Humanoid Control Documentation

Version: 3.3

Date: 18 September, 2020

Table of Contents

Prerequisites

- Unity 5.5 or higher

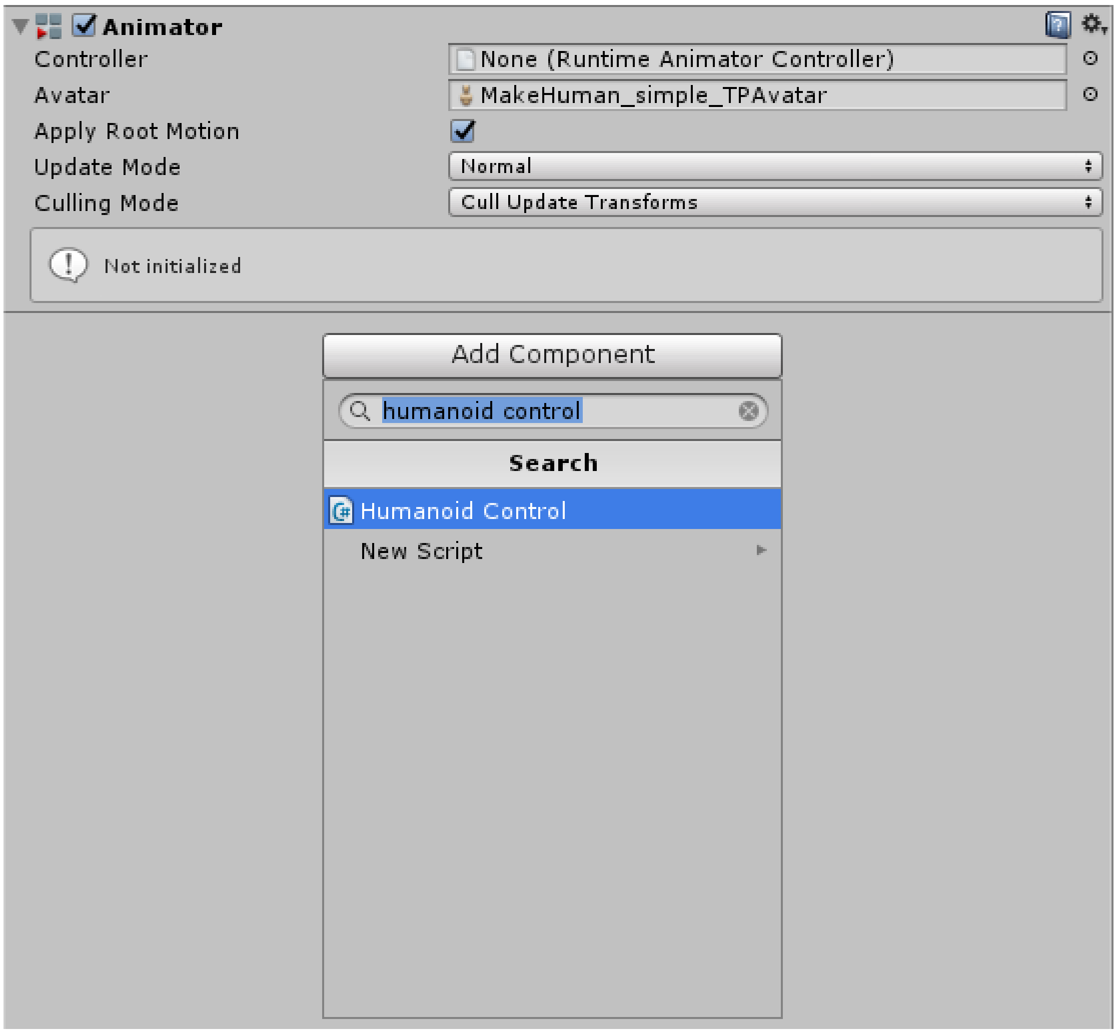
Getting started

There are two ways to include an humanoid into a scene: starting with an avatar and starting with the Humanoid Control script.

Starting with an avatar

In this case you start by putting an avatar prefab into the scene. This avatar needs to fulfil the requirements of a Mecanim avatar. A couple of avatar prefabs are included in the package in the Humanoid/Demo/Characters/MakeHuman_simpleRig folder.

The next step is to attach the Humanoid Control script to the avatar. You can do this by selecting the *Add Component* button in the Inspector and selecting the *Humanoid Control* script.



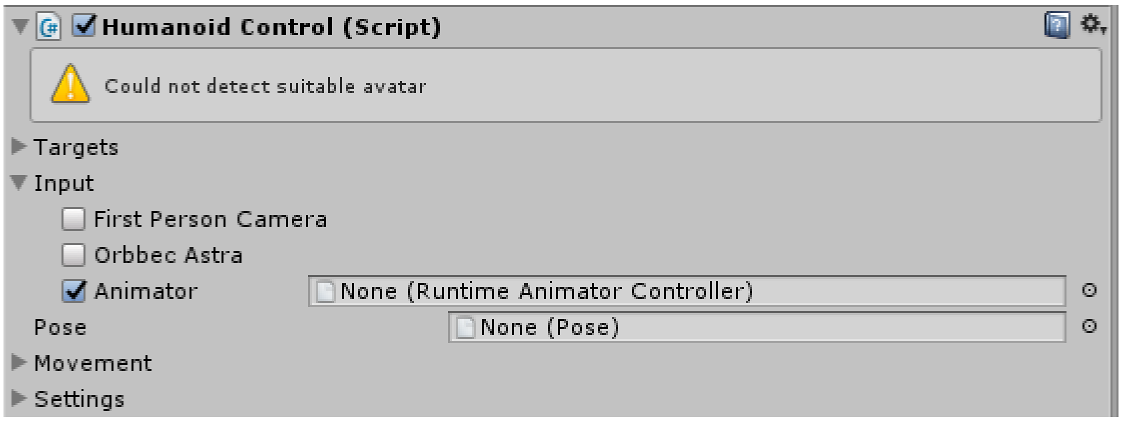
Animator

Controller: None (Runtime Animator Controller)
Avatar: MakeHuman_simple_TPAvatar
Apply Root Motion: ✓
Update Mode: Normal
Culling Mode: Cull Update Transforms

Not initialized

Starting with the Humanoid Control script

In this case we start with an Empty GameObject. We will then add the Humanoid Control script to the object by selecting the *Add Component* button in the Inspector and selecting the *Humanoid Control* script.


Humanoid Control (Script)

Could not detect suitable avatar

Targets

Input

First Person Camera

Orbbec Astra

Animator None (Runtime Animator Controller)

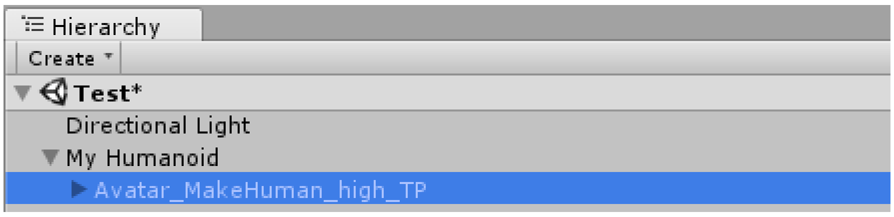
Pose None (Pose)

Movement

Settings

You will see that the script could not detect a suitable avatar because there isn't an avatar attached yet.

We can now add an avatar by dropping an avatar onto the Empry GameObject we created. It will become a child of this GameObject.


Hierarchy

Create

Test*

Directional Light

My Humanoid

Avatar_MakeHuman_high_TP

This makes it easier to replace an avatar at a later moment.

Video

[Humanoid Control Introduction]

Extension Configuration

Humanoid Control can be used in combination with many different body tracking devices. The availability of these tracking extensions depends on the purchased package.

Preferences

In the Preferences you can select which features are available for this project. Disabling a feature implies that all relevant code for that feature is excluded from the build. This can help to reduce the build size, but it also makes it possible to build an executable which does not include certain libraries, as may be required for a submission to the Oculus Store for example.

Some extensions require additional packages to be installed for support. As soon as these additional package have been installed, the extension will be enabled automatically.

The Preferences can be found in the Edit Menu->Preferences->Humanoid section.

OpenVR (SteamVR compatible devices)

OpenVR (SteamVR) is supported for Head Mounted Devices and Hand Controllers.

Note: *Humanoid Control does not need the SteamVR SDK installed in the project because it uses the low-level OpenVR API*I.

The Vive Trackers are documented separately.

Information about the use of VRTK with Humanoid Control is found here.

Prerequisites

Humanoid Control

OpenVR is supported in the Humanoid Control VR, Plus and Pro packages

Unity

OpenVR is supported in Unity 2017.4 LTS up to Unity 2019.4.

Unity version 2020.1 and higher are supported using Unity XR.

Hardware

HTC Vive, Valve Index, Oculus and Mixed Reality are supported through OpenVR .

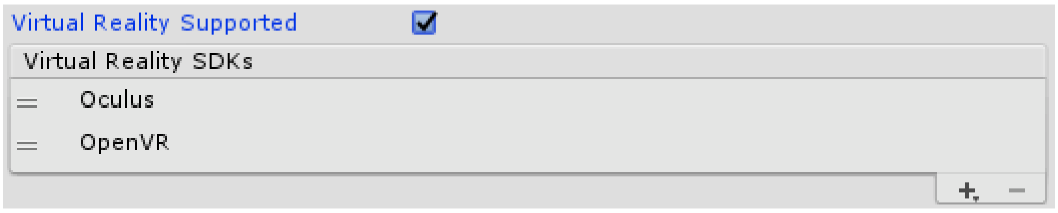
Operating System

OpenVR is supported on Windows 10.

Setup

Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *OpenVR* needs to be added to the *Virtual Reality SDKs*.

Note: if both Oculus and SteamVR need to be supported in one build, make sure *Oculus* is listed higher than *OpenVR* in the *Virtual Reality SDKs:*

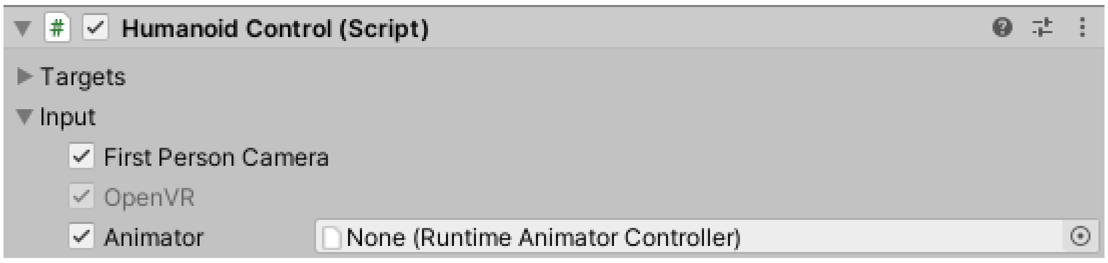

Virtual Reality Supported ✓
Virtual Reality SDKs
Oculus
OpenVR

In Humanoid Control v3 and higher, OpenVR needs to be enabled in the *Edit Menu->Preferences->Humanoid->SteamVR Support.* Legacy OpenVR is no longer supported.

Disabling *SteamVR Support* ensures that no code related to SteamVR is included in the build.

Configuration

To enable body tracking with SteamVR for an avatar, *OpenVR* needs to be enabled in the Humanoid Control component.

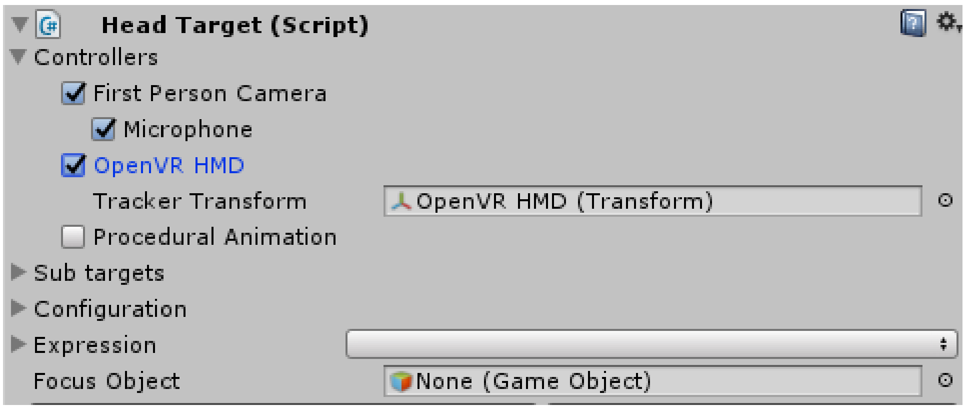

Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ OpenVR
✓ Animator None (Runtime Animator Controller)

This option will be enabled automatically when OpenVR is added to the Unity Virtual Reality SDKs.

Head Target

First Person Camera needs to be supported for the OpenVR HMD. For convenience, this option is also found on the Humanoid Control script.

The *OpenVR HMD* option is added with a reference to the Real World HMD:

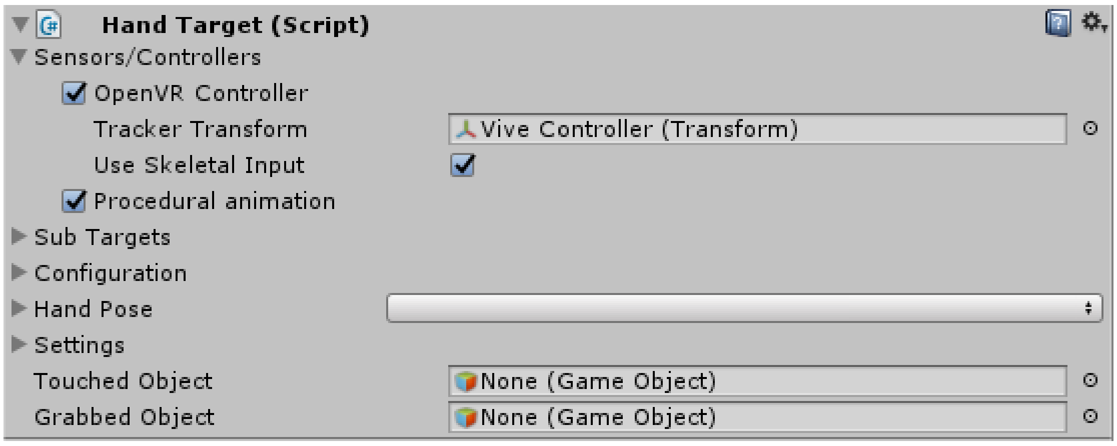

Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone
✓ OpenVR HMD
Tracker Transform OpenVR HMD (Transform)
Procedural Animation
Sub targets
Configuration
Expression
Focus Object None (Game Object)

Hand Target

The Steam VR Controller needs to be enabled on the Hand Targets for controller support.

Controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.

Next to that, as option is available to enable the SteamVR Skeletal Input. In this mode the hand poses will be directly taken from the SteamVR tracking information.


Hand Target (Script)
Sensors/Controllers
✓ OpenVR Controller
Tracker Transform Vive Controller (Transform)
Use Skeletal Input ✓
✓ Procedural animation
Sub Targets
Configuration
Hand Pose
Settings
Touched Object None (Game Object)
Grabbed Object None (Game Object)

Controller input

The buttons of the Steam VR controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

Menu button	controller.left.option, .button[0]
Touchpad touch	controller.left.stickTouch
Touchpad press	controller.left.stickButton
Touchpad movements	controller.left.stickHorizontal, .stickVertical
Trigger	controller.left.trigger1
Grip	controller.left.trigger2

Right Controller

Menu button	controller.right.option, .button[0]
Touchpad touch	controller.right.stickTouch
Touchpad press	controller.right.stickButton
Touchpad movements	controller.right.stickHorizontal, .stickVertical
Trigger	controller.right.trigger1
Grip	controller.right.trigger2

Calibration

The tracking position and orientation can be calibrated during gameplay by calling the Calibrate() function on the HumanoidControl object. This is often implemented using the Controller Input component.

Oculus Rift/Touch

Oculus Rift and Touch are supported to track head and hand of an avatar.

Prerequisites

Oculus Rift and Touch are supported in Humanoid Control VR, Plus and Pro.

Unity

Oculus on PC is supported for Unity 2017.4 LTS up to Unity 2019.4

Unity version 2020.1 and higher are supported using Unity XR.

Hardware

Oculus Rift DK2, Rift (S)+Touch and Quest (via Oculus Link) are supported.

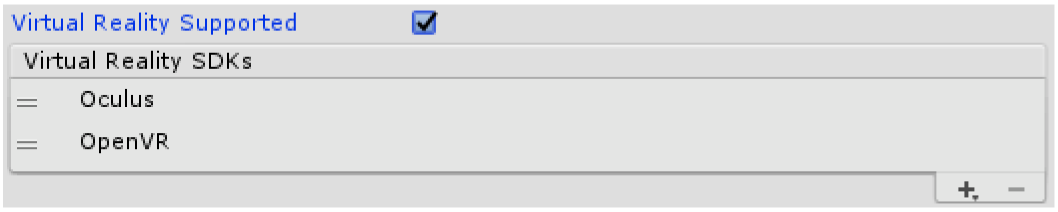
Operating System

Oculus Rift & Touch are supported on Microsoft Windows. Ensure *PC, Mac & Linux Standalone* is selected as platform in Build settings

Setup

Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Note: if both Oculus and OpenVR need to be supported in one build, make sure *Oculus* is listed higher than *OpenVR* in the *Virtual Reality SDKs:*



Virtual Reality Supported ✓

Virtual Reality SDKs

Oculus

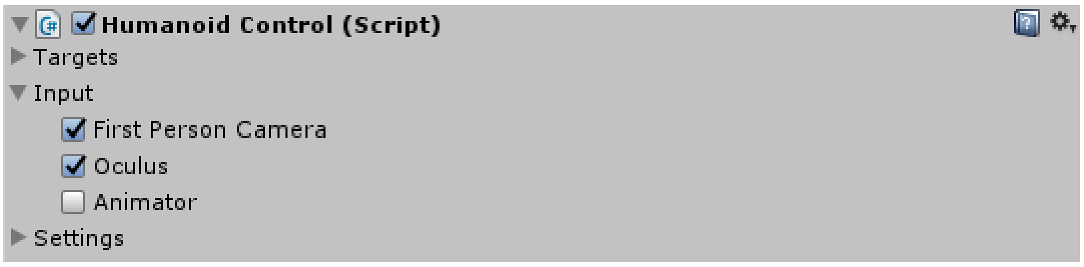
OpenVR

Oculus needs to be enabled in the *Edit Menu->Preferences->Humanoid->Oculus Support.*

Disabling *Oculus Support* ensures that no code related to Oculus is included in the build.

Targets

To enable body tracking with Oculus for an avatar, *Oculus* needs to be enabled in the Humanoid Control component.



✓ Humanoid Control (Script)

Targets

Input

✓ First Person Camera

✓ Oculus

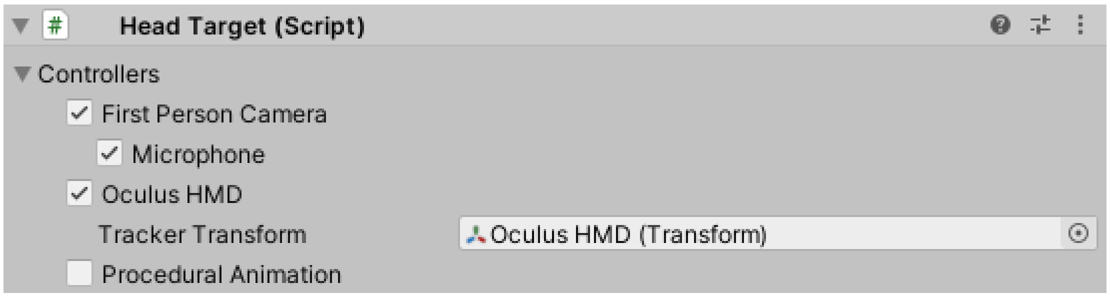
☐ Animator

Settings

Head Target

To enable head tracking with Oculus for an avatar, *Oculus HMD* needs to be enabled in the Humanoid Control component.
The *Tracker Transform* is a reference to the HMD itself in the Real World pose.

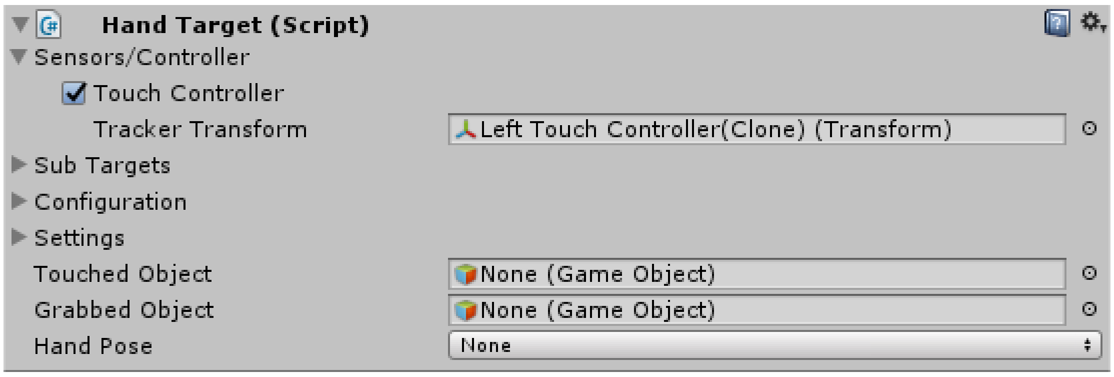
You can disable the First Person Camera to get a head pose for the avatar without the camera being on the head of the camera. This can be useful if you want to create a third-person view of the user for example.


Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone
✓ Oculus HMD
Tracker Transform: Oculus HMD (Transform)
Procedural Animation

Hand Target

The *Touch Controller* needs to be enabled on the Hand Targets for controller support.

Oculus Touch Controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.


Hand Target (Script)
Sensors/Controller
✓ Touch Controller
Tracker Transform: Left Touch Controller(Clone) (Transform)
Sub Targets
Configuration
Settings
Touched Object: None (Game Object)
Grabbed Object: None (Game Object)
Hand Pose: None

Controller input

The buttons of the Oculus Touch controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

X button	controller.left.button[0]
Y button	controller.left.button[1]
Thumbstick touch	controller.left.stickTouch
Thumbstick press	controller.left.stickButton
Thumbstick movements	controller.left.stickHorizontal, .stickVertical
Index Trigger	controller.left.trigger1

Hand Trigger	controller.left.trigger2

Right Controller

A button	controller.right.button[0]
B button	controller.right.button[1]
Thumbstick touch	controller.right.stickTouch
Thumbstick press	controller.right.stickButton
Thumbstick movements	controller.right.stickHorizontal, .stickVertical
Index Trigger	controller.right.trigger1
Hand Trigger	controller.right.trigger2

Calibration

The tracking position and orientation can be calibrated during gameplay by calling the Calibrate() function on the HumanoidControl object. This is often implemented using the Controller Input component.

Oculus Quest

Oculus Quest supported to track head and hands of an avatar.

In the Editor an Oculus Rift can be used for testing the scene even when the platform is set to *Android*. For final testing on the Oculus Quest, a build needs to be made and transferred to the Quest.

The easiest solution for building for Quest is to connect the Quest to the PC with an USB cable and then select *Build And Run* in the *File* menu of Unity. Please note that the Oculus Quest needs to have Developer Mode enabled for this (see https://developer.oculus.com/documentation/quest/latest/concepts/mobile-device-setup-quest/)

Prerequisites

Oculus Quest is supported in Humanoid Control VR, Plus and Pro version 2.2 and higher.

Hand tracking with the Oculus Quest is supported in Humanoid Control Plus and Pro version 3.1 and higher.

Unity

Oculus Quest is supported for Unity 2017.4 LTS up to Unity 2019.4

Unity version 2020.1 and higher are supported using Unity XR.

Operating System

Oculus Quest is supported on the Android platform. Ensure *Android* is selected as platform in Build settings

Setup

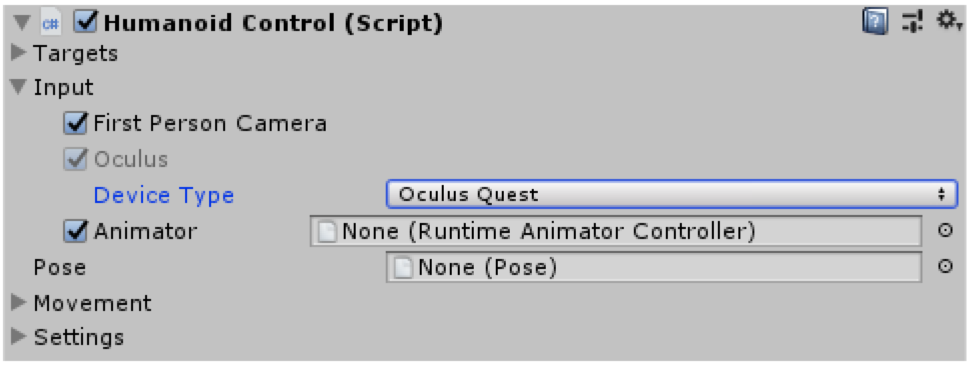
Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Oculus needs to be enabled in the *Edit Menu->Preferences->Humanoid->Oculus Support*.

Disabling *Oculus Support* ensures that no code related to Oculus is included in the build.

Targets

To enable body tracking with the Oculus Quest for an avatar, *Oculus* needs to be enabled in the Humanoid Control component.

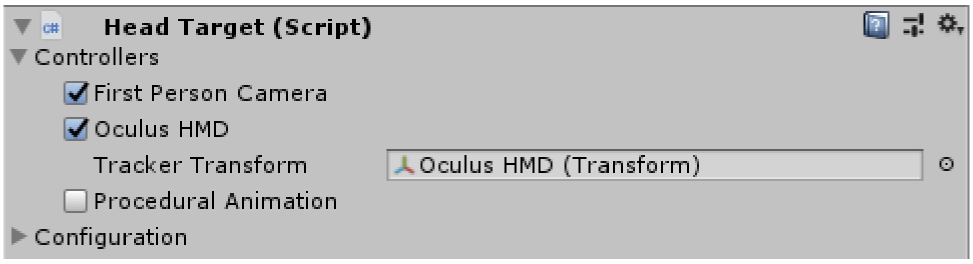


Humanoid Control (Script) [✓]
- Targets
- Input
  - [✓] First Person Camera
  - [✓] Oculus
    - Device Type: Oculus Quest
  - [✓] Animator: None (Runtime Animator Controller)
- Pose: None (Pose)
- Movement
- Settings

The *Device Type* needs to be set to *Oculus Quest*. This will ensure that the controllers in both hands will have positional tracking.

Head Target

First Person Camera needs to be supported for the Oculus Rift. For convenience, this option is also found on the Humanoid Control script.

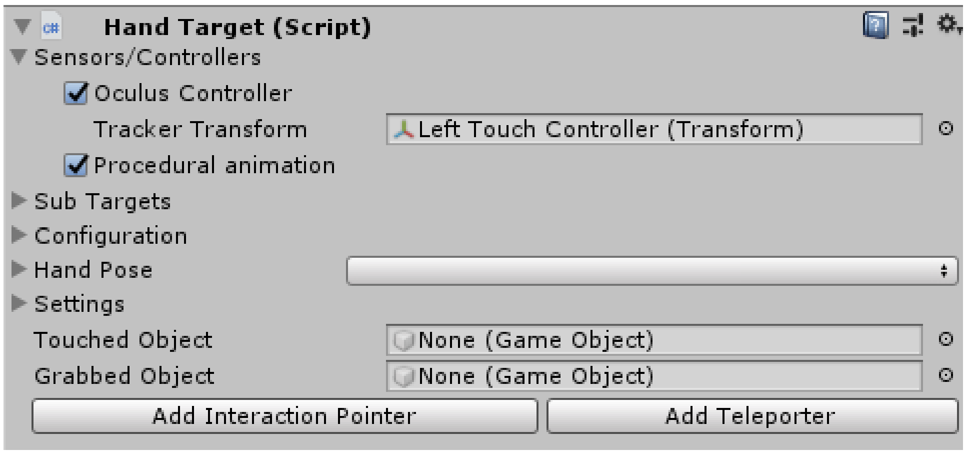


Head Target (Script)
- Controllers
  - [✓] First Person Camera
  - [✓] Oculus HMD
    - Tracker Transform: Oculus HMD (Transform)
  - [ ] Procedural Animation
- Configuration

Hand Target

The *Oculus Controller* needs to be enabled on the Hand Targets for controller support.

Oculus Quest Controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.



Hand Target (Script)
- Sensors/Controllers
  - [✓] Oculus Controller
    - Tracker Transform: Left Touch Controller (Transform)
  - [✓] Procedural animation
- Sub Targets
- Configuration
- Hand Pose
- Settings
- Touched Object: None (Game Object)
- Grabbed Object: None (Game Object)

Controller Input

The buttons of the Oculus Quest controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

X button	controller.left.button[0]
Y button	controller.left.button[1]
Thumbstick touch	controller.left.stickTouch
Thumbstick press	controller.left.stickButton
Thumbstick movements	controller.left.stickHorizontal, .stickVertical
Index Trigger	controller.left.trigger1
Hand Trigger	controller.left.trigger2

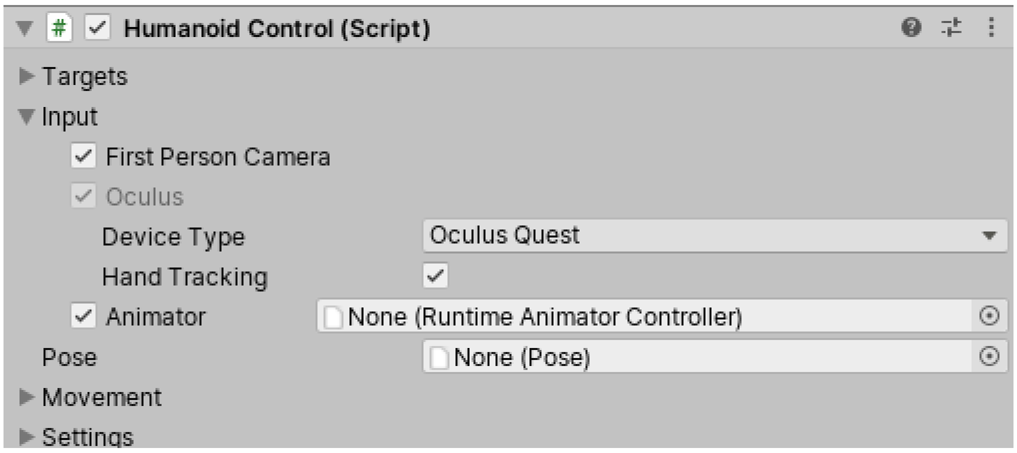
Right Controller

A button	controller.right.button[0]
B button	controller.right.button[1]
Thumbstick touch	controller.right.stickTouch
Thumbstick press	controller.right.stickButton
Thumbstick movements	controller.right.stickHorizontal, .stickVertical
Index Trigger	controller.right.trigger1
Hand Trigger	controller.right.trigger2

Hand Tracking (Plus & Pro)

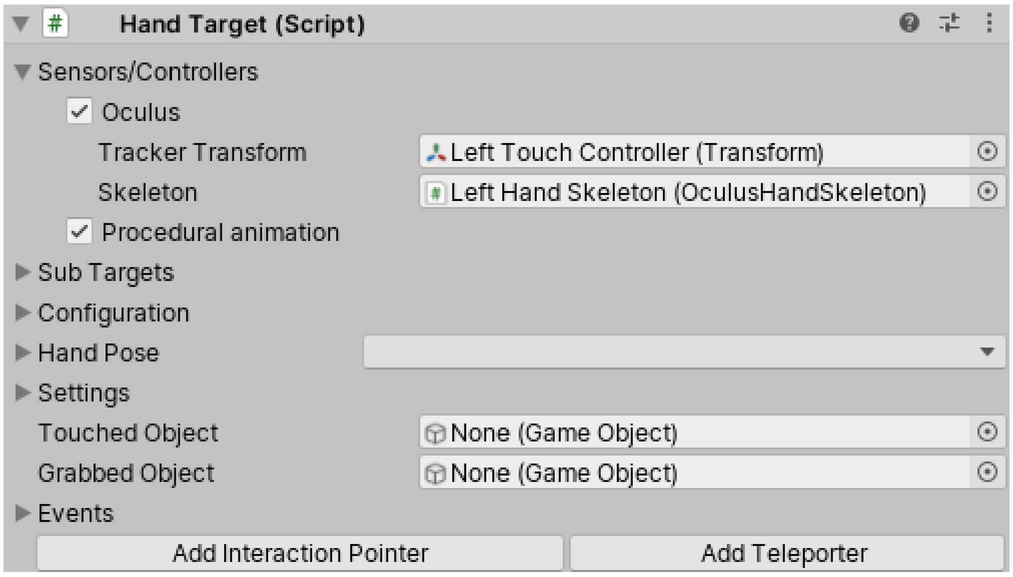
For Hand Tracking in Humanoid Control Plus and Pro, make sure the installed OVR Utilities Plugin is at least version 1.44.0. This can be ensured by installing the Oculus Integration package in the project and upgrading to the latest plugin version. It is expected that future versions of Unity have version 1.44.0 or higher already built in, so that importing the Oculus Integration Package is no longer necessary.

You can enable hand tracking in the Humanoid Control component:


Humanoid Control (Script)

- Targets
- Input
  - ✓ First Person Camera
  - ✓ Oculus
    - Device Type: Oculus Quest
    - Hand Tracking: ✓
  - ✓ Animator: None (Runtime Animator Controller)
- Pose: None (Pose)
- Movement
- Settings

On the Hand Targets you will find a reference to the Hand Skeleton:


Hand Target (Script)

- Sensors/Controllers
  - ✓ Oculus
    - Tracker Transform: Left Touch Controller (Transform)
    - Skeleton: Left Hand Skeleton (OculusHandSkeleton)
  - ✓ Procedural animation
- Sub Targets
- Configuration
- Hand Pose
- Settings
- Touched Object: None (Game Object)
- Grabbed Object: None (Game Object)
- Events

The skeleton is a representation of the user's hands. They will be rendered when 'Show Real Objects' is enabled.

Please make sure that for hand tracking, 'Use Hands' is enabled in the Settings of the Oculus Quest itself or hand tracking will not work.

Calibration

The tracking position and orientation can be calibrated during gameplay by calling the Calibrate() function on the Humanoid Control object. This is often implemented using the Controller Input component.

Samsung Gear VR, Oculus Go

Prerequisites

Gear VR / Oculus Go are supported in Humanoid Control VR, Plus and Pro packages.

Unity

Gear VR / Oculus Go are supported in Unity versions 2017.4 LTS up to Unity 2019.4.

Hardware

Samsung Gear VR Innovator & Consumer versions and Oculus Go are supported.

Operating System

The extension supports Samsung Gear VR/Oculus Go on Android. Ensure *Android* is selected as platform in Build Settings.

Setup

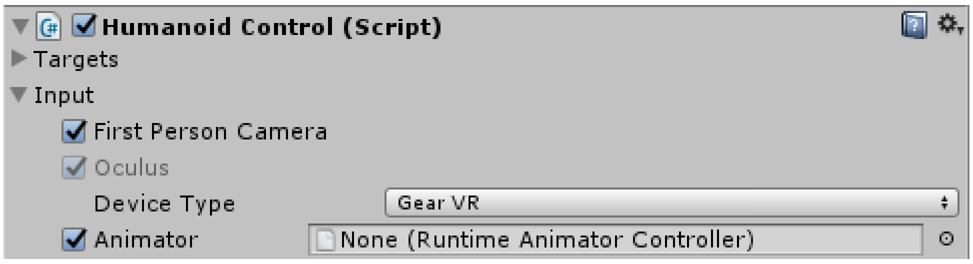
Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Oculus needs to be enabled in the *Edit Menu->Preferences->Humanoid->Oculus Support.*

Disabling *Oculus Support* ensures that no code related to Gear VR is included in the build.

Targets

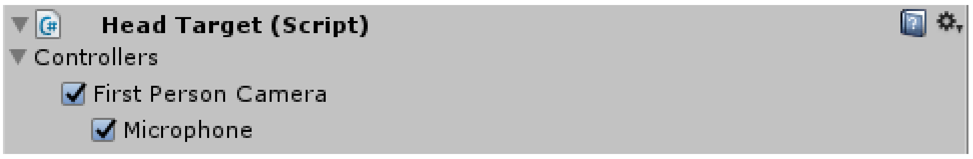
To enable tracking with Gear VR/Oculus Go for an avatar, *Oculus* needs to be enabled in the Humanoid Control component. The Device Type can be used to select the specific Gear VR or Oculus Go device.



Humanoid Control (Script)
Targets
Input
☑ First Person Camera
☑ Oculus
Device Type: Gear VR
☑ Animator: None (Runtime Animator Controller)

Head Target

First Person Camera needs to be enabled for the Gear VR/Oculus Go. For convenience, this option is also found on the Humanoid Control script.



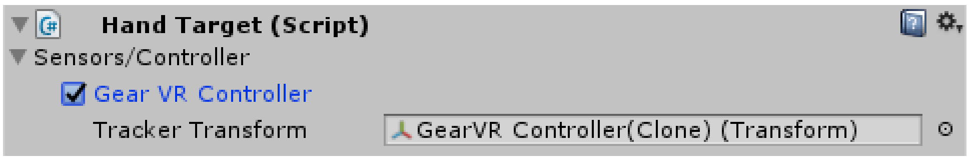
Head Target (Script)
Controllers
☑ First Person Camera
☑ Microphone

Hand Target

The *Oculus Controller* needs to be enabled on the Hand Targets for controller support.

Note: Gear VR/Oculus Go only supports one controller per headset.

The controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.



Hand Target (Script)
Sensors/Controller
☑ Gear VR Controller
Tracker Transform: GearVR Controller(Clone) (Transform)

Controller input

The buttons of the Gear VR controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

Dpad press	controller.left.stickButton
Dpad movements	controller.left.stickHorizontal, .stickVertical
Back	controller.left.option

Right Controller

Dpad press	controller.right.stickButton
Dpad movements	controller.right.stickHorizontal, .stickVertical
Back	controller.right.option

Google VR / Cardboard

Prerequisites

Google VR is supported in Humanoid Control VR, Plus and Pro packages.

Unity

Unity versions 2017.4 up to 2019.4 are supported.

Hardware

Google Cardboard v1 & v2 are supported in combination with suitable Android phones.

Operating System

Google VR is supported for Android. iOS may work, but hasn't been tested.

Setup

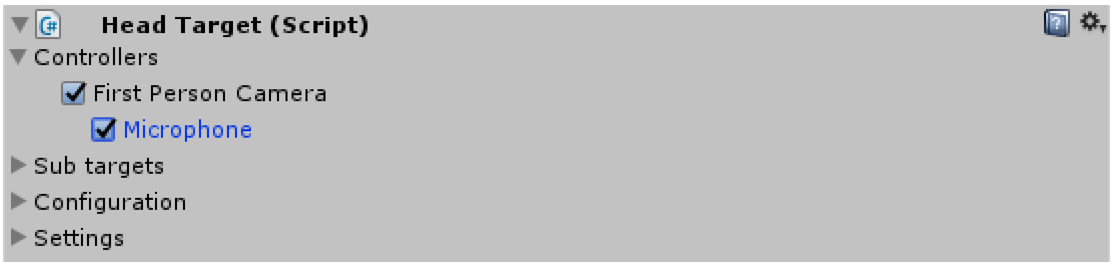
Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Google VR* needs to be added to the *Virtual Reality SDKs*.

No further preparations are necessary for Google VR support.

Targets

Head Target

First Person Camera needs to be supported for Google VR. For convenience, this option is also found on the Humanoid Control script.



Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone
Sub targets
Configuration
Settings

Windows Mixed Reality

The Windows Mixed Reality platform supports various Headsets and Motion Controllers based on the Microsoft Mixed Reality platform.

Prerequisites

Windows Mixed Reality is supported in Humanoid Control VR, Plus and Pro packages.

Unity

Unity versions 2017.4 up to 2019.4 are supported

Unity version 2020.1 and higher are supported using Unity XR.

Hardware

All Mixed Reality Headsets and Motion Controllers are supported

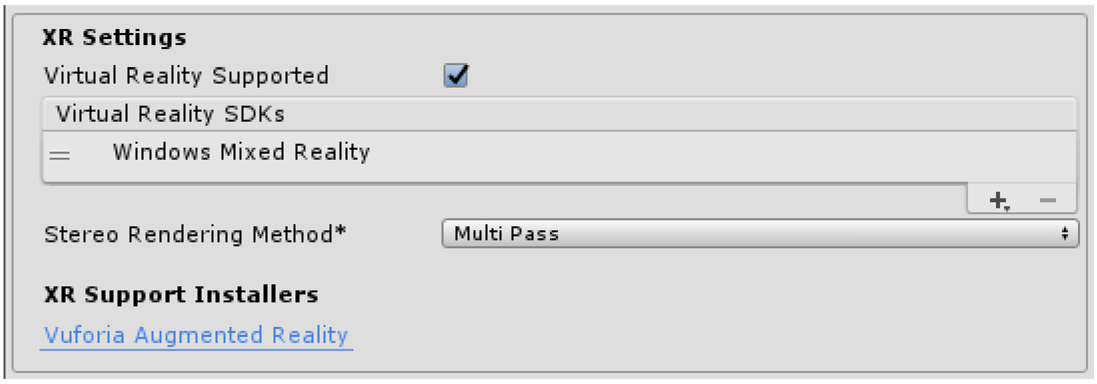
Operating System

Windows Mixed Reality is only supported on Microsoft Windows 10 Fall Creators Editor or newer.

Setup

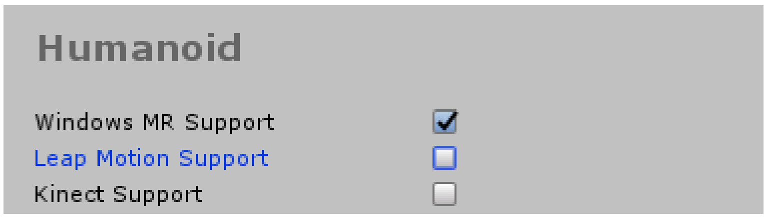
Ensure *Universal Windows Platform* is selected as platform in the Unity Build settings found in the File Menu.

Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Windows Mixed Reality* needs to be added to the *Virtual Reality SDKs*.



XR Settings
Virtual Reality Supported ✓
Virtual Reality SDKs
Windows Mixed Reality
Stereo Rendering Method* Multi Pass
XR Support Installers
Vuforia Augmented Reality

For full Windows Mixed Reality support you needs to ensure *Windows MR Support* is enabled in *Edit Menu->Preferences->Humanoid*.

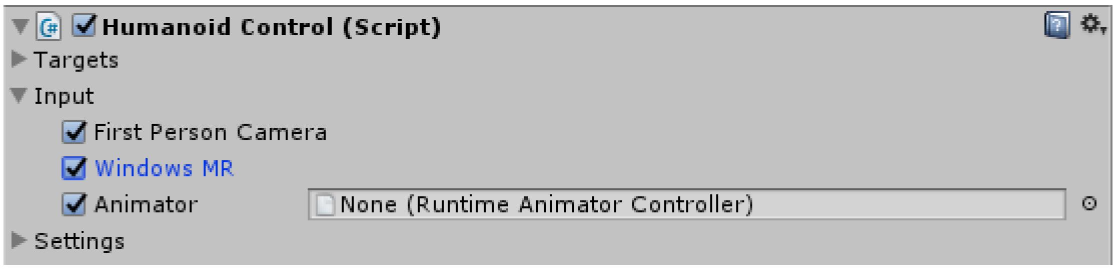


Humanoid
Windows MR Support ✓
Leap Motion Support ☐
Kinect Support ☐

Disabling *Windows MR Support* ensures that no code related to Windows Mixed Reality is included in the build.

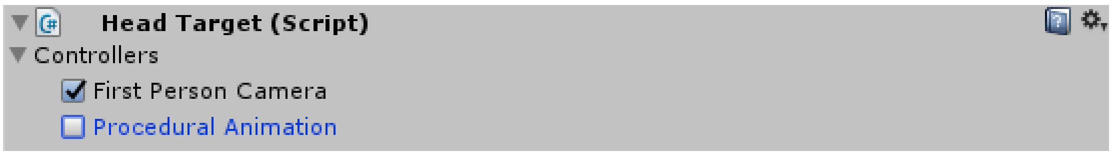
Targets

To enable Windows Mixed Reality tracking for an avatar, *First Person Camera* and *Windows MR* need to be enabled in the Humanoid Control component.


Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Windows MR
✓ Animator None (Runtime Animator Controller)
Settings

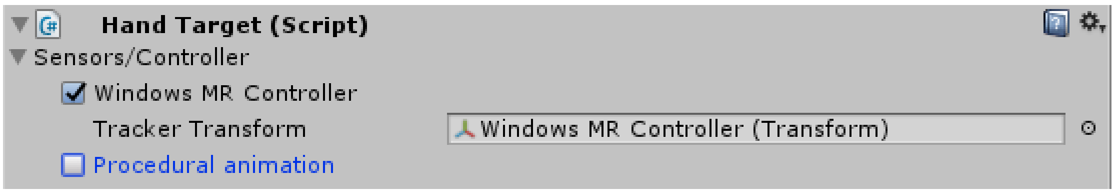
Head Target

Full positional and rotational tracking is supported.


Head Target (Script)
Controllers
✓ First Person Camera
☐ Procedural Animation

Hand Targets

Full positional and rotational tracking is supported. Full positional tracking is only working when the controllers are visible by the cameras in the headset.


Hand Target (Script)
Sensors/Controller
✓ Windows MR Controller
Tracker Transform Windows MR Controller (Transform)
☐ Procedural animation

Controller input

The buttons of the Motion Controllers can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

Touchpad	controller.left.stickHorizontal/stickVertical
Touchpad touch	controller.left.stickTouch
Stick movements	controller.left.stickHorizontal/stickVertical
Stick press	controller.left.stickButton
Trigger	controller.left.trigger1
Grip	controller.left.trigger2
Menu button	controller.left.option

Right Controller

Touchpad	controller.right.stickHorizontal/stickVertical
Touchpad touch	controller.right.stickTouch
Stick movements	controller.right.stickHorizontal/stickVertical
Stick press	controller.right.stickButton
Trigger	controller.right.trigger1
Grip	controller.right.trigger2
Menu button	controller.right.option

Calibration

The tracking position and orientation can be calibrated during gameplay by calling the Calibrate() function on the HumanoidControl object. This is often implemented using the Controller Input component.

