Humanoid Control for Unity

AUTHOR
Version v4
Thu Aug 18 2022

Table of Contents

Table of contents

Humanoid Control for Unity

Author

PasserVR

Version

4.0

Note: this documentation is still work in progress...

Main components

- Humanoid Control
- Humanoid Site
- Humanoid Visitor

Devices

- UnityXR
- Oculus
- Leap Motion
- Vive Tracker
- Perception Neuron

Input

- ControllerInput
- InteractionPointer

Generic tools

- TeleportTarget
- Socket
- Handle

Networking

- NetworkingStarter

Quick start

If you want to contribute to this project, make sure you have access to the repository. You can request access by logging into our GitLab site with an GitHub or Google account. Besides that, it is best to contract us about you wanting to contribute to this project. This will ensure you will get access more quickly.

Installation

Import this package in Unity directly with the Package Manager git package importer.

See Unity: Installing from a Git URL

Use the link from 'Clone with HTTP' (for example: https://http://gitlab.passervr.com/passer/unity/humanoidcontrol4_free.git) In this way you can always retrieve the latest version by pressing the `Update` button in the Package Manager.

Optionally, you can use a tag to retrieve a specific version. For example:http://gitlab.passervr.com/passer/unity/humanoidcontrol4_free.git#4.1.0. This will give you a stable version which does not change. Updating can be done by retrieving the package with a link to a new release.

ChangeLog and Releases

You will find all releases with UnityPackages and links for the Unity Package Manager in the Releases

Documentation

See PasserVR HumanoidControl Documentation

Video

https://passervr.com/PasserVR/videos/HC4Free.mp4

Extensions and tracking

Version

Humanoid Control 4

Note: this documentation is still work in progress...

The Humanoid Control framework allows you to add support for additional devices which are not supported in the main packages. This document decribes how this can be done.

Basic architecture

Every extension consists of two main parts:

1. The tracking device itself
2. The projection of the tracking device on the humanoid This chapter will decribe the principles for thes two parts:

The tracking device

All tracking devices are represented as GameObject in the Real World of the Humanoid. Most tracking devices consist of multiple parts. Usually we have one Tracker and multiple Sensors

Tracker

The tracker GameObject is the representation of the full tracking device and its Transform is the origin of the tracking space. Examples of this is the Kinect Camera itself, the origin of the OpenVR tracking space or the BaseStation of the Razer Hydra. Every tracker GameObject should have a specific implementation of a Tracking::TrackerComponent attached. Examples these tracker components are Tracking::AzureKinect, Tracking::OpenVR and Tracking::HydraBaseStation.

Sensor

Each tracker GameObject can have multiple Sensor children. A Sensor represents a tracking point within the tracking space around the tracker. Examples of these are a tracked hand from the Kinect Camera, an OpenVR Hmd or an Razer Hydra Controller. Every sensor GameObject should have a specific implementation of a Tracking::SensorComponent attached. Examples of these sensor components are Tracking::OpenVRHmd and Tracking::HydraController. The Kinect does not have specific sensor children, but uses Tracking::TrackedBone for skeletal tracking.

Note: The TrackerComponent and SensorComponent are still simple. In the future they are expected to be extended to simplify the implementation of new custom extsnsions.

The project of the tracking device on the humanoid

Like with the tracking device we have a similar setup for the project on the humanoid.

Humanoid::HumanoidTracker

This implements the projection for the humanoid as a whole. It has references to HumanoidSensors for the different parts of the body: head, hands, hips and feet For every extension, an specific implementation of the Humanoid::HumanoidTracker should be created.

Examples of this are the Humanoid::KinectTracker, Humanoid::OpenVRHumanoidTracker and Humanoid::HydraTracker.

Humanoid::HumanoidSensor

The humanoid sensor takes care of the projection of the sensor to the specific bone in the targets rig of the Humanoid. Most sensors will project to a fixed bone in the rig. For example a VR HMD will always project onto the head bone of the target rig. But sometimes it is possible to select to which bone the sensor will be projected. An example of this is the Humanoid::ViveTrackerArm where you can choose to put it on the hand, forearm, upper arm or shoulder in the rig. Exmples of humanoid sensors are Humanoid::Kinect4Arm, Humanoid::OpenVRHmd and Humanoid::RazerHydraHand.

The rigs

Humanoid Control uses two rigs: the tracking rig and the avatar rig.

Avatar rig

The avatar rig is the rig of the avatar as you know it: the pose of this rig will determine the pose of the avatar how you will see it. When you change the position of the hand for example, the avatar's hand will follow that hand directly.

Tracking rig

The tracking rig is the target pose which is derived from the tracking devices. Humanoid Control will try to align the avatar rig to the tracking rig using forward and inverse kinematics. Note however that differeces will appear between the two rig in many situations. Most important are:

- Physical limitations of the avatar: for example the arms of the avatar are not long enough to reach the hand target or the target rig will result in unnatural poses.
- Physics: for example the hand target is inside a wall and Huamnoid::HumanoidControl::physics is enabled the avatar's hand will collide with the wall preventing the hand to reacht the target.

The tracking rig has actually two levels of detail:

- The target rig This is a complete rig for all supported bones in the humanoid. It is often visible in the Unity Inspector as a TargetRig GameObject.
- The targets. There are 6 main targets: Head Target, Hips Target, 2 Hand Targets and 2 Foot Targets. These targets are often directly visible in the Unity Inspector.

The targets will actually match the transforms of the matching bones in the target rig. For example the head target will have the same position and rotation as the head bone in the target rig.

Tracking confidence

A key component in the use of multiple tracking devices simultaneously is the tracking confidence. Every sensor component will have a specific quality in tracking. For example the Kinect has good positional tracking quality but rotations are of lower quality. This is because of the Kinect technology using a depth camera. On the other hand Perception Neuton has good rotational quality but not so good positional quality. This is because Perception Neuron uses IMUs (rotational sensors) to determine the pose of the user. To cope with this and to be able to merge the data coming from different tracking devices we use tracking confidence.

