VR Easy Getting Started V1.6

Introduction

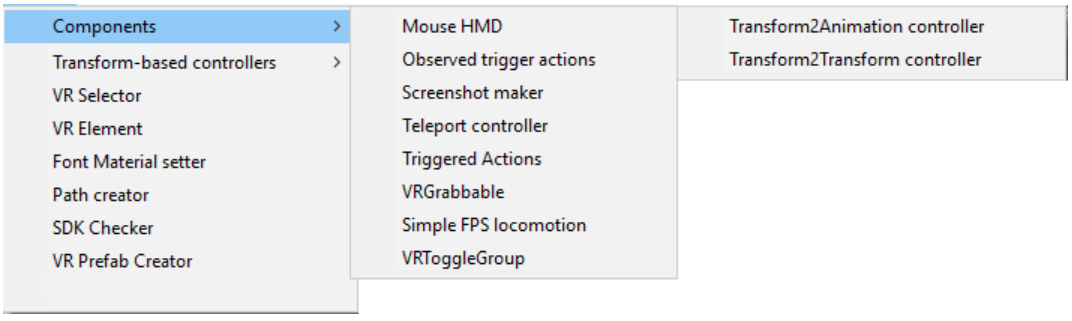
Over the last several years, Virtual Reality (VR) has taken a huge leap in terms development and usage, especially to the tools and affordability that game engine Unity is offering. This is a great opportunity for developers to build such VR games and applications. For non-developers this development process is more painful because most of the time even creating basic GUI elements and interacting with your scene is very cumbersome. With VR Easy we wanted to help the non-programmers and even programmers to build VR applications with drag and drop user interface and interact with your Selector or Touch Controller eg. Vive controller and Leap Motion.

VR Easy has been built with what we called the *Selector Selectable metaphor*. A **Selector** is an object in the scene that is able to interact with VR ready elements; those elements are called **Selectables**. Selectors and Selectables can take many shapes and forms, but at their core they are just objects that can interact, and fruit of that interaction, an **action** (or list of) is triggered. The level of interaction, the nature of the Selectable and the type of Selector is defined by the specific Selector / Selectable used. There are 4 types of Selectors, based on how they interact with Selectables:

1) Sight Selector
2) Touch Selector
3) Pointer Selector
4) Mixed Reality

VR Easy comes with a wide variety of Selectable objects such as 2D and 3D buttons, sliders, display buttons and trigger areas (see VR Element section for more info). In this guide, we will walk you through how to setup each of the core areas of VR Easy:

1. **Setting up a selector (eg. SightSelector);**
2. **Setting up a selectable (eg. button);**
3. **Adding and configuring actions on a Selectable.**



This manual describes all the functions in the menus. Separate video tutorials are available to explain the core functionalities of VR Easy and can be found here.

For any questions or help please contact: support@avrworks.com

Supported VR Platforms

Below table shows the currently supported VR platforms by **VR Easy 1.6**. *VR Easy* gives the user the option to use external SDKs or use our generic camera rig for natively Unity3d supported platforms without importing any external SDKs.

VR Platforms	Headset Supported	Controller Supported	Version	Selector Type
HTC Vive/Focus/Pro	Yes	Yes	SteamVR 2.x	Sight/Pointer/Touch
Oculus Rift	Yes	Yes	Oculus Integration 1.37	Sight/Pointer/Touch
Oculus Rift S	Yes	Yes	Oculus Integration 1.37	Sight/Pointer/Touch
Oculus Go	Yes	Yes	Oculus Integration 1.37	Sight/Pointer/Touch
Oculus Quest	Yes	Yes	Oculus Integration 1.37	Sight/Pointer/Touch
Gear VR	Yes	Yes	Oculus Integration 1.37	Sight/Pointer
Google Cardboard	Yes	Yes	GVR SDK v1.200.0	Sight
Google Daydream	Yes	Yes	GVR SDK v1.200.0	Sight/Pointer
Windows MR	Yes	Yes	Native	Sight/Pointer
Leap Motion			Core Assets 4.4.0	Touch

Unity3d native support is available for the following VR platforms and supported by *VR Easy Generic Camera Rig*:

- Oculus Rift
- Gear VR
- PlayStation VR
- Microsoft HoloLens
- Steam VR
- Google Daydream

VR Easy v1.6 is fully compatible with Google Instant Preview

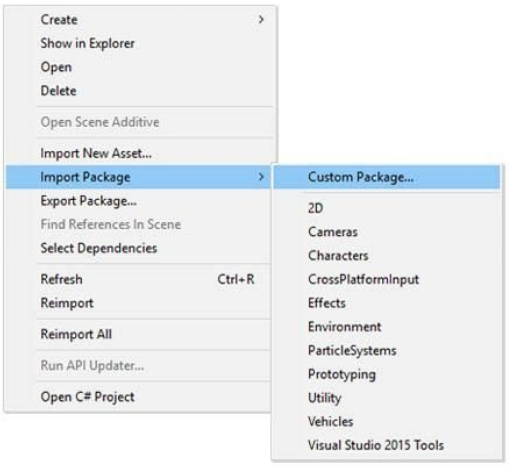
VR Easy is now fully compatible with Google Instant Preview to accelerate development in Daydream applications.

To be able to use it in conjunction with VR Easy, follow the steps:

1. Download and import the Google VR sdk from here https://developers.google.com/vr/develop/unity/download
2. Open the project in Unity and enable GoogleVR integration with SDK checker
3. Create a new scene
4. Create a player with the Prefab Creator that targets Daydream as HMD and with a pointer selector using GoogleVRControllerTrigger
5. Drag and drop the GvrInstantPreviewMain from the GoogleVR SDK to the scene that you can find on GoogleVR>Prefabs>InstantPreview
6. Follow the steps 3 and 4 on Getting started here to run the app on the phone whilst on the Unity Editor https://developers.google.com/vr/develop/unity/instant-preview

Installation

Double click on the VREasy.unitypackage file and click import or right-click in the Project window in the folder you want to import VR Easy. This process is also shown in video tutorial which you can find
here:

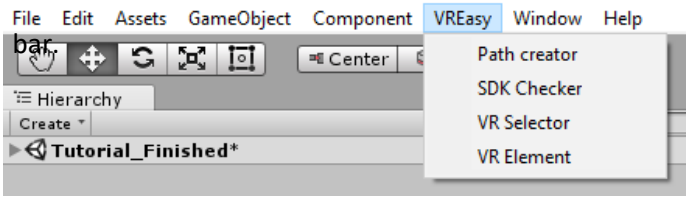


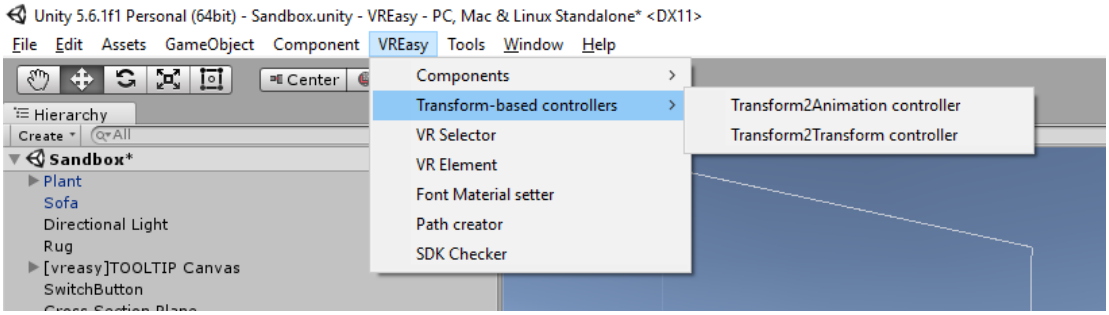
VR Easy Menus

An introduction to VR Easy Menus can be found here

Contextual menu for all VR Easy functionality

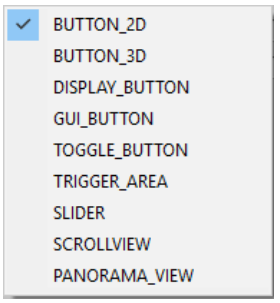
For convenience, *VR Easy* exposes all its script-based functionality via a menu in the editor top bar.





Note that GUI-based functionality (such as VR elements, selectors, Path creator and SDK checker) are accessible through the same menu.

VR Element Window



The VR Element options are the type of VR elements you can create for your VR application to interact with your scene. All VR element components support multi-object editing, which means you can set properties across objects in one go by selecting multiple elements and inspect the components.
These are the type of elements you can create:

Button_2D: Will give you a 2D Button with a canvas and text based on your scale, icons and name of your button

Button_3D: Will turn any GameObject in your scene into a interactive button with all the same functionalities of a Button_2D

Display_Button: This VR element is a specialized renderer object for buttons with a Switch action associated to them. It is very useful for buttons that control switchable objects (textures, meshes…)

GUI_Button: VREasy now supports interaction of all our Selectors (pointer, sight and touch) with Unity's Default GUI buttons.

Toggle_Button: A VR Toggle is an element similar to a 2D and 3D buttons, but instead of having a single set of actions associated with it that are triggered every time the button is pressed, it has two sets of actions.

Trigger_Area: Trigger areas are Box colliders with trigger properties to trigger actions/ events that you want to occur when entering or exiting these trigger areas

Slider: Sliders can be used for various purposes in your scene from controlling light properties to rotation of objects.

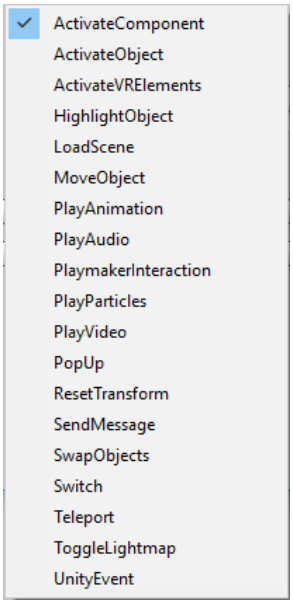
ScrollView: This VR element that allows allows the display of GUI elements in an scrollable panel.

Panorama_View: This VR element that allows users to create Panorama 360 views with ease both images and videos are supported.

Actions

This section described all the actions that are currently available in VR Easy. Actions can be used as single but you can also trigger multiple actions at the with one button or trigger. Currently we have 12 (twelve) actions and more will be added. For each of the actions there are self explanatory videos and links are included .

Available Action Types

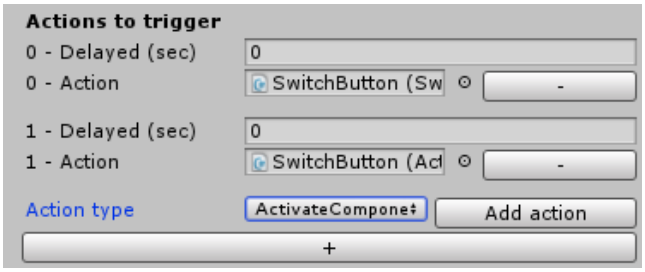


Add new slots for new actions to trigger single or multiple actions

We can add or remove extra or new actions. It is possible to trigger more than 1 action at a time

Delayed action

All actions have now a *delay* property that inserts a time (in seconds) between the activation of the element and triggering the actions. The default value is 0 (no delay) and an individual delay can be specified per action.

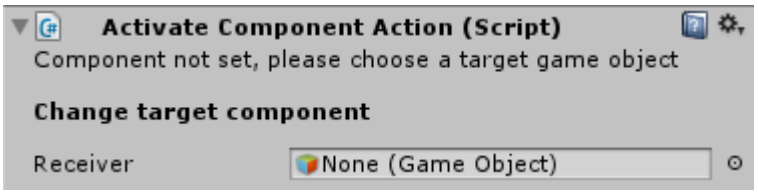


You can introduce delays in the GUI of any of the VR elements that has an action list associated with it (2D and 3D buttons, VR toggles, Trigger areas, etc.).

Action Types

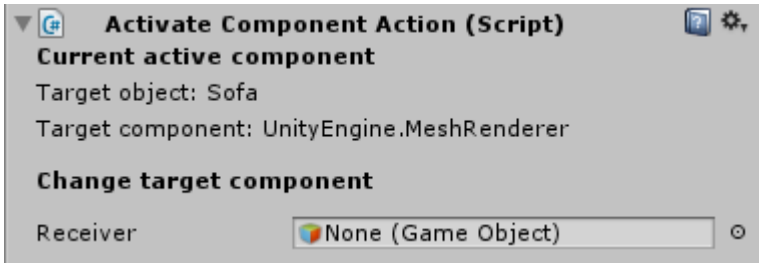
Activate Component Action

In addition to the Activate Object and Activate VR element actions, we have introduced a new action to activate specific components within a game object. Link the *ActivateComponentAction* script to any VR element (2D / 3D buttons, Trigger Areas, VR Toggles, etc.) and configure it by dragging the game object that contains the component you wish to activate / deactivate to the *Receiver* property:



Once you do so, the GUI will show a list of components that you can control. Select the one you need and click on *Set component*.

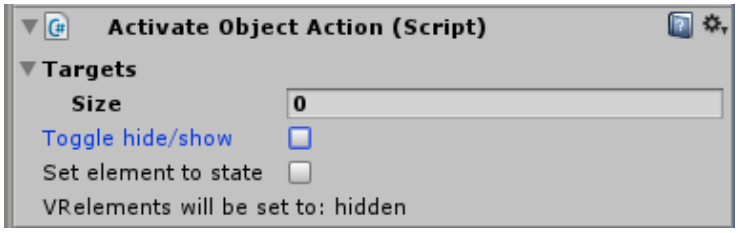
You should be able to see the *Target object* and *Target component* now linked. The *ActivateComponentAction* acts as a toggle, thus each time it is triggered the state of the component will switch, from on to off and vice versa.



We have included an example on the scene Sandbox within *SwitchButton* game object.

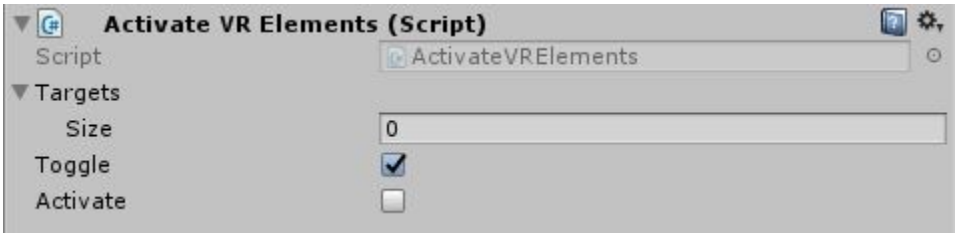
Note: Only works for components that have an *enabled* property.

ActivateObject As with ActivateVRElement action, ActivateObject actions can always activate or deactivate, in addition to toggle the state of a list of objects.



(this action is explained in video tutorial Button 2D ActivateAction which can be found here)

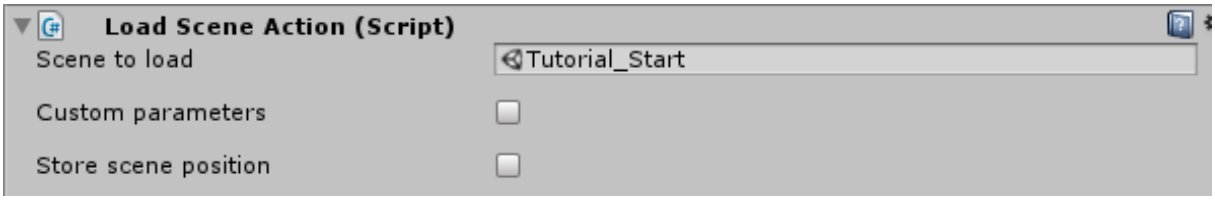
ActivateVRElements can be used to activate and deactivate VR elements. When the *toggle* option is checked, each time the action is invoked the VR element(s) will change from activate to deactivate, or vice versa. If the *toggle* option is not checked, you can set whether you want the action to always activate or deactivate the VR element by checking the *activate* option accordingly.



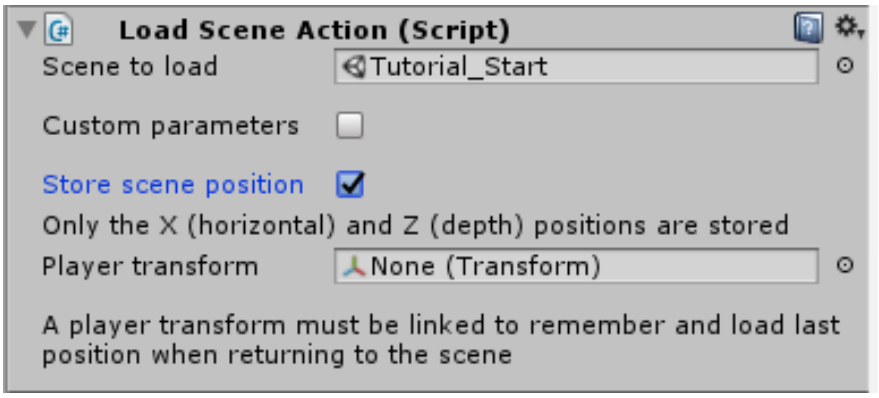
(this action is explained in video tutorial Button 2D ActivateVRElements which can be found here)

LoadScene will load the scene you have selected. Scenes are normally loaded with default parameters (delay time to load and whether to use fade in and out), but you can specify what each LoadScene action does by checking the *Use custom parameters* option.

Use custom parameters: When checked, specific options are shown to determine how to load the scene (delay time and whether to use fade in or not)

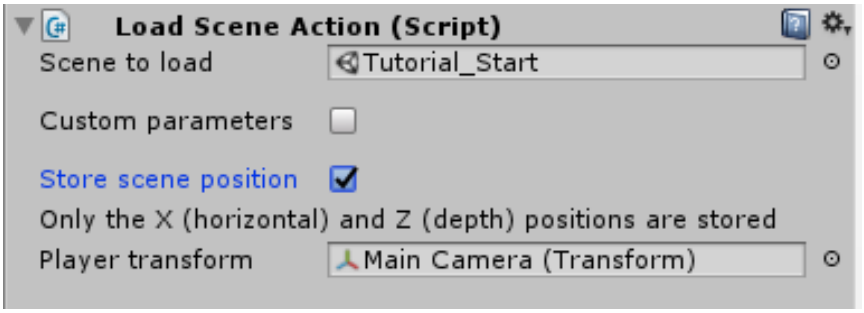


When moving from one scene to the next, now there is an option to store the last player's position so when the scene is loaded again it gets automatically reloaded.



By checking the Store scene position VR easy will store the position of the player when the Load Scene was triggered. When the previous scene is loaded again, the system will load and restore the player to that position.

Note: the position storage is temporary and will get deleted when the application is closed -but it is persistent whilst the application is alive.
You are required to provide a reference to the Player on the scene -normally the Main Camera if using Unity default support for HMD, or the Camera Rig prefab if using SteamVR plugin.



Please note that the Player transform's object must have a unique name in the scene for the system to be able to locate it upon load. VREasy will notify you if this is not the case.
(this action is explained in video tutorial Button 2D LoadScene which can be found here)

MoveObject

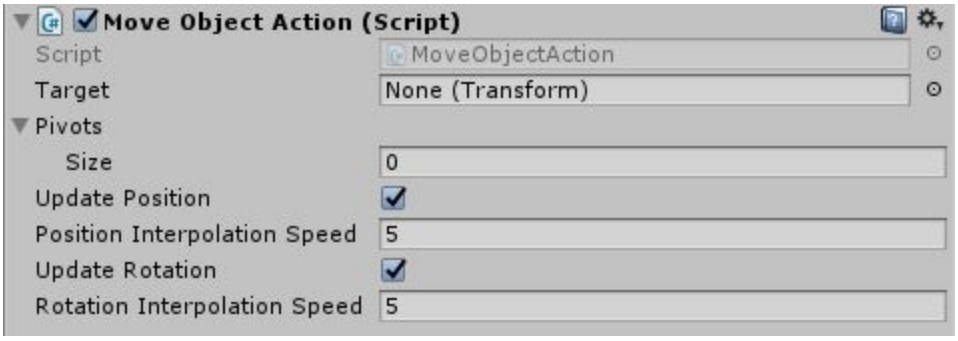
With MoveObject action you can move a target between two or more pivot positions (specified by empty game objects in your scene).

The *target* is the Object you want to move.

Pivots is a list that contains the pivot positions you want the *target* object to move to. This is a simple array and you can add as many pivots as you need.

Update position and *Update rotation* determine whether you want your object to match exclusively the position of your pivot location, or you wish to also adopt its rotation -i.e. where its facing. Since your pivots are Transforms in Unity, they have both position and rotation, allowing you to animate not only an object's position but also its rotation.

There is an extra parameter to specify the interpolation speed for both rotation and position update; this determines how fast the object is translated from one pivot to the next.

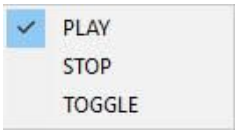


(this action is explained in video tutorial Button 2D MoveObject which can be found here)

PlayAnimation will play an animation. Currently VREasy supports the use of both the Animator and the Legacy animation systems in Unity (for more information on what are the differences and how to use each, please check out the Unity Manuals on the Animation system and the new Animator system).

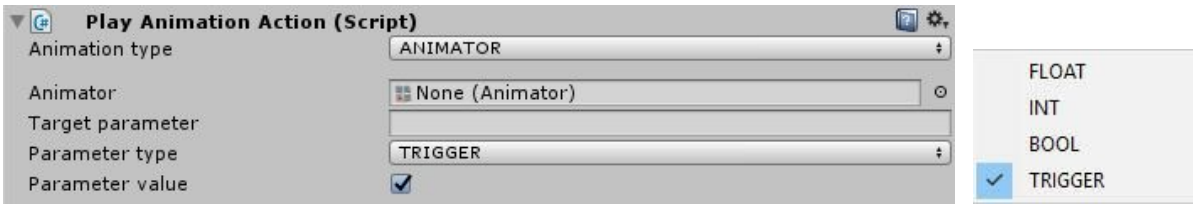
When using the *Legacy Animation*, the action allows you to specify which object you want to animate via the *Animation* component -the scene object that contains the animation you wish to trigger- and the *clip* to play, i.e. what animation to play.

Animation action gives you control over the behaviour to be triggered; you can either always play or stop a particular animation (useful if you are creating a remote control type of menu), or you can toggle between the two with a single action.



When using the Animator system the setup is similar, but you use an Animator object instead of an Animation component -the object that contains the animation should have an Animator component. Since animations using the new animator system are controlled using a state machine that transitions via parameters, your action can affect any of those parameters via the target parameter field (visit Unity tutorials for more information regarding animation parameters and state machines).

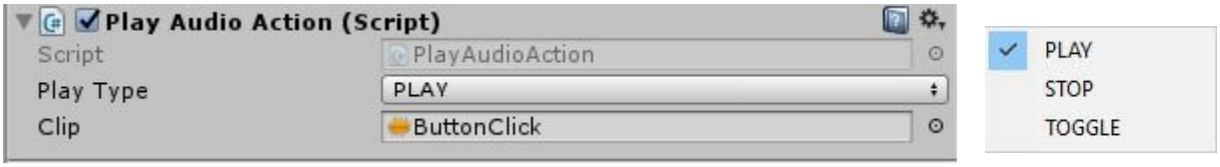
The *speed* parameter allows you to set the direction and how fast the animation is played (negative values will play it backwards!)



(this action is explained in video tutorial Button 3D PlayAnimationAction which can be found here)

PlayAudio

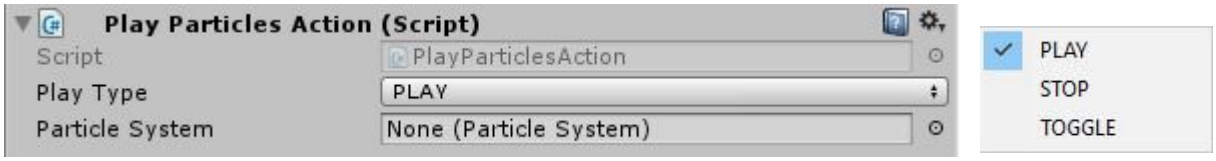
PlayAction will play an audio on the spot the VR element is.



(this action is explained in video tutorial Button 2D PlayAudio which can be found here)

PlayParticles

PlayParticles will play or disable the particle that you have selected. Simply drag and drop your particle system into the *Particle System* field and determine the *play type* (play, stop or toggle).



this action is explained in video tutorial Button 2D PlayParticles which can be found here

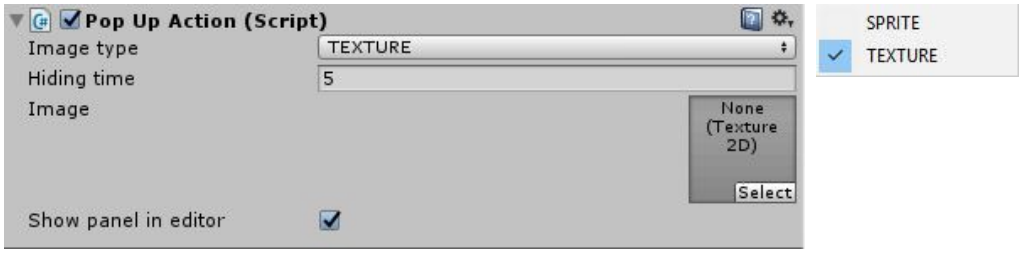
PlayMakerInteraction

Due to popular demand, we have introduced a new action that allows you to link VREasy PlayMaker events. To use it, we assume you have PlayMaker installed and configured in your Unity project. To enable interaction between the two plugins, go to SDK Checker panel and check the PlayMaker SDK box. This will allow VREasy to find PlayMaker scripts. If you do not have PlayMaker installed in your project you may see errors in your Console related to PlayMaker scripts. To fix them, either uncheck the PlayMaker SDK box or install PlayMaker.

The action that allows you to interact with PlayMaker is called *PlaymakerInteractionAction*, and can be used just as any other action (in your 2D or 3D buttons, in your grab events or as part of your trigger area actions, to name a few!). It accepts a PlayMaker FSM object and it can be used to trigger events within that FSM. Select the event from the *Playmaker event* drop down menu and click on Set Event to finish the linking.

PopUp

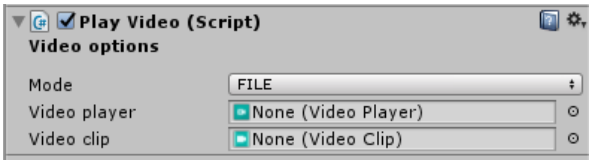
PopUp action can show a Texture or Sprite based image and with the hiding time you can control the time the PopUp will appear before hiding.



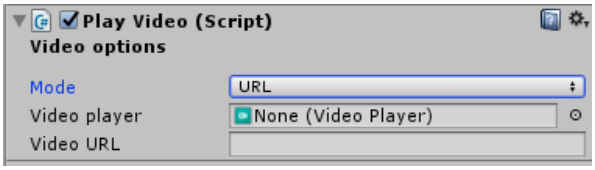
(this action is explained in video tutorial Button 2D PopUp which can be found here)

PlayVideo action

With PlayVideo action, users can play any video format supported by VideoPlayer Unity component. *Video player* property should be linked to the VideoPlayer component that needs to be controlled (can be the same object owning the action or any third object). *Video clip* property determines what video will be played when the action is triggered.

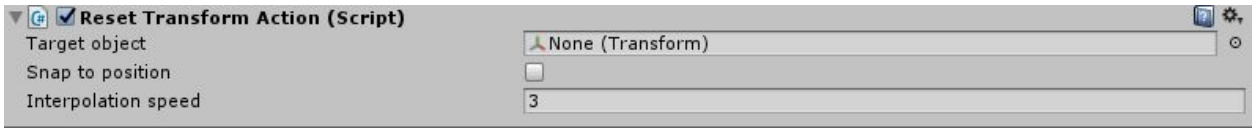


The PlayVideo action supports *FILE* and *URL* video modes. Note that the URL must point to a valid format file (see VideoPlayer Unity documentation for more information).



ResetTransform

The reset transform action is handy if you wish to have an “undo all” type of button in your scene, for instance when the user can move an object and rotate it, but you want to offer a way to roll back to initial position. The *target object* is the object you want to move to initial position. When the action is triggered, the object will either snap to its initial coordinates -if *snap to position* is checked- or will interpolate from current position to initial one -if snap is not checked.



SendMessage

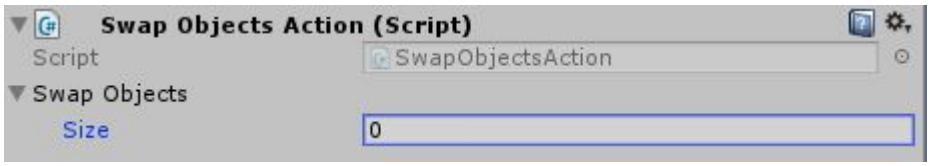
This action sends a message to a game object via the {ŜyŘ a Ŝăăl·ƎŜ function and allows you to call any* method on any script or component attached to a gameobject.
The Current active message section provides information on what message has been linked with your action -by default this will be empty.
Messages need to be linked before you are able to trigger them. To do so, you can use the change your active message section. Simply drag the target game object to the receiver field and select the component and method you wish to invoke

*current version of VREasy (v1.4) is restricted to methods with no parameters. This limitation will likely be removed in future iterations.

(this action is explained in video tutorial Button 3D SendMessage which can be found KŜNŜú

SwapObjects

This action can be used to swap objects. Swap Objects can be any type of GameObject eg. Prefabs, Meshes, Lights



(this action is explained in video tutorial Button 2D SwapObjects which can be found here)

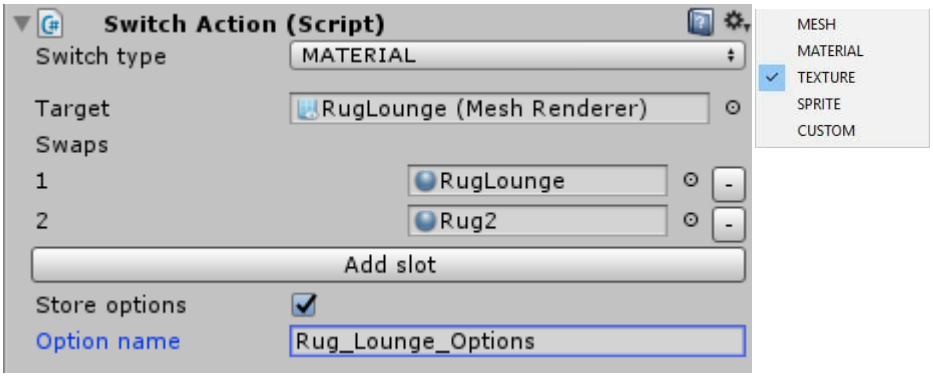
Switch

With Switch Action you can make particular properties of your game objects switch between options. We have defined some of the most commonly used properties (what users are most likely to want to switch): Mesh, Material, Texture, Sprite and Audio.

Irrespective of what you wish to switch, the action setup is the same: select a *target* (i.e. the **container** of your property) and specify the elements that will be switched (switch action will loop through each **item** each time the action is triggered). A container is the component or object that holds the property you wish to switch; here is a list of the containers for each of the switch types:

- Mesh → Mesh Filter component
- Material → Renderer component
- Texture → Material object
- Sprite → SpriteRenderer component
- Sound → Audio Source Component

The *Custom* switch type has been added for advanced purposes. It allows you to specify any container (as long as it is a Unity Object) and switch any of its public properties. Please note that this support is still experimental.



(this action is explained in video tutorial Button 2D SwitchAction which can be found here)

By enabling the option *store options*, the system will record the latest user choice in a text file called "VREasy_options.txt" (defined in VREasy_Utils.STORE_OPTIONS_FILE). Since an application can have multiple switch actions at a time, the system uses the *option name* string to identify which option is being referred to.

The *VREasy_options.txt* file follows the format:

```
[Option name 1] , [choice made by the user]

[Option name 21] , [choice made by the user]
```

Where the choice made by the user is the name of the object used last in the switch action. In the example setup above, if the user chooses the DinnerTable material, the VREasy_options.txt file will reflect the following:

```
Ceiling_option, DinnerTable
```

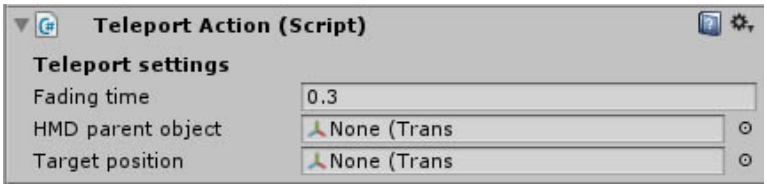
The file can be located at the application's persistent path on all platforms:
https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

Since all **VR Display Buttons** have internally switch actions, this new functionality can be used to store the user choice for them too.

A video explaining the store option can be found here

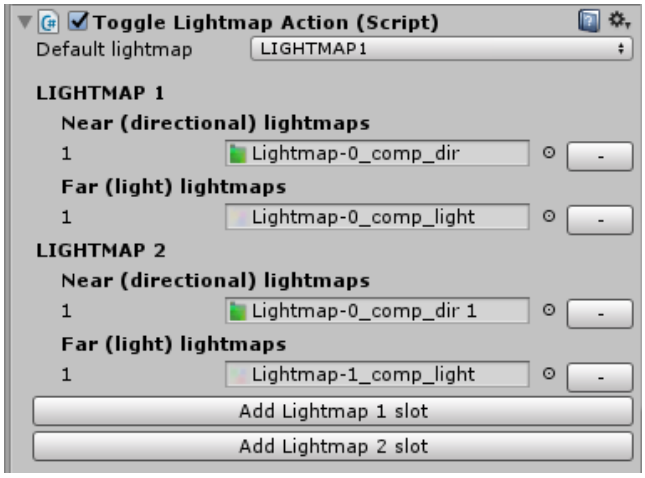
Teleport

We have added a new way of moving players around. Now, in addition to the VRSimpleFPSLocomotion and the MouseHMD, we have included a TeleportController that allows teleport-like motion through a level.



Toggle Lightmap Action

This new action allows you to toggle between lightmaps (both directional and light maps) in the current scene with ease. To use it, link the action to any VR element (2D / 3D button, toggle, etc.) and configure it by dragging your lightmaps to the appropriate slots. You can add slots to either lightmap option by pressing the buttons below.



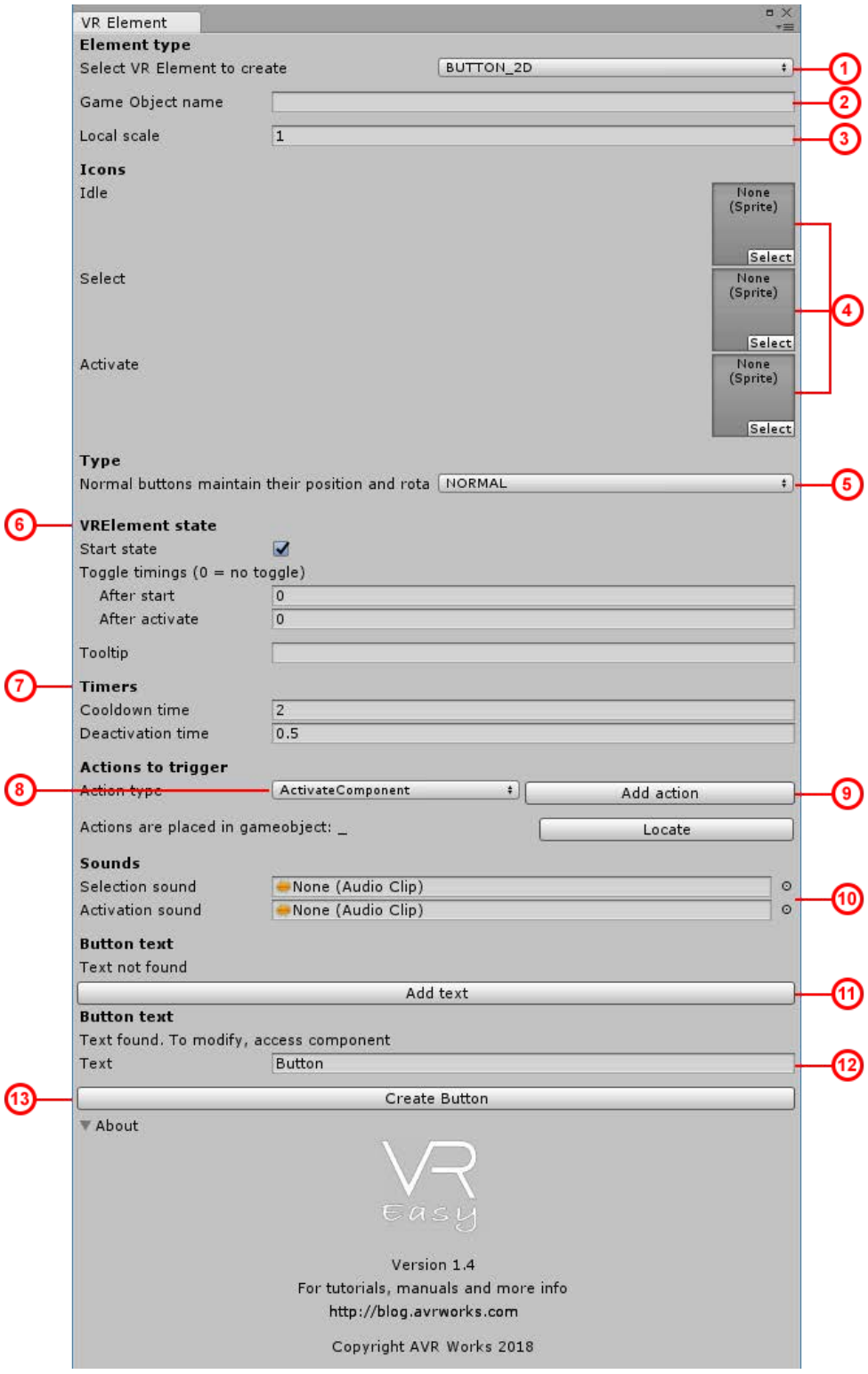
For information on how to create lightmaps on your scenes, please see the official documentation: https://docs.unity3d.com/Manual/GIIntro.html. ***Hint**: if you want to create multiple lightmaps, change the name of the filename of the current lightmap (it should be next to your scene file) and bake the lights again.*

Note: Within each of the lightmap options, the near and ***far lightmap lists must be of the same length*** (i.e. near and far lists should match in length). Lightmap 1 and lightmap 2 can have different lengths.

We have included an example on the scene LightMapSwitch, under the VR button named *LM_Switch*.

BUTTON 2D Creation Window

With Button_2D element you are able to create a 2D Button with a canvas to trigger the actions that are available in VR Easy. This chapter will go through each of the options that are available in Button_2D.



1. Type of VR Element

Button_2D will give you a 2D Button with a canvas and text based on your scale, icons and name of your button

2. Game Object Name

Type here the name of the element that you want to create and will appear in the Hierarchy

3. Scale

The scale value is is a measurement of the global size of the vr element -it roughly makes 1 = something easily selectable in a VR scene with normal scale (1 unit = 1 meter)

4. Icons

You can set there the icon for each of the states in Idle, Select and Activate

When dealing with actionable VR elements, the concept of states represents the current level of interaction between a Selector and a Selectable:

- IDLE: No interaction
- SELECT: The interaction has been initiated but the element is not yet activated
- ACTIVATE: The interaction has been completed and the element will trigger its actions

5. Type

We can set here how our button will face the camera, we have 3 options: Normal, Sticky and Billboard.

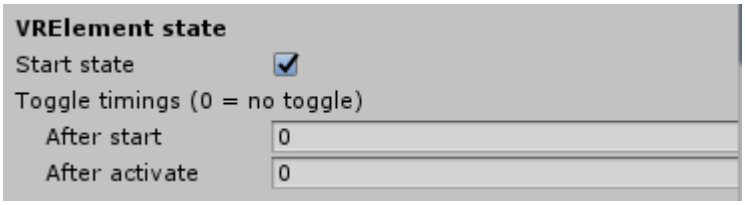
In **Normal** mode the elements will be placed in World Space mode

In **Sticky** mode elements follow the movements of the HMD and appear to be stuck to the camera.

In **Billboard** mode the elements will be placed in World Space mode always facing the camera

6. VR Elelement State

From VR Easy version 1.3 all VR elements expose a state property that can be specified. *Start state* checkbox defines whether the VR element starts enabled or disabled.



VR elements also offer the opportunity to toggle their state after two specific events:

1) *After start* box indicates how many seconds after starting the scene the element state should be toggled (0 for no toggle).
2) *After activate* indicates how many seconds after activating the element the state should be toggled (0 for no toggle).

Once a VR element is switched off, the renderer and colliders will be disabled and hence no further interaction is possible. You can however reactivate it using an Activate VR Element action.

7. Timers

See section **Timers** for more detailed description on Timers

8. Action Type

With this button you can select the type of action you want to perform with your VR Button. For more detailed description on Actions and action Types please click **here** and **here**

9. Add Action

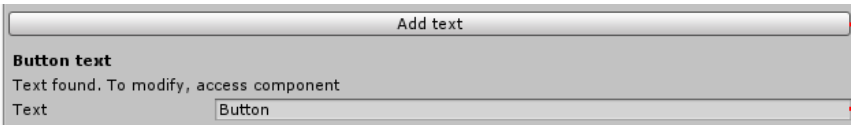
We can add or remove extra or new actions. It is possible to trigger more than 1 action at a time. Click **here** to see all available actions

10. Add Sounds

Here we can add the selection and activation sounds to our VR Elements. Audio formats that Unity can import are **.aif**, **.wav**, **.mp3**, and **.ogg**

11&12. Add Text

With this option we can add a button text. We have used a large Font Size and scaled it down to have nice crisp text in your scene

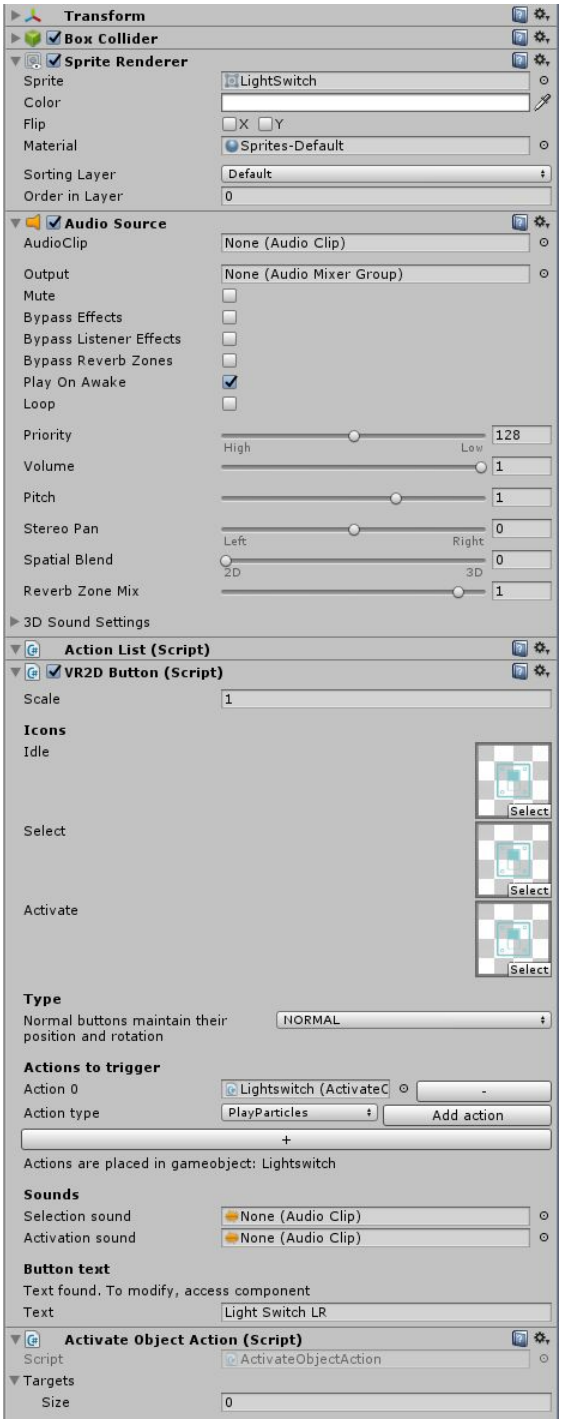


13. Create Button

This will create your 2D button with the parameters you have chosen

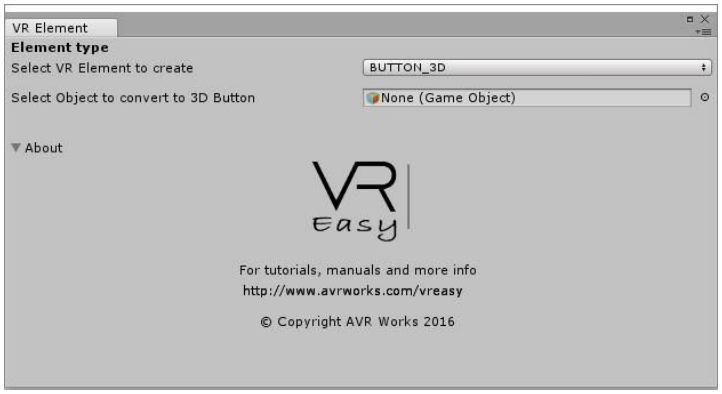
RESULT OF BUTTON_2D

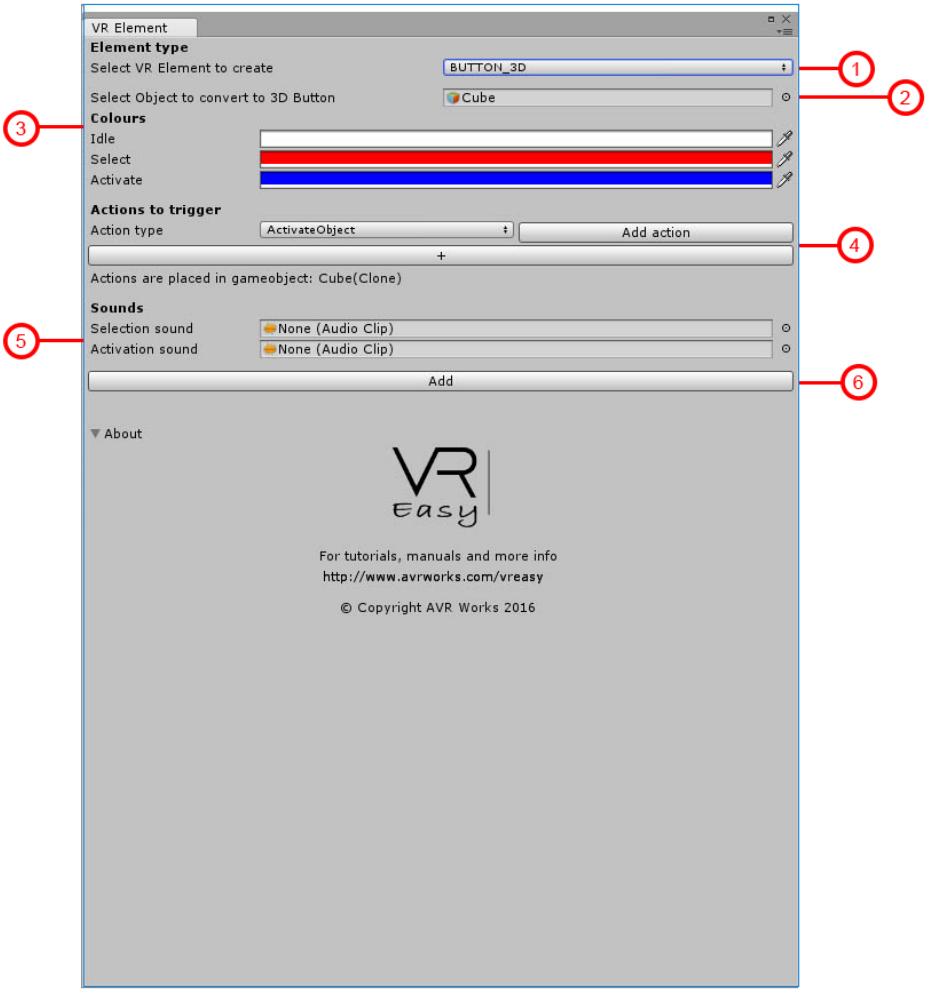
Below image shows the completed button with all the required actions and scripts you have selected after creating your button.



BUTTON_3D Creation Window

2D buttons are great when you want to create a standard VR button (sprite image and box collider). However, it is perfectly acceptable to want to have a more complex representation for a button, such as a mesh object with a complex collider. For those types of buttons, VREasy brings 3D buttons. You can create a 3D button from a prefab (in which case it will be instantiated as a new game object) or you can add 3D button functionality to an existing game object. This chapter will go through each of the options that Button_3D offers.





1. Type of VR Element

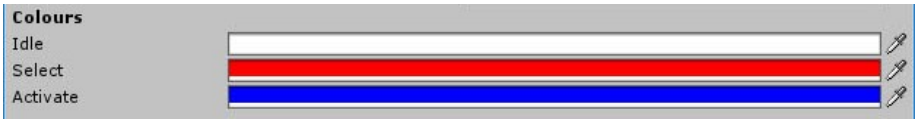
Selecting Button_3D will give you all the options you have with a Button_2D except the icons because we are using a 3D Object as our button

2. Select Object to Convert to 3D Button

Drag or Select your object you want to behave as a 3D Button -either an existing game object in your scene, or a new prefab to be instantiated.

3. Colours

With Idle, Select and Activate Colors you can set the different selection states of your button



4. Actions to Trigger

Select here the type of actions to you want to perform with your button. You can stack or remove as many as you want by clicking “Add Action” each time you add an action or click this button to remove unwanted actions

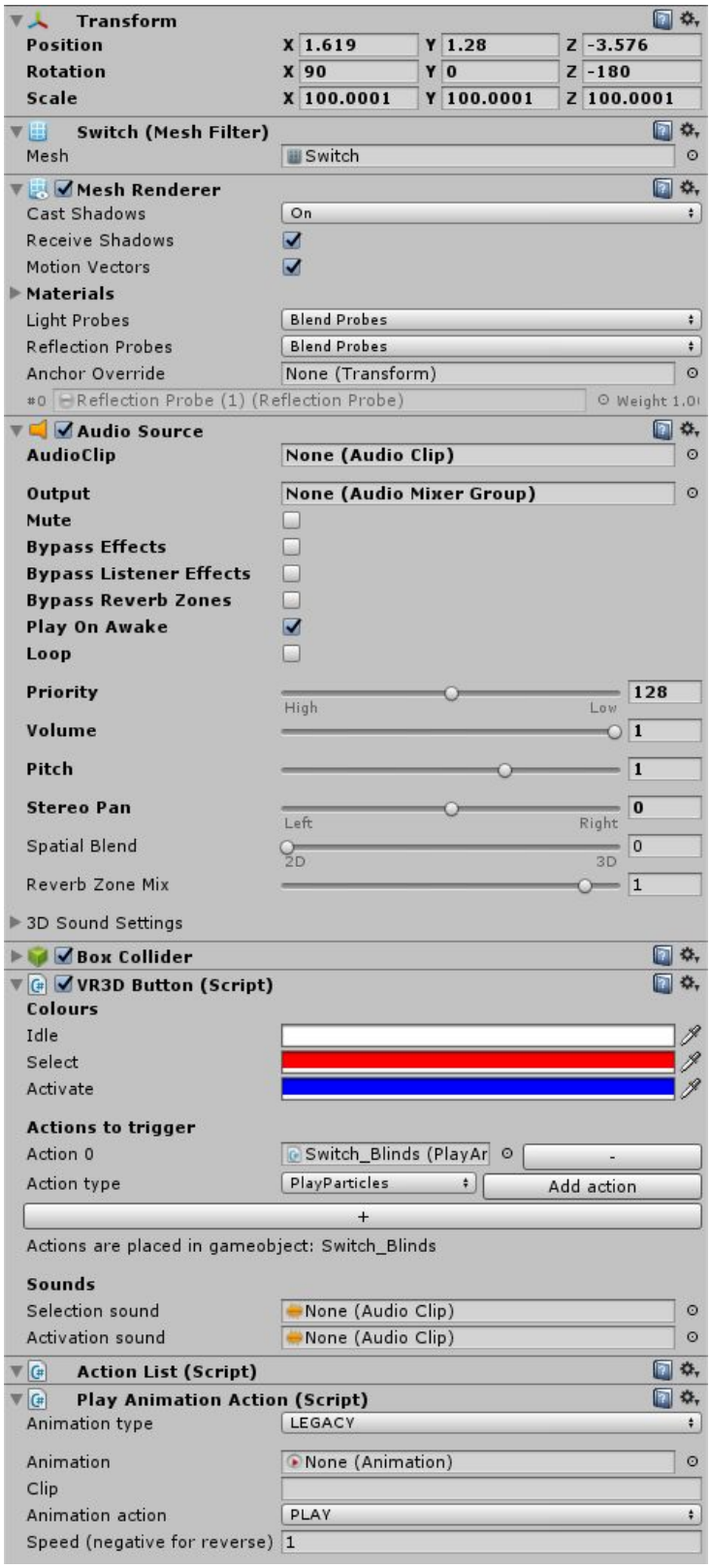
5. Sounds

Here we can add the selection and activation sounds to our VR Elements. Audio formats that Unity can import are **.aif**, **.wav**, **.mp3**, and **.ogg**

6. Add

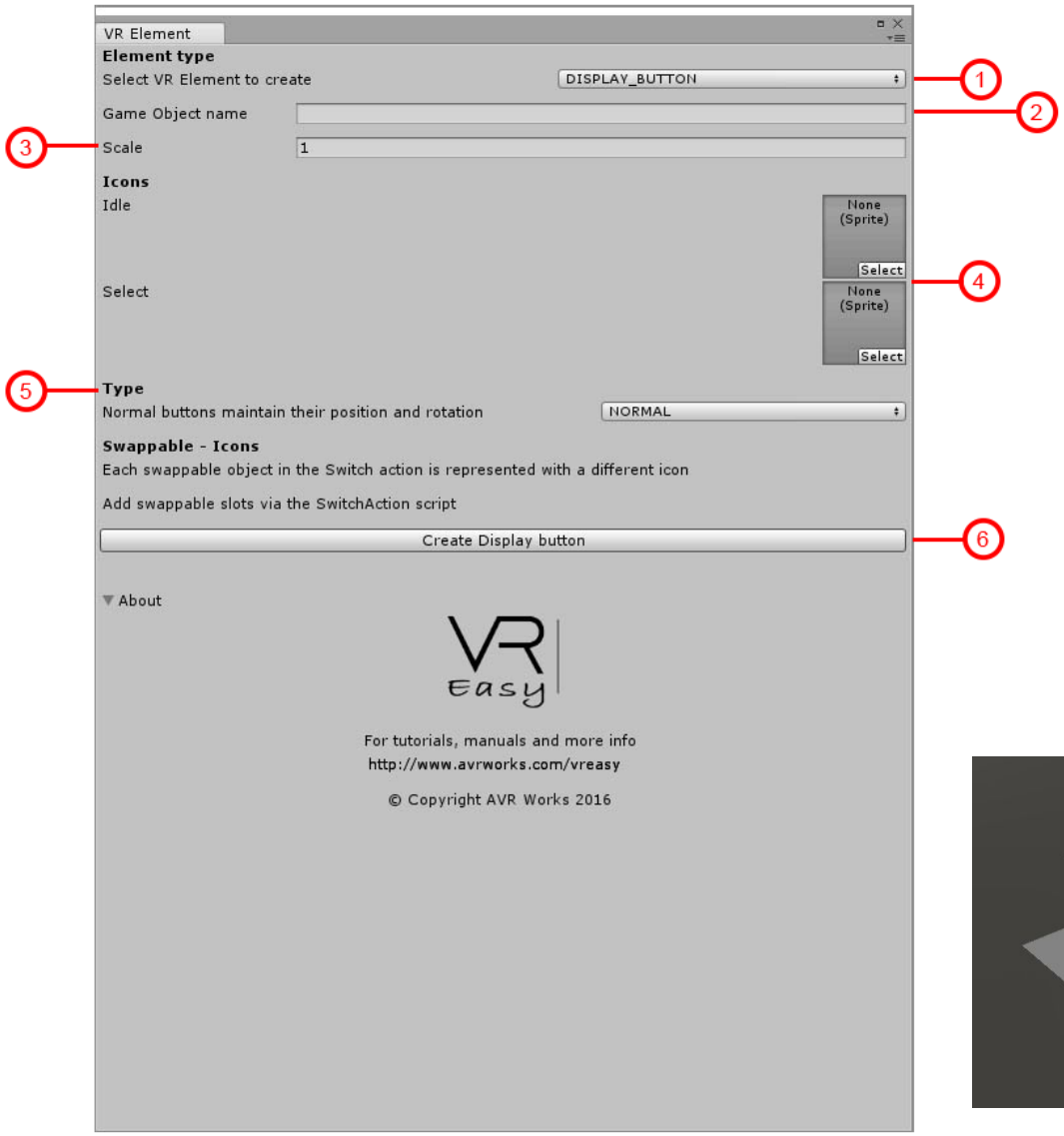
This will create your 3D button with the selected parameters you have chosen. This time a separate GameObject is not created instead all the necessary actions are added to the 3D Object you selected as your 3D Button

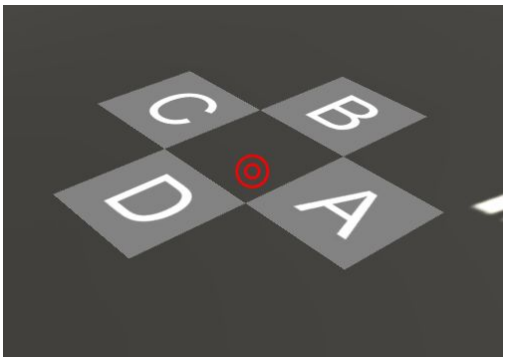
RESULT BUTTON_3D



DISPLAY_BUTTON Creation Window

This VR element is a specialised renderer object for buttons with a Switch action associated to them. Why, you may ask? It is very useful for buttons that control switchable objects (textures, meshes...) to want to display each of the switch options for the user to pick one -rather than having a button and cycling through the alternatives on every push. With Display_Button you are creating a 2D Button that dynamically displays any number of switch elements for you. When the button is activated as usual, the system generates and display individual buttons around it with each switch option.





(process of creating DISPLAY_BUTTON element is explained in a video tutorial which can be found here)

1. Type of VR Element

DISPLAY_BUTTON will give you a 2D Button with a canvas and text based on your scale, icons and name of your button

2. Game Object Name

Type here the name of the element that you want to create and will appear in the Hierarchy

3. Scale

The scale value is is a measurement of the global size of the vr element -it roughly makes 1 = something easily selectable in a VR scene with normal scale (1 unit = 1 meter)

4. Icons

You can set there the icon for each of the states in Idle and Select.

5. Type

We can set here how our button will face the camera, we have 3 options: Normal, Sticky and Billboard.

In **Normal** mode the elements will be placed in World Space mode

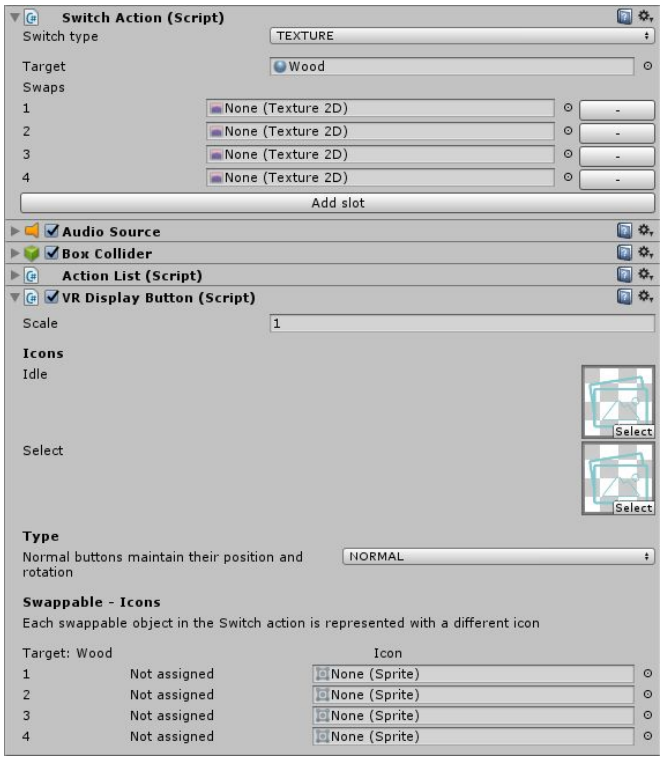
In **Sticky** mode elements follow the movements of the HMD and appear to be stuck to the camera.

In **Billboard** mode the elements will be placed in World Space mode always facing the camera

6. Create Display Button

This will create your 2D button with the parameters you have chosen.

Modifying the Display Button



1. Switch Type

A Switch Type (i.e. a container) is the component or object that holds the property you wish to switch; here is a list of the containers for each of the switch types:

- Mesh → Mesh Filter component
- Material → Renderer component
- Texture → Material object
- Sprite → SpriteRenderer component

The *Custom* switch type has been added for advanced purposes. It allows you to specify any container (as long as it is a Unity Object) and switch any of its public properties. Please note that this support is still experimental.

2. **Target**

The component or object that holds the property you wish to switch

3. **Swaps**

Swaps are the properties you want to change in between them

4. **Add Slots**

Add Slot will add a new swap slot each time you click this button

5. **Scale**

The scale value is is a measurement of the global size of the vr element -it roughly makes 1 = something easily selectable in a VR scene with normal scale (1 unit = 1 meter)

6. **Type**

We can set here how our button will face the camera, we have 3 options: Normal, Sticky and Billboard.

In **NNormal** mode the elements will be placed in World Space mode

In **Sticky** mode elements follow the movements of the HMD and appear to be stuck to the camera.

In **BBillboard** mode the elements will be placed in World Space mode always facing the camera

7. **Swappable Icons**

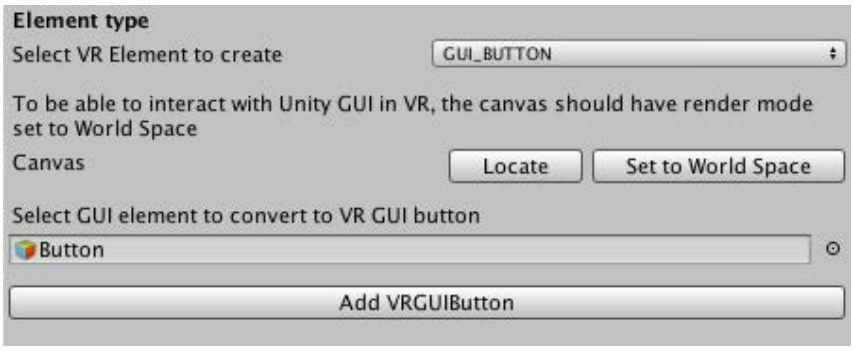
Swappable icons are sprite images that represents the swaps you have selected as Switch options

8. **Target**

Here we have to set the thumbnail that represents the texture, mesh or material that will be displayed as your button icon

GUI Button

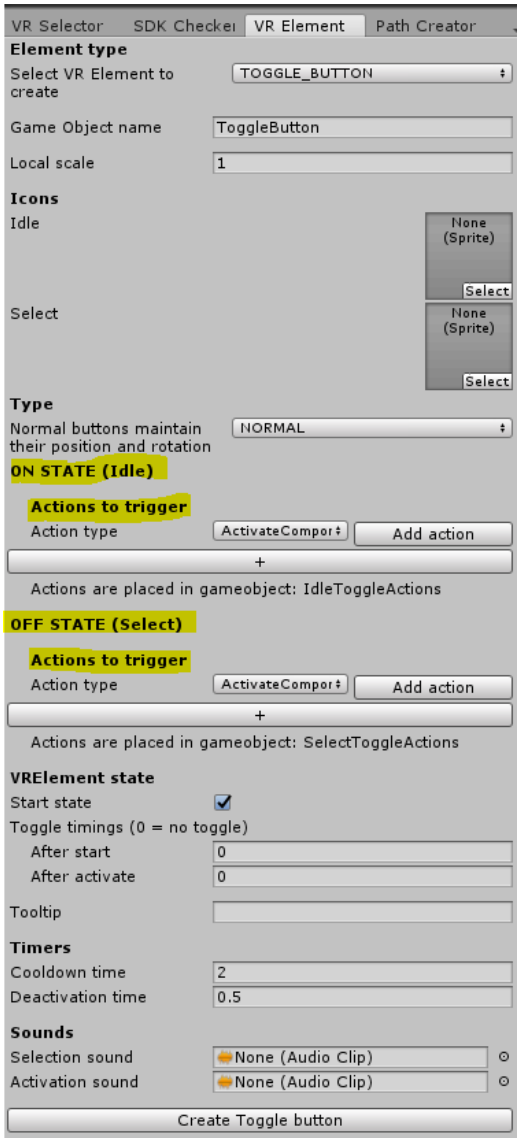
VREasy now supports interaction of all our Selectors (pointer, sight and touch) with Unity's Default GUI buttons. To make a GUI button interact with a selector, you must upgrade it via the VR Element panel.



Use your GUI button as the GUI element to convert in the VR Element panel and click on Add VRGUIButton. Since selectors operate in the world space, your Unity GUI (the canvas) must be set to World Space too. You can do this manually or via the button offered within the VR Element panel.

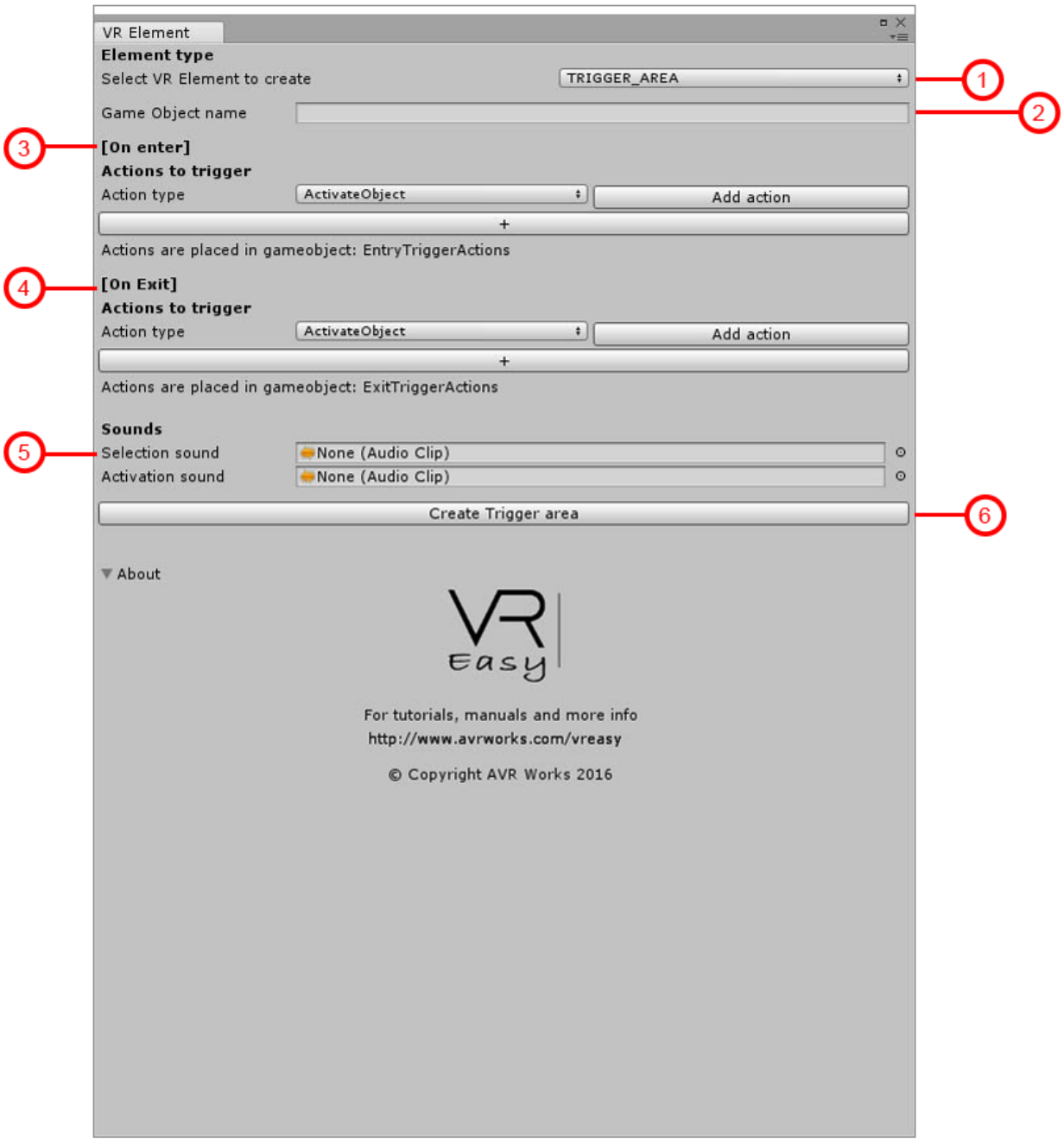
Toggle Button

A VR Toggle is an element similar to a 2D and 3D buttons, but instead of having a single set of actions associated with it that are triggered every time the button is pressed, it has two sets of actions. VR Toggles have an internal state, ON and OFF, and when they are activated they trigger the set of actions linked to the current state. They are represented by two icons, one for each state



TRIGGER_AREA Creation Window

Trigger areas are Box colliders with trigger properties to trigger actions/events that you want to occur when entering or exiting these trigger areas



(process of creating a TRIGGER_AREA element is explained in a video tutorial which can be found here)

1. Element Type

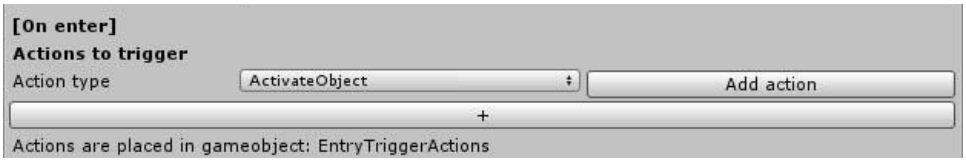
TRIGGER_AREA will give you a Collision Box with 2 GameObjects attached to control the On Enter and On Exit actions

2. Game Object name

Type here the name of the element that you want to create and will appear in the Hierarchy

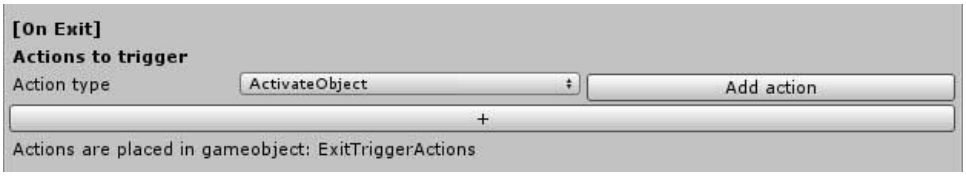
3. On Enter Actions to trigger

Here you can add the actions that you want to happen when entering the trigger area



4. On Exit Actions to trigger

Here you can add the actions that you want to happen when entering the trigger area



5. Sounds

This will give you the option to select a selection and activation sound for your trigger area

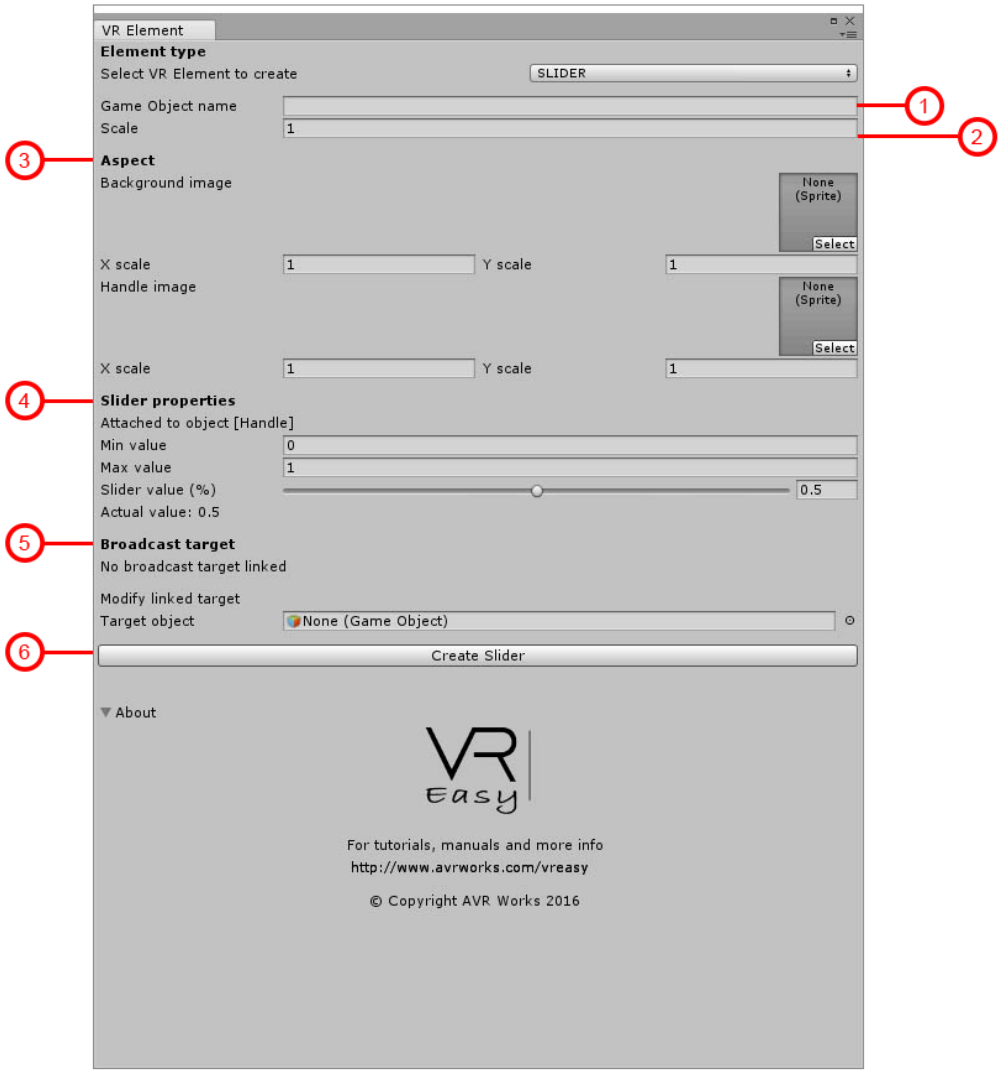
6. Create Trigger Area

With this button you will create your TRIGGER_AREA element and add it to the scene

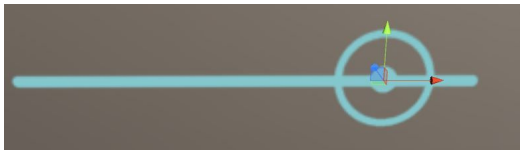
Result of the TRIGGER_AREA element in the scene Hierarchy

SLIDER Creation Window

Sliders can be used for various purposes in your scene from controlling light properties to rotation of objects.



(process of creating SLIDER element is explained in a video tutorial which can be found here)



Example of a SLIDER

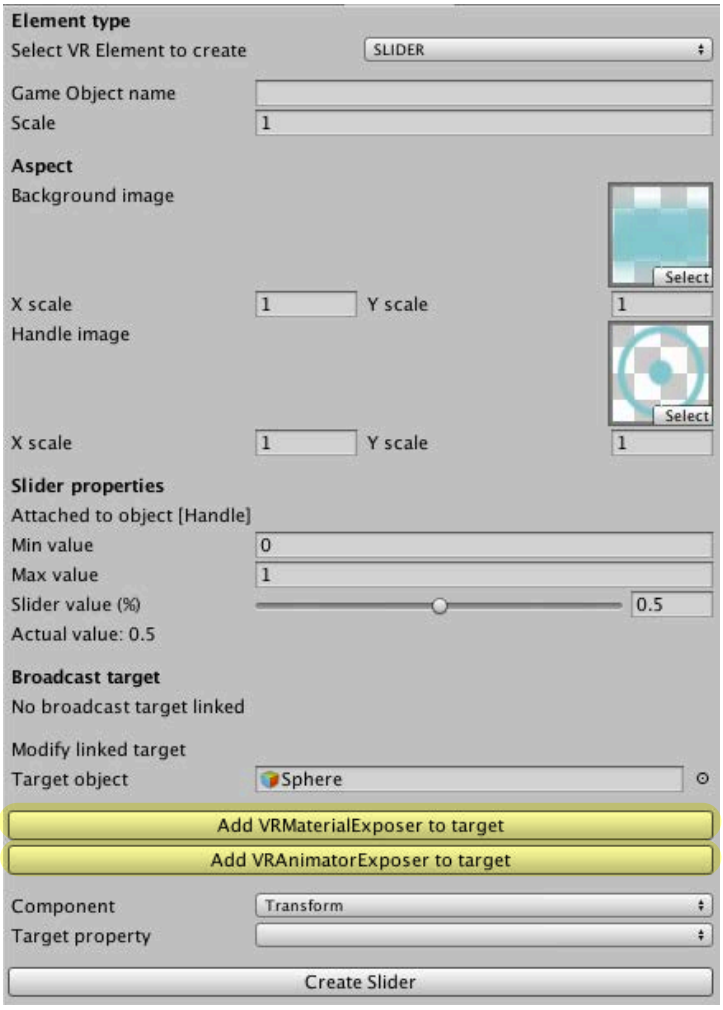
Transform controller

As an example of how VR sliders can interact and control the world around them, we have included TransformController (located at VREasy/Scripts/Movement). It is a very simple wrapper script that exposes the position and rotation of a game object as public properties that can be linked with a VR Slider -see Slider manual section for more information.
Since each Slider can be linked to control and broadcast a value to any public property on any game object, by attaching the Transform Controller script to any game object your sliders will be able to alter the X, Y and Z position and rotation of that object.

Video tutorial on how to setup the slider to control transform can be found **here**

VRSlider controls numbers, vectors and colours

In addition to the original support to control numeric valued properties, VRSliders can now control directly Vector, Colour and Animator properties. To do so, within the VRSlider component panel, select as your *Target object* the object with the property you want to control. Then click on *Add VRMaterialExposer* if you want to control a property in the material of that object (Vector or Colour), or click on *Add VRAnimatorExposer* to control properties of the animator component on that object.
After adding the appropriate exposer component, you can select the property by selecting the *Component* and *Target property* from the drop down menus and finally clicking on *link*.



Remember! VRSlider will always control a single valued property, but with exposers now you can control a value within a Vector, or Colour, or Animator. Of course you can have more than one VRSlider controlling the values of a single Vector!

1. Game Object Name

Type here the name of the element that you want to create and will appear in the Hierarchy

2. Scale

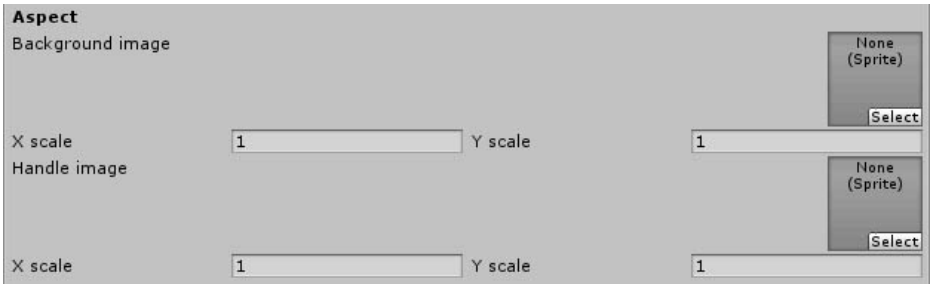
The scale value is is a measurement of the global size of the vr element -it roughly makes 1 = something easily selectable in a VR scene with normal scale (1 unit = 1 meter)

3. Aspect

Sliders are formed using two sprites: the background (the static image) and the handle (the image that is used to control the slider and reflects its current value). This section allows you to specify the looks of your slider.

Normally, if you have designed your background and handle sprites to scale (with the adequate proportion) then you are not likely to need to tweak the scale of each component separately. But if you wish to stretch either the background or the handle images, the *aspect* allows you to do so. You can scale both X and Y axis independently.

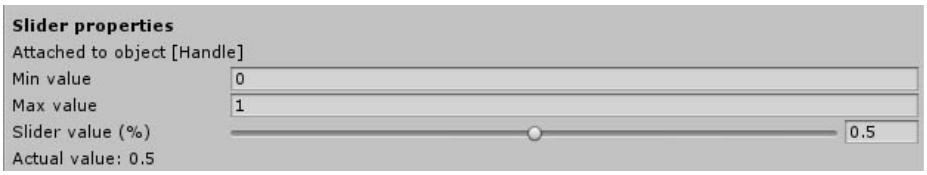
NOTE: Please do not use the background or the handle transforms directly to scale those game objects as the slider internal behaviour sets them for you via the *aspect* section.



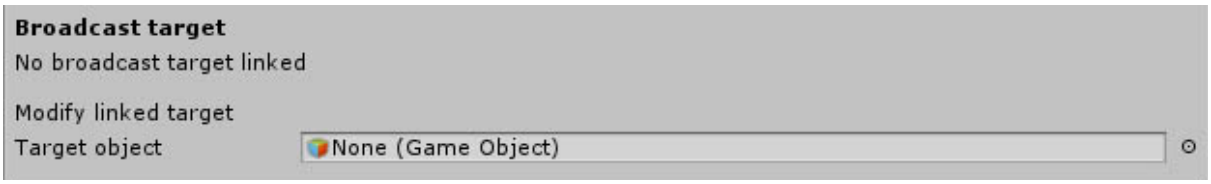
4. Slider Properties

By default sliders have a range of possible values between 0 and 1. If this does not suit your needs, you can change the minimum and the maximum values by setting the *min value* and *max value* fields.

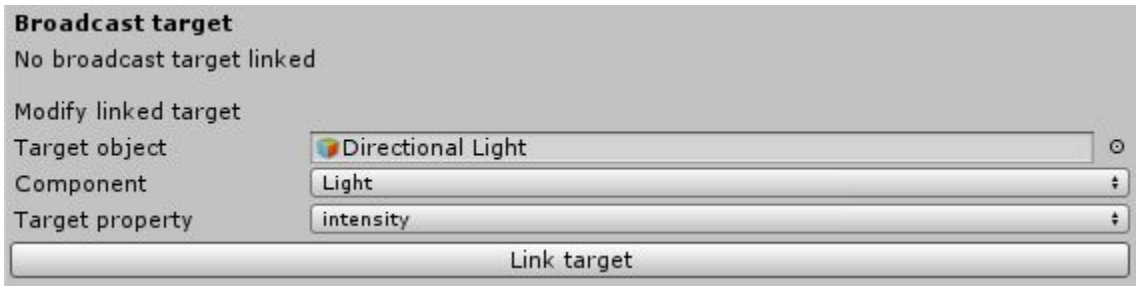
The *slider value* shows you the current raw value the slider holds (always between 0 and 1). If your slider has a different range, the *Actual value* shows the final selected value.

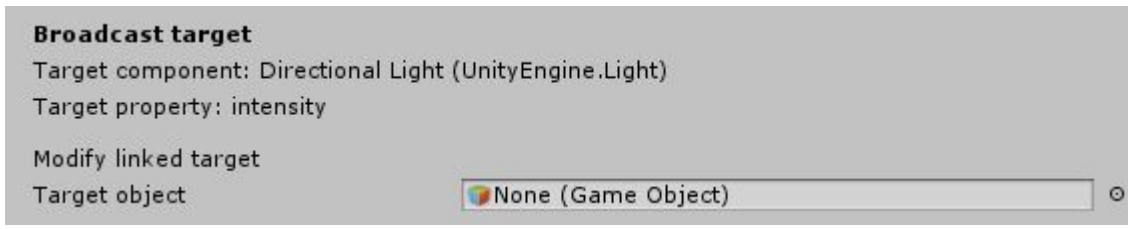


5. Broadcast Target



Sliders are fun and beautiful, but they are not much use if you cannot link its value to an object! The *broadcast target* section allows you to make other objects aware that the slider has changed its value - hence the name broadcast. This can be used to make the slider control a particular property on any component of a game object. To do so, drag the object you want to control to the *target object* property. VREasy automatically displays any components attached to the game object; if the component has any property that can be controlled with a slider (i.e. any property that accepts a numerical value), VREasy will show a drop down menu *target property*. Choose the property you wish to link to the slider and click on Link target. If all goes correctly, you should see the property selected under the Broadcast target



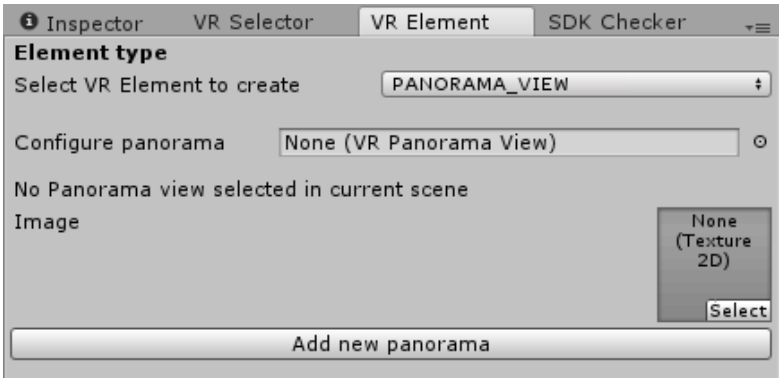


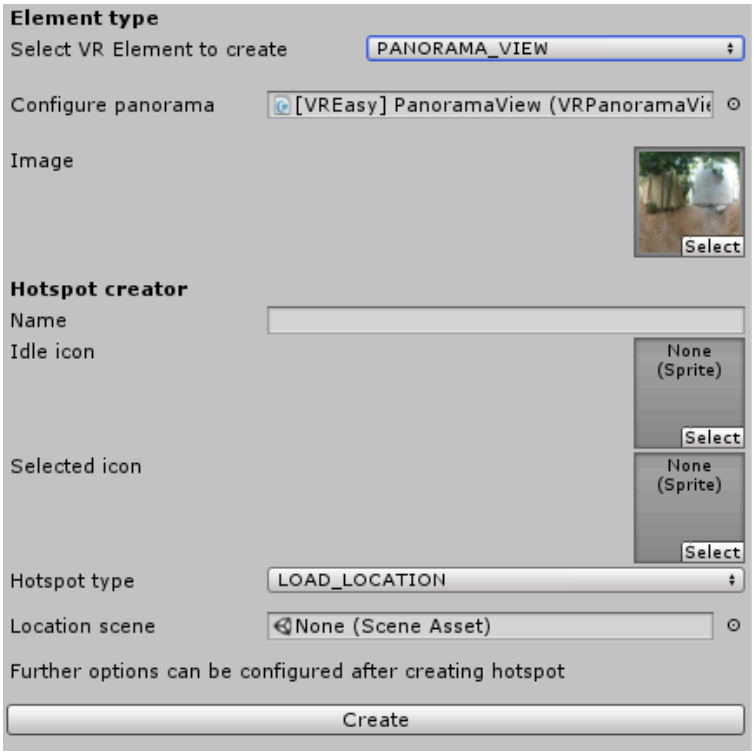
6. Create Slider

This will create your Slider button with the parameters you have chosen.

VR Panorama View

This is a brand-new VR element that allows users to create Panorama 360 views with ease both images and videos are supported. To add one to the current scene, use the VR Element Panel and select Panorama_View as VR element.



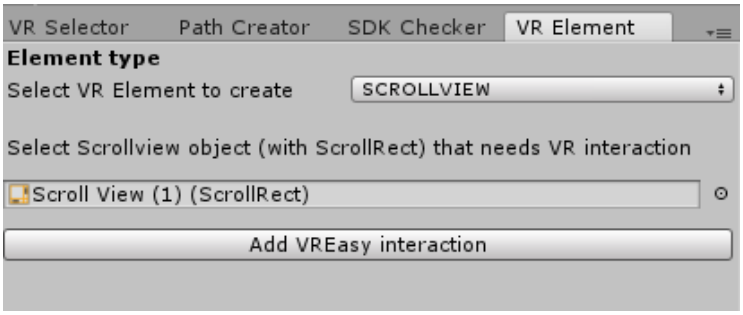


The top part of the panel will show the object created and the image selected (which can be modified at any point). The bottom part is a *otspot Creator* GUI that helps you add hotspots to your panorama view. A hotspot is an interactable element that can show information (like a PopUpAction) or load a new location (like a LoadSceneAction). Once you have configured your Hotspot, click Create to spawn the hotspot in the scene. By default, it is created in front of the HMD camera in your scene, but you can move it anywhere you like.

If at a later stage you wish to either create more hotspots or to change the image of the Panorama View object, select the Panorama object in your scene and you can alter any properties from the Inspector.

Scroll View

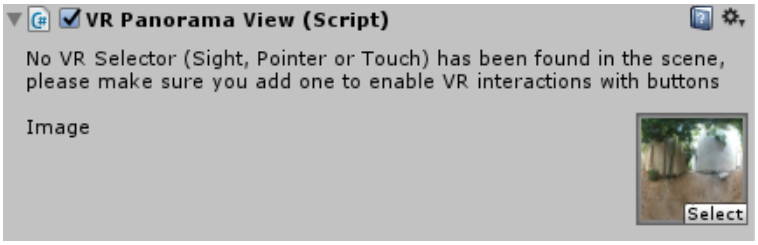
A new VR Element has been added to version 1.5, which allows the display of GUI elements in an scrollable panel. VREasy adds VR interaction to ScrollView objects, making the scrollbars interactable with compatible selectors (pointer and touch). Any already created ScrollView GUI object can be transformed into VR compatible by using VR Element panel.



Note that because the GUI must be in the 3D world to interact with selectors, the Canvas containing the ScrollView should be in World Space mode -the VR Element panel can assist on how to do this automatically.

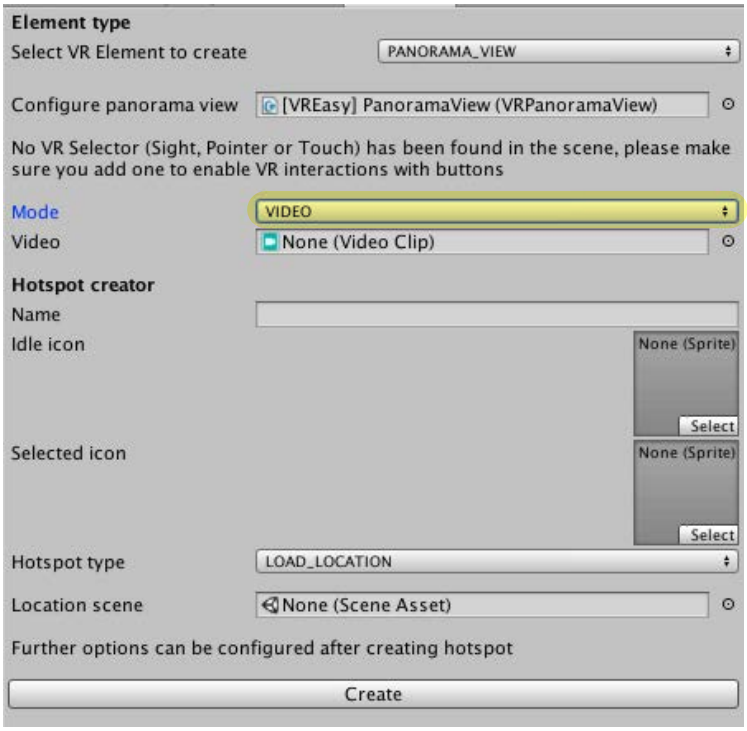
Once the ScrollView is VR interactable, any type of GUI on the Viewport child game object will be scrollable. This can of course be VRGUIButtons (Unity 2D buttons with VREasy interaction activated), which can trigger actions within VREasy.

Note that to interact with hotspots, there must be a Selector active in the scene (touch, sigh or pointer). If you do not have one, Panorama View objects will show a warning in the inspector:



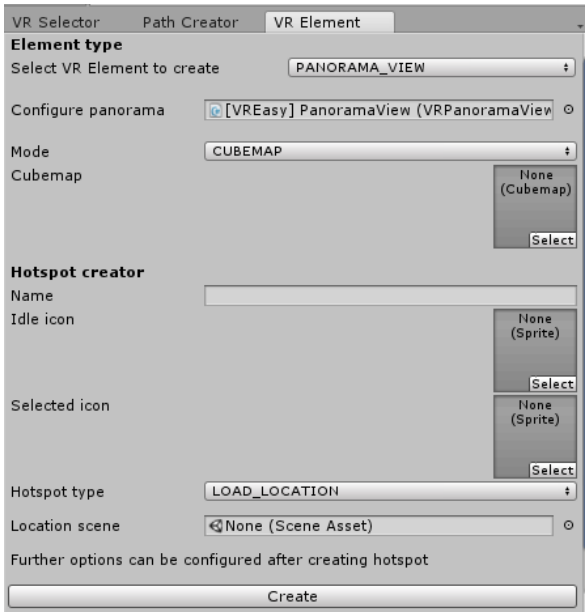
Panorama View video support

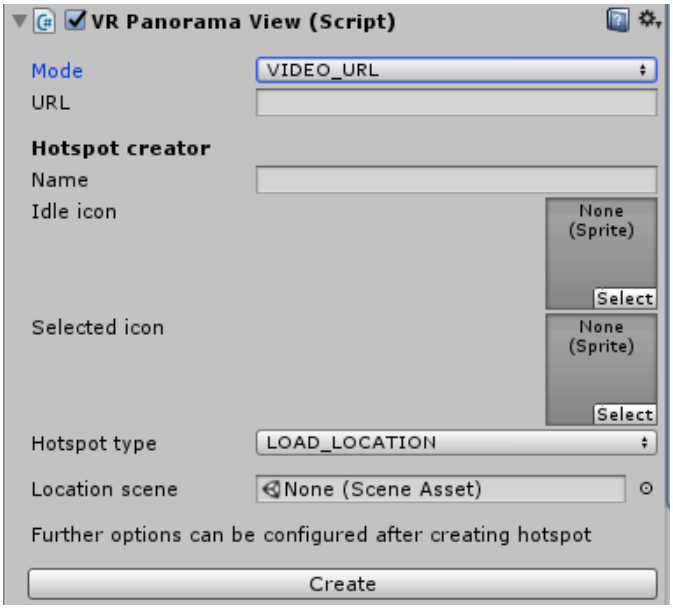
VREasy already supported Panorama View, which allowed users to be surrounded by a 360 still image in which you could place interactive hotspots. Now, since version 1.4, we also support videos, so your Panorama View can display 360 videos to immerse your users. To do so, change the Mode on the Panorama View component to video (or when you create it through the VR Element panel) and link your video to the Video property.



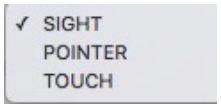
Panorama view support for Cubemaps

Panorama view objects now have three modes: Video (from file), Video (from URL) and Cubemap (where the image displayed in the panorama is loaded from a cubemap file).





VR Selector Creation Window



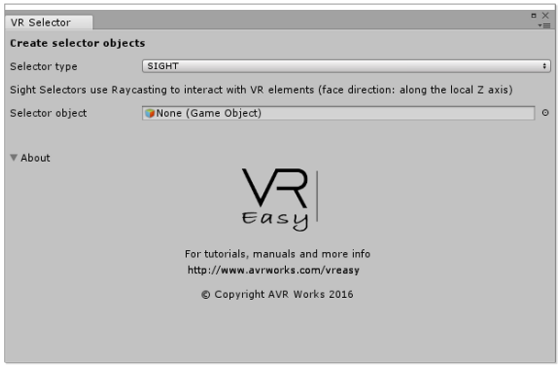
This menu aids in the creation of Selector objects for your VR scenes. A Selector is an object that can interact with VR elements. The current version of VREasy (v1.0) supports two types of Selectors:

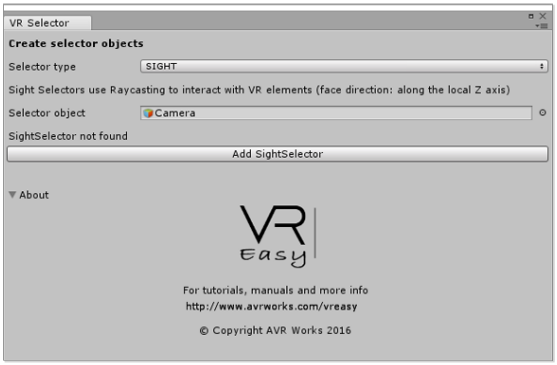
- Sight Selector: an object that interacts with the world via a ray casted along its Z axis (it can select elements from a distance by *looking* at them).
- Touch Selector: an object that interacts with the world via physical contact -colliders.

SIGHT

When you want to add Sight selection capabilities to your current camera, drag the camera eye to the *Selector object* field and click Add Sight Selector.

Sight selection is not restricted to cameras and you can very easily create a pointer-like selector by adding a sight selector to a normal object. From that moment on, the object will be able to interact with VR elements by pointing at them along the Z axis (future versions of VREasy will include a more separate advanced Pointer selector).

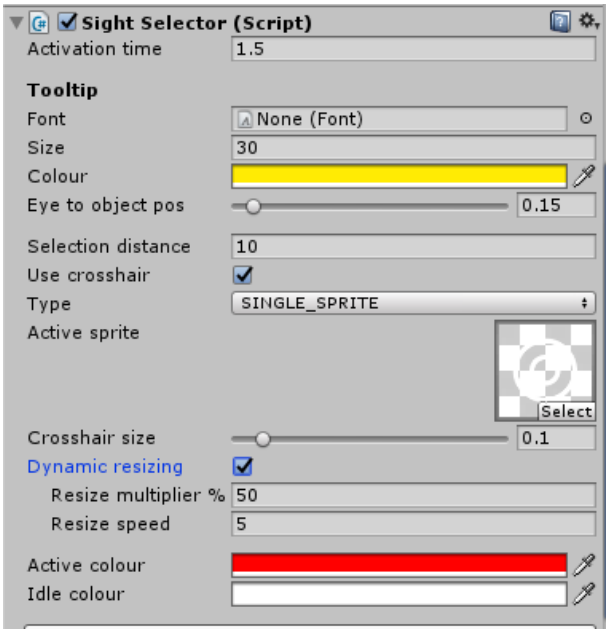




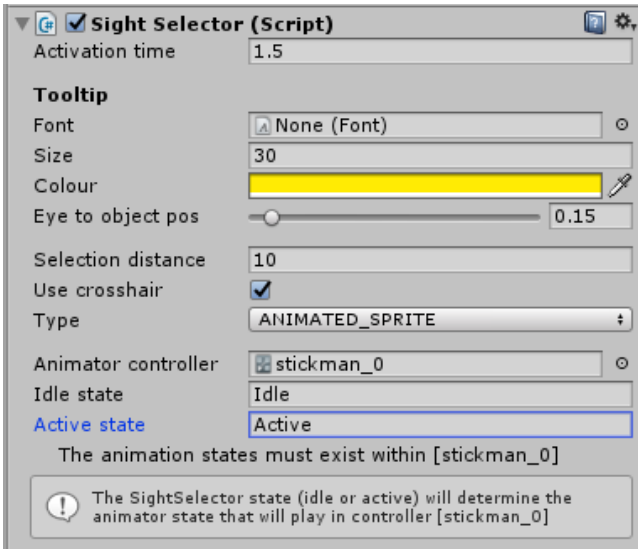
(process of setting up the SIGHT selector is explained in a video tutorial which can be found **here**)

Dynamic scaling of crosshair for SightSelector

When using either Single or Double sprite *type* on SightSelector, users have the option to apply dynamic resizing of the crosshair, which will increase the size of the crosshair when it hovers over an interactable VR object. Both the amount increased (*resize multiplier*) and the speed of transition (*resize speed*) can be modified via the inspector.



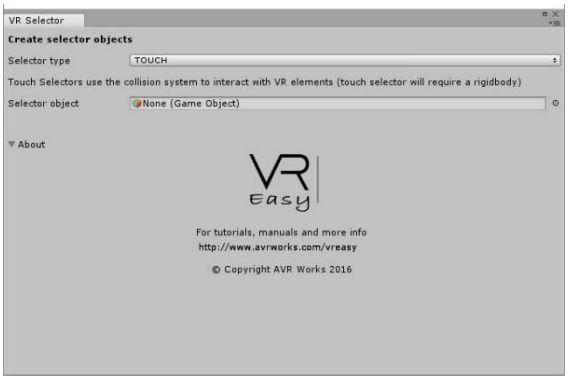
In addition, a new crosshair type has been added: *Animated* sprite. This allows the crosshair to be controlled by an Animator (via an Animator Controller), permitting state-based animated sprites. Users have to link an *AnimatorController* which should have at least two states, one for each level of interaction of the SightSelector (idle and active). *Idle state* and *Active state* should match the name of the correspondent state within the Animator Controller. See Unity Animator documentation for more information on how to setup and configure Animators.

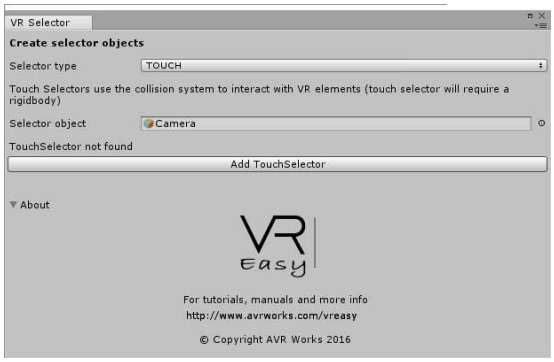


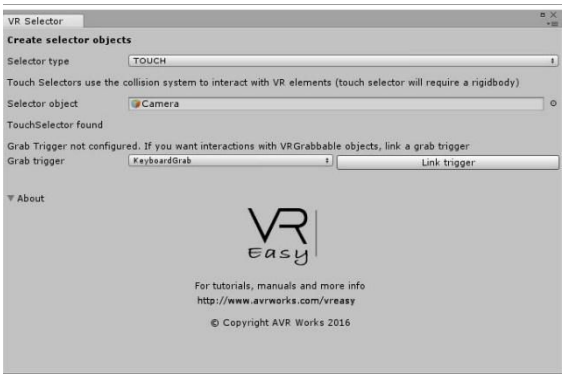
TOUCH

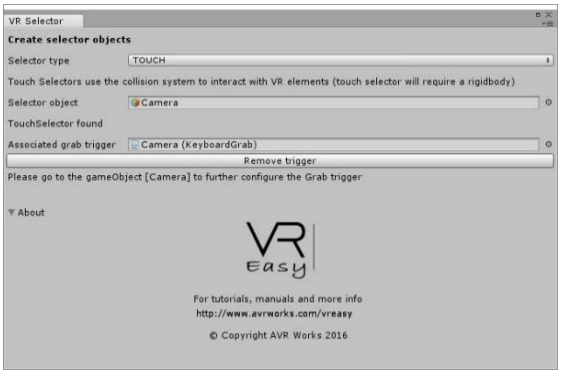
To add touch selection capabilities to an existing object in your scene (a steam VR controller, a leap motion hand, a custom object controlled with a gamepad), drag the object to the Selector object field and click on Add Touch Selector.

VREasy uses the Unity physics engine to detect collisions. Hence, the touch selector must have a valid collider and a rigidbody attached to work. When adding a touch selector using the GUI these elements get added and configured automatically. If your object already has any of those, it will reuse them.



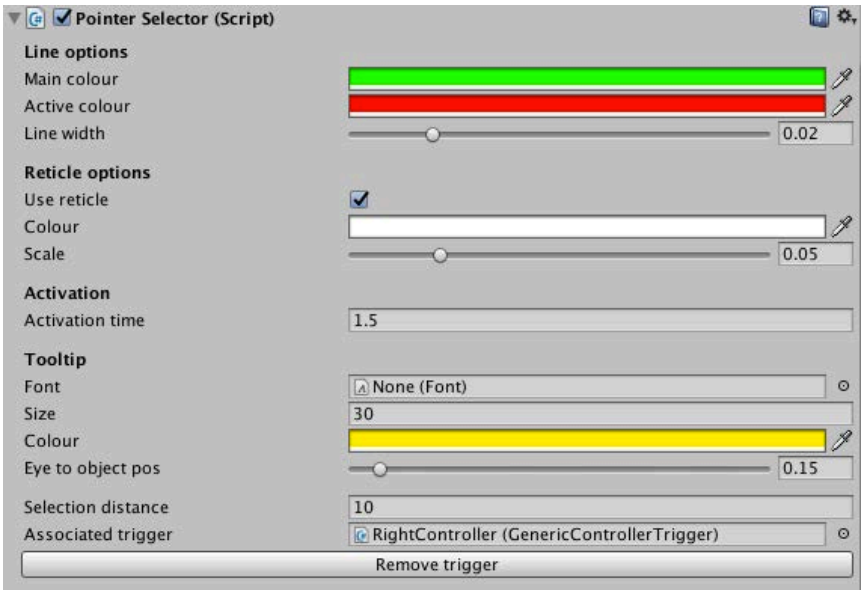




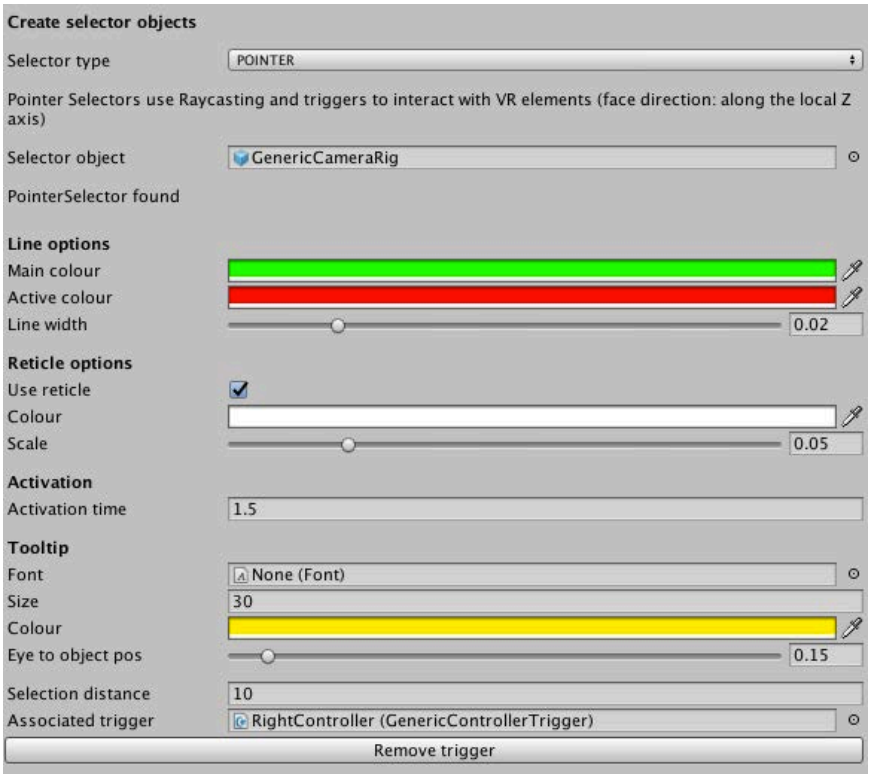


POINTER

A Pointer Selector adds to the pool of available selectors offered -Sight Selector and Touch Selector. It behaves like the Sight Selector, but instead of being attached to the main camera, it is normally used to give pointer-like features to any game object. When attached to an object, like the SteamVR controller, it allows users to shoot a laser pointer which interacts with VR elements (buttons, grabbable objects, etc.). The laser is always shot along the Z+ axis of the object, and it is active when triggered -as with VRSimpleFPSLocomotion and Grab features, VRTriggers can be linked to control this activation.



Users can add pointer-like features to any game object via the VR Selector panel, choosing a Pointer type selector object and dragging their object to the field.



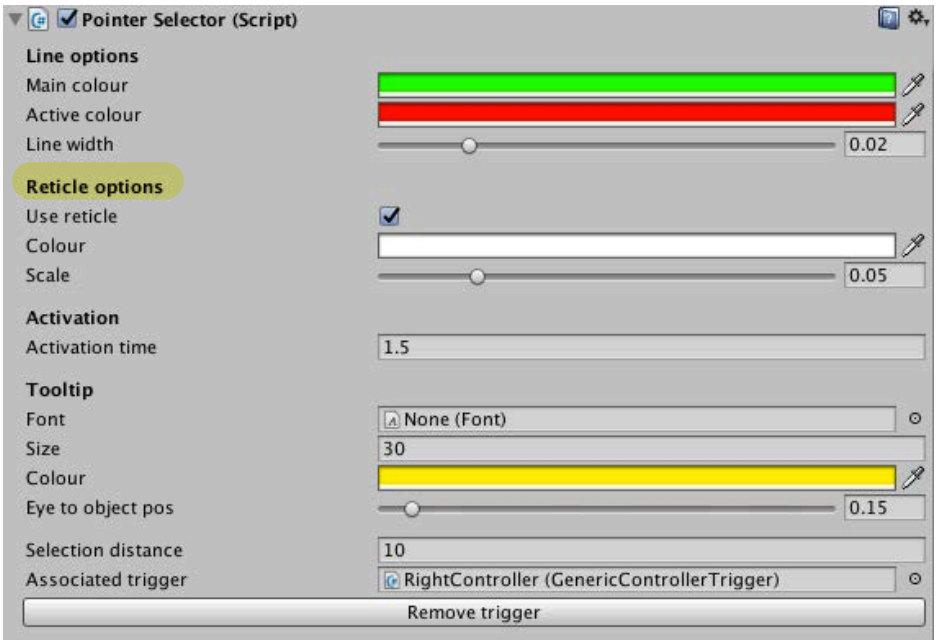
Normally Pointer Selectors are ideally paired with SteamVR Controller objects, as one can then use their hands to point and select / grab objects in the scene. Just drag the SteamVR Controller object to the Selector type field in the VR Selector panel and add a Pointer Selector.

Reticle

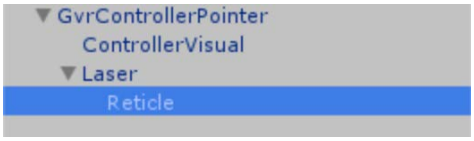
To facilitate the interaction with your PointerSelectors we have included two improvements: added a reticle and a change of colour for the selection laser.

The reticle is a virtual sphere that is rendered in the position that represents the end of your laser pointer, i.e. the end of the selection radius. If your object is interacting with a VR element, the reticle will appear as if hovering that object to give users a visual clue that they are interacting with an object.

The colour of the laser will now change depending on the state of the selection; if it is not interacting with a VR element, it will display the Main colour (green by default), if it is interacting then it will display the Active colour (red by default).



Note that other solutions do use their own reticle. For instance if you are using the GvrControllerPointer prefab from Google VR SDK, this comes with a similar reticle. If you are using that object as your basis for a Pointer Selector, we recommend disabling either our reticle (via the Pointer Selector component, Use reticle unchecked) or their reticle (by disabling the Reticle child game object as below).



Normally Pointer Selectors are ideally paired with SteamVR Controller objects, as one can then use their hands to point and select / grab objects in the scene. Just drag the SteamVR Controller object to the Selector type field in the VR Selector panel and add a Pointer Selector.

If you want to use the Oculus prefab (that is included in the Oculus Utilities for Unity Core package)

1) drag the OVRPlayerController (or the OVRCameraRig) to the scene,

2) navigate through the children objects until you find the CenterEyeAnchor object.

3) Create a SightSelector with the VREasy VR Selector panel using the CenterEyeAnchor.

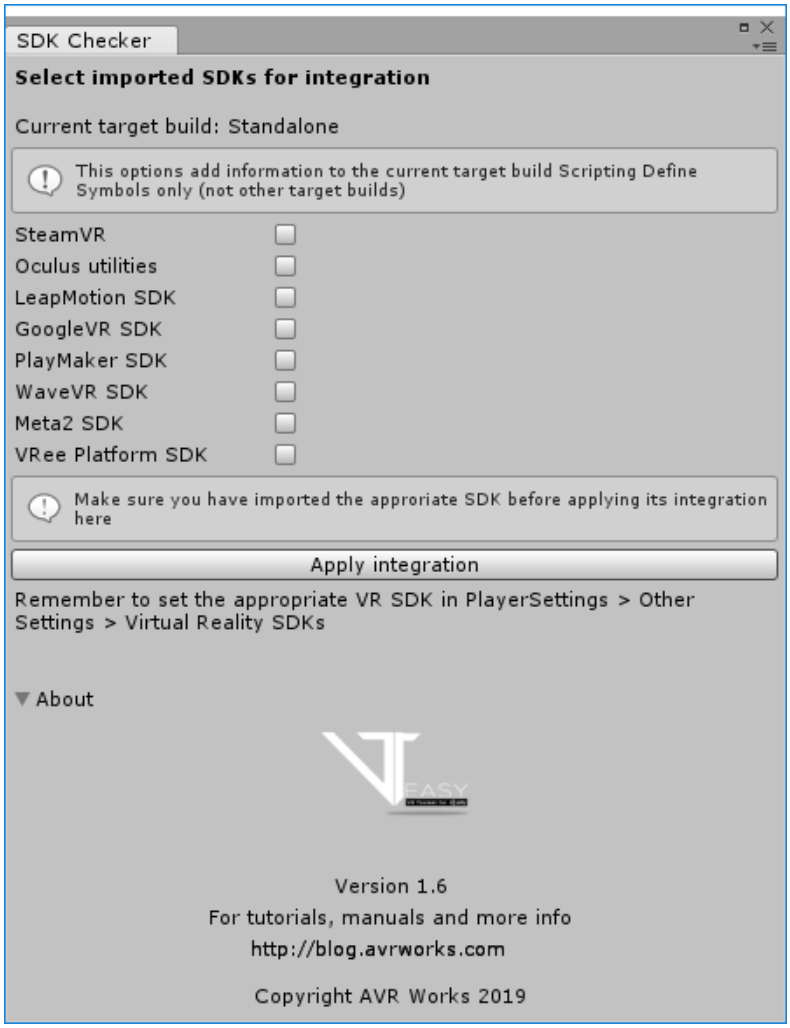
4) Make sure you configure it to have a crosshair via the inspector

5) Set the crosshair sprite if not already loaded

SDK Checker Window

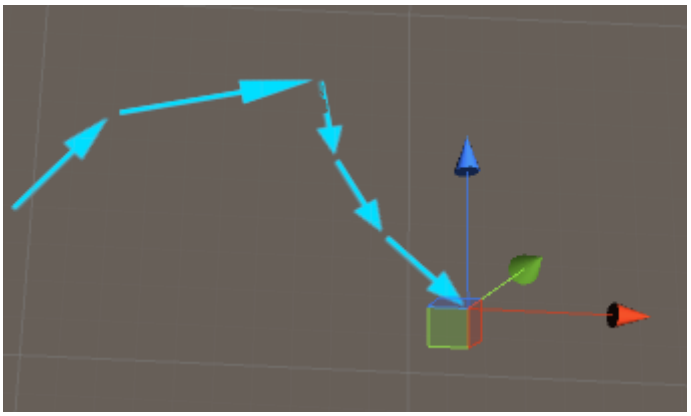
This window is only required if you wish to activate VR Easy integration with external plugins and assets. At the moment VR Easy has a specific integration with the Steam, Oculus and Google VR Controllers (for basic locomotion and grab). If you plan to use it, import the appropriate SDK first and then check the imported SDKs.

NOTE: You may see some compiler errors if you check SteamVR integration but have not imported the SteamVR plugin to Unity.

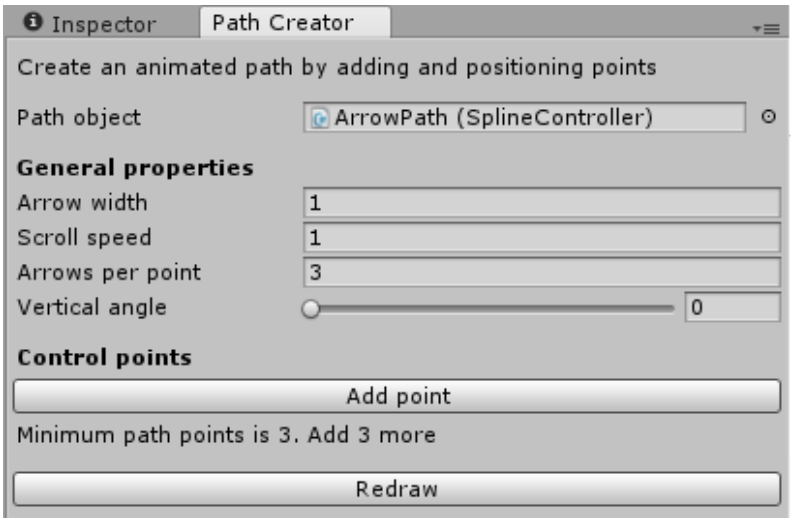


Spline path creator

To make walkthroughs easy to design, VR Easy 1.2 introduces a Spline Path creator. With it, you can easily generate dynamic paths that will be rendered with a scrolling arrow to help navigating your scenes.



To help with the creation of paths, we have set up a GUI panel called *Path Creator* that can be accessed in your Unity menu via VR Easy > Path Creator.



The GUI allows you to: 1) create a path by adding control points to it; 2) configure your path properties. The properties directly control the look and dynamics of the path, from the width of each arrow to how quickly they scroll.

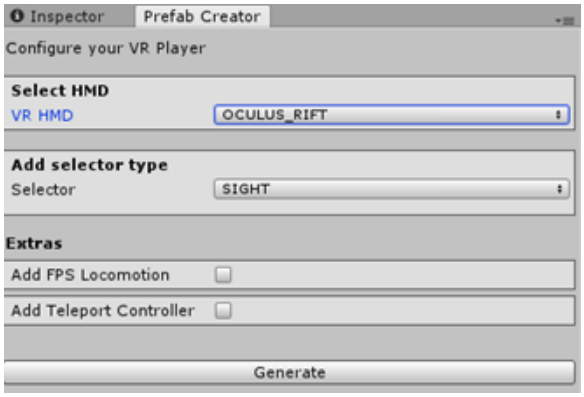
Creating a path can be done by adding a point at a time, using the button provided *Add point*. Once the point is created, the scene view will select it automatically so you can reposition it. Once you are happy with it, you can click *Add point* again to generate another point. Note that the position adopted by a new point is that of the last point, but you can move it using the Unity gizmos.

In order to generate a path, VREasy uses the points you generate as control points for a **Be ier curve**. The mathematical details are not important, but you should be aware that the path you will see is the smooth curve generated from those control points, not a rigid path from control point to control point. For technical reasons the path must contain a certain number of points (minimum of 3 and then 2 more at a time). That is why VREasy will display a message regarding how many points you should add to complete the path. The path will still be drawn if you do not comply, but some of the final points may be ignored if you do not match the number of points.

VR Prefab Creator

With the Prefab Creator you can add a VR enabled character to your scene and pre-configure it to be able to interact with VR elements (such as having selectors, or being able to teleport).

To access the Prefab Creator, open the top menu VR Easy and select Prefab Creator. A new panel will appear:

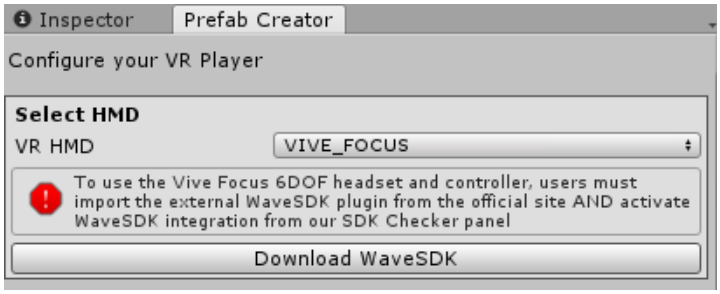


The panel is divided in three sections: Select HMD, Add selector type, and Extras.

Select HMD allows you to configure your VR player based on the headset. Currently supported headsets are:

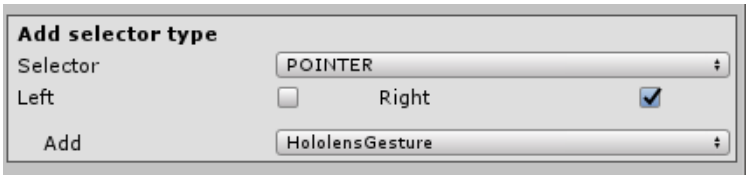
OCULUS_RIFT,
OCULUS_GO,
OCULUS_QUEST,
HTC_VIVE,
VIVE_FOCUS,
GOOGLE_DAYDREAM,
GOOGLE_CARDBOARD,
WINDOWS_MIXED_REALITY

Some headsets are only functional with the use of proprietary external plugins (such as the Vive Focus). Tn those cases, the Prefab Creator will prompt you to download and import the necessary packages.

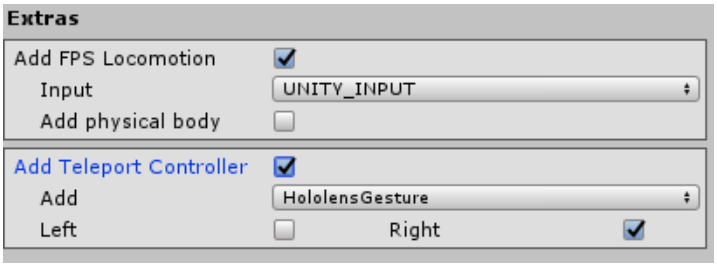


Add selector type facilitates the process of setting up selectors, which are ways to interact with VR Elements. You can easily select the selector needed via the dropdown menu. If required, the Prefab

Creator will allow you to preconfigure a trigger to activate the selector. If two-handed controllers are supported, the panel will allow you to select which hand you need the selector in.

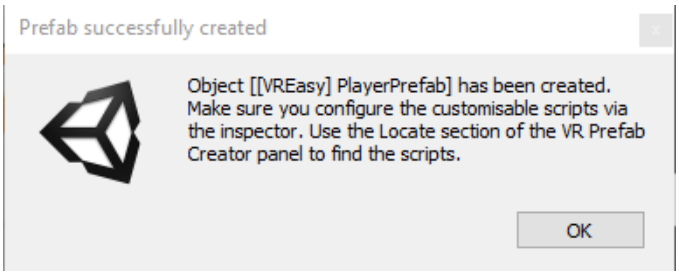


Extras: you can add FPS locomotion and Teleport functionality to your VR players by ticking the appropriate checkbox in the Extras section.

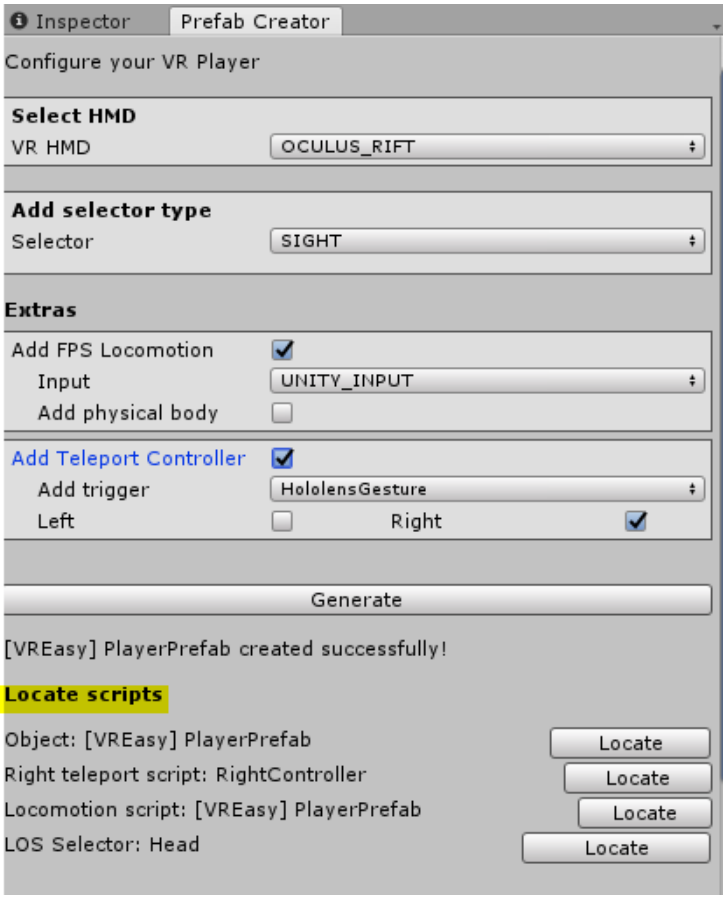


As with selectors, the Prefab Creator will guide you through adding necessary triggers, or any other preconfiguration for locomotion and teleportation.

Once the options are finalised, you can click on Generate Prefab. VR Easy will instantiate a player in your scene named *[VR Easy] PlayerPrefab.*



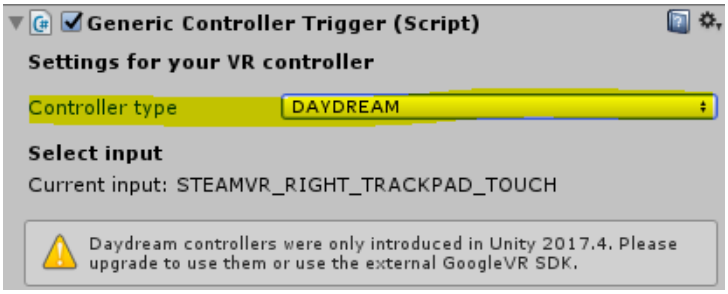
The selectors, triggers, locomotion and teleport systems ***may need further fine-configuration*** which can be done via the scripts inspectors. To make it easy to locate them, the Prefab Creator lists them and allows you to locate them by clicking on the Locate button.



More input support

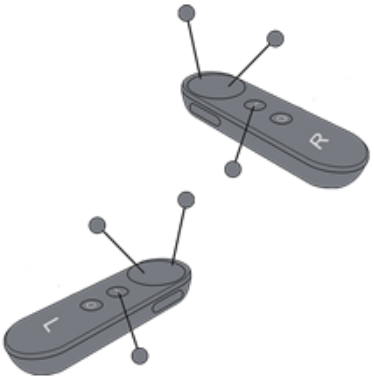
Generic Daydream controller and Left and right daydream controllers

We now support the Daydream controller input detection for triggers without requiring an external SDK via the GenericControllerTrigger.



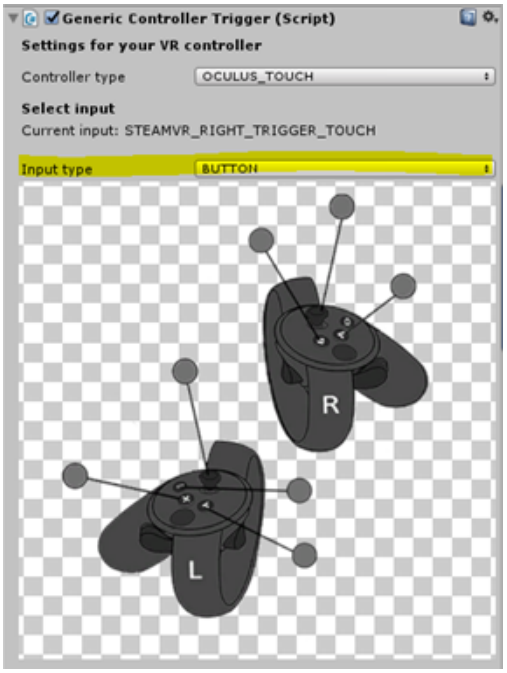
Please note that you must have Unity 2017.4 and above to be able to use the Daydream native integration.

In addition, both our generic support and the external daydream support offered through GoogleVRControllerTrigger now include both left and right controllers if users have them.



Touch and button interaction for oculus touch (generic)

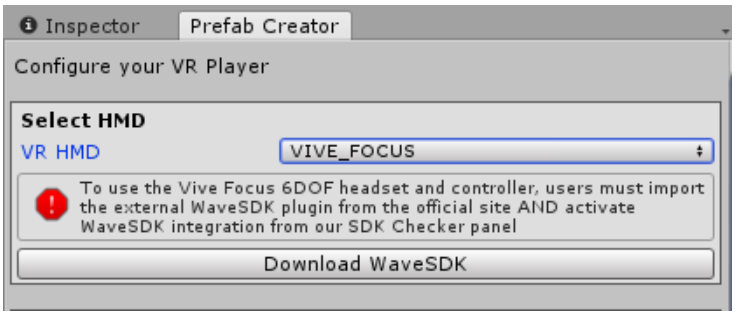
Our generic support for Oculus touch controllers via the GenericControllerTrigger now offers the possibility to read button and touch input separately.



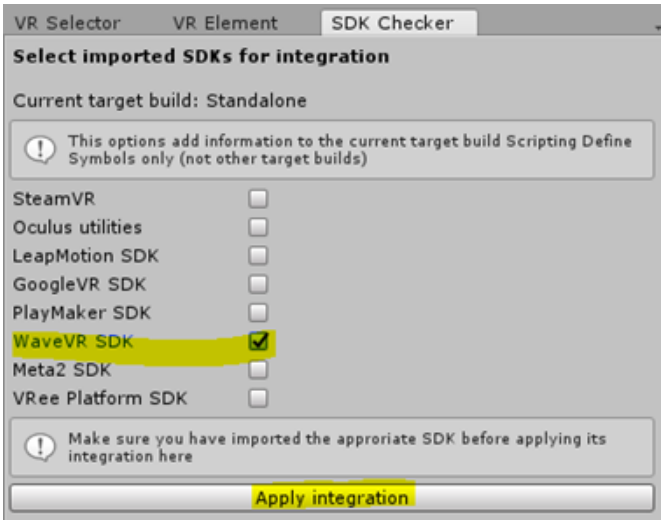
More support for WaveVR SDK

Support for headset and controllers via the VR prefab creator.

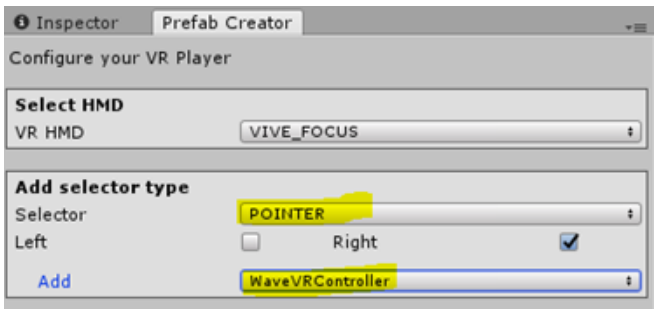
- Open Prefab Creator and select VR HMD Vive Focus.



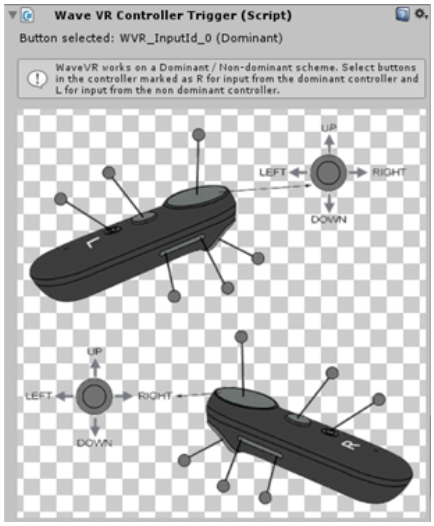
- If you don't have the WaveSDK imported, it will point you to the download page. Download and import
- Activate the WaveVR SDK integration in the SDK Checker and click on Apply integration. This settings are done per targeted platform, so you may have to switch to Android build platform in your Build Settings before this step.



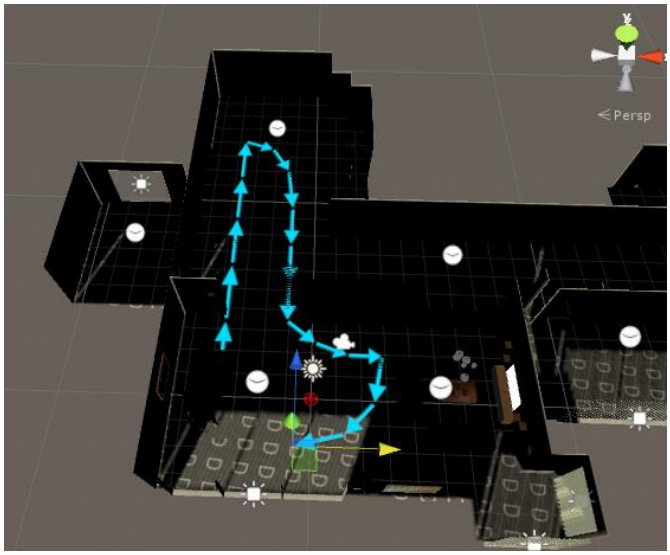
- Now you can progress with the VR Prefab creator to add selectors and any other extras. For example, if you want to add a Pointer selector that will be activated using the Vive Focus controller input, you can first select a Pointer as Selector and Add a WaveVRController trigger as shown:



We have also improved the support for Vive Focus input control via the WaveVRControllerTrigger, which can be used with any VR Easy that requires a trigger (Pointer and Touch selectors, VRSimpleFPSLocomotion, TeleportController,



Below is an example of a more complex path and the Path Curve GUI that generated it.



Inspector | Path Creator

Create an animated path by adding and positioning points

Path object	ArrowPath (SplineController)
General properties	
Arrow width	1
Scroll speed	1
Arrows per point	3
Vertical angle	0

Control points

Point0	point0 (Transform)			
	X -0.003	Y 0	Z -0.204	Delete
Point1	point1 (Transform)			
	X -0.188	Y 0	Z -0.318	Delete
Point2	point2 (Transform)			
	X -0.392	Y 0	Z -0.414	Delete
Point3	point3 (Transform)			
	X -0.392	Y 0	Z -0.414	Delete
Point4	point4 (Transform)			
	X -0.683	Y 0	Z -0.418	Delete
Point5	point5 (Transform)			
	X -0.886	Y 0	Z -0.372	Delete
Point6	point6 (Transform)			
	X -1.078	Y 0	Z -0.289	Delete
Point7	point7 (Transform)			
	X -1.548	Y 0	Z -0.222	Delete
Point8	point8 (Transform)			
	X -1.915	Y 0	Z -0.437	Delete
Point9	point9 (Transform)			
	X -2.2	Y 0	Z -0.616	Delete
Point10	point10 (Transform)			
	X -2.725	Y 0	Z -0.754	Delete
Point11	point11 (Transform)			
	X -3.06	Y 0	Z -0.781	Delete
Point12	point12 (Transform)			
	X -3.388	Y 0	Z -0.733	Delete

Add point

Redraw

You can also configure the Path directly from the game object's inspector, as each SplineController script will display the helper menu

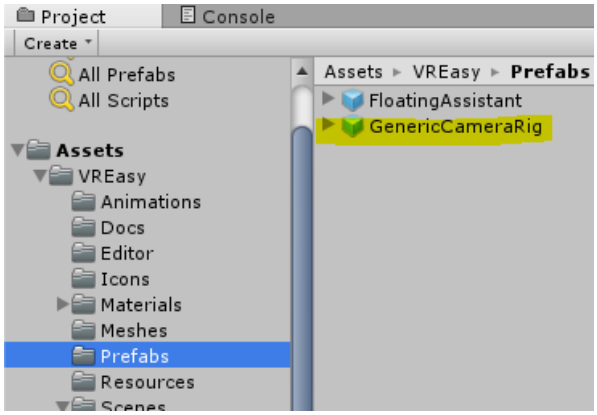
A video setting the Spline Path Creatore can be found here

VR Easy Prefabs

This sections describes the generic prefabs that are part of VR Easy.

Generic VR Controller prefab – Unity 5.6 and above

One of the biggest innovations of VR Easy version 1.3 is the inclusion of a Generic VR Controller prefab that offers similar functionality to the Camera Rig (SteamVR and Oculus) and GvrControllerPointer (Daydream) prefabs without the need for external SDKs. It offers a simple integration of VR camera and hand controllers that can be used with all major VR headsets supported by Unity (HTC Vive and Oculus Rift).

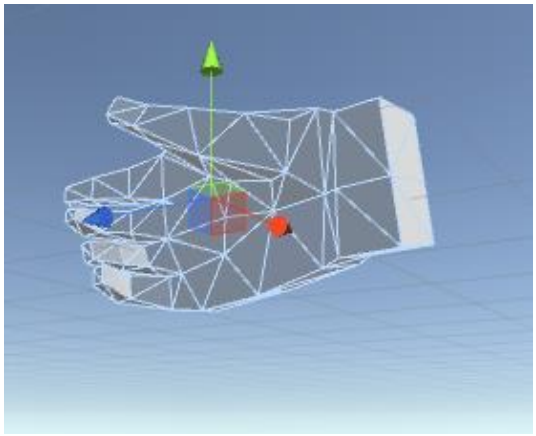


To use it, just drag and drop the *GenericCameraRig* prefab on *VR Easy/Prefabs* to your scene and remove any default cameras.

The prefab includes the following elements:

- Head: a Unity Camera that will render the VR scene directly to your headset
- Right and left controllers: they contain a script that will probe the physical controllers for their position and rotation via the UnityEngine.VR API and will update an associated hand model accordingly.

If you wish to use your own hand models this is possible by just removing the Hand game object under Right and left Controller objects and replacing it with your own. For a correct display, the prefab expects the orientation of the hand to match the following (blue Z axis, red X axis, green Y axis):



Note: because the controller script makes use of the UnityEngine.VR API, the controllers are only compatible with Unity 5.6 and above. The prefab can be used in Unity 5.4 as well but only for the camera functionality.

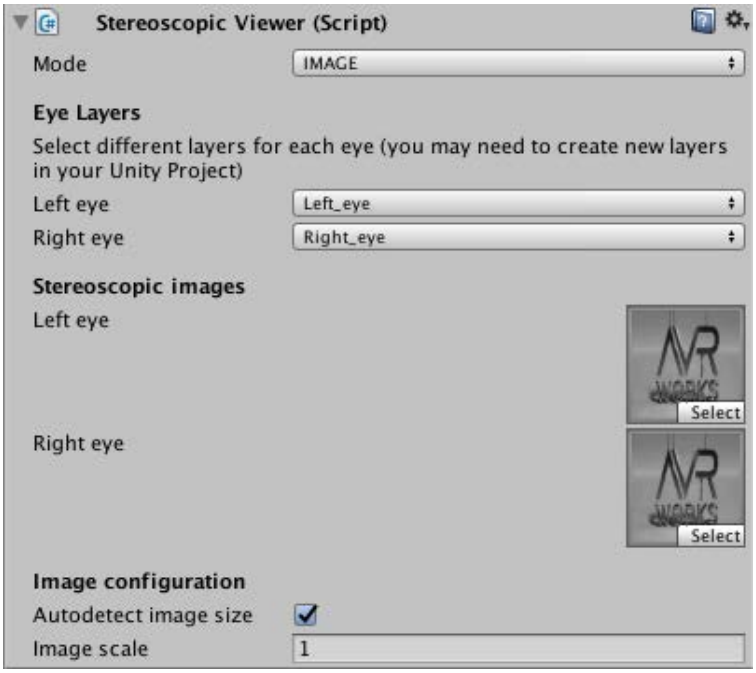
Once the prefab is in your scene, you can configure it as a selector just like with any other:

- Sight and Pointer Selector: use the Head object as the *Selector object*
- Touch Selector: use either of the controller objects (RightController and LeftController) as the *Selector object*.

Stereoscopic Image and video support

VR Easy now provides a prefab that allows you to display stereoscopic images and videos within your VR scenes. The Stereoscopic Viewer will display two images (or videos), one targeting each eye in your HMD (or side of the screen for mobile setups) giving the illusion of depth.
To add Stereoscopic Viewer to your scene, follow the steps:

1) Remove or disable the default camera in the scene.
2) Add the StereoscopicViewer prefab in VREasy/Prefabs folder to your scene.
3) Configure the StereoscopicViewer component.



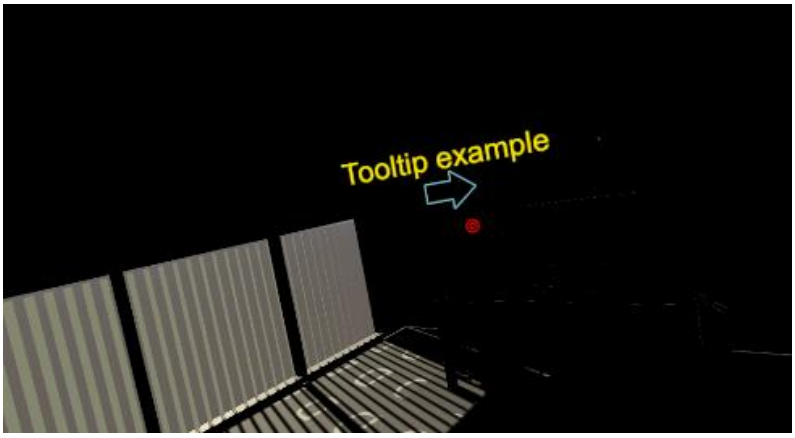
For Unity to correctly identify each image and render it to the corresponding eye, select a separate layer for each eye. It does not matter what they are called (in your project they may have any other name) but they must be different for each eye.

If the image mode is selected, two still images must be added as stereoscopic images. If the video mode is selected, then a stereoscopic video per eye must be added.
Further configurations are possible:

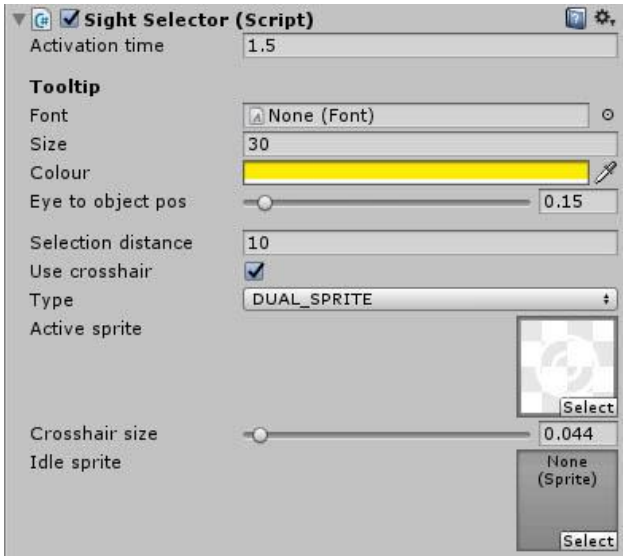
- Autodetect image size: the height/width ratio of the panel in which the image / video is rendered is detected automatically from the image.
- Image scale: scales the image by a factor defined by this property.

Tooltip

When displaying tooltips for object information, the default behaviour is to display it at a point between the selector (sight, touch or pointer) and the object. This is naturally a dynamic position that depends on the observer's position (the selector). Sometimes it is desirable to display the tooltip for a particular object in a specific location (for instance, a monitor in the scene, or a fixed billboard). Now users can determine, on an object by object basis, whether they want the tooltip to be anchored, by using the Anchored tooltip option that appears within the object's tooltip options.

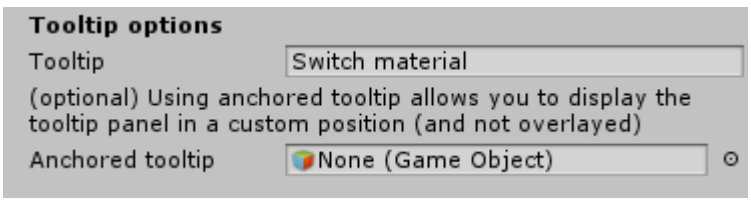


To enable your selectors to display tooltips, just click on the button *add tooltip* on any of your Selectors (Touch, Pointer or Sight). Only one tooltip needs to be added per scene, so if you have several selectors, you only need to do this once, and then other selectors will detect the tooltip manager automatically.



The most important one is to define a Font -by default, Unity comes with an Arial font that you can use off-the-shelf, but any font is compatible with our Tooltip. You can also specify the size and colour of the font, as well as the point at which the tooltip gets rendered; the value *Eye to object pos* determines where its rendered, 0 being near the VR element,

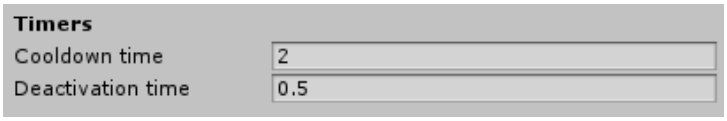
After adding a tooltip, you will be able to modify its properties from the selector inspector



Any game object in a scene can be used as an anchor, and the tooltip will appear always in that position (irrespective of the object's or the selector's position).

Various timers now exposed

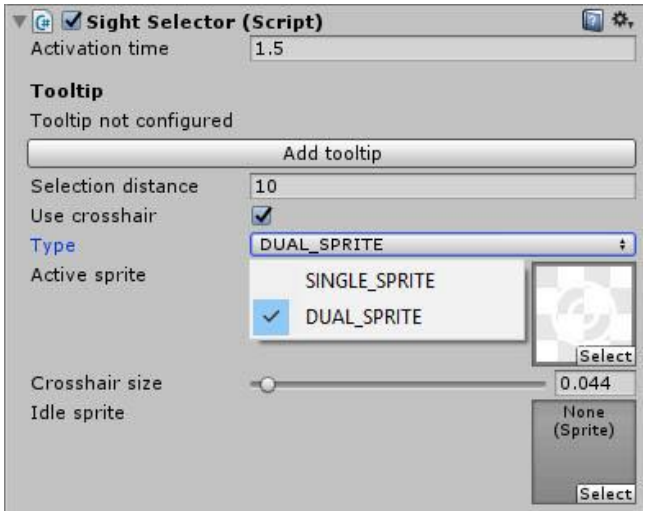
All buttons (2D and 3D) now have two timers exposed in the Inspector



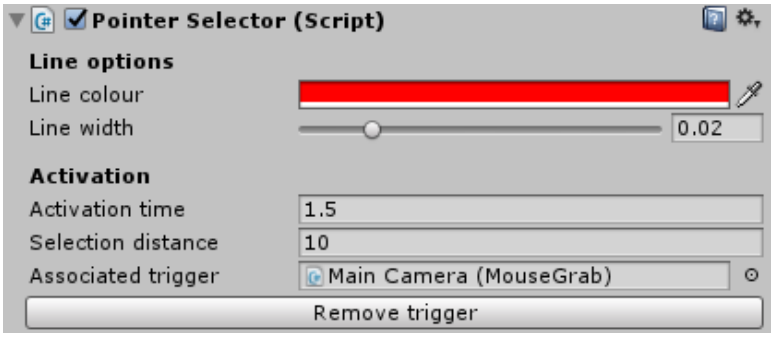
Cooldown time is used to determine for how long a button remains inactive after it has been activated - this is done to prevent multiple accidental selections.

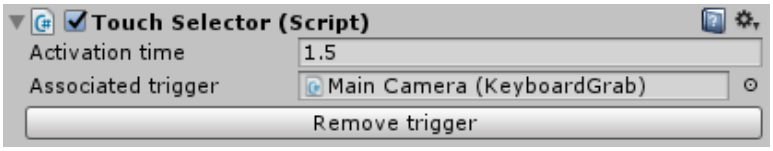
Deactivation time is used as the period between activating a button and setting it back to idle state. This mostly affects the aesthetics of the button, as it is the period in which it shows the Activate icon / highlighted colour.

In addition to those, all Selectors (Pointer, Sight and Touch) can define their own activation time in their respective Inspectors. This timer controls how long an interaction between Selector and VR element needs to last to trigger a selection.



Sight Selectors can now have two crosshair sprites, one for idle and a second one to be displayed when the sight selector interacts with a VR element (instead of using the default behaviour of colouring the sprite when active). You can choose to use single or dual sprites from the SightSelector inspector.

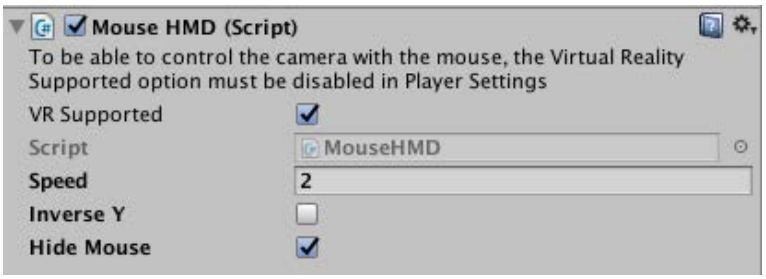




VR Easy Helper Scripts

Mouse HMD

Helper script that can be added to any game object. It allows users to control the rotation of that object with the mouse. It is located together with other helper scripts at 'VREasy/Scripts/Movement' folder.

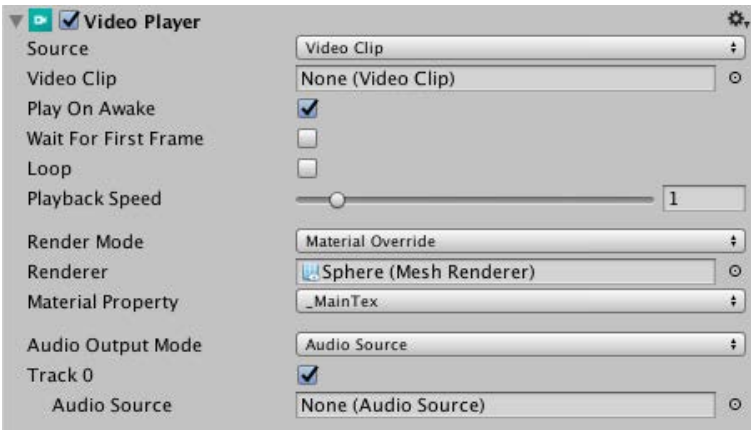


Ideally it is meant to be used as a substitute of HMD movement in situations where users want to control the camera / head but the physical HMD is not connected (e.g. for demonstrations on a PC or Mac).

To use it to control camera rotation, Virtual Reality Supported option in Unity Player Settings must be disabled (otherwise Unity will reset the camera position to match the HMD and mouse movement won't be reflected). To facilitate this, we have added a checkbox 'VR Supported' and a warning message to easily toggle this option from the inspector.

MoviePlayer

Since Unity 5.6 now supports reproducing videos through the *VideoPlayer* component, the *MoviePlayer* script in VR Easy now supports *VideoPlayer* too. Just add our *MoviePlayer* to the desired game object and configure its *VideoPlayer* component.



VR Grabbable

VRGrabbable objects are now highlighted

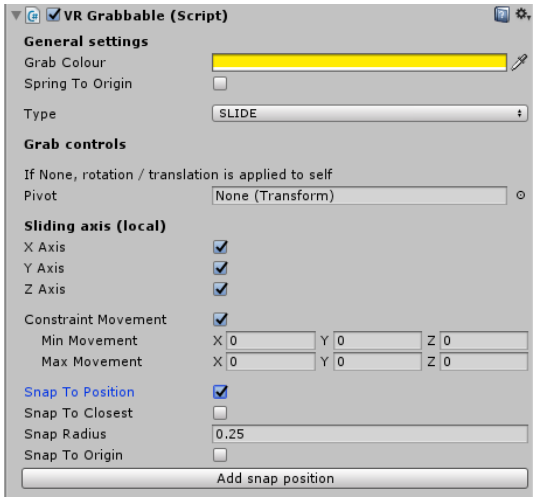
When interacting with **VRGrabbable** objects (via a touch or a pointer selector) now the object is highlighted with a custom colour to make the interaction more recognisable. The colour used can be defined per grabbable object, in the object's inspector. All **VRGrabbable** objects have multiple snapping positions when the grabbable is released (not just limited to snapping to origin). By enabling Snap to Position, users can add new snapping positions clicking on Add snap position button. Users can also select to always snap to the closest position by enabling Snap to Closest, or only snap to a position if it is close enough (via Snap Radius).

VRGrabbable types (drag, slide and rotate)

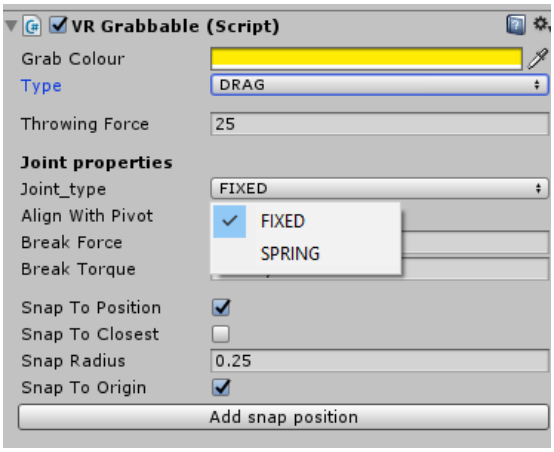
Now any grabbable object can select amongst 3 types of grabbing behavior via its Inspector:

1. **SLIDE:** Same behaviour as in previous versions; the object is slid along one or more axis (determined by ticking the appropriate axis option) but the object's rotation is unchanged

Slide mode can be constrained to any of the three axes (X, Y or Z), which will restrict its movement to the correspondent axis.



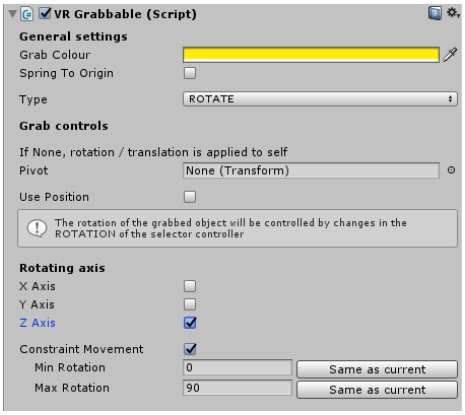
2. DRAG: The grabbed object will act as a real-life object that is being picked up, i.e. the object's position and rotation will follow the selector. This is done via Unity physics joints, so any movement applied to the object is done within the physics engine. In Drag mode, users can determine how the grabbable is joint to the selector. Two modes are offered via the joint type property: Fixed (rigid grip) and Spring (held using a spring). Each choice exposes parameters that customise the joint (see FixedJoint and SpringJoint Unity documentation for more information).



3. ROTATION constraints

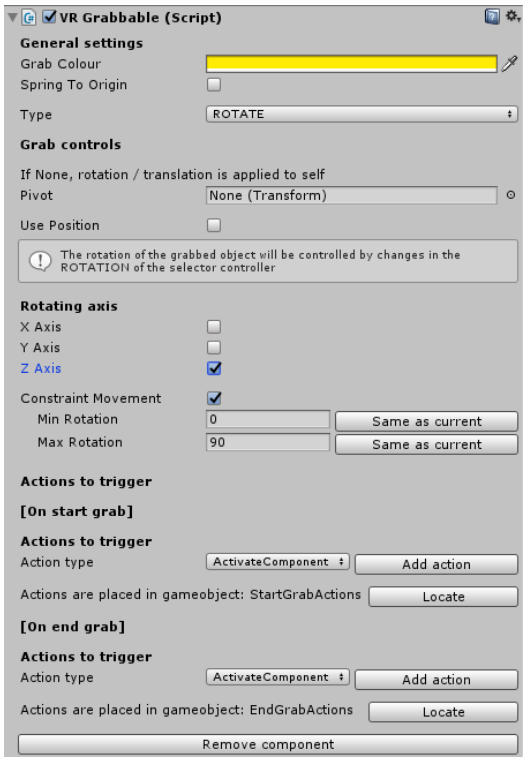
Only available for VRGrabbable rotation on a single axis.

Rotation for VRGrabbable objects can be constraint to within two angles in a single axis, so your grabbable objects will only rotate as far as you decide. To enable this feature, check the Constraint Rotation; this will display two properties that allow you to set the minimum and maximum angle of rotation: min rotation and max rotation.



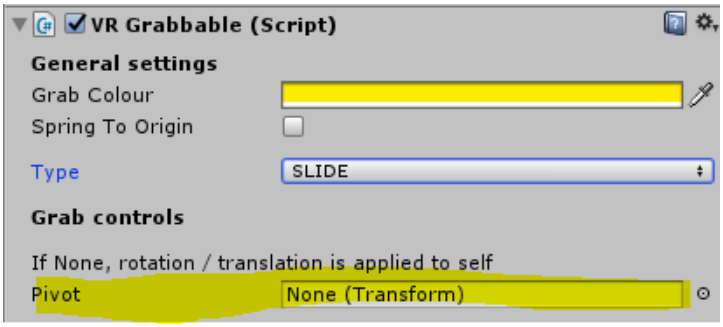
Action lists at the beginning and end of grabbing

In addition to Constraint Rotation, all grabbable objects have now the possibility of linking actions to the start and end of grab events, so you can trigger a set of actions when an object is grabbed and/or released. Both action lists can be configured from the VRGrabbable component.



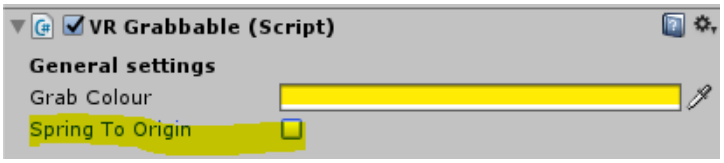
You can add and remove actions from this panel using the Add action and remove action buttons. As with any other action in VR Easy, to further configure the action you need to do so from the action component panel, which in the case of start and end grab action lists is situated in two child game objects of your grabbable object. To help you find them you can use the Locate button in the VRGrabbable component.

The VRGrabbable script allows now to specify a pivot object to make the rotation or sliding of the grabbable applicable to a third object. This is useful for instance in situations like pushing a lever, where the object grabbed is the handle but the rotation may be applied to the stick, instead of the handle. The pivot is only available to grab types Slide and Rotate.



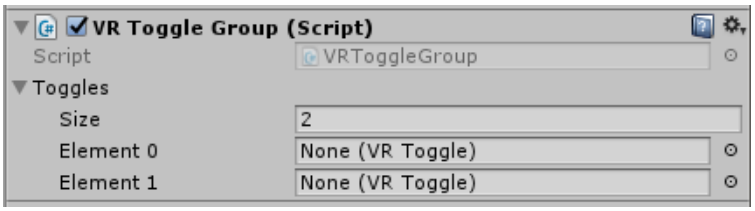
Just drag and drop the object you want the transformation to be applied to in the Pivot field.

In addition, grabbables can be set to spring to origin once they are released. To activate this behaviour, tick the Sprint To Origin checkbox in the VRGrabbable inspector.



VR Toggle Group

VR Toggle Groups can be used to create radio buttons out of a group of VR Toggle elements. This is useful when we want to restrict to one the number of active (toggled, or selected) Toggles out of a group of options.

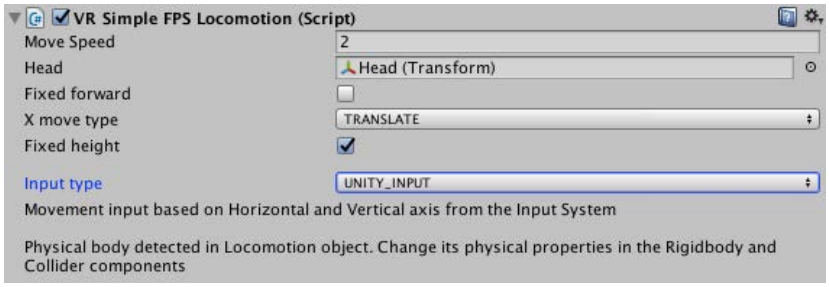


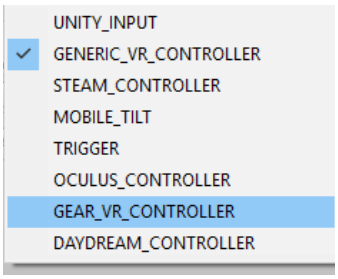
To use it, just drag the VRToggleGroup script (manually or through the top menu VREasy>Components>VRToggleGroup) to a game object, and link any number of toggles to the Toggles list.

Hint: through each VRToggle inspector you can specify if you want an element active or not at start. This way you can control which toggle is active from the group at start. Afterwards, the system manages selection and deselection of individual toggles to maintain only one active at any given time.

VR Locomotion

VRSimpleFPSLocomotion has been extended to accept mobile input (in addition to Unity in and Steam VR controller input). To make your player interact with the physical world -gravity, stairs, etc. whilst you move around, we have added the possibility to add a physical body to your VRSimpleFPSLocomotion component. This will in turn add a collider and a rigidbody to the object, which will make it interact with other physical bodies you may have in the scene, such as stairs, walls, etc. It will also make it operate under gravity, so make sure you have a floor to fall into!



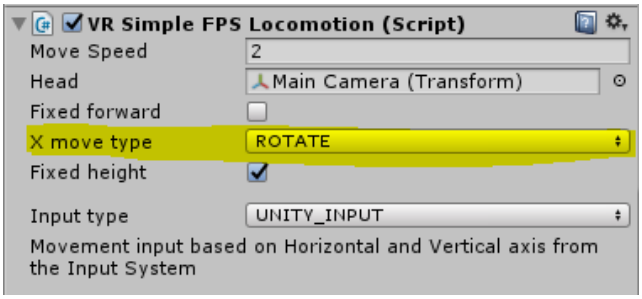


Note that for it to work, the appropriate SDK must be imported from the proprietary website and the SDK selected in the import settings (from VREasy > SDK Checker).

If you wish to control FPS movement with just your sight (ideal in mobile platforms and where other types of controllers are not convenient or available), use the Simple FPS Locomotion script included and select Mobile_Tilt as your Input type.

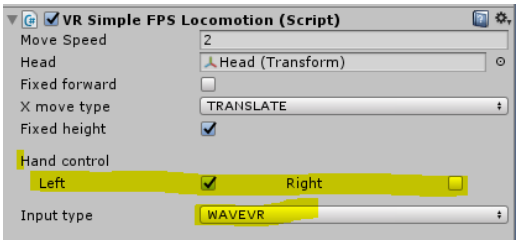
The *Start movement tilt field* indicates at which angle (looking down) your object starts moving forward. Once movement has been activated (by looking down), the object will continue moving forward until you look up the same angle as you did to start moving.

VRSimpleFPSLocomotion has been updated to include an option to allow to change the way the horizontal axis controls the movement of the player. The property *X move type* determines whether the horizontal axis is used to do strafing (moving from side to side) or to rotate the point of view.



It also includes a property, *fixed forward,* to force the player to have constant forward movement.

The VR Simple FPS Locomotion script now supports the option to determine which hand controller to use to control motion. This is only possible for those VR controllers that have support for left and right controllers.



We have included WaveVR input type to be able to use the Vive Focus controllers as motion input. This requires the use of the external WaveVR SDK (see SDK Checker for more info).

Screenshot capture

Users can now take screenshots of their VR world by simply adding the new *ScreenshotMaker* script to their scenes. This script uses VR Triggers to initiate the screenshot making action, hence an appropriate trigger must be linked and configured (as shown)

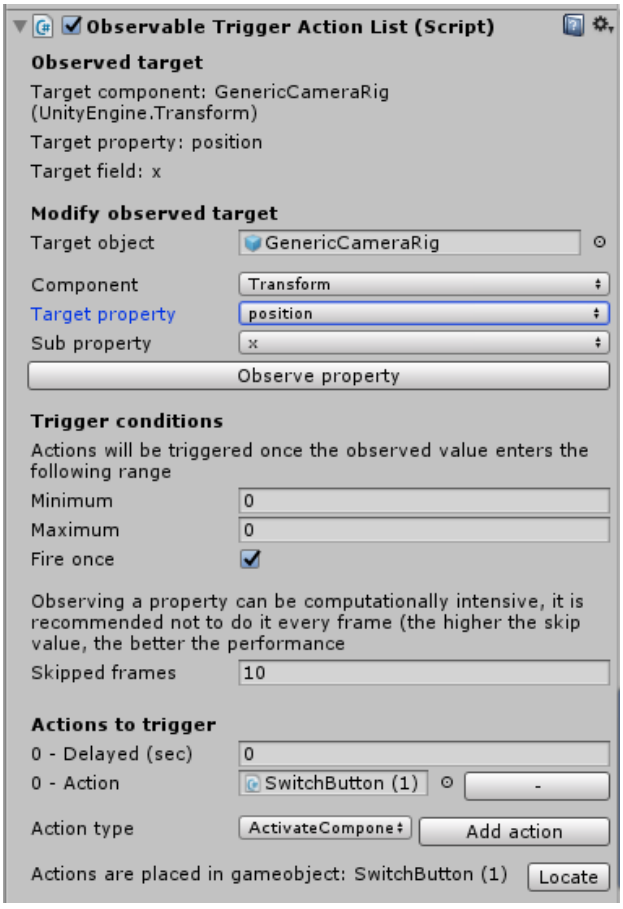
VR Easy includes a default “Camera flash shot” effect that gives feedback to the user to acknowledge the screenshot has been made. Even though the script can be located anywhere in your scene to create the screenshots, if you wish to have this image effect, *ScreenshotMaker* must be attached to the Main Camera object. If you are not using the Main Camera to render your VR world and are using other HMD prefab such as Camera Rig from SteamVR, you should place the script in the game object that contains the camera (in the case of Steam VR, that’s the child object called “Camera
(eye)”).

As with the store switch options, the screenshot is saved into a timestamped file on the application persistent path https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

Video tutorial on how to setup the screenshot capture option can be found **here**

Observable trigger

VR Easy actions are normally triggered via Selector-Selectable interaction (2D and 3D buttons, trigger areas, grabbables, etc.). From version 1.5, users can also trigger actions based on any property of any game object. The ObservableTriggerActionList observes the desired property (any single valued property within any component) and when it reaches a value (or a range of values), it triggers the associated actions.



The *observed property* can be selected by dragging the observed game object to the *target object* field. This will expose all its valid components, properties and subproperties (in the case of Vector or Colour properties). Once the property is selected, it can be observed by clicking on *Observe Property*.

The range of values of the observed property that will trigger the actions is determined by the *trigger conditions* values. Users can determine if the actions should be fired once (one time every time the observed property enters the range) or multiple times.

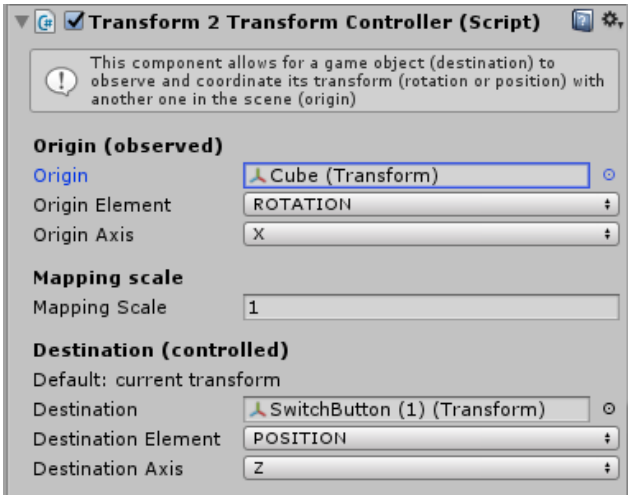
Because observing a property can be computationally expensive, this is done only in key frames. The user can specify the frequency of observing by modifying the *skipped frames* value (1 to check on every frame).

Transform controlling functionality

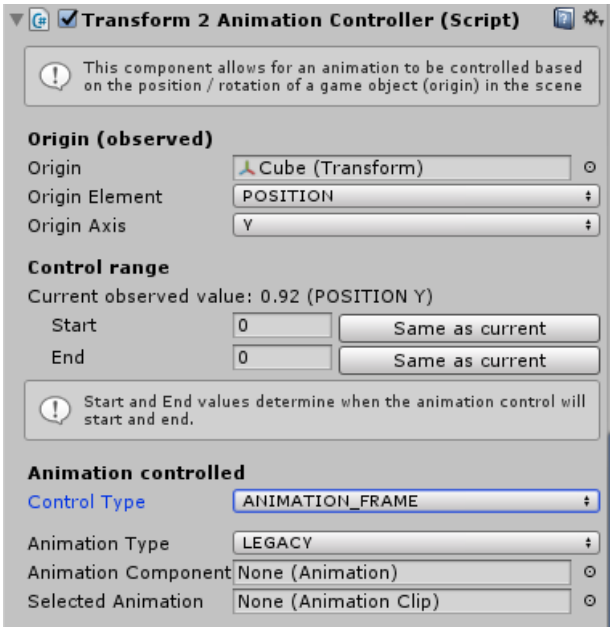
In VR scenes, it is often useful to control the position or rotation of an object based on the position or rotation of another one. For instance, this can be used when pushing a lever that controls the

elevation of a forklift. Two new components have been added to VR Easy in version 1.5 that facilitate object control: **Transform2Transform controller** and **Transform2Animation controller.**

Transform2Transform controller allows for a target game object (destination) to observe and follow the changes of a second transform (origin). The control is flexible and any transform element can target any other element -e.g. the origin rotation in axis X can control the Z axis position of destination.



The Transform2Animation controller takes the same principle (observing the transform of the origin object) and applies it to control an animation (both legacy and animator are supported). A *control* range can be defined; when the observed value (axis of rotation or position) enters this range, its value controls the state of the animation. Three type of state controls are available: Animation frame (the value in the range controls which frame in the animation is the object in), animation speed (the value in the range controls the speed at which the animation plays) and numeric parameter (only for animators; the value in the range controls the value of a parameter in the animator controller).

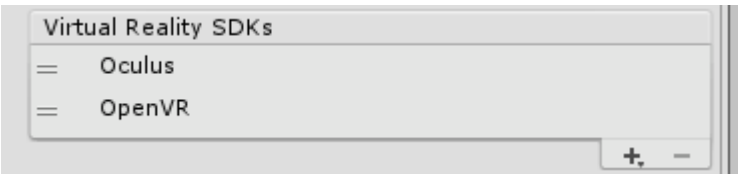


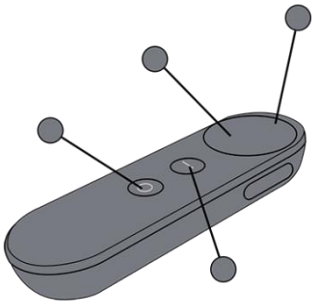
VR Triggers

All trigger interfaces for controllers now have a graphical interface to select input, substituting the list-based selection for a more intuitive interface. Tutorials on how to setup each Controller can be found **here:**

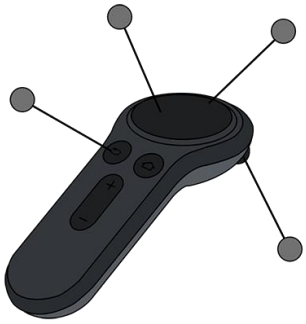
Oculus touch controllers bug fix

All oculus based controllers (GearVR controller, Oculus Go, Oculus touch) require that you have selected Oculus as your top option in Virtual Reality SDK in Player Settings.

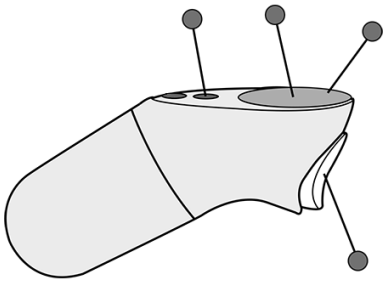




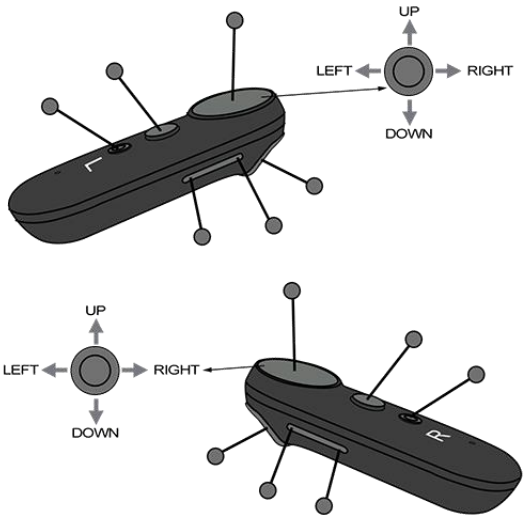
Google Daydream Controller



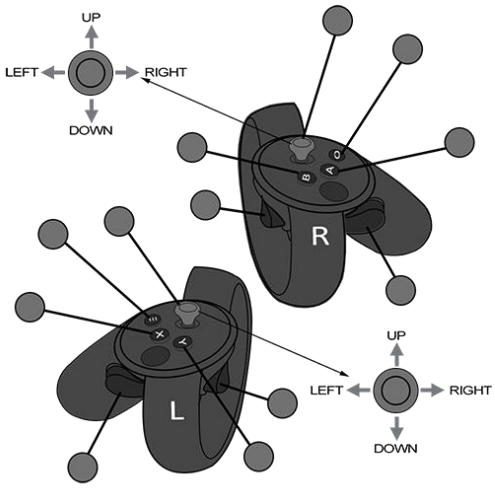
Samsung Gear VR Controller



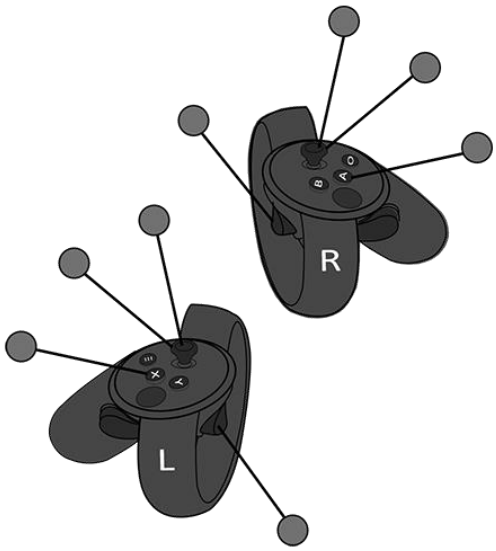
Oculus Go Controller



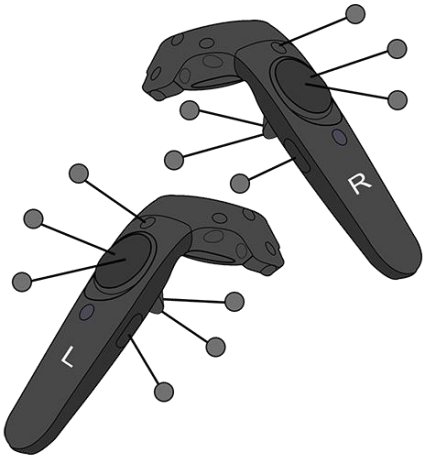
HTC Vive Focus Controller



Oculus Touch Controller

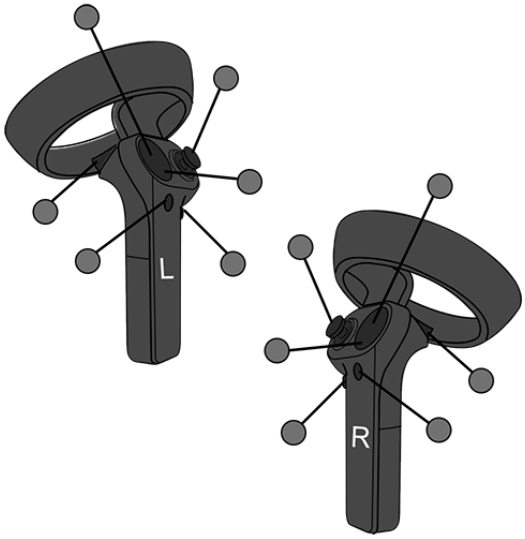


Oculus Touch Generic Controller



HTC Vive Controller

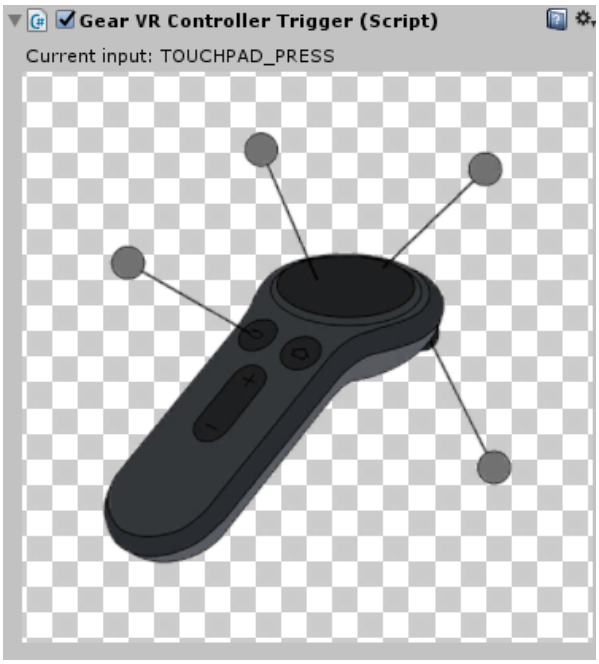
HTC Vive Generic Controller

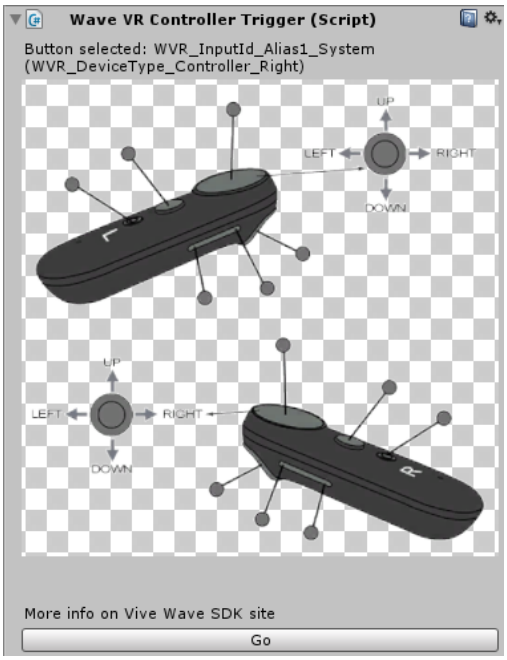


MR Controller

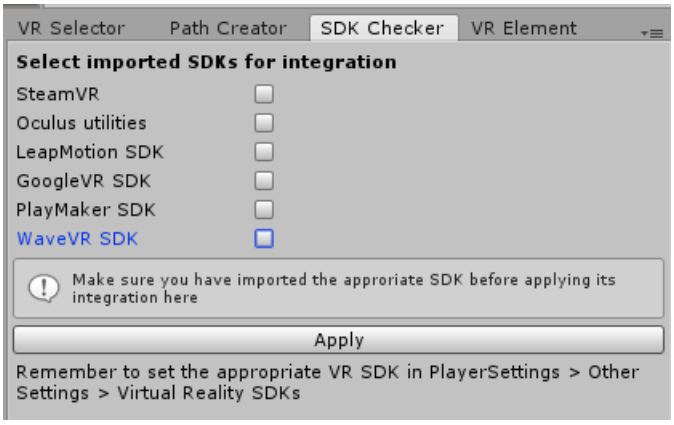
Added support for GearVR controller as trigger

Two new triggers have been added to interface with proprietary controllers: GearVR controller and WaveVR controller (Vive Focus).





Note that because interaction with buttons is handled via plugins, the correspondent SDK for the controller must be imported, and its VR Easy interaction activated via the SDK Checker. WaveVR controller (Vive Focus) requires the WaveVR SDK, whereas the GearVR controller requires the Oculus Utilities.

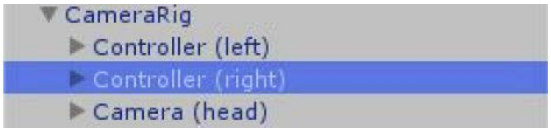


TeleportController

We have added a new way of moving players around. Now, in addition to the VRSimpleFPSLocomotion and the MouseHMD, we have included a TeleportController that allows teleport-like motion through a level.

The script can be found in the Assets/Scripts/Movement folder. To start using it, drag it to a game object in your scene that you want to use as a controller for teleporting. In this section we will assume the chosen controller is a SteamVR Controller prefab, but note that the steps for any other controller would be the same irrespective of the game object and the HMD used.

We recommend that you use the SteamVR prefab [CameraRig], since it has both the HMD and objects for both controllers. If you want to add the teleport controller to the right hand controller, drag the *TeleportController* script to the Controller (right) child gameobject of your [CameraRig].

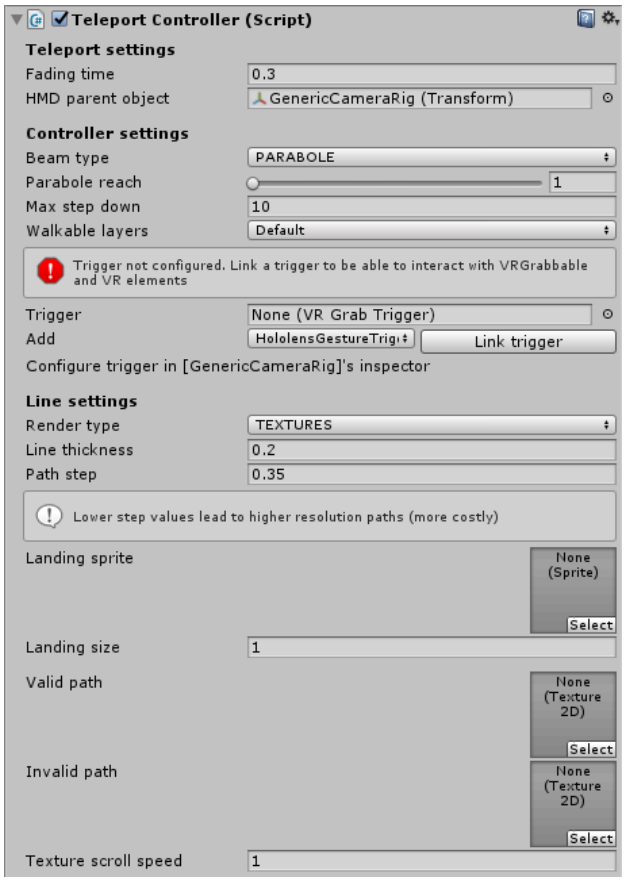


The GUI inspector for the script should guide you whilst configuring it.

Note: The teleport controller now supports moving across different heights (vertical freedom), removing the restriction it has in version 1.1.

Teleport Controller includes landing sprite

TeleportController offers the option to draw the teleport line as a line (with a Line Renderer) or as a special texture. This can be selected from the *Render Type* property. When TEXTURE is selected, the user can choose textures for when the teleport path is valid and invalid.

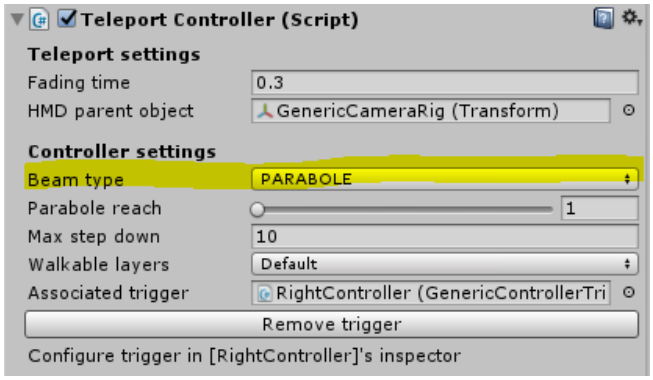


In addition, users can now select a ***Landing Sprite*** as well as a ***Landing size*** (scale of the sprite image), which will be drawn on top of the surface where the teleport line lands.

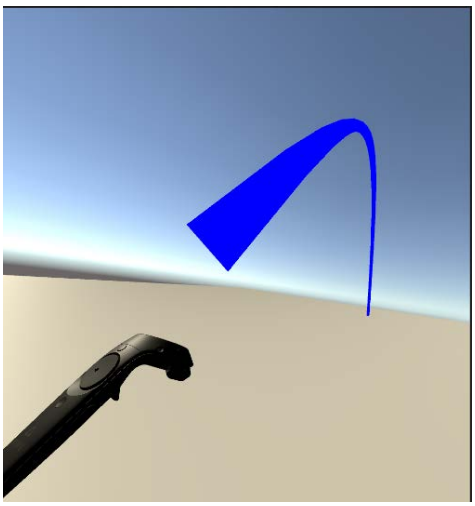
The top part (Teleport settings) allows you to customise general settings for your teleporter, whilst the bottom part (Line settings) exposes properties of the rendered line used to point to the destination whilst using the teleport controller. Here is a description of what each of those options do:

- *Walkable layers* → determines which objects in your scene will be usable to walk around (i.e. if you want your player to only be able to move on your floor, and you have assigned a new layer 'FLOOR' to your floor mesh, then this property should be marked as FLOOR; for more information about layers, please visit Unity's documentation)
- *Fading time* → When teleporting, a brief fade in / out animation is used to reduce the motion sickness effect that sudden changes of position has on players; this option determines how long this animation lasts
- *HMD parent object* → Drag and drop your HMD game object. If using the SteamVR prefab, this is your [CameraRig]; if using Unity's default main camera, then this is your Main Camera; it is in essence what will be moved when teleporting
- *Trigger* → As with other components in VR Easy (Locomotion, grab), you need an input trigger for the system to know when users wish to teleport. You can select a keyboard, a mouse or a SteamVR controller trigger.
- *Valid / Invalid dest colour* → these two options indicate what colours are used by the line renderer to indicate that the destination currently selected is valid or invalid.
- *Line thickness* → how thick the line rendered is (used to point to destinations)

The Teleport Controller supports now the option to define the beam type to be used. We now offer the traditional Parabole beam and an additional Straight beam.

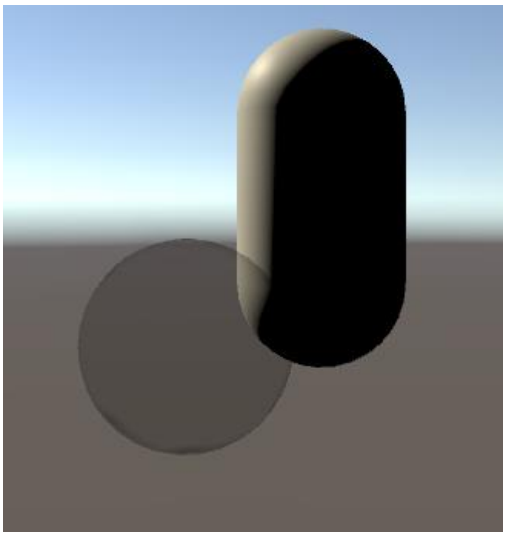


How to use it: The teleport controller is very easy to use. Following what seems to be a standard, users press and hold the trigger to select a destination; once the trigger is pressed, an arched line is rendered and users can choose easily their destination. Once the destination is found, release the trigger to instantly teleport to it.

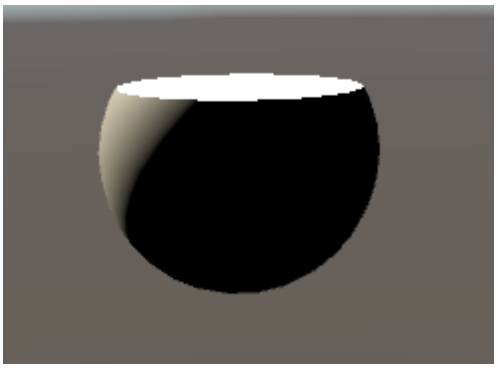


X-ray and cross section materials

Since VR Easy 1.2 introduced two new shaders: The X-ray sh: gives objects a see-through effect, similar to real x-ray machines (you can specify the general colour as a property of the material)

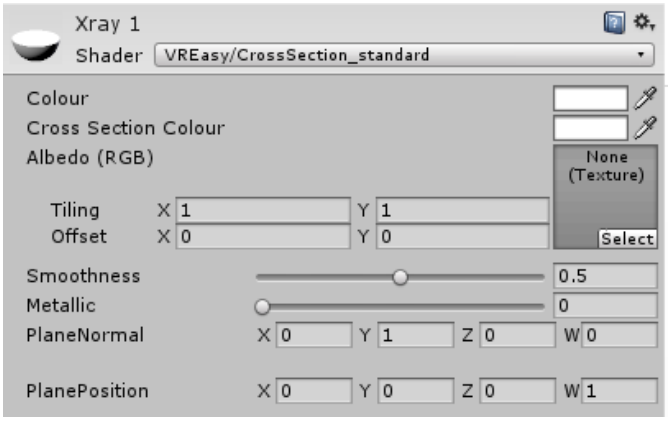


- The Cross-section shader: allows users to render only a section of the object, useful to see the inside of a complex structure



This two shaders can be used in any object material, and for convenience we have included two materials in *VReasy/Materials* (*Xray* and *CrossSection*).

The cross section shader, internally, computes an imaginary plane (defined by its position and its normal, or where it is facing) and renders only the part of the object that lies below it. You can control the imaginary plane through the shader properties:

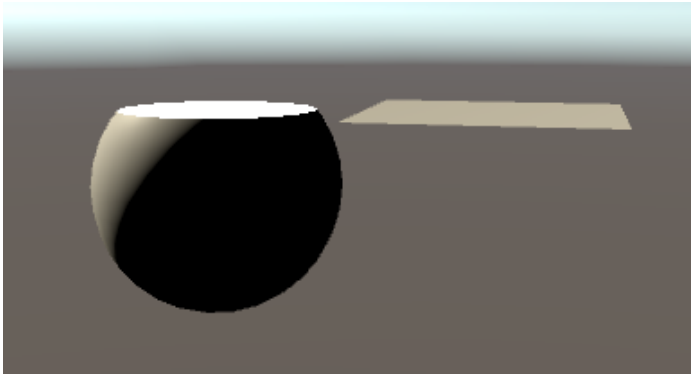


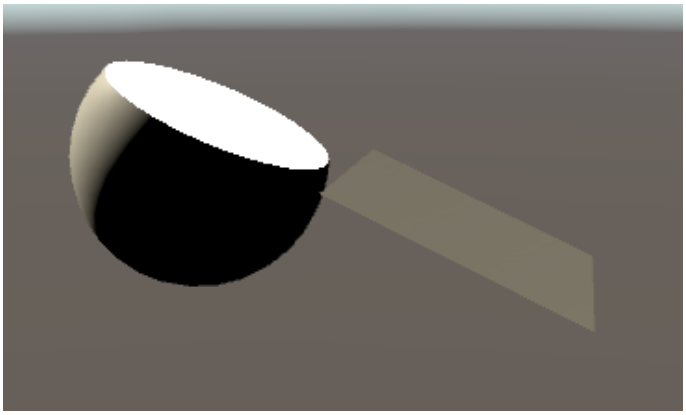
The two bottom properties, *Plane ormal* and *PlanePosition* define the imaginary plane. Note that the coordinates are in world position, so those will be compared with the coordinates of your object.

To make the control of the cross section shader more dynamic and intuitive, we have included a script under VREasy/Scripts/Demo named *CrossSectionController* that helps you use a real plane to cut through objects. To use it, simply:

1) Attach the script to the game object with the Cross section material
2) Create a plane in your scene (actually, any game object can be used)
3) Link your plane to the *Cross Section Plane* property of the *CrossSectionController*

After doing so, you can move and rotate your plane and see the effects it has on rendering your object! Note that naturally the plane must intersect the object to see the effect (though they do not have to physically be on the same position). Also note that the Cross Section Plane object does not need to be rendered, it can be an empty shell.





Now by controlling the plane, you dynamically cut through the object. You could, for example, attach the plane to a SteamVR controller (or directly use the controller as Cross Section Plane object) and then use your hands to cut through your sphere! For more details, please check out our video on the cross-section and xray materials

Video tutorial showing how to setup and use the X-Ray material and Cross Section can be found here

Extending VR Easy

Creating your own VRActions

VR Easy ships with a good collection of actions that can be triggered via VR elements. Although we have aimed to include the most popular use cases, we acknowledge some very specific functionality you require may have slipped through the cracks! That is why we have designed the system to be easily extendible via scripts.

Technical details

At the core of VR Easy we use reflection to be able to read code (ours and yours) and add functionality to the base system. All VRActions -Switch, PlayAnimation, MoveObject, ActivateVRElement, etc.- derives from a base class named **VRAction**. This is an abstract class that indicates that any action should implement a particular method, *Trigger()*. This method is the one that VR Easy calls when your VR Element is selected via either a TouchSelector or a SightSelector, and should contain whatever piece of code you want to invoke.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;
using System.Linq;

namespace VREasy
{
    public abstract class VRAction : MonoBehaviour
    {
        public abstract void Trigger(); // must be overriden, determines the action to be triggered by the VRSelectable

    }
}
```

How to add actions

To add new actions, create a new script and make sure:

1) It is included in the namespace VREasy (as in the screenshot above)
2) It derives from the VRAction class

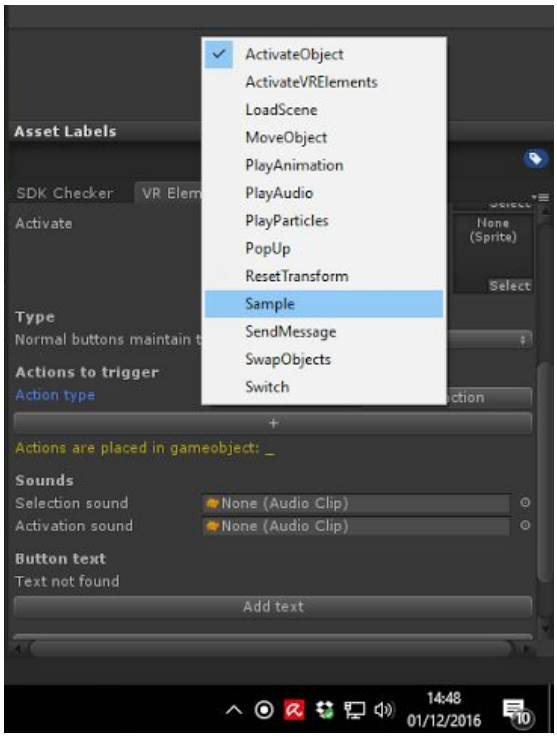
Here is a screenshot of a dummy new action called SampleAction:

```
namespace VREasy
{
    public class SampleAction : VRAction
    {


    }
}
```

Because we are inheriting from a class with an abstract method, we must implement the method *Trigger()* -if you don't, Unity will throw an error such as *SampleAction does not implement inherited abstract member VRAction.Trigger()* . Make sure you override such method and fill it with your functionality.

```
namespace VREasy
{
    public class SampleAction : VRAction
    {
        public override void Trigger()
        {
            // YOUR CODE GOES HERE!
            throw new NotImplementedException();
        }

    }
}
```

Once you have compiled your code, you will be able to use SampleActions just as you would any other action. For instance, when creating a 2D Button, you can now link it to the new action:



Note that if your script name contains the word Action, it will be cropped when displayed in the menus -as above, where the script *SampleAction* is displayed as *Sample*. This is purely cosmetic and does not affect the functionality.

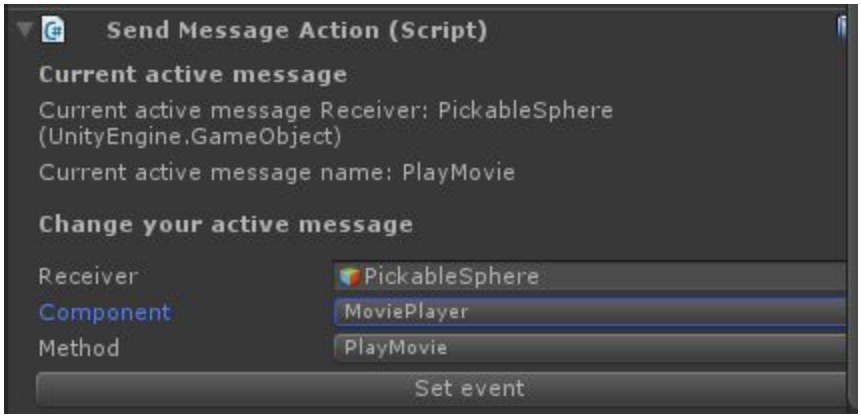
Extending Actions via SendMessageAction

It is possible that you already have a working script responsible for some specific behavior and even though you wish to link this to VR Easy, you do not want to create a VRAction class. In fact this is so common that we have created an Action that essentially links a VR element with a *foreign* script: *SendMessageAction*. With it you can trigger public methods on any script -or even component- on a specified game object (click here to see a video tutorial on how to use SendMessage).

We have included an example of how to use this type of interaction via the MoviePlayer script. The script itself is very basic, it uses Unity MovieTextures to play a movie on an object, and has a separate audio track for the sound. There are 3 public methods: *PlayMovie, PauseMovie* and *StopMovie*.

```
public void PlayMovie()
{
    Debug.Log("Play");
    if(movie) movie.Play();
    if (audio) audio.Play();
}
public void PauseMovie()
{
    Debug.Log("Pause");
    if (movie) movie.Pause();
    if (audio) audio.Pause();
}

public void StopMovie()
{
    Debug.Log("Stop");
    if (movie) movie.Stop();
    if (audio) audio.Stop();
}
```

When it is attached to a game object, one can call any of those public methods from a SendMessageAction. Once that action is linked to a VR Element -using the VR Element GUI-, and the *receiver* game object is set using the object containing our MoviePlayer script, you can choose which method is called when the action is triggered via the intuitive GUI in the SendMessageAction. Once you have chosen the target Component (which script you want to call) and the target Method (which function you want to call), click on Set Event to finish the link.



Creating more Grab Triggers

This section shows how you can extend the basic offerings of VR Easy when it comes to triggering a Grab action. This is required when you have a TouchSelector that you want to use to grab VR Grabbable objects in your scene, but the default Grab Triggers are not good enough for your project (please read this document on how to setup your scene to grab and move objects in your scene. A step by step video tutorial will follow soon).

We refer as "Grab triggers" any action(s) that initiates and finishes a grab interaction. For instance, we have included 3 types of grab triggers with VR Easy: *Keyboard*, *Mouse* and *SteamVRController*. Each of them can be used by a TouchSelector to initiate grab actions.

As with VRAction, we use reflection to read your code to incorporate custom scripts to VR Easy system. All grab triggers must derive from the class VRGrabTrigger, which is a class that enforces its children to implement the method *bool Triggered()*. This method needs to return a boolean (true or false) that indicates whether the grab should be active or not.

```
namespace VREasy
{
    public abstract class VRGrabTrigger : MonoBehaviour
    {
        public abstract bool Triggered();
    }
}
```

Let's see an example to understand how this works. The class MouseGrab, shipped with VR Easy, includes the full implementation on how to use a mouse button as a grab trigger.

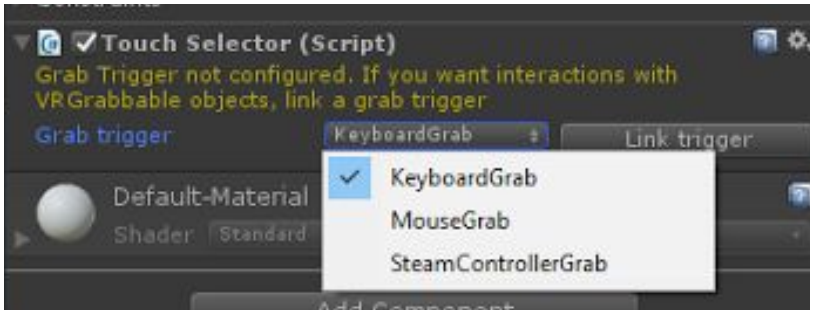
```
namespace VREasy {
    public class MouseGrab : VRGrabTrigger
    {
        public int mouseButton = 0;
        public override bool Triggered()
        {
            return Input.GetMouseButton(mouseButton);
        }
    }
}
```

This class simply returns true when the button with id specified by *mouseButton* is being pressed. So if *mouseButton* then the grab is triggered when the left-click button is pressed, and the grab is released when the button is not pressed.

To implement your own behavior, you need to create a new script with the following features:

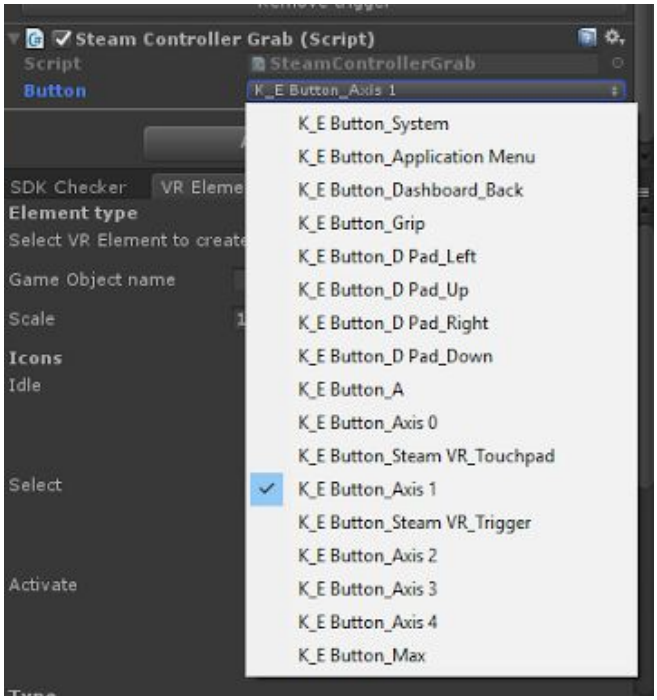
1) Must reside in the VR Easy namespace
2) Must derive from the *VRGrabTrigger* class

Once you have defined your own behavior, you will be able to use your new class in your TouchSelector objects. (please read this document on how to setup your scene to grab and move objects in your scene. A step by step video tutorial will follow soon).

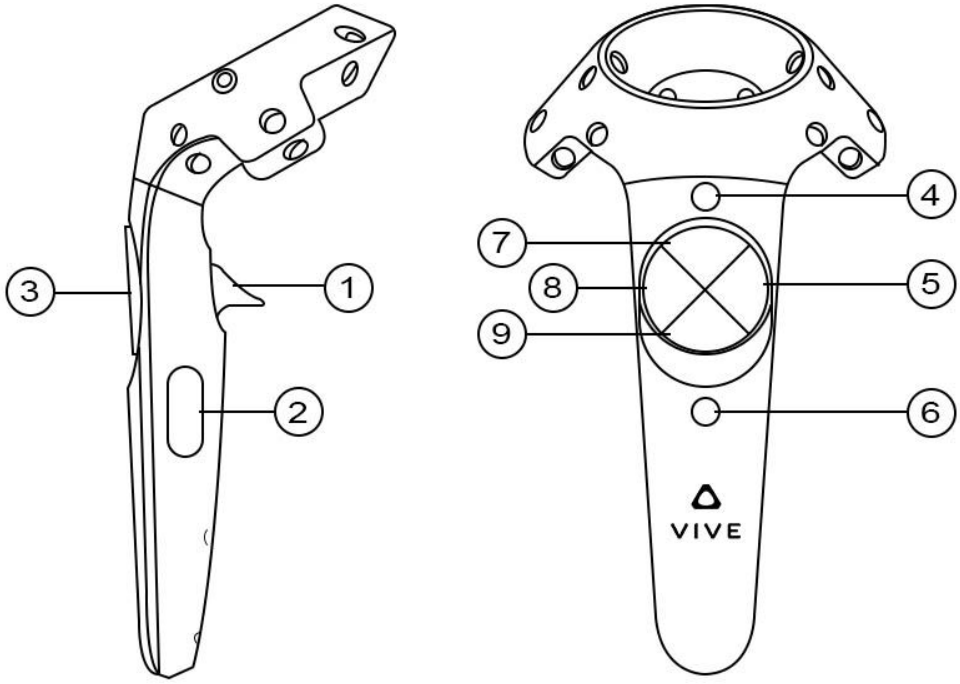


Steam VR Controller buttons

We have included a class that uses the Steam VR Controller buttons to trigger grab actions.



To configure it, all you need to do is choose which button will be read as trigger input. You can set this up via the Inspector of the *SteamControllerGrab*, which includes a public property named *Button*. This property enumerates all the buttons and axis on a Steam Controller. For a description of which name correspond to which button, use the reference below you can see the mapping.



Mapping the Controller buttons:

1 - KE_Button_Application_Menu

2 - KE_Button_D_Pad_Left

3 - KE_Button_D_Pad_Up

4 - KE_Button_D_Pad_Right

5 - KE_Button_D_Pad_Down

6 - KE_Button_System

7 - KE_Button_Steam_VR_Trigger (for the trigger click) and KE_Button_Axis_1 for the trigger axis

8 - KE_Button_Grip

9 - KE_Button_Steam_VR_Touchpad

FAQ

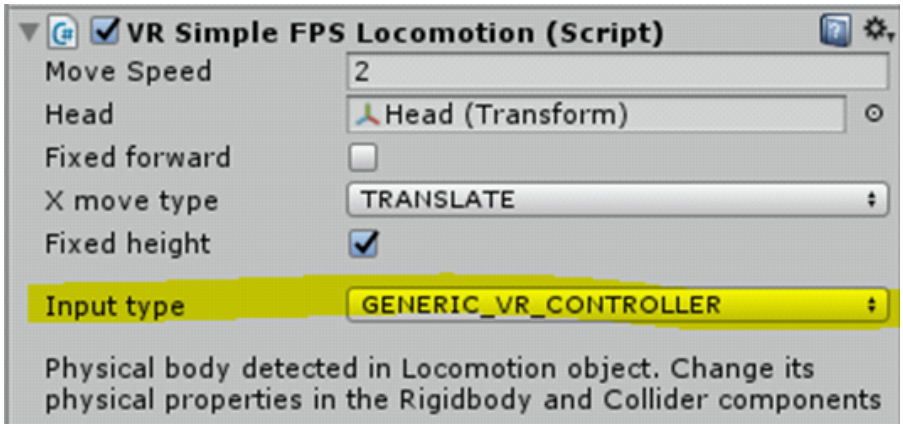
This sections describes the **FAQ** for **VR Easy**.

VRSimpleFPSLocomotion using GenericVRcontroller

Unity interfaces with OpenVR controllers via the Input Manager class, using the following mapping
https://docs.unity3d.com/Manual/OpenVRControllers.html

In order to use any of the buttons or axis in an organic way, one needs to map the project´s Input Manager accordingly.

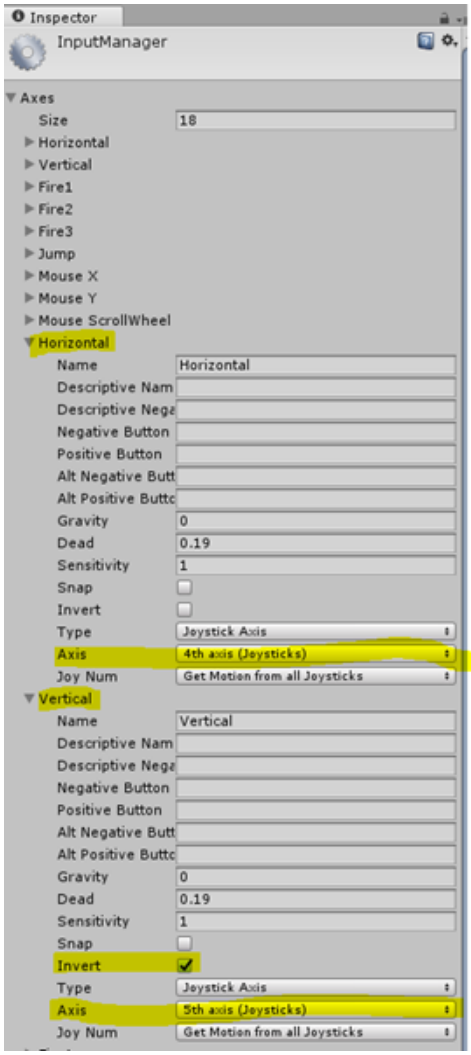
Using VRSimpleFPSLocomotion to move your character around with our Generic VR Controller input type is a good idea as you do not have to rely on several SDKs and different scene configurations to match your hardware controller. Instead, the Input Manager class interfaces with the specific controller so you don´t have to change anything.
In the current implementation, the ***Generic VR Controller*** looks at the *"**Horizontal**"* and *"**Vertical**"* axis in your Input Manager. By default, those axes are wired to be controlled by the 1st and 2ndj joystick axis, which according to the mapping in the link above corresponds to the left controller touchpad in SteamVR controllers and the primary thumbstick in Oculus. Since the SteamVR right controller and the secondary thumbstick in Oculus affect the 4th and 5th axes, you need to change the Input Manager class if you wish to use those for movement in VRSimpleFPSLocomotion.



First, open the Input Manager configurator in Unity by going through the menus *Edit > Project Settings > Input*. You should see two ***"Horizontal"*** and two ***"Vertical"*** input axes. Open the second set of "Horizontal" and "Vertical" axes and make sure they are set to ***Type = Joystick*** Axis. Then choose the appropriate Axis, following the official mapping:

https://docs.unity3d.com/Manual/OpenVRControllers.html

As a reminder, SteamVR right controller and the Oculus touch secondary thumbstick is the 4th axis for horizontal movement and 5th axis for vertical.



Consider the ***Invert*** checkbox for the *"Vertical"* axis to match your preferences.