Unity XR

The Unity XR package manager is a new approach of Unity to supporting XR devices. This implementation forms the new basis for VR devices like Oculus, WIndows Mixed Reality and OpenVR.

Note: this is still in development in both Unity and Humanoid Control. Therefore features can be missing (like Oculus hand tracking and OpenVR skeletal input) and contains bugs.

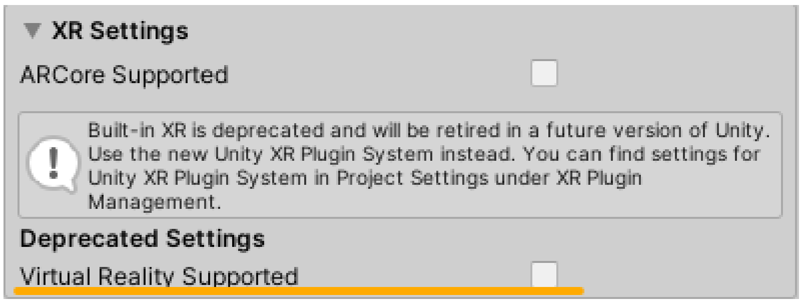
Prerequisites

Unity

Preliminary Unity XR support is available in version 3.2 for Unity 2020.1, Unity 2019.3 and higher are supported from version 3.3.

Setup

For Unity XR support in Unity 2019, first make sure Legacy XR suport is disabled. Go to *Edit Menu->Project Settings->Player->XR Settings* and ensure *Virtual Reality Supported* is disabled:



XR Settings

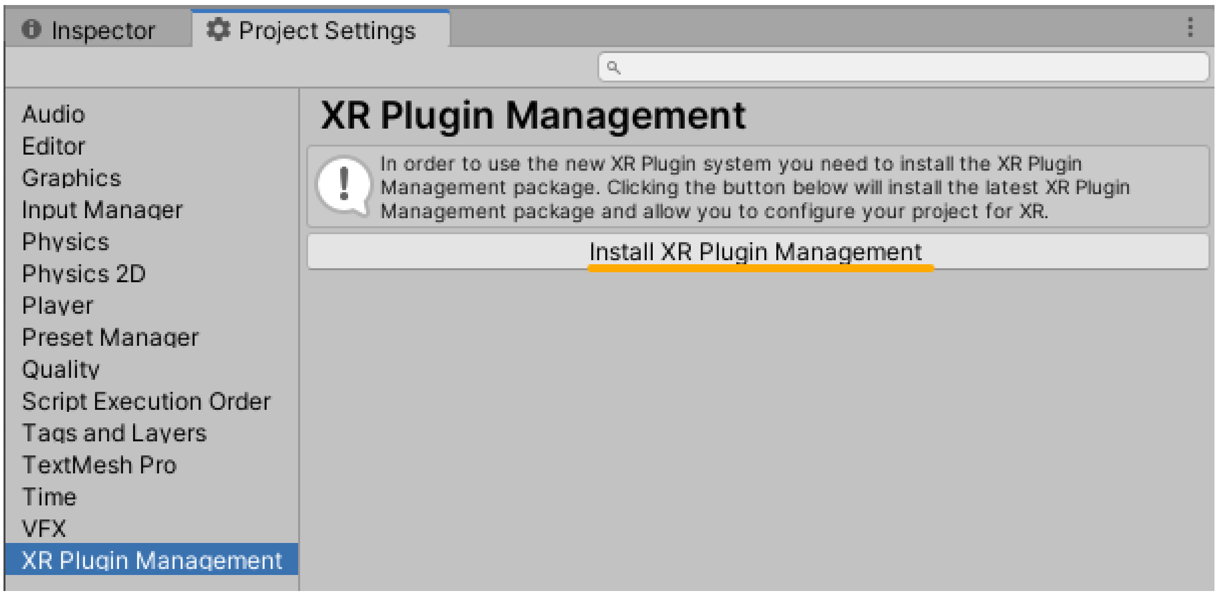
ARCore Supported

Built-in XR is deprecated and will be retired in a future version of Unity. Use the new Unity XR Plugin System instead. You can find settings for Unity XR Plugin System in Project Settings under XR Plugin Management.

Deprecated Settings

Virtual Reality Supported

Go to *Edit Menu->Project Settings->XR Plugin Management* and click on the *Install XR Plugin Management* button:

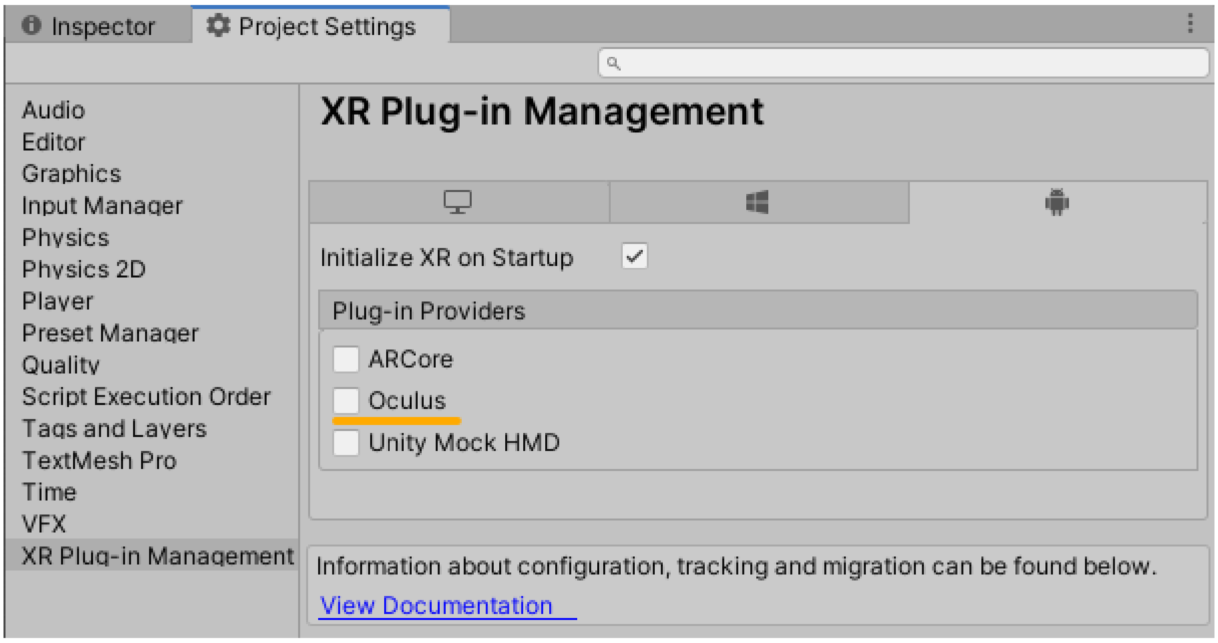


XR Plugin Management

In order to use the new XR Plugin system you need to install the XR Plugin Management package. Clicking the button below will install the latest XR Plugin Management package and allow you to configure your project for XR.

Install XR Plugin Management

Then enable the desired XR Plugin. For example the Oculus plugin for Android:



XR Plug-in Management

Initialize XR on Startup ✓

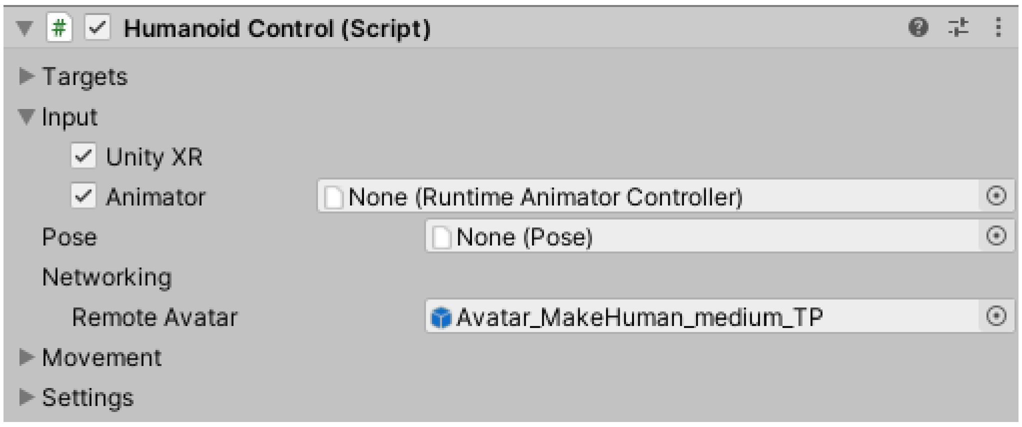
Plug-in Providers

- ☐ ARCore
- ☐ Oculus
- ☐ Unity Mock HMD

Information about configuration, tracking and migration can be found below.
View Documentation

Configuration

To enable body tracking with Unity XR for an avatar, *Unity XR* needs to be enabled in the *Humanoid Control* component:

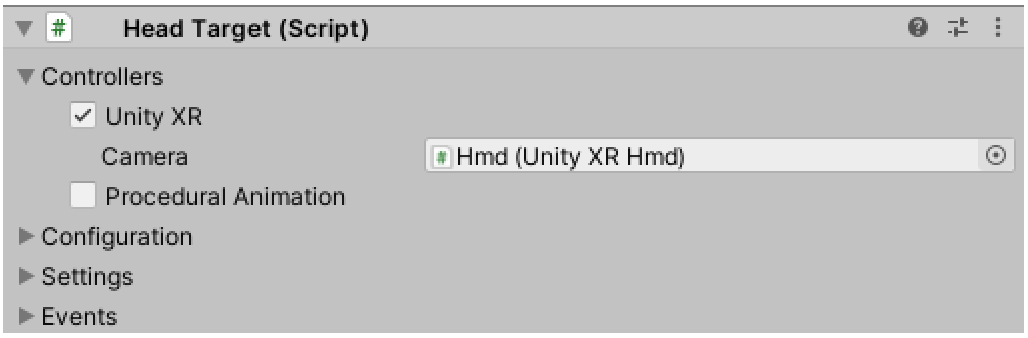


Humanoid Control (Script)

- Targets
- Input
  - ✓ Unity XR
  - ✓ Animator: None (Runtime Animator Controller)
  - Pose: None (Pose)
  - Networking
    - Remote Avatar: Avatar_MakeHuman_medium_TP
- Movement
- Settings

Head Target

To use an HMD tracking for the head of the avatar *Unity XR* needs to be enabled on the *Head Target* too. This is enabled by default:



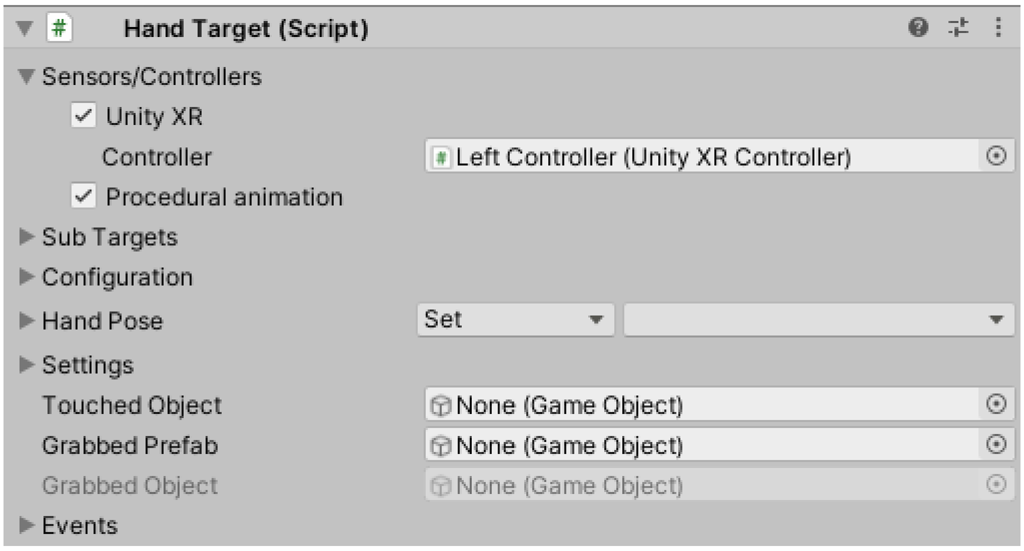
Head Target (Script)

- Controllers
  - ✓ Unity XR
    - Camera: Hmd (Unity XR Hmd)
  - ☐ Procedural Animation
- Configuration
- Settings
- Events

The *Camera* parameter is the camera which is used for the first-person view. It can be cleared when no camera view is desired and only the tracking of the head is needed. For example, if you want to see the avatar’s movements in third person view.

Hand Target

When you want to control the hands of the avatar using Unity XR you need to enable *Unity XR* on the *Hand Target*:



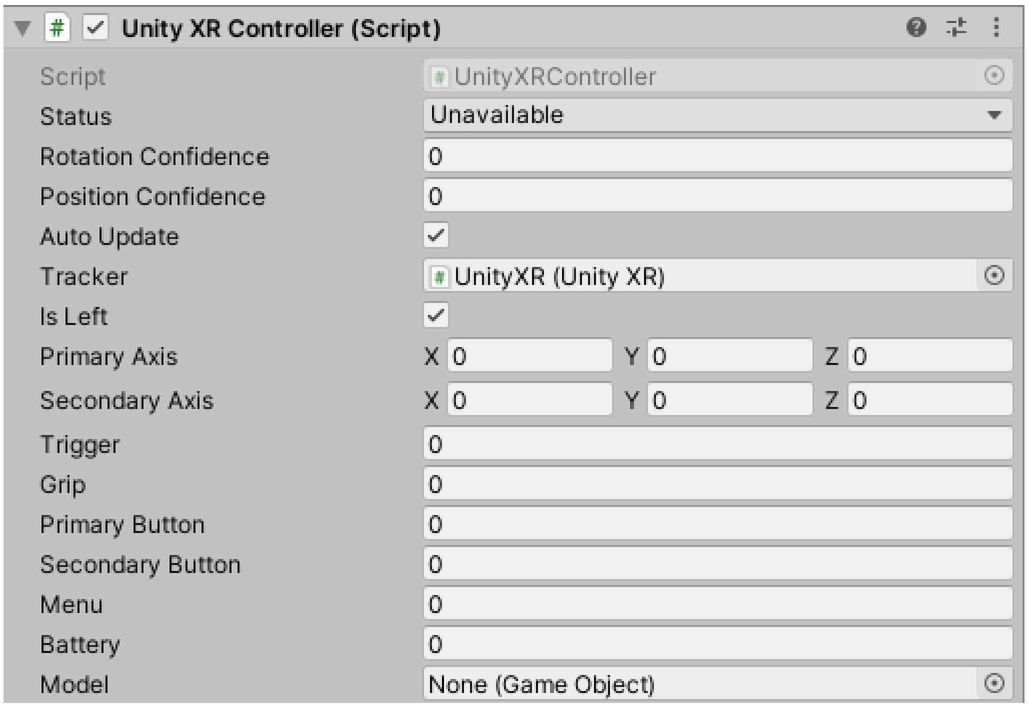
Hand Target (Script)

Sensors/Controllers	
✓ Unity XR	
Controller	Left Controller (Unity XR Controller)
✓ Procedural animation	
Sub Targets	
Configuration	
Hand Pose	Set
Settings	
Touched Object	None (Game Object)
Grabbed Prefab	None (Game Object)
Grabbed Object	None (Game Object)
Events	

The *Controller* parameter is a reference to the *Unity XR Controller* in the Real World representing the actual controller used.

Unity XR Controller

This component is a representation of the Unity XR controller in the real world. It provides all the input information of the controller.



✓ Unity XR Controller (Script)

Script	UnityXRController
Status	Unavailable
Rotation Confidence	0
Position Confidence	0
Auto Update	✓
Tracker	UnityXR (Unity XR)
Is Left	✓
Primary Axis	X 0 Y 0 Z 0
Secondary Axis	X 0 Y 0 Z 0
Trigger	0
Grip	0
Primary Button	0
Secondary Button	0
Menu	0
Battery	0
Model	None (Game Object)

The *Transform* of the *Unity XR Controller* will always contain the position and rotation of the controller when it is being tracked.

Status	The tracking status of the controller. *Unavailable*: Controller is not connected *Present*: Controller is available but is not currently being tracked

	Tracking: Controller is being tracked
Rotation Confidence	A quality indication of the rotation value. *0*: lowest confidence *1*: highest confidence
Position Confidence	A quality indication of the position value. *0*: lowest confidence *1*: highest confidence
Auto Update	*True*, the controller value are automatically updated every frame *False*, you need to call the UpdateComponent() function to update the values.
Tracker	The *Unity XR* tracker
isLeft	*True*: this is the left-hand controller. *False*: this is the right-hand controller.
Primary Axis	The value of the primary joystick or touchpad. The X and Y values represent the movements of the joystick or the position on the touchpad. The Z value represents the touching and/or pressing of the joystick.
Secondary Axis	The value of the secondary joystick or touchpad (if present)
Trigger	The value of the index finger trigger.
Grip	The value of the middle finger / hand trigger.
Primary Button	The value of the primary button
Secondary Button	The value of the secondary button
Menu	The value of the menu button
Battery	The value of the battery status of the controller
Model	A reference to the GameObject representing the model of the controller.

The buttons and Z-axis for the joystick/touchpad can have the following values:
< 0: no touching, *> 0* touching
-1..0: closeness to the joystick or touchpad
0..1: press value
The actual values depend on the hardware: whether a touch sensor or proximity sensor is available and/or whether the button is an on/off button or can measure the amount of pressure.

VRTK

VRTK is a popular virtual reality toolkit which supports various VR devices. It includes functionality for many useful actions and interactions.

With this extension it is possible to attach a humanoid avatar to the tracking of VRTK. Next to that networking is supported such that multiple players can share the same scene.

Prerequisites

Humanoid Control

VRTK is supported in all Humanoid Control packages in version 2.1 and higher. It can be used with any platform or hardware supported by VRTK itself.

SDK

VRTK version 3.2 and 3.3 are supported.

Setup

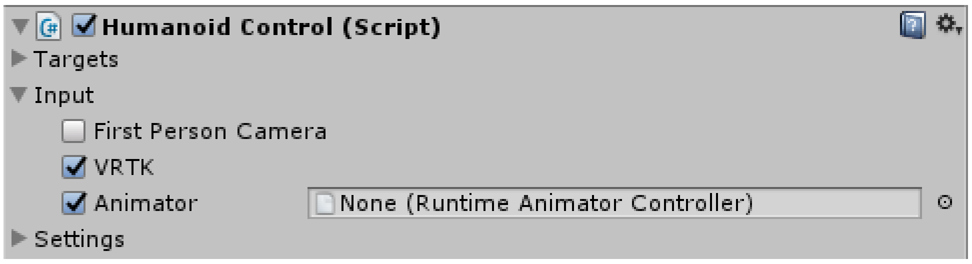
It is assumed that a working VRTK scene has been setup first. Information on how can be done is found on the VRTK support pages.

For Humanoid Control support, VRTK support needs to be enabled in the *Edit Menu->Preferences->Humanoid->VRTK Support*.

Disabling VRTK support ensures that no code related to VRTK is included in the build.

Configuration

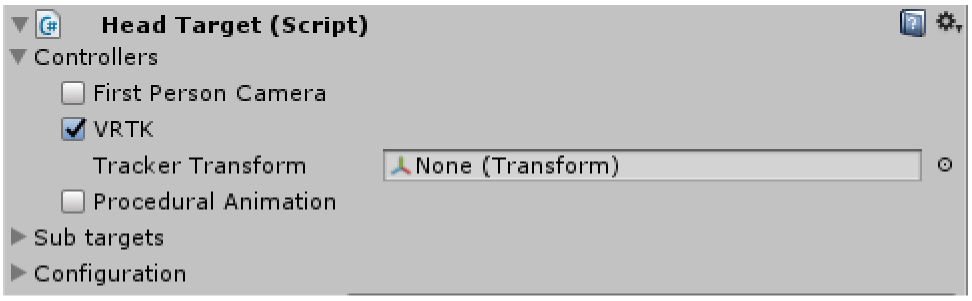
To enable VRTK based tracking for a humanoid, *VRTK* needs to be enabled in the Humanoid Control component:


Humanoid Control (Script)
▸ Targets
▾ Input
☐ First Person Camera
☑ VRTK
☑ Animator None (Runtime Animator Controller)
▸ Settings

Head Target

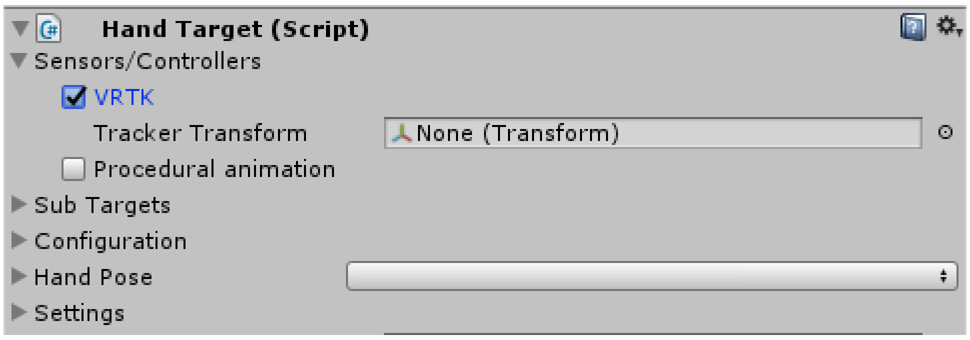
For the Head Target, *First Person Camera* should be disabled because VRTK will manage the camera and the headset transform.

VRTK needs to be enabled on the Head Target. When running the scene, the *Tracker Transform* will point to the Camera which is selected based on the VRTK SDK settings.


Head Target (Script)
▾ Controllers
☐ First Person Camera
☑ VRTK
Tracker Transform None (Transform)
☐ Procedural Animation
▸ Sub targets
▸ Configuration

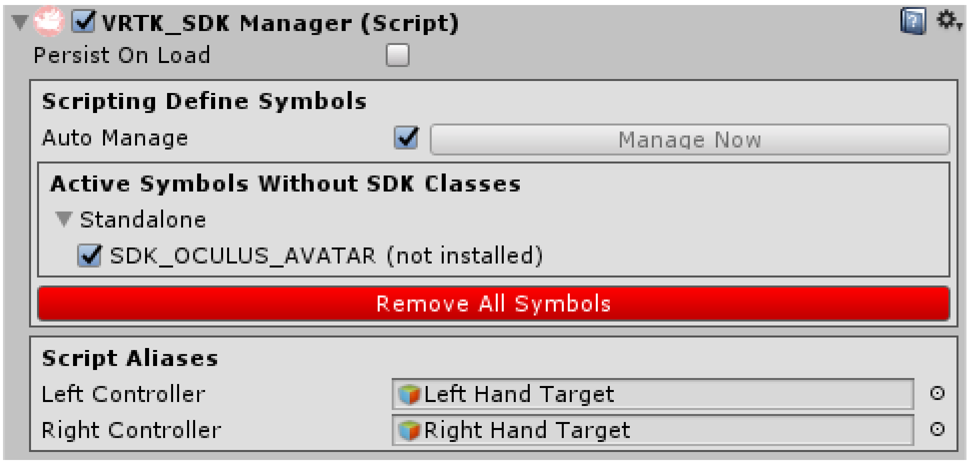
Hand Target

For hand tracking, VRTK needs to be enabled on the Hand Targets. The Tracker Transform will point to the VRTK Tracked Controller selected by the VRTK SDK at runtime.



Hand Target (Script)
Sensors/Controllers
☑ VRTK
Tracker Transform: None (Transform)
☐ Procedural animation
Sub Targets
Configuration
Hand Pose
Settings

Humanoid Control will add a VRTK_ControllerEvents component to the Hand Target for supporting controller input. It will also automatically set the Left and Right Controller Script Aliases to the Left and Right Hand Targets in order to support controller input:



☑ VRTK_SDK Manager (Script)
Persist On Load ☐
Scripting Define Symbols
Auto Manage ☑ Manage Now
Active Symbols Without SDK Classes
Standalone
☑ SDK_OCULUS_AVATAR (not installed)
Remove All Symbols
Script Aliases
Left Controller: Left Hand Target
Right Controller: Right Hand Target

Controller input

The VRTK buttons are mapped to the Humanoid Controller Input as follows:

Left Controller

Touchpad Touch	controller.left.stickTouch
Touchpad Press	controller.left.stickButton
Trigger	controller.left.trigger1
Grip	controller.left.trigger2
Button One	controller.left.button[0]
Button Two	controller.left.button[1]
Start Menu Button	controller.left.option

Right Controller

Touchpad Touch	controller.right.stickTouch
Touchpad Press	controller.right.stickButton

Trigger	controller.right.trigger1
Grip	controller.right.trigger2
Button One	controller.right.button[0]
Button Two	controller.right.button[1]
Start Menu Button	controller.right.option

Game Controller

A growing number of game controllers is supported natively. The following controllers are currently supported:

- Microsoft Xbox controller (360, One)
- Playstation4 controller
- Steelseries XL controller
- GameSmart controllers (e.g. MadCatz C.T.R.L.R.)
- Sweex GA100 (Yes, it is obscure but I happen to have one :-)

Additionally, the following hand trackers are also supported like game controllers:

- Razer Hydra
- SteamVR Controllers (HTC Vive)
- Oculus Touch
- Mixed Reality Motion Controllers
- Gear VR Controller
- Oculus Go Controller

To support Game controllers, you need to update the InputManager Settings. The package does contain an archive called 'GameControllerInputSettings.zip' which contains universal InputManager settings for a the game controllers. Move this to the ProjectSettings folder to get maximum support for your controller.

Controller Input Sides

Controllers are split in a left and right side which support the same buttons. On each side the following buttons are supported:

- Thumbstick horizontal and vertical (float values)
- Thumbstick button press/touch (boolean)
- Directional Pad (Up, down, left & right) (boolean)
- Buttons 0..3 (boolean)
- Bumper (float value)
- Trigger (float value)
- Option (boolean)

For each game controller most buttons can be mapped to these buttons.

Multiple Controllers

Currently, the game controller input is limited to one game controller. A pair of Hydra, SteamVR or Touch controllers is considered as one controller. All available game controllers will be mapped to same input.

Scripting

The game controller input can be retrieved using:

```
Controllers.GetController(0)
```

PlayMaker

For PlayMaker, two action scripts are provided:

Get Controller Axis

Gets the values of the horizontal and vertical thumbstick axis

Controller Side	left or right side of the controller
Store Vector	the Vector3 which should store the values of the thumbstick input

Get Controller Button

Get the status of one of the game controller buttons

Controller Side	left or right side of the controller.
Button	the button which we want to read.
Store Bool	stores the value of the button as a boolean. All buttons statuses are converted to booleans.
Store Float	stores the value of the button as a float. All button statuses are converted to float values.
Button Pressed	event to send when the button is pressed.
Button Released	event to send when the button is released.

Leap Motion (Plus)

Leap Motion enabled detailed hand tracking with markerless optical detection. Individual finger movements can be tracked.

Prerequisites

Leap Motion is supported in Humanoid Control Plus and Pro.

Hardware

HMD mounted Leap Motion is supported when it is mounted on Oculus Rift, HTC Vive or Windows Mixed Reality using the Leap Motion VR Developer Mount.

Operating System

Leap Motion is supported on Microsoft Windows 10.

Runtime

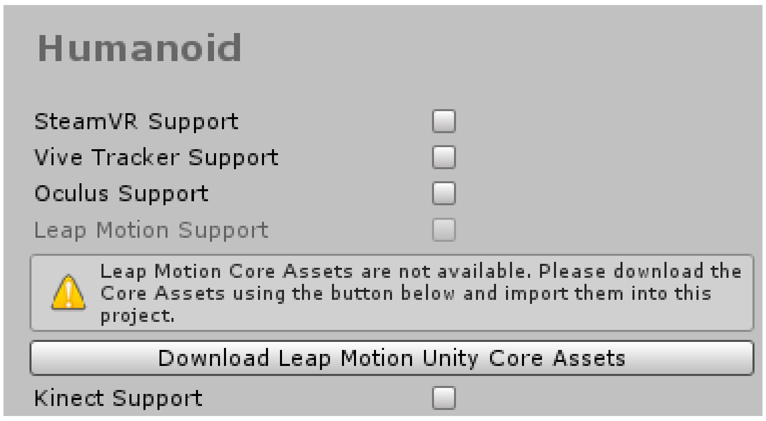
Leap Motion software version 3 or higher is required.

Unity

Unity version 5.5 or higher is required.

Setup

For Leap Motion to work, the Leap Motion Unity Code Assets needs to be imported into the project.
Go to *Edit Menu->Preferences->Humanoid* and look for the *Leap Motion Support* entry.



Humanoid

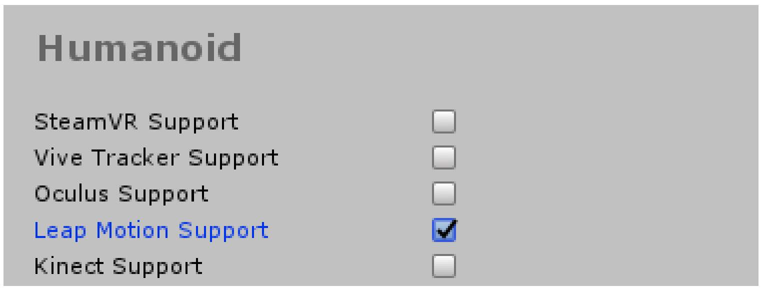
SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Leap Motion Support	☐

Leap Motion Core Assets are not available. Please download the Core Assets using the button below and import them into this project.

Kinect Support	☐

Click the button to go to the download page for the Leap Motion Assets.

After the assets have been imported, *Leap Motion Support* can be enabled in the preferences.

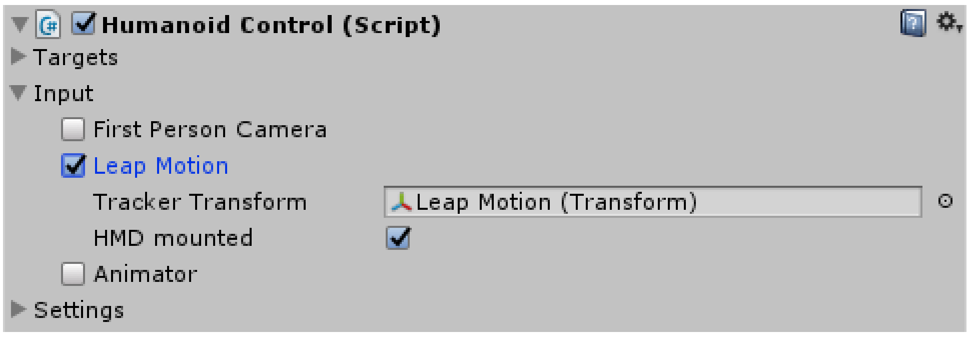

Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Leap Motion Support	✓
Kinect Support	☐

Disabling Leap Motion support ensures that no code related to Leap Motion is included in the build.

Targets

To enable tracking with the Leap Motion for an avatar, *Leap Motion* needs to be enabled in the *Humanoid Control* component.


✓ Humanoid Control (Script)

Targets

Input

☐ First Person Camera

✓ Leap Motion

Tracker Transform	Leap Motion (Transform)
HMD mounted	✓

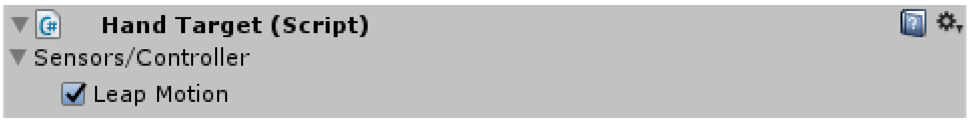
☐ Animator

Settings

The *Leap Motion (Transform)* is a reference to the Transform in the scene representing the Leap Motion Sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Leap Motion (Transform)* can be used to set the position of the tracking relative to the player in the scene.

Hand Target


Hand Target (Script)

Sensors/Controller

✓ Leap Motion

Hands are fully tracked (positional and rotational) while the hands are in the tracking range of the Leap Motion.

Full hand movements are supported with bending values for each finger individually while the hands are in the tracking range of the Leap Motion.

Microsoft Kinect 360 / Kinect for Windows 1 (Plus)

Microsoft Kinect 360 & Microsoft Kinect for Windows enable full body tracking.

Prerequisites

Humanoid Control

Microsoft Kinect 360/Kinect for Windows 1 is supported in Humanoid Control Plus and Pro version 2.1 and higher.
The issues with the driver in Unity 5 have been solved by writing a new library for Kinect 1 support.

Hardware

Microsoft Kinect 360 and Kinect for Windows are supported. An adapter is required for later revisions of the Microsoft Kinect 360.

Operating System

Microsoft Kinect is supported on Microsoft Windows 7 and above.

Runtime

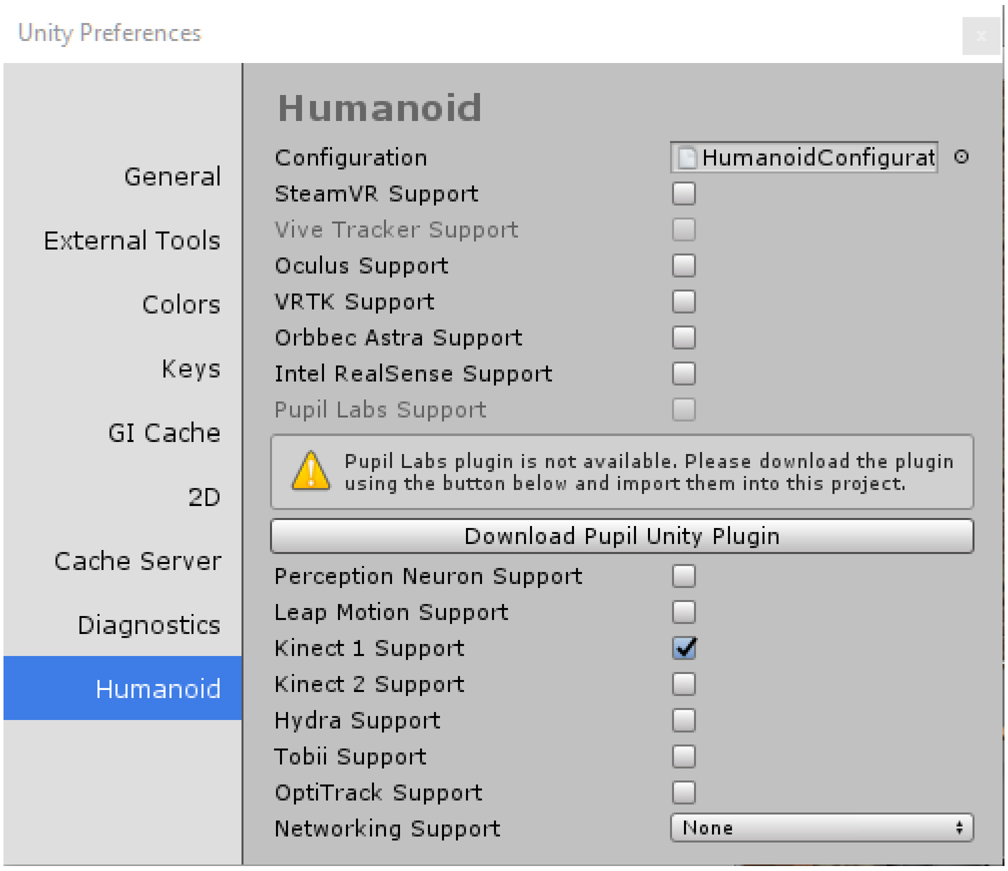
The Kinect for Windows SDK v1.7.0 or v1.8.0 is required for Microsoft Kinect support. It can be downloaded from the Microsoft Download Center.

Unity

Humanoid Control requires Unity version 5.5 and higher.

Setup

Microsoft Kinect support needs to be enabled in *Edit Menu->Preferences->Humanoid->Kinect 1 Support*.


Unity Preferences

General
External Tools
Colors
Keys
GI Cache
2D
Cache Server
Diagnostics
Humanoid

Humanoid

Configuration	HumanoidConfigurat
SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
VRTK Support	☐
Orbbec Astra Support	☐
Intel RealSense Support	☐
Pupil Labs Support	☐

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

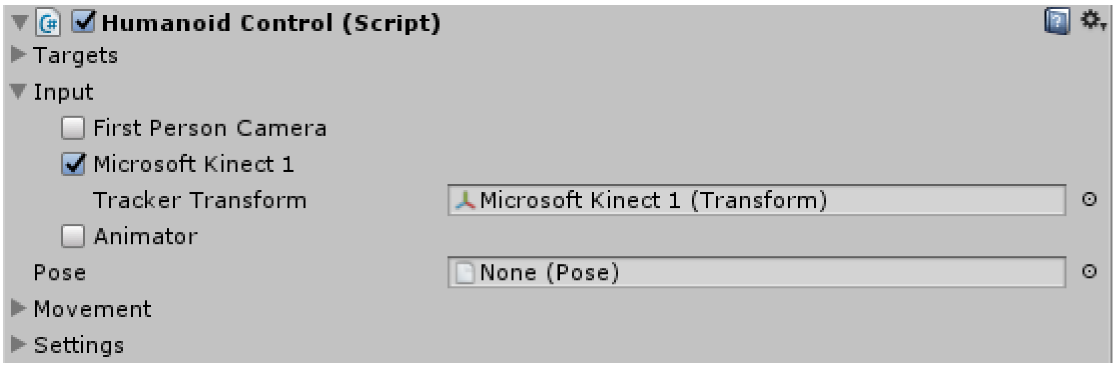
Download Pupil Unity Plugin

Perception Neuron Support	☐
Leap Motion Support	☐
Kinect 1 Support	✓
Kinect 2 Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

Disabling *Kinect 1 Support* ensures that no code related to Kinect 1 is included in the build.

Targets

To enable tracking for an avatar with Kinect, *Microsoft Kinect 2* needs to be enabled in the *Humanoid Control* component.


✓ Humanoid Control (Script)
Targets
Input
☐ First Person Camera
✓ Microsoft Kinect 1
Tracker Transform: Microsoft Kinect 1 (Transform)
☐ Animator
Pose: None (Pose)
Movement
Settings

The *Microsoft Kinect 1 (Transform)* is a reference to the Transform in the scene representing the Kinect Sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Microsoft Kinect 1 (Transform)* can be used to change the position of the tracking relative to the player in the scene.

When the Kinect 1 is used in combination with a VR headset, the position of the *Microsoft Kinect 1 (Transform)* is automatically determined.

Microsoft Kinect One / Kinect for Windows 2 (Plus)

The body tracking of the Microsoft Kinect One or 2 has been improved compared to the Xbox 360/Kinect. Although the functionality is largely the same, you will get better results with the new Kinect.

Kinect can be combined with Oculus Rift to get full body tracking.

Important Note: the combination of Kinect with HTC Vive is supported but will lead to major tracking issues for the HTC Vive because the IR beam from the Kinect will interfere with the IR signals coming from the lighthouses. It is recommended to position the Kinect at an angle of 90 relative to the lighthouses.

Prerequisites

Microsoft Kinect 2 is supported in the Humanoid Control Plus and Pro packages.

Hardware

Microsoft Kinect for Windows 2 and Microsoft Kinect One using the applicable adapter are supported.

Operating System

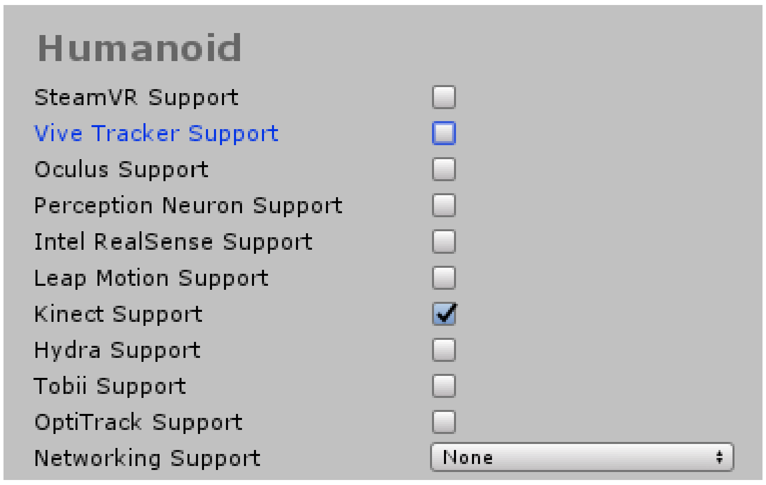
Microsoft Kinect 2 requires Microsoft Windows 10 or higher.

SDK

The Kinect for Windows SDK v2 is required for the Microsoft Kinect 2 support. It can be downloaded from the Microsoft Download Center.

Setup

Microsoft Kinect support needs to be enabled in *Edit Menu->Preferences->Humanoid->Kinect Support*.

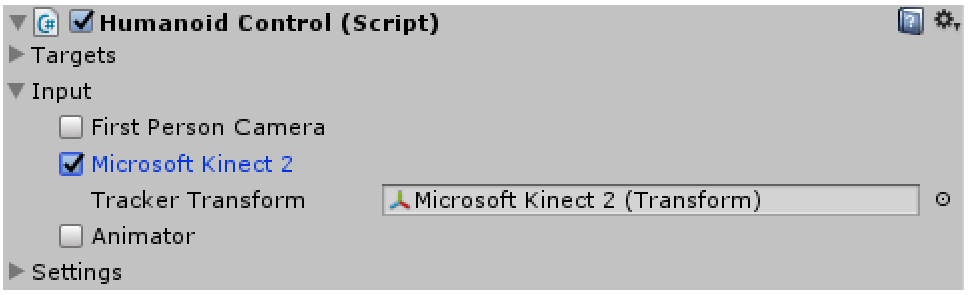

Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Perception Neuron Support	☐
Intel RealSense Support	☐
Leap Motion Support	☐
Kinect Support	✓
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

Disabling *Kinect Support* ensures that no code related to Kinect is included in the build.

Targets

To enable tracking for an avatar with Kinect, *Microsoft Kinect 2* needs to be enabled in the *Humanoid Control* component.


Humanoid Control (Script)
Targets
Input
☐ First Person Camera
☑ Microsoft Kinect 2
Tracker Transform: Microsoft Kinect 2 (Transform)
☐ Animator
Settings

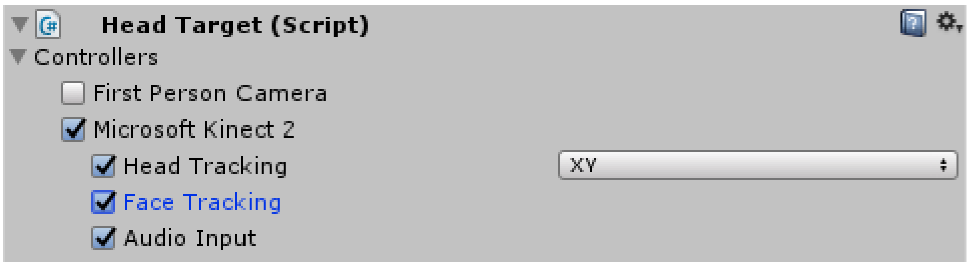
The *Microsoft Kinect 2 (Transform)* is a reference to the Transform in the scene representing the Kinect Sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Microsoft Kinect 2 (Transform)* can be used to changethe position of the tracking relative to the player in the scene.

When the Kinect 2 is used in combination with the Oculus Rift in a higher position, the position of the *Microsoft Kinect 2 (Transform)* is automatically determined.

Head Target

Head tracking, face tracking and audio input can be enabled separately on the head target. *Face Tracking* is only available in Humanoid Control Pro, which supports Face Tracking.


Head Target (Script)
Controllers
☐ First Person Camera
☑ Microsoft Kinect 2
☑ Head Tracking: XY
☑ Face Tracking
☑ Audio Input

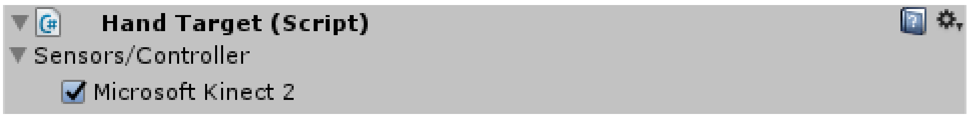
Two options are available for rotational head tracking:

- *XY*: which only rotates the head around the X and Y axis and therefore keeps the horizon horizontal
- *XYZ*: full three degrees of freedom tracking.

In both options, full positional tracking of the head within the range of the camera is supported.

Hand Targets

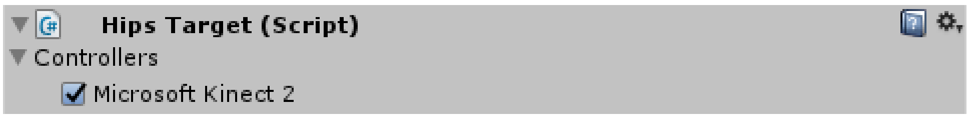
Positional and limited rotation tracking is supported. Due to the limitation of the Kinect, only the X and Y rotation axis are supported.


Hand Target (Script)
Sensors/Controller
☑ Microsoft Kinect 2

Hand movement input is limited to closing, opening the hand and the ‘ lasso’ position: index and middle finger pointing, all other fingers closed.

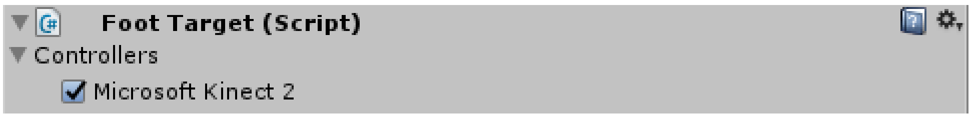
Hip Target

Only positional tracking is supported.


Hips Target (Script)
Controllers
☑ Microsoft Kinect 2

Foot Targets

Only positional tracking is possible.


Foot Target (Script)
Controllers
☑ Microsoft Kinect 2

Azure Kinect (Plus)

With the Azure Kinect it is possible to track bodies in many orientations. Compared to older Kinect devices and other 3D cameras it is no longer necessary to face the camera in order to have good tracking.

Prerequisites

Humanoid Control

Azure Kinect is supported in Humanoid Control Plus and Pro version 3.3 and higher.

Only one single user is supported for tracking.

Hardware

Body tracking with the Azure Kinect requires a recent NVidia graphics card. See the Azure Kinect sensor SDK system requirements page for a more detailed description.

Only a single Azure Kinect camera is supported.

Operating System

Azure Kinect is supported Microsoft Windows 10 64-bit.

SDK

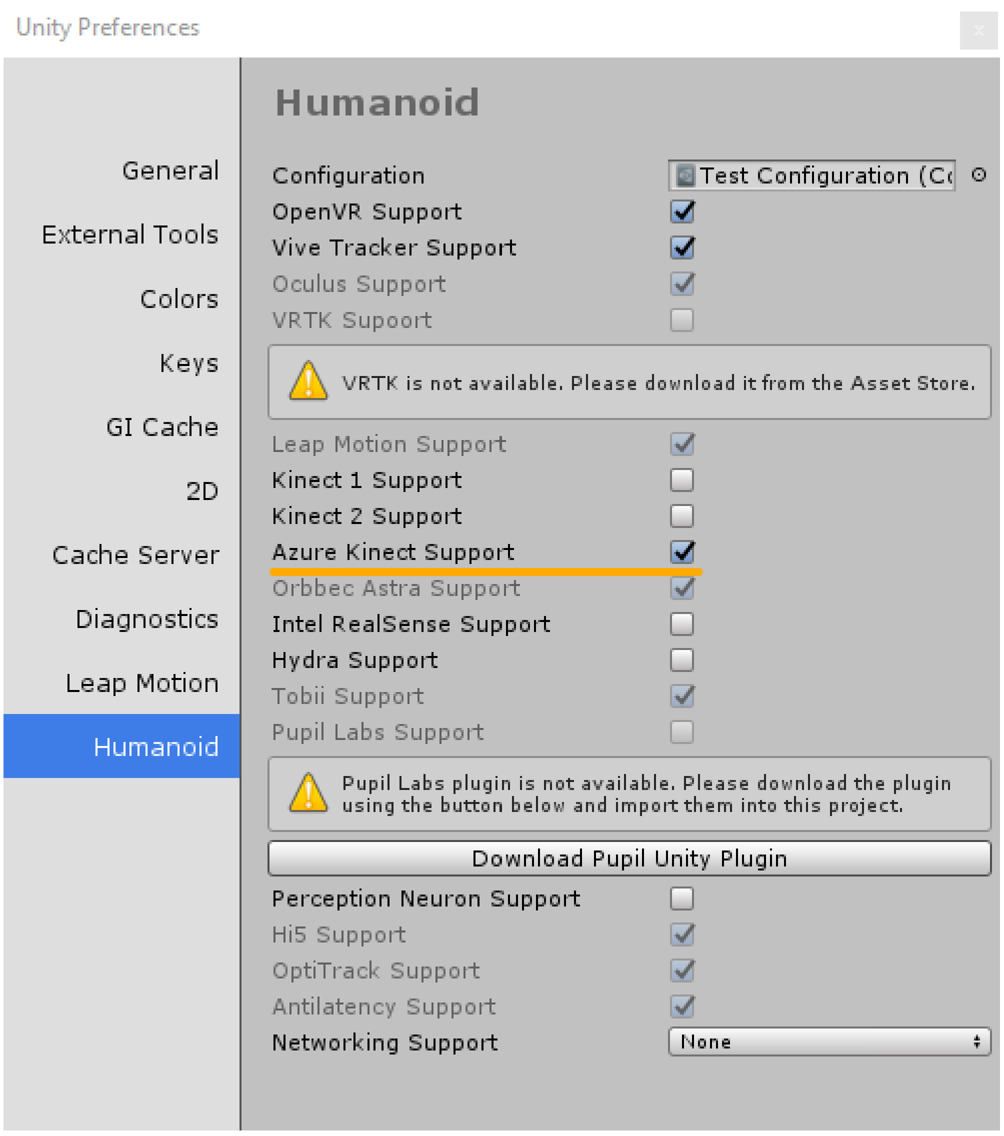
The following Azure Kinect SDKs are required:

Azure Kinect SDK v1.3 or higher

Azure Kinect Body Tracking SDK v1.0.1 or higher

Setup

Azure Kinect support needs to be enabled in the Preferences: *Edit Menu->Preferences->Humanoid->Azure Kinect Support*


Unity Preferences

General	
External Tools	
Colors	
Keys	
GI Cache	
2D	
Cache Server	
Diagnostics	
Leap Motion	
Humanoid	

Humanoid

Configuration	Test Configuration (C
OpenVR Support	✓
Vive Tracker Support	✓
Oculus Support	✓
VRTK Support	
Leap Motion Support	✓
Kinect 1 Support	
Kinect 2 Support	
Azure Kinect Support	✓
Orbbec Astra Support	✓
Intel RealSense Support	
Hydra Support	
Tobii Support	✓
Pupil Labs Support	
Perception Neuron Support	
Hi5 Support	✓
OptiTrack Support	✓
Antilatency Support	✓
Networking Support	None

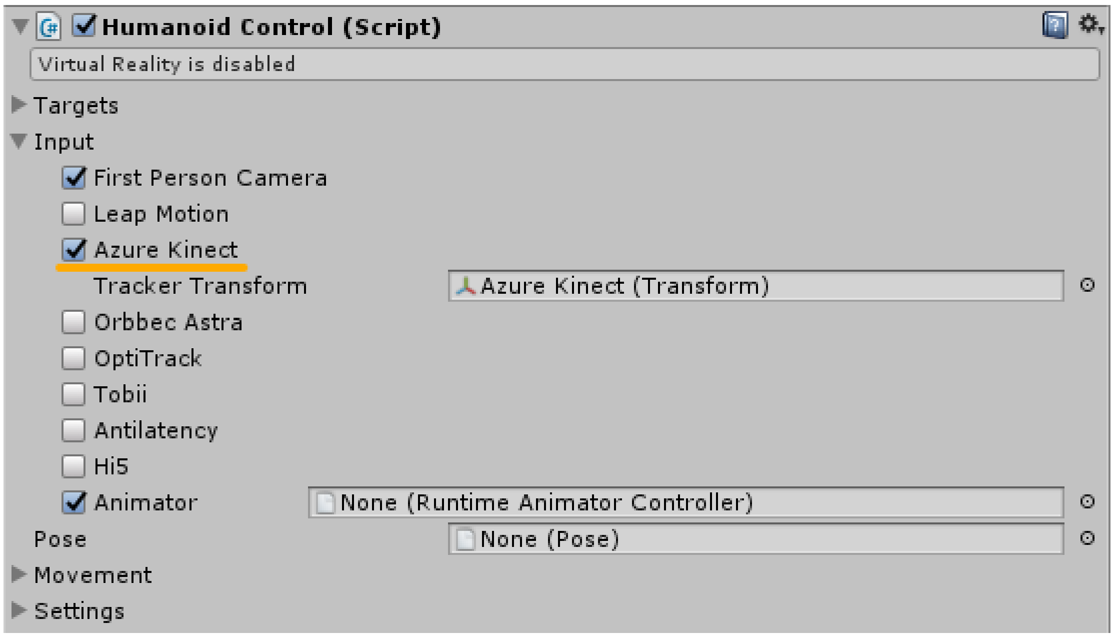
VRTK is not available. Please download it from the Asset Store.

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

Disabling Azure Kinect Support in the Preferences ensures that no code related to Azure Kinect is included in the build.

Tracking

To enable tracking for a humanoid with Azure Kinect, *Azure Kinect* needs to be enabled in the *Humanoid Control* component for that humanoid.


Humanoid Control (Script)

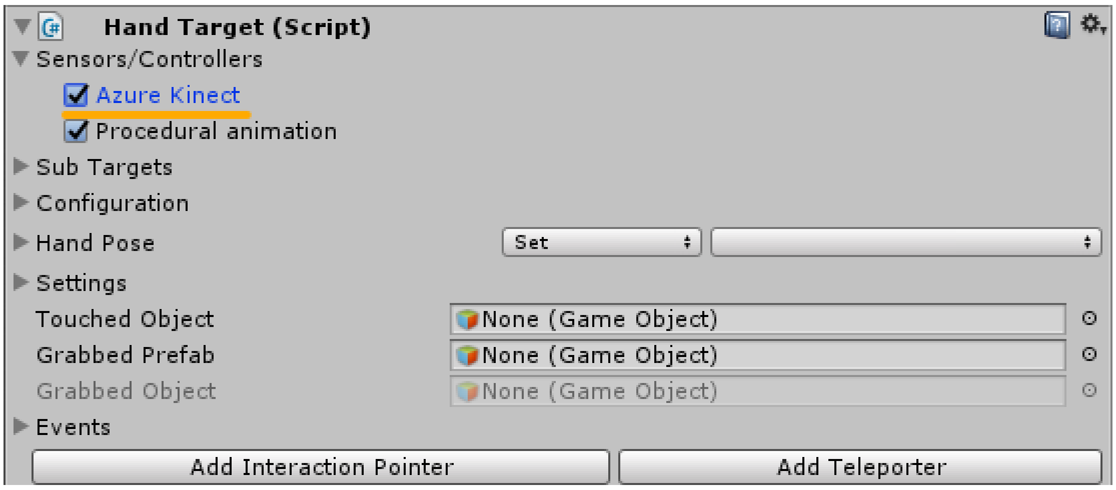
Virtual Reality is disabled

- Targets
- Input
  - [✓] First Person Camera
  - [ ] Leap Motion
  - [✓] Azure Kinect
    - Tracker Transform: Azure Kinect (Transform)
  - [ ] Orbbec Astra
  - [ ] OptiTrack
  - [ ] Tobii
  - [ ] Antilatency
  - [ ] Hi5
  - [✓] Animator: None (Runtime Animator Controller)
- Pose: None (Pose)
- Movement
- Settings

The *Tracker Transform* is a reference to the Transform in the scene representing the Azure Kinect Camera. The *Azure Kinect* GameObject is found as a child of the *Real World* GameObject in the hierarchy.

The *Azure Kinect* GameObject can be used to change the position and rotation of the tracking within the scene.

It is possible to disable tracking of parts of the body by disabling *Azure Kinect* on the Head, Hand, Hips or Foot Targets.


Hand Target (Script)

- Sensors/Controllers
  - [✓] Azure Kinect
  - [✓] Procedural animation
- Sub Targets
- Configuration
- Hand Pose: Set
- Settings
- Touched Object: None (Game Object)
- Grabbed Prefab: None (Game Object)
- Grabbed Object: None (Game Object)
- Events

Orbbec Astra (Plus)

Orbbec Astra provides full body tracking.

Prerequisites

Humanoid Control

Orbbec Astra is supported in Humanoid Control Plus and Pro version 2.0 and higher.

Hardware

Orbbec Astra, Astra S and Astra Pro are supported.

Operating System

Orbbec Astra is only supported on Microsoft Window 10.

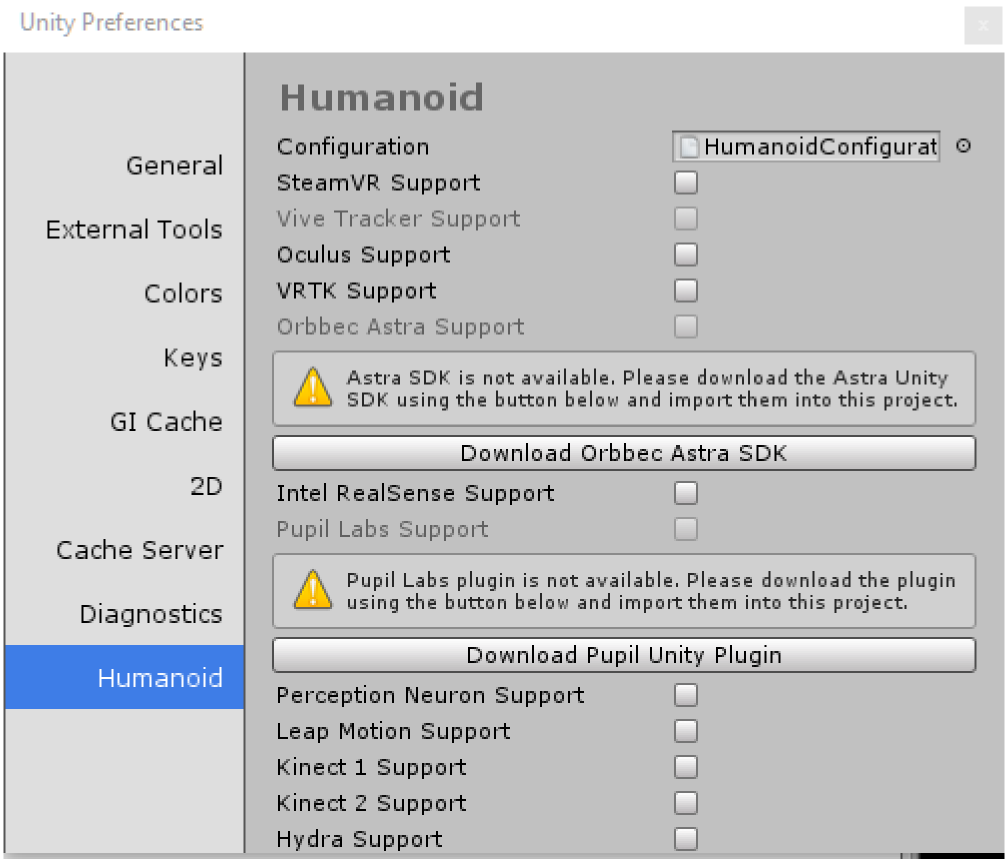
Unity

Unity version 2017.4 or higher is required.

Setup

For the Orbbec Astra to work, the Astra SDK Package for Unity needs to be imported into the project.

Go to Edit *Menu->Preferences->Humanoid* and look for the *Orbbec Astra Support* entry:


Unity Preferences

Humanoid

Configuration: HumanoidConfigurat
SteamVR Support
Vive Tracker Support
Oculus Support
VRTK Support
Orbbec Astra Support

Astra SDK is not available. Please download the Astra Unity SDK using the button below and import them into this project.

Download Orbbec Astra SDK

Intel RealSense Support
Pupil Labs Support

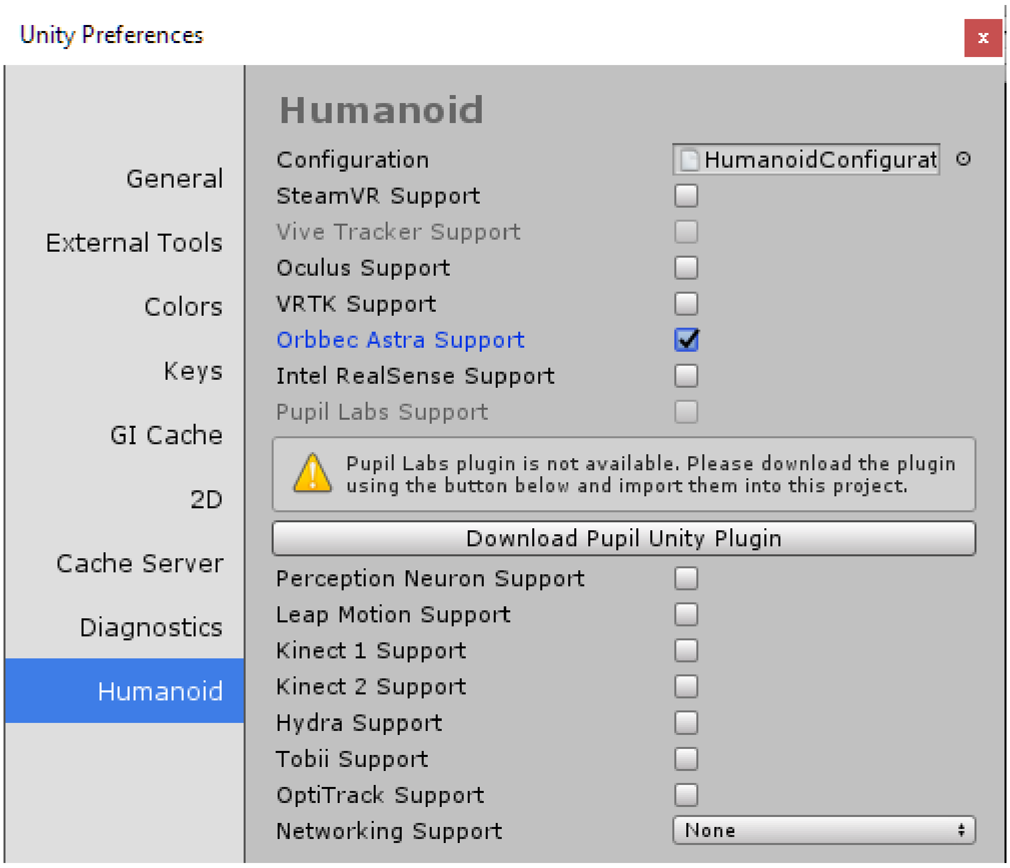
Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

Download Pupil Unity Plugin

Perception Neuron Support
Leap Motion Support
Kinect 1 Support
Kinect 2 Support
Hydra Support

Click the button *Download Orbbec Astra SDK* to go to the download page for the Astra SDK.

After the SDK has been imported, the *Orbbec Astra Support* can be enabled in the preferences.


Unity Preferences

General | External Tools | Colors | Keys | GI Cache | 2D | Cache Server | Diagnostics | Humanoid

Humanoid

Configuration	HumanoidConfigurat
SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
VRTK Support	☐
Orbbec Astra Support	✓
Intel RealSense Support	☐
Pupil Labs Support	☐

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

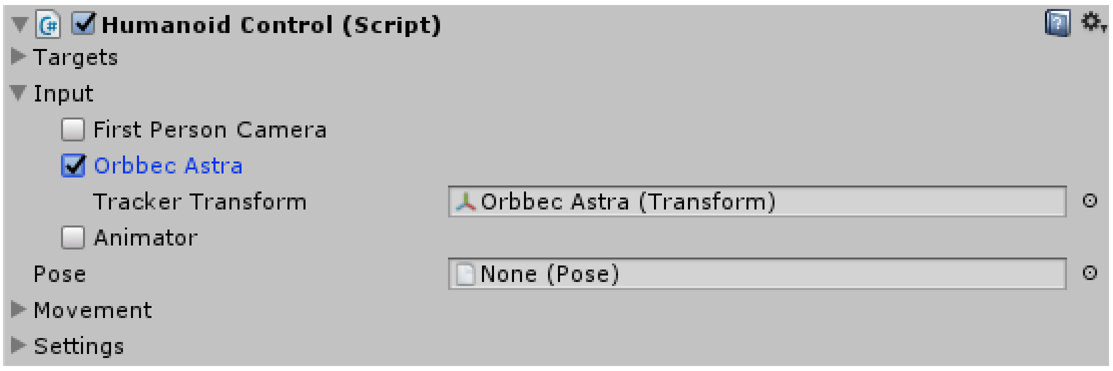
Download Pupil Unity Plugin

Perception Neuron Support	☐
Leap Motion Support	☐
Kinect 1 Support	☐
Kinect 2 Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

Disabling *Orbbec Astra Support* ensures that no code related to the Orbbec Astra is included in the build.

Targets

To enable Orbbec Astra tracking for an avatar, *Orbbec Astra* needs to be enabled in the *Humanoid Control* component.


Humanoid Control (Script)
- Targets
- Input
  - ☐ First Person Camera
  - ✓ Orbbec Astra
    - Tracker Transform: Orbbec Astra (Transform)
  - ☐ Animator
- Pose: None (Pose)
- Movement
- Settings

The *Orbbec Astra (Transform)* is a reference to the Transform in the scene representing the Astra sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Orbbec Astra (Transform)* can be used to change the position of the tracking relative to the player in the scene.

When the Astra is used in combination with a VR headset, the position of the *Orbbec Astra Transform* is determined automatically.

HTC Vive Trackers (Plus)

Vive Trackers are an extension for the OpenVR support which makes it possible to tracks body parts like head, arms, hips and feet by attaching them to the various body parts.

Prerequisites

Vive Trackers are supported in the Plus and Pro packages.

Vive trackers or not yet supported in combination with Unity XR.

Unity

Unity versions 2017.4 up to 2019.4 are supported.

Hardware

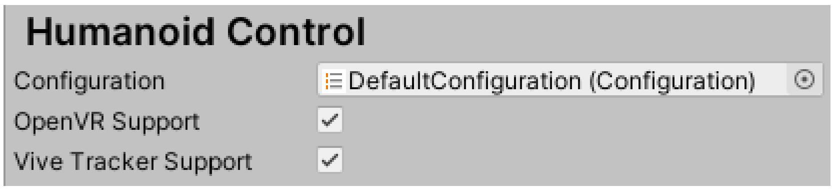
HTC Vive Trackers require the HTC Vive headset connected to the same system.

Operating System

Vive Trackers are supported on the Microsoft Windows platform.

Setup

For HTC Vive Tracker support, both *SteamVR Support* and *Vive Tracker Support* need to be enabled in the Humanoid Preferences. You can find these in the *Edit Menu->Preferences->Humanoid*.



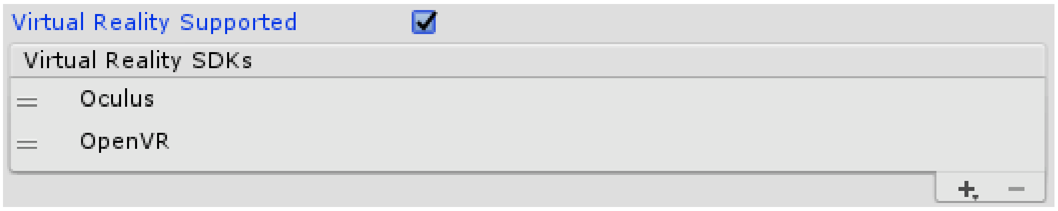
Humanoid Control

Configuration: DefaultConfiguration (Configuration)

OpenVR Support: ✓

Vive Tracker Support: ✓

Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *OpenVR* needs to be added to the *Virtual Reality SDKs*.



Virtual Reality Supported: ✓

Virtual Reality SDKs

- Oculus
- OpenVR

Targets

The location of Vive Trackers on the body is recognized automatically, so there is no need to register tracker IDs. The conditions for detection are listed below for each target.

When not all trackers have been identified yet, it is possible that they change location on the body. For example when a tracker is first recognized as a hip tracker and later moves above 1.2m and no head tracker has been detected yet, it will transform into a head tracker.

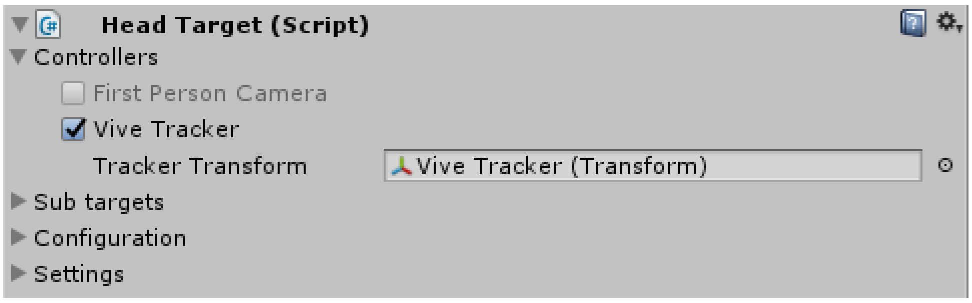
Trackers are only recognized being on a certain body part if the Vive Tracker has been enabled for that body part. So if Vive Trackers are not enabled on the feet, no Vive Tracker will be recognized as a foot tracker.

The tracker assignment process is restarted when Calibrate() is called on the Humanoid Control component.

Head Target

A Vive Tracker can be used on the head instead of an HMD. There is no need to have a First Person Camera or any camera at all in the scene.

It is not possible to have both a HMD (First Person Camera) and Vive Tracker on the head.



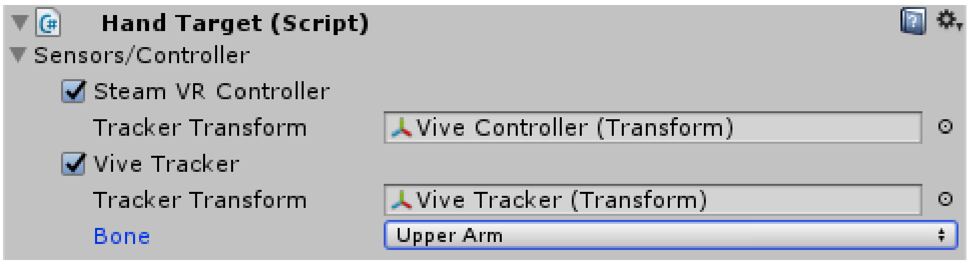
Head Target (Script)
Controllers
First Person Camera
✓ Vive Tracker
Tracker Transform: Vive Tracker (Transform)
Sub targets
Configuration
Settings

A head mounted Vive Tracker is recognized automatically when an active tracker is detected at least 1.2m above ground level.

Hand Target

Vive Trackers can be placed on different bones on the arm. This makes it possible to combine a SteamVR controller with a Vive Tracker on the arm to improve the Inverse Kinematics solution. The bone on which the tracker in mounted should be set using the *Bone* parameter.

It is not possible to have both a SteamVR controller and a Vive Tracker on the hand.



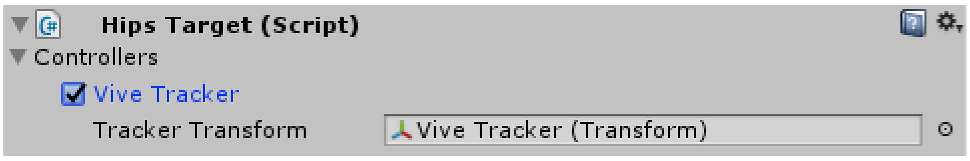
Hand Target (Script)
Sensors/Controller
✓ Steam VR Controller
Tracker Transform: Vive Controller (Transform)
✓ Vive Tracker
Tracker Transform: Vive Tracker (Transform)
Bone: Upper Arm

Trackers on the arm are recognized at the left and rightmost tracker.

Left and right are relative to the forward direction of the headset. So even if the HMD is not used, it still needs to be place such that it is pointing in the forward direction at start.

Hips Target

A Vive Tracker can be placed on the hips. In this position the orientation of the tracker is not used, so it is not important how the tracker is rotated around the Vive logo (forward direction).

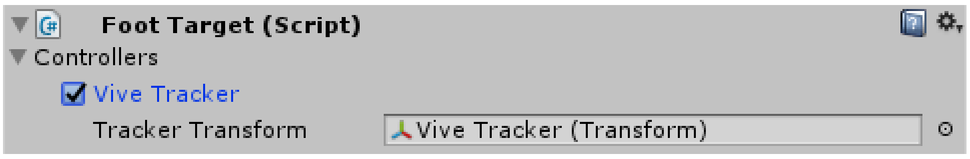


Hips Target (Script)
Controllers
✓ Vive Tracker
Tracker Transform: Vive Tracker (Transform)

A Tracker on the hips is recognized when an active tracker is found between 0.3m and 1m above ground level.

Foot Target

Vive Trackers on the foot are recognized when they are less than 0.2m above ground level. The left foot tracker is the leftmost tracker below 0.2m, the right foot tracker the rightmost tracker below 0.2m.


Foot Target (Script)
Controllers
☑ Vive Tracker
Tracker Transform: Vive Tracker (Transform)

Left and right are relative to the forward direction of the headset. So even if the HMD is not used, it still needs to be place such that it is pointing in the forward direction at start.

Perception Neuron (Pro)

Full body tracking is supported with Perception Neuron up to 32 bones.

Prerequisites

Perception Neuron is supported in Humanoid Control Pro.

Hardware

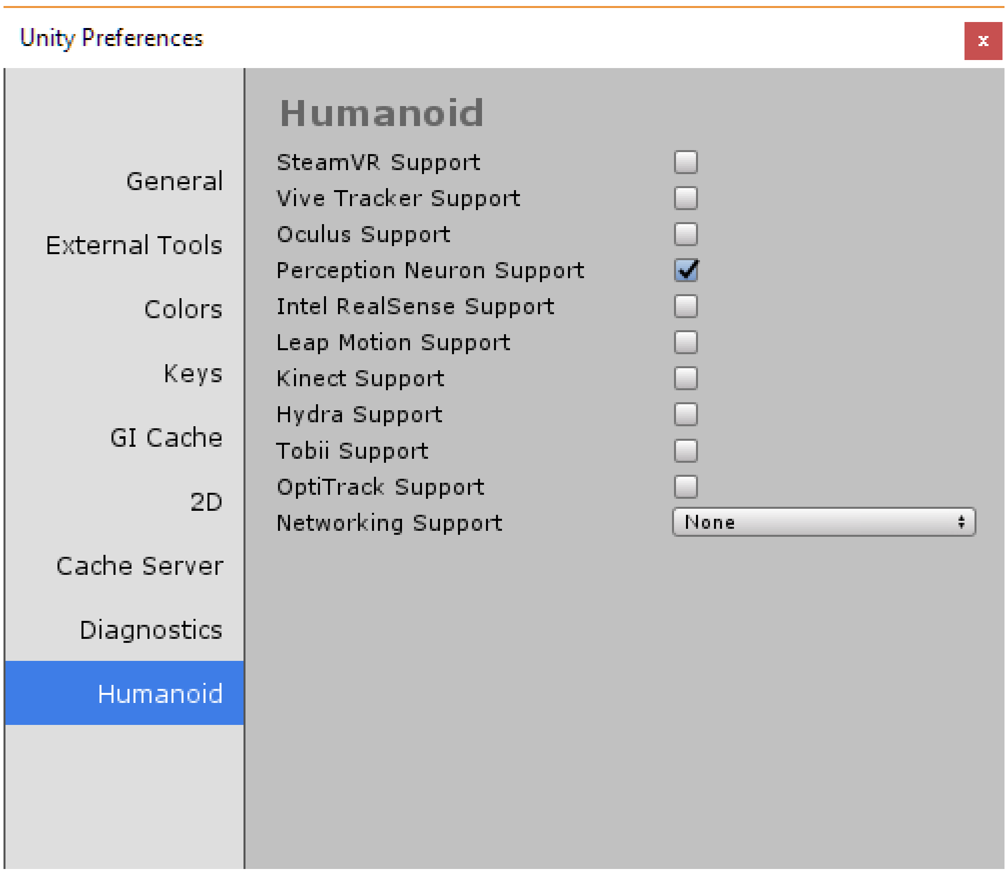
Perception Neuron with up to 32 neurons is supported.

Operating System

Microsoft Windows is required.

Setup

Make sure Perception Neuron support is enabled. Go to *Edit Menu->Preferences->Humanoid* and make sure *Perception Neuron Support* is enabled.



Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Perception Neuron Support	✓
Intel RealSense Support	☐
Leap Motion Support	☐
Kinect Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

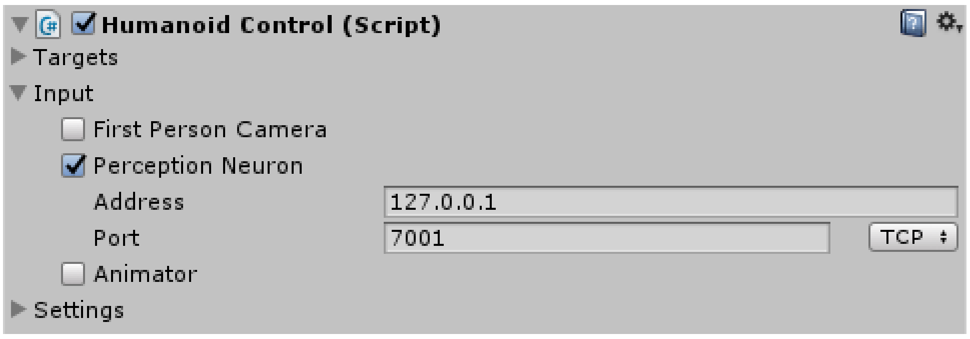
Disabling Perception Neuron support ensures that no code related to Perception Neuron is included in the build.

Axis Neuron

Axis Neuron software is required to support Perception Neuron. In the Settings-Broadcasting, make sure that *BVH* is enabled in *Binary* format, without *Use old header*. *ServerPort* needs to be set to 7001. The protocol needs to be set to *TCP*.

Targets

To enable Perception Neuron tracking for an avatar, *Perception Neuron* needs to be enabled in the Humanoid Control component.



Humanoid Control (Script)
- Targets
- Input
  - [ ] First Person Camera
  - [✓] Perception Neuron
    - Address: 127.0.0.1
    - Port: 7001 TCP
  - [ ] Animator
- Settings

The *Address* should match the IP address of the computer running the Axis Neuron software. The Port and Protocol TCP/UDP should match the settings in Axis Neuron as described above.

Neuron can be enabled separately for the head, hand, hips and foot targets. Disabling Neuron on one of these targets will disable tracking of the complete body part associated to that target. E.g. the hand will control the full arm including the shoulder.

Finger Movements

Finger movements can be tracked directly from the Neuron sensors on the hand.

Tobii Eyetracking (Pro)

Tobii Eyetracking is supported for head and eye tracking.

Prerequisites

Tobii Eyetracking is supported in the Humanoid Control Pro package.

Hardware

- Tobii Eye Tracker 4C
- Tobii EyeX (untested)
- Eye Tracking Laptops and Monitors (untested)

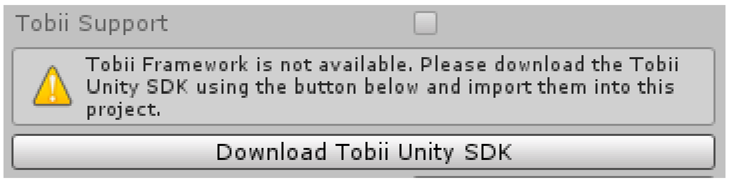
Operating System

Microsoft Windows is required.

Setup

For Tobii Eyetracking to work, the Tobii Unity SDK needs to be imported into the project.

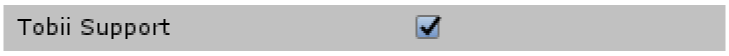
Go to *Edit Menu->Preferences->Humanoid* and look for the *Tobii Support* entry:


Tobii Support

Tobii Framework is not available. Please download the Tobii Unity SDK using the button below and import them into this project.

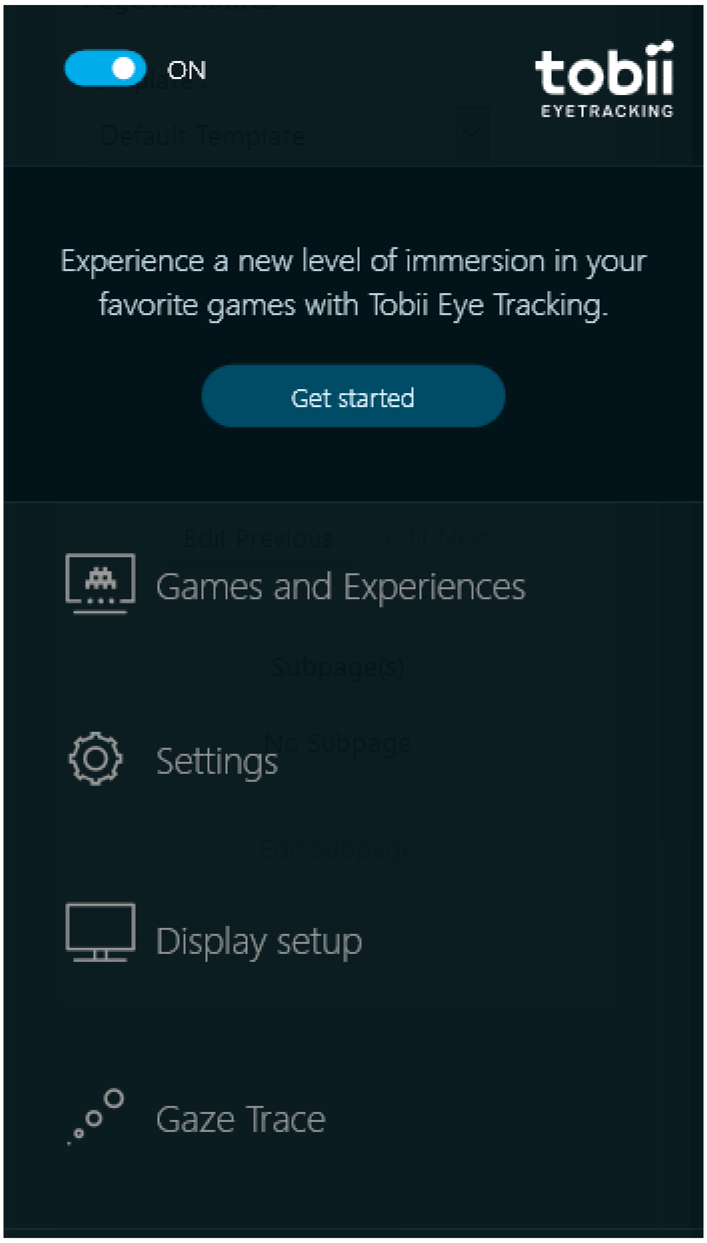
Click the button to go to the download page for the Tobii SDK.

After the SDK has been imported, *Tobii Support* can be enabled in the preferences


Tobii Support ✓

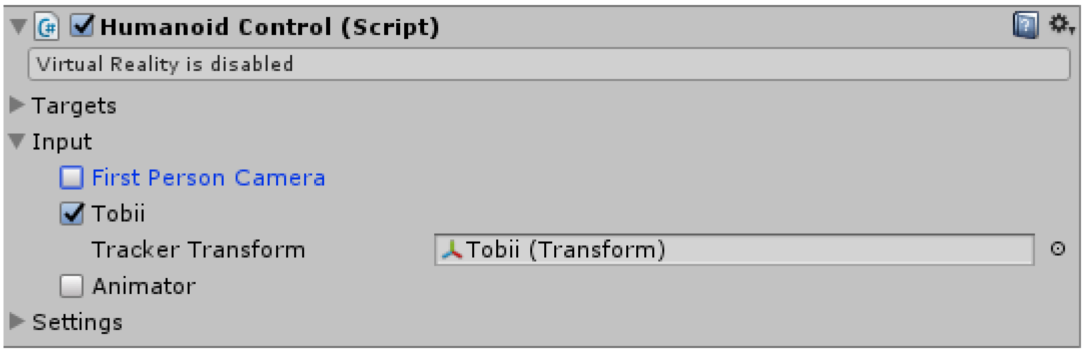
Disabling Tobii Support ensures that no code related to Tobii EyeTracking is included in the build.

Note: Tobii Eyetracking has to be enabled in the Control panel for tracking to work:


ON
tobii EYETRACKING
Experience a new level of immersion in your favorite games with Tobii Eye Tracking.
Get started
Games and Experiences
Settings
Display setup
Gaze Trace

Targets

To enable tracking with the Tobii eyetracker for an avatar, *Tobii* needs to be enabled in the Humanoid Control component.



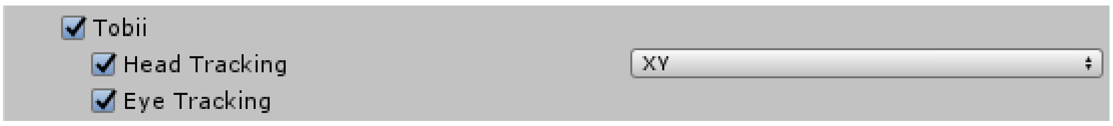
Humanoid Control (Script)
Virtual Reality is disabled
Targets
Input
☐ First Person Camera
☑ Tobii
Tracker Transform: Tobii (Transform)
☐ Animator
Settings

The *Tobii (Transform)* is a reference to the Transform in the scene representing the Tobii Tracker. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The Tobii (Transform) can be used to set the position of the tracking relative to the player in the scene.

Head Target

Head and eye tracking can be enabled separately on the head target.



☑ Tobii
☑ Head Tracking XY
☑ Eye Tracking

Two options are available for rotational head tracking:

- *XY*: which only rotates the head around the X and Y axis and therefore keeps the horizon horizontal
- *XYZ*: full three degrees of freedom tracking.

In both options, full positional tracking of the head within the range of the camera is supported.

OptiTrack (Pro)

Rigidbody and skeleton tracking is supported with OptiTrack

Prerequisites

OptiTrack is supported in Humanoid Control Pro.

Hardware

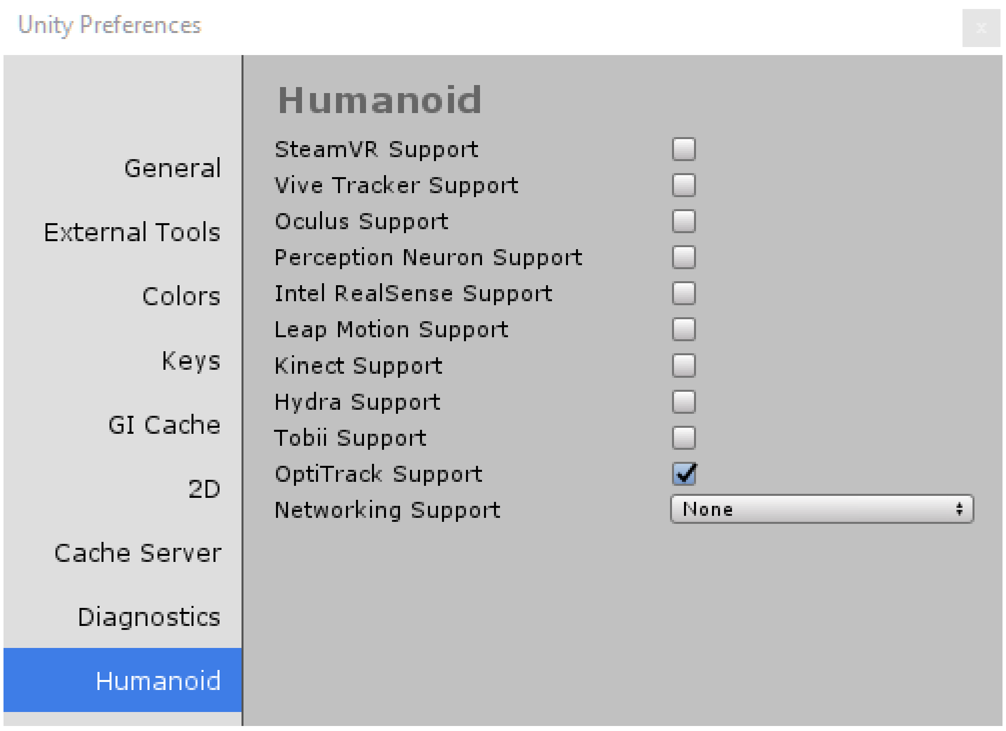
All OptiTrack hardware supported by Motive Tracker or Body 2.0.

Operating System

Microsoft Windows is required.

Setup

Make sure OptiTrack support is enabled. Go to *Edit Menu->Preferences->Humanoid* and make sure *OptiTrack Support* is enabled.



Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Perception Neuron Support	☐
Intel RealSense Support	☐
Leap Motion Support	☐
Kinect Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	✓
Networking Support	None

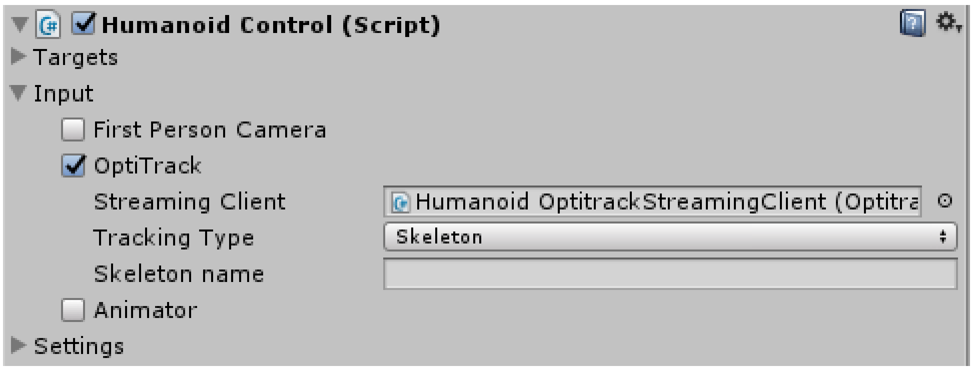
Disabling OptiTrack support ensures that no code related to OptiTrack is included in the build.

Motive

Motive:Tracker or Body is required for rigidbody respectively skeleton tracking. Ensure that Data Streaming is enabled.

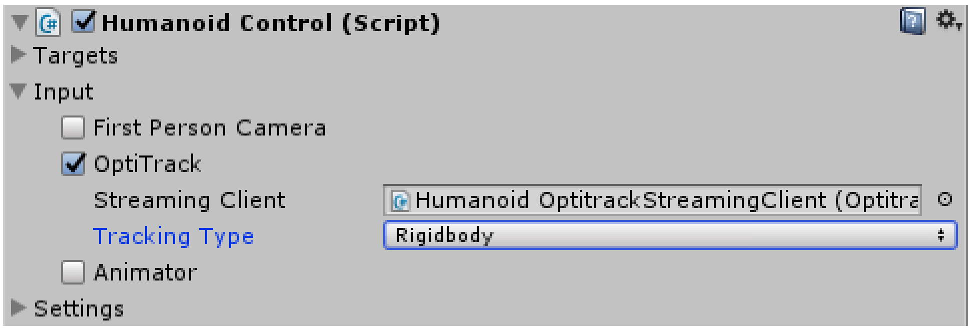
Targets

To enable OptiTrack tracking for an avatar, *OptiTrack* needs to be enabled in the Humanoid Control component.


Humanoid Control (Script)
Targets
Input
First Person Camera
OptiTrack
Streaming Client: Humanoid OptitrackStreamingClient (Optitra
Tracking Type: Skeleton
Skeleton name:
Animator
Settings

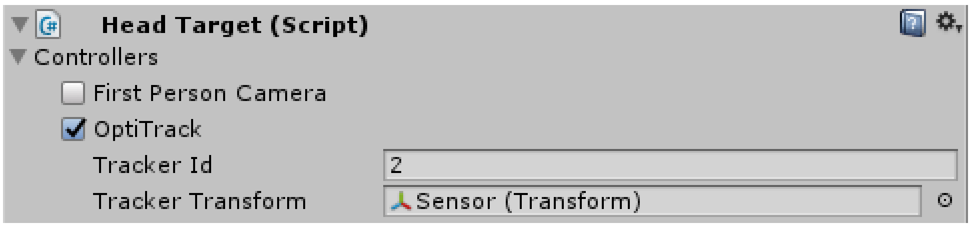
Enabling OptiTrack will automatically add an Humanoid OptitrackStreamingClient gameObject to the scene which will implement the communication with Motive. Make sure the *Streaming Client* parameter is pointing to this object.

When *Skeleton Tracking* is selected as the Tracking Type you will need Motive:Body to stream the captured skeleton information to Humanoid Control. The *Skeleton name* parameter determines which OptiTrack skeleton is moving this Humanoid avatar.


Humanoid Control (Script)
Targets
Input
First Person Camera
OptiTrack
Streaming Client: Humanoid OptitrackStreamingClient (Optitra
Tracking Type: Rigidbody
Animator
Settings

When *Rigidbody* is selected as the Tracking Type, either Motive:Tracker or Body can be used to stream the captured data. One can then select which rigidbody is tracking a body part in the Targets.

OptiTrack can be enabled separately for the head, hand, hips and foot targets.


Head Target (Script)
Controllers
First Person Camera
OptiTrack
Tracker Id: 2
Tracker Transform: Sensor (Transform)

For each target, a Tracker Id can be set to select which rigidbody is controlling this target. This Id corresponds with the Steaming ID of the Rigidbody in Motive.

Antilatency (Pro)

Prerequisites

Humanoid Control

Antilatency is support in Humanoid Control Pro version 3.2 and higher.

Hardware

Currently only a HMD Radio Socket + bracers are supported in combination with an Oculus Rift headset.

Operating System

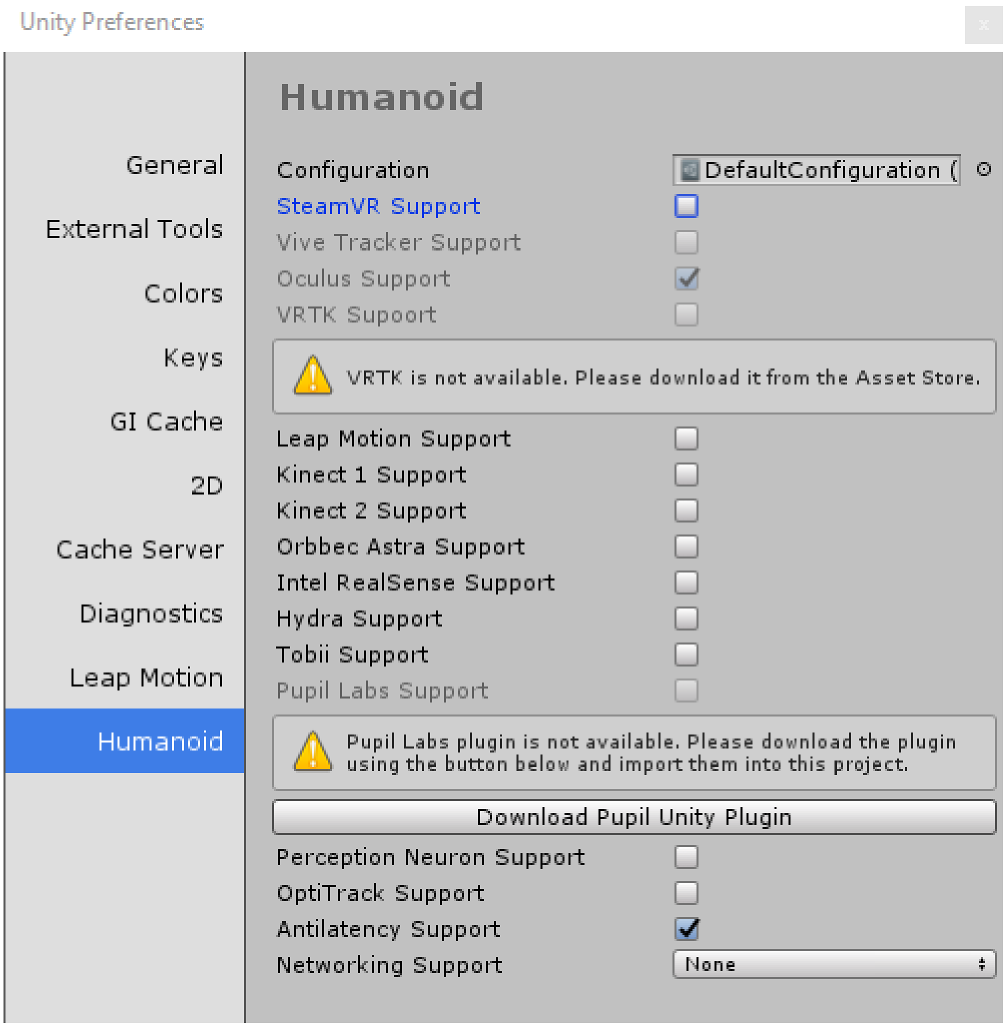
Currently only Microsoft Windows is supported

Antilatency SDK

Antilatency supports requires the Antilatency SDK v2.0.0 or higher to be included in the project.

Setup

Make sure Antilatency Support is enabled in the preferences. Go to *Edit Menu->Preferences->Humanoid* and make sure *Antilatency Support* is enabled.


Unity Preferences

Humanoid

Configuration: DefaultConfiguration (
SteamVR Support: ☐
Vive Tracker Support: ☐
Oculus Support: ✓
VRTK Support: ☐

VRTK is not available. Please download it from the Asset Store.

Leap Motion Support: ☐
Kinect 1 Support: ☐
Kinect 2 Support: ☐
Orbbec Astra Support: ☐
Intel RealSense Support: ☐
Hydra Support: ☐
Tobii Support: ☐
Pupil Labs Support: ☐

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

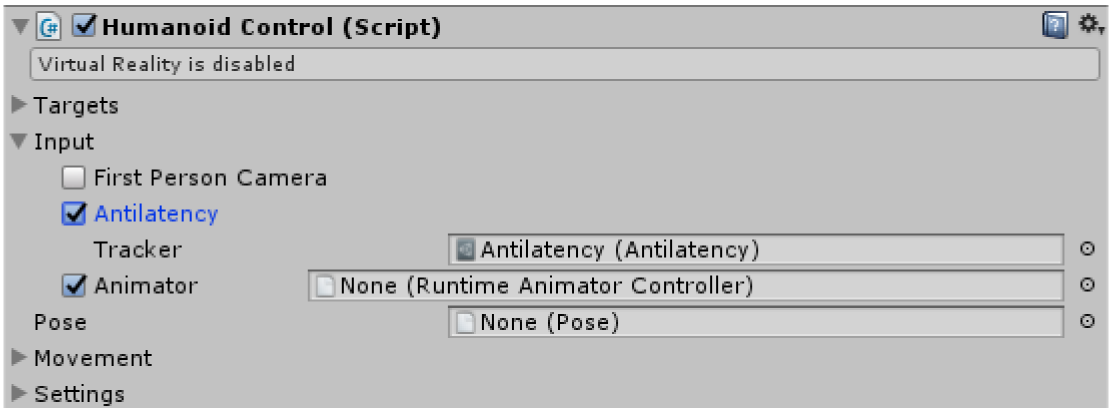
Download Pupil Unity Plugin

Perception Neuron Support: ☐
OptiTrack Support: ☐
Antilatency Support: ✓
Networking Support: None

Disabling Antilatency support ensures that no code related to Antilatency is included in the build.

Targets

To enable tracking for an avatar with Antilatency, *Antilatency* needs to be enabled in the *Humanoid Control* component:



Humanoid Control (Script)
Virtual Reality is disabled
Targets
Input
☐ First Person Camera
☑ Antilatency
Tracker: Antilatency (Antilatency)
☑ Animator: None (Runtime Animator Controller)
Pose: None (Pose)
Movement
Settings

The *Tracker* is a reference to the Antilatency Tracker in the Real World GameObject representing the Antilatency tracking origin.

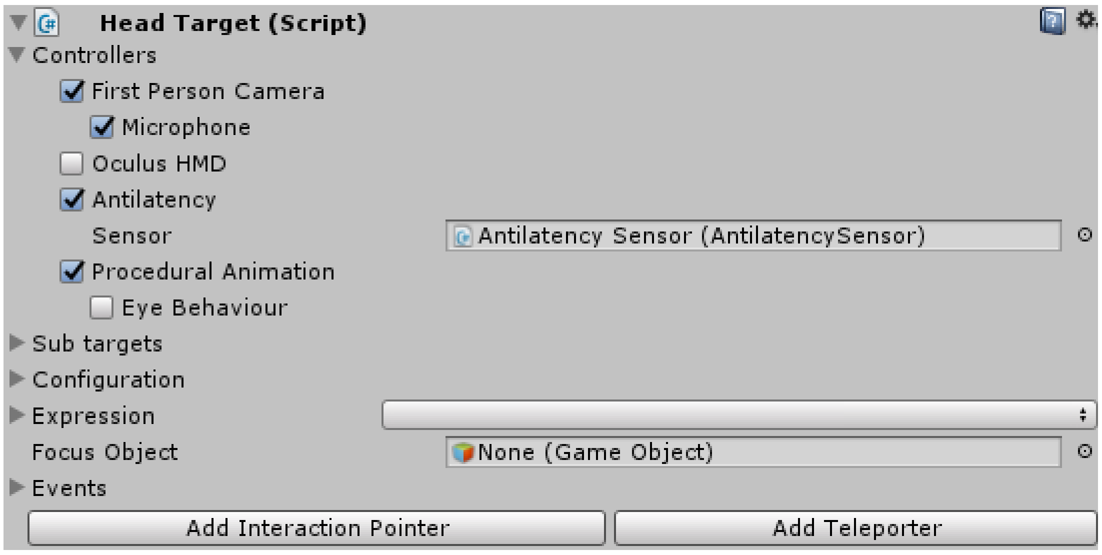
The Tracker can be used to change the origin of the tracking space relative to the player in the scene.

Head Target

Antilatency can be used to track a headset. You can rely solely on the Antilatency tracking or you can combine the Antilatency tracking with the native headset tracking.

Pure Antilatency Head Tracking

In this setup make sure you have Antilatency enabled and the native HMD disabled on the Head Target:

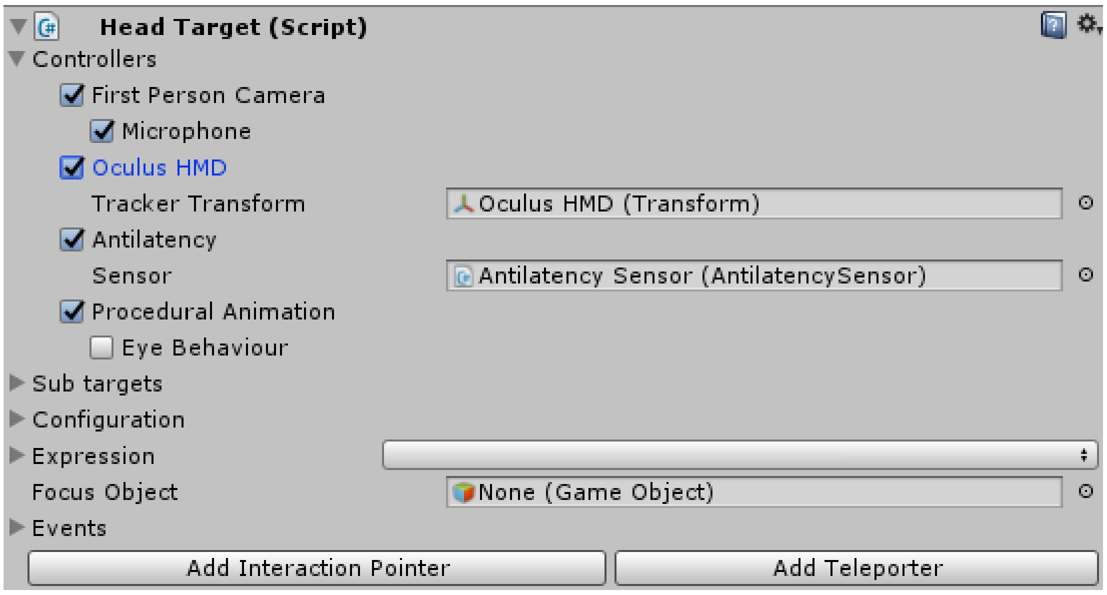


Head Target (Script)
Controllers
☑ First Person Camera
☑ Microphone
☐ Oculus HMD
☑ Antilatency
Sensor: Antilatency Sensor (AntilatencySensor)
☑ Procedural Animation
☐ Eye Behaviour
Sub targets
Configuration
Expression
Focus Object: None (Game Object)
Events
Add Interaction Pointer
Add Teleporter

In this setup, the native tracking of the headset (Oculus in this example) is ignored and the headset will be tracked fully by the input of the Antilatency tracking system.

Fused Native and Antilatency Tracking

If you enable both Antilatency and the native HMD tracking Humanoid Control will use the Anatilatency tracking in combination with the native headset tracking.


Head Target (Script)

Controllers
- ✓ First Person Camera
  - ✓ Microphone
- ✓ Oculus HMD
  - Tracker Transform: Oculus HMD (Transform)
- ✓ Antilatency
  - Sensor: Antilatency Sensor (AntilatencySensor)
- ✓ Procedural Animation
  - ☐ Eye Behaviour

Sub targets
Configuration
Expression
Focus Object: None (Game Object)
Events

The native headset tracking will be used primarily, but Antilatency is used to set the player at the right position and orientation relative to the Antilatency floor. This ensures that the relative positions of the avatars will match the relative positions and orientations of the users in real life.

Besides that, other objects or body parts tracked by Antilatency will appear in the correct positions.

Furthermore, the Antilatency tracking is used to correct possible inaccuracies in the positional tracking. This is most notable when using the Oculus Quest or Oculus Rift S tracking over larger distances (> 3m) where the accuracy decreases.

Lastly, Antilatency can be used to add positional tracking to headsets which provide only rotational tracking data, like the Samsung Gear VR or Oculus GO.

Antilatency Sensor

The Antilatency Sensor gives information about the tracking at runtime and has a Alt Tracking Sensor Component which can be used to change the Socket Tag of the Antilatency Tag.

The default value of the Socket Tag for the Head Sensor is *HMD*. Make sure this matches the Tag set using the AntilatencyService for the HMD Radio Socket Tag.

Hand Target

The Antilatency Bracers can be used to track the hand for the avatar. The bracers have a built-in touch sensor which can be used to closed the hand of the avatar.

The Bracer Touch is mapped to Trigger 1 of the Controller Input. When Finger Movements have been enabled in the Controller Input, this will close the hand of the avatar when the Bracer touch sensor is touched.

Antilatency Sensor

The Antilatency Sensor gives information about the tracking at runtime and has a Alt Tracking Sensor Component which can be used to change the Socket Tag of the Antilatency Tag.

The default value of the Socket Tag for the Hand Sensors is *LeftHand* and *RightHand*. Make sure this matches the Tag set using the AntilatencyService for the Bracers.

Apple ARKit (Pro)

With the Apple ARkit it is possible to do accurate face tracking in combination with Humanoid Control

Prerequisites

Apple ARKit face tracking is supported in the Humanoid Control Pro edition.

Unity

ARKit face tracking requires Unity 2019.1 or higher.

Hardware

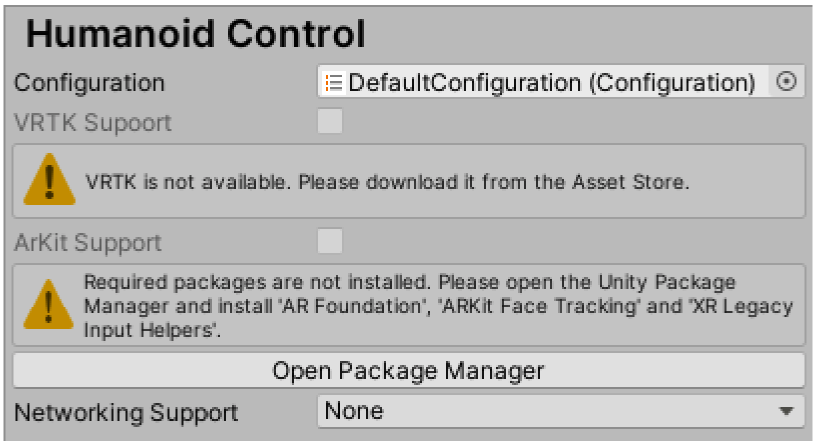
Apple iOS hardware compatible with ARKit face tracking is required.

Operating System

ARKit face tracking requires iOS and MacOS for development.

Setup

For ARKit face tracking to work, a number of packages need to be enabled in Unity



Humanoid Control

Configuration: DefaultConfiguration (Configuration)

VRTK Support

VRTK is not available. Please download it from the Asset Store.

ArKit Support

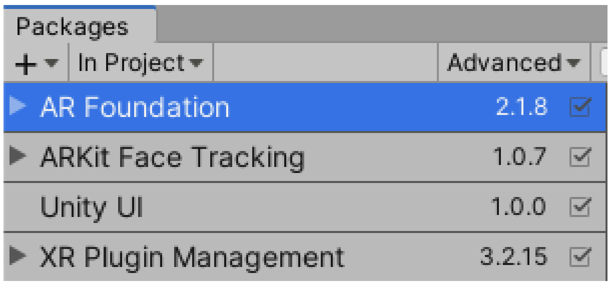
Required packages are not installed. Please open the Unity Package Manager and install 'AR Foundation', 'ARKit Face Tracking' and 'XR Legacy Input Helpers'.

Networking Support: None

In the Unity Package Manager install the following packages:

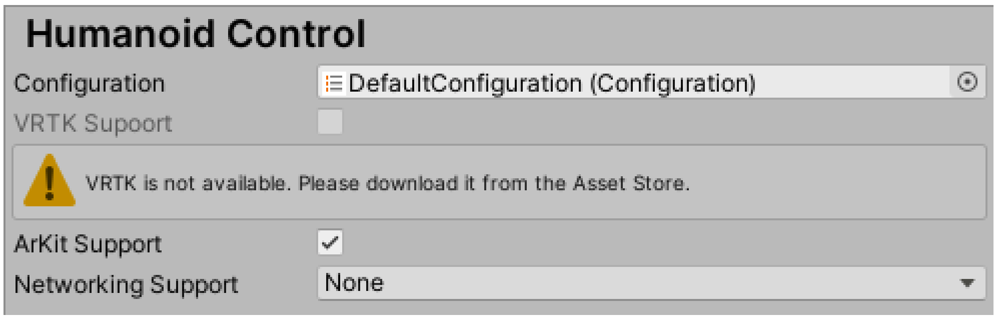
AR Foundation

ARKit Face Tracking



AR Foundation	2.1.8	✓
ARKit Face Tracking	1.0.7	✓
Unity UI	1.0.0	✓
XR Plugin Management	3.2.15	✓

Then the ARKit extension will be enabled automatically:


Humanoid Control

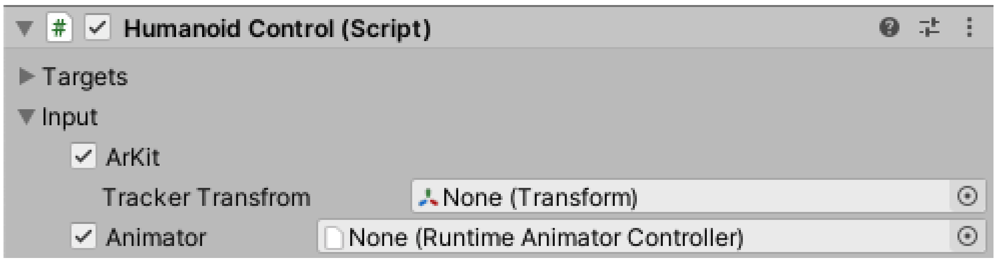
Configuration	DefaultConfiguration (Configuration)
VRTK Supoort	

VRTK is not available. Please download it from the Asset Store.

ArKit Support	✓
Networking Support	None

Configuration

To enable ARKit face tracking for an avatar *ARKit* needs to be enabled in the *Humanoid Control* component.

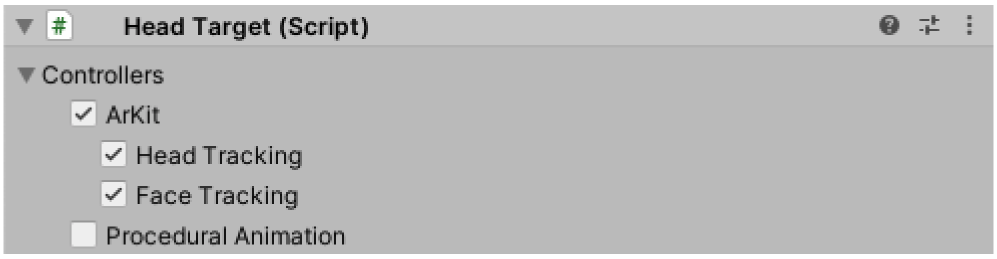

Humanoid Control (Script)

- Targets
- Input
  - ✓ ArKit
    - Tracker Transfrom: None (Transform)
  - ✓ Animator: None (Runtime Animator Controller)

The *Tracker Transform* will be a reference to the Transform in the scene representing the ARKit tracker at runtime.

Head Target

On the *Head Target* component it is possible to enable or disable the Head and/or Face Tracking for the avatar:

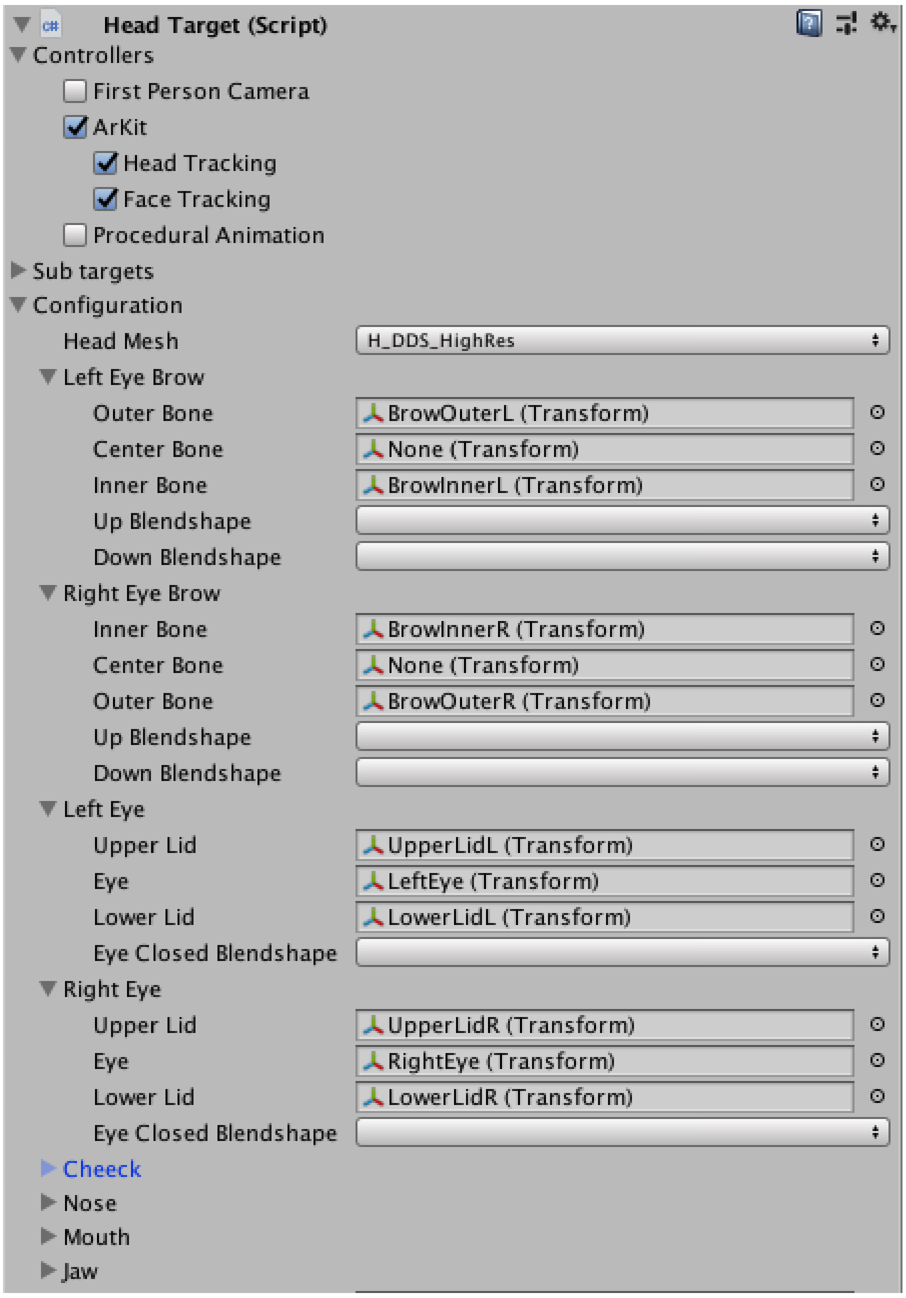

Head Target (Script)

- Controllers
  - ✓ ArKit
    - ✓ Head Tracking
    - ✓ Face Tracking
  - Procedural Animation

Facial Bones Configuration

For the face tracking to work well you may need to configure the face in the Head Target. The best results are currently achieved with facial bones because they have a more clear definition for how the face will move when they are adjusted.

We try to detect facial bones automatically, but it is good to check and adjust these bones where necessary. The facial bones can be found in the *Configuration* section of the Head Target:



Head Target (Script)

Controllers

- [ ] First Person Camera
- [x] ArKit
  - [x] Head Tracking
  - [x] Face Tracking
- [ ] Procedural Animation

Sub targets

Configuration

Head Mesh	H_DDS_HighRes
Left Eye Brow	
Outer Bone	BrowOuterL (Transform)
Center Bone	None (Transform)
Inner Bone	BrowInnerL (Transform)
Up Blendshape	
Down Blendshape	
Right Eye Brow	
Inner Bone	BrowInnerR (Transform)
Center Bone	None (Transform)
Outer Bone	BrowOuterR (Transform)
Up Blendshape	
Down Blendshape	
Left Eye	
Upper Lid	UpperLidL (Transform)
Eye	LeftEye (Transform)
Lower Lid	LowerLidL (Transform)
Eye Closed Blendshape	
Right Eye	
Upper Lid	UpperLidR (Transform)
Eye	RightEye (Transform)
Lower Lid	LowerLidR (Transform)
Eye Closed Blendshape	

Cheeck

Nose

Mouth

Jaw

The following bones are used for facial tracking with ARKit:

Left Eye Brow, Outer Bone

Left Eye Brow, Center Bone

Left Eye Bron, Inner Bone

Right Eye Brow, Inner Bone

Right Eye Brow, Center Bone

Right Eye Brow, Outer Bone

Left Eye, Upper Lid

Right Eye, Upper Lid

Mouth, Upper Lip Left

Mouth, Upper Lip

Mouth, Upper Lip Right

Mouth, Left Lip Corner

Mouth, Right Lip Corner

Mouth, Lower Lip Left

Mouth, Lower Lip

Mouth, Lower Lip Right

Jaw

Blendshapes Configuration

Alternatively, you can use blendshapes to drive the facial expressions. Blendshapes need currently always be selected manually. They can be found in the same *Configuration* section of the *Head Target* as the facial bones.

The following blendshapes are used by facial tracking for ARKit:

Left Eye Brow, Up Blendshape

Left Eye Brow, Down Blendshape

Right Eye Brow, Up Blendshape

Right Eye Brow, Down Blendshape

Left Eye, Eye Closed Blendshape

Right Eye, Eye Closed Blendshape

Mouth, Raise Left Blendshape

Mouth, Raise Right Blendshape

Mouth, Lower Left Blendshape

Mouth, Lower Right Blendshape

Mouth, Narrow Left Blendshape

Mouth, Stretch Left Blendshape

Mouth, Narrow Right Blendshape

Mouth, Stretch Right Blendshape

Razer Hydra (Pro)

The Razer Hydra offers high precision hand tracking with additional input buttons and sticks.

Prerequisites

Razer Hydra is supported in the Humanoid Control Pro package.

Hardware

Razer Hydra hardware is supported.

Operating System

Razer Hydra is only supported on Microsoft Windows.

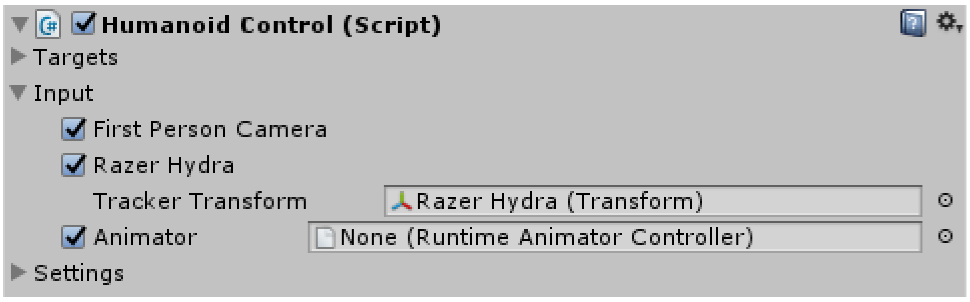
Setup

Razer Hydra support needs to be enabled in the *Edit Menu->Preferences->Humanoid->Hydra Support.*

Disabling *Hydra Support* ensures that no code related to the Razer Hydra is included in the build.

Targets

To enable hand tracking with the Razer Hydra, *Razer Hydra* needs to be enabled in the Humanoid Control component.



Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Razer Hydra
Tracker Transform: Razer Hydra (Transform)
✓ Animator: None (Runtime Animator Controller)
Settings

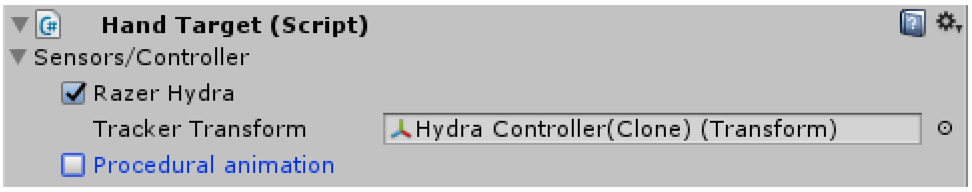
The *Razer Hydra (Transform)* is a reference to the Transform in the scene representing the Razer Hydra Basestation. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Razer Hydra (Transform)* can be used to set the position of the tracking relative to the player in the scene.

Hand Targets

The *Razer Hydra* controller needs to be enabled on the Hand Target to enable tracking.

Hydra controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to get the controllers to the right position in the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.



Hand Target (Script)
Sensors/Controller
✓ Razer Hydra
Tracker Transform: Hydra Controller(Clone) (Transform)
☐ Procedural animation

Controller Input

The buttons of the Hydra controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Hand

Joystick Movements	controller.left.stickHorizontal/stickVertical
Joystick Press	controller.left.stickButton

Trigger	controller.left.trigger1
Bumper	controller.left.trigger2
Button 1	controller.left.button[0]
Button 2	controller.left.button[1]
Button 3	controller.left.button[2]
Button 4	controller.left.button[3]
Option	controller.left.option

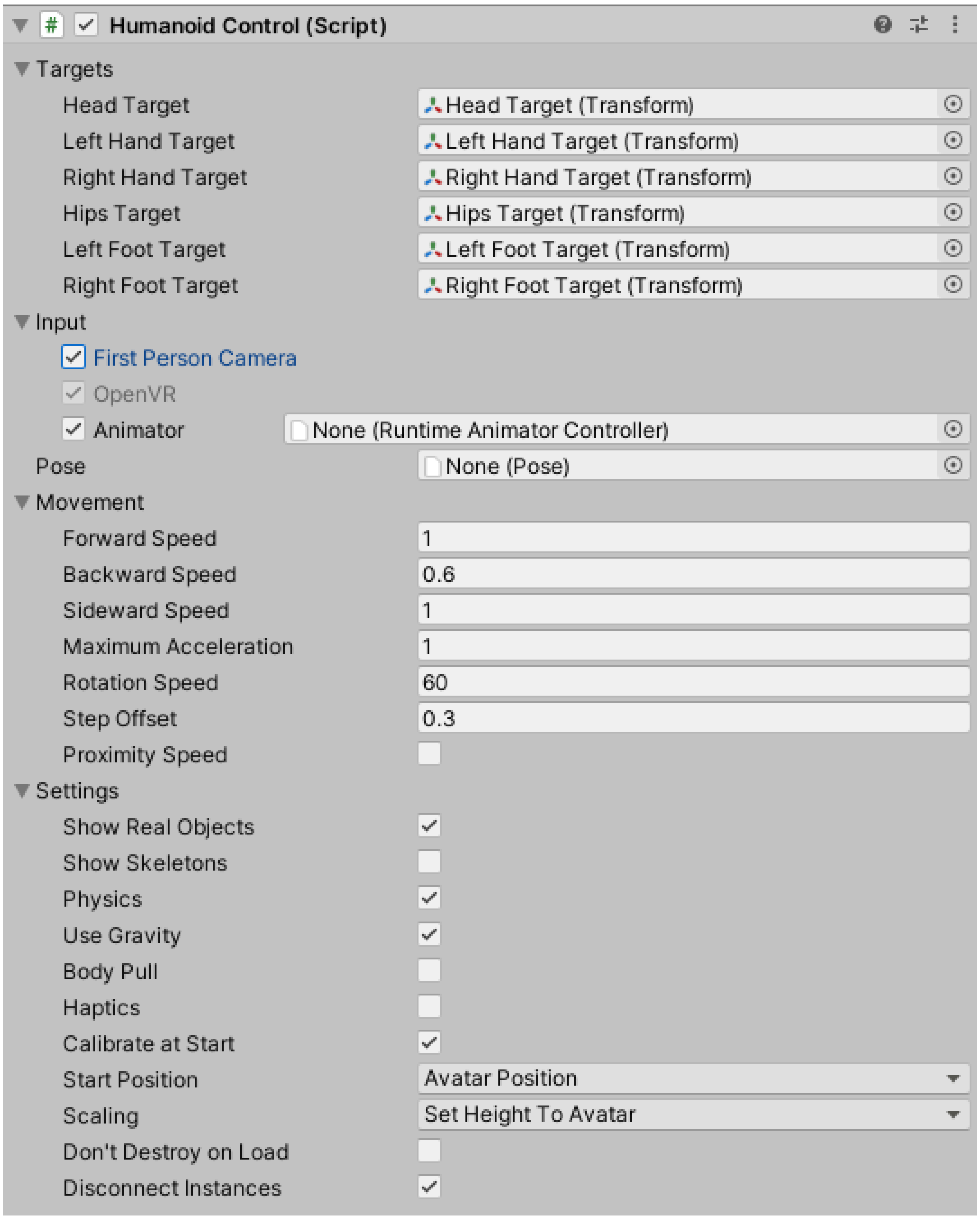
Right Hand

Joystick Movements	controller.right.stickHorizontal/stickVertical
Joystick Press	controller.right.stickButton
Trigger	controller.right.trigger1
Bumper	controller.right.trigger2
Button 1	controller.right.button[0]
Button 2	controller.right.button[1]
Button 3	controller.right.button[2]
Button 4	controller.right.button[3]
Option	controller.right.option

Components

Humanoid Control

The Humanoid Control component has all the options to control your avatar in a simple way. This document describes the available settings.



Humanoid Control (Script)

Targets

Head Target	Head Target (Transform)
Left Hand Target	Left Hand Target (Transform)
Right Hand Target	Right Hand Target (Transform)
Hips Target	Hips Target (Transform)
Left Foot Target	Left Foot Target (Transform)
Right Foot Target	Right Foot Target (Transform)

Input

- ✓ First Person Camera
- ✓ OpenVR
- ✓ Animator: None (Runtime Animator Controller)

Pose: None (Pose)

Movement

Forward Speed	1
Backward Speed	0.6
Sideward Speed	1
Maximum Acceleration	1
Rotation Speed	60
Step Offset	0.3
Proximity Speed	☐

Settings

Show Real Objects	✓
Show Skeletons	☐
Physics	✓
Use Gravity	✓
Body Pull	☐
Haptics	☐
Calibrate at Start	✓
Start Position	Avatar Position
Scaling	Set Height To Avatar
Don't Destroy on Load	☐
Disconnect Instances	✓

Virtual Reality

Headsets like HTC Vive and Oculus Rift are supported when Unity is configured for their support.

In Unity 2019.3 and newer, the *XR Plug-in Management* needs to be enabled in the Project settings. See for more details in the Unity XR page.

In Unity 2019.4 and older, *Virtual Reality Supported* should be enabled in the Player Settings-XR Settings. If *Virtual Reality Supported* is deselected, a message will appear in the top of the script to remind you that the virtual reality is switched off.

Targets

Targets are used to move the body. The control of an avatar is split into six body parts: head, 2 arms, torso and 2 legs. Each body part is controlled by a target: *Head, Left/Right Hand, Hips* and *Left/Right*

Foot. Targets are not shown in the hierarchy by default but can be made visible by clicking on the *Show* button for that target.

Instead of the normal targets, you can also use custom target by replacing the default targets with references to other transforms. A good example are hands which are connected to a steering wheel. In this example two empty *GameObjects* are set at the right locations of the steering wheel. The Left and Right Hand Target show then point to the *Transforms* of these empty *GameObjects*.

Every target will have *Target* script attached which give additional control over these targets:

- Head Target
- Hand Target
- Hips Target
- Foot Target

Input

Enables you to choose which tracker devices are supported. Any combination of trackers can be chosen, even when the device itself is not present. Only when the devices are actually present in the system they will be used in the tracking. During game play you can see in the inspector which Input devices are actually present and tracking at any point.

Humanoid Control will combine multiple trackers using sensor fusion to achieve the best possible tracking result.

First Person Camera

Adds a camera at the eye position of the avatar. If virtual reality has been enabled, this will also enable head tracking for VR headsets.

Animator

Procedural and Mecanim animations are supported for body parts not being tracked by an input device. Mecanim animation is used when an appropriate Animation Controller is selected as parameter. Built-in procedural animation will be used when no Animation Controller is selected. See also Animations.

Pose

This will set the base pose of the avatar. For more information see Pose.

Networking

Remote Avatar

For networking setups, a remote avatar has to be selected which is used for the representation of the avatar on remote clients. This enables you to optimise the avatar mesh between first and third person views of the same avatar.

Movement

This sets a number of parameters for the locomotion of the avatar:

Forward Speed	The maximum forward speed in units/sec.
Backward Speed	The maximum backward speed in units/sec.
Sideward Speed	The maximum speed while strafing in units/sec.

Maximum acceleration	The maximum acceleration allowed when changing speed in units/sec./sec When set to 0, no acceleration limit is used.
Rotation Speed	The maximum rotation speed along the Y axis in degrees/sec.
Step Offset	The maximum ground height different which the humanoid can pass.
Proximity Speed	This will decrease the movement speed of the humanoid when being close to static objects.

Settings

Show Real objects	Shows the tracking devices in the scene. See also The Real World.
Show Skeletons	Shows skeletons for optical hand and/or body tracking systems.
Physics	Enables hand physics and collisions during walking. See also Full Physics.
Use Gravity	When this is enabled the avatar will fall down when there is no object underneath its feet.
Body Pull	When enabled, you can pull the humanoid around while holding static objects with handles.
Haptics	Will use haptic feedback on supported devices when the hands are colliding or touching objects.
Calibrate at Start	Will initiate a scaling calibration when the tracking of the player starts.
Start Position	*Avatar Position*: the user will start at the position of the avatar in the scene. *Player Position*: the user will start at the location within the physical tracking space.
Scaling	*None*: no scaling is done. *Set Height To Avatar*: adjusts the vertical tracking to match the avatar. *Scale Tracking to Avatar*: scales the tracking input to match the size of the avatar. *Scale Avatar to Tracking*: resizes the avatar to match the player size.
Don't Destroy on Load	This option will make sure that the humanoid is not destroyed when the scene is changed.
Disconnect Instances	(Prefabs only) Will disconnect the instance from the prefab when they are included in the scene.

The Real World

An important part of the Humanoid Control component is called *Real World*. All objects inside this GameObject are representations of real-world objects. They can be made visible using the *Show Real Objects* setting of the Humanoid Control component.

Tracker Objects

One of the most important real world objects are the Tracker Objects. These are things like the Leap Motion or Kinect camera or Oculus Touch controllers.

Some tracker object positions are detected automatically:

- Oculus Sensors
- Lighthouse lasers
- Razer Hydra controllers
- SteamVR controllers
- Oculus Touch controllers
- Headsets like Rift, Vive and GearVR

Other tracker objects positions are only detected automatically in specific cases:

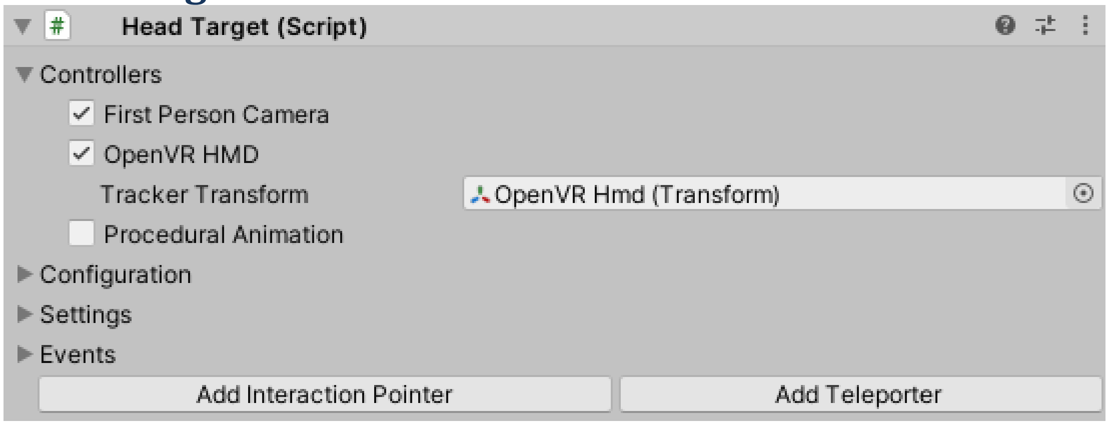
- Kinect camera: when an headset is tracking an Kinect is tracking the head
- Leap Motion camera: when the camera is head mounted

In all other cases it is suggested to place the tracker objects at the right positions manually. You can do this but moving the tracker object like the *Leap Motion Camera* to the location which matches the location in the real world. For instance, my Leap Motion camera is currently placed at my desk at a height of 73 cm from the floor and 55 cm in front of me. The local position of the *Leap Motion Camera* should therefore be set to 0, 0.73, 0.55.

Other real world objects

You are free to add other real world objects yourself to the *Real World*. For instance, you can add a model of the table on which the Leap Motion is placed or add a tripod as a child of the Kinect camera. This will show up in the virtual scene and can help to orientate yourself.

Head Target


Head Target (Script)

Controllers

✓ First Person Camera

✓ OpenVR HMD

Tracker Transform: OpenVR Hmd (Transform)

☐ Procedural Animation

Configuration

Settings

Events

Controllers

Depending on the selected *Inputs* in Humanoid Control, a number of Controllers are available for the Head Target. These can be individually enabled or disabled to suit your needs. For example you can disable head tracking using Kinect while still have body tracking on other parts of the body.

First Person Camera

When this controller is enabled, a camera will be attached to the head of the avatar at eye level. When virtual reality is enabled, the camera will be controlled by the headset.

The location of the camera on the head can be changed in the scene view. The position will be reset to the eyes location when the *First Person Camera* is disabled and enabled again.

The microphone can be used to measure audio energy. See Head Input.

Other controllers

See the list of supported devices to get information on the head target of each device.

Sub Targets (Pro)

Sub targets are used in combination with facial tracking. Depending on the tracking device, additional facial target can be tracked and used.

When the microphone has been enabled, the Audio Energy will show the received volume of sound.

Configuration

Configuration is used in combination with facial tracking.

Expressions (Pro)

In Humanoid Control Pro, facial expressions can be defined and set. For more information see Facial Expressions.

Focus Object (Pro)

This is the object the humanoid is looking at. With eye tracking, this is determined from the detected gaze direction, without eye tracking a raycast from the eyes is used in the forward direction of the head.

Settings

Collision Fader Adds a screen fader which blacks out the camera when the head enters objects.

Events

Tracking Event	Use to call functions based on the tracking status of the headset.
Audio Event	Use to call functions based on the audio level measured with the microphone.
In Collider Event	Use to call functions based on the state of the head being inside colliders.

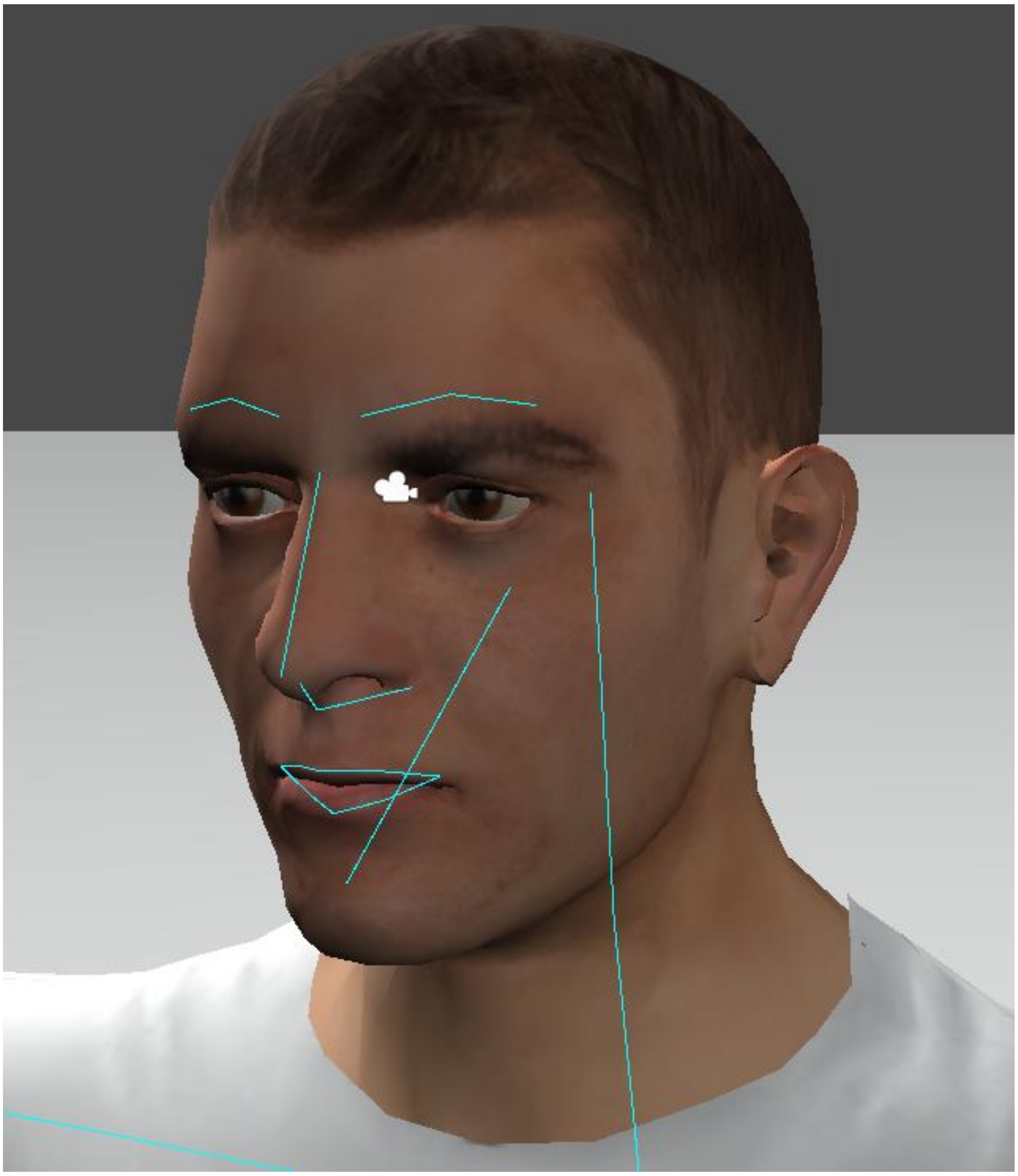
Buttons

- **Add Interaction Pointer** Adds a gaze interaction pointer to the head target. See Interaction Pointer. For more information about interaction see Interaction, Event System and UI.
- **Add Teleporter** Adds a preconfigured gaze interaction pointer to the head target which can teleport the avatar by pointing to new positions. See Teleporter.

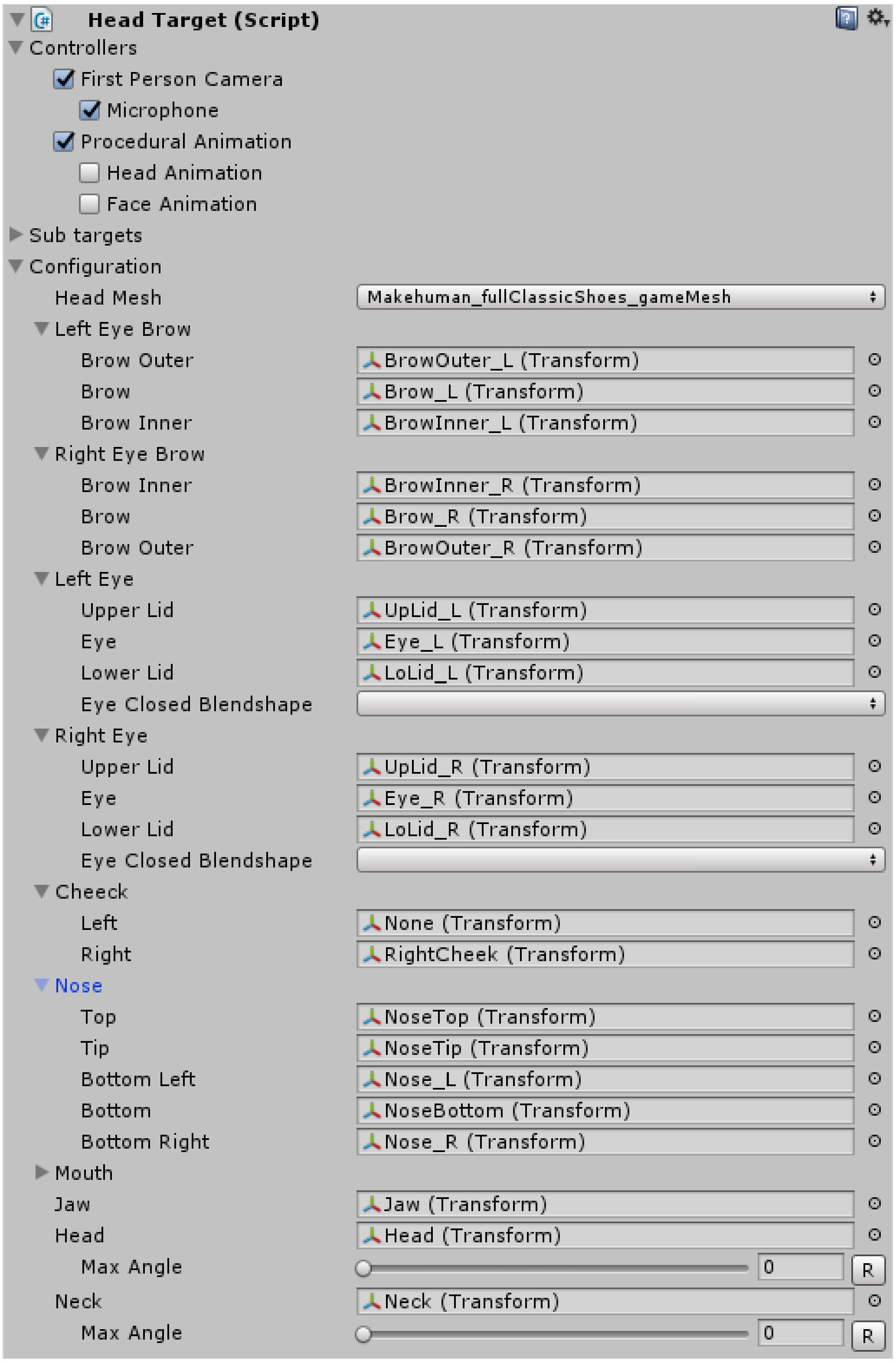
Face Tracking (Pro)

Face Tracking is supported in Humanoid Control Pro for facial bone rigs using Microsoft Kinect 2, Intel RealSense and Tobii Eyetracking (eye tracking only).

Configuration



Humanoid Control tries to recognize supported facial bones automatically. You can check the facial bone configuration in the scene editor. Additionally, facial bones can be checked and edited in the Head Target Configuration section.



Head Target (Script)

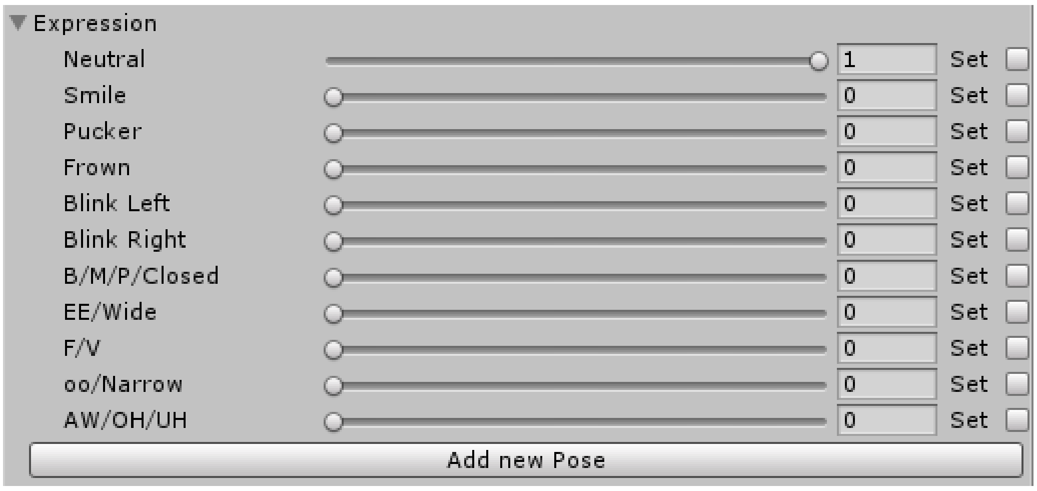
Controllers
- [✓] First Person Camera
  - [✓] Microphone
- [✓] Procedural Animation
  - [ ] Head Animation
  - [ ] Face Animation

Sub targets

Configuration

Head Mesh	Makehuman_fullClassicShoes_gameMesh
Left Eye Brow	
Brow Outer	BrowOuter_L (Transform)
Brow	Brow_L (Transform)
Brow Inner	BrowInner_L (Transform)
Right Eye Brow	
Brow Inner	BrowInner_R (Transform)
Brow	Brow_R (Transform)
Brow Outer	BrowOuter_R (Transform)
Left Eye	
Upper Lid	UpLid_L (Transform)
Eye	Eye_L (Transform)
Lower Lid	LoLid_L (Transform)
Eye Closed Blendshape	
Right Eye	
Upper Lid	UpLid_R (Transform)
Eye	Eye_R (Transform)
Lower Lid	LoLid_R (Transform)
Eye Closed Blendshape	
Cheeck	
Left	None (Transform)
Right	RightCheek (Transform)
Nose	
Top	NoseTop (Transform)
Tip	NoseTip (Transform)
Bottom Left	Nose_L (Transform)
Bottom	NoseBottom (Transform)
Bottom Right	Nose_R (Transform)
Mouth	
Jaw	Jaw (Transform)
Head	Head (Transform)
Max Angle	0
Neck	Neck (Transform)
Max Angle	0

Face Expressions (Pro)



Expression				
Neutral	slider	1	Set	☐
Smile	slider	0	Set	☐
Pucker	slider	0	Set	☐
Frown	slider	0	Set	☐
Blink Left	slider	0	Set	☐
Blink Right	slider	0	Set	☐
B/M/P/Closed	slider	0	Set	☐
EE/Wide	slider	0	Set	☐
F/V	slider	0	Set	☐
oo/Narrow	slider	0	Set	☐
AW/OH/UH	slider	0	Set	☐

Face expressions are preconfigured configurations of the facial bones which can be used to show and mix realistic facial expressions.

Humanoid Control provides 11 preconfigured expressions. More expressions can be added using the *Add New Pose* button.

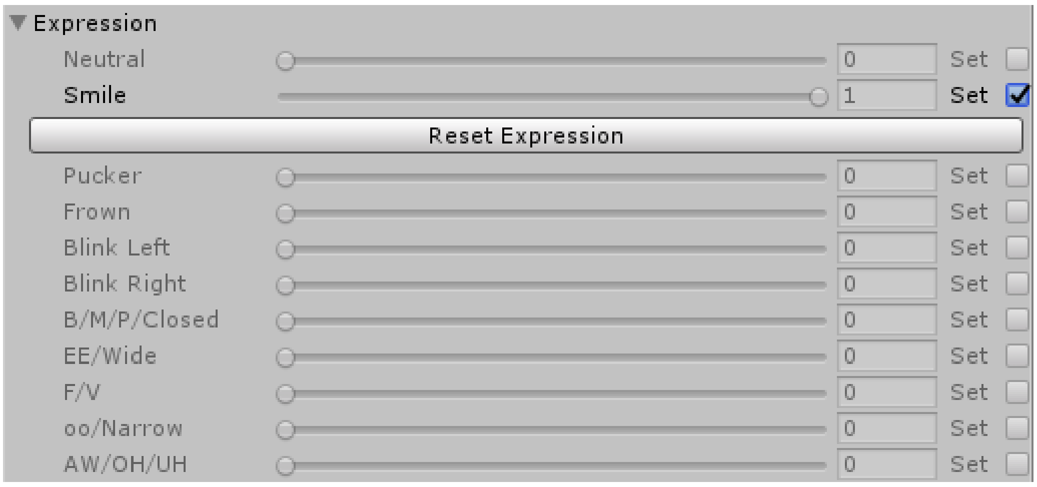
Posing

Expressions can be selected and mixed using the sliders or by using the function *SetExpression(HandPose pose, float weight)*. All expressions will be mixed using averaging. The total of all hand poses weight will always be 1. Note that this differs from blend shapes which are combined in an additive way.

If a new face expression is set with a weight smaller than 1, the weight for the old expression combination is decreased by ratio such that the total weight is 1 again.

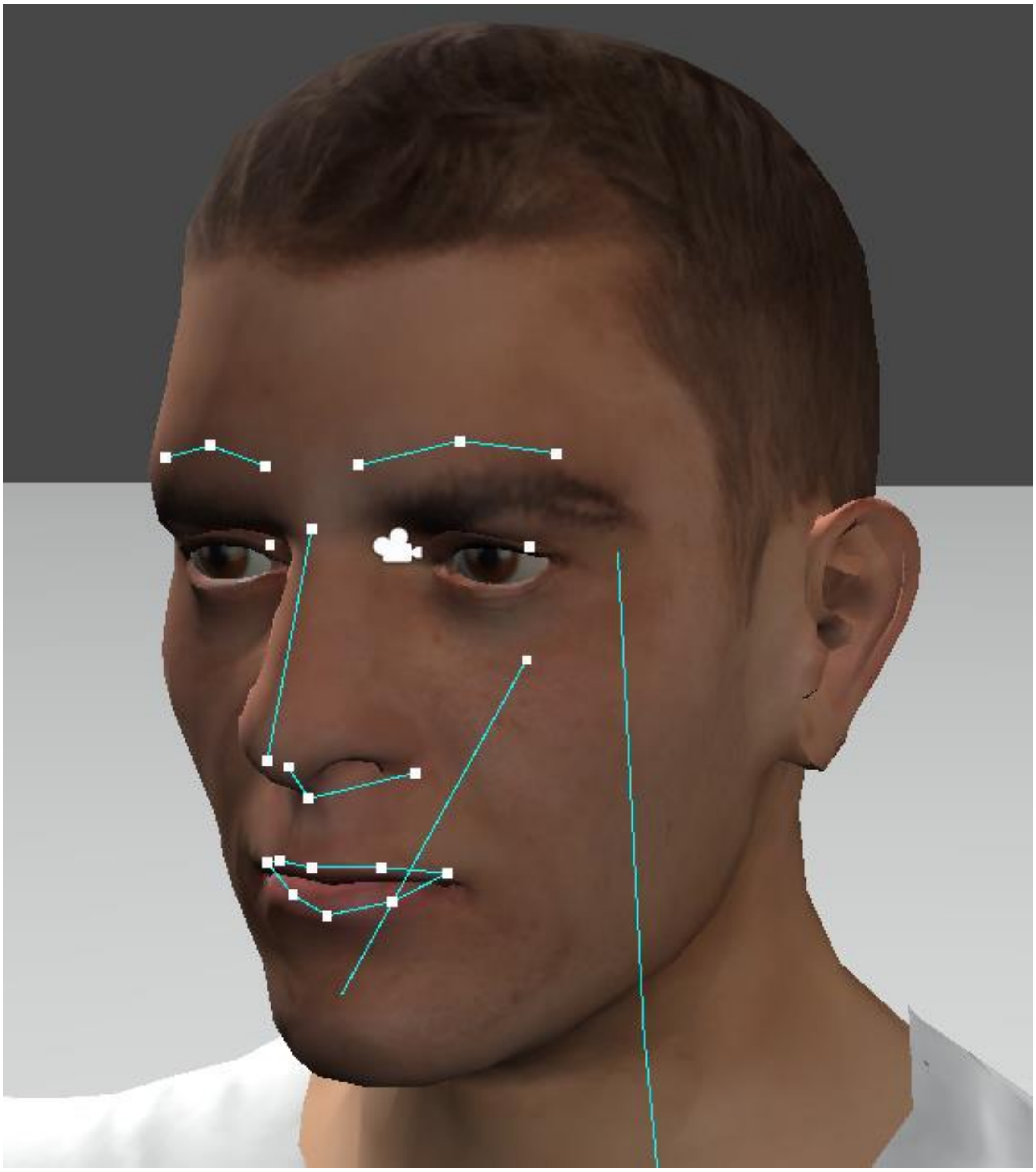
Editing expressions

Face expressions can be edited in the scene view by enabling *Set* for an expression.

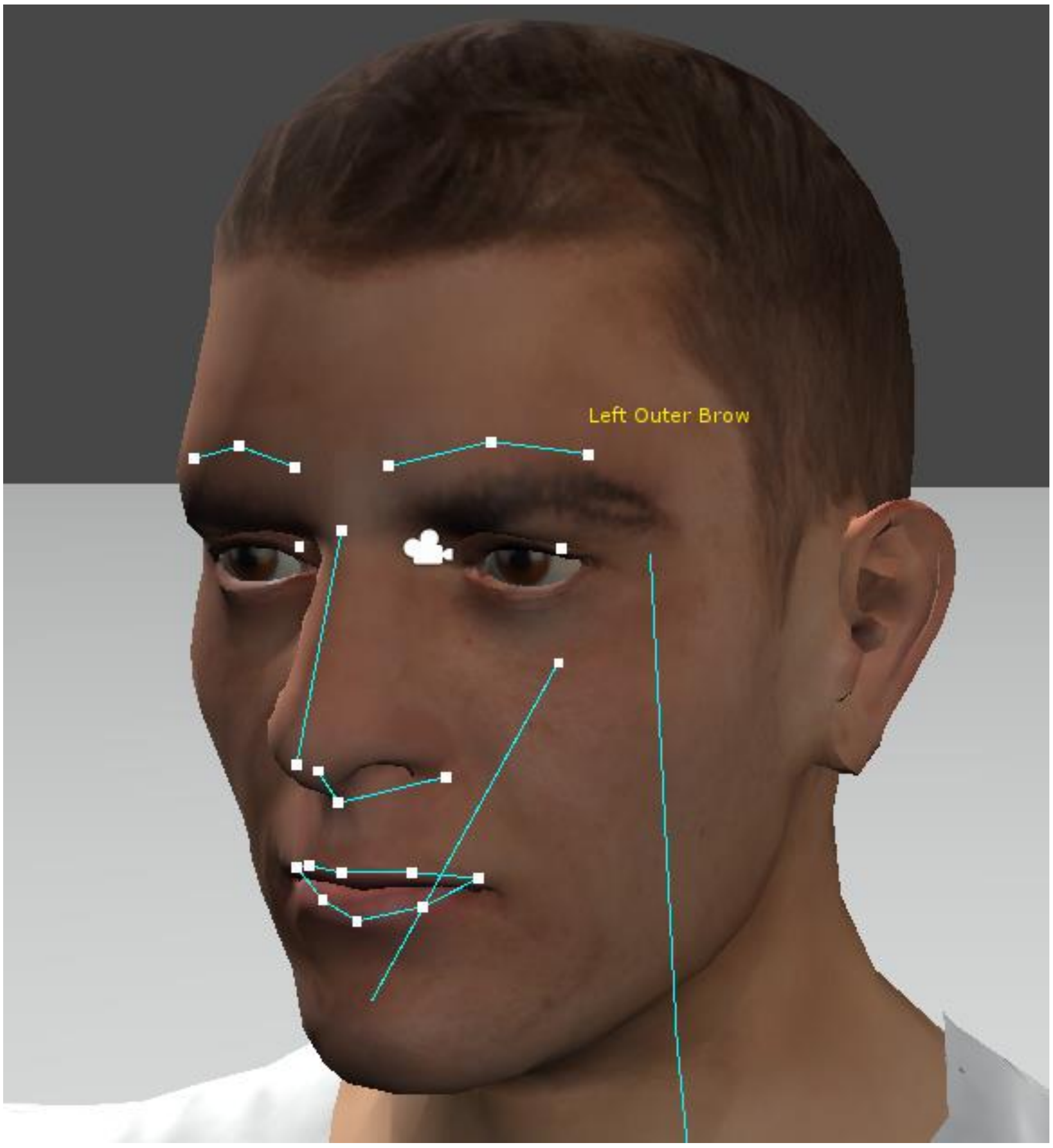


Expression				
Neutral	slider	0	Set	☐
Smile	slider	1	Set	✓
Pucker	slider	0	Set	☐
Frown	slider	0	Set	☐
Blink Left	slider	0	Set	☐
Blink Right	slider	0	Set	☐
B/M/P/Closed	slider	0	Set	☐
EE/Wide	slider	0	Set	☐
F/V	slider	0	Set	☐
oo/Narrow	slider	0	Set	☐
AW/OH/UH	slider	0	Set	☐

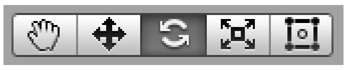
Reset Expression can be used to reset all facial bones to the predefined pose or the neutral expression for custom expressions.

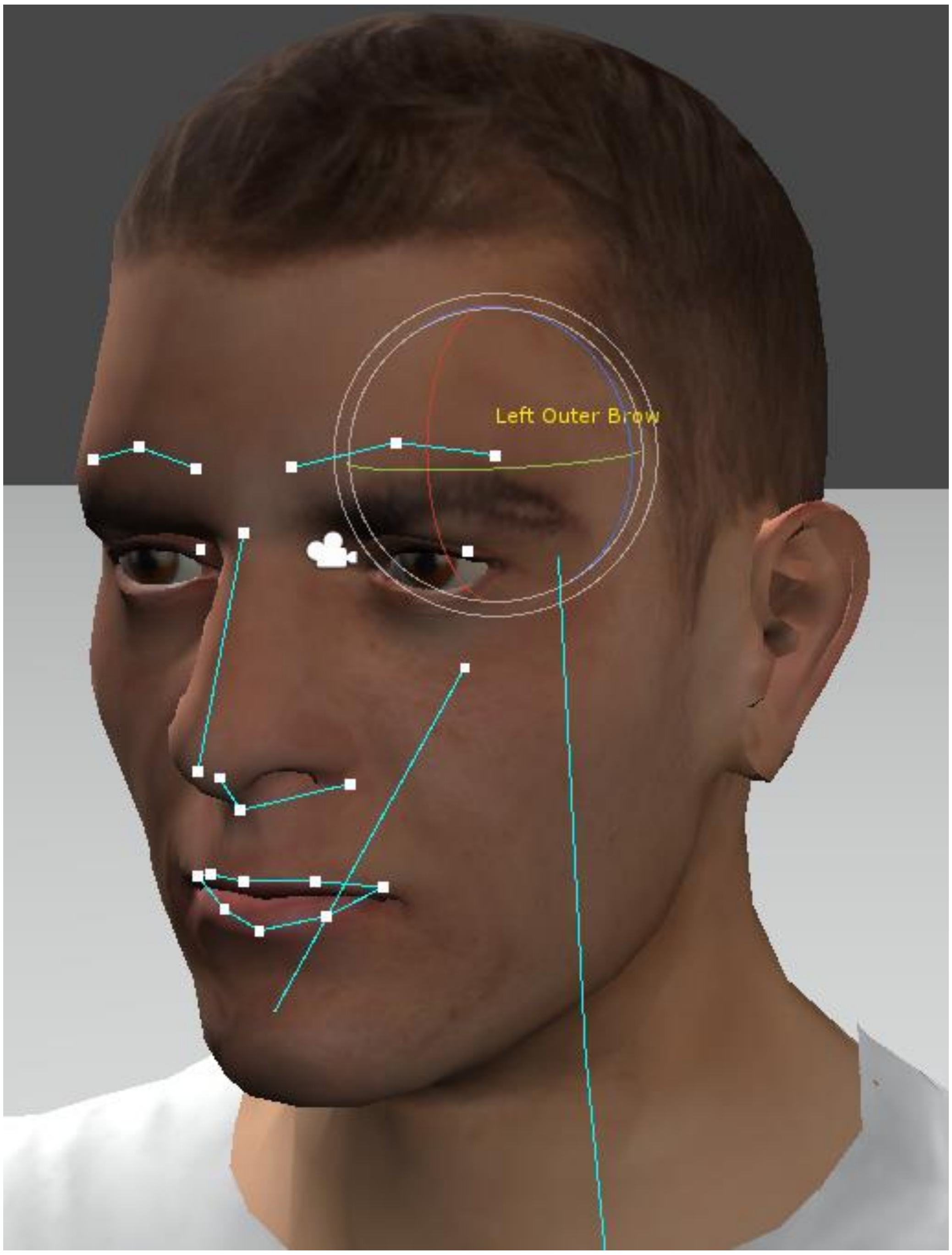


In the scene view you will see an handle for each bone which can be used to set it to the desired pose. A handle can be selected by clicking on it. This will show the name of the bone and will enable you to change its position.



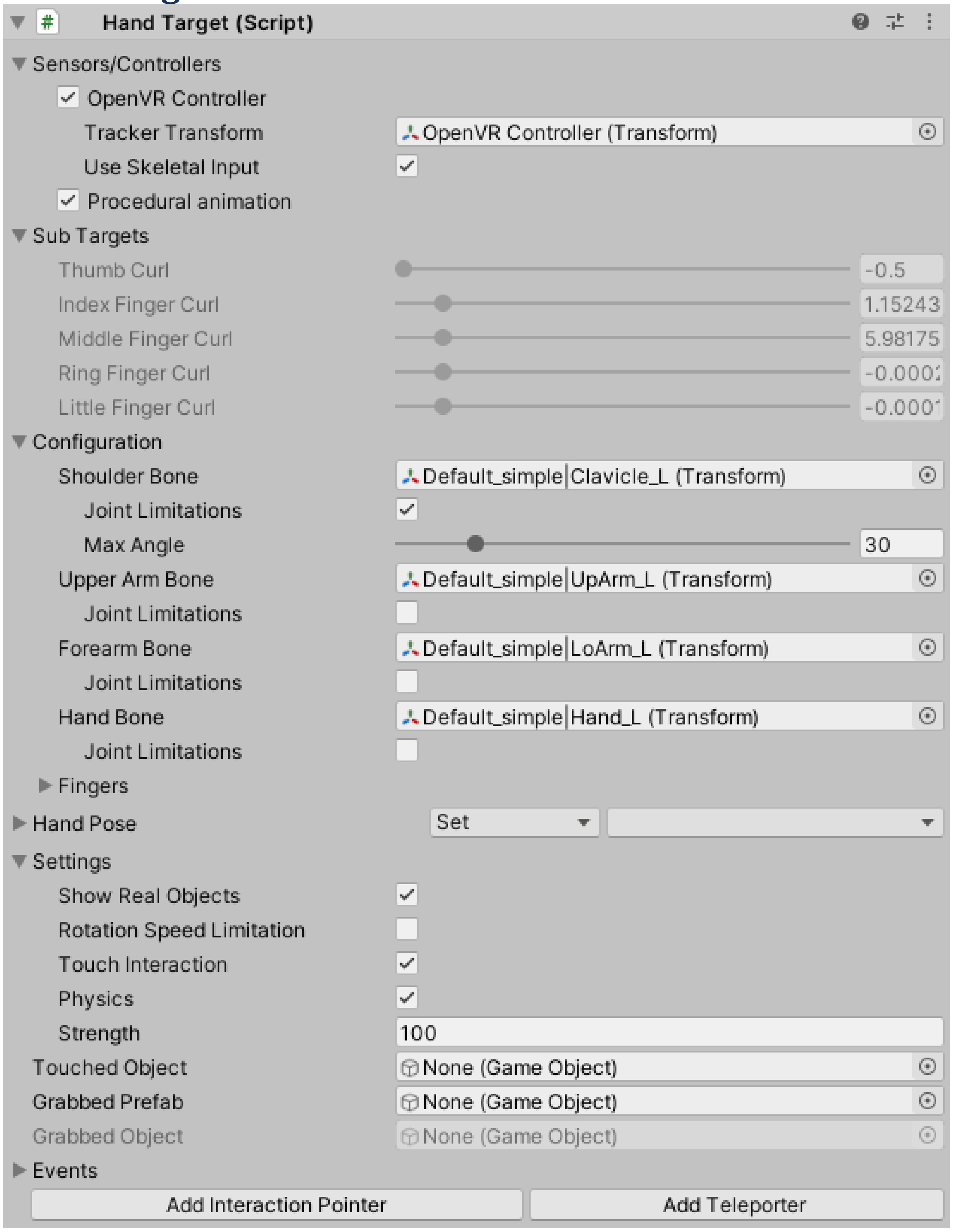
Additionally, you can change the rotation or scale for each bone by selecting the Rotation or Scale tool from the Unity toolbar.





When *Set* is deselected, the altered expression is stored and can be used.

Hand Target



Hand Target (Script)

Sensors/Controllers

✓ OpenVR Controller

Tracker Transform	OpenVR Controller (Transform)
Use Skeletal Input	✓

✓ Procedural animation

Sub Targets

Thumb Curl	-0.5
Index Finger Curl	1.15243
Middle Finger Curl	5.98175
Ring Finger Curl	-0.000:
Little Finger Curl	-0.000'

Configuration

Shoulder Bone	Default_simple|Clavicle_L (Transform)
Joint Limitations	✓
Max Angle	30
Upper Arm Bone	Default_simple|UpArm_L (Transform)
Joint Limitations	
Forearm Bone	Default_simple|LoArm_L (Transform)
Joint Limitations	
Hand Bone	Default_simple|Hand_L (Transform)
Joint Limitations	

Fingers

Hand Pose	Set

Settings

Show Real Objects	✓
Rotation Speed Limitation	
Touch Interaction	✓
Physics	✓
Strength	100
Touched Object	None (Game Object)
Grabbed Prefab	None (Game Object)
Grabbed Object	None (Game Object)

Events

Sensors/Controllers

See the list of supported devices to get information on the hand target of each device.

Sub Targets

Thumb and Finger Curl

The current curl value of the thumb and/or fingers is found here. It is possible to control the curl value of a finger of thumb directly using the slider during play if it is not directly controlled by input or hand poses. If it is controlled by input, the current curl value can be seen here.

Configuration

Bones

For Mecanim compatible avatars, the correct bones are detected automatically. For other avatars, the correct bone Transforms can be assigned manually using the *Bone* parameters.

It is also possible to override the default bones from Mecanim with your own choice by manual assignment of the bone. To return to the default bone, it is sufficient to clear the applicable *Bone* parameter.

For the thumb and finger bones, only the first, proximal bone is needed, other child bones will be detected automatically.

Limits

For the arm bones, it is possible to configure the limits of movement. The maximum angle can be set when *Joint Limitations* is enabled.

Hand Pose

Shows the currently detected hand pose.

More information on hand poses can be found here.

Settings

Show Real Objects	Will show controller models of input devices where applicable.
Rotation Speed Limitation	Limits the maximum rotation speed of the joints. This can result in more natural movements with optical tracking solutions.
Touch Interaction	Enables touch interaction with the environment. See Interaction.
Physics	Enables physics for this hand. See Physics.
Strength	Determines the strength of the hand when using physics.

Events

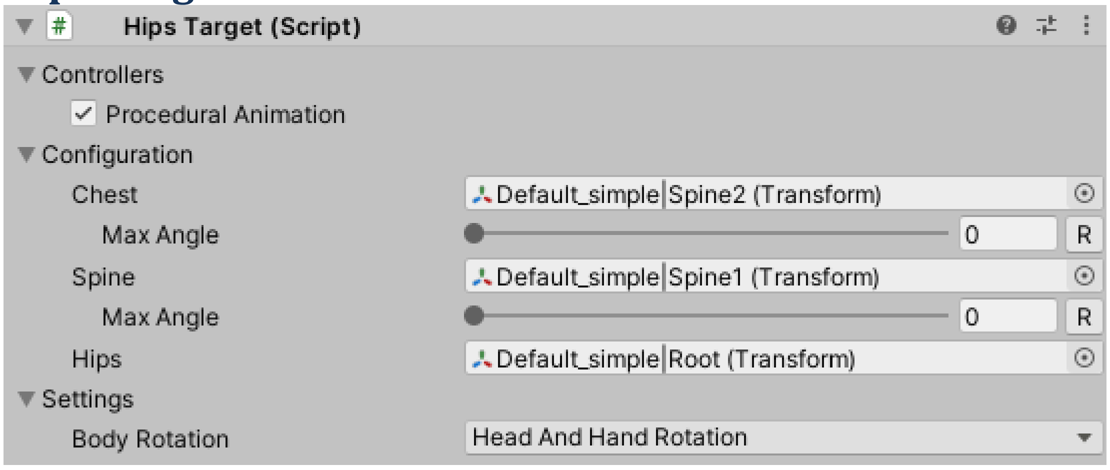
Pose Event	Use to call functions based on the defined poses of the hand.
Touch Event	Use to call functions based on the hand touching objects

Grab Event	Use to call functions based on the hand grabbing objects

Tools

- **Interaction Pointer** Adds a interaction pointer to the hand target. For more information about interaction see Interaction, Event System and UI.
- **Teleporter** Adds a preconfigured interaction pointer to the hand target which can teleport the avatar by pointing to new positions.

Hips Target


Hips Target (Script)

Controllers

✓ Procedural Animation

Configuration

Chest	Default_simple|Spine2 (Transform)
Max Angle	0
Spine	Default_simple|Spine1 (Transform)
Max Angle	0
Hips	Default_simple|Root (Transform)

Settings

Body Rotation	Head And Hand Rotation

Controllers

See the list of supported devices to get information on the hand target of each device.

Configuration

Bones

For Mecanim compatible avatars, the correct bones are detected automatically. For other avatars, the correct bone Transforms can be assigned manually using the *Bone* parameters.

It is also possible to override the default bones from Mecanim with your own choice by manual assignment of the bone. To return to the default bone, it is sufficient to clear the applicable *Bone* parameter.

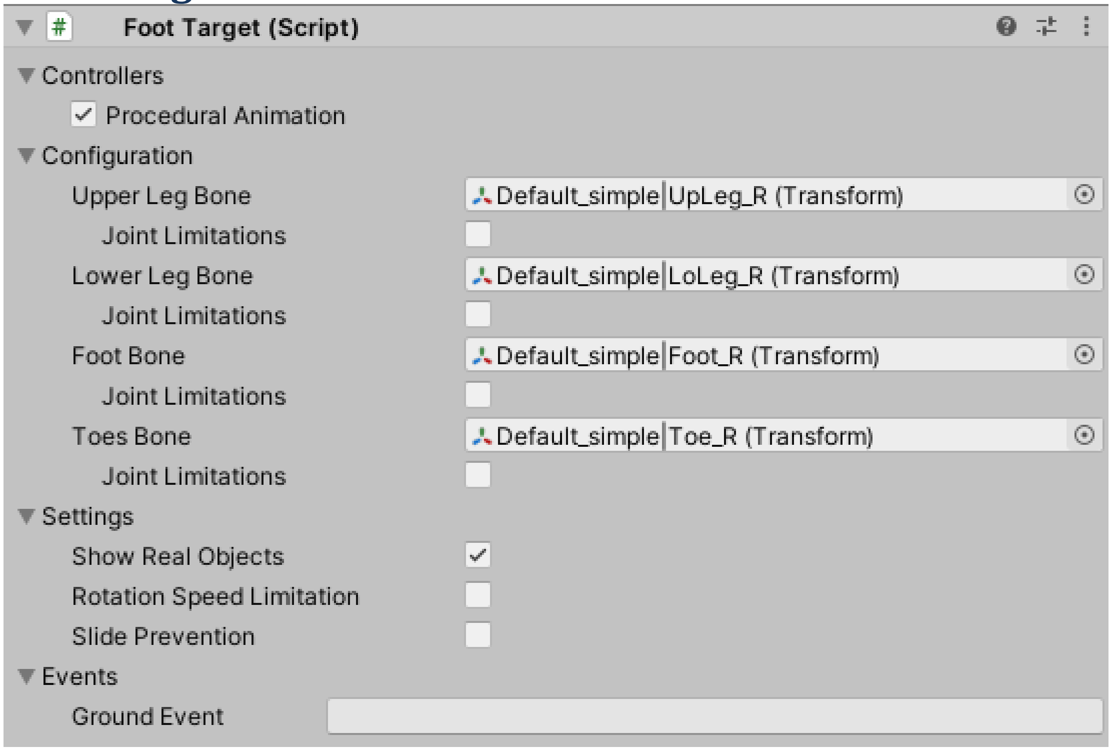
Limits

For the spine and chest bones, it is possible to configure the limits of movement. The maximum angle can be set when *Joint Limitations* is enabled.

Settings

Body Rotation	Selected the way the hips rotation is determined – *Head Rotation*: The hips rotation will just use the head rotation. – *Head and Hand Rotation* (default): The hips rotation will be based on the poses of the head and hands together. – *No Rotation*: The hips will not rotate, but will only follow the root Transform's rotation.

Foot Target


Foot Target (Script)

Controllers

✓ Procedural Animation

Configuration

Upper Leg Bone	Default_simple|UpLeg_R (Transform)
Joint Limitations	☐
Lower Leg Bone	Default_simple|LoLeg_R (Transform)
Joint Limitations	☐
Foot Bone	Default_simple|Foot_R (Transform)
Joint Limitations	☐
Toes Bone	Default_simple|Toe_R (Transform)
Joint Limitations	☐

Settings

Show Real Objects	✓
Rotation Speed Limitation	☐
Slide Prevention	☐

Events

Ground Event

Controllers

See the list of supported devices to get information on the hand target of each device.

Configuration

Bones

For Mecanim compatible avatars, the correct bones are detected automatically. For other avatars, the correct bone Transforms have to be assigned manually using the *Bone* parameters.

It is also possible to override the default bones from Mecanim with your own choice by manual assignment of the bone. To return to the default bone, it is sufficient to clear the applicable *Bone* parameter.

Limits

It is possible to configure the limits of movement. These values are used when *Joint Limitations* are enabled.

The range of movement of each joint is shown in the scene when the applicable joint is selected.

Settings

Show Real Objects	Will show controller models of input devices where applicable.
Rotation Speed Limitation	Limits the maximum rotation speed of the joints. This can result in more natural movements with optical tracking solutions.
Slide Prevention	Prevents movement of the foot when it is on the ground.

Events

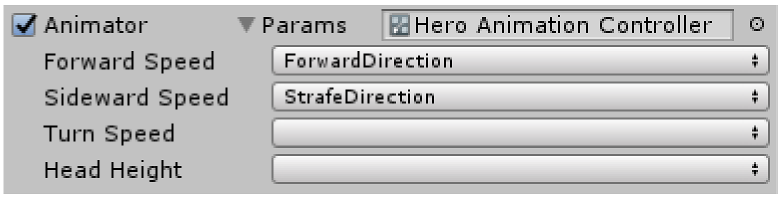
Ground Event Use to call functions based on the foot touching the ground.

Animations

Animations can be used when other trackers are not tracking or not available. They can be enabled using the *Animator* option in the *Input* section of the *Humanoid Control* script.

By default, the animator uses a builtin procedural animation. This is overridden when setting the *Runtime Animator Controller* parameter which is standard Unity *Animator Controller*.

Humanoid Control can control the animator controller using a number of *float* Animation Parameters. These can be set in the *Params* section and refer to the Animation Parameters of the Animator Controller.



✓ Animator ▼ Params Hero Animation Controller

Forward Speed	ForwardDirection
Sideward Speed	StrafeDirection
Turn Speed	
Head Height	

float **Forward Speed**	The forward/backward speed. Negative is backward walking. Unit is units(meters) per second.
float **Sideward Speed**	The left/right speed. Negative is left strafing. Unit is units(meters) per second
float **Turn Speed**	The turning speed around the Y axis. Negative is turning left. Unit is full rotations (360 degrees) per second.
float **Head Height**	The head height relative to the standing position. Negative is crouching. Positive is reaching up. Unit is meters.

Please note the unit of the parameters. Great care should be paid to this when creating the animations, because it is not possible to use root motion (see below).

If the animation root speed does not match the required units of speed, you can adjust them using the *Speed* parameter of the Animation State (see Unity3d - Animation States).
For example, when the turning animation is set up such that it takes 4 seconds for a full rotation, the *Speed* parameter should be set to 0.25.

Root motion

For the animations to work well they should not contain root motion. This is because we need to be able to move the character from the VR headset, not from the animation, or players will get motion sickness.

To get the right behaviour, ensure that the following options are checked on the imported animations:

Loop Time -> Loop Pose
Root Transform Rotation -> Bake Into PoseRoot Transform Position (Y) -> Bake Into Pose
Root Transform Position (XZ) -> Bake Into Pose

Animation restrictions

The animations are controlled by the Animation Parameters which are derived from the head movements. In order to match the movements of the player, the animations should be set in a specific way.

- The animation being player when *Forward Speed* = 1 should ensure that the head speed is exactly 1 unit(meter) per second. If the head speed in the animation is different you may see feet slipping over the ground because the animation is played too fast or too slow.
- The *Forward Speed* should only have effect on the forward/backward (Z-axis) speed of the avatar. If this is not ensured, the animations will be wrong. For example when the head moves to the left when forward speed = 1, the sideward speed will also be activated, resulting in unwanted animations.
- The same is true for the *Sideward Speed* and *Turn Speed*, but then limited to the sideward (X-axis) motion and rotation along the Y axis.
- The Head height is used to change the vertical position of the avatar. A value of -0.3 can result in a crouching with even lower values resulting in a crawling position. Positive values like 0.1 can result in a reaching position when the avatar stands on his toes.
- As with the other parameters, the Head Height parameter should only have effect on the Y position of the head. All other movements must be avoided in this animation.

Targets

When the *Animation* option is enabled on the *Humanoid Control* script, it is possible to select whether procedural animation should be used for each target. If *Procedural Animation* is deselected the animation controlled by the *Runtime Animator Controller* set in the *Humanoid Control* script will be used. If the *Runtime Animator Controller* is not set, no animation will be used.

Controller Input

Setup

The *Controller Input* script can be used to assign functions to various controller and keyboard input events.

Finger Movements	This enables a built-in support for finger movements from the controller buttons.
Game Controller	This selects which controller type is showed. This setting has no effect on the working of the Controller Input. It helps to determine how the actions are assigned to buttons or various supported controllers. Controller Type *Keyboard* is special because that enables you to assign keyboard key presses to function calls instead of controller buttons.

Controllers

You can assign controller buttons to certain functions. Controller input is split into a *left* and *right* side. For some controllers, this corresponds to the left or right controller (e.g. SteamVR or Oculus Touch controllers). For game controllers like the Xbox Controller, this corresponds to the left and right side of the gamepad.

Note that all controllers use the assignment and setting the *Left Vertical* Input for one controller also changes the *Left Vertical* input for all other controllers. Inputs on the same line of the editor of each controller are the same.

Every input is using an Event Handler to define its behaviour.

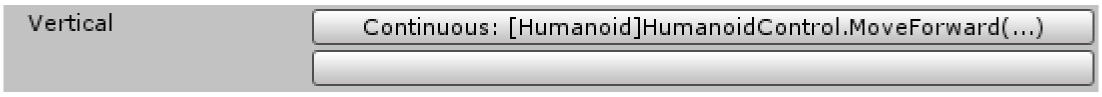
Event Handlers

Event Handling

An Event Handler provides an easy universal way to attach script functions to events and statuses. This is used in many components like Controller Input, Collision Event Handlers, Trigger Event Handlers and Head Target Event Handlers.

Editing

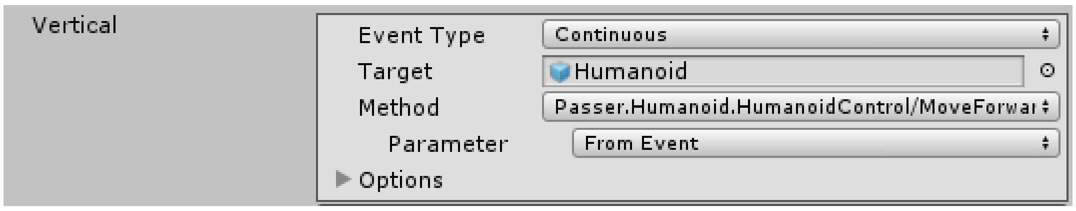
An Event Handler can be edited by pressing on the button. For every event It is possible to attach multiple event Handlers. When an event Handler has been defined, a new empty button will appear below which you can use to add an additional event Handler.



Vertical

Continuous: [Humanoid]HumanoidControl.MoveForward(...)

When an event Handler has been selected, a number of fields become visible:



Vertical

Event Type: Continuous
Target: Humanoid
Method: Passer.Humanoid.HumanoidControl/MoveForwar
Parameter: From Event
Options

Event Type

The labels of this drop down field different depending on the event, but in general the working is as follows:

Never	The function is never called.
On Start	The function is called when the event starts.
On End	The function is called when the event ends.
While Active	The function is called while the event is active.
Wile Inactive	The function is called while the event is not active.
On Change	The function is called when the event starts or ends.
Always	The function is called every frame.

Target

The target GameObject in the scene on which the function is called.

Method

The method to call on the GameObject.
For each component on the GameObject an entry will be listed in the drop down. When selecting a component, the desired function can be chosen from the list of available functions.

Method Parameter

This can be set to a constant value or to *From Event*. In the second option, the parameter value will come from the event itself. Like the status of the button pressed, the GameObject which has entered the trigger collider or the GameObject to which the user is looking.

Options

Depending on the parameter type additional settings can be set in the *Options* section.

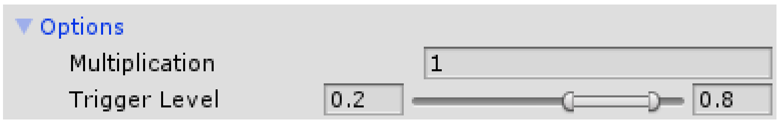
Boolean Parameter

- **Inverse** The parameter value will be inverted before it is sent to the function

Integer Parameter

- **Mutliplication** The parameter value will be multiplied by this value before it is sent to the function.

Float parameter



Multiplication 1
Trigger Level 0.2 0.8

- **Multiplication** The parameter value will be multiplied by this value before it is sent to the function
- **Trigger Level** This determines at which values the Event Start and End happen. In the example above, the event will end when the value drops below 0.2 and will start again when the value raises above 0.8.

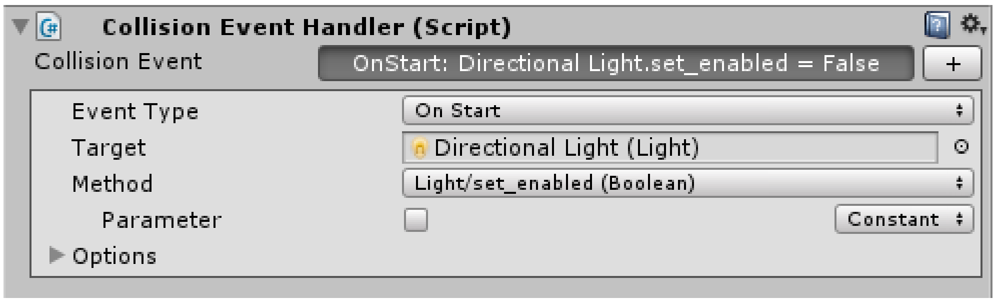
Collision Event Handler

→ Collision Event Handler API

The Collision Event Handler is a convenience component to act on collision events without programming. It is a specific implementation of an Event Handler.

The Event

The Collision Event Handler can be placed on GameObjects and Rigidbodies to catch collision events and execute functions when this happens.



Collision Event Handler (Script)

Collision Event	OnStart: Directional Light.set_enabled = False
Event Type	On Start
Target	Directional Light (Light)
Method	Light/set_enabled (Boolean)
Parameter	☐ Constant
Options	

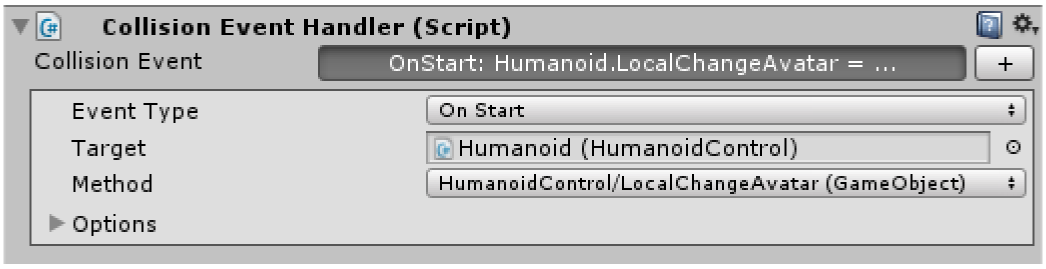
The Event Type works as follows:

- **Never**: The event handler is disabled, the function is never called
- **On Start**: The Target Method is called when the collision starts. This is equivalent to the OnCollisionEnter() message of Unity.
- **On End**: The Target Method is called when the collision ends. This is equivalent to the OnCollisionExit() message of Unity.
- **While Active**: The Target Method is called while the collider is colliding with another collider. In that case it will be called in every frame. This is equivalent to the OnCollisionStay() message of Unity.
- **While Inactive**: The Target Method is called while the collider is not colliding. In that case it will be called in every frame.
- **On Change**: The Target Method is called when the collision starts or ends.
- **Continuous**: The Target Method will always be called in every frame, independent from the collision events.

Target Method Parameters

GameObject

When the Target Method takes a GameObject as parameter , the Target method will receive the GameObject of the collider which is touching this collider.



Collision Event Handler (Script)

Collision Event	OnStart: Humanoid.LocalChangeAvatar = ...
Event Type	On Start
Target	Humanoid (HumanoidControl)
Method	HumanoidControl/LocalChangeAvatar (GameObject)
Options	

Boolean

When the Target Method takes a boolean parameter, a constant value can be used or the parameter can be set to From Event.

When the parameter comes from the event, the boolean value is true when the collider is touching another collider and false when not.

Integer

When the Target Method takes a integer (Int32) parameter, a constant value can be used or the parameter can be set to From Event.

When the parameter comes from the event, the integer value is 1 when the collider is touching another collider and 0 when not.

Float

When the Target Method takes a float (Single) parameter, a constant value can be used or the parameter can be set to From Event.

When the parameters comes from the event, the float value is 1.0 when the collider is touching another collider and 0.0 when not.

Trigger Event Handler

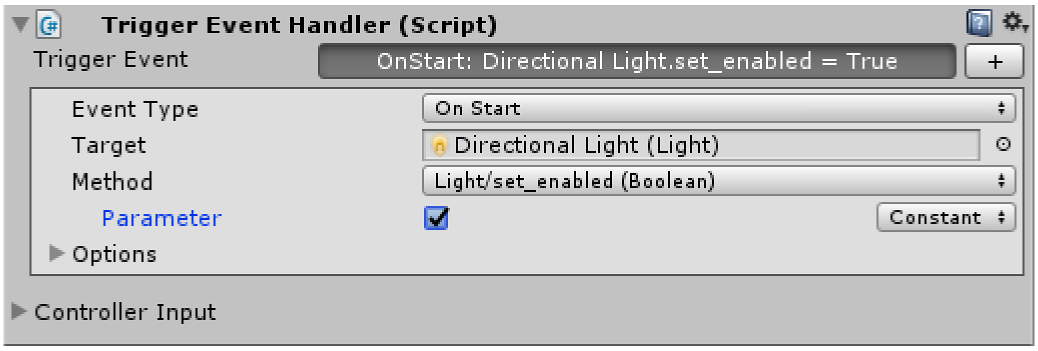
→ Trigger Event Handler API

The Trigger Event Handler is a convenience component to act on trigger events without programmer. It is a specific implementation of an Event Handler.

It differs from the Collision Event Handler in that it uses trigger events and can also change Controller Input.

The Event

The Trigger Event Handler can be placed on GameObjects to catch trigger events and execute functions when this happens.



Trigger Event: OnStart: Directional Light.set_enabled = True
Event Type: On Start
Target: Directional Light (Light)
Method: Light/set_enabled (Boolean)
Parameter: ☑ Constant
Options
Controller Input

The Event Type works as follows:

- **Never**: The event handler is disabled, the function is never called
- **On Start**: The Target Method is called when the trigger collider is entered by a collider. This is equivalent to the OnTriggerEnter() message of Unity.
- **On End**: The Target Method is called when the trigger is exited by a collider. This is equivalent to the OnTriggerExit() message of Unity.
- **While Active**: The Target Method is called while a collider is touching the trigger collider. In that case it will be called in every frame. This is equivalent to the OnTriggerStay() message of Unity.
- **While Inactive**: The Target Method is called while no collider is in the trigger collider. In that case it will be called in every frame.
- **On Change**: The Target Method is called when a collider enters or exits the trigger collider.

- **Continuous**: The Target Method will always be called in every frame, independent from the trigger events.

Target Method Parameters

GameObject

When the Target Method takes a GameObject as parameter, the Target Method will receive the GameObject of the collider which is touching the trigger collider.

Boolean

When the Target Method takes a boolean parameter, a *Constant* value can be used or the parameter can be set to *From Event*.

When the parameter is from the event, the boolean is *true* when the collider is touching the trigger collider and *false* when not.

Integer

When the Target Method takes an integer (Int32) parameter, a *Constant* value can be used or the parameter can be set to *From Event*.

When the parameter is from the event, the integer is 1 when the collider is touching the trigger collider and 0 when not.

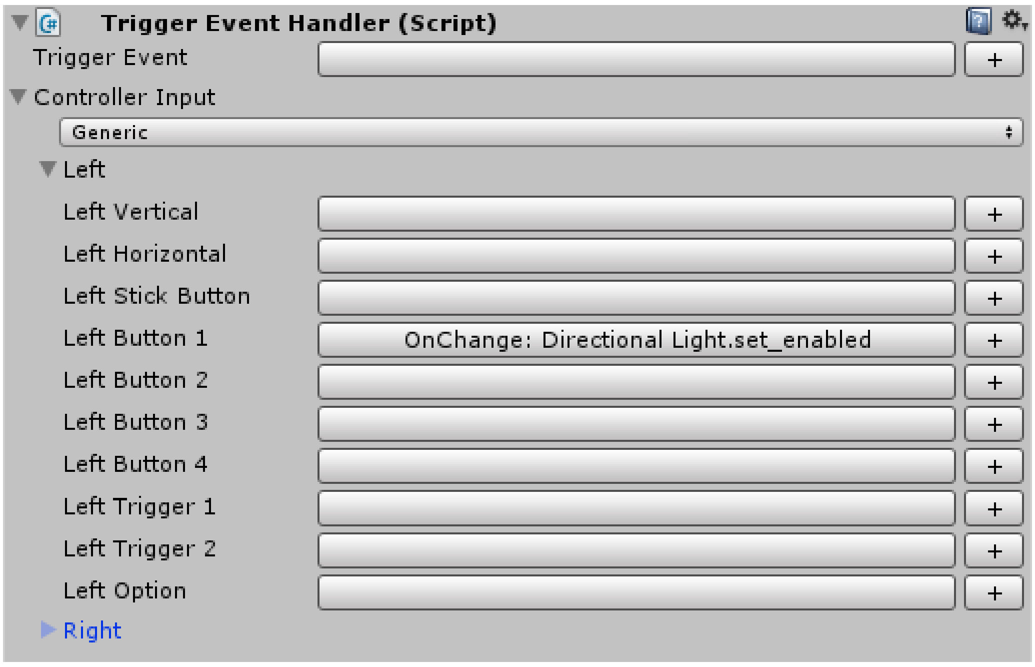
Float

When the Target Method takes a float (Single) parameter, a *Constant* value can be used or the parameter can be set to *From Event*.

When the parameter is from the event, the float value is 1.0 when the collider is touching the trigger collider and 0.0 when not.

Controller Input

If the GameObject entering the trigger collider has Controller Input you can override the Controller Input while the GameObject is in the trigger collider. This enables you to assign a button to open a door when the player is close to that door for example.


Trigger Event Handler (Script)

Trigger Event	
Controller Input	
Generic	
Left	
Left Vertical	
Left Horizontal	
Left Stick Button	
Left Button 1	OnChange: Directional Light.set_enabled
Left Button 2	
Left Button 3	
Left Button 4	
Left Trigger 1	
Left Trigger 2	
Left Option	
Right	

The configuration is similar to the normal Controller Input. The difference is that empty entries do not override the Controller Input configuration. In the example above only the left Button 1 will be overridden, all other input will not be changed.

When the GameObject leaves the trigger collider, the Controller Input is restored to the original configuration.

Interaction, EventSystem and UI

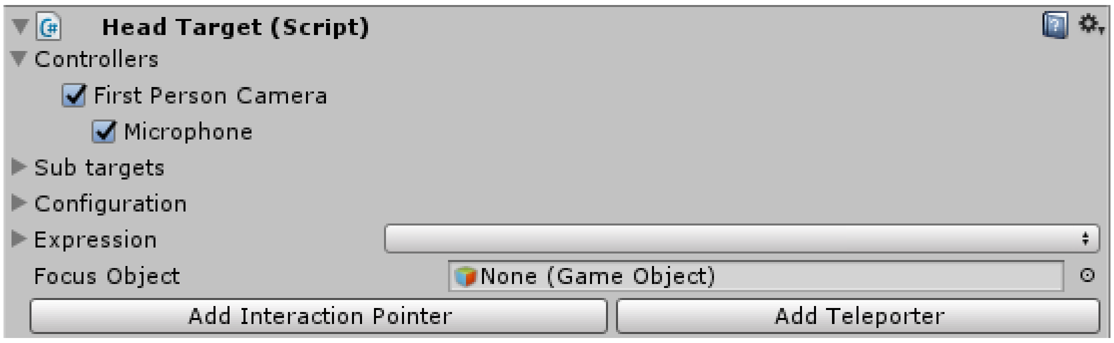
Unity provides a great event system which can be used to interact with a scene. This solution is particularly useful for UI interfaces. Humanoid Control provide extensive and flexible support for the event system using the Interaction module.

The interaction module is supported for gazing/pointing and finger touching. This means that you can trigger events by looking at objects, pointing at them or by touching them. You can determine yourself which body part is used for the gazing/pointing.

Note that you currently can use only one interaction module at the same time!

Interaction Pointer

An interaction pointer can be attached to many items, but it is often used for head gaze and finger pointing interaction. For you convenience, buttons are added to the Head and Hand Targets to add Interaction Pointers.


Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone
Sub targets
Configuration
Expression
Focus Object: None (Game Object)

The *Add Interaction Pointer* button adds an interaction pointer GameObject as a child of the target.

To remove an interaction pointer, the *Interaction Pointer* child object of the Target can be removed. After this, it is possible to add a new interaction pointer again with the button.

When an Interaction Pointer is active, you can select objects like UI elements. These objects will receive the Unity EventSystem events PointerEnter, PointerExit, PointerSelect and PointerDeselect. You can use an Event Trigger component to handle these events for generic objects with colliders.

When a click is performed with an interaction pointer, the object will receive the PointerDown, PointerUp and PointerClick events. Besides that, when you move the pointer while clicking, the events BeginDrag, Drag and EndDrag are triggered. These events are used by the UI element to implement clicking on buttons and dragging sliders for example.

A detailed description of the Interaction Pointer component is found here: Interaction Pointer.

Head Gaze for the Head Target

When adding the Interaction Pointer with the button, this interaction pointer will have a sphere as the Focus Point Object which visualizes the point the user is looking at. This sphere will be visible while the Interaction Pointer is active.

This sphere can be replaced by other visualizations if you want, for example a sprite renderer.

Finger Pointing for the Hand Target

For finger pointing, a Line Renderer is used for the Focus Point Object to show where the user it pointing. This line renderer will be visible while the Interaction Pointer is active.

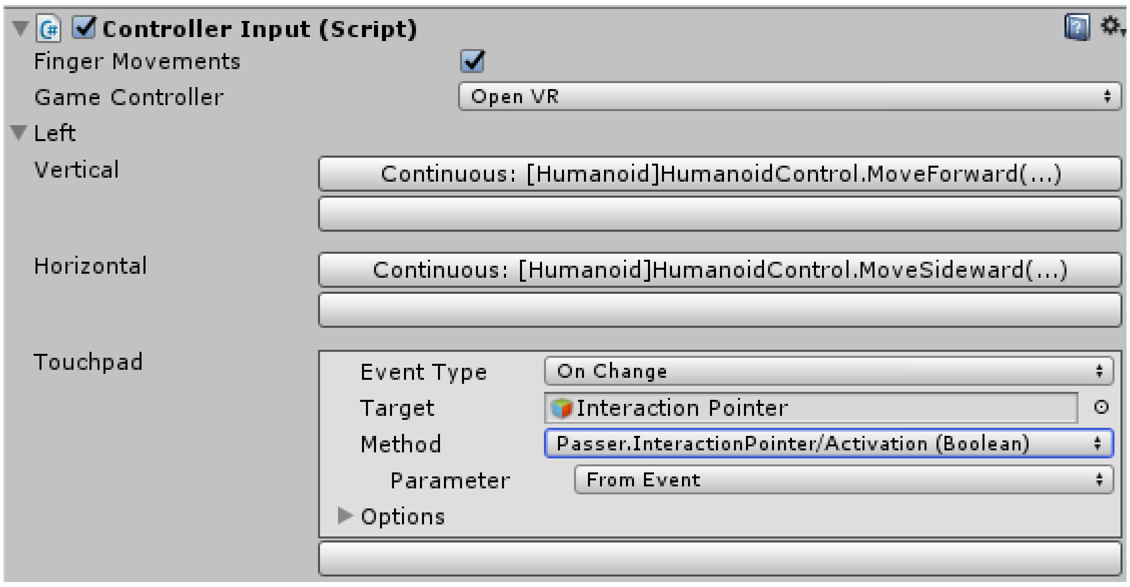
The line renderer can be removed or altered. Additionally further visualizations can be added to the Focus Point Object.

Controller Input

The Add Interaction Button add by default the following controller input actions:

- Button 1 press: activate Interaction Pointer
- Trigger 1 press: click with the Interaction Pointer

This can be altered easily with the Controller Input script for the humanoid. For example, the SteamVR Controller does not have a ‘Button one’, so you can assign the touchpad press to activate the Interaction Pointer like this in Humanoid Control v2:


Controller Input (Script)

Finger Movements ✓

Game Controller: Open VR

Left

Vertical: Continuous: [Humanoid]HumanoidControl.MoveForward(...)

Horizontal: Continuous: [Humanoid]HumanoidControl.MoveSideward(...)

Touchpad

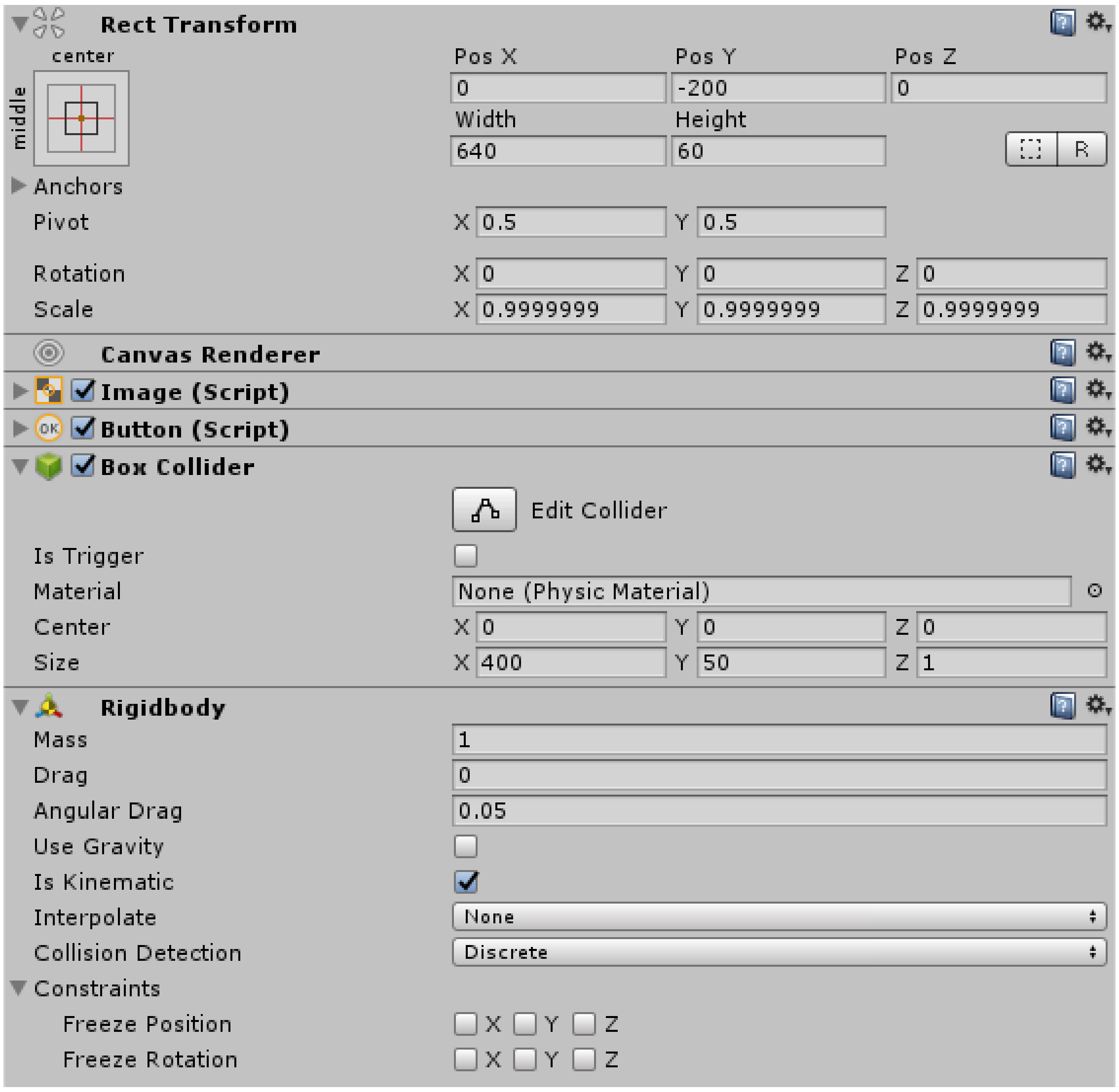
Event Type	On Change
Target	Interaction Pointer
Method	Passer.InteractionPointer/Activation (Boolean)
Parameter	From Event
Options	

The Interaction Pointer object field should refer to the interaction pointer on the correct hand, in this case the left hand.

UI elements

The interaction Pointer supports interaction with Unity's UI elements like UI Buttons, but it is necessary to add Colliders and Kinematic Rigidbodies to the UI elements you want to interact with. This is because the Interaction Pointer uses a raycast which does not interact with UI elements.

For example, for a UI button you can add a Box Collider. Make sure the X and Y size match the size of the button itself, the Z size I usually set to 1:



Rect Transform

center

middle

	Pos X	Pos Y	Pos Z
	0	-200	0
	Width	Height	
	640	60	

Anchors

Pivot	X 0.5	Y 0.5	
Rotation	X 0	Y 0	Z 0
Scale	X 0.9999999	Y 0.9999999	Z 0.9999999

Canvas Renderer

☑ Image (Script)

☑ Button (Script)

☑ Box Collider

Edit Collider

Is Trigger	☐		
Material	None (Physic Material)		
Center	X 0	Y 0	Z 0
Size	X 400	Y 50	Z 1

Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☐
Is Kinematic	☑
Interpolate	None
Collision Detection	Discrete

Constraints

Freeze Position	☐ X ☐ Y ☐ Z
Freeze Rotation	☐ X ☐ Y ☐ Z

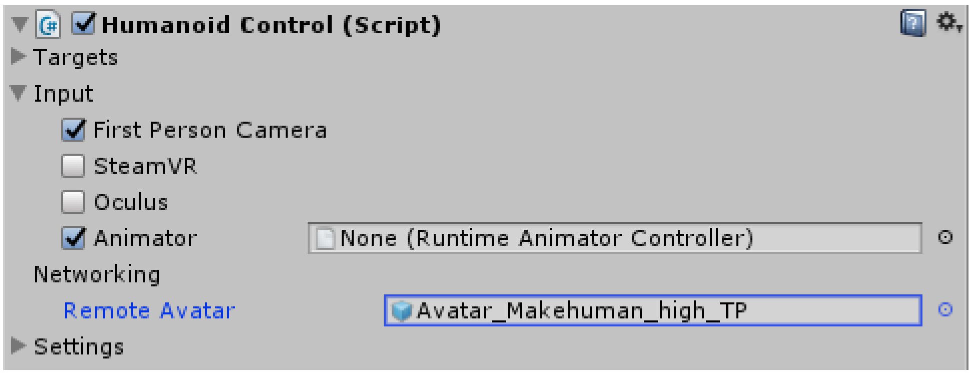
Networking

With Humanoid Control you can extend your project to a multiplayer environment in an east way. The following networking solutions are supported:

- Unity Networking
- Photon PUN Classic
- Photon PUN 2 (v2.1+)
- Photon Bolt (v3+)
- Mirror Networking (v3+)

Remote avatar

For networked setup, you need to select an avatar to be used on remote clients. This avatar can be different from the local avatar. This enables you to optimize the first person avatar for the local player (for example an avatar without a head and high-poly hands) while having an avatar optimized for third person at remote clients.



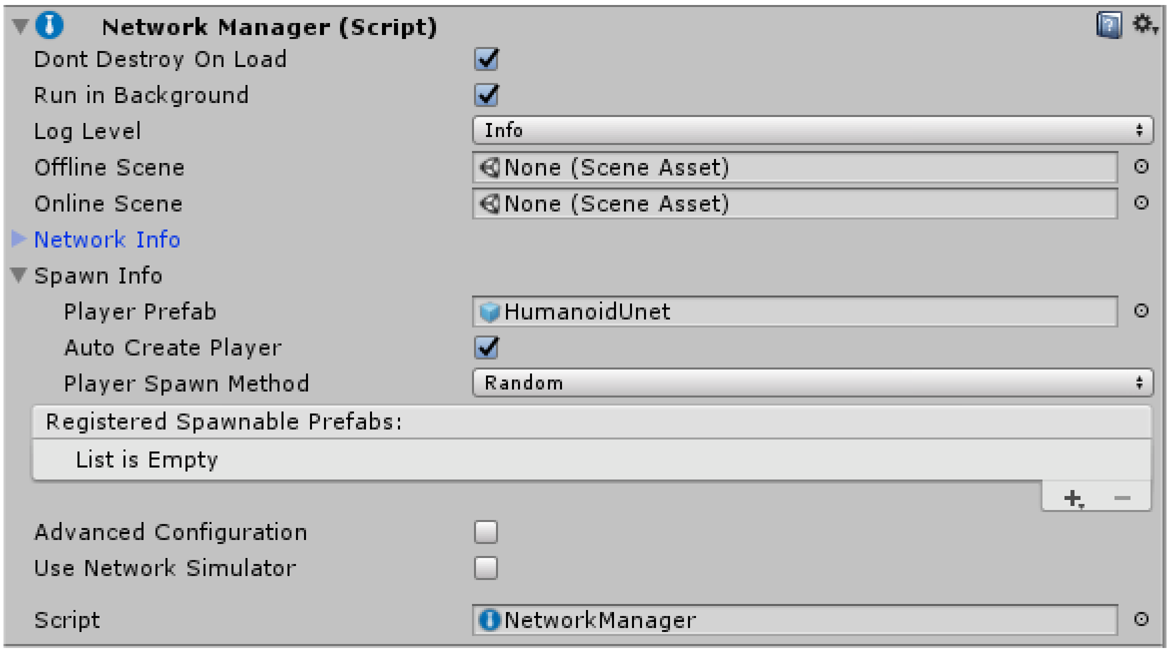
Humanoid Control (Script)
Targets
Input
[✓] First Person Camera
[ ] SteamVR
[ ] Oculus
[✓] Animator — None (Runtime Animator Controller)
Networking
Remote Avatar — Avatar_Makehuman_high_TP
Settings

This avatar is a normal avatar with an Animator and a Humanoid Rig and without a Humanoid Control script.

Make a multiplayer scene

The HumanoidPlayer player prefab implements all networking functionality necessary to turn a single player scene into a multiplayer scene.

This player prefabs can be spawned on the networking by the Humanoid Networking Starter script or the Unity Network Manager (HUD) script as the Player Prefab.

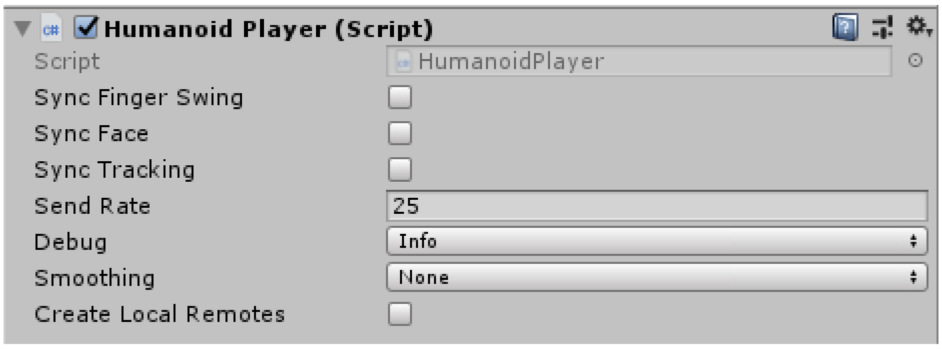


Network Manager (Script)

Dont Destroy On Load	✓
Run in Background	✓
Log Level	Info
Offline Scene	None (Scene Asset)
Online Scene	None (Scene Asset)
Network Info	
Spawn Info	
Player Prefab	HumanoidUnet
Auto Create Player	✓
Player Spawn Method	Random
Registered Spawnable Prefabs:	List is Empty
Advanced Configuration	☐
Use Network Simulator	☐
Script	NetworkManager

Humanoid Player

When this player prefab is spawned, they will scan the scene for existing local Humanoids and it will monitor the instantiation of new humanoids. It will spawn remote humanoids on every remote client so that remote players will see these humanoids in their environment. From then on, these humanoids will be fully synchronized, including actions like grabbing and dropping objects and changing the avatar.



Humanoid Player (Script)

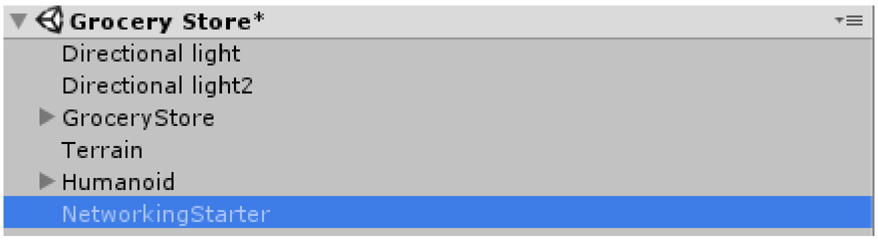
Script	HumanoidPlayer
Sync Finger Swing	☐
Sync Face	☐
Sync Tracking	☐
Send Rate	25
Debug	Info
Smoothing	None
Create Local Remotes	☐

Sync Finger Swing	If this is disabled only finger curl values will be synchronized. This will reduce the needed bandwidth. When enabled finger swing is also synchronized. This will lead to better hand poses.
Sync Face	(Pro) When enabled will sync facial movements across the network.
Sync Tracking	Will sync the Lighthouse positions when using SteamVR.
Send Rate	The number of updates per second communicated through the network
Debug	Controls the amount of debugging information in the Console Window of Unity.
Smoothing	Applies Interpolation or Extrapolation to achieve smoother movements for remote humanoids.
Create Local Remotes	This will create local remote avatars which can be convenient for debugging purposes.

Networking Starter

The easiest way to make a networked multiplayer environment is to use the Networking Starter component. The starting point is a single player environment. The only thing you need to take care of is that you set the Remote Avatar (see below).

To turn this in a multiplayer environment you need to add the Networking Starter component to the scene. You can find this in the *Humanoid->Prefabs->Networking* folder.



This script will take care of all networking. At launch, it scans the scene for Humanoids and recreates the Humanoid at each remote client, synchronizing all movements and actions over the network.

The exact parameters of the script is found at the page for the relevant networking solution:

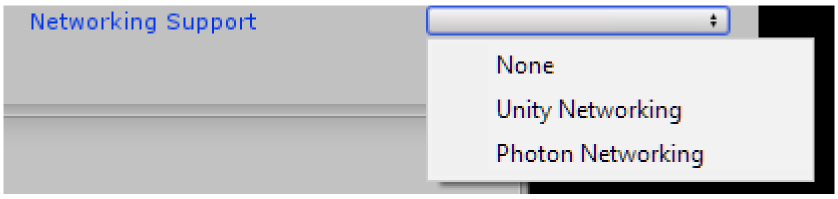
Unity Networking

Prerequisites

Unity Networking works with Unity 5.5 up to 2018.4.

Setup

In the Humanoid Preferences you can select which networking package you want to use. Go to *Edit Menu->Preferences->Humanoid->Networking Support* and select the desired networking package.



Networking Starter

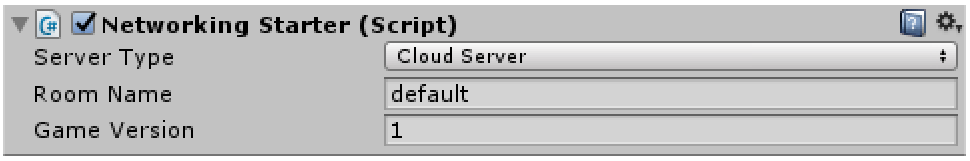
For the configuration we have two options: *Cloud Server* and *Own Server*.

Warning: the standard free Unity Networking Service is very limited in its bandwidth. With more two players and a high send rate, you will soon see disconnects. These disconnects happen if the bandwidth limit of the Unity Networking Service is exceeded. For this reason we advise to the a paid service for Unity Networking using the *Cloud Server* or to use your *Own Server*.

Cloud Server

In this option you will use a server in the cloud, provided and hosted by a third party like Unity or Photon.

In the Cloud server option you can specify the following parameters:



Networking Starter (Script) ✓

Server Type	Cloud Server
Room Name	default
Game Version	1

Room Name	The name of the environment shared by the players.
Game Version	The version of the environment shared by the players.

Own Server

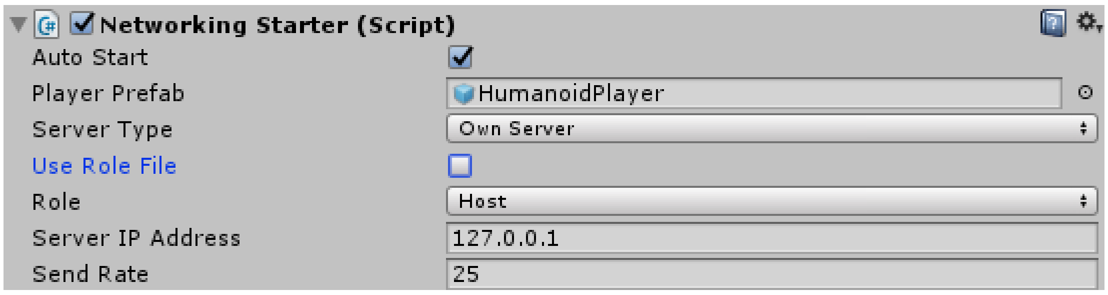
When using your own server, you have two major options:

Use different builds for the Host and the Client. In this case, do not *Use Role File*.

Use a single build for all uses. In this case you need to select *Use Role File*.

Using different builds for Host and Client

Ensure that *Use Role File* is disabled. You will have the following options:



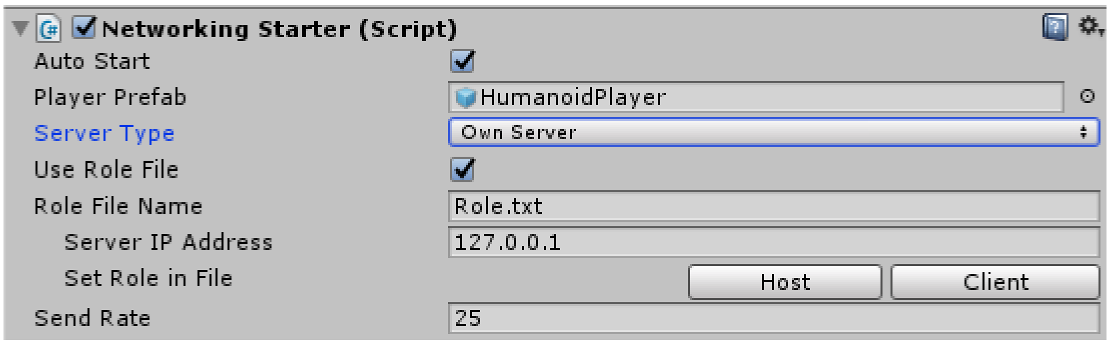
Networking Starter (Script) ✓

Auto Start	✓
Player Prefab	HumanoidPlayer
Server Type	Own Server
Use Role File	☐
Role	Host
Server IP Address	127.0.0.1
Send Rate	25

Ensure that the *Server IP Address* is set to the IP address of the Host. Then you can make a build for the Host with the *Role* setting on *Host*. When this build is completed, you can switch the *Role* to *Client* and make the build for the client.

Using a single build for the Host and Client

With this option a *Role.txt* file is used by the bulld to determine whether it should run as Host or Client. It also defined the IP address of the Host so that it can be changed easily without making a new build.



Networking Starter (Script) ✓

Auto Start	✓
Player Prefab	HumanoidPlayer
Server Type	Own Server
Use Role File	✓
Role File Name	Role.txt
Server IP Address	127.0.0.1
Set Role in File	Host / Client
Send Rate	25

Role File	The filename of the role file. This file should be located in the Assets/StreamingAssets/

Name	folder in your project.
Server IP Address	The IP address of the host which will be written into the role file.
Set Role in File	These buttons will create or overwrite the role file in Assets/StreamingAssets/ folder using the *Role File Name*. The *Server IP Address* and the chosen button will deteremine the contents of this file. The file van be edited manually afterwards.

The Role File Format

The role file is has a very basic 2-line format.

Line 1: 'Host' or 'Client' depending on the desired running mode
Line 2: The IP address of the host.

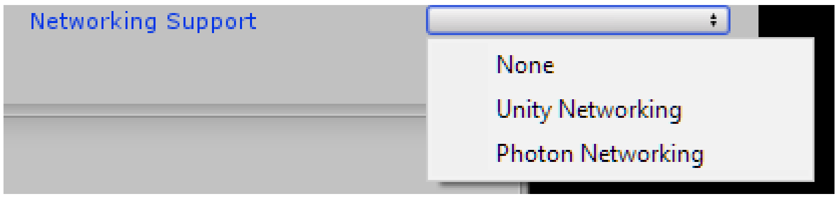
Example:
```
Host
192.168.1.37
```

Photon PUN Classic

Setup

In order to use Photon PUN in your project. You need to import the Photon PUN package first from the asset store.

After this, go to the Edit Menu->Preferences. Humanoid Control will check the availability of the package and the Photon Networking option should appear in the Networking Support property:

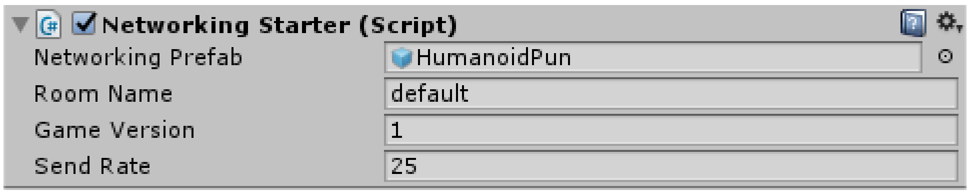


When Photon Networking is selected for Networking Support, networking support components like the Networking Starter and Networking Transform should adapt automatically to Photon Networking.

The Grocery Store Networking demo scene will now support Photon Networking.

Networking Starter

When using Photon PUN Class, the Networking Starter has the following options:


Networking Starter (Script)
Networking Prefab: HumanoidPun
Room Name: default
Game Version: 1
Send Rate: 25

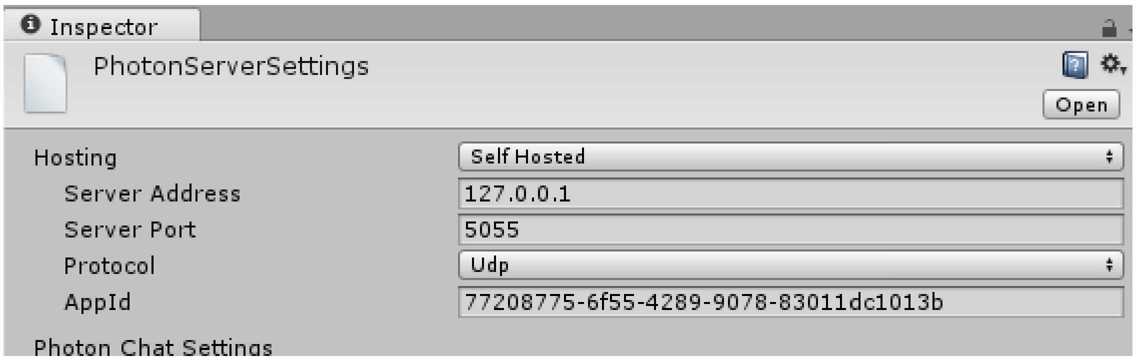
Networking Prefab	The player prefab which will be spawned across the network. This defaults to HumanoidPun for Photon Networking.
Room Name	The name of the environment shared by the players.
Game Version	The version of the environment shared by the players

Send Rate	The number of updates per second communicated through the network.

Self Hosted Networking

The Photon Networking default is to use a cloud based server hosted by Photon. As an alternative, you can use your own Photon Server. This can be downloaded here on the Photon Engine SDKs page.

Then you should change the Photon Server Settings to 'Self Hosted' and set the Server Address to the ip address of the machine running the server:



PhotonServerSettings

Hosting	Self Hosted
Server Address	127.0.0.1
Server Port	5055
Protocol	Udp
AppId	77208775-6f55-4289-9078-83011dc1013b

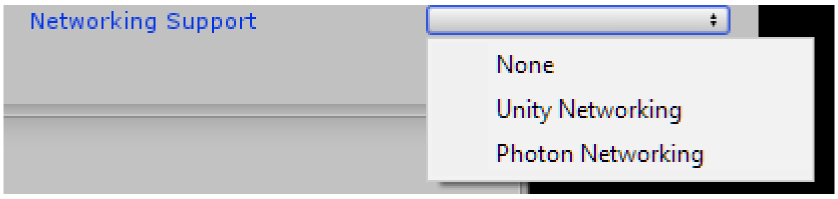
Photon Chat Settings

Photon PUN 2

Setup

In order to use Photon PUN 2 in your project. You need to import the Photon PUN 2 package first from the asset store.

After this, go to the Edit Menu->Preferences. Humanoid Control will check the availability of the package and the Photon Networking option should appear in the Networking Support property:

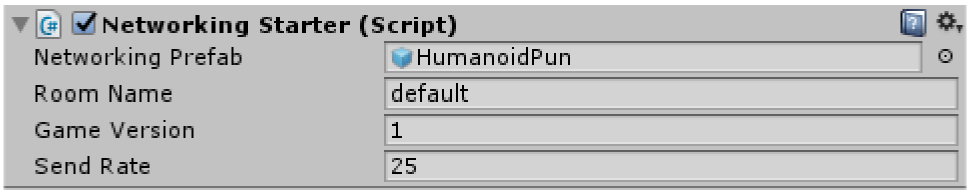


When Photon Networking is selected for Networking Support, networking support components like the Networking Starter and Networking Transform should adapt automatically to Photon Networking.

The Grocery Store Networking demo scene will now support Photon Networking.

Networking Starter

When using Photon PUN 2, the Networking Starter component has the following options:



Networking Starter (Script)

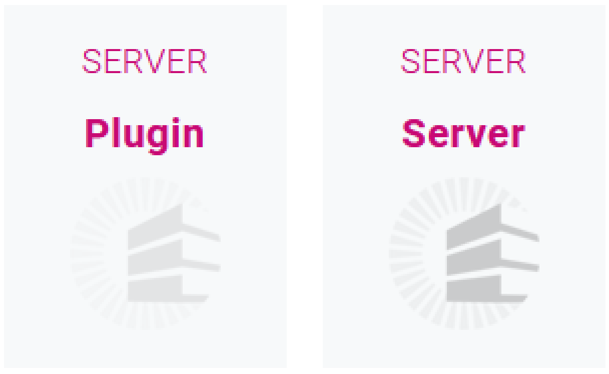
Networking Prefab	HumanoidPun
Room Name	default
Game Version	1
Send Rate	25

Networking Prefab	The player prefab which will be spawned across the network. This defaults to HumanoidPun for Photon Networking.
Room Name	The name of the environment shared by the players

Game Version	The version of the environment shared by the players
Send Rate	The number of updates per second communicated through the network.

Self Hosted Networking

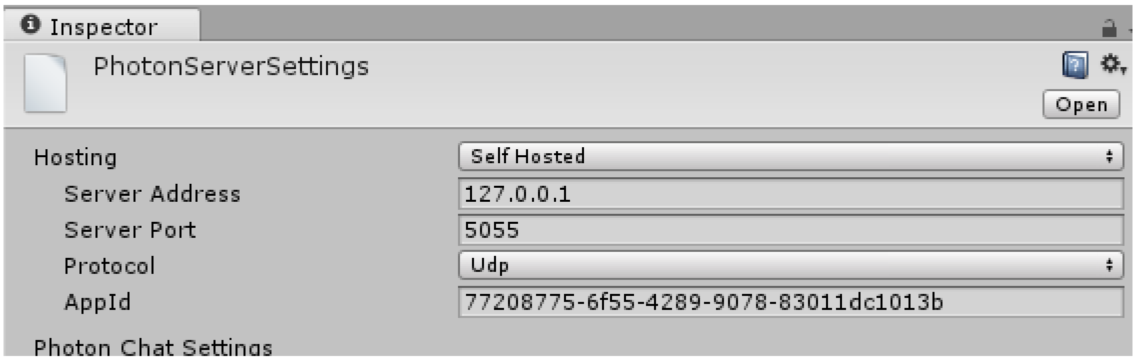
For local networking Photon PUN, go to the following page:https://www.photonengine.com/en-US/sdks#server-sdkserverserver. Then select ‘Server’:



And download the SDK.

Instructions on how to set it up can be found here: https://doc.photonengine.com/en-us/server/current/getting-started/photon-server-in-5min

The settings you need to use in Unity for Photon 2 are as follows:



Hosting	Self Hosted
Server Address	127.0.0.1
Server Port	5055
Protocol	Udp
AppId	77208775-6f55-4289-9078-83011dc1013b

Photon Chat Settings

In this case, we use a direct IP Address (localhost/127.0.01), so Use Name Server is disabled. As the protocol is Udp, you need to set port to 5055. Other port numbers can be found here: https://doc.photonengine.com/en-us/realtime/current/connection-and-authentication/tcp-and-udp-port-numbers

If you start the scene now, Photon should connect to the local server.

Photon Bolt

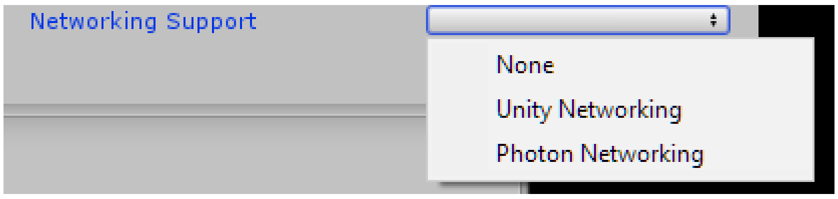
Prerequisites

Photon Bolt support requires Humanoid Control v3 or higher.

Setup

In order to use Photon Bolt in your project. You need to import and setup the Photon Bolt package first from the asset store.

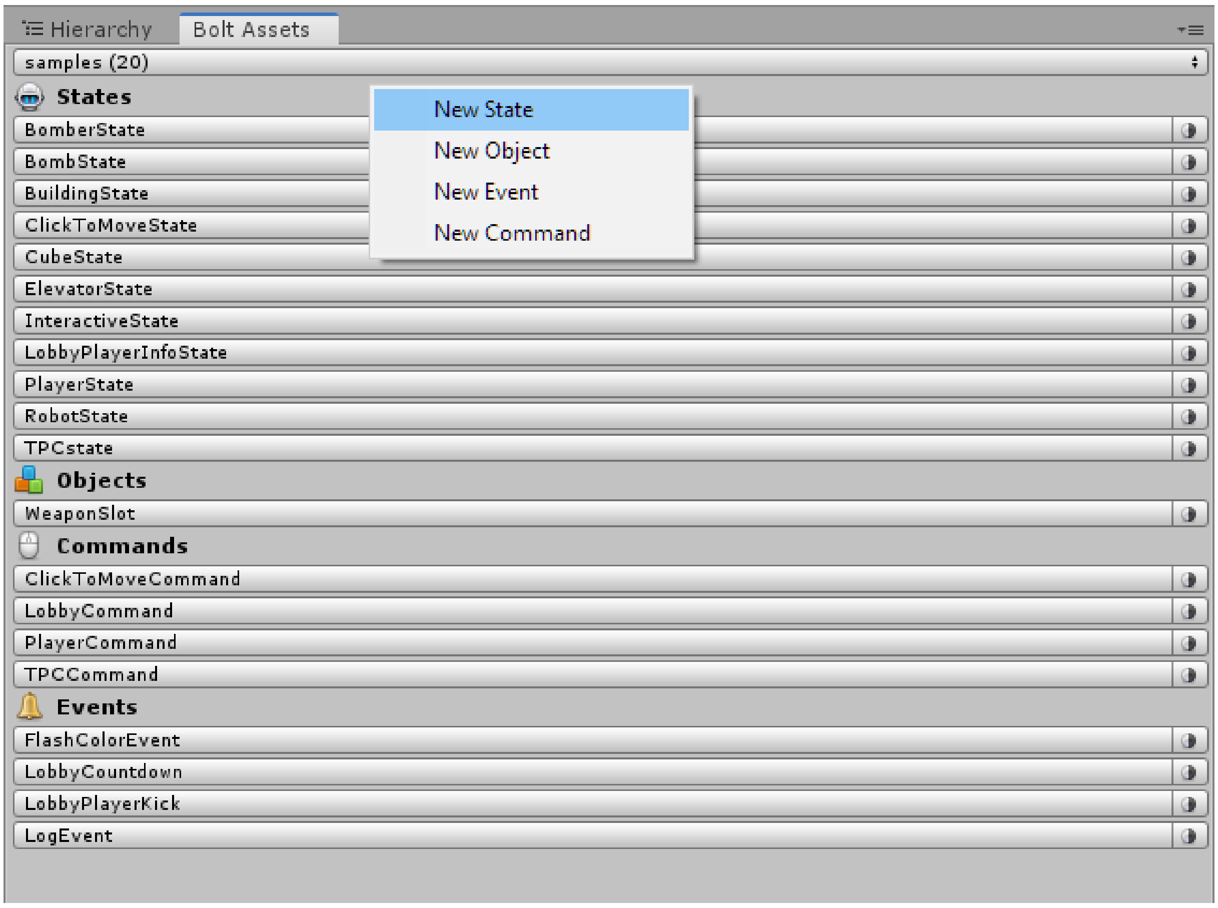
After this, go to the Edit Menu->Preferences. Humanoid Control will check the availability of the package and the Photon Bolt option should appear in the Networking Support property:



When Photon Networking is selected for Networking Support, networking support components like the Networking Starter and Networking Transform should adapt automatically to Photon Networking.

Bolt Setup

To support Humanoid Control, a Bolt State Asset needs to be added to the project. Go to the Bolt Menu->Assets, right-click in the new Bolt Assets window and select 'New State':



samples (20)

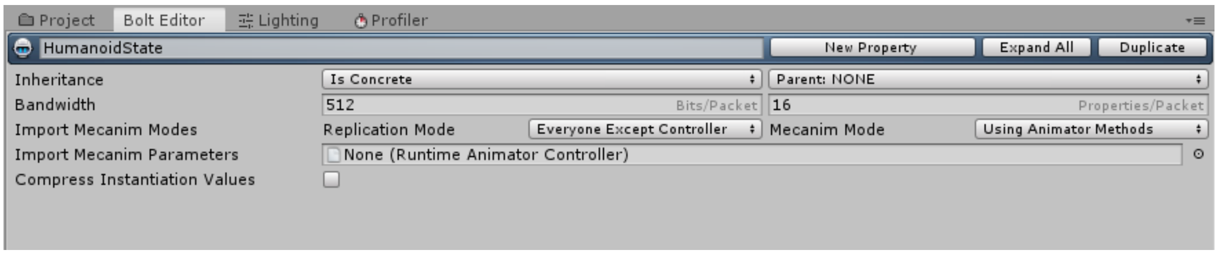
States
BomberState
BombState
BuildingState
ClickToMoveState
CubeState
ElevatorState
InteractiveState
LobbyPlayerInfoState
PlayerState
RobotState
TPCstate

Objects
WeaponSlot

Commands
ClickToMoveCommand
LobbyCommand
PlayerCommand
TPCCommand

Events
FlashColorEvent
LobbyCountdown
LobbyPlayerKick
LogEvent

Name this new State 'HumanoidState'. No additional settings need to be done.

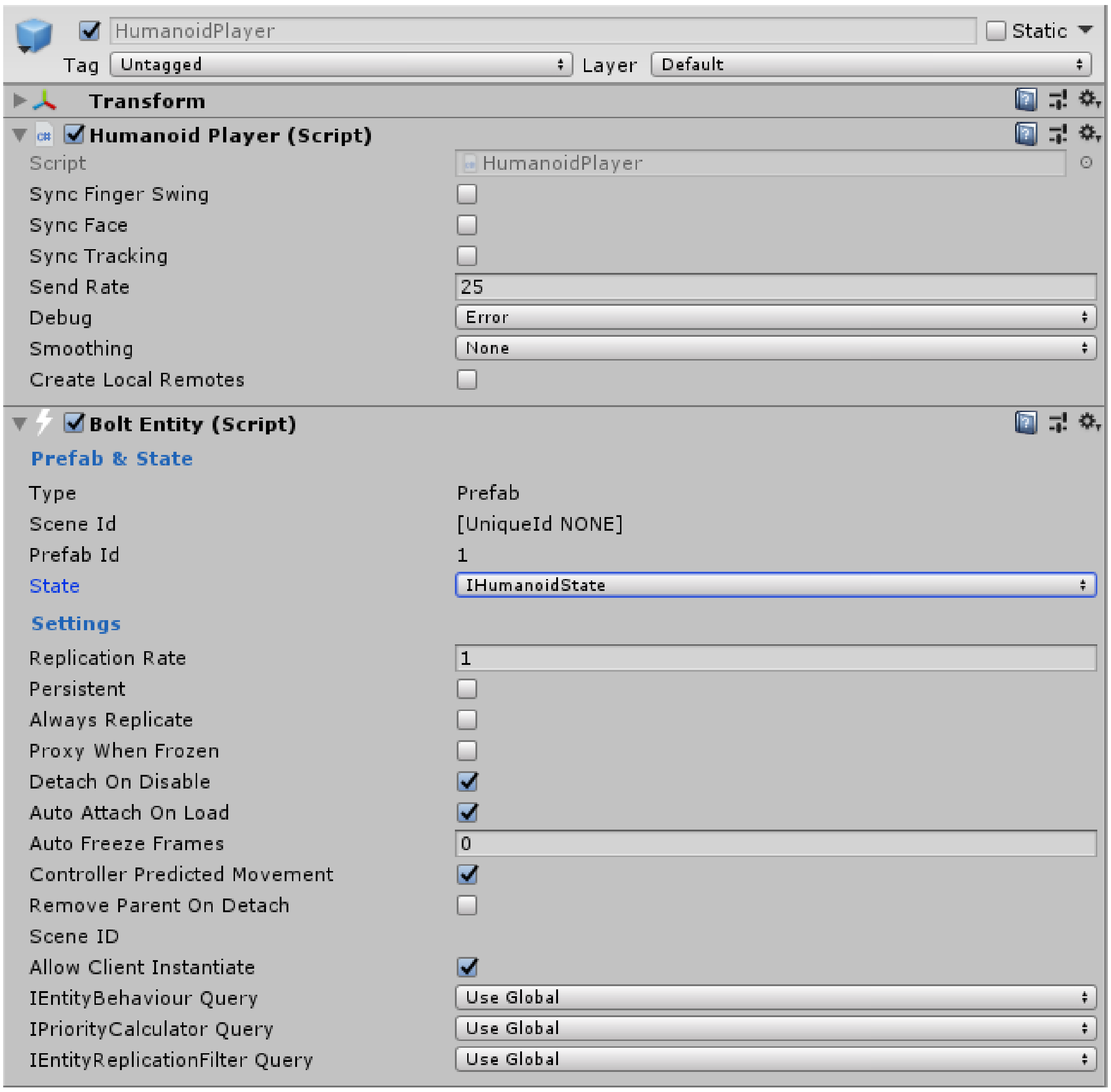


HumanoidState

Inheritance	Is Concrete	Parent: NONE
Bandwidth	512 Bits/Packet	16 Properties/Packet
Import Mecanim Modes	Replication Mode: Everyone Except Controller	Mecanim Mode: Using Animator Methods
Import Mecanim Parameters	None (Runtime Animator Controller)	
Compress Instantiation Values	☐	

Now select Bolt Menu->Compile Assemble to ensure that this new state becomes available.

Next in your project Window navigate to Assets/Humanoid/Prefabs/Networking/Resources/ and select HumanoidPlayer.prefab. In the Bolt Entity component of the prefab, set the State to IHumanoidState:



HumanoidPlayer | Static
Tag: Untagged | Layer: Default

Transform

Humanoid Player (Script) [✓]

Script	HumanoidPlayer
Sync Finger Swing	☐
Sync Face	☐
Sync Tracking	☐
Send Rate	25
Debug	Error
Smoothing	None
Create Local Remotes	☐

Bolt Entity (Script) [✓]

Prefab & State

Type	Prefab
Scene Id	[UniqueId NONE]
Prefab Id	1
State	IHumanoidState

Settings

Replication Rate	1
Persistent	☐
Always Replicate	☐
Proxy When Frozen	☐
Detach On Disable	✓
Auto Attach On Load	✓
Auto Freeze Frames	0
Controller Predicted Movement	✓
Remove Parent On Detach	☐
Scene ID	
Allow Client Instantiate	✓
IEntityBehaviour Query	Use Global
IPriorityCalculator Query	Use Global
IEntityReplicationFilter Query	Use Global

Now everything is setup to get Humanoid Control working with Photon Bolt. You can try out the Grocery Store Networking demo scene to test it yourself.

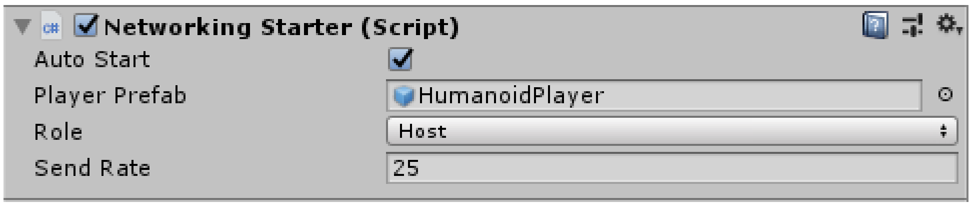
Grabbing Objects

Each object which can be picked up by a humanoid should have a BoltEntity with a set State. If this is not present, an error will be raised and the remote humanoid will not grab the object.

Due to limitations in the Bolt package, it is not possible to automatically add an Bolt Entity to each object in the Grocery Store Networking scene. These should be added manually.

Networking Starter

When using Photon Bolt, the Networking Starter component has the following options:


Networking Starter (Script)

Auto Start	✓
Player Prefab	HumanoidPlayer
Role	Host
Send Rate	25

Auto Start	Starts networking automatically when the scene starts.
Player Prefab	The prefab which will be spawned across the network. For Humanoid Control this is typically the HumanoidPlayer prefab.
Role	Whether the networking should start as a Host (= Bolt Server) or Client.
Send Rate	The number of network updates per second used to update the pose of the humanoids.

Photon Voice

Photon Classic

For voice chat support, make sure that you have included the Photon Voice package.

In the Photon Server Settings (found in Photon Unity Networking/Resources), set the Voice AppId to the same App Id you use for setting up Photon.

Make sure that the humanoid has an audio listener on the head target. The humanoid prefab already has this included

Select the Photon PUN prefab which is spawned across the network. If you use the Networking Starter this usually is Humanoid PUN which is found in Assets/Humanoid/Prefabs/Networking/Resources/

Add the Photon Voice Recorder script to the Player Prefab. Doing this will also add an Audio Source and Photon Voice Speaker automatically.

That’s it. Voice chat should work now.

Photon 2

For photon 2, you need to include the Photon Voice 2 package in your project.

In the Photon Server Settings (see Window Menu->Photon Unity Networking->Highlight Server Settings, set the Settings->App Id Voice to the same App Id you use for setting up Photon 2. This App Id is also found in the App Id Realtime field.

Select the Photon PUN prefab which is spawned across the network. If you use the Networking Starter this usually is the HumanoidPun prefab which is found in Assets/Humanoid/Prefabs/Networking/Resources/

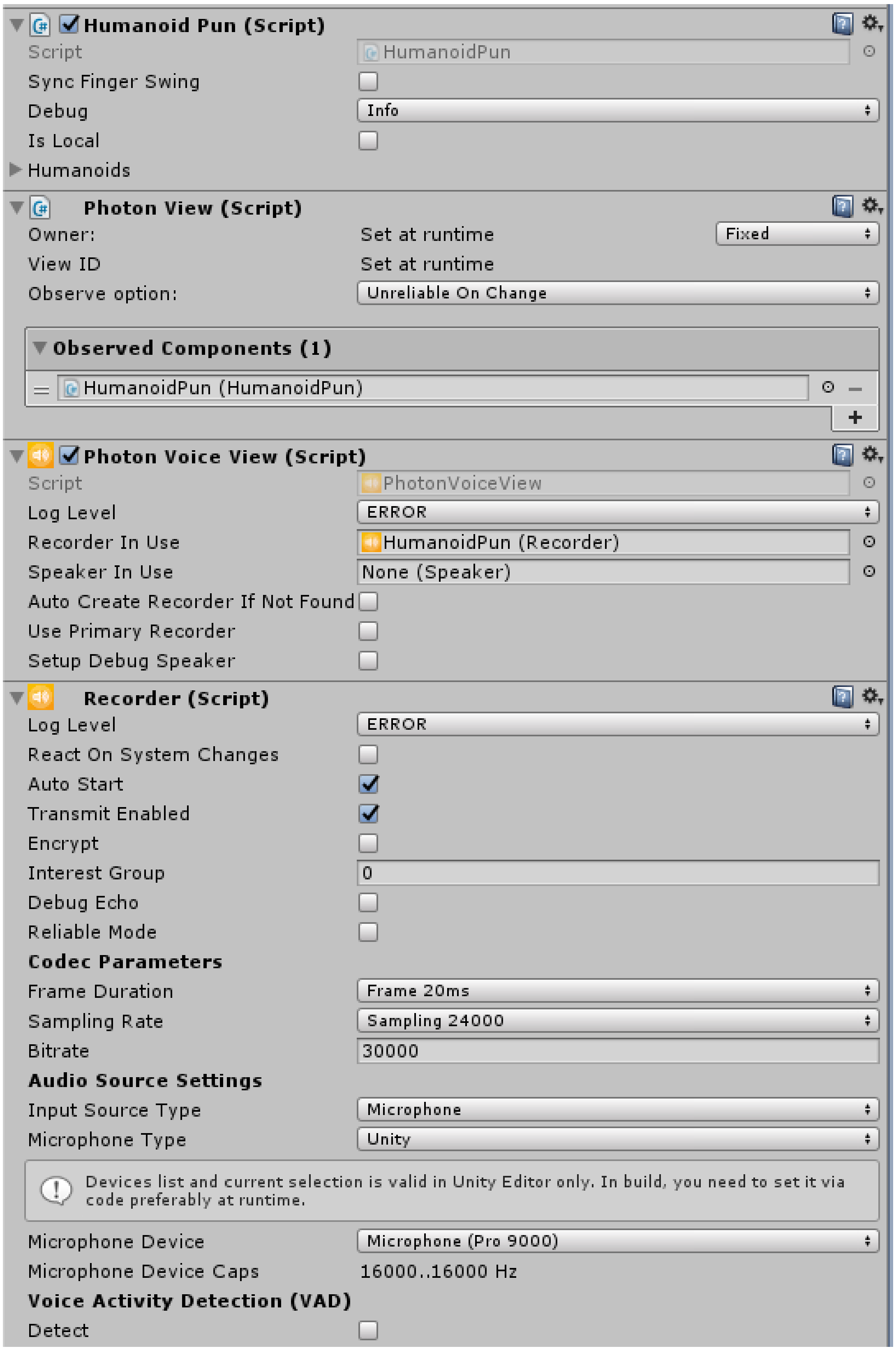
Add the following components to the HumanoidPun prefab:

Add a *Photon Voice View* component

Add a *Recorder* component, make sure you have *Auto Start* and *Transmit Enabled* switched on

Set *Photon Voice View->Recorder in Use* to this Recorder

The result should look like this:


Humanoid Pun (Script)

Script	HumanoidPun
Sync Finger Swing	☐
Debug	Info
Is Local	☐
Humanoids	

Photon View (Script)

Owner:	Set at runtime — Fixed
View ID	Set at runtime
Observe option:	Unreliable On Change

Observed Components (1)

HumanoidPun (HumanoidPun)

Photon Voice View (Script)

Script	PhotonVoiceView
Log Level	ERROR
Recorder In Use	HumanoidPun (Recorder)
Speaker In Use	None (Speaker)
Auto Create Recorder If Not Found	☐
Use Primary Recorder	☐
Setup Debug Speaker	☐

Recorder (Script)

Log Level	ERROR
React On System Changes	☐
Auto Start	✓
Transmit Enabled	✓
Encrypt	☐
Interest Group	0
Debug Echo	☐
Reliable Mode	☐
Codec Parameters	
Frame Duration	Frame 20ms
Sampling Rate	Sampling 24000
Bitrate	30000
Audio Source Settings	
Input Source Type	Microphone
Microphone Type	Unity

Devices list and current selection is valid in Unity Editor only. In build, you need to set it via code preferably at runtime.

Microphone Device	Microphone (Pro 9000)
Microphone Device Caps	16000..16000 Hz
Voice Activity Detection (VAD)	
Detect	☐

Now if you start you project, Photon Voice should work.

Mirror Networking

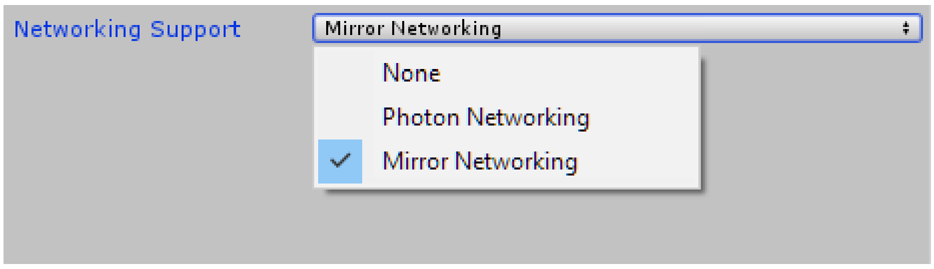
Prerequisites

Mirror Networking support requires Humanoid Control v3.0 or higher.

Setup

In order to use Mirror Networking in your project. You need to import the Mirror Networking package first from the asset store.

After this, go to the Edit Menu->Preferences. Humanoid Control will check the availability of the package and the Mirror Networking option should appear in the Networking Support property:

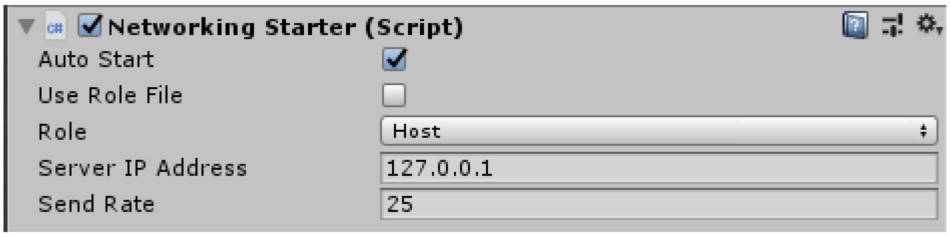

Networking Support: Mirror Networking

When Mirror Networking is selected for Networking Support, networking support components like the Networking Starter and Networking Transform should adapt automatically to Mirror Networking.

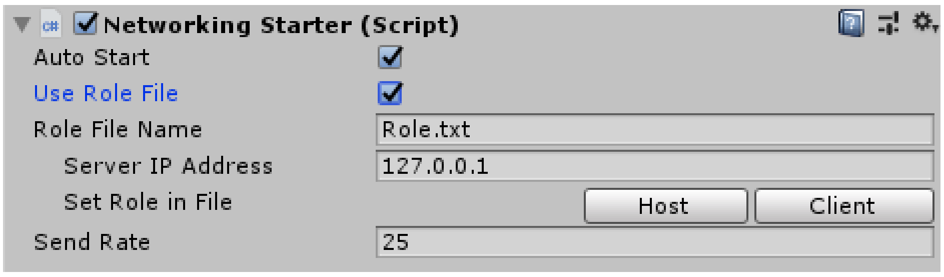
The Grocery Store Networking demo scene will now support Mirror Networking.

Networking Starter

When using Mirror Networking, the Networking Starter component has the following options:


Networking Starter (Script)

Auto Start	✓
Use Role File	☐
Role	Host
Server IP Address	127.0.0.1
Send Rate	25


Networking Starter (Script)

Auto Start	✓
Use Role File	✓
Role File Name	Role.txt
Server IP Address	127.0.0.1
Set Role in File	Host / Client
Send Rate	25

Auto Start	Will automatically start a Host or Client depending on the Role parameter.
Use Role File	Will use an role file specifying the networking role (Host or Client) and server IP address.
Role File Name	The name of the Role File. The Role File will be found in *Application.dataPath.*
Server IP Address	The ip addres of the server.
Set Role in File	Will generate a Role File in *Application.dataPath*. using the specified role and server

	IP address.
Send Rate	The number of updates per second communicated through the network.

Tools

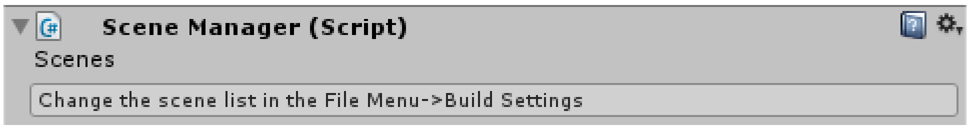
Scene Manager

The scene manager can be used to synchronize scene changes with humanoid across a network.

Setup

The Scene Manager script can be found in Assets/Humanoid/Scripts/Tools are can be attached to an (empty) GameObject.

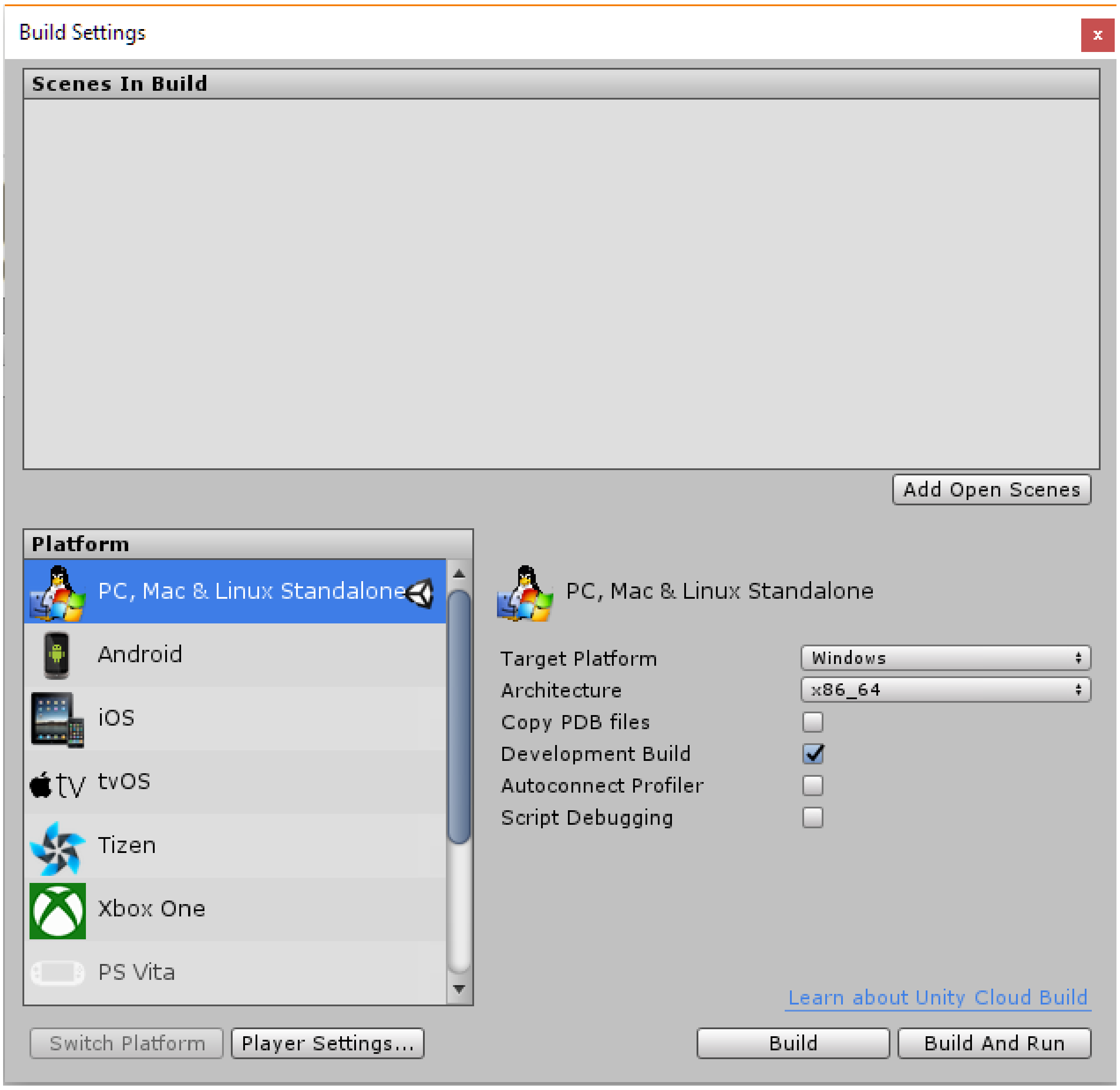
Initially, the scene manager can have no Scene. It does not have any fields which can be changed, because the scenes are controlled by the Build Settings.



Change the scene list in the File Menu->Build Settings

Build Settings

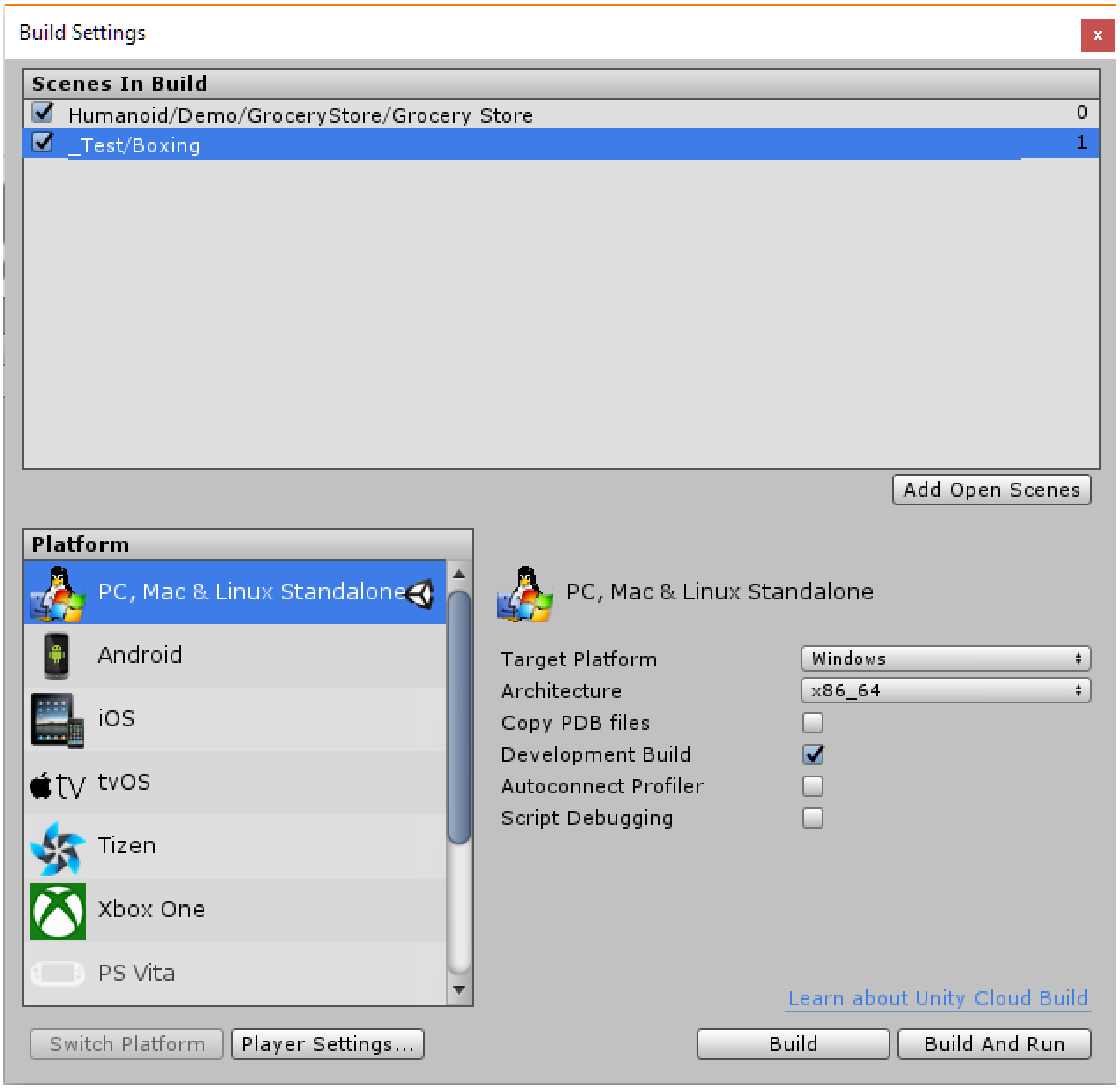
The scenes managed by the Scene Manager are set using the Build Settings which can be accessed in the File Menu->Build Settings.



PC, Mac & Linux Standalone

Target Platform: Windows
Architecture: x86_64
Copy PDB files: ☐
Development Build: ☑
Autoconnect Profiler: ☐
Script Debugging: ☐

You can add the currently opened scene by pressing the *Add Open Scenes* button or you can drag scenes from the Project Window into the *Scenes In Build* area.


Build Settings

Scenes In Build

✓	Humanoid/Demo/GroceryStore/Grocery Store	0
✓	_Test/Boxing	1

Add Open Scenes

Platform

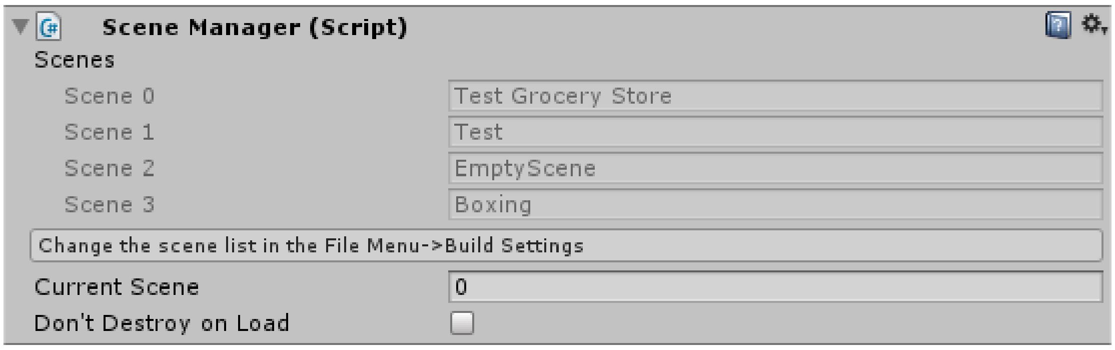
- PC, Mac & Linux Standalone
- Android
- iOS
- tvOS
- Tizen
- Xbox One
- PS Vita

PC, Mac & Linux Standalone

Target Platform	Windows
Architecture	x86_64
Copy PDB files	☐
Development Build	✓
Autoconnect Profiler	☐
Script Debugging	☐

Scenes can be deleted by selecting the in the *Scenes In Build* Area and then press *Delete* on the keyboard.

The information in the Scene Manager will be updated when it has been deselected and selected again in the Hierarchy Window:


Scene Manager (Script)

Scenes

Scene 0	Test Grocery Store
Scene 1	Test
Scene 2	EmptyScene
Scene 3	Boxing

Change the scene list in the File Menu->Build Settings

Current Scene	0
Don't Destroy on Load	☐

It has the following parameters:

Scenes	The list of scenes from the Build Settings.
Current Scene	The build index of the current scene. Can be higher than the number of scenes if the current scene is not in the list of scenes.
Don't Destroy on Load	Will prevent the scene manager from being destroyed when the scene changes.

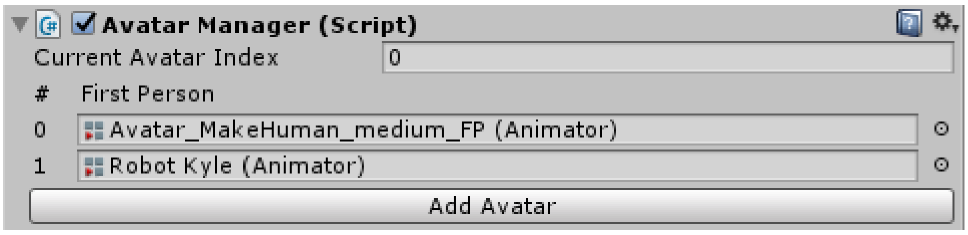
Avatar Manager

The avatar manger can be used to manage multiple avatar meshes for a single humanoid. It is supported single player and networking setups.

Setup

The Avatar Manager script can be found in Assets/Humanoid/Scripts/Tools/ and should be attached to an GameObject with the Humanoid Control component script.

Single Player


Avatar Manager (Script)

Current Avatar Index: 0

#	First Person
0	Avatar_MakeHuman_medium_FP (Animator)
1	Robot Kyle (Animator)

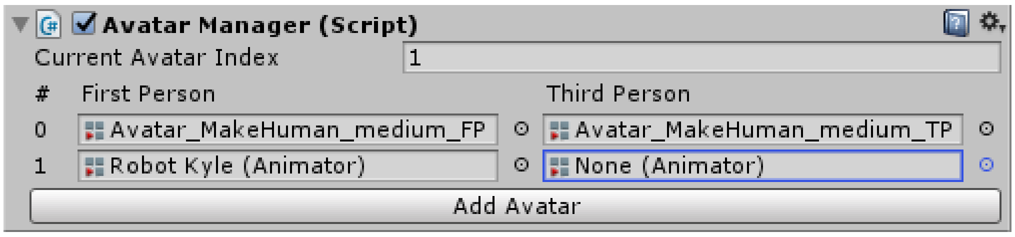
The Avatar Manger script shows the list of available avatars to use. Every avatar needs to have a Animator component attached and only avatars from the Project Window (prefabs, models) are supported, so you cannot use avatar in you scene.

New avatars can be added to the list by clicking *Add Avatar* while empty slots will be cleaned up automatically.

The *Current Avatar Index* shows the index number of the avatar which is currently used. At startup, this value will determine which avatar is used when the scene is started. Note: this will override the avatar which is attached to the Humanoid Control!

Networking

When Networking Support is set to Unity Networking or Photon Networking, additional entries for the third person avatar will be shown.


Avatar Manager (Script)

Current Avatar Index: 1

#	First Person	Third Person
0	Avatar_MakeHuman_medium_FP	Avatar_MakeHuman_medium_TP
1	Robot Kyle (Animator)	None (Animator)

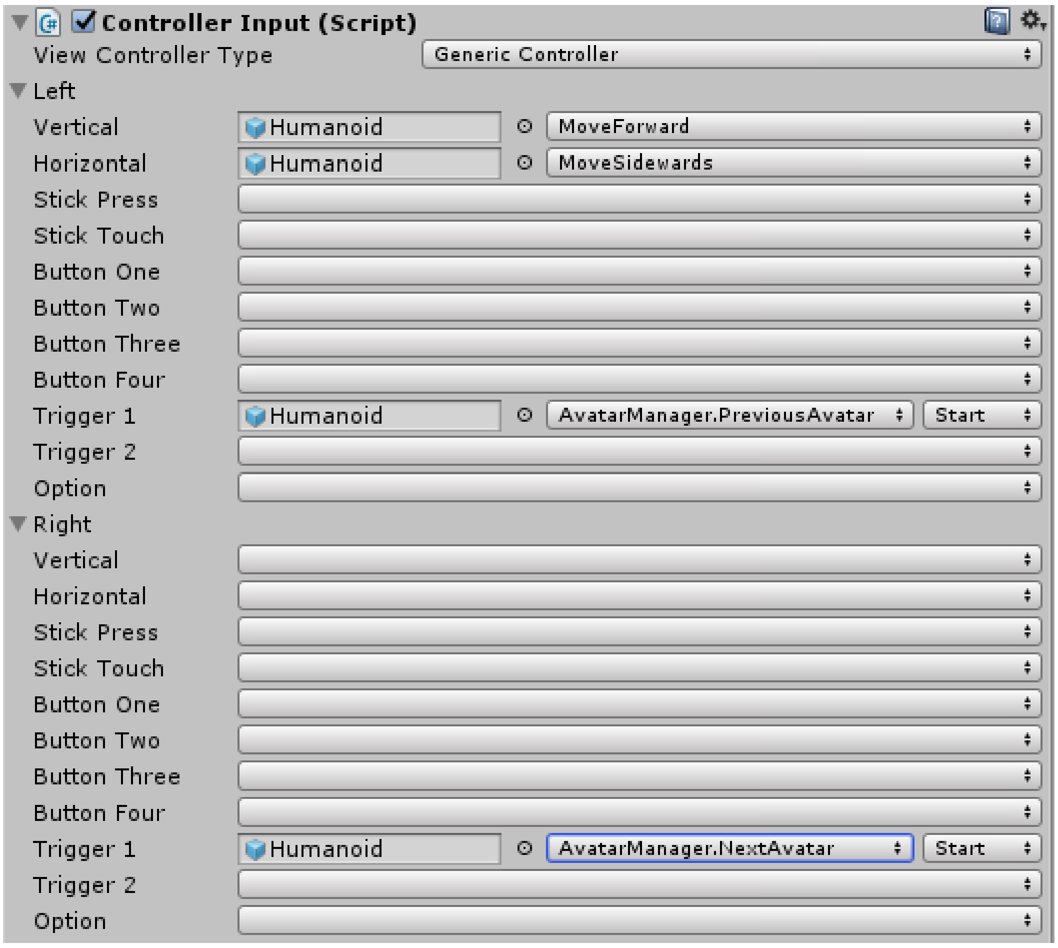
These values match the behaviour of the Remote Avatar in the Humanoid Control script: at the local client, the first person avatar is used, while the third person avatar is used on remote clients. This will enable you to optimize the avatars for each case.

If the Third Person avatar is left to *None*, like for Robot Kyle in the example above, the First Person avatar will also be used for remote clients.

Changing Avatars

The avatar can be set using one of the function described in Scripting.

Especially the Previous/Next Avatar function are suitable to be used in combination with Input like the Controller Input:



Controller Input (Script)

View Controller Type	Generic Controller	
Left		
Vertical	Humanoid	MoveForward
Horizontal	Humanoid	MoveSidewards
Stick Press		
Stick Touch		
Button One		
Button Two		
Button Three		
Button Four		
Trigger 1	Humanoid	AvatarManager.PreviousAvatar — Start
Trigger 2		
Option		
Right		
Vertical		
Horizontal		
Stick Press		
Stick Touch		
Button One		
Button Two		
Button Three		
Button Four		
Trigger 1	Humanoid	AvatarManager.NextAvatar — Start
Trigger 2		
Option		

Teleporter

Humanoid Control comes with built-in teleport support which can be customized using standard Unity solutions. The Teleporter is a specific implementation of an Interaction Pointer.

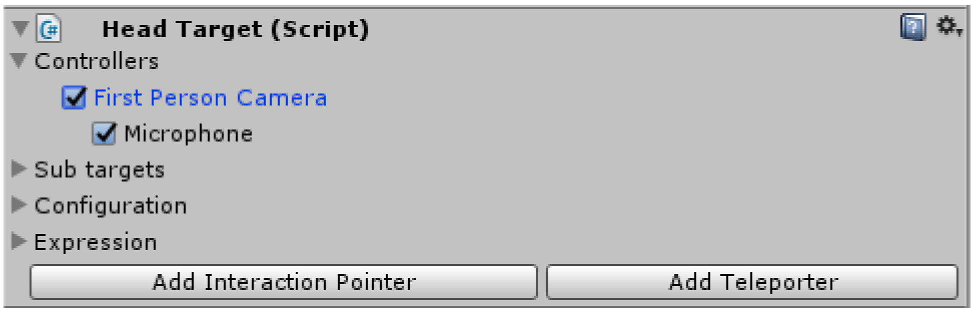
Setup

Two types of teleportation are supported:

1. Gaze based teleportation
2. Hand pointing based teleportation

Gaze Teleportation

You can enable gaze based teleportation on the Head Target of the Humanoid. Here you will find an 'Add Teleporter' button:

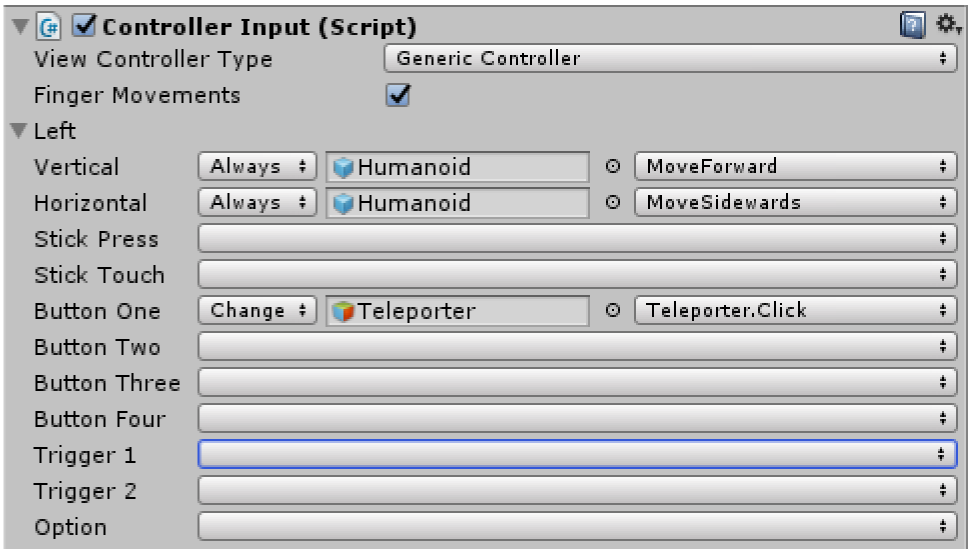


Head Target (Script)
- Controllers
  - ✓ First Person Camera
    - ✓ Microphone
- Sub targets
- Configuration
- Expression

When this button is pressed, a Teleporter GameObject (see below) will be attached to the Head Target. It will be active by default so that you will see the focus point continuously when running the scene.

The focus object is a simple sphere, while no line renderer is present.

Additionally, the Left Button One of the controller will be set to the Click event of the Teleport which will teleport the humanoid to the location in focus:



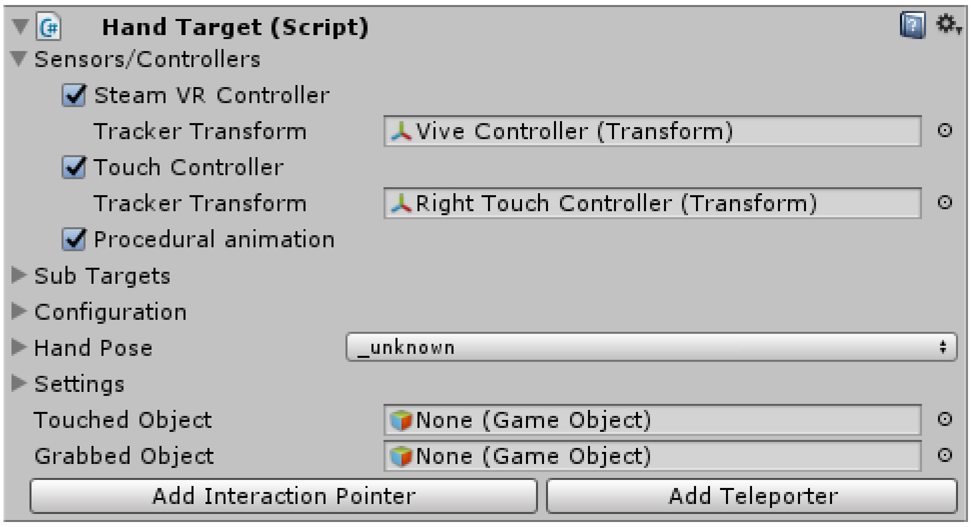
Controller Input (Script) ✓

View Controller Type	Generic Controller		
Finger Movements	✓		
Left			
Vertical	Always	Humanoid	MoveForward
Horizontal	Always	Humanoid	MoveSidewards
Stick Press			
Stick Touch			
Button One	Change	Teleporter	Teleporter.Click
Button Two			
Button Three			
Button Four			
Trigger 1			
Trigger 2			
Option			

The Left Button One will match the Menu Button on the SteamVR Controller and the X button of the Left Oculus Touch Controller.

Pointing Teleportation

Hand pointing based teleportation is activated on either Hand Target. Like above, you will find an 'Add Teleporter' button here:



Hand Target (Script)

Sensors/Controllers

- ✓ Steam VR Controller
  - Tracker Transform: Vive Controller (Transform)
- ✓ Touch Controller
  - Tracker Transform: Right Touch Controller (Transform)
- ✓ Procedural animation

Sub Targets

Configuration

Hand Pose: _unknown

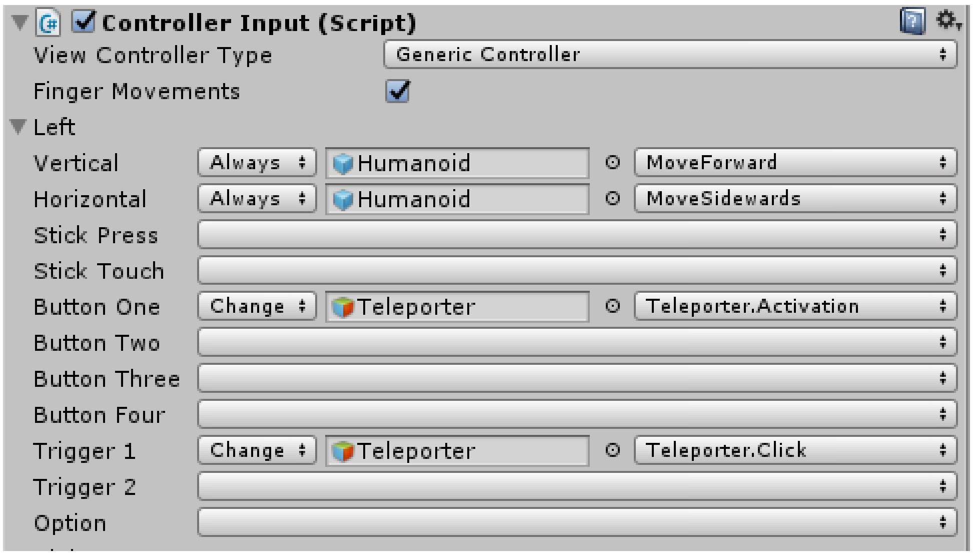
Settings

Touched Object: None (Game Object)

Grabbed Object: None (Game Object)

This will add a Teleporter GameObject to the Hand Target. In this case, no focus object is used, but an line renderer is used to show the pointing direction. This is visible as a standard pink ray on the hand.

Pointing teleporting is activated when the 'One' button is pressed. While the Click event is matched to the Trigger button. The former matches to the Menu Button on the SteamVR controller which the latter is the Trigger on this controller. On the Oculus Touch, the One button is the X or A button, while the trigger button is the Index Finger Trigger button.



✓ Controller Input (Script)

View Controller Type: Generic Controller

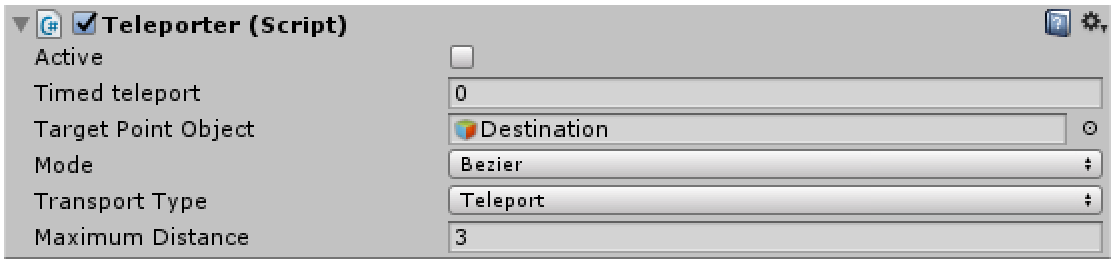
Finger Movements: ✓

Left

Vertical	Always	Humanoid	MoveForward
Horizontal	Always	Humanoid	MoveSidewards
Stick Press			
Stick Touch			
Button One	Change	Teleporter	Teleporter.Activation
Button Two			
Button Three			
Button Four			
Trigger 1	Change	Teleporter	Teleporter.Click
Trigger 2			
Option			

Of course you can change these button assignments through the editing of the Controller Input, setting the desired button to the Teleporter.Activation and -.Click functions.

Configuration



Teleporter (Script)

Active	☐
Timed teleport	0
Target Point Object	Destination
Mode	Bezier
Transport Type	Teleport
Maximum Distance	3

Active	Activates and shows the Target Point Object. It will update the Object's Transform and Line Renderer if available.
Timed Teleport	Automatically teleports after the set amount of seconds. The value 0 disables this function.
Target Point Object	This is the GameObject which represents the target for the teleportation when it is active.
Mode	The ray mode for the pointer. You can choose between a straight line, a bezier curve and a gravity based curve.
Transport Type	Determines how the Transform is moved to the Target Point. Teleport = direct placement to the target point. Dash = a quick movement is a short time from the originating point to the target point.

For more information on these parameters, see the Interaction Pointer.

Target Point Object

The target point object is disabled or enabled automatically when the Teleporter is activates or deactivated.

The Transform of the target point object will be updated based on the ray curve to match the location where the teleport will go. It will be aligned with the Normal of the surface.

This object can be used to show the target of the teleportation in the way you like.

Line Renderer

When an line renderer is attached to the Target Point Object, it will automatically be updated to show the line ray casting curve. You can change this line render to your likings. Only the positions will be overwritten when the teleporter is active.

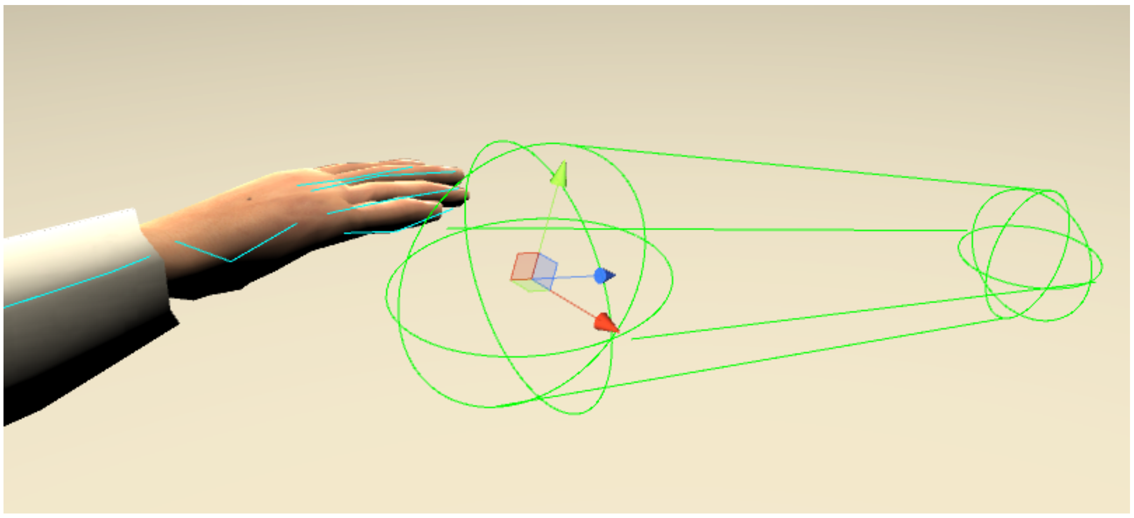
Telegrabber

The Humanoid Telegrabber enables humanoids to grab objects which are normally out of reach for the hands. It is a specific implementation of an Interaction Pointer.

Setup

Telegrabbing can be attached to the Hand Targets. The easiest way to do this is to drop the Telegrabber prefab on either Hand Target.

The location and orientation of the telegrabber needs to be adjusted for the right results. The ray or capsule should not overlap with the hand colliders to prevent grabbing your own hands. This can be achieved by adjusting the position of the Telegrabber Transform. The orienation of the telegrabbing ray can be adjusted by changing the orientation of the Telegrabber Transform.



As the telegrabber is an interaction pointer, it needs to be activated and clicked to work. While it is activated, potential objects to grab are identified by casting a ray or sphere from the hand. When it is clicked, the grabbing takes place. Grabbing can be done automatically at activation by setting the Timed Click value to a small value.

Configuration

As the telegrabber is an Interaction Pointer, the configuration of it is the same as the Interaction Pointer.

Teleport Target

The Teleport Target provides more control to where a player can teleport than the basic Teleporter and can be used in combination with a generic Interaction Pointer.

Setup

The Humanoid Teleport Target can be placed on a static or moving Rigidbody object with a collider. It has been implemented as a Event Trigger and will teleport when an PointerDown event is received.

It has the following parameters:

Transform To Teleport	if it is set this transform will be teleported. If it is not set the transform connected to the Interaction Pointer will be teleported.
Teleport Root	if this is enabled the root of the transform (the topmost transform) will be teleported instead of the transform itself.
Check Collision	if enabled this will check if the location to which the pointer is pointing contains a collider. Teleporting will only take place if no collider has been found. The check is executed using a capsule of 2 meters/units high and 0.2 meters/units radius.
Movement Type	determines how the Transform is moved to the Target Point. Teleport = direct placement a the target point. Dash = a quick movement is a short time from the originating point to the target point.
Pose	is an optional Pose of the Humanoid after it has been teleported. This enables you to teleport to a different pose like a seating pose for instance.
Enable Foot Animator	This will enable or disable the foot animator after teleporting. For poses like seating a walking foot animation is not required. In such cases the foot animator can be switch off with this setting.

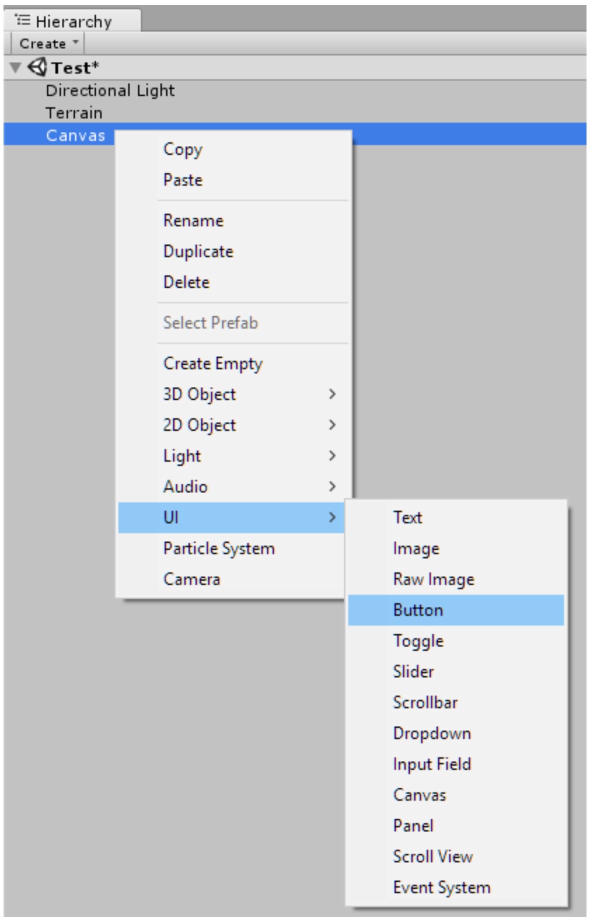
Humanoid Button

Unity provides an great UI system which includes a Button component which can call functions on objects when it is pressed. Humanoid Control supports this in different ways. A limitation is that you cannot determine who has pressed the button which can be useful in multiplayer environments. For this case we provide the Humanoid Button.

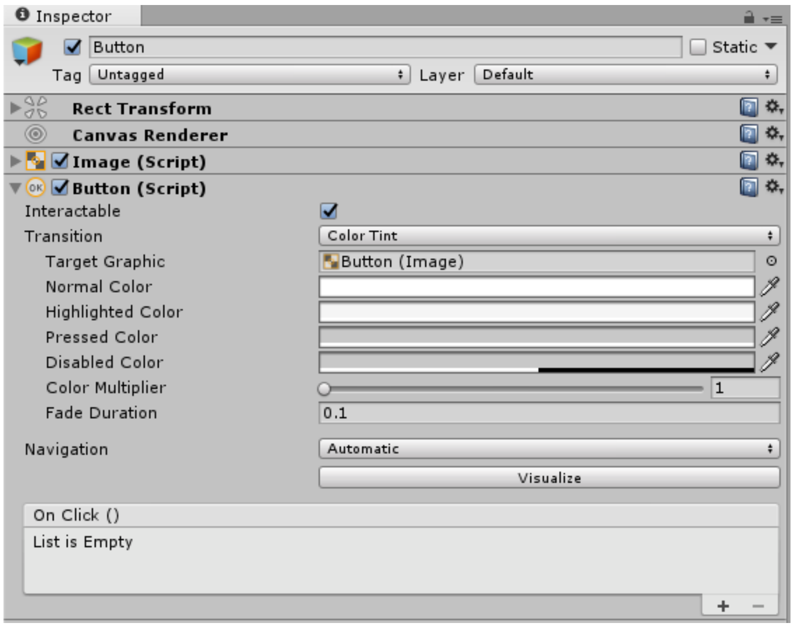
When a Humanoid Button is pressed, a function can be called which takes a HumanoidControl parameter representing the humanoid who pressed the button. This parameter can be used to make the functionality dependent on who pressed the button.

Setup

The easiest way to use the Humanoid Button is to start with a normal UI Button in the scene. This can be done by right clicking in the hierarchy and choosing UI->Button:



Now we get a standard button attached to the Canvas


Inspector

Button — Static

Tag Untagged — Layer Default

Rect Transform

Canvas Renderer

Image (Script)

Button (Script)

Interactable ✓

Transition: Color Tint

Target Graphic: Button (Image)

Normal Color

Highlighted Color

Pressed Color

Disabled Color

Color Multiplier: 1

Fade Duration: 0.1

Navigation: Automatic

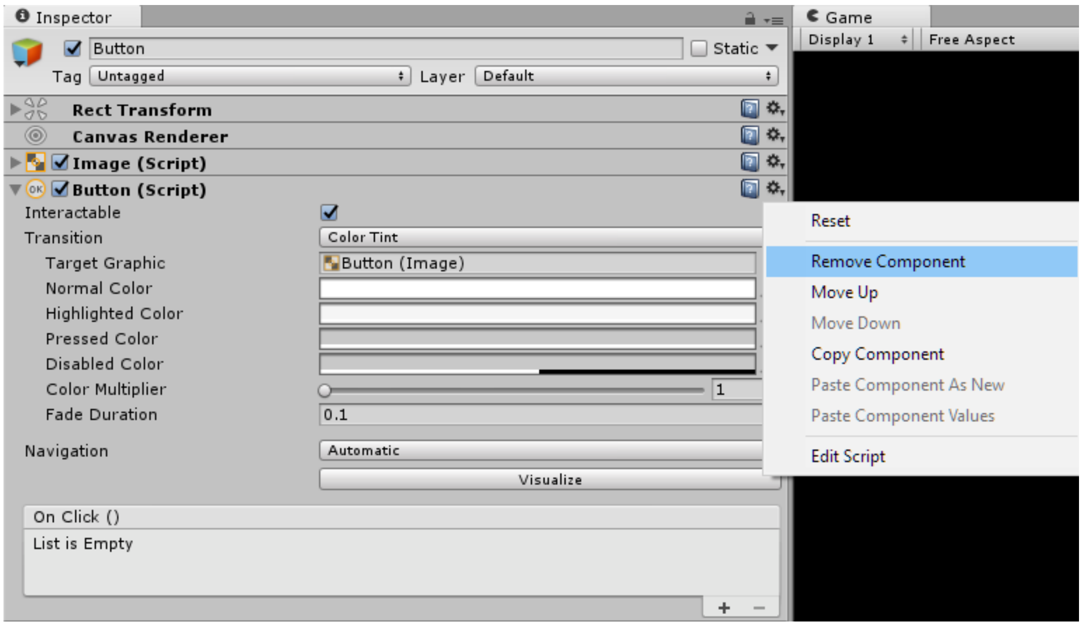
Visualize

On Click ()

List is Empty

At the button you can see the Unity Event which is used to call a function when the button is pressed. From the title bar of this Unity Event you can see that it can only call function without parameters On Click ().

Now we should remove the standard Button from the object using the little 'cog wheel' icon on the right:


Inspector

Button — Static

Tag Untagged — Layer Default

Rect Transform

Canvas Renderer

Image (Script)

Button (Script)

Interactable ✓

Transition: Color Tint

Target Graphic: Button (Image)

Normal Color

Highlighted Color

Pressed Color

Disabled Color

Color Multiplier: 1

Fade Duration: 0.1

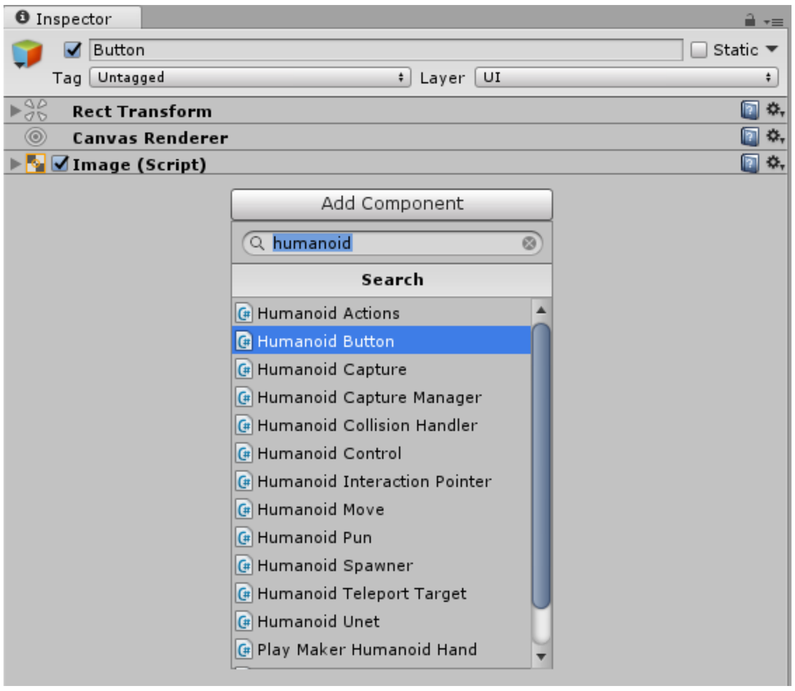
Navigation: Automatic

Visualize

On Click ()

List is Empty

After that, we add the Humanoid Button to replace the standard Button:


Inspector

Button

Static

Tag Untagged

Layer UI

Rect Transform

Canvas Renderer

Image (Script)

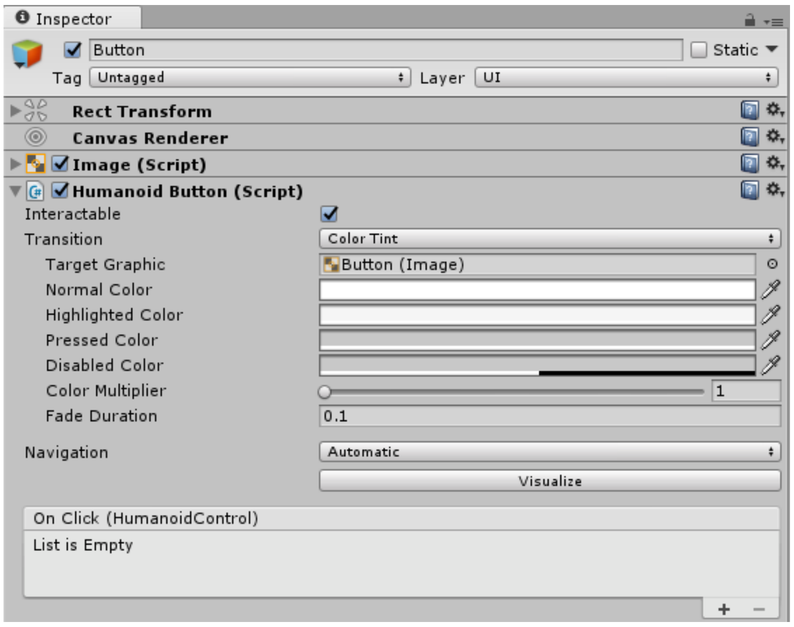
Add Component

humanoid

Search

- Humanoid Actions
- Humanoid Button
- Humanoid Capture
- Humanoid Capture Manager
- Humanoid Collision Handler
- Humanoid Control
- Humanoid Interaction Pointer
- Humanoid Move
- Humanoid Pun
- Humanoid Spawner
- Humanoid Teleport Target
- Humanoid Unet
- Play Maker Humanoid Hand

You'll see that it is almost the same as the normal Button, so we can reuse everything we know about buttons. The different lies in the On Click event:


Inspector

Button

Static

Tag Untagged

Layer UI

Rect Transform

Canvas Renderer

Image (Script)

Humanoid Button (Script)

Interactable	✓
Transition	Color Tint
Target Graphic	Button (Image)
Normal Color	
Highlighted Color	
Pressed Color	
Disabled Color	
Color Multiplier	1
Fade Duration	0.1
Navigation	Automatic

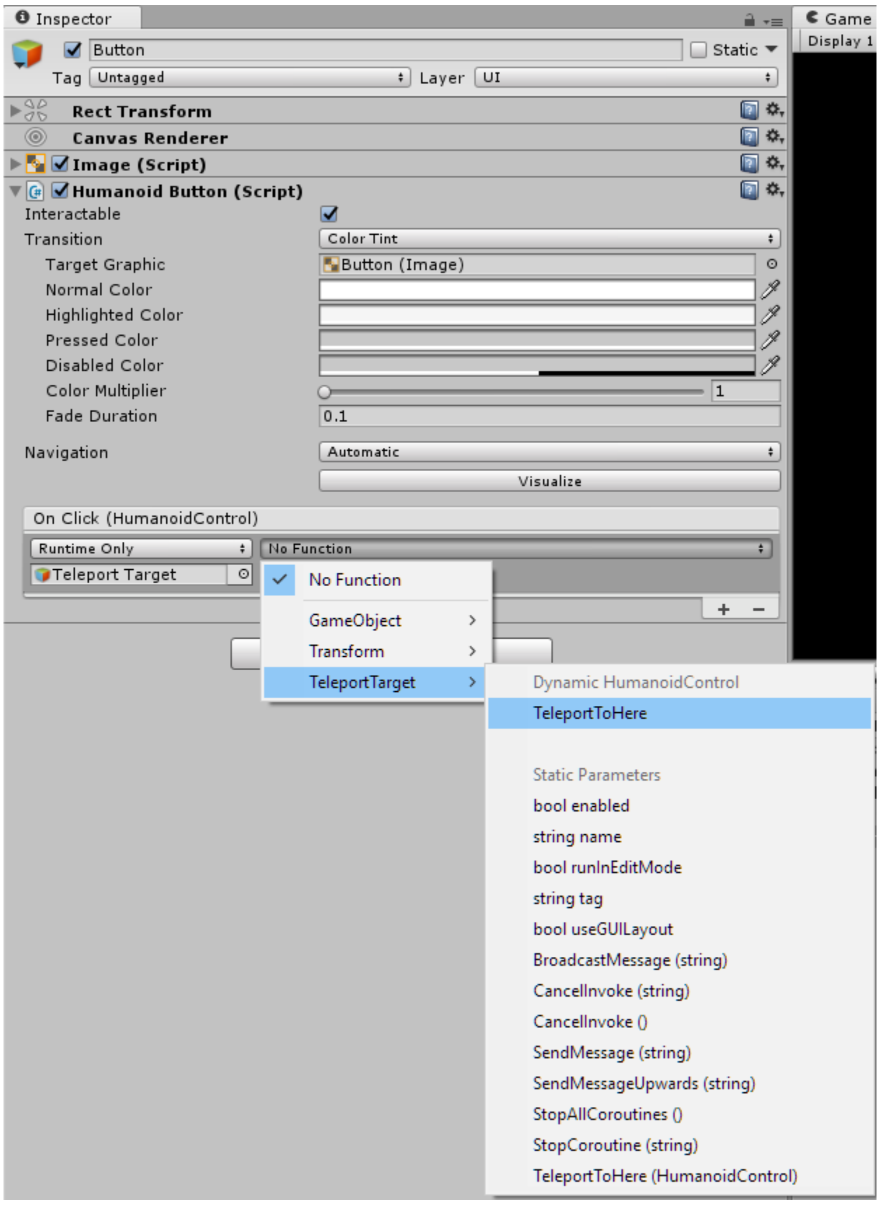
Visualize

On Click (HumanoidControl)

List is Empty

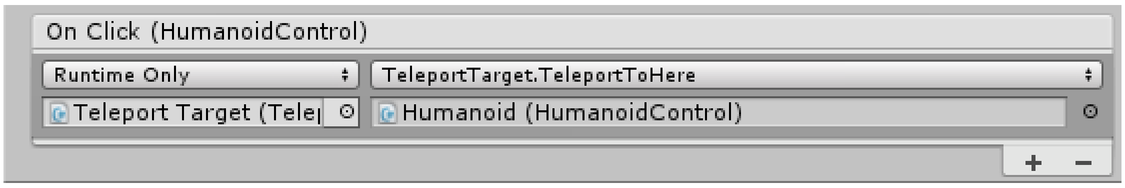
In the Humanoid Button, the On Click event has a HumanoidControl parameter. This can be used to call functions which take a HumanoidControl parameter.

For example, we can now call the TeleportToHere(HumanoidControl humanoid) function on a Teleport Target. This will have the effect that the humanoid who pressed the button will be teleported to the location of the Teleport Target.


Inspector
Button
Static
Tag Untagged
Layer UI
Rect Transform
Canvas Renderer
Image (Script)
Humanoid Button (Script)
Interactable ✓
Transition Color Tint
Target Graphic Button (Image)
Normal Color
Highlighted Color
Pressed Color
Disabled Color
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
On Click (HumanoidControl)
Runtime Only
No Function
Teleport Target

You see that for a Teleport Target, you can now choose the Dynamic HumanoidControl-TeleportToHere. Choosing this option will teleport the humanoid who pressed the button to the location of the Teleport Target.

At the bottom you will also see the Static Parameters-TeleportToHere (HumanoidControl). With this option the humanoid who pressed the button will be ignored and you can select explicitly which humanoid will be teleported. This humanoid can thus be different from the humanoid who pressed the button.


On Click (HumanoidControl)
Runtime Only
TeleportTarget.TeleportToHere
Teleport Target (Tele
Humanoid (HumanoidControl)

Note, this last option is also possible with the standard Button.

Interaction Pointer

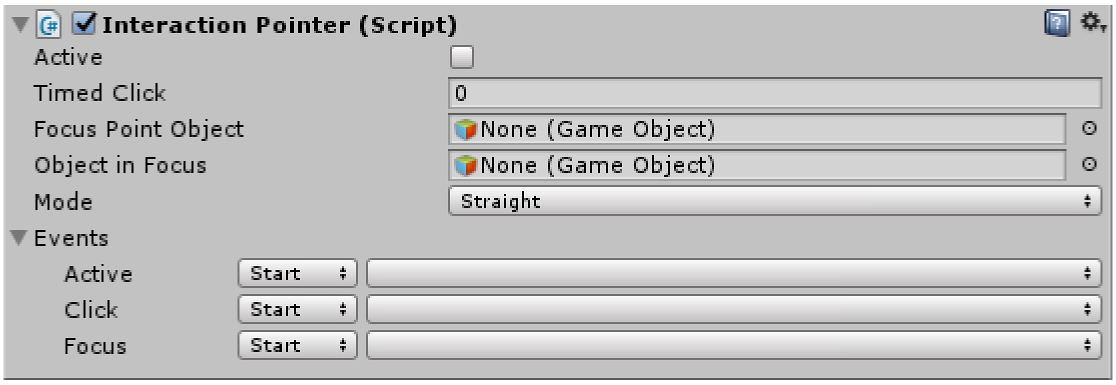
→ Interaction Pointer API

The interaction pointer is a generic pointer to interact with objects in the scene using the Unity Event system. The objects can receive and react on these interactions when an Unity Event Trigger component has been added to them. The Humanoid Teleporter is an specific implementation of an Interaction Pointer.

Setup

An interaction pointer can be added to any object. The objects Transform determines the starting position of the pointer and the pointer will always point in the direction of the Transform Forward.

Configuration



Interaction Pointer (Script)

Active	☐
Timed Click	0
Focus Point Object	None (Game Object)
Object in Focus	None (Game Object)
Mode	Straight
Events	
Active	Start
Click	Start
Focus	Start

Active	Activates and shows the Focus Point Object. It will update the Focus Point Object's Transform and Line Renderer if available.
Timed Click	Automatically initiates a click when an Object is in focus for the set amount of seconds. The value 0 disables this function.
Focus Point Object	this is the GameObject which represents the focus point of the Interaction Pointer when it is active.
Object in Focus	this is the object to which the Interaction Pointer is pointing at. Is updated at runtime.
Mode	The ray mode for the pointer. See below for more information.

Straight mode

The straight mode will cast a ray from the Transform position in the forward direction until it hits an object.

Bezier mode

In bezier mode, a curve will be cast from the Transform position in the forward direction. The distance reached by this curve is limited. The curve is determine by the following parameters:

Maximum Distance	This determines the maximum distance in units you can reach with the curve.
Resolution	The number of segments of the curve. Higher gives are smoother curve but requires more performance.
Maximum Distance	This determines the maximum distance in units you can reach with the curve.
Resolution	The number of segments of the curve. Higher gives are smoother curve but

	requires more performance.

Gravity mode

In gravity mode, a curve will be cast from the Transform position in the forward direction. This curve follows a gravity path as if an object is launched. The curve can be configured with the following parameters:

Maximum Distance	The maximum length of the curve in units.
Resolution	The number of segments of the curve. Higher gives are smoother curve but requires more performance.
Speed	the horizontal speed determining the curve. A higher value can reach further positions.

Sphere mode

Is similar to the Straight mode, but casts a sphere instead of a ray. It has the following configuration parameters:

Maximum Distance	The distance the sphere is cast.
Radius	The radius of the sphere.

Input

The interaction pointer has three input events which can be used to call functions when the event occurs. More information on how to use this can be found on the Input page.

Active	this function is called based on the value of the Active parameter of the Interaction Pointer.
Click	this function is called based on the status of the click.
Focus	this function is called based on the object in focus.

Focus Point Object

The Focus Point Object is disabled or enabled automatically when the Interaction Pointer is activated or deactivated.

The Transform of the target point object will be updated based on the ray curve to match the focus point of the Interaction Pointer. It will be aligned with the normal of the surface.

This object can be used to show the focus point of the interaction pointer in the way you like.

Line Renderer

When an line renderer is attached to the Focus Point Object, it will automatically be updated to show the line ray casting curve. You can change this line render to your likings. Only the positions will be overwritten when the Interaction Pointer is active.

Events

The following event of the Event System are currently supported:

PointerDown	is sent to the object in focus when the click starts
PointerUp	is sent to the object in focus when the click ends

PointerClick	is sent to the object when a click has been completed
PointerEnter	is sent to an object when it comes into focus of the Interaction Pointer
PointerExit	is sent to an object when it no longer is in focus of the Interaction Pointer
BeginDrag	is sent to the object in focus when a click has started and the focus point starts to move
Drag	is sent to the object in focus while the focus point changes during a click
EndDrag	is sent to the object in focus when the click ends after the focus point started to move

Functions

void Activation(bool active)	can be used to activate the Interaction Pointer.
void Click(bool clicking)	can be used to change the click state of the Interaction Pointer.

Handle

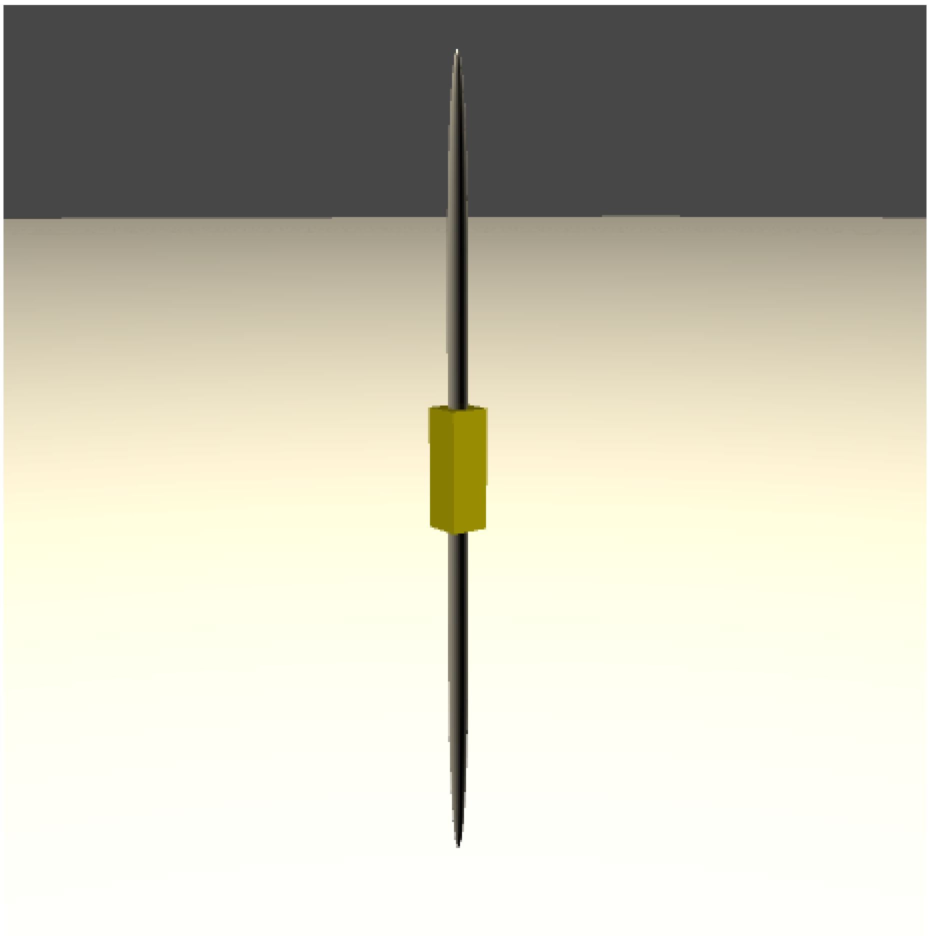
→ Handle API

An Handle can be used to give direction on how an object can be grabbed. A sword is usually grabbed by the hilt, a gun by the grip.

When a Rigidbody with a Hadle is grabbed, the Rigidbody will move into the hand such that the Handle will fit in the palm of the hand.

When a static object with a Handle is grabbed, the hand itself will move such that the Handle fits in the palm of the hand.

The handle is visualized in the scene view using a yellow cuboid.



Setup

An handle has the following properties:

Position	The local position of the handle.
Rotation	The local rotation in euler angles of the handle.
Hand	Sets which hand can grab the handle. Some handles may only be grabbed by the left or right hand.
Grab Type	The *Default Grab* is the *Bar Grab*. If *No Grab* is selected the object cannot be grabbed.
Range	The range within the handle will work. Outside this range normal grabbing is used.
Hand Pose	The Hand Pose which will be active while the Handle is grabbed.
Hand Target	When the Handle is grabbed this will contain the Hand Target of the grabbing hand. When the Hand Target of a Handle is set in the editor while editing the scene the applicable hand will try to grab the Handle.

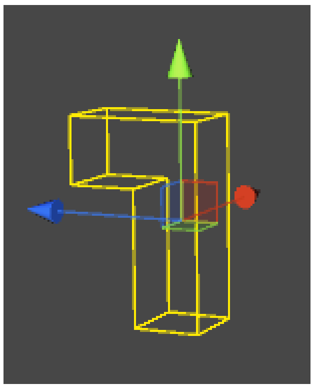
Sockets

→ Socket API

Sockets are GameObjects which can hold Handles.

Gizmo

A socket will be visible in the Scene View when selected as a wire version of the Handle Gizmo:



Attaching and Releasing

Attaching Handles using a Trigger Collider

When an trigger collider has been attached to the GameObject with the Socket an Handle will be attached to the Socket when it enters the trigger collider.

Attaching Handles using a Normal Collider

When a normal collider has been attached to the HameObject with the Socket a Handle will be attached to the Socket when it collides with the collider.

Attaching Handles using Function Calls

You can attach handles to the Socket GameObject by calling an Attach() function on the Socket. This will try to attach the Transform, Rigidbody or Handle parameter to the Socket. On the other hand you can release Handles again by calling an Release() function on the Socket. This will release the currently attached Handle from the Socket.

Attaching Handles Prefab in the Editor

It is possible to attach a Handle prefab to a socket in the Unity editor. In this way the socket will start with a Handle attached.
This is done by assigning a Prefab to the *Attached Prefab* field of the Socket. This will automatically create an instance of the prefab which is attached to the socket.
When the *Attached Prefab* field is cleared again, the prefab instance attached to the socket is released and destroyed again.

Releasing Handles to another Socket

When an handle is attached to a socket and the Handle is attached to another socket it will be automatically released from the original socket.

The Socket Tag

With the Socket Tag field to is possible to limit the Handles which can attach to this socket to those who have the right Tag. This construction is using the standard Unity Tags.

Events

The Socket has one Event Handler which can be used to call function based on the attachment status.

The Attachment Implementation

Depending on the type of socket and its GameObject handles are attached to sockets in various ways.

A Static Socket...

... with a Non-Kinematic Rigidbody Handle

A non-kinematic Rigidbody will get a Joint which limits its movement to the location and orientation of the Socket. This joint is implemented using a Configurable Joint.
When the Handle is released, the joint is destroyed again.

... with a Kinematic Rigidbody Handle

A kinematic Rigidbody with an handle will the parented to the static Socket Transform. The result will be that the Transform of the Rigidbody is a child of the Socket Transform.
The Rigidbody will be temporarily removed. The settings will be conserved in a RigidbodyDisabled Component which is used to restore the Rigidbody when the Handle is released from the Socket.

... with a Static Handle

Static Handles cannot be attached to static Sockets. This will lead to a warning message in the Console: “Cannot attach static handle to static socket”.

A Kinematic Rigidbody Socket...

... with a Non-Kinematic Rigidbody Handle

If the Non-Kinematic Rigidbody Handle has one or more Joints or when it has constraints set, it will be attached to the Socket using a Joint. This joint is implemented using a Configurable Joint.
When the Handle is released, the joint is destroyed again.
In all other case, the non-kinematic Rigidbody will the parented to the kinematic Rigidbody

Socket Transform. The result will be that the Transform of the Rigidbody is a child of the Socket Transform.
The Rigidbody will be temporarily removed. The settings will be conserved in a RigidbodyDisabled Component which is used to restore the Rigidbody when the Handle is released from the Socket.

... with a Kinematic Rigidbody Handle

A kinematic Rigidbody with an handle will the parented to the kinematic Rigidbody Socket Transform. The result will be that the Transform of the Rigidbody is a child of the Socket Transform.

... with a Static Handle

In this case, the kinematic Rigidbody Socket is parented to the static Handle Transform. The result will be that the Transform of the Scoket is a child of the Handle Transform.
The Rigidbody of the Socket will be temporarily removed. The settings will be conserved in a RigidbodyDisabled Component which is used to restore the Rigidbody when the Handle is released from the Socket.

A Non-Kinematic Rigidbody Socket...

... with a Non-Kinematic Rigidbody Handle

If the Non-Kinematic Rigidbody Handle has one or more Joints or when it has constraints set, it will be attached to the Socket using a Joint. This joint is implemented using a Configurable Joint.
When the Handle is released, the joint is destroyed again.
In all other cases, the non-kinematic Rigidbody will the parented to the kinematic Rigidbody Socket Transform. The result will be that the Transform of the Rigidbody is a child of the Socket Transform.
The Rigidbody will be temporarily removed. The settings will be conserved in a RigidbodyDisabled Component which is used to restore the Rigidbody when the Handle is released from the Socket.

... with a Kinematic Rigidbody Handle

A kinematic Rigidbody with an handle will the parented to the non-kinematic Socket Transform. The result will be that the Transform of the Rigidbody is a child of the Socket Transform.
The Rigidbody will be temporarily removed. The settings will be conserved in a RigidbodyDisabled Component which is used to restore the Rigidbody when the Handle is released from the Socket.

... with a Static Handle

In this case, the socket Rigidbody will be attached to the Handle using a Joint. This joint is implemented using a Configurable Joint such that it will follow the Handle’s Transform as closely as possible.
When the Handle is released, the joint is destroyed again.

Mechanical Joints

[→ Mechanical Joint API]

Mechanical Joints can be used to limit the movements of Rigidbody in local space. It is primarily intended for kinematic Rigidbodies.

Setup

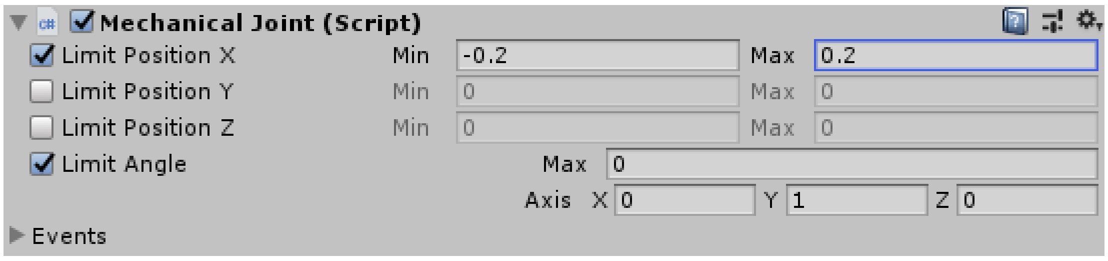
The Mechanical Joint component should be placed on a GameObject with a Rigidbody with the *isKinematic* propery enabled.

Kinematic Rigidbodies will not move in the scene under influence of physics, only when they are moved by scripting or grabbed by an humanoid.

The Mechanical Joint component can limit these movements.

All limitations work in local space and are relative to the local location of the object at compile time.

Parameters



Mechanical Joint (Script)

✓ Limit Position X	Min	-0.2	Max	0.2		
Limit Position Y	Min	0	Max	0		
Limit Position Z	Min	0	Max	0		
✓ Limit Angle	Max	0				
	Axis	X 0	Y 1	Z 0		

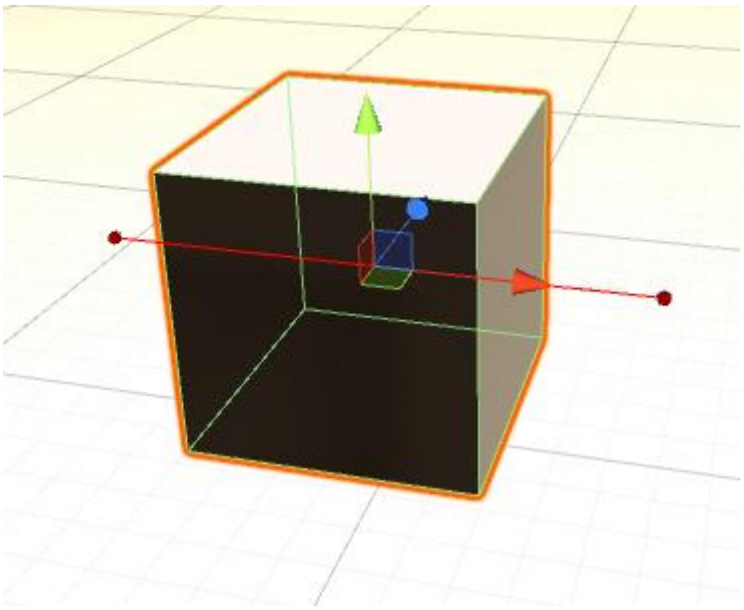
Events

Limit Position X	Limits the local X-position to the given minimum and maximum bounds. When the bounds are both zero the local X position is locked.
Limit Position Y	Limits the local Y-position to the given minimum and maximum bounds. When the bounds are both zero the local Y position is locked.
Limit Position Z	Limits the local Z-position to the given minimum and maximum bounds. When the bounds are both zero the local Z position is locked.
Limit Angle	Limits the local relative to the local rotation at compile time. When the angle is zero the local rotation is locked. The axis is the axis around which the rotation is possible.

Scene

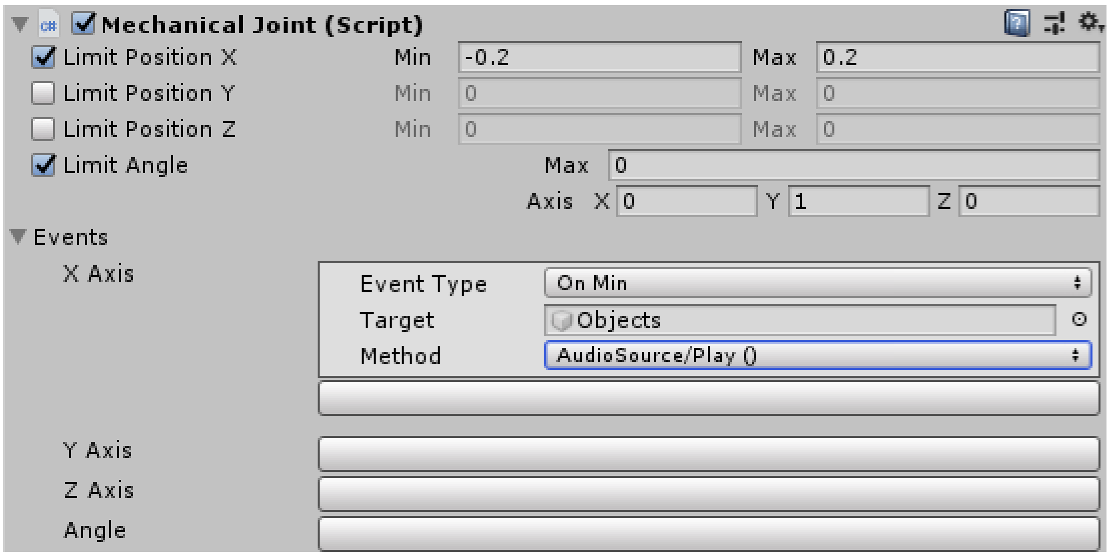
The Mechanical Joint limitations will be shown in the scene using lines capped with spheres in the colour of the axis.

An example of a cube with a limitation on the X axis of 0.2 units to both sides:



Events

The Mechanical Joint component has events for executing functions based on the limitations. These work just like any other event:



Mechanical Joint (Script)

✓ Limit Position X	Min	-0.2	Max	0.2
Limit Position Y	Min	0	Max	0
Limit Position Z	Min	0	Max	0
✓ Limit Angle			Max	0
	Axis	X 0	Y 1	Z 0

Events

X Axis

Event Type	On Min
Target	Objects
Method	AudioSource/Play ()

Y Axis

Z Axis

Angle

In the example above, a sound will play when the local X position of the Rigidbody will reach the minimum value. In this case -0.2. Multiple events can be attached to the limitation to provide many possibilities.

Examples

The following instructions use Mechanical Joints:

A Gun with a Sliding Barrel

A Pump-Action Gun