Every bone in the target rig has a positionConfidence and rotationConfidence. When an Humanoid::HumanoidSensor updates the position and rotation of a bone in the target rig, it should also update the positionConfidence and rotationConfidences of that bone. The basic rule with this is: when the sensor data has a higher confidence than the target bone, it will update the position and/or rotation and update the confidence of that bone. The tracking confidence is a float in the range 0..1 where 1 represents an 100% certainty that the bone is at that position. There are no hard rules about these values, but we use 0.2 for animated bones (no tracking), 0.5 for fair tracking or values derived from other tracking data. 0.9 for good tracking and 0.99 for excellent tracking. The value 1 is never use because then we have no possibility to override it if this is required.

Visitor

Version

Humanoid Control 4

A visitor is a special Scene with a Humanoid which can be used to visit a Site.

More documentation will follow...

Android (Including Oculus Quest)

The application should have internet access permission. So the following line should be added to the AndroidManifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Third-Party-Notices

This asset is governed by the Asset Store EULA; however, the following components are governed by the licenses indicated below:

Component Name: MakeHuman Characters

License Type: "CC0-1.0"

MakeHuman License

Component Name: Humaonid Control Free

License Type: "MPL-2.0"

Humanoid Control Free License

HTC Vive Tracker

Author

Passer

Version

4

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

Prerequisites

Vive Trackers are supported in the Humanoid Control Plus and Pro packages.

Hardware

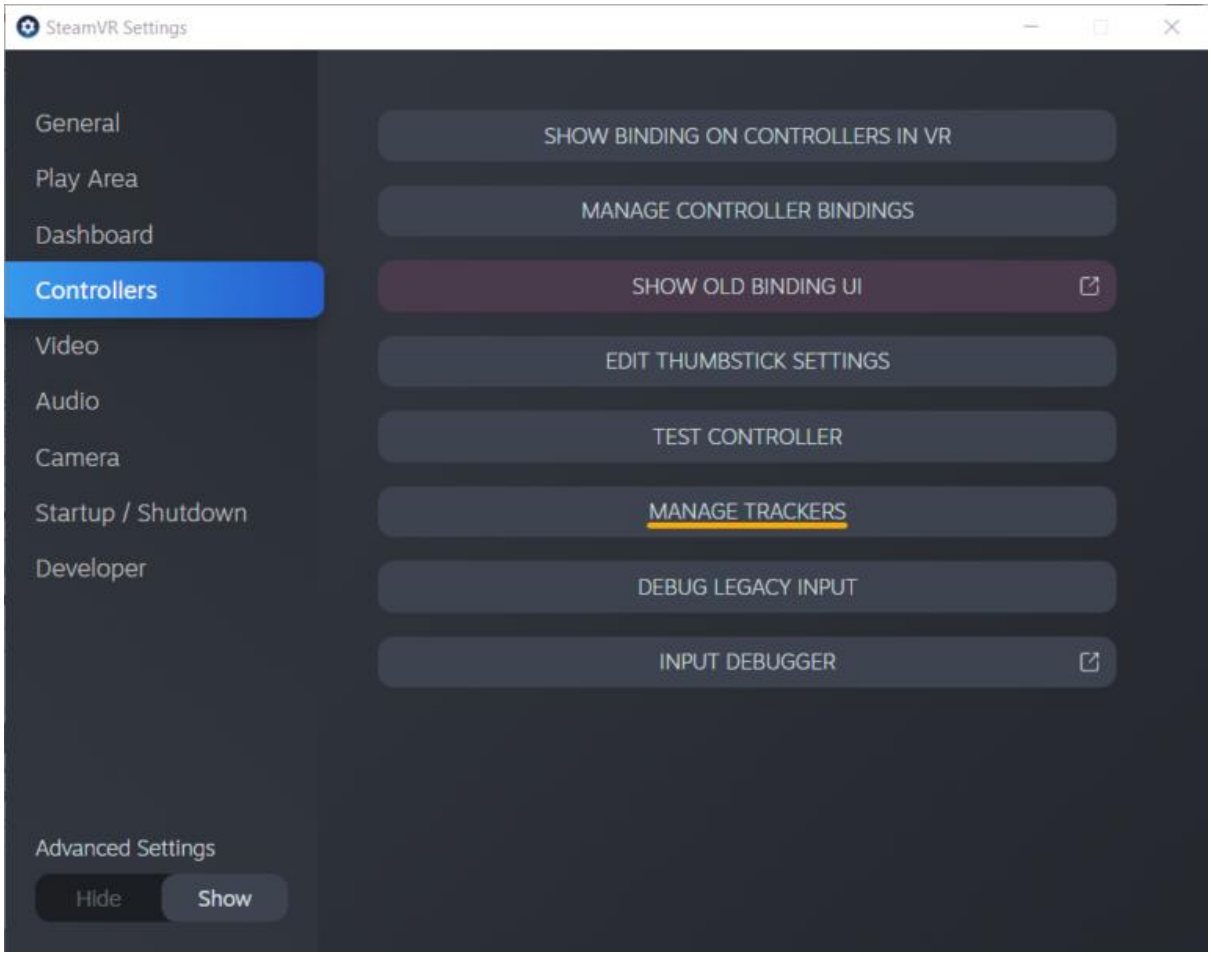
Vive Tracker v1.0 - v3.0 are supported

Operating system

Vive Trackers are supported on the Microsoft Windows platform.

Setup

In Humanoid Control v4, the Vive Tracker Role can be set to assign a Vive Tracker to a specific body part. This can be set in the SteamVR Settings. In the Controller section, click on the *Manage Trackers* button:

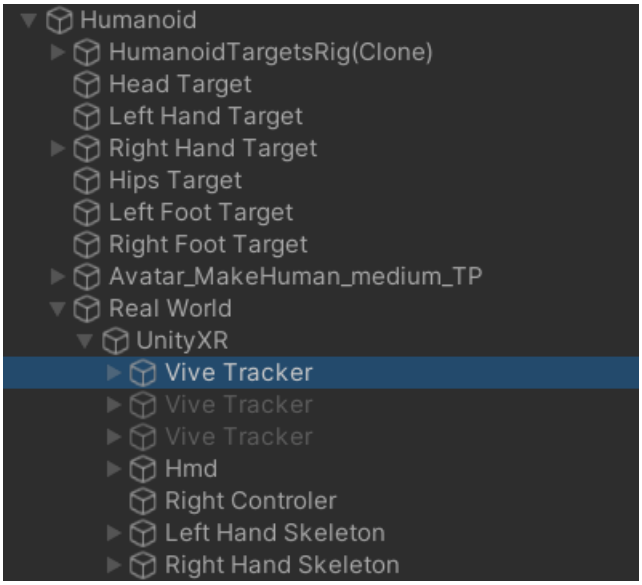


The following Roles are supported:

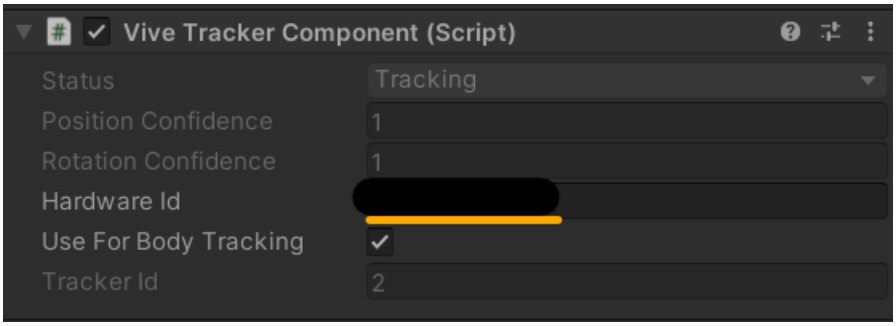
- Held In Hand/Left Hand = Left Hand Target
- Held in Hand/Right Hand = Right Hand Target
- Chest(option Head seems to be missing...) = Head Target
- Waist = Hips Target
- Left Foot = Left Foot Target
- Right Foot = Right Foot Target

Manual tracker location assignment

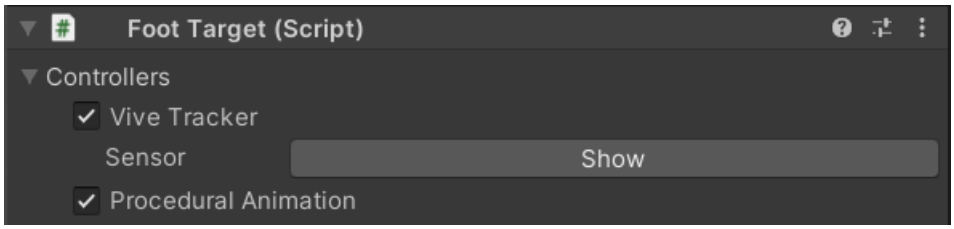
It is possible to assign specific trackers to certain body parts maually. To do this you need to know the tracker hardware Id of each tracker. You can find these in the in the SteamVR settings: see Setup above, you can find them in the Manage Trackers section. You can also find these by running the scene. All active trackers should appear in the scene and can be found as active Vive Trackers in the humanoid/Real World/UnityXR object in the hierarchy:



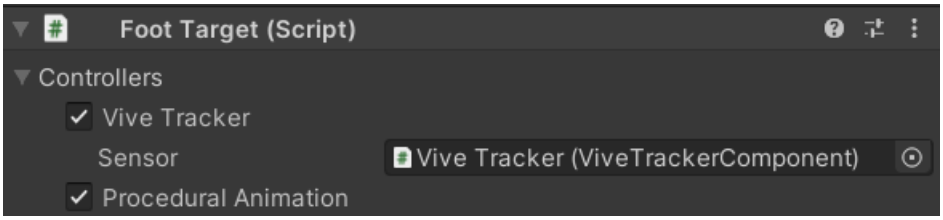
Then you can find the Hardware Id in the Vive Tracker Component of that object.



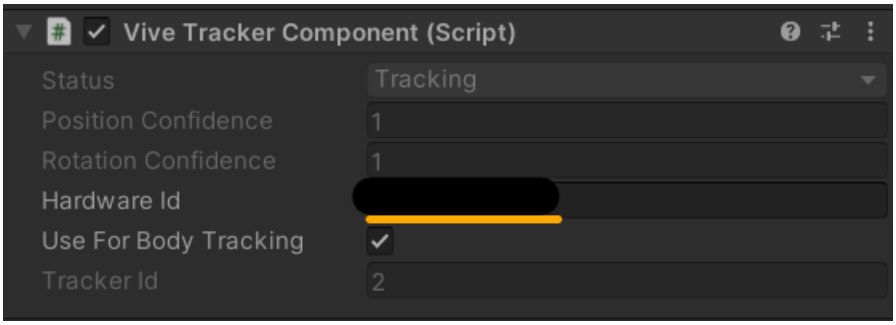
Now we can assign this tracker to the right part of the body. To do this, the tracker should first be made visible in the Scene by pressing the *Show* button for the Vive Tracker on the applicable Target:



Double click on the *Vive Tracker (Transform)* to open the tracker assigned to that target:



On this Vive Tracker Component, you can now fill in the Hardware Id of the Vive Tracker you want to use for this target:

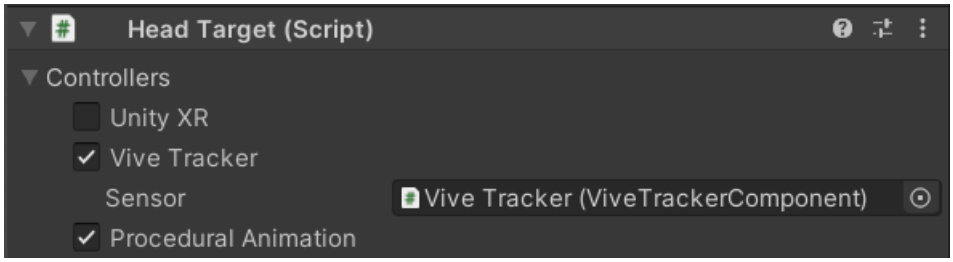


Inspector

- Tracker Transform: a reference the the object representing the real world Vive tracker.

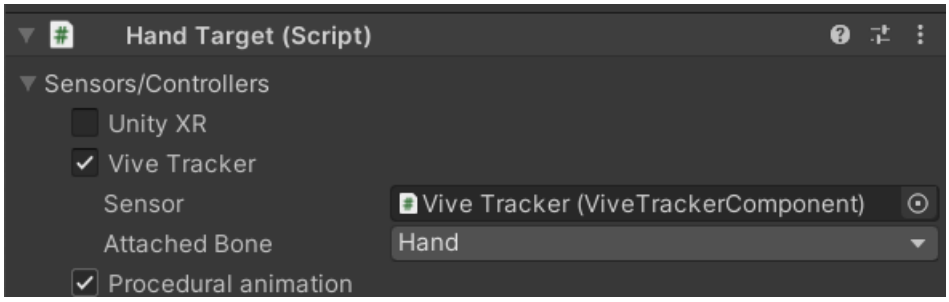
HeadTarget

A Vive Tracker can be used on the head instead of an HMD. There is no need to have a UnityXR controller or any camera at all in the scene. It is not possible to have both a Unity XR and Vive Tracker enabled on the head.



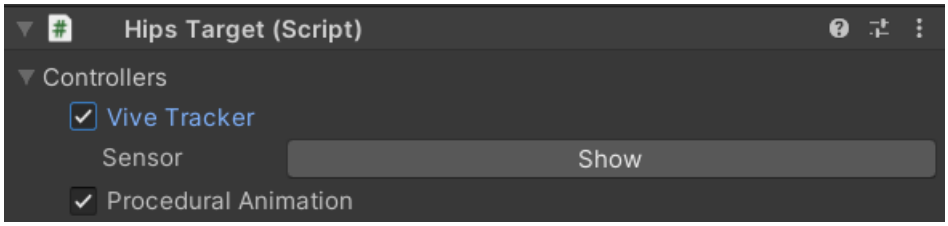
HandTarget

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 UnityXR controller and a Vive Tracker on the hand.



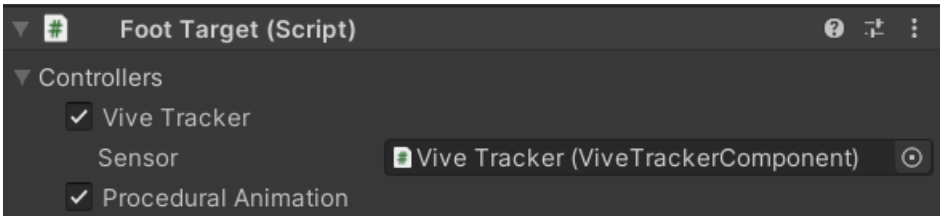
HipsTarget

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).



FootTarget

Vive Trackers can be placed on the feet.



See also

- Head Target: ViveTrackerHead
- Hand Target: ViveTrackerArm
- Hips Target: ViveTrackerTorso
- Foot Target: ViveTrackerLeg
- Passer::Tracking::ViveTrackerComponent

Meta/Oculus

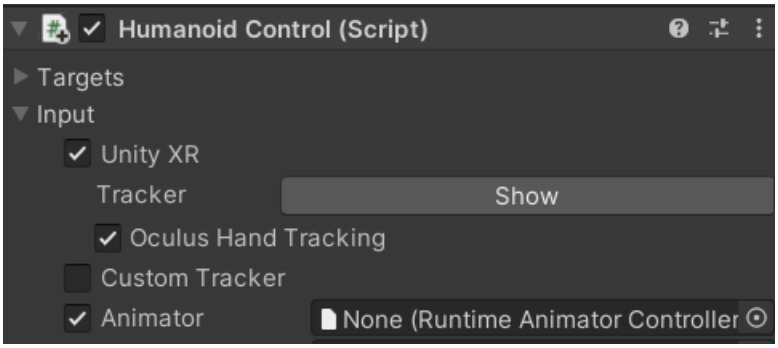
Oculus Rift and Quest are supported using the UnityXR component. This pages gives additional specific information for using the Oculus devices

Unity XR Plugin

Oculus devices can be used with both the Oculus and OpenXR plugin for Unity XR. However, it should be mentioned that hand tracking with the Oculus Quest is only supported using the Oculus plugin.

Hand Tracking (Plus & Pro)

For hand tracking, make sure the Unity XR plugin is set to Oculus. Hand tracking is currently not supported for OpenXR. When the platform is Android, an additional option will appear in the inspector



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.

Controller Buttons

The buttons of the Oculus controller can be accessed using ControllerInput. The buttons are mapped as follows:

Left Controller

Button	ControllerInput
X button	controller.left.button[0]
Y button	controller.left.button[1]
Thumbstick touch	controller.left.stickTouch
Thumbstick press	controller.left.stickPress
Thumbstick movements	controller.left.stickHorizontal & .stickVertical
Index trigger	controller.left.trigger1
Hand trigger	controller.left.trigger2
Menu button	controller.left.option

Right Controller

Button	ControllerInput
A button	controller.right.button[0]
B button	controller.right.button[1]
Thumbstick touch	controller.right.stickTouch

Thumbstick press	controller.right.stickPress
Thumbstick movements	controller.right.stickHorizontal & .stickVertical
Index trigger	controller.right.trigger1
Hand trigger	controller.right.trigger2

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Namespace Documentation

HutongGames Namespace Reference

intel Namespace Reference

intel.rssdk Namespace Reference

Passer Namespace Reference

Classes

- class BoolEvent
  An event handler calling a function with a boolean parameter
- class BoolEventHandlers
  A list of event handlers with a boolean parameter
- class CollisionEventHandler
  Implements behaviour based on collisions
- class Controller
  Controller input for a single controller
- class ControllerEventHandlers
  A list of EventHandlers for takeing care of controller input
- class ControllerInput
  A unified method for using controller input
- class Controllers
  Controller input for all controllers
- class ControllerSide
  Controller input for the left or right side of the controller (pair)
- class Counter
  A Counter can be used to record a integer number.
- class Destroyer
  Destroyer of things
- class EventHandler
  Easy universal way to attach script functions to events and statuses
- class EventHandlers
  A list of event Handlers
- class FloatEvent
  An event handler calling a function with a float parameter
- class FloatEventHandlers
  A list of event handlers with a float parameter
- class FunctionCall
  A function which can be called
- class GameObjectEvent
  An EventHandler calling a function with a GameObject type parameter
- class GameObjectEventHandlers
  A list of event handlers with GameObject parameters
- class Handle
  Component to specify behaviour when grabbing a GameObject
- class HumanoidAttachments
  Attaches additional functionality to Humanoid on a Site
- interface ICerebellum
  The Cerebellum interface defines the interface to control
- interface ICerebellumJoint
  The joint itself
- interface ICerebellumTarget
  A joint target which is used to calculate how the joints need to move
- class InteractionEventHandler

Implements behaviour using interaction

- class InteractionPointer
  A generic pointer to interact with objects in the scene using the Unity Event system.
- class IntEventHandler
  An event handler calling a function with an integer parameter
- class IntEventHandlers
  A list of event handlers with an integer parameter
- class MechanicalJoint
  Mechanical Joints can be used to limit the movements of a Kinematic Rigidbody in local space.
- class MenuManager
  The Menu Manager uses two Interaction Pointers for each hand:
- class NetworkingStarter
  Setup and start of networking
- class PlayMakerHumanoidHand
  Script for automatic sending of Grabbing and LettingGo events
- class Possessable
  Possessions can be owned by an Humanoid
- class SceneManager
  The scene manager synchronizes scene changes with humanoids across a network.
- class Sensor
  A sensor used to track an object
- class Site
  With this component you can make a Humanoid Site which can be build using SiteBuilder.
- class SiteBuilder
  A component for building Humanoid Sites
- class SiteNavigator
  The component which takes care of site navigation
- class Socket
  Sockets can hold a Handles.
- class Spawner
  Component for spawning objects
- class SpawnPoint
  A Transform represeting a place where objects can spawn
- class Target
  A main tracking transform
- class Teleporter
  The teleporter is a simple tool to teleport transforms
- class TeleportTarget
  The Teleport Target provides control to where a player can teleport and can be used in combination with a generic Interaction Pointer.
- class Transportation
  Utility functions to move transformes in the world.
- class TriggerEventHandler
  Implements input behaviour using Trigger Colliders
- class Vector3Event
  An event Handler calling a function with a Vector3 parameter
- class Vector3EventList
  A list of event handlers with a Vector3 parameter
- class VisitorPossessions

The Possession of a Humanoid Visitor

Enumerations

- enum Side { **AnySide**, **Left**, **Right** }
  Target side
- enum **GameControllers** { **Generic**, **Xbox**, **PS4**, **Steelseries**, **GameSmart**, **Oculus**, **OpenVR**, **Daydream** }
- enum **InputDeviceIDs** { **Head**, **LeftHand**, **RightHand**, **Controller** }
- enum **MovementType** { **Teleport**, **Dash**, **Walk** }

Passer.Humanoid Namespace Reference

Classes

- class ArmSensor
  An sensor used on the arm of a humanoid
- class AvatarManager
  Manage avatar meshes for a humanoid
- class CustomArm
  A custom sensor used on the arm of a humanoid
- class CustomHead
  A custom sensor used on the head of a humanoid.
- class CustomLeg
  A custom sensor used on the leg of a humanoid.
- class CustomTorso
  A custom sensor used on the torso of a humanoid.
- class CustomTracker
  A tracker wich can be used for custom tracking solutions
- class FootTarget
  Humanoid Control options for leg related things
- class HandSocket
  A Socket attached to a hand
- class HandTarget
  Humanoid Control options for hand-related things
- class HeadSensor
  A sensor used on the head of a humanoid
- class HeadTarget
  Humanoid Control options for head-related things
- class HipsTarget
  Humanoid Control options for torso related things
- class HumanoidButton
  Unity UI button with information on which humanoid pressed the button
- class HumanoidControl
  Control avatars using tracking and animation options
- class HumanoidNetworking
  Humanoid Networking
- class HumanoidPreferences
  Sets preferences for Humanoid Control
- class HumanoidSensor
  A sensor used to control a humanoid
- class HumanoidSpawner
  Component for spawning humanoids
- class HumanoidSpawnPoint
  Specifies where a new Humanoid will spawn
- class HumanoidTarget
  A tracking transform for humanoids
- class HumanoidTracker
  A Humanoid tracker
- interface IHandCollisionEvents

Interface for handling touch events on objects

- interface IHandGrabEvents
  Interface for handling grabbing events on objects
- interface IHandTouchEvents
  Interface for handling touch events on objects
- interface IHandTriggerEvents
  Interface for handling touch events on objects
- interface IHumanoidNetworking
  Interface for Humanoid Networking functions
- class LeapHand
  Leap Motion Hand to on the arm of a Humanoid
- class LeapTracker
  Leap Motion enables detailed hand tracking with markerless optical detection. Individual finger movements can be tracked.
- class NeuronTracker
  Full body tracking is supported with Perception Neuron up to 32 bones.
- class Telegrabber
  Have a humaonid grab objects from a distance
- class UnityXRTracker
  Universal API for tracking XR devices.
- class ViveTrackerArm
  Vive Tracker used on the leg of a Humanoid
- class ViveTrackerHead
  Vive Tracker used on the head of a Humanoid
- class ViveTrackerLeg
  Vive Tracker used on the leg of a Humanoid
- class ViveTrackerTorso
  Vive Tracker used on the torso of a Humanoid

Enumerations

- enum **NetworkingSystems** { **None**, **UnityNetworking**, **PhotonNetworking**, **PhotonBolt**, **MirrorNetworking** }
- enum **LegBones** { **UpperLeg**, **LowerLeg**, **Foot**, **Toes** }
- enum **ArmBones** { **Hand**, **Forearm**, **UpperArm**, **Shoulder** }
- enum **HandBones** { **ThumbProximal** = 0, **ThumbIntermediate** = 1, **ThumbDistal** = 2, **IndexProximal** = 3, **IndexIntermediate** = 4, **IndexDistal** = 5, **MiddleProximal** = 6, **MiddleIntermediate** = 7, **MiddleDistal** = 8, **RingProximal** = 9, **RingIntermediate** = 10, **RingDistal** = 11, **LittleProximal** = 12, **LittleIntermediate** = 13, **LittleDistal** = 14, **LastHandBone** = 15 }
- enum **TorsoBones** { **Hips**, **Spine**, **Chest** }

Passer.Humanoid.Cerebellum Namespace Reference

Passer.Humanoid.Tracking Namespace Reference

Classes

- class Sensor
  Humanoid Tracking sensor

Enumerations

- enum **FingerBones** { **Proximal** = 0, **Intermediate** = 1, **Distal** = 2, **LastBone** = 3 }
- enum **FaceBone** { **LeftOuterBrow**, **LeftBrow**, **LeftInnerBrow**, **RightInnerBrow**, **RightBrow**, **RightOuterBrow**, **LeftCheek**, **RightCheek**, **NoseTopLeft**, **NoseTop**, **NoseTopRight**, **NoseTip**, **NoseBottomLeft**, **NoseBottom**, **NoseBottomRight**, **UpperLipLeft**, **UpperLip**, **UpperLipRight**, **LipLeft**, **LipRight**, **LowerLipLeft**, **LowerLip**, **LowerLipRight**, **LastBone** }
- enum **Status** { **Unavailable**, **Present**, **Tracking** }

Passer.Pawn Namespace Reference

Passer.Tracking Namespace Reference

Classes

- class Antilatency
  Antilatency tracking device
- class BodySkeleton
  The representation of a tracked body
- class HandSkeleton
  The representation of a tracked hand
- class HydraBaseStation
  A Razer Hydra hand tracking device
- class HydraController
  An Razer Hydra controller
- class LeapHandSkeleton
  A hand tracked by LeapMotion
- class LeapMotion
  A representation of the real-world LeapMotion camera
- class Oculus
  The Oculus Device
- class OculusController
  An Oculus controller
- class OculusHandSkeleton
  A hand from Oculus hand tracking
- class OculusHmd
  The Oculus Head Mounted Device
- class PerceptionNeuron
  Component to receive tracking information from Axis Neuron
- class SensorComponent
  A sensor component is used to add tracking to a transform
- class Tracker
  A tracker
- class TrackerComponent
  Generic Tracking device
- class UnityXRHandSkeleton
  HandSkeleton component for hand tracking with UnityXR
- class ViveHandSkeleton
  A HTC Vive hand tracking skeleton
- class ViveTrackerComponent
  A representation of a real-world Vive Tracker

Enumerations

- enum **TrackerId** { **Oculus**, **SteamVR**, **WindowsMR**, **LeapMotion**, **Kinect1**, **Kinect2**, **OrbbecAstra**, **Realsense**, **RazerHydra**, **Optitrack**, **Tobii** }

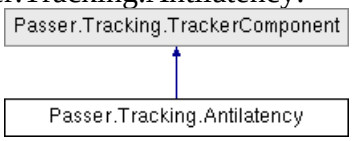
Class Documentation

Passer.Tracking.Antilatency Class Reference

Description

Antilatency tracking device

Inheritance diagram for Passer.Tracking.Antilatency:



Public Types

- enum **SensorType** { **Tag**, **Bracer** }

Public Member Functions

- AntilatencySensor FindSensor (string socketTag=null)
  Find an Antilatency Tag

- AntilatencySensor GetSensor (Vector3 position, Quaternion rotation, string socketTag=null, bool usePlacement=false)
  Find or Create a new Antilatency Sensor

- AntilatencySensor GetTag (Vector3 position, Quaternion rotation, string socketTag=null, bool usePlacement=false)
  Find or Create a new Antilatency Tag

- AntilatencyBracer GetBracer (Vector3 position, Quaternion rotation, bool isLeft)
  Find or Create a new Antilatency Bracer

- AntilatencyBracer GetBracer (Vector3 position, Quaternion rotation, bool isLeft, string socketTag)
  Find or Create a new Antilatency Bracer

- virtual void **ShowSkeleton** (bool shown)

Static Public Member Functions

- static Antilatency Find (Transform parentTransform)
  Find an Antilatency Tracker

- static Antilatency Get (Transform parentTransform, Vector3 position, Quaternion rotation)
  Create a new Antilatency Tracker

Public Attributes

- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- override void Start ()
- override void Update ()

Protected Attributes

- DeviceNetwork **network**
- AltEnvironment **environment**
- Transform **realWorld**

Member Function Documentation

static Antilatency Passer.Tracking.Antilatency.Find (Transform *parentTransform*) `[static]`

Find an Antilatency Tracker

Parameters

parentTransform	

Returns

static Antilatency Passer.Tracking.Antilatency.Get (Transform *parentTransform*, Vector3 *position*, Quaternion *rotation*) `[static]`

Create a new Antilatency Tracker

Parameters

parentTransform	
position	
rotation	

Returns

AntilatencySensor Passer.Tracking.Antilatency.FindSensor (string *socketTag* = `null`)

Find an Antilatency Tag

Parameters

antilatency	

Returns

AntilatencySensor Passer.Tracking.Antilatency.GetSensor (Vector3 *position*, Quaternion *rotation*, string *socketTag* = `null`, bool *usePlacement* = `false`)

Find or Create a new Antilatency Sensor

Parameters

position	
rotation	
socketTag	
usePlacement	

Returns

AntilatencySensor Passer.Tracking.Antilatency.GetTag (Vector3 *position*, Quaternion *rotation*, string *socketTag* = `null`, bool *usePlacement* = `false`)

Find or Create a new Antilatency Tag

Parameters

position	
rotation	
socketTag	
usePlacement	

Returns

AntilatencyBracer Passer.Tracking.Antilatency.GetBracer (Vector3 *position*, Quaternion *rotation*, bool *isLeft*)

Find or Create a new Antilatency Bracer

Parameters

antilatency	
position	
rotation	
isLeft	

Returns

AntilatencyBracer Passer.Tracking.Antilatency.GetBracer (Vector3 *position*, Quaternion *rotation*, bool *isLeft*, string *socketTag*)

Find or Create a new Antilatency Bracer

Parameters

antilatency	
position	
rotation	
isLeft	

Returns

override void Passer.Tracking.Antilatency.Start () `[protected], [virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

override void Passer.Tracking.Antilatency.Update () `[protected], [virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

The documentation for this class was generated from the following file:

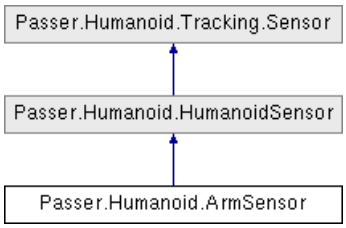
- Assets/Passer/HumanoidControlPro/Runtime/Scripts/Extensions/Antilatency/Antilatency.cs

Passer.Humanoid.ArmSensor Class Reference

Description

An sensor used on the arm of a humanoid

Inheritance diagram for Passer.Humanoid.ArmSensor:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- virtual void CheckSensor (HandTarget _handTarget)
  Checks if a sensorComponent is present and will create one if it does not exist

- virtual void **Init** (HandTarget handTarget)
- override void Start (HumanoidControl _humanoid, Transform targetTransform)
- void **InitController** (SerializedProperty sensorProp, HandTarget handTarget)
- void **RemoveController** (SerializedProperty sensorProp)
- void **CheckSensorTransform** (Transform targetTransform, bool isLeft)
- virtual void **CreateSensorTransform** (Transform targetTransform, bool isLeft)
- override void SetSensor2Target ()
- virtual void **SetSensor2Target** (Vector3 targetPosition, Quaternion targetRotation)
- float **ArmConfidence** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **Vibrate** (float length, float strength)
- virtual void **CheckSensorTransform** ()
- override void Update ()
  Update the sensor state

- virtual void **UpdateSensorTransformFromTarget** (Transform targetTransform)
- virtual void **Stop** ()
- virtual void **RefreshSensor** ()
- virtual void **ShowSensor** (HumanoidTarget target, bool shown)

Static Public Member Functions

- static Vector3 **InverseTransformPointUnscaled** (Transform transform, Vector3 position)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- HandSkeleton handSkeleton
  The sensor used for this arm

- Vector3 **sensor2TargetPosition**

- Quaternion **sensor2TargetRotation**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Static Public Attributes

- const string **_name** = ""

Protected Member Functions

- virtual HandSkeleton **FindHandSkeleton** (bool isLeft)
- virtual HandSkeleton **CreateHandSkeleton** (bool isLeft, bool showRealObjects)
- virtual void **CreateSensorTransform** (string resourceName, Vector3 sensor2TargetPosition, Quaternion sensor2TargetRotation)
- void **UpdateArm** (Humanoid.Tracking.ArmSensor armSensor)
- void **UpdateShoulder** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **UpdateUpperArm** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **UpdateForearm** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **UpdateHand** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **UpdateHandTargetTransform** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **UpdateFingers** (Humanoid.Tracking.ArmSensor armSensor)
- virtual void **UpdateHandFromSkeleton** ()
- virtual void **UpdateThumbFromSkeleton** ()
- void **UpdateIndexFingerFromSkeleton** ()
- void **UpdateMiddleFingerFromSkeleton** ()
- void **UpdateRingFingerFromSkeleton** ()
- void **UpdateLittleFingerFromSkeleton** ()
- virtual void **UpdateFingerBoneFromSkeleton** (Transform targetTransform, Finger finger, FingerBone fingerBone)
- virtual void **CreateSensorTransform** ()
- void **CreateSensorTransform** (Transform targetTransform, string resourceName, Vector3 _sensor2TargetPosition, Quaternion _sensor2TargetRotation)
- void **RemoveSensorTransform** ()
- void **UpdateSensorTransform** (Tracking.Sensor sensor)
- virtual void **UpdateTargetTransform** ()
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, Transform sensorTransform)
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, SensorComponent sensorComponent)
- Vector3 **GetTargetPosition** (Transform sensorTransform)
- Quaternion **GetTargetRotation** (Transform sensorTransform)
- void **UpdateSensor** ()

Static Protected Member Functions

- static Vector3 **TransformPointUnscaled** (Transform transform, Vector3 position)

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**

Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- HandTarget **handTarget** `[get]`
  The HandTarget for this sensor

- HumanoidControl **humanoid** `[get]`
  The Humanoid for this sensor

- virtual new HumanoidTracker **tracker** `[get]`
- override string **name** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`

Member Function Documentation

virtual void Passer.Humanoid.ArmSensor.CheckSensor (HandTarget *_handTarget*) `[virtual]`

Checks if a sensorComponent is present and will create one if it does not exist

Parameters

_handTarget	The HandTarget to which the sensor is assigned

override void Passer.Humanoid.ArmSensor.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) `[virtual]`

Reimplemented from Passer.Humanoid.HumanoidSensor.

override void Passer.Humanoid.ArmSensor.SetSensor2Target () `[virtual]`

Reimplemented from Passer.Humanoid.HumanoidSensor.

override void Passer.Humanoid.HumanoidSensor.Update () `[virtual]`, `[inherited]`

Update the sensor state

Returns

Status of the sensor after the update

Reimplemented from Passer.Humanoid.Tracking.Sensor.

Reimplemented in Passer.Humanoid.CustomHead.

Member Data Documentation

HandSkeleton Passer.Humanoid.ArmSensor.handSkeleton

The sensor used for this arm

The skeleton for the hand

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/ArmSensor.cs

Passer.Humanoid.AvatarManager Class Reference

Description

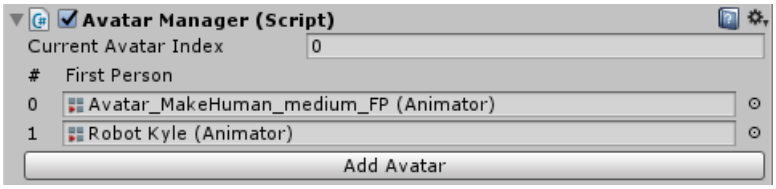
Manage avatar meshes for a humanoid

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 should be attached to an GameObject with the HumanoidControl component script.

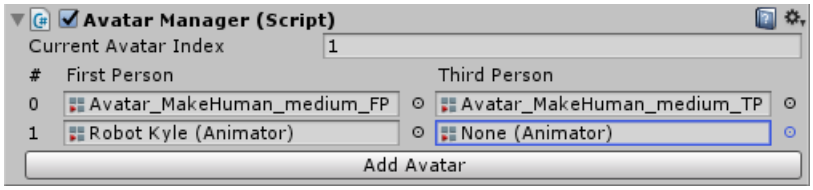
Single User



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 HumanoidControl component!*

Networking

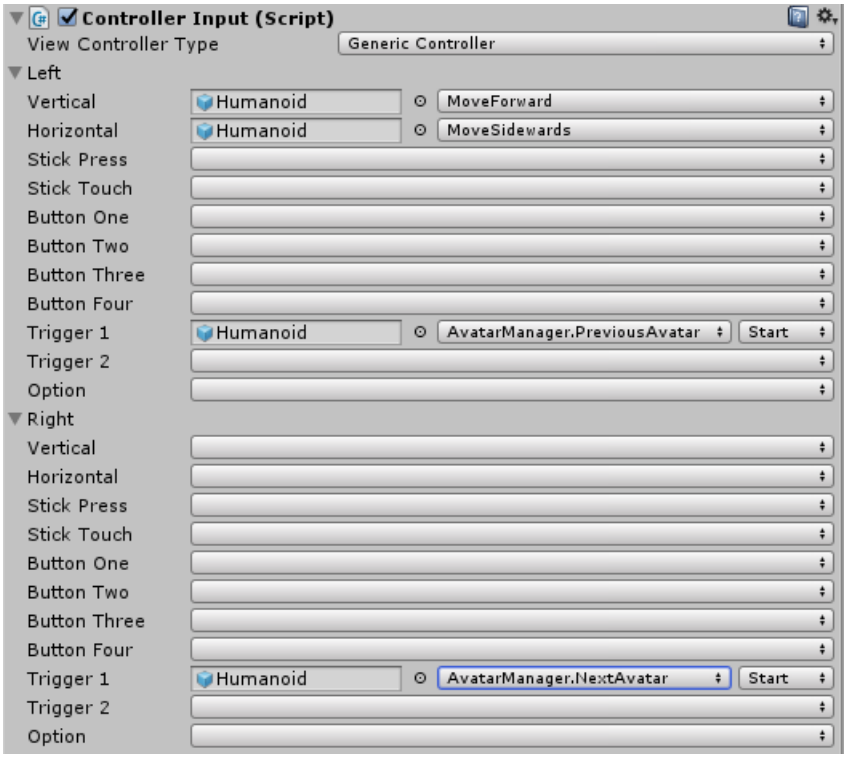
When Networking Support is enabled additional entries for the third person avatar will be shown.



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 on of the scripting functions like PreviousAvastar, NextAvatar and SetAvatar. These functions can also be used in combination with Scripts or ControllerInput:



Inherits MonoBehaviour.

Public Member Functions

- void NextAvatar ()
  Replaces the current avatar by the next avatar in the list.

- void PreviousAvatar ()
  Replaces the current avatar by the previous avatar in the list.

- void SetAvatar (int avatarIndex)
  This will replace the current avatar with the avatar indicated by the avatarIndex.

Static Public Member Functions

- static int **mod** (int k, int n)

Public Attributes

- int **currentAvatarIndex** = 0
  The index of the current avatar in the list

- Animator[] fpAvatars = new Animator[0]
  The list of avatars for the humanoid

- Animator[] tpAvatars = new Animator[0]
  The list of third person avatar for networked humanoids

Protected Member Functions

- virtual void **Start** ()

Member Function Documentation

void Passer.Humanoid.AvatarManager.NextAvatar ()

Replaces the current avatar by the next avatar in the list.

This will wrap around when the last avatar is the current avatar.

void Passer.Humanoid.AvatarManager.PreviousAvatar ()

Replaces the current avatar by the previous avatar in the list.

This will wrap around when the first avatar is the current avatar.

void Passer.Humanoid.AvatarManager.SetAvatar (int *avatarIndex*)

This will replace the current avatar with the avatar indicated by the avatarIndex.

Parameters

avatarIndex	The index of the avatar in the list of avatars

Member Data Documentation

Animator [] Passer.Humanoid.AvatarManager.fpAvatars = new Animator[0]

The list of avatars for the humanoid

For networked avatars this avatar will be used for the local client.

Animator [] Passer.Humanoid.AvatarManager.tpAvatars = new Animator[0]

The list of third person avatar for networked humanoids

This is the avatar which will be used to show a player of remote clients. When no third person avatar is specified, the first person avatar will be used as the third person avatar.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Tools/AvatarManager.cs

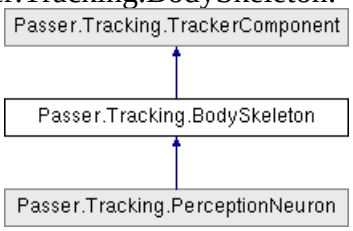
Passer.Tracking.BodySkeleton Class Reference

Description

The representation of a tracked body

The bone hierarchy will be created below this transform at runtime.

Inheritance diagram for Passer.Tracking.BodySkeleton:



Public Member Functions

- Transform GetBoneTransform (Humanoid.Tracking.Bone boneId)
  Gets the transform of the tracked bone

- TrackedBone GetBone (Humanoid.Tracking.Bone boneId)
  Gets the tracked bone

- virtual void **ShowSkeleton** (bool shown)

Public Attributes

- bool **show** = false
  Determines whether the skeleton should be rendered

- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- abstract void InitializeSkeleton ()
  This function is used to intialize the tracked bones.

- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()
- virtual void **Start** ()
- virtual void **Update** ()

Protected Attributes

- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **realWorld**

Member Function Documentation

abstract void Passer.Tracking.BodySkeleton.InitializeSkeleton () `[protected], [pure virtual]`

This function is used to intialize the tracked bones.

Implemented in Passer.Tracking.PerceptionNeuron.

Transform Passer.Tracking.BodySkeleton.GetBoneTransform (Humanoid.Tracking.Bone *boneId*)

Gets the transform of the tracked bone

Parameters

boneId	The requested bone

Returns

The tracked bone transform or *null* if it does not exist

TrackedBone Passer.Tracking.BodySkeleton.GetBone (Humanoid.Tracking.Bone *boneId*)

Gets the tracked bone

Parameters

boneId	The requested bone

Returns

The tracked bone or *null* if it does not exist

The documentation for this class was generated from the following file:

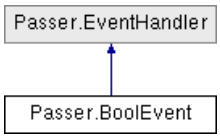
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/BodySkeleton.cs

Passer.BoolEvent Class Reference

Description

An event handler calling a function with a boolean parameter

Inheritance diagram for Passer.BoolEvent:



Public Types

- enum Type { Never, OnStart, OnEnd, WhileActive, WhileInactive, OnChange, Continuous }
  The different types of events when the function is called
- enum OverrideMode { Prepend, Append, Replace }

Public Member Functions

- **BoolEvent** (Type newEventType=Type.OnChange)
- void **SetMethod** (Type newEventType, UnityