ScriptingDefineSymbolPredicateAttribute` and the method info of the method the attribute is defined on.

Inspector Parameters

- **Auto Manage Script Defines:** Determines whether the scripting define symbols required by the installed SDKs are automatically added to and removed from the player settings.
- **Script Alias Left Controller:** A reference to the `GameObject` that contains any scripts that apply to the Left Hand Controller.
- **Script Alias Right Controller:** A reference to the `GameObject` that contains any scripts that apply to the Right Hand Controller.
- **Auto Manage VR Settings:** Determines whether the VR settings of the Player Settings are automatically adjusted to allow for all the used SDKs in the SDK Setups list below.
- **Auto Load Setup:** Determines whether the SDK Setups list below is used whenever the SDK Manager is enabled. The first loadable Setup is then loaded.
- **Setups:** The list of SDK Setups to choose from.

Class Variables

- `public readonly SDK_ScriptingDefineSymbolPredicateAttribute attribute` - The predicate attribute.
- `public readonly MethodInfo methodInfo` - The method info of the method the attribute is defined on.
- `public static ReadOnlyCollection<ScriptingDefineSymbolPredicateInfo> AvailableScriptingDefineSymbolPredicateInfos { get private set }` - All found scripting define symbol predicate attributes with associated method info.
- `public static readonly Dictionary<Type, Type> SDKFallbackTypesByBaseType` - Specifies the fallback SDK types for every base SDK type. Default: `new Dictionary<Type, Type>`
- `public static ReadOnlyCollection<VRTK_SDKInfo> AvailableSystemSDKInfos { get private set }` - All available system SDK infos.
- `public static ReadOnlyCollection<VRTK_SDKInfo> AvailableBoundariesSDKInfos { get private set }` - All available boundaries SDK infos.

- `private set }` - All available boundaries SDK infos.
- `public static ReadOnlyCollection<VRTK_SDKInfo> AvailableHeadsetSDKInfos { get private set }` - All available headset SDK infos.
- `public static ReadOnlyCollection<VRTK_SDKInfo> AvailableControllerSDKInfos { get private set }` - All available controller SDK infos.
- `public static ReadOnlyCollection<VRTK_SDKInfo> InstalledSystemSDKInfos { get private set }` - All installed system SDK infos. This is a subset of `AvailableSystemSDKInfos`. It contains only those available SDK infos for which an exists that uses the same symbol and whose associated method returns true.
- `public static ReadOnlyCollection<VRTK_SDKInfo> InstalledBoundariesSDKInfos { get private set }` - All installed boundaries SDK infos. This is a subset of `AvailableBoundariesSDKInfos`. It contains only those available SDK infos for which an exists that uses the same symbol and whose associated method returns true.
- `public static ReadOnlyCollection<VRTK_SDKInfo> InstalledHeadsetSDKInfos { get private set }` - All installed headset SDK infos. This is a subset of `AvailableHeadsetSDKInfos`. It contains only those available SDK infos for which an exists that uses the same symbol and whose associated method returns true.
- `public static ReadOnlyCollection<VRTK_SDKInfo> InstalledControllerSDKInfos { get private set }` - All installed controller SDK infos. This is a subset of `AvailableControllerSDKInfos`. It contains only those available SDK infos for which an exists that uses the same symbol and whose associated method returns true.
- `public static VRTK_SDKManager instance` - The singleton instance to access the SDK Manager variables from.
- `public List<SDK_ScriptingDefineSymbolPredicateAttribute> activeScriptingDefineSymbolsWithoutSDKClasses` - The active (i.e. to be added to the `ScriptingDefineSymbols`) scripting define symbol predicate attributes that have no associated SDK classes. Default: `new List<SDK_ScriptingDefineSymbolPredicateAttribute>()`
- `public VRTK_SDKSetup loadedSetup { get private set }` - The loaded SDK Setup. if no setup is currently loaded.
- `public ReadOnlyCollection<Behaviour> behavioursToToggleOnLoadedSetupChange { get private set }` - All behaviours that need toggling whenever changes.

Class Events

- `LoadedSetupChanged` - The event invoked whenever the loaded SDK Setup changes.

Unity Events

Adding the `VRTK_SDKManager_UnityEvents` component to `VRTK_SDKManager` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Event Payload

- `VRTK_SDKSetup previousSetup` - The previous loaded Setup. if no previous Setup was loaded.
- `VRTK_SDKSetup currentSetup` - The current loaded Setup. if no Setup is loaded anymore. See to check whether this is because of an error.

- `string errorMessage` - Explains why loading a list of Setups wasn't successful if is and an error occurred. if no error occurred.

Class Methods

ScriptingDefineSymbolPredicateInfo/2

```
public
ScriptingDefineSymbolPredicateInfo(SDK_ScriptingDefineSymbolPredicateAttribu
te attribute, MethodInfo methodInfo)
```

- Parameters
 - `SDK_ScriptingDefineSymbolPredicateAttribute attribute` - The predicate attribute.
 - `MethodInfo methodInfo` - The method info of the method the attribute is defined on.
- Returns
 - *none*

Event Payload. Constructs a new instance with the specified predicate attribute and associated method info.

ManageScriptingDefineSymbols/2

```
public bool ManageScriptingDefineSymbols(bool ignoreAutoManageScriptDefines,
bool ignoreIsActiveAndEnabled)
```

- Parameters
 - `bool ignoreAutoManageScriptDefines` - Whether to ignore while deciding to manage.
 - `bool ignoreIsActiveAndEnabled` - Whether to ignore while deciding to manage.
- Returns
 - `bool` - Whether the ' scripting define symbols were changed.

Manages (i.e. adds and removes) the scripting define symbols of the for the currently set SDK infos. This method is only available in the editor, so usage of the method needs to be surrounded by `#if UNITY_EDITOR` and `#endif` when used in a type that is also compiled for a standalone build.

ManageVRSettings/1

```
public void ManageVRSettings(bool force)
```

- Parameters
 - `bool force` - Whether to ignore while deciding to manage.
- Returns
 - *none*

Manages (i.e. adds and removes) the VR SDKs of the for the currently set SDK infos. This method is only available in the editor, so usage of the method needs to be surrounded by `#if UNITY_EDITOR` and `#endif` when used in a type that is also compiled for a standalone build.

AddBehaviourToToggleOnLoadedSetupChange/1

```
public void AddBehaviourToToggleOnLoadedSetupChange(Behaviour behaviour)
```

- Parameters

- Behaviour behaviour - The behaviour to add.

- Returns

- *none*

Adds a behaviour to the list of behaviours to toggle when changes.

RemoveBehaviourToToggleOnLoadedSetupChange/1

```
public void RemoveBehaviourToToggleOnLoadedSetupChange(Behaviour behaviour)
```

- Parameters

- Behaviour behaviour - The behaviour to remove.

- Returns

- *none*

Removes a behaviour of the list of behaviours to toggle when changes.

TryLoadSDKSetupFromList/1

```
public void TryLoadSDKSetupFromList(bool tryUseLastLoadedSetup = true)
```

- Parameters

- *none*

- Returns

- *none*

Tries to load a valid from .

TryLoadSDKSetup/3

```
public void TryLoadSDKSetup(int startIndex, bool tryToReinitialize, params  
VRTK_SDKSetup[] sdkSetups)
```

- Parameters

- `int startIndex` - The index of the to start the loading with.
- `bool tryToReinitialize` - Whether or not to retry initializing and using the currently set but unusable VR Device.
- `params VRTK_SDKSetup[] sdkSetups` - The list to try to load a from.

- Returns

- *none*

Tries to load a valid from a list. The first loadable in the list will be loaded. Will fall back to disable VR if none of the provided Setups is useable.

SetLoadedSDKSetupToPopulateObjectReferences/1

```
public void SetLoadedSDKSetupToPopulateObjectReferences(VRTK_SDKSetup setup)
```

- Parameters

- `VRTK_SDKSetup setup` - The SDK Setup to set as the loaded SDK.

- Returns

- *none*

Sets a given as the loaded SDK Setup to be able to use it when populating object references in the SDK Setup. This method should only be called when not playing as it's only for populating the object references. This method is only available in the editor, so usage of the method needs to be surrounded by `#if UNITY_EDITOR` and `#endif` when used in a type that is also compiled for a standalone build.

UnloadSDKSetup/1

```
public void UnloadSDKSetup(bool disableVR = false)
```

- Parameters

- `bool disableVR` - Whether to disable VR altogether after unloading the SDK Setup.

- Returns

- *none*

Unloads the currently loaded , if there is one.

SDK Setup Switcher (VRTK_SDKSetupSwitcher)

Overview

The SDK Setup Switcher adds a GUI overlay to allow switching the loaded `VRTK_SDKSetup` of the the

current `VRTK_SDKManager`.

Use the `SDKSetupSwitcher` prefab to use this.

SDK Setup (VRTK_SDKSetup)

Overview

The SDK Setup describes a list of SDKs and game objects to use.

Inspector Parameters

- **Auto Populate Object References:** Determines whether the SDK object references are automatically set to the objects of the selected SDKs. If this is true populating is done whenever the selected SDKs change.
- **Actual Boundaries:** A reference to the GameObject that is the user's boundary or play area, most likely provided by the SDK's Camera Rig.
- **Actual Headset:** A reference to the GameObject that contains the VR camera, most likely provided by the SDK's Camera Rig Headset.
- **Actual Left Controller:** A reference to the GameObject that contains the SDK Left Hand Controller.
- **Actual Right Controller:** A reference to the GameObject that contains the SDK Right Hand Controller.
- **Model Alias Left Controller:** A reference to the GameObject that models for the Left Hand Controller.
- **Model Alias Right Controller:** A reference to the GameObject that models for the Right Hand Controller.

Class Variables

- `public VRTK_SDKInfo systemSDKInfo` - The info of the SDK to use to deal with all system actions. By setting this to the fallback SDK will be used.
- `public VRTK_SDKInfo boundariesSDKInfo` - The info of the SDK to use to utilize room scale boundaries. By setting this to the fallback SDK will be used.
- `public VRTK_SDKInfo headsetSDKInfo` - The info of the SDK to use to utilize the VR headset. By setting this to the fallback SDK will be used.
- `public VRTK_SDKInfo controllerSDKInfo` - The info of the SDK to use to utilize the input devices. By setting this to the fallback SDK will be used.
- `public SDK_BaseSystem systemSDK` - The selected system SDK.
- `public SDK_BaseBoundaries boundariesSDK` - The selected boundaries SDK.
- `public SDK_BaseHeadset headsetSDK` - The selected headset SDK.
- `public SDK_BaseController controllerSDK` - The selected controller SDK.
- `public string[] usedVRDeviceNames` - The VR device names used by the currently selected SDKs.
- `public bool isValid` - Whether it's possible to use the Setup. See for more info.

Unity Events

Adding the `VRTK_SDKSetup_UnityEvents` component to `VRTK_SDKSetup` object allows access to `UnityEvents` that will react identically to the Class Events.

- All C# delegate events are mapped to a Unity Event with the `On` prefix. e.g. `MyEvent` -> `OnMyEvent`.

Class Methods

PopulateObjectReferences/1

```
public void PopulateObjectReferences(bool force)
```

- Parameters
 - `bool force` - Whether to ignore while deciding to populate.
- Returns
 - *none*

Populates the object references by using the currently set SDKs.

GetSimplifiedErrorDescriptions/0

```
public string[] GetSimplifiedErrorDescriptions()
```

- Parameters
 - *none*
- Returns
 - `string[]` - An array of all the error descriptions. Returns an empty array if no errors are found.

Checks the setup for errors and creates an array of error descriptions. The returned error descriptions handle the following cases for the current SDK infos: * Its type doesn't exist anymore. * It's a fallback SDK. * It doesn't have its scripting define symbols added. * It's missing its vendor SDK. Additionally the current SDK infos are checked whether they use multiple VR Devices.

SDK Info (VRTK_SDKInfo)

```
extends ISerializationCallbackReceiver
```

Overview

Holds all the info necessary to describe an SDK.

Class Variables

- `public Type type { get private set }` - The type of the SDK.
- `public string originalTypeNameWhenFallbackIsUsed { get private set }` - The name of the type of which this SDK info was created from. This is only used if said type wasn't found.
- `public SDK_DescriptionAttribute description { get private set }` - The description of the SDK.

Class Methods

ActualType>/0

```
public static VRTK_SDKInfo[] Create<BaseType, FallbackType, ActualType>()  
where BaseType : SDK_Base where FallbackType : BaseType where ActualType :  
BaseType
```

- Type Params
 - `FallbackType`, - The SDK base type. Must be a subclass of `.`
 - `FallbackType`, - The SDK type to fall back on if problems occur. Must be a subclass of `.`
 - `FallbackType`, - The SDK type to use. Must be a subclass of `.`
- Parameters
 - *none*
- Returns
 - `FallbackType`, - Multiple newly created instances.

Creates new SDK infos for a type that is known at compile time.

FallbackType>/1

```
public static VRTK_SDKInfo[] Create<BaseType, FallbackType>(Type actualType)  
where BaseType : SDK_Base where FallbackType : BaseType
```

- Type Params
 - `Create<BaseType`, - The SDK base type. Must be a subclass of `.`
 - `Create<BaseType`, - The SDK type to fall back on if problems occur. Must be a subclass of `.`
- Parameters
 - `Type actualType` - The SDK type to use. Must be a subclass of `.`
- Returns
 - `Create<BaseType`, - Multiple newly created instances.

Creates new SDK infos for a type.

VRTK_SDKInfo/1

```
public VRTK_SDKInfo(VRTK_SDKInfo infoToCopy)
```

- Parameters
 - `VRTK_SDKInfo infoToCopy` - The SDK info to copy.
- Returns
 - *none*

Creates a new SDK info by copying an existing one.

Device Finder (VRTK_DeviceFinder)

Overview

The Device Finder offers a collection of static methods that can be called to find common game devices such as the headset or controllers, or used to determine key information about the connected devices.

Class Variables

- `public enum Devices` - Possible devices.
 - `Headset` - The headset.
 - `LeftController` - The left hand controller.
 - `RightController` - The right hand controller.
- `public enum Headsets` - Possible headsets
 - `Unknown` - An unknown headset.
 - `OculusRift` - A summary of all Oculus Rift headset versions.
 - `OculusRiftCV1` - A specific version of the Oculus Rift headset, the Consumer Version 1.
 - `Vive` - A summary of all HTC Vive headset versions.
 - `ViveMV` - A specific version of the HTC Vive headset, the first consumer version.

Class Methods

GetCurrentControllerType/0

```
public static SDK_BaseController.ControllerType GetCurrentControllerType()
```

- Parameters
 - *none*
- Returns
 - `SDK_BaseController.ControllerType` - The `ControllerType` based on the SDK and headset being used.

The `GetCurrentControllerType` method returns the current used `ControllerType` based on the SDK

and headset being used.

GetControllerIndex/1

```
public static uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The controller object to get the index of a controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method is used to find the index of a given controller object.

GetControllerByIndex/2

```
public static GameObject GetControllerByIndex(uint index, bool getActual)
```

- Parameters

- `uint index` - The index of the actual controller to find.
- `bool getActual` - An optional parameter that if true will return the game object that the SDK controller is attached to.

- Returns

- `GameObject` - The actual controller `GameObject` that matches the given index.

The `GetControllerByIndex` method is used to find a controller based on its unique index.

GetControllerOrigin/1

```
public static Transform GetControllerOrigin(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the origin for.

- Returns

- `Transform` - The transform of the controller origin or if an origin is not set then the transform parent.

The `GetControllerOrigin` method is used to find the controller's origin.

DeviceTransform/1

```
public static Transform DeviceTransform(Devices device)
```

- Parameters
 - `Devices device` - The Devices enum to get the transform for.
- Returns
 - `Transform` - The transform for the given Devices enum.

The `DeviceTransform` method returns the transform for a given Devices enum.

GetControllerHandType/1

```
public static SDK_BaseController.ControllerHand GetControllerHandType(string hand)
```

- Parameters
 - `string hand` - The string representation of the hand to retrieve the type of. `left` or `right`.
- Returns
 - `SDK_BaseController.ControllerHand` - A ControllerHand representing either the Left or Right hand.

The `GetControllerHandType` method is used for getting the enum representation of ControllerHand from a given string.

GetControllerHand/1

```
public static SDK_BaseController.ControllerHand GetControllerHand(GameObject controller)
```

- Parameters
 - `GameObject controller` - The controller game object to check the hand of.
- Returns
 - `SDK_BaseController.ControllerHand` - A ControllerHand representing either the Left or Right hand.

The `GetControllerHand` method is used for getting the enum representation of ControllerHand for the given controller game object.

GetControllerLeftHand/1

```
public static GameObject GetControllerLeftHand(bool getActual = false)
```

- Parameters
 - `bool getActual` - An optional parameter that if true will return the game object that the SDK controller is attached to.
- Returns

- `GameObject` - The left hand controller.

The `GetControllerLeftHand` method retrieves the game object for the left hand controller.

GetControllerRightHand/1

```
public static GameObject GetControllerRightHand(bool getActual = false)
```

- **Parameters**

- `bool getActual` - An optional parameter that if true will return the game object that the SDK controller is attached to.

- **Returns**

- `GameObject` - The right hand controller.

The `GetControllerRightHand` method retrieves the game object for the right hand controller.

IsControllerOfHand/2

```
public static bool IsControllerOfHand(GameObject checkController,  
SDK_BaseController.ControllerHand hand)
```

- **Parameters**

- `GameObject checkController` - The actual controller object that is being checked.
- `SDK_BaseController.ControllerHand hand` - The representation of a hand to check if the given controller matches.

- **Returns**

- `bool` - Is true if the given controller matches the given hand.

The `IsControllerOfHand` method is used to check if a given controller game object is of the hand type provided.

IsControllerLeftHand/1

```
public static bool IsControllerLeftHand(GameObject checkController)
```

- **Parameters**

- `GameObject checkController` - The controller object that is being checked.

- **Returns**

- `bool` - Is true if the given controller is the left controller.

The `IsControllerLeftHand` method is used to check if a given controller game object is the left handed controller.

IsControllerRightHand/1

```
public static bool IsControllerRightHand(GameObject checkController)
```

- Parameters

- `GameObject checkController` - The controller object that is being checked.

- Returns

- `bool` - Is true if the given controller is the right controller.

The `IsControllerRightHand` method is used to check if a given controller game object is the right handed controller.

GetActualController/1

```
public static GameObject GetActualController(GameObject givenController)
```

- Parameters

- `GameObject givenController` - The `GameObject` of the controller.

- Returns

- `GameObject` - The `GameObject` that is the actual controller.

The `GetActualController` method will attempt to get the actual SDK controller object.

GetScriptAliasController/1

```
public static GameObject GetScriptAliasController(GameObject givenController)
```

- Parameters

- `GameObject givenController` - The `GameObject` of the controller.

- Returns

- `GameObject` - The `GameObject` that is the alias controller containing the scripts.

The `GetScriptAliasController` method will attempt to get the object that contains the scripts for the controller.

GetModelAliasController/1

```
public static GameObject GetModelAliasController(GameObject givenController)
```

- Parameters

- `GameObject givenController` - The `GameObject` of the controller.

- Returns

- `GameObject` - The `GameObject` that is the alias controller containing the controller model.

The `GetModelAliasController` method will attempt to get the object that contains the model for the controller.

GetModelAliasControllerHand/1

```
public static SDK_BaseController.ControllerHand  
GetModelAliasControllerHand(GameObject givenObject)
```

- **Parameters**

- `GameObject givenObject` - The `GameObject` that may represent a model alias.

- **Returns**

- `SDK_BaseController.ControllerHand` - The enum of the `ControllerHand` that the given `GameObject` may represent.

The `GetModelAliasControllerHand` method will return the hand that the given model alias `GameObject` is for.

GetControllerVelocity/1

```
public static Vector3 GetControllerVelocity(VRTK_ControllerReference  
controllerReference)
```

- **Parameters**

- `VRTK_ControllerReference controllerReference` - The reference to the controller.

- **Returns**

- `Vector3` - A 3 dimensional vector containing the current real world physical controller velocity.

The `GetControllerVelocity` method is used for getting the current velocity of the physical game controller. This can be useful to determine the speed at which the controller is being swung or the direction it is being moved in.

GetControllerAngularVelocity/1

```
public static Vector3 GetControllerAngularVelocity(VRTK_ControllerReference  
controllerReference)
```

- **Parameters**

- `VRTK_ControllerReference controllerReference` - The reference to the controller.

- **Returns**

- `Vector3` - A 3 dimensional vector containing the current real world physical controller angular (rotational) velocity.

The `GetControllerAngularVelocity` method is used for getting the current rotational velocity of the physical game controller. This can be useful for determining which way the controller is being rotated and at what speed the rotation is occurring.

GetHeadsetVelocity/0

```
public static Vector3 GetHeadsetVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` containing the current velocity of the headset.

The `GetHeadsetVelocity` method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public static Vector3 GetHeadsetAngularVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

HeadsetTransform/0

```
public static Transform HeadsetTransform()
```

- Parameters
 - *none*
- Returns
 - `Transform` - The transform of the VR Headset component.

The `HeadsetTransform` method is used to retrieve the transform for the VR Headset in the scene. It can be useful to determine the position of the user's head in the game world.

HeadsetCamera/0

```
public static Transform HeadsetCamera()
```

- Parameters
 - *none*
- Returns
 - `Transform` - The transform of the VR Camera component.

The `HeadsetCamera` method is used to retrieve the transform for the VR Camera in the scene.

ResetHeadsetTypeCache/0

```
public static void ResetHeadsetTypeCache()
```

- Parameters
 - *none*
- Returns
 - *none*

The `ResetHeadsetTypeCache` resets the cache holding the current headset type value.

GetHeadsetType/1

```
public static Headsets GetHeadsetType(bool summary = false)
```

- Parameters
 - `bool summary` - If this is true, then the generic name for the headset is returned not including the version type (e.g. `OculusRift` will be returned for `DK2` and `CV1`).
- Returns
 - `Headsets` - The Headset type that is connected.

The `GetHeadsetType` method returns the type of headset connected to the computer.

PlayAreaTransform/0

```
public static Transform PlayAreaTransform()
```

- Parameters
 - *none*
- Returns
 - `Transform` - The transform of the VR Play Area component.

The `PlayAreaTransform` method is used to retrieve the transform for the play area in the scene.

Shared Methods (VRTK_SharedMethods)

Overview

The Shared Methods script is a collection of reusable static methods that are used across a range of different scripts.

Class Methods

GetBounds/3

```
public static Bounds GetBounds(Transform transform, Transform  
excludeRotation = null, Transform excludeTransform = null)
```

- Parameters

- Transform transform -
- Transform excludeRotation - Resets the rotation of the transform temporarily to 0 to eliminate skewed bounds.
- Transform excludeTransform - Does not consider the stated object when calculating the bounds.

- Returns

- Bounds - The bounds of the transform.

The GetBounds methods returns the bounds of the transform including all children in world space.

IsLowest/2

```
public static bool IsLowest(float value, float[] others)
```

- Parameters

- float value - The value to check to see if it is lowest.
- float[] others - The array of values to check against.

- Returns

- bool - Returns true if the value is lower than all numbers in the given array, returns false if it is not the lowest.

The IsLowest method checks to see if the given value is the lowest number in the given array of values.

AddCameraFade/0

```
public static Transform AddCameraFade()
```


- Parameters
 - *none*
- Returns
 - `Transform` - The transform of the headset camera.

The `AddCameraFade` method finds the headset camera and adds a headset fade script to it.

CreateColliders/1

```
public static void CreateColliders(GameObject obj)
```

- Parameters
 - `GameObject obj` - The game object to attempt to add the colliders to.
- Returns
 - *none*

The `CreateColliders` method attempts to add box colliders to all child objects in the given object that have a renderer but no collider.

CloneComponent/3

```
public static Component CloneComponent(Component source, GameObject destination, bool copyProperties = false)
```

- Parameters
 - `Component source` - The component to copy.
 - `GameObject destination` - The game object to copy the component to.
 - `bool copyProperties` - Determines whether the properties of the component as well as the fields should be copied.
- Returns
 - `Component` - The component that has been cloned onto the given game object.

The `CloneComponent` method takes a source component and copies it to the given destination game object.

ColorDarken/2

```
public static Color ColorDarken(Color color, float percent)
```

- Parameters
 - `Color color` - The source colour to apply the darken to.
 - `float percent` - The percent to darken the colour by.
- Returns

- `Color` - The new colour with the darken applied.

The `ColorDarken` method takes a given colour and darkens it by the given percentage.

RoundFloat/3

```
public static float RoundFloat(float givenFloat, int decimalPlaces, bool rawFidelity = false)
```

- Parameters

- `float givenFloat` - The float to round.
- `int decimalPlaces` - The number of decimal places to round to.
- `bool rawFidelity` - If this is true then the decimal places must be given in the decimal multiplier, e.g. 10 for 1dp, 100 for 2dp, etc.

- Returns

- `float` - The rounded float.

The `RoundFloat` method is used to round a given float to the given decimal places.

IsEditTime/0

```
public static bool IsEditTime()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if Unity is in the Unity Editor and not in play mode.

The `IsEditTime` method determines if the state of Unity is in the Unity Editor and the scene is not in play mode.

Mod/2

```
public static float Mod(float a, float b)
```

- Parameters

- `float a` - The dividend value.
- `float b` - The divisor value.

- Returns

- `float` - The remainder value.

The `Mod` method is used to find the remainder of the sum a/b .

FindEvenInactiveGameObject/1

```
public static GameObject FindEvenInactiveGameObject<T>(string gameObjectName  
= null) where T : Component
```

- Type Params
 - `GameObject` - The component type that needs to be on an ancestor of the wanted . Must be a subclass of .
- Parameters
 - `string gameObjectName` - The name of the wanted . If it contains a '/' character, this method traverses the hierarchy like a path name, beginning on the game object that has a component of type .
- Returns
 - `GameObject` - The with name and an ancestor that has a . If no such is found is returned.

Finds the first with a given name and an ancestor that has a specific component. This method returns active as well as inactive s in the scene. It doesn't return assets. For performance reasons it is recommended to not use this function every frame. Cache the result in a member variable at startup instead.

FindEvenInactiveComponents/0

```
public static T[] FindEvenInactiveComponents<T>() where T : Component
```

- Type Params
 - `T[]` - The component type to search for. Must be a subclass of .
- Parameters
 - *none*
- Returns
 - `T[]` - All the found components. If no component is found an empty array is returned.

Finds all components of a given type. This method returns components from active as well as inactive s in the scene. It doesn't return assets. For performance reasons it is recommended to not use this function every frame. Cache the result in a member variable at startup instead.

FindEvenInactiveComponent/0

```
public static T FindEvenInactiveComponent<T>() where T : Component
```

- Type Params
 - `T` - The component type to search for. Must be a subclass of .
- Parameters

- *none*
- Returns
 - `T` - The found component. If no component is found is returned.

Finds the first component of a given type. This method returns components from active as well as inactive s in the scene. It doesn't return assets. For performance reasons it is recommended to not use this function every frame. Cache the result in a member variable at startup instead.

GenerateVRTKObjectName/2

```
public static string GenerateVRTKObjectName(bool autoGen, params object[]
replacements)
```

- Parameters
 - `bool autoGen` - An additiona [AUTOGEN] prefix will be added if this is true.
 - `params object[] replacements` - A collection of parameters to add to the generated name.
- Returns
 - `string` - The generated name string.

The `GenerateVRTKObjectName` method is used to create a standard name string for any VRTK generated object.

GetGPUPTimeLastFrame/0

```
public static float GetGPUPTimeLastFrame()
```

- Parameters
 - *none*
- Returns
 - `float` - The total GPU time utilized last frame as measured by the VR subsystem.

The `GetGPUPTimeLastFrame` retrieves the time spent by the GPU last frame, in seconds, as reported by the VR SDK.

Vector2ShallowCompare/3

```
public static bool Vector2ShallowCompare(Vector2 vectorA, Vector2 vectorB,
int compareFidelity)
```

- Parameters
 - `Vector2 vectorA` - The Vector2 to compare against.
 - `Vector2 vectorB` - The Vector2 to compare with
 - `int compareFidelity` - The number of decimal places to use when doing the comparison on

the float elements within the Vector2.

- Returns
 - `bool` - Returns true if the given Vector2 objects match based on the given fidelity.

The `Vector2ShallowCompare` method compares two given Vector2 objects based on the given fidelity, which is the equivalent of comparing rounded Vector2 elements to determine if the Vector2 elements are equal.

NumberPercent/2

```
public static float NumberPercent(float value, float percent)
```

- Parameters
 - `float value` - The value to determine the percentage from
 - `float percent` - The percentage to find within the given value.
- Returns
 - `float` - A float containing the percentage value based on the given input.

The `NumberPercent` method is used to determine the percentage of a given value.

SetGlobalScale/2

```
public static void SetGlobalScale(this Transform transform, Vector3 globalScale)
```

- Parameters
 - `this Transform transform` - The reference to the transform to scale.
 - `Vector3 globalScale` - A Vector3 of a global scale to apply to the given transform.
- Returns
 - *none*

The `SetGlobalScale` method is used to set a transform scale based on a global scale instead of a local scale.

GetTypeUnknownAssembly/1

```
public static Type GetTypeUnknownAssembly(string typeName)
```

- Parameters
 - `string typeName` - The name of the type to get.
- Returns
 - `Type` - The Type, or null if none is found.

The `GetTypeUnknownAssembly` method is used to find a `Type` without knowing the exact assembly it is in.

Policy List (VRTK_PolicyList)

Overview

The Policy List allows to create a list of either tag names, script names or layer names that can be checked against to see if another operation is permitted.

A number of other scripts can use a Policy List to determine if an operation is permitted based on whether a game object has a tag applied, a script component on it or whether it's on a given layer.

For example, the Teleporter scripts can ignore game object targets as a teleport location if the game object contains a tag that is in the identifiers list and the policy is set to ignore.

Or the teleporter can only allow teleport to targets that contain a tag that is in the identifiers list and the policy is set to include.

Add the Policy List script to a game object (preferably the same component utilising the list) and then configure the list accordingly.

Then in the component that has a Policy List paramter (e.g. BasicTeleporter has `Target List Policy`) simply select the list that has been created and defined.

Inspector Parameters

- **Operation:** The operation to apply on the list of identifiers.
- **Check Type:** The element type on the game object to check against.
- **Identifiers:** A list of identifiers to check for against the given check type (either tag or script).

Class Variables

- `public enum OperationTypes` - The operation to apply on the list of identifiers.
 - `Ignore` - Will ignore any game objects that contain either a tag or script component that is included in the identifiers list.
 - `Include` - Will only include game objects that contain either a tag or script component that is included in the identifiers list.
- `public enum CheckTypes` - The types of element that can be checked against.
 - `Tag` - The tag applied to the game object.
 - `Script` - A script component added to the game object.
 - `Layer` - A layer applied to the game object.

Class Methods

Find/1

```
public virtual bool Find(GameObject obj)
```

- Parameters

- `GameObject obj` - The game object to check if it has a tag or script that is listed in the identifiers list.

- Returns

- `bool` - If the operation is `Ignore` and the game object is matched by an identifier from the list then it returns true. If the operation is `Include` and the game object is not matched by an identifier from the list then it returns true.

The Find method performs the set operation to determine if the given game object contains one of the identifiers on the set check type. For instance, if the Operation is `Ignore` and the Check Type is `Tag` then the Find method will attempt to see if the given game object has a tag that matches one of the identifiers.

Check/2

```
public static bool Check(GameObject obj, VRTK_PolicyList list)
```

- Parameters

- `GameObject obj` - The game object to check.
- `VRTK_PolicyList list` - The policy list to use for checking.

- Returns

- `bool` - Returns true if the given game object matches the policy list or given string logic.

The Check method is used to check if a game object should be ignored based on a given string or policy list.

Custom Raycast (VRTK_CustomRaycast)

Overview

A Custom Raycast allows to specify custom options for a `Physics.Raycast`.

A number of other scripts can utilise a Custom Raycast to further customise the raycasts that the scripts use internally.

For example, the `VRTK_BodyPhysics` script can be set to ignore trigger colliders when casting to see if it should teleport up or down to the nearest floor.

Inspector Parameters

- **Layers To Ignore:** The layers to ignore when raycasting.
- **Trigger Interaction:** Determines whether the ray will interact with trigger colliders.

Class Methods

Raycast/6

```
public static bool Raycast(VRTK_CustomRaycast customCast, Ray ray, out RaycastHit hitData, LayerMask ignoreLayers, float length = Mathf.Infinity, QueryTriggerInteraction affectTriggers = QueryTriggerInteraction.UseGlobal)
```

- **Parameters**

- `VRTK_CustomRaycast customCast` - The optional object with customised cast parameters.
- `Ray ray` - The Ray to cast with.
- `out RaycastHit hitData` - The raycast hit data.
- `LayerMask ignoreLayers` - A layermask of layers to ignore from the raycast.
- `float length` - The maximum length of the raycast.
- `QueryTriggerInteraction affectTriggers` - Determines the trigger interaction level of the cast.

- **Returns**

- `bool` - Returns true if the raycast successfully collides with a valid object.

The Raycast method is used to generate a raycast either from the given CustomRaycast object or a default Physics.Raycast.

Linecast/6

```
public static bool Linecast(VRTK_CustomRaycast customCast, Vector3 startPosition, Vector3 endPosition, out RaycastHit hitData, LayerMask ignoreLayers, QueryTriggerInteraction affectTriggers = QueryTriggerInteraction.UseGlobal)
```

- **Parameters**

- `VRTK_CustomRaycast customCast` - The optional object with customised cast parameters.
- `Vector3 startPosition` - The world position to start the linecast from.
- `Vector3 endPosition` - The world position to end the linecast at.
- `out RaycastHit hitData` - The linecast hit data.
- `LayerMask ignoreLayers` - A layermask of layers to ignore from the linecast.
- `QueryTriggerInteraction affectTriggers` - Determines the trigger interaction level of the cast.

- **Returns**

- `bool` - Returns true if the linecast successfully collides with a valid object.

The Linecast method is used to generate a linecast either from the given CustomRaycast object or a default Physics.Linecast.

CapsuleCast/9

```
public static bool CapsuleCast(VRTK_CustomRaycast customCast, Vector3 point1,
    Vector3 point2, float radius, Vector3 direction, float maxDistance, out
    RaycastHit hitData, LayerMask ignoreLayers, QueryTriggerInteraction
    affectTriggers = QueryTriggerInteraction.UseGlobal)
```

• Parameters

- `VRTK_CustomRaycast customCast` - The optional object with customised cast parameters.
- `Vector3 point1` - The center of the sphere at the start of the capsule.
- `Vector3 point2` - The center of the sphere at the end of the capsule.
- `float radius` - The radius of the capsule.
- `Vector3 direction` - The direction into which to sweep the capsule.
- `float maxDistance` - The max length of the sweep.
- `out RaycastHit hitData` - The linecast hit data.
- `LayerMask ignoreLayers` - A layermask of layers to ignore from the linecast.
- `QueryTriggerInteraction affectTriggers` - Determines the trigger interaction level of the cast.

• Returns

- `bool` - Returns true if the linecast successfully collides with a valid object.

The CapsuleCast method is used to generate a linecast either from the given CustomRaycast object or a default Physics.Linecast.

CustomRaycast/3

```
public virtual bool CustomRaycast(Ray ray, out RaycastHit hitData, float
    length = Mathf.Infinity)
```

• Parameters

- `Ray ray` - The Ray to cast with.
- `out RaycastHit hitData` - The raycast hit data.
- `float length` - The maximum length of the raycast.

• Returns

- `bool` - Returns true if the raycast successfully collides with a valid object.

The CustomRaycast method is used to generate a raycast based on the options defined in the CustomRaycast object.

CustomLinecast/3

```
public virtual bool CustomLinecast(Vector3 startPosition, Vector3
endPosition, out RaycastHit hitData)
```

- Parameters

- Vector3 startPosition - The world position to start the linecast from.
- Vector3 endPosition - The world position to end the linecast at.
- out RaycastHit hitData - The linecast hit data.

- Returns

- bool - Returns true if the line successfully collides with a valid object.

The CustomLinecast method is used to generate a linecast based on the options defined in the CustomRaycast object.

CustomCapsuleCast/6

```
public virtual bool CustomCapsuleCast(Vector3 point1, Vector3 point2, float
radius, Vector3 direction, float maxDistance, out RaycastHit hitData)
```

- Parameters

- Vector3 point1 - The center of the sphere at the start of the capsule.
- Vector3 point2 - The center of the sphere at the end of the capsule.
- float radius - The radius of the capsule.
- Vector3 direction - The direction into which to sweep the capsule.
- float maxDistance - The max length of the sweep.
- out RaycastHit hitData - The capsulecast hit data.

- Returns

- bool - Returns true if the capsule successfully collides with a valid object.

The CustomCapsuleCast method is used to generate a capsulecast based on the options defined in the CustomRaycast object.

Adaptive Quality (VRTK_AdaptiveQuality)

Overview

Adaptive Quality dynamically changes rendering settings to maintain VR framerate while maximizing GPU utilization.

Only Compatible With Unity 5.4 and above

There are two goals:

- Reduce the chances of dropping frames and reprojection
- Increase quality when there are idle GPU cycles

This script currently changes the following to reach these goals:

- Rendering resolution and viewport size (aka Dynamic Resolution)

In the future it could be changed to also change the following:

- MSAA level
- Fixed Foveated Rendering
- Radial Density Masking
- (Non-fixed) Foveated Rendering (once HMDs support eye tracking)

Some shaders, especially Image Effects, need to be modified to work with the changed render scale. To fix them pass `1.0f / VRSettings.renderViewportScale` into the shader and scale all incoming UV values with it in the vertex program. Do this by using `Material.SetFloat` to set the value in the script that configures the shader.

In more detail:

- In the `.shader` file: Add a new runtime-set property value `float _InverseOfRenderViewportScale` and add `vertexInput.texcoord *= _InverseOfRenderViewportScale` to the start of the vertex program
- In the `.cs` file: Before using the material (eg. `Graphics.Blit`) add

```
material.SetFloat("_InverseOfRenderViewportScale", 1.0f /  
VRSettings.renderViewportScale)
```

Inspector Parameters

- **Draw Debug Visualization:** Toggles whether to show the debug overlay. Each square represents a different level on the quality scale. Levels increase from left to right, the first green box that is lit above represents the recommended render target resolution provided by the current `VRDevice`, the box that is lit below in cyan represents the current resolution and the filled box represents the current viewport scale. The yellow boxes represent resolutions below the recommended render target resolution. The currently lit box becomes red whenever the user is likely seeing reprojection in the HMD since the application isn't maintaining VR framerate. If lit, the box all the way on the left is almost always lit red because it represents the lowest render scale with reprojection on.
- **Allow Keyboard Shortcuts:** Toggles whether to allow keyboard shortcuts to control this script.
 - The supported shortcuts are:
 - `Shift+F1`: Toggle debug visualization on/off
 - `Shift+F2`: Toggle usage of override render scale on/off
 - `Shift+F3`: Decrease override render scale level

- `Shift+F4`: Increase override render scale level
- **Allow Command Line Arguments:** Toggles whether to allow command line arguments to control this script at startup of the standalone build.
 - The supported command line arguments all begin with '-' and are:
 - `-noaq`: Disable adaptive quality
 - `-aqminscale X`: Set minimum render scale to X
 - `-aqmaxscale X`: Set maximum render scale to X
 - `-aqmaxres X`: Set maximum render target dimension to X
 - `-aqfillratestep X`: Set render scale fill rate step size in percent to X (X from 1 to 100)
 - `-aqoverride X`: Set override render scale level to X
 - `-vrdebug`: Enable debug visualization
 - `-msaa X`: Set MSAA level to X
- **Msaa Level:** The MSAA level to use.
- **Scale Render Viewport:** Toggles whether the render viewport scale is dynamically adjusted to maintain VR framerate. If unchecked, the renderer will render at the recommended resolution provided by the current `VRDevice`.
- **Minimum Render Scale:** The minimum allowed render scale.
- **Maximum Render Scale:** The maximum allowed render scale.
- **Maximum Render Target Dimension:** The maximum allowed render target dimension. This puts an upper limit on the size of the render target regardless of the maximum render scale.
- **Render Scale Fill Rate Step Size In Percent:** The fill rate step size in percent by which the render scale levels will be calculated.
- **Scale Render Target Resolution:** Toggles whether the render target resolution is dynamically adjusted to maintain VR framerate. If unchecked, the renderer will use the maximum target resolution specified by `maximumRenderScale`.
- **Override Render Viewport Scale:** Toggles whether to override the used render viewport scale level.
- **Override Render Viewport Scale Level:** The render viewport scale level to override the current one with.

Class Variables

- `public readonly ReadOnlyCollection<float> renderScales` - All the calculated render scales. The elements of this collection are to be interpreted as modifiers to the recommended render target resolution provided by the current `VRDevice`.
- `public static float CurrentRenderScale` - The current render scale. A render scale of 1.0 represents the recommended render target resolution provided by the current `VRDevice`.
- `public Vector2 defaultRenderTargetResolution` - The recommended render target resolution provided by the current `VRDevice`.
- `public Vector2 currentRenderTargetResolution` - The current render target resolution.

Class Methods

RenderTargetResolutionForRenderScale/1

```
public static Vector2 RenderTargetResolutionForRenderScale(float renderScale)
```

- Parameters

- `float renderScale` - The render scale to calculate the render target resolution with.

- Returns

- `Vector2` - The render target resolution for `renderScale`.

Calculates and returns the render target resolution for a given render scale.

BiggestAllowedMaximumRenderScale/0

```
public float BiggestAllowedMaximumRenderScale()
```

- Parameters

- *none*

- Returns

- `float` - The biggest allowed maximum render scale.

Calculates and returns the biggest allowed maximum render scale to be used for `maximumRenderScale` given the current `maximumRenderTargetDimension`.

ToString/0

```
public override string ToString()
```

- Parameters

- *none*

- Returns

- `string` - The summary.

A summary of this script by listing all the calculated render scales with their corresponding render target resolution.

Example

`VRTK/Examples/039_CameraRig_AdaptiveQuality` displays the frames per second in the centre of the headset view. The debug visualization of this script is displayed near the top edge of the headset view. Pressing the trigger generates a new sphere and pressing the touchpad generates ten new spheres. Eventually when lots of spheres are present the FPS will drop and demonstrate the script.

Object Follow (VRTK_ObjectFollow)

Overview

Abstract class that allows to change one game object's properties to follow another game object.

Inspector Parameters

- **Game Object To Follow:** The game object to follow. The followed property values will be taken from this one.
- **Game Object To Change:** The game object to change the property values of. If left empty the game object this script is attached to will be changed.
- **Follows Position:** Whether to follow the position of the given game object.
- **Smooths Position:** Whether to smooth the position when following `gameObjectToFollow`.
- **Max Allowed Per Frame Distance Difference:** The maximum allowed distance between the unsmoothed source and the smoothed target per frame to use for smoothing.
- **Follows Rotation:** Whether to follow the rotation of the given game object.
- **Smooths Rotation:** Whether to smooth the rotation when following `gameObjectToFollow`.
- **Max Allowed Per Frame Angle Difference:** The maximum allowed angle between the unsmoothed source and the smoothed target per frame to use for smoothing.
- **Follows Scale:** Whether to follow the scale of the given game object.
- **Smooths Scale:** Whether to smooth the scale when following `gameObjectToFollow`.
- **Max Allowed Per Frame Size Difference:** The maximum allowed size between the unsmoothed source and the smoothed target per frame to use for smoothing.

Class Variables

- `public Vector3 targetPosition { get private set }` - The position that results by following `gameObjectToFollow`.
- `public Quaternion targetRotation { get private set }` - The rotation that results by following `gameObjectToFollow`.
- `public Vector3 targetScale { get private set }` - The scale that results by following `gameObjectToFollow`.

Class Methods

Follow/0

```
public virtual void Follow()
```

- Parameters
 - *none*
- Returns
 - *none*

Follow `gameObjectToFollow` using the current settings.

Rigidbody Follow (VRTK_RigidbodyFollow)

extends [VRTK_ObjectFollow](#)

Overview

Changes one game object's rigidbody to follow another game object's rigidbody.

Inspector Parameters

- **Movement Option:** Specifies how to position and rotate the rigidbody.

Class Variables

- `public enum MovementOption` - Specifies how to position and rotate the rigidbody.
 - `Set` - Use and `.`
 - `Move` - Use and `.`
 - `Add` - Use and `.`
-

Transform Follow (VRTK_TransformFollow)

extends [VRTK_ObjectFollow](#)

Overview

Changes one game object's transform to follow another game object's transform.

Class Variables

- `public enum FollowMoment` - The moment at which to follow.
 - `OnUpdate` - Follow in the Update method.
 - `OnLateUpdate` - Follow in the LateUpdate method.
 - `OnPreRender` - Follow in the OnPreRender method. (This script doesn't have to be attached to a camera.)
 - `OnPreCull` - Follow in the OnPreCull method. (This script doesn't have to be attached to a camera.)
-

SDK Object Alias (VRTK_SDKObjectAlias)

Overview

The GameObject that the SDK Object Alias script is applied to will become a child of the selected SDK Object.

Inspector Parameters

- **Sdk Object:** The specific SDK Object to child this GameObject to.

Class Variables

- `public enum SDKObject` - Valid SDK Objects
 - `Boundary` - The main camera rig/play area object that defines the player boundary.
 - `Headset` - The main headset camera defines the player head.
-

SDK Transform Modify (VRTK_SDKTransformModify)

Overview

The SDK Transform Modify can be used to change a transform orientation at runtime based on the currently used SDK or SDK controller.

Inspector Parameters

- **Loaded SDK Setup:** An optional SDK Setup to use to determine when to modify the transform.
 - **Controller Type:** An optional SDK controller type to use to determine when to modify the transform.
 - **Position:** The new local position to change the transform to.
 - **Rotation:** The new local rotation in euler angles to change the transform to.
 - **Scale:** The new local scale to change the transform to.
 - **Target:** The target transform to modify on enable. If this is left blank then the transform the script is attached to will be used.
 - **Sdk Overrides:** A collection of SDK Transform overrides to change the given target transform for each specified SDK.
-

Simulating Headset Movement (VRTK_Simulator)

Overview

To test a scene it is often necessary to use the headset to move to a location. This increases turn-around times and can become cumbersome.

The simulator allows navigating through the scene using the keyboard instead, without the need to put on the headset. One can then move around (also through walls) while looking at the monitor and still use the controllers to interact.

Supported movements are: forward, backward, strafe left, strafe right, turn left, turn right, up, down.

Inspector Parameters

- **Keys:** Per default the keys on the left-hand side of the keyboard are used (WASD). They can be individually set as needed. The reset key brings the camera to its initial location.
 - **Only In Editor:** Typically the simulator should be turned off when not testing anymore. This option will do this automatically when outside the editor.
 - **Step Size:** Depending on the scale of the world the step size can be defined to increase or decrease movement speed.
 - **Cam Start:** An optional game object marking the position and rotation at which the camera should be initially placed.
-

Base SDK (VRTK/SDK/Base)

The base scripts used to determine the interface for interacting with a Unity VR SDK.

- [SDK Base](#)
 - [SDK Description](#)
 - [SDK Scripting Define Symbol Predicate](#)
 - [Base System](#)
 - [Base Headset](#)
 - [Base Controller](#)
 - [Base Boundaries](#)
-

SDK Base (SDK_Base)

extends ScriptableObject

Overview

Abstract superclass that defines that a particular class is an SDK.

This is an abstract class to mark all different SDK endpoints with. This is used to allow for type safety when talking about 'an SDK' instead of one of the different endpoints (System, Boundaries,

Headset, Controller).

Class Methods

OnBeforeSetupLoad/1

```
public virtual void OnBeforeSetupLoad(VRTK_SDKSetup setup)
```

- Parameters
 - `VRTK_SDKSetup setup` - The SDK Setup which is using this SDK.
- Returns
 - *none*

This method is called just before loading the that's using this SDK.

OnAfterSetupLoad/1

```
public virtual void OnAfterSetupLoad(VRTK_SDKSetup setup)
```

- Parameters
 - `VRTK_SDKSetup setup` - The SDK Setup which is using this SDK.
- Returns
 - *none*

This method is called just after loading the that's using this SDK.

OnBeforeSetupUnload/1

```
public virtual void OnBeforeSetupUnload(VRTK_SDKSetup setup)
```

- Parameters
 - `VRTK_SDKSetup setup` - The SDK Setup which is using this SDK.
- Returns
 - *none*

This method is called just before unloading the that's using this SDK.

OnAfterSetupUnload/1

```
public virtual void OnAfterSetupUnload(VRTK_SDKSetup setup)
```

- Parameters
 - `VRTK_SDKSetup setup` - The SDK Setup which is using this SDK.

- Returns
 - *none*

This method is called just after unloading the that's using this SDK.

SDK Description (SDK_DescriptionAttribute)

extends Attribute

Overview

Describes a class that represents an SDK. Only allowed on classes that inherit from

.

Class Variables

- `public readonly string prettyName` - The pretty name of the SDK. Uniquely identifies the SDK.
- `public readonly string symbol` - The scripting define symbol needed for the SDK. Needs to be the same as to add and remove the scripting define symbol automatically using .
- `public readonly string vrDeviceName` - The name of the VR Device to load.
- `public readonly int index` - The index of this attribute, in case there are multiple on the same target.
- `public BuildTargetGroup buildTargetGroup` - The build target group this SDK is for.
- `public bool describesFallbackSDK` - Whether this description describes a fallback SDK.

Class Methods

SDK_DescriptionAttribute/5

```
public SDK_DescriptionAttribute(string prettyName, string symbol, string
vrDeviceName, string buildTargetGroupName, int index = 0)
```

- Parameters
 - `string prettyName` - The pretty name of the SDK. Uniquely identifies the SDK. and aren't allowed.
 - `string symbol` - The scripting define symbol needed for the SDK. Needs to be the same as to add and remove the scripting define symbol automatically using . and are allowed.
 - `string vrDeviceName` - The name of the VR Device to load. Set to or if no VR Device is needed.
 - `string buildTargetGroupName` - The name of a constant of . "", and are not allowed.
 - `int index` - The index of this attribute, in case there are multiple on the same target.
- Returns

- *none*

Creates a new attribute.

SDK_DescriptionAttribute/2

```
public SDK_DescriptionAttribute(Type typeToCopyExistingDescriptionFrom, int
index = 0)
```

- Parameters

- `Type typeToCopyExistingDescriptionFrom` - The type to copy the existing from. is not allowed.
- `int index` - The index of the description to copy from the the existing .

- Returns

- *none*

Creates a new attribute by copying from another attribute on a given type.

SDK Scripting Define Symbol Predicate (SDK_ScriptingDefineSymbolPredicateAttribute)

extends Attribute, ISerializationCallbackReceiver

Overview

Specifies a method to be used as a predicate to allow

to automatically add and remove scripting define symbols. Only allowed on

methods that take no arguments and return

.

Class Variables

- `public const string RemovableSymbolPrefix` - The prefix of scripting define symbols that must be used to be able to automatically remove the symbols. Default: "VRTK_DEFINE_"
- `public string symbol` - The scripting define symbol to conditionally add or remove.
- `public BuildTargetGroup buildTargetGroup` - The build target group to use when conditionally adding or removing .

Class Methods

SDK_ScriptingDefineSymbolPredicateAttribute/2

```
public SDK_ScriptingDefineSymbolPredicateAttribute(string symbol, string
buildTargetGroupName)
```

- Parameters

- `string symbol` - The scripting define symbol to conditionally add or remove. Needs to start with `to` to be able to automatically remove the symbol. `and` aren't allowed.
- `string buildTargetGroupName` - The name of a constant of `.`, `,` and aren't allowed.

- Returns

- *none*

Creates a new attribute.

SDK_ScriptingDefineSymbolPredicateAttribute/1

```
public
SDK_ScriptingDefineSymbolPredicateAttribute(SDK_ScriptingDefineSymbolPredica
teAttribute attributeToCopy)
```

- Parameters

- `SDK_ScriptingDefineSymbolPredicateAttribute attributeToCopy` - The attribute to copy.

- Returns

- *none*

Creates a new attribute by copying an existing one.

Base System (SDK_BaseSystem)

extends SDK_Base

Overview

The Base System SDK script provides a bridge to core system SDK methods.

This is an abstract class to implement the interface required by all implemented SDKs.

Class Methods

IsDisplayOnDesktop/0

```
public abstract bool IsDisplayOnDesktop();
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public abstract bool ShouldAppRenderWithLowResources();
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public abstract void ForceInterleavedReprojectionOn(bool force);
```

- Parameters
 - `bool force` - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.
- Returns
 - *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

Base Headset (SDK_BaseHeadset)

extends [SDK_Base](#)

Overview

The Base Headset SDK script provides a bridge to SDK methods that deal with the VR Headset.

This is an abstract class to implement the interface required by all implemented SDKs.

Class Methods

ProcessUpdate/1

```
public abstract void ProcessUpdate(Dictionary<string, object> options);
```

- Parameters
 - Dictionary<string, object> options - A dictionary of generic options that can be used to within the update.
- Returns
 - *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public abstract void ProcessFixedUpdate(Dictionary<string, object> options);
```

- Parameters
 - Dictionary<string, object> options - A dictionary of generic options that can be used to within the fixed update.
- Returns
 - *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetHeadset/0

```
public abstract Transform GetHeadset();
```

- Parameters
 - *none*
- Returns
 - Transform - A transform of the object representing the headset in the scene.

The GetHeadset method returns the Transform of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public abstract Transform GetHeadsetCamera();
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object holding the headset camera in the scene.

The GetHeadsetCamera method returns the Transform of the object that is used to hold the headset camera in the scene.

GetHeadsetVelocity/0

```
public abstract Vector3 GetHeadsetVelocity();
```

- Parameters

- *none*

- Returns

- Vector3 - A Vector3 containing the current velocity of the headset.

The GetHeadsetVelocity method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public abstract Vector3 GetHeadsetAngularVelocity();
```

- Parameters

- *none*

- Returns

- Vector3 - A Vector3 containing the current angular velocity of the headset.

The GetHeadsetAngularVelocity method is used to determine the current angular velocity of the headset.

HeadsetFade/3

```
public abstract void HeadsetFade(Color color, float duration, bool  
fadeOverlay = false);
```

- Parameters

- Color color - The colour to fade to.

- float duration - The amount of time the fade should take to reach the given colour.

- bool fadeOverlay - Determines whether to use an overlay on the fade.

- Returns
 - *none*

The HeadsetFade method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public abstract bool HasHeadsetFade(Transform obj);
```

- Parameters
 - Transform obj - The Transform to check to see if a camera fade is available on.
- Returns
 - bool - Returns true if the headset has fade functionality on it.

The HasHeadsetFade method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public abstract void AddHeadsetFade(Transform camera);
```

- Parameters
 - Transform camera - The Transform to with the camera on to add the fade functionality to.
- Returns
 - *none*

The AddHeadsetFade method attempts to add the fade functionality to the game object with the camera on it.

Base Controller (SDK_BaseController)

extends SDK_Base

Overview

The Base Controller SDK script provides a bridge to SDK methods that deal with the input devices.

This is an abstract class to implement the interface required by all implemented SDKs.

Class Variables

- `public enum ButtonTypes` - Types of buttons on a controller
 - `ButtonOne` - Button One on the controller.
 - `ButtonTwo` - Button Two on the controller.
 - `Grip` - Grip on the controller.
 - `GripHairline` - Grip Hairline on the controller.
 - `StartMenu` - Start Menu on the controller.
 - `Trigger` - Trigger on the controller.
 - `TriggerHairline` - Trigger Hairline on the controller.
 - `Touchpad` - Touchpad on the controller.
- `public enum ButtonPressTypes` - Concepts of controller button press
 - `Press` - The button is currently being pressed.
 - `PressDown` - The button has just been pressed down.
 - `PressUp` - The button has just been released.
 - `Touch` - The button is currently being touched.
 - `TouchDown` - The button has just been touched.
 - `TouchUp` - The button is no longer being touched.
- `public enum ControllerElements` - The elements of a generic controller
 - `AttachPoint` - The default point on the controller to attach grabbed objects to.
 - `Trigger` - The trigger button.
 - `GripLeft` - The left part of the grip button collection.
 - `GripRight` - The right part of the grip button collection.
 - `Touchpad` - The touch pad/stick.
 - `ButtonOne` - The first generic button.
 - `ButtonTwo` - The second generic button.
 - `SystemMenu` - The system menu button.
 - `Body` - The encompassing mesh of the controller body.
 - `StartMenu` - The start menu button.
- `public enum ControllerHand` - Controller hand reference.
 - `None` - No hand is assigned.
 - `Left` - The left hand is assigned.
 - `Right` - The right hand is assigned.
- `public enum ControllerType` - SDK Controller types.
 - `Undefined` - No controller type.
 - `Custom` - A custom controller type.
 - `Simulator_Hand` - The Simulator default hand controller.
 - `SteamVR_ViveWand` - The HTC Vive wand controller for SteamVR.
 - `SteamVR_OculusTouch` - The Oculus Touch controller for SteamVR.
 - `Oculus_OculusTouch` - The Oculus Touch controller for Oculus Utilities.
 - `Daydream_Controller` - The Daydream controller for Google Daydream SDK.
 - `Ximmerse_Flip` - The Flip controller for Ximmerse SDK.

Class Methods

ProcessUpdate/2

```
public abstract void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options);
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public abstract void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options);
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetCurrentControllerType/0

```
public abstract ControllerType GetCurrentControllerType();
```

- Parameters

- *none*

- Returns

- `ControllerType` - The ControllerType based on the SDK and headset being used.

The GetCurrentControllerType method returns the current used ControllerType based on the SDK and headset being used.

GetControllerDefaultColliderPath/1

```
public abstract string GetControllerDefaultColliderPath(ControllerHand hand);
```

- Parameters

- `ControllerHand hand` - The controller hand to check for

- Returns

- `string` - A path to the resource that contains the collider GameObject.

The `GetControllerDefaultColliderPath` returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public abstract string GetControllerElementPath(ControllerElements element, ControllerHand hand, bool fullPath = false);
```

- Parameters

- `ControllerElements element` - The controller element to look up.
- `ControllerHand hand` - The controller hand to look up.
- `bool fullPath` - Whether to get the initial path or the full path to the element.

- Returns

- `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public abstract uint GetControllerIndex(GameObject controller);
```

- Parameters

- `GameObject controller` - The GameObject containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public abstract GameObject GetControllerByIndex(uint index, bool actual =
```

```
false);
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` of the controller

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public abstract Transform GetControllerOrigin(VRTK_ControllerReference  
controllerReference);
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A `Transform` containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public abstract Transform GenerateControllerPointerOrigin(GameObject  
parent);
```

- Parameters

- `GameObject parent` - The `GameObject` that the origin will become parent of. If it is a controller then it will also be used to determine the hand if required.

- Returns

- `Transform` - A generated `Transform` that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin `Transform` to represent the pointer position and forward.

GetControllerLeftHand/1

```
public abstract GameObject GetControllerLeftHand(bool actual = false);
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the left hand controller.

The `GetControllerLeftHand` method returns the `GameObject` containing the representation of the left hand controller.

GetControllerRightHand/1

```
public abstract GameObject GetControllerRightHand(bool actual = false);
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the right hand controller.

The `GetControllerRightHand` method returns the `GameObject` containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public abstract bool IsControllerLeftHand(GameObject controller);
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/1

```
public abstract bool IsControllerRightHand(GameObject controller);
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public abstract bool IsControllerLeftHand(GameObject controller, bool actual);
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public abstract bool IsControllerRightHand(GameObject controller, bool actual);
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public abstract GameObject GetControllerModel(GameObject controller);
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public abstract GameObject GetControllerModel(ControllerHand hand);
```

- Parameters

- `ControllerHand hand` - The hand enum of which controller model to retrieve.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given controller hand.

GetControllerModelHand/1

```
public virtual ControllerHand GetControllerModelHand(GameObject  
controllerModel)
```

- Parameters

- `GameObject controllerModel` - The controller model `GameObject` to get the hand for.

- Returns

- `ControllerHand` - The hand enum for which the given controller model is for.

The `GetControllerModelHand` method returns the hand for the given controller model `GameObject`.

GetControllerRenderModel/1

```
public abstract GameObject GetControllerRenderModel(VRTK_ControllerReference  
controllerReference);
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to check.

- Returns

- `GameObject` - A `GameObject` containing the object that has a render model for the controller.

The `GetControllerRenderModel` method gets the game object that contains the given controller's render model.

SetControllerRenderModelWheel/2

```
public abstract void SetControllerRenderModelWheel(GameObject renderModel,  
bool state);
```

- Parameters

- `GameObject renderModel` - The `GameObject` containing the controller render model.
- `bool state` - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.
- Returns
 - *none*

The `SetControllerRenderModelWheel` method sets the state of the scroll wheel on the controller render model.

HapticPulse/2

```
public abstract void HapticPulse(VRTK_ControllerReference
    controllerReference, float strength = 0.5f);
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
 - `float strength` - The intensity of the rumble of the controller motor. 0 to 1.
- Returns
 - *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public abstract bool HapticPulse(VRTK_ControllerReference
    controllerReference, AudioClip clip);
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
 - `AudioClip clip` - The audio clip to use for the haptic pattern.
- Returns
 - *none*

The `HapticPulse/2` method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public abstract SDK_ControllerHapticModifiers GetHapticModifiers();
```

- Parameters

- *none*

- Returns

- `SDK_ControllerHapticModifiers` - An `SDK_ControllerHapticModifiers` object with a given `durationModifier` and an `intervalModifier`.

The `GetHapticModifiers` method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public abstract Vector3 GetVelocity(VRTK_ControllerReference  
controllerReference);
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current velocity of the tracked object.

The `GetVelocity` method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public abstract Vector3 GetAngularVelocity(VRTK_ControllerReference  
controllerReference);
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the tracked object.

The `GetAngularVelocity` method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public abstract bool IsTouchpadStatic(bool isTouched, Vector2  
currentAxisValues, Vector2 previousAxisValues, int compareFidelity);
```

- Parameters

- `Vector2 currentAxisValues` -
- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- **Returns**

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public abstract Vector2 GetButtonAxis(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference);
```

- **Parameters**

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.

- **Returns**

- `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public abstract float GetButtonHairlineDelta(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference);
```

- **Parameters**

- `ButtonTypes buttonType` - The type of button to get the hairline delta for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.

- **Returns**

- `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public abstract bool GetControllerButtonState(ButtonTypes buttonType,
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference);
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the state of.
- `ButtonPressTypes pressType` - The button state to check for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.

- Returns

- `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the given press type on the given controller reference is currently taking place.

Base Boundaries (SDK_BaseBoundaries)

extends [SDK_Base](#)

Overview

The Base Boundaries SDK script provides a bridge to SDK methods that deal with the play area of SDKs that support room scale play spaces.

This is an abstract class to implement the interface required by all implemented SDKs.

Class Methods

InitBoundaries/0

```
public abstract void InitBoundaries();
```

- Parameters

- *none*

- Returns

- *none*

The `InitBoundaries` method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public abstract Transform GetPlayArea();
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the play area in the scene.

The `GetPlayArea` method returns the `Transform` of the object that is used to represent the play area in the scene.

GetPlayAreaVertices/0

```
public abstract Vector3[] GetPlayAreaVertices();
```

- Parameters
 - *none*
- Returns
 - `Vector3[]` - A `Vector3` array of the points in the scene that represent the play area boundaries.

The `GetPlayAreaVertices` method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public abstract float GetPlayAreaBorderThickness();
```

- Parameters
 - *none*
- Returns
 - `float` - The thickness of the drawn border.

The `GetPlayAreaBorderThickness` returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public abstract bool IsPlayAreaSizeCalibrated();
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the play area size has been auto calibrated and set by external sensors.

The `IsPlayAreaSizeCalibrated` method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public abstract bool GetDrawAtRuntime();
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the drawn border is being displayed.

The `GetDrawAtRuntime` method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public abstract void SetDrawAtRuntime(bool value);
```

- Parameters
 - `bool value` - The state of whether the drawn border should be displayed or not.
- Returns
 - *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.

Fallback SDK (VRTK/SDK/Fallback)

The scripts used to provide a default/null fallback for any implemented SDK.

- [Fallback System](#)
- [Fallback Headset](#)
- [Fallback Controller](#)
- [Fallback Boundaries](#)

Fallback System (SDK_FallbackSystem)

extends [SDK_BaseSystem](#)

Overview

The Fallback System SDK script provides a fallback collection of methods that return null or default system values.

This is the fallback class that will just return default values.

Class Methods

IsDisplayOnDesktop/0

```
public override bool IsDisplayOnDesktop()
```

- Parameters
 - *none*
- Returns
 - *bool* - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public override bool ShouldAppRenderWithLowResources()
```

- Parameters
 - *none*
- Returns
 - *bool* - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public override void ForceInterleavedReprojectionOn(bool force)
```

- Parameters
 - *bool force* - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.
- Returns
 - *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

Fallback Headset (SDK_FallbackHeadset)

extends SDK_BaseHeadset

Overview

The Fallback Headset SDK script provides a fallback collection of methods that return null or default headset values.

This is the fallback class that will just return default values.

Class Methods

ProcessUpdate/1

```
public override void ProcessUpdate(Dictionary<string, object> options)
```

- Parameters
 - Dictionary<string, object> options - A dictionary of generic options that can be used to within the update.
- Returns
 - *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public override void ProcessFixedUpdate(Dictionary<string, object> options)
```

- Parameters
 - Dictionary<string, object> options - A dictionary of generic options that can be used to within the fixed update.
- Returns
 - *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetHeadset/0

```
public override Transform GetHeadset()
```

- Parameters

- *none*

- Returns

- `Transform` - A transform of the object representing the headset in the scene.

The `GetHeadset` method returns the `Transform` of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public override Transform GetHeadsetCamera()
```

- Parameters

- *none*

- Returns

- `Transform` - A transform of the object holding the headset camera in the scene.

The `GetHeadsetCamera` method returns the `Transform` of the object that is used to hold the headset camera in the scene.

GetHeadsetVelocity/0

```
public override Vector3 GetHeadsetVelocity()
```

- Parameters

- *none*

- Returns

- `Vector3` - A `Vector3` containing the current velocity of the headset.

The `GetHeadsetVelocity` method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public override Vector3 GetHeadsetAngularVelocity()
```

- Parameters

- *none*

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

HeadsetFade/3

```
public override void HeadsetFade(Color color, float duration, bool
fadeOverlay = false)
```

- **Parameters**

- `Color color` - The colour to fade to.
- `float duration` - The amount of time the fade should take to reach the given colour.
- `bool fadeOverlay` - Determines whether to use an overlay on the fade.

- **Returns**

- *none*

The `HeadsetFade` method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public override bool HasHeadsetFade(Transform obj)
```

- **Parameters**

- `Transform obj` - The Transform to check to see if a camera fade is available on.

- **Returns**

- `bool` - Returns true if the headset has fade functionality on it.

The `HasHeadsetFade` method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public override void AddHeadsetFade(Transform camera)
```

- **Parameters**

- `Transform camera` - The Transform to with the camera on to add the fade functionality to.

- **Returns**

- *none*

The `AddHeadsetFade` method attempts to add the fade functionality to the game object with the camera on it.

Fallback Controller (SDK_FallbackController)

extends [SDK_BaseController](#)

Overview

The Fallback Controller SDK script provides a fallback collection of methods that return null or default headset values.

This is the fallback class that will just return default values.

Class Methods

ProcessUpdate/2

```
public override void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public override void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetCurrentControllerType/0

```
public override ControllerType GetCurrentControllerType()
```

- Parameters

- *none*

- Returns

- `ControllerType` - The `ControllerType` based on the SDK and headset being used.

The `GetCurrentControllerType` method returns the current used `ControllerType` based on the SDK and headset being used.

GetControllerDefaultColliderPath/1

```
public override string GetControllerDefaultColliderPath(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The controller hand to check for

- Returns

- `string` - A path to the resource that contains the collider `GameObject`.

The `GetControllerDefaultColliderPath` returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public override string GetControllerElementPath(ControllerElements element,  
ControllerHand hand, bool fullPath = false)
```

- Parameters

- `ControllerElements element` - The controller element to look up.
- `ControllerHand hand` - The controller hand to look up.
- `bool fullPath` - Whether to get the initial path or the full path to the element.

- Returns

- `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public override uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public override GameObject GetControllerByIndex(uint index, bool actual = false)
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` of the controller

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public override Transform GetControllerOrigin(VRTK_ControllerReference controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A `Transform` containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public override Transform GenerateControllerPointerOrigin(GameObject parent)
```

- Parameters

- `GameObject parent` - The `GameObject` that the origin will become parent of. If it is a controller then it will also be used to determine the hand if required.

- Returns

- `Transform` - A generated `Transform` that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin `Transform` to represent the pointer position and forward.

GetControllerLeftHand/1

```
public override GameObject GetControllerLeftHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the left hand controller.

The `GetControllerLeftHand` method returns the `GameObject` containing the representation of the left hand controller.

GetControllerRightHand/1

```
public override GameObject GetControllerRightHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the right hand controller.

The `GetControllerRightHand` method returns the `GameObject` containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public override bool IsControllerLeftHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/1

```
public override bool IsControllerRightHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public override bool IsControllerLeftHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public override bool IsControllerRightHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public override GameObject GetControllerModel(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public override GameObject GetControllerModel(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The hand enum of which controller model to retrieve.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given controller hand.

GetControllerRenderModel/1

```
public override GameObject GetControllerRenderModel(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to check.

- Returns

- `GameObject` - A `GameObject` containing the object that has a render model for the controller.

The `GetControllerRenderModel` method gets the game object that contains the given controller's render model.

SetControllerRenderModelWheel/2

```
public override void SetControllerRenderModelWheel(GameObject renderModel,  
bool state)
```

- Parameters

- `GameObject renderModel` - The `GameObject` containing the controller render model.
- `bool state` - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.

- Returns

- *none*

The `SetControllerRenderModelWheel` method sets the state of the scroll wheel on the controller render model.

HapticPulse/2


```
public override void HapticPulse(VRTK_ControllerReference  
controllerReference, float strength = 0.5f)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public override bool HapticPulse(VRTK_ControllerReference  
controllerReference, AudioClip clip)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `AudioClip clip` - The audio clip to use for the haptic pattern.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public override SDK_ControllerHapticModifiers GetHapticModifiers()
```

- Parameters

- *none*

- Returns

- `SDK_ControllerHapticModifiers` - An `SDK_ControllerHapticModifiers` object with a given `durationModifier` and an `intervalModifier`.

The `GetHapticModifiers` method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public override Vector3 GetVelocity(VRTK_ControllerReference
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current velocity of the tracked object.

The `GetVelocity` method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public override Vector3 GetAngularVelocity(VRTK_ControllerReference
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the tracked object.

The `GetAngularVelocity` method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public override bool IsTouchpadStatic(bool isTouched, Vector2
currentAxisValues, Vector2 previousAxisValues, int compareFidelity)
```

- Parameters

- `Vector2 currentAxisValues` -
- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- Returns

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public override Vector2 GetButtonAxis(ButtonTypes buttonType,
```

```
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.

- Returns

- `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public override float GetButtonHairlineDelta(ButtonTypes buttonType,  
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to get the hairline delta for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.

- Returns

- `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public override bool GetControllerButtonState(ButtonTypes buttonType,  
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the state of.
- `ButtonPressTypes pressType` - The button state to check for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.

- Returns

- `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the

given press type on the given controller reference is currently taking place.

Fallback Boundaries (SDK_FallbackBoundaries)

extends SDK_BaseBoundaries

Overview

The Fallback Boundaries SDK script provides a fallback collection of methods that return null or default headset values.

This is the fallback class that will just return default values.

Class Methods

InitBoundaries/0

```
public override void InitBoundaries()
```

- Parameters
 - *none*
- Returns
 - *none*

The InitBoundaries method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public override Transform GetPlayArea()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the play area in the scene.

The GetPlayArea method returns the Transform of the object that is used to represent the play area in the scene.

GetPlayAreaVertices/0

```
public override Vector3[] GetPlayAreaVertices()
```

- Parameters

- *none*

- Returns

- Vector3[] - A Vector3 array of the points in the scene that represent the play area boundaries.

The GetPlayAreaVertices method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public override float GetPlayAreaBorderThickness()
```

- Parameters

- *none*

- Returns

- float - The thickness of the drawn border.

The GetPlayAreaBorderThickness returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public override bool IsPlayAreaSizeCalibrated()
```

- Parameters

- *none*

- Returns

- bool - Returns true if the play area size has been auto calibrated and set by external sensors.

The IsPlayAreaSizeCalibrated method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public override bool GetDrawAtRuntime()
```

- Parameters

- *none*

- Returns

- bool - Returns true if the drawn border is being displayed.

The GetDrawAtRuntime method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public override void SetDrawAtRuntime(bool value)
```

- Parameters

- `bool value` - The state of whether the drawn border should be displayed or not.

- Returns

- *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.

Simulator SDK (VRTK/SDK/Simulator)

The scripts used to utilise the VR Simulator to build without VR Hardware.

- [Simulator System](#)
 - [Simulator Headset](#)
 - [Simulator Controller](#)
 - [Simulator Boundaries](#)
-

Simulator System (SDK_SimSystem)

extends [SDK_BaseSystem](#)

Overview

The Sim System SDK script provides dummy functions for system functions.

Class Methods

IsDisplayOnDesktop/0

```
public override bool IsDisplayOnDesktop()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public override bool ShouldAppRenderWithLowResources()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public override void ForceInterleavedReprojectionOn(bool force)
```

- Parameters

- `bool force` - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.

- Returns

- *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

Simulator Headset (SDK_SimHeadset)

extends [SDK_BaseHeadset](#)

Overview

The Sim Headset SDK script provides dummy functions for the headset.

Class Methods

ProcessUpdate/1

```
public override void ProcessUpdate(Dictionary<string, object> options)
```

- Parameters

- Dictionary<string, object> options - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public override void ProcessFixedUpdate(Dictionary<string, object> options)
```

- Parameters

- Dictionary<string, object> options - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetHeadset/0

```
public override Transform GetHeadset()
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object representing the headset in the scene.

The GetHeadset method returns the Transform of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public override Transform GetHeadsetCamera()
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object holding the headset camera in the scene.

The `GetHeadsetCamera/0` method returns the Transform of the object that is used to hold the headset camera in the scene.

GetHeadsetVelocity/0

```
public override Vector3 GetHeadsetVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` containing the current velocity of the headset.

The `GetHeadsetVelocity` method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public override Vector3 GetHeadsetAngularVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

HeadsetFade/3

```
public override void HeadsetFade(Color color, float duration, bool  
fadeOverlay = false)
```

- Parameters
 - `Color color` - The colour to fade to.
 - `float duration` - The amount of time the fade should take to reach the given colour.
 - `bool fadeOverlay` - Determines whether to use an overlay on the fade.
- Returns
 - *none*

The `HeadsetFade` method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public override bool HasHeadsetFade(Transform obj)
```

- Parameters

- Transform obj - The Transform to check to see if a camera fade is available on.

- Returns

- bool - Returns true if the headset has fade functionality on it.

The HasHeadsetFade method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public override void AddHeadsetFade(Transform camera)
```

- Parameters

- Transform camera - The Transform to with the camera on to add the fade functionality to.

- Returns

- *none*

The AddHeadsetFade method attempts to add the fade functionality to the game object with the camera on it.

Simulator Controller (SDK_SimController)

extends SDK_BaseController

Overview

The Sim Controller SDK script provides functions to help simulate VR controllers.

Class Methods

ProcessUpdate/2

```
public override void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- VRTK_ControllerReference controllerReference - The reference for the controller.
- Dictionary<string, object> options - A dictionary of generic options that can be used to

within the update.

- Returns
 - *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public override void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters
 - VRTK_ControllerReference controllerReference - The reference for the controller.
 - Dictionary<string, object> options - A dictionary of generic options that can be used to within the fixed update.
- Returns
 - *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetCurrentControllerType/0

```
public override ControllerType GetCurrentControllerType()
```

- Parameters
 - *none*
- Returns
 - ControllerType - The ControllerType based on the SDK and headset being used.

The GetCurrentControllerType method returns the current used ControllerType based on the SDK and headset being used.

GetControllerDefaultColliderPath/1

```
public override string GetControllerDefaultColliderPath(ControllerHand hand)
```

- Parameters
 - ControllerHand hand - The controller hand to check for
- Returns
 - string - A path to the resource that contains the collider GameObject.

The GetControllerDefaultColliderPath returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public override string GetControllerElementPath(ControllerElements element,
ControllerHand hand, bool fullPath = false)
```

- Parameters

- `ControllerElements element` - The controller element to look up.
- `ControllerHand hand` - The controller hand to look up.
- `bool fullPath` - Whether to get the initial path or the full path to the element.

- Returns

- `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public override uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public override GameObject GetControllerByIndex(uint index, bool actual =
false)
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` of the controller

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public override Transform GetControllerOrigin(VRTK_ControllerReference
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A Transform containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public override Transform GenerateControllerPointerOrigin(GameObject parent)
```

- Parameters

- `GameObject parent` - The `GameObject` that the origin will become parent of. If it is a controller then it will also be used to determine the hand if required.

- Returns

- `Transform` - A generated Transform that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin Transform to represent the pointer position and forward.

GetControllerLeftHand/1

```
public override GameObject GetControllerLeftHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the left hand controller.

The `GetControllerLeftHand` method returns the `GameObject` containing the representation of the left hand controller.

GetControllerRightHand/1

```
public override GameObject GetControllerRightHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.
- Returns
 - `GameObject` - The `GameObject` containing the right hand controller.

The `GetControllerRightHand` method returns the `GameObject` containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public override bool IsControllerLeftHand(GameObject controller)
```

- Parameters
 - `GameObject controller` - The `GameObject` to check.
- Returns
 - `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/1

```
public override bool IsControllerRightHand(GameObject controller)
```

- Parameters
 - `GameObject controller` - The `GameObject` to check.
- Returns
 - `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public override bool IsControllerLeftHand(GameObject controller, bool actual)
```

- Parameters
 - `GameObject controller` - The `GameObject` to check.
 - `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.
- Returns
 - `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public override bool IsControllerRightHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public override GameObject GetControllerModel(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public override GameObject GetControllerModel(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The hand enum of which controller model to retrieve.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given controller hand.

GetControllerRenderModel/1

```
public override GameObject GetControllerRenderModel(VRTK_ControllerReference
```

```
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to check.

- Returns

- `GameObject` - A `GameObject` containing the object that has a render model for the controller.

The `GetControllerRenderModel` method gets the game object that contains the given controller's render model.

SetControllerRenderModelWheel/2

```
public override void SetControllerRenderModelWheel(GameObject renderModel,  
bool state)
```

- Parameters

- `GameObject renderModel` - The `GameObject` containing the controller render model.
- `bool state` - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.

- Returns

- *none*

The `SetControllerRenderModelWheel` method sets the state of the scroll wheel on the controller render model.

HapticPulse/2

```
public override void HapticPulse(VRTK_ControllerReference  
controllerReference, float strength = 0.5f)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public override bool HapticPulse(VRTK_ControllerReference
```



```
controllerReference, AudioClip clip)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `AudioClip clip` - The audio clip to use for the haptic pattern.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public override SDK_ControllerHapticModifiers GetHapticModifiers()
```

- Parameters

- *none*

- Returns

- `SDK_ControllerHapticModifiers` - An `SDK_ControllerHapticModifiers` object with a given `durationModifier` and an `intervalModifier`.

The `GetHapticModifiers` method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public override Vector3 GetVelocity(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current velocity of the tracked object.

The `GetVelocity` method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public override Vector3 GetAngularVelocity(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the tracked object.

The `GetAngularVelocity` method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public override bool IsTouchpadStatic(bool isTouched, Vector2
currentAxisValues, Vector2 previousAxisValues, int compareFidelity)
```

- Parameters

- `Vector2 currentAxisValues` -
- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- Returns

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public override Vector2 GetButtonAxis(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.

- Returns

- `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public override float GetButtonHairlineDelta(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to get the hairline delta for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.

- Returns

- `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public override bool GetControllerButtonState(ButtonTypes buttonType,  
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the state of.
- `ButtonPressTypes pressType` - The button state to check for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.

- Returns

- `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the given press type on the given controller reference is currently taking place.

Simulator Boundaries (SDK_SimBoundaries)

extends [SDK_BaseBoundaries](#)

Overview

The Sim Boundaries SDK script provides dummy functions for the play area boundaries.

Class Methods

InitBoundaries/0

```
public override void InitBoundaries()
```

- Parameters
 - *none*
- Returns
 - *none*

The `InitBoundaries` method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public override Transform GetPlayArea()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the play area in the scene.

The `GetPlayArea` method returns the `Transform` of the object that is used to represent the play area in the scene.

GetPlayAreaVertices/0

```
public override Vector3[] GetPlayAreaVertices()
```

- Parameters
 - *none*
- Returns
 - `Vector3[]` - A `Vector3` array of the points in the scene that represent the play area boundaries.

The `GetPlayAreaVertices` method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public override float GetPlayAreaBorderThickness()
```

- Parameters
 - *none*
- Returns
 - `float` - The thickness of the drawn border.

The `GetPlayAreaBorderThickness` returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public override bool IsPlayAreaSizeCalibrated()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the play area size has been auto calibrated and set by external sensors.

The `IsPlayAreaSizeCalibrated` method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public override bool GetDrawAtRuntime()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the drawn border is being displayed.

The `GetDrawAtRuntime` method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public override void SetDrawAtRuntime(bool value)
```

- Parameters

- `bool value` - The state of whether the drawn border should be displayed or not.

- Returns

- *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.

SteamVR SDK (VRTK/SDK/SteamVR)

The scripts used to utilise the SteamVR Unity Plugin SDK.

- [SteamVR Defines](#)
- [SteamVR System](#)
- [SteamVR Headset](#)

- [SteamVR Controller](#)
 - [SteamVR Boundaries](#)
-

SteamVR Defines (SDK_SteamVRDefines)

Overview

Handles all the scripting define symbols for the SteamVR SDK.

Class Variables

- `public const string ScriptingDefineSymbol` - The scripting define symbol for the SteamVR SDK.
Default: `SDK_ScriptingDefineSymbolPredicateAttribute.RemovableSymbolPrefix + "SDK_STEAMVR"`
-

SteamVR System (SDK_SteamVRSystem)

Overview

The SteamVR System SDK script provides a bridge to the SteamVR SDK.

Class Methods

IsDisplayOnDesktop/0

```
public override bool IsDisplayOnDesktop()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public override bool ShouldAppRenderWithLowResources()
```

- Parameters
 - *none*
- Returns

- `bool` - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public override void ForceInterleavedReprojectionOn(bool force)
```

- Parameters

- `bool force` - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.

- Returns

- *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

SteamVR Headset (SDK_SteamVRHeadset)

Overview

The SteamVR Headset SDK script provides a bridge to the SteamVR SDK.

Class Methods

ProcessUpdate/1

```
public override void ProcessUpdate(Dictionary<string, object> options)
```

- Parameters

- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The `ProcessUpdate` method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public override void ProcessFixedUpdate(Dictionary<string, object> options)
```

- Parameters

- Dictionary<string, object> options - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetHeadset/0

```
public override Transform GetHeadset()
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object representing the headset in the scene.

The GetHeadset method returns the Transform of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public override Transform GetHeadsetCamera()
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object holding the headset camera in the scene.

The GetHeadsetCamera method returns the Transform of the object that is used to hold the headset camera in the scene.

GetHeadsetVelocity/0

```
public override Vector3 GetHeadsetVelocity()
```

- Parameters

- *none*

- Returns

- Vector3 - A Vector3 containing the current velocity of the headset.

The GetHeadsetVelocity method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public override Vector3 GetHeadsetAngularVelocity()
```

- Parameters

- *none*

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

HeadsetFade/3

```
public override void HeadsetFade(Color color, float duration, bool  
fadeOverlay = false)
```

- Parameters

- `Color color` - The colour to fade to.
- `float duration` - The amount of time the fade should take to reach the given colour.
- `bool fadeOverlay` - Determines whether to use an overlay on the fade.

- Returns

- *none*

The `HeadsetFade` method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public override bool HasHeadsetFade(Transform obj)
```

- Parameters

- `Transform obj` - The `Transform` to check to see if a camera fade is available on.

- Returns

- `bool` - Returns true if the headset has fade functionality on it.

The `HasHeadsetFade` method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public override void AddHeadsetFade(Transform camera)
```

- Parameters
 - `Transform camera` - The Transform to with the camera on to add the fade functionality to.
- Returns
 - *none*

The `AddHeadsetFade` method attempts to add the fade functionality to the game object with the camera on it.

SteamVR Controller (SDK_SteamVRController)

Overview

The SteamVR Controller SDK script provides a bridge to SDK methods that deal with the input devices.

Class Methods

OnAfterSetupUnload/1

```
public override void OnAfterSetupUnload(VRTK_SDKSetup setup)
```

- Parameters
 - `VRTK_SDKSetup setup` - The SDK Setup which is using this SDK.
- Returns
 - *none*

This method is called just after unloading the that's using this SDK.

ProcessUpdate/2

```
public override void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference for the controller.
 - `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.
- Returns
 - *none*

The `ProcessUpdate` method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public override void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The `ProcessFixedUpdate` method enables an SDK to run logic for every Unity FixedUpdate

GetCurrentControllerType/0

```
public override ControllerType GetCurrentControllerType()
```

- Parameters

- *none*

- Returns

- `ControllerType` - The `ControllerType` based on the SDK and headset being used.

The `GetCurrentControllerType` method returns the current used `ControllerType` based on the SDK and headset being used.

GetControllerDefaultColliderPath/1

```
public override string GetControllerDefaultColliderPath(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The controller hand to check for

- Returns

- `string` - A path to the resource that contains the collider `GameObject`.

The `GetControllerDefaultColliderPath` returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public override string GetControllerElementPath(ControllerElements element,  
ControllerHand hand, bool fullPath = false)
```

- Parameters

- `ControllerElements element` - The controller element to look up.
- `ControllerHand hand` - The controller hand to look up.
- `bool fullPath` - Whether to get the initial path or the full path to the element.

- Returns

- `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public override uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public override GameObject GetControllerByIndex(uint index, bool actual = false)
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` of the controller

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public override Transform GetControllerOrigin(VRTK_ControllerReference controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A Transform containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public override Transform GenerateControllerPointerOrigin(GameObject parent)
```

- Parameters

- `GameObject parent` - The `GameObject` that the origin will become parent of. If it is a controller then it will also be used to determine the hand if required.

- Returns

- `Transform` - A generated Transform that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin Transform to represent the pointer position and forward.

GetControllerLeftHand/1

```
public override GameObject GetControllerLeftHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the left hand controller.

The `GetControllerLeftHand` method returns the `GameObject` containing the representation of the left hand controller.

GetControllerRightHand/1

```
public override GameObject GetControllerRightHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the right hand controller.

The `GetControllerRightHand` method returns the `GameObject` containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public override bool IsControllerLeftHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/1

```
public override bool IsControllerRightHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public override bool IsControllerLeftHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public override bool IsControllerRightHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public override GameObject GetControllerModel(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public override GameObject GetControllerModel(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The hand enum of which controller model to retrieve.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given controller hand.

GetControllerRenderModel/1

```
public override GameObject GetControllerRenderModel(VRTK_ControllerReference controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to check.

- Returns

- `GameObject` - A `GameObject` containing the object that has a render model for the controller.

The `GetControllerRenderModel` method gets the game object that contains the given controller's

render model.

SetControllerRenderModelWheel/2

```
public override void SetControllerRenderModelWheel(GameObject renderModel,  
bool state)
```

- Parameters

- `GameObject renderModel` - The `GameObject` containing the controller render model.
- `bool state` - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.

- Returns

- *none*

The `SetControllerRenderModelWheel` method sets the state of the scroll wheel on the controller render model.

HapticPulse/2

```
public override void HapticPulse(VRTK_ControllerReference  
controllerReference, float strength = 0.5f)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public override bool HapticPulse(VRTK_ControllerReference  
controllerReference, AudioClip clip)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `AudioClip clip` - The audio clip to use for the haptic pattern.

- Returns

- *none*

The HapticPulse/2 method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public override SDK_ControllerHapticModifiers GetHapticModifiers()
```

- Parameters

- *none*

- Returns

- SDK_ControllerHapticModifiers - An SDK_ControllerHapticModifiers object with a given durationModifier and an intervalModifier.

The GetHapticModifiers method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public override Vector3 GetVelocity(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- VRTK_ControllerReference controllerReference - The reference to the tracked object to check for.

- Returns

- Vector3 - A Vector3 containing the current velocity of the tracked object.

The GetVelocity method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public override Vector3 GetAngularVelocity(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- VRTK_ControllerReference controllerReference - The reference to the tracked object to check for.

- Returns

- Vector3 - A Vector3 containing the current angular velocity of the tracked object.

The `GetAngularVelocity` method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public override bool IsTouchpadStatic(bool isTouched, Vector2
currentAxisValues, Vector2 previousAxisValues, int compareFidelity)
```

- Parameters

- `Vector2 currentAxisValues` -
- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- Returns

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public override Vector2 GetButtonAxis(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.

- Returns

- `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public override float GetButtonHairlineDelta(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to get the hairline delta for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.

- Returns
 - `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public override bool GetControllerButtonState(ButtonTypes buttonType,  
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference)
```

- Parameters
 - `ButtonTypes buttonType` - The type of button to check for the state of.
 - `ButtonPressTypes pressType` - The button state to check for.
 - `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.
- Returns
 - `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the given press type on the given controller reference is currently taking place.

SteamVR Boundaries (SDK_SteamVRBoundaries)

Overview

The SteamVR Boundaries SDK script provides a bridge to the SteamVR SDK play area.

Class Methods

InitBoundaries/0

```
public override void InitBoundaries()
```

- Parameters
 - *none*
- Returns
 - *none*

The `InitBoundaries` method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public override Transform GetPlayArea()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the play area in the scene.

The `GetPlayArea` method returns the `Transform` of the object that is used to represent the play area in the scene.

GetPlayAreaVertices/0

```
public override Vector3[] GetPlayAreaVertices()
```

- Parameters
 - *none*
- Returns
 - `Vector3[]` - A `Vector3` array of the points in the scene that represent the play area boundaries.

The `GetPlayAreaVertices` method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public override float GetPlayAreaBorderThickness()
```

- Parameters
 - *none*
- Returns
 - `float` - The thickness of the drawn border.

The `GetPlayAreaBorderThickness` returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public override bool IsPlayAreaSizeCalibrated()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the play area size has been auto calibrated and set by external sensors.

The `IsPlayAreaSizeCalibrated` method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public override bool GetDrawAtRuntime()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the drawn border is being displayed.

The `GetDrawAtRuntime` method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public override void SetDrawAtRuntime(bool value)
```

- Parameters
 - `bool value` - The state of whether the drawn border should be displayed or not.
- Returns
 - *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.

Oculus SDK (VRTK/SDK/Oculus)

The scripts used to utilise the Oculus Utilities Unity Package SDK.

- [Oculus Defines](#)
- [Oculus System](#)
- [Oculus Headset](#)
- [Oculus Controller](#)
- [Oculus Boundaries](#)

Oculus Defines (SDK_OculusDefines)

Overview

Handles all the scripting define symbols for the Oculus and Avatar SDKs.

Class Variables

- `public const string ScriptingDefineSymbol` - The scripting define symbol for the Oculus SDK.
Default: `SDK_ScriptingDefineSymbolPredicateAttribute.RemovableSymbolPrefix + "SDK_OCULUS"`
 - `public const string AvatarScriptingDefineSymbol` - The scripting define symbol for the Oculus Avatar SDK. Default: `SDK_ScriptingDefineSymbolPredicateAttribute.RemovableSymbolPrefix + "SDK_OCULUS_AVATAR"`
-

Oculus System (SDK_OculusSystem)

Overview

The Oculus System SDK script provides a bridge to the Oculus SDK.

Class Methods

IsDisplayOnDesktop/0

```
public override bool IsDisplayOnDesktop()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public override bool ShouldAppRenderWithLowResources()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public override void ForceInterleavedReprojectionOn(bool force)
```

- Parameters

- `bool force` - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.

- Returns

- *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

Oculus Headset (SDK_OculusHeadset)

Overview

The Oculus Headset SDK script provides a bridge to the Oculus SDK.

Class Methods

ProcessUpdate/1

```
public override void ProcessUpdate(Dictionary<string, object> options)
```

- Parameters

- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The `ProcessUpdate` method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public override void ProcessFixedUpdate(Dictionary<string, object> options)
```

- Parameters

- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The `ProcessFixedUpdate` method enables an SDK to run logic for every Unity `FixedUpdate`

GetHeadset/0

```
public override Transform GetHeadset()
```

- Parameters

- *none*

- Returns

- `Transform` - A transform of the object representing the headset in the scene.

The `GetHeadset` method returns the `Transform` of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public override Transform GetHeadsetCamera()
```

- Parameters

- *none*

- Returns

- `Transform` - A transform of the object holding the headset camera in the scene.

The `GetHeadsetCamera` method returns the `Transform` of the object that is used to hold the headset camera in the scene.

GetHeadsetVelocity/0

```
public override Vector3 GetHeadsetVelocity()
```

- Parameters

- *none*

- Returns

- `Vector3` - A `Vector3` containing the current velocity of the headset.

The `GetHeadsetVelocity` method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public override Vector3 GetHeadsetAngularVelocity()
```


- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

HeadsetFade/3

```
public override void HeadsetFade(Color color, float duration, bool
fadeOverlay = false)
```

- Parameters
 - `Color color` - The colour to fade to.
 - `float duration` - The amount of time the fade should take to reach the given colour.
 - `bool fadeOverlay` - Determines whether to use an overlay on the fade.
- Returns
 - *none*

The `HeadsetFade` method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public override bool HasHeadsetFade(Transform obj)
```

- Parameters
 - `Transform obj` - The `Transform` to check to see if a camera fade is available on.
- Returns
 - `bool` - Returns true if the headset has fade functionality on it.

The `HasHeadsetFade` method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public override void AddHeadsetFade(Transform camera)
```

- Parameters
 - `Transform camera` - The `Transform` to with the camera on to add the fade functionality to.
- Returns
 - *none*

The AddHeadsetFade method attempts to add the fade functionality to the game object with the camera on it.

Oculus Controller (SDK_OculusController)

Overview

The Oculus Controller SDK script provides a bridge to SDK methods that deal with the input devices.

Class Methods

OnAfterSetupLoad/1

```
public override void OnAfterSetupLoad(VRTK_SDKSetup setup)
```

- Parameters
 - `VRTK_SDKSetup setup` - The SDK Setup which is using this SDK.
- Returns
 - *none*

This method is called just after loading the that's using this SDK.

ProcessUpdate/2

```
public override void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference for the controller.
 - `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.
- Returns
 - *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public override void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The `ProcessFixedUpdate` method enables an SDK to run logic for every Unity `FixedUpdate`

GetCurrentControllerType/0

```
public override ControllerType GetCurrentControllerType()
```

- Parameters

- *none*

- Returns

- `ControllerType` - The `ControllerType` based on the SDK and headset being used.

The `GetCurrentControllerType` method returns the current used `ControllerType` based on the SDK and headset being used.

GetControllerDefaultColliderPath/1

```
public override string GetControllerDefaultColliderPath(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The controller hand to check for

- Returns

- `string` - A path to the resource that contains the collider `GameObject`.

The `GetControllerDefaultColliderPath` returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public override string GetControllerElementPath(ControllerElements element,  
ControllerHand hand, bool fullPath = false)
```

- Parameters

- `ControllerElements element` - The controller element to look up.
- `ControllerHand hand` - The controller hand to look up.
- `bool fullPath` - Whether to get the initial path or the full path to the element.

- Returns

- `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public override uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public override GameObject GetControllerByIndex(uint index, bool actual = false)
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` of the controller

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public override Transform GetControllerOrigin(VRTK_ControllerReference controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A `Transform` containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public override Transform GenerateControllerPointerOrigin(GameObject parent)
```

- Parameters

- `GameObject parent` - The `GameObject` that the origin will become parent of. If it is a controller then it will also be used to determine the hand if required.

- Returns

- `Transform` - A generated `Transform` that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin `Transform` to represent the pointer position and forward.

GetControllerLeftHand/1

```
public override GameObject GetControllerLeftHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the left hand controller.

The `GetControllerLeftHand` method returns the `GameObject` containing the representation of the left hand controller.

GetControllerRightHand/1

```
public override GameObject GetControllerRightHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the right hand controller.

The `GetControllerRightHand` method returns the `GameObject` containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public override bool IsControllerLeftHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/1

```
public override bool IsControllerRightHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public override bool IsControllerLeftHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public override bool IsControllerRightHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public override GameObject GetControllerModel(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public override GameObject GetControllerModel(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The hand enum of which controller model to retrieve.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given controller hand.

GetControllerRenderModel/1

```
public override GameObject GetControllerRenderModel(VRTK_ControllerReference controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to check.

- Returns

- `GameObject` - A `GameObject` containing the object that has a render model for the controller.

The `GetControllerRenderModel` method gets the game object that contains the given controller's render model.

SetControllerRenderModelWheel/2

```
public override void SetControllerRenderModelWheel(GameObject renderModel,
bool state)
```

- Parameters

- `GameObject renderModel` - The `GameObject` containing the controller render model.
- `bool state` - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.

- Returns

- *none*

The `SetControllerRenderModelWheel` method sets the state of the scroll wheel on the controller render model.

HapticPulse/2

```
public override void HapticPulse(VRTK_ControllerReference
controllerReference, float strength = 0.5f)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public override bool HapticPulse(VRTK_ControllerReference
controllerReference, AudioClip clip)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `AudioClip clip` - The audio clip to use for the haptic pattern.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public override SDK_ControllerHapticModifiers GetHapticModifiers()
```

- Parameters

- *none*

- Returns

- SDK_ControllerHapticModifiers - An SDK_ControllerHapticModifiers object with a given durationModifier and an intervalModifier.

The GetHapticModifiers method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public override Vector3 GetVelocity(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- VRTK_ControllerReference controllerReference - The reference to the tracked object to check for.

- Returns

- Vector3 - A Vector3 containing the current velocity of the tracked object.

The GetVelocity method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public override Vector3 GetAngularVelocity(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- VRTK_ControllerReference controllerReference - The reference to the tracked object to check for.

- Returns

- Vector3 - A Vector3 containing the current angular velocity of the tracked object.

The GetAngularVelocity method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public override bool IsTouchpadStatic(bool isTouched, Vector2
currentAxisValues, Vector2 previousAxisValues, int compareFidelity)
```

- **Parameters**

- `Vector2 currentAxisValues` -
- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- **Returns**

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public override Vector2 GetButtonAxis(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- **Parameters**

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.

- **Returns**

- `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public override float GetButtonHairlineDelta(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- **Parameters**

- `ButtonTypes buttonType` - The type of button to get the hairline delta for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.

- **Returns**

- `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public override bool GetControllerButtonState(ButtonTypes buttonType,  
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the state of.
- `ButtonPressTypes pressType` - The button state to check for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.

- Returns

- `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the given press type on the given controller reference is currently taking place.

Oculus Boundaries (SDK_OculusBoundaries)

Overview

The Oculus Boundaries SDK script provides a bridge to the Oculus SDK play area.

Class Methods

InitBoundaries/0

```
public override void InitBoundaries()
```

- Parameters

- *none*

- Returns

- *none*

The `InitBoundaries` method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public override Transform GetPlayArea()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the play area in the scene.

The `GetPlayArea` method returns the `Transform` of the object that is used to represent the play area in the scene.

GetPlayAreaVertices/0

```
public override Vector3[] GetPlayAreaVertices()
```

- Parameters
 - *none*
- Returns
 - `Vector3[]` - A `Vector3` array of the points in the scene that represent the play area boundaries.

The `GetPlayAreaVertices` method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public override float GetPlayAreaBorderThickness()
```

- Parameters
 - *none*
- Returns
 - `float` - The thickness of the drawn border.

The `GetPlayAreaBorderThickness` returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public override bool IsPlayAreaSizeCalibrated()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the play area size has been auto calibrated and set by external sensors.

The `IsPlayAreaSizeCalibrated` method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public override bool GetDrawAtRuntime()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the drawn border is being displayed.

The `GetDrawAtRuntime` method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public override void SetDrawAtRuntime(bool value)
```

- Parameters
 - `bool value` - The state of whether the drawn border should be displayed or not.
- Returns
 - *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.

GetAvatar/0

```
public virtual OvrAvatar GetAvatar()
```

- Parameters
 - *none*
- Returns
 - `OvrAvatar` - The `OvrAvatar` script for managing the Oculus Avatar.

The `GetAvatar` method is used to retrieve the Oculus Avatar object if it exists in the scene. This method is only available if the Oculus Avatar package is installed.

Daydream SDK (VRTK/SDK/Daydream)

The scripts used to utilise the Google VR SDK for Unity.

- [Daydream Defines](#)
- [Daydream System](#)
- [Daydream Headset](#)

- [Daydream Controller](#)
 - [Daydream Boundaries](#)
-

Daydream Defines (SDK_DaydreamDefines)

Overview

Handles all the scripting define symbols for the Daydream SDK.

Class Variables

- `public const string ScriptingDefineSymbol` - The scripting define symbol for the Daydream SDK. Default: `SDK_ScriptingDefineSymbolPredicateAttribute.RemovableSymbolPrefix + "SDK_DAYDREAM"`
-

Daydream System (SDK_DaydreamSystem)

Overview

The Daydream System SDK script provides dummy functions for system functions.

Class Methods

IsDisplayOnDesktop/0

```
public override bool IsDisplayOnDesktop()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public override bool ShouldAppRenderWithLowResources()
```

- Parameters
 - *none*
- Returns

- `bool` - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public override void ForceInterleavedReprojectionOn(bool force)
```

- Parameters

- `bool force` - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.

- Returns

- *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

Daydream Headset (SDK_DaydreamHeadset)

Overview

The Daydream Headset SDK script provides dummy functions for the headset.

Class Methods

ProcessUpdate/1

```
public override void ProcessUpdate(Dictionary<string, object> options)
```

- Parameters

- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The `ProcessUpdate` method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public override void ProcessFixedUpdate(Dictionary<string, object> options)
```

- Parameters

- Dictionary<string, object> options - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetHeadset/0

```
public override Transform GetHeadset()
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object representing the headset in the scene.

The GetHeadset method returns the Transform of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public override Transform GetHeadsetCamera()
```

- Parameters

- *none*

- Returns

- Transform - A transform of the object holding the headset camera in the scene.

The GetHeadsetCamera/0 method returns the Transform of the object that is used to hold the headset camera in the scene.

GetHeadsetVelocity/0

```
public override Vector3 GetHeadsetVelocity()
```

- Parameters

- *none*

- Returns

- Vector3 - A Vector3 containing the current velocity of the headset.

The GetHeadsetVelocity method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public override Vector3 GetHeadsetAngularVelocity()
```

- Parameters

- *none*

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

HeadsetFade/3

```
public override void HeadsetFade(Color color, float duration, bool  
fadeOverlay = false)
```

- Parameters

- `Color color` - The colour to fade to.
- `float duration` - The amount of time the fade should take to reach the given colour.
- `bool fadeOverlay` - Determines whether to use an overlay on the fade.

- Returns

- *none*

The `HeadsetFade` method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public override bool HasHeadsetFade(Transform obj)
```

- Parameters

- `Transform obj` - The `Transform` to check to see if a camera fade is available on.

- Returns

- `bool` - Returns true if the headset has fade functionality on it.

The `HasHeadsetFade` method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public override void AddHeadsetFade(Transform camera)
```

- Parameters
 - `Transform camera` - The Transform to with the camera on to add the fade functionality to.
- Returns
 - *none*

The `AddHeadsetFade` method attempts to add the fade functionality to the game object with the camera on it.

Daydream Controller (SDK_DaydreamController)

Overview

The Daydream Controller SDK script provides a bridge to SDK methods that deal with the input devices.

Class Methods

ProcessUpdate/2

```
public override void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference for the controller.
 - `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.
- Returns
 - *none*

The `ProcessUpdate` method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public override void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference for the controller.
 - `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.
- Returns
 - *none*

The `ProcessFixedUpdate` method enables an SDK to run logic for every Unity `FixedUpdate`

GetCurrentControllerType/0

```
public override ControllerType GetCurrentControllerType()
```

- Parameters
 - *none*
- Returns
 - `ControllerType` - The `ControllerType` based on the SDK and headset being used.

The `GetCurrentControllerType` method returns the current used `ControllerType` based on the SDK and headset being used.

GetControllerDefaultColliderPath/1

```
public override string GetControllerDefaultColliderPath(ControllerHand hand)
```

- Parameters
 - `ControllerHand hand` - The controller hand to check for
- Returns
 - `string` - A path to the resource that contains the collider `GameObject`.

The `GetControllerDefaultColliderPath` returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public override string GetControllerElementPath(ControllerElements element,  
ControllerHand hand, bool fullPath = false)
```

- Parameters
 - `ControllerElements element` - The controller element to look up.
 - `ControllerHand hand` - The controller hand to look up.
 - `bool fullPath` - Whether to get the initial path or the full path to the element.
- Returns
 - `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public override uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public override GameObject GetControllerByIndex(uint index, bool actual = false)
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` -

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public override Transform GetControllerOrigin(VRTK_ControllerReference controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A `Transform` containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public override Transform GenerateControllerPointerOrigin(GameObject parent)
```

- Parameters

- `GameObject parent` - The `GameObject` that the origin will become parent of. If it is a controller

then it will also be used to determine the hand if required.

- Returns
 - `Transform` - A generated Transform that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin Transform to represent the pointer position and forward.

GetControllerLeftHand/1

```
public override GameObject GetControllerLeftHand(bool actual = false)
```

- Parameters
 - `bool actual` - If true it will return the actual controller, if false it will return the script alias controller GameObject.
- Returns
 - `GameObject` - The GameObject containing the left hand controller.

The `GetControllerLeftHand` method returns the GameObject containing the representation of the left hand controller.

GetControllerRightHand/1

```
public override GameObject GetControllerRightHand(bool actual = false)
```

- Parameters
 - `bool actual` - If true it will return the actual controller, if false it will return the script alias controller GameObject.
- Returns
 - `GameObject` - The GameObject containing the right hand controller.

The `GetControllerRightHand` method returns the GameObject containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public override bool IsControllerLeftHand(GameObject controller)
```

- Parameters
 - `GameObject controller` - The GameObject to check.
- Returns
 - `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand

controller.

IsControllerRightHand/1

```
public override bool IsControllerRightHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public override bool IsControllerLeftHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public override bool IsControllerRightHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public override GameObject GetControllerModel(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public override GameObject GetControllerModel(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The hand enum of which controller model to retrieve.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given controller hand.

GetControllerRenderModel/1

```
public override GameObject GetControllerRenderModel(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to check.

- Returns

- `GameObject` - A `GameObject` containing the object that has a render model for the controller.

The `GetControllerRenderModel` method gets the game object that contains the given controller's render model.

SetControllerRenderModelWheel/2

```
public override void SetControllerRenderModelWheel(GameObject renderModel,  
bool state)
```

- Parameters

- `GameObject renderModel` - The `GameObject` containing the controller render model.

- `bool state` - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.
- Returns
 - *none*

The `SetControllerRenderModelWheel` method sets the state of the scroll wheel on the controller render model.

HapticPulse/2

```
public override void HapticPulse(VRTK_ControllerReference
    controllerReference, float strength = 0.5f)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
 - `float strength` - The intensity of the rumble of the controller motor. 0 to 1.
- Returns
 - *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public override bool HapticPulse(VRTK_ControllerReference
    controllerReference, AudioClip clip)
```

- Parameters
 - `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
 - `AudioClip clip` - The audio clip to use for the haptic pattern.
- Returns
 - *none*

The `HapticPulse/2` method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public override SDK_ControllerHapticModifiers GetHapticModifiers()
```

- Parameters

- *none*
- Returns
 - SDK_ControllerHapticModifiers - An SDK_ControllerHapticModifiers object with a given durationModifier and an intervalModifier.

The GetHapticModifiers method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public override Vector3 GetVelocity(VRTK_ControllerReference
controllerReference)
```

- Parameters
 - VRTK_ControllerReference controllerReference - The reference to the tracked object to check for.
- Returns
 - Vector3 - A Vector3 containing the current velocity of the tracked object.

The GetVelocity method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public override Vector3 GetAngularVelocity(VRTK_ControllerReference
controllerReference)
```

- Parameters
 - VRTK_ControllerReference controllerReference - The reference to the tracked object to check for.
- Returns
 - Vector3 - A Vector3 containing the current angular velocity of the tracked object.

The GetAngularVelocity method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public override bool IsTouchpadStatic(bool isTouched, Vector2
currentAxisValues, Vector2 previousAxisValues, int compareFidelity)
```

- Parameters
 - Vector2 currentAxisValues -

- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- **Returns**

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public override Vector2 GetButtonAxis(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- **Parameters**

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.

- **Returns**

- `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public override float GetButtonHairlineDelta(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- **Parameters**

- `ButtonTypes buttonType` - The type of button to get the hairline delta for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.

- **Returns**

- `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public override bool GetControllerButtonState(ButtonTypes buttonType,
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the state of.
- `ButtonPressTypes pressType` - The button state to check for.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.

- Returns

- `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the given press type on the given controller reference is currently taking place.

Daydream Boundaries (SDK_DaydreamBoundaries)

Overview

The Daydream Boundaries SDK script provides dummy functions for the play area boundaries.

Class Methods

InitBoundaries/0

```
public override void InitBoundaries()
```

- Parameters

- *none*

- Returns

- *none*

The `InitBoundaries` method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public override Transform GetPlayArea()
```

- Parameters

- *none*

- Returns

- `Transform` - A transform of the object representing the play area in the scene.

The `GetPlayArea` method returns the `Transform` of the object that is used to represent the play area

in the scene.

GetPlayAreaVertices/0

```
public override Vector3[] GetPlayAreaVertices()
```

- Parameters

- *none*

- Returns

- `Vector3[]` - A `Vector3` array of the points in the scene that represent the play area boundaries.

The `GetPlayAreaVertices` method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public override float GetPlayAreaBorderThickness()
```

- Parameters

- *none*

- Returns

- `float` - The thickness of the drawn border.

The `GetPlayAreaBorderThickness` returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public override bool IsPlayAreaSizeCalibrated()
```

- Parameters

- *none*

- Returns

- `bool` - Returns true if the play area size has been auto calibrated and set by external sensors.

The `IsPlayAreaSizeCalibrated` method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public override bool GetDrawAtRuntime()
```

- Parameters

- *none*

- Returns
 - `bool` - Returns true if the drawn border is being displayed.

The `GetDrawAtRuntime` method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public override void SetDrawAtRuntime(bool value)
```

- Parameters
 - `bool value` - The state of whether the drawn border should be displayed or not.
- Returns
 - *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.

Ximmerse SDK (VRTK/SDK/Ximmerse)

The scripts used to utilise the Ximmerse SDK for Unity.

- [Ximmerse Defines](#)
- [Ximmerse System](#)
- [Ximmerse Headset](#)
- [Ximmerse Controller](#)
- [Ximmerse Boundaries](#)

Ximmerse Defines (SDK_XimmerseDefines)

Overview

Handles all the scripting define symbols for the Ximmerse SDK.

Class Variables

- `public const string ScriptingDefineSymbol` - The scripting define symbol for the Ximmerse SDK. Default: `SDK_ScriptingDefineSymbolPredicateAttribute.RemovableSymbolPrefix + "SDK_XIMMERSE"`

Ximmerse System (SDK_XimmerseSystem)

Overview

The Ximmerse System SDK script provides a bridge to the Ximmerse SDK.

Class Methods

IsDisplayOnDesktop/0

```
public override bool IsDisplayOnDesktop()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the display is extending the desktop

The `IsDisplayOnDesktop` method returns true if the display is extending the desktop.

ShouldAppRenderWithLowResources/0

```
public override bool ShouldAppRenderWithLowResources()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the Unity app should render with low resources.

The `ShouldAppRenderWithLowResources` method is used to determine if the Unity app should use low resource mode. Typically true when the dashboard is showing.

ForceInterleavedReprojectionOn/1

```
public override void ForceInterleavedReprojectionOn(bool force)
```

- Parameters
 - `bool force` - If true then Interleaved Reprojection will be forced on, if false it will not be forced on.
- Returns
 - *none*

The `ForceInterleavedReprojectionOn` method determines whether Interleaved Reprojection should be forced on or off.

Ximmerse Headset (SDK_XimmerseHeadset)

Overview

The Ximmerse Headset SDK script provides a bridge to the Ximmerse SDK.

Class Methods

ProcessUpdate/1

```
public override void ProcessUpdate(Dictionary<string, object> options)
```

- Parameters
 - `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.
- Returns
 - *none*

The `ProcessUpdate` method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/1

```
public override void ProcessFixedUpdate(Dictionary<string, object> options)
```

- Parameters
 - `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.
- Returns
 - *none*

The `ProcessFixedUpdate` method enables an SDK to run logic for every Unity FixedUpdate

GetHeadsetVelocity/0

```
public override Vector3 GetHeadsetVelocity()
```

- Parameters
 - *none*

- Returns
 - `Vector3` - A `Vector3` containing the current velocity of the headset.

The `GetHeadsetVelocity` method is used to determine the current velocity of the headset.

GetHeadsetAngularVelocity/0

```
public override Vector3 GetHeadsetAngularVelocity()
```

- Parameters
 - *none*
- Returns
 - `Vector3` - A `Vector3` containing the current angular velocity of the headset.

The `GetHeadsetAngularVelocity` method is used to determine the current angular velocity of the headset.

GetHeadset/0

```
public override Transform GetHeadset()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the headset in the scene.

The `GetHeadset` method returns the `Transform` of the object that is used to represent the headset in the scene.

GetHeadsetCamera/0

```
public override Transform GetHeadsetCamera()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object holding the headset camera in the scene.

The `GetHeadsetCamera` method returns the `Transform` of the object that is used to hold the headset camera in the scene.

HeadsetFade/3


```
public override void HeadsetFade(Color color, float duration, bool
fadeOverlay = false)
```

- Parameters

- `Color color` - The colour to fade to.
- `float duration` - The amount of time the fade should take to reach the given colour.
- `bool fadeOverlay` - Determines whether to use an overlay on the fade.

- Returns

- *none*

The `HeadsetFade` method is used to apply a fade to the headset camera to progressively change the colour.

HasHeadsetFade/1

```
public override bool HasHeadsetFade(Transform obj)
```

- Parameters

- `Transform obj` - The Transform to check to see if a camera fade is available on.

- Returns

- `bool` - Returns true if the headset has fade functionality on it.

The `HasHeadsetFade` method checks to see if the given game object (usually the camera) has the ability to fade the viewpoint.

AddHeadsetFade/1

```
public override void AddHeadsetFade(Transform camera)
```

- Parameters

- `Transform camera` - The Transform to with the camera on to add the fade functionality to.

- Returns

- *none*

The `AddHeadsetFade` method attempts to add the fade functionality to the game object with the camera on it.

Ximmerse Controller (SDK_XimmerseController)

Overview

The Ximmerse Controller SDK script provides a bridge to SDK methods that deal with the input devices.

Class Methods

ProcessUpdate/2

```
public override void ProcessUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the update.

- Returns

- *none*

The ProcessUpdate method enables an SDK to run logic for every Unity Update

ProcessFixedUpdate/2

```
public override void ProcessFixedUpdate(VRTK_ControllerReference  
controllerReference, Dictionary<string, object> options)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference for the controller.
- `Dictionary<string, object> options` - A dictionary of generic options that can be used to within the fixed update.

- Returns

- *none*

The ProcessFixedUpdate method enables an SDK to run logic for every Unity FixedUpdate

GetCurrentControllerType/0

```
public override ControllerType GetCurrentControllerType()
```

- Parameters

- *none*

- Returns

- `ControllerType` - The ControllerType based on the SDK and headset being used.

The GetCurrentControllerType method returns the current used ControllerType based on the SDK

and headset being used.

GetControllerDefaultColliderPath/1

```
public override string GetControllerDefaultColliderPath(ControllerHand hand)
```

- Parameters

- `ControllerHand hand` - The controller hand to check for

- Returns

- `string` - A path to the resource that contains the collider `GameObject`.

The `GetControllerDefaultColliderPath` returns the path to the prefab that contains the collider objects for the default controller of this SDK.

GetControllerElementPath/3

```
public override string GetControllerElementPath(ControllerElements element,  
ControllerHand hand, bool fullPath = false)
```

- Parameters

- `ControllerElements element` - The controller element to look up.
- `ControllerHand hand` - The controller hand to look up.
- `bool fullPath` - Whether to get the initial path or the full path to the element.

- Returns

- `string` - A string containing the path to the game object that the controller element resides in.

The `GetControllerElementPath` returns the path to the game object that the given controller element for the given hand resides in.

GetControllerIndex/1

```
public override uint GetControllerIndex(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` containing the controller.

- Returns

- `uint` - The index of the given controller.

The `GetControllerIndex` method returns the index of the given controller.

GetControllerByIndex/2

```
public override GameObject GetControllerByIndex(uint index, bool actual =
```

```
false)
```

- Parameters

- `uint index` - The index of the controller to find.
- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` -

The `GetControllerByIndex` method returns the `GameObject` of a controller with a specific index.

GetControllerOrigin/1

```
public override Transform GetControllerOrigin(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the controller to retrieve the origin from.

- Returns

- `Transform` - A `Transform` containing the origin of the controller.

The `GetControllerOrigin` method returns the origin of the given controller.

GenerateControllerPointerOrigin/1

```
public override Transform GenerateControllerPointerOrigin(GameObject parent)
```

- Parameters

- *none*

- Returns

- `Transform` - A generated `Transform` that contains the custom pointer origin.

The `GenerateControllerPointerOrigin` method can create a custom pointer origin `Transform` to represent the pointer position and forward.

GetControllerLeftHand/1

```
public override GameObject GetControllerLeftHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias

controller GameObject.

- Returns

- `GameObject` - The `GameObject` containing the left hand controller.

The `GetControllerLeftHand` method returns the `GameObject` containing the representation of the left hand controller.

GetControllerRightHand/1

```
public override GameObject GetControllerRightHand(bool actual = false)
```

- Parameters

- `bool actual` - If true it will return the actual controller, if false it will return the script alias controller `GameObject`.

- Returns

- `GameObject` - The `GameObject` containing the right hand controller.

The `GetControllerRightHand` method returns the `GameObject` containing the representation of the right hand controller.

IsControllerLeftHand/1

```
public override bool IsControllerLeftHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/1` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/1

```
public override bool IsControllerRightHand(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/1` method is used to check if the given controller is the the right hand controller.

IsControllerLeftHand/2

```
public override bool IsControllerLeftHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the left hand controller.

The `IsControllerLeftHand/2` method is used to check if the given controller is the the left hand controller.

IsControllerRightHand/2

```
public override bool IsControllerRightHand(GameObject controller, bool actual)
```

- Parameters

- `GameObject controller` - The `GameObject` to check.
- `bool actual` - If true it will check the actual controller, if false it will check the script alias controller.

- Returns

- `bool` - Returns true if the given controller is the right hand controller.

The `IsControllerRightHand/2` method is used to check if the given controller is the the right hand controller.

GetControllerModel/1

```
public override GameObject GetControllerModel(GameObject controller)
```

- Parameters

- `GameObject controller` - The `GameObject` to get the model alias for.

- Returns

- `GameObject` - The `GameObject` that has the model alias within it.

The `GetControllerModel` method returns the model alias for the given `GameObject`.

GetControllerModel/1

```
public override GameObject GetControllerModel(ControllerHand hand)
```

- Parameters

- ControllerHand hand - The hand enum of which controller model to retrieve.

- Returns

- GameObject - The GameObject that has the model alias within it.

The GetControllerModel method returns the model alias for the given controller hand.

GetControllerRenderModel/1

```
public override GameObject GetControllerRenderModel(VRTK_ControllerReference  
controllerReference)
```

- Parameters

- VRTK_ControllerReference controllerReference - The reference to the controller to check.

- Returns

- GameObject - A GameObject containing the object that has a render model for the controller.

The GetControllerRenderModel method gets the game object that contains the given controller's render model.

SetControllerRenderModelWheel/2

```
public override void SetControllerRenderModelWheel(GameObject renderModel,  
bool state)
```

- Parameters

- GameObject renderModel - The GameObject containing the controller render model.
- bool state - If true and the render model has a scroll wheel then it will be displayed, if false then the scroll wheel will be hidden.

- Returns

- *none*

The SetControllerRenderModelWheel method sets the state of the scroll wheel on the controller render model.

HapticPulse/2

```
public override void HapticPulse(VRTK_ControllerReference  
controllerReference, float strength = 0.5f)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `float strength` - The intensity of the rumble of the controller motor. 0 to 1.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a simple haptic pulse on the tracked object of the given controller reference.

HapticPulse/2

```
public override bool HapticPulse(VRTK_ControllerReference  
controllerReference, AudioClip clip)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to initiate the haptic pulse on.
- `AudioClip clip` - The audio clip to use for the haptic pattern.

- Returns

- *none*

The `HapticPulse/2` method is used to initiate a haptic pulse based on an audio clip on the tracked object of the given controller reference.

GetHapticModifiers/0

```
public override SDK_ControllerHapticModifiers GetHapticModifiers()
```

- Parameters

- *none*

- Returns

- `SDK_ControllerHapticModifiers` - An `SDK_ControllerHapticModifiers` object with a given `durationModifier` and an `intervalModifier`.

The `GetHapticModifiers` method is used to return modifiers for the duration and interval if the SDK handles it slightly differently.

GetVelocity/1

```
public override Vector3 GetVelocity(VRTK_ControllerReference  
controllerReference)
```


- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current velocity of the tracked object.

The `GetVelocity` method is used to determine the current velocity of the tracked object on the given controller reference.

GetAngularVelocity/1

```
public override Vector3 GetAngularVelocity(VRTK_ControllerReference
controllerReference)
```

- Parameters

- `VRTK_ControllerReference controllerReference` - The reference to the tracked object to check for.

- Returns

- `Vector3` - A `Vector3` containing the current angular velocity of the tracked object.

The `GetAngularVelocity` method is used to determine the current angular velocity of the tracked object on the given controller reference.

IsTouchpadStatic/4

```
public override bool IsTouchpadStatic(bool isTouched, Vector2
currentAxisValues, Vector2 previousAxisValues, int compareFidelity)
```

- Parameters

- `Vector2 currentAxisValues` -
- `Vector2 previousAxisValues` -
- `int compareFidelity` -

- Returns

- `bool` - Returns true if the touchpad is not currently being touched or moved.

The `IsTouchpadStatic` method is used to determine if the touchpad is currently not being moved.

GetButtonAxis/2

```
public override Vector2 GetButtonAxis(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- Parameters

- `ButtonTypes buttonType` - The type of button to check for the axis on.
- `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button axis on.
- Returns
 - `Vector2` - A `Vector2` of the X/Y values of the button axis. If no axis values exist for the given button, then a `Vector2.Zero` is returned.

The `GetButtonAxis` method retrieves the current X/Y axis values for the given button type on the given controller reference.

GetButtonHairlineDelta/2

```
public override float GetButtonHairlineDelta(ButtonTypes buttonType,
VRTK_ControllerReference controllerReference)
```

- Parameters
 - `ButtonTypes buttonType` - The type of button to get the hairline delta for.
 - `VRTK_ControllerReference controllerReference` - The reference to the controller to get the hairline delta for.
- Returns
 - `float` - The delta between the button presses.

The `GetButtonHairlineDelta` method is used to get the difference between the current button press and the previous frame button press.

GetControllerButtonState/3

```
public override bool GetControllerButtonState(ButtonTypes buttonType,
ButtonPressTypes pressType, VRTK_ControllerReference controllerReference)
```

- Parameters
 - `ButtonTypes buttonType` - The type of button to check for the state of.
 - `ButtonPressTypes pressType` - The button state to check for.
 - `VRTK_ControllerReference controllerReference` - The reference to the controller to check the button state on.
- Returns
 - `bool` - Returns true if the given button is in the state of the given press type on the given controller reference.

The `GetControllerButtonState` method is used to determine if the given controller button for the given press type on the given controller reference is currently taking place.

Ximmerse Boundaries (SDK_XimmerseBoundaries)

Overview

The Ximmerse Boundaries SDK script provides a bridge to the Ximmerse SDK play area.

Class Methods

InitBoundaries/0

```
public override void InitBoundaries()
```

- Parameters
 - *none*
- Returns
 - *none*

The InitBoundaries method is run on start of scene and can be used to initialise anything on game start.

GetPlayArea/0

```
public override Transform GetPlayArea()
```

- Parameters
 - *none*
- Returns
 - `Transform` - A transform of the object representing the play area in the scene.

The GetPlayArea method returns the Transform of the object that is used to represent the play area in the scene.

GetPlayAreaVertices/0

```
public override Vector3[] GetPlayAreaVertices()
```

- Parameters
 - *none*
- Returns
 - `Vector3[]` - A Vector3 array of the points in the scene that represent the play area boundaries.

The `GetPlayAreaVertices` method returns the points of the play area boundaries.

GetPlayAreaBorderThickness/0

```
public override float GetPlayAreaBorderThickness()
```

- Parameters
 - *none*
- Returns
 - `float` - The thickness of the drawn border.

The `GetPlayAreaBorderThickness` returns the thickness of the drawn border for the given play area.

IsPlayAreaSizeCalibrated/0

```
public override bool IsPlayAreaSizeCalibrated()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the play area size has been auto calibrated and set by external sensors.

The `IsPlayAreaSizeCalibrated` method returns whether the given play area size has been auto calibrated by external sensors.

GetDrawAtRuntime/0

```
public override bool GetDrawAtRuntime()
```

- Parameters
 - *none*
- Returns
 - `bool` - Returns true if the drawn border is being displayed.

The `GetDrawAtRuntime` method returns whether the given play area drawn border is being displayed.

SetDrawAtRuntime/1

```
public override void SetDrawAtRuntime(bool value)
```

- Parameters

- `bool value` - The state of whether the drawn border should be displayed or not.
- Returns
 - *none*

The `SetDrawAtRuntime` method sets whether the given play area drawn border should be displayed at runtime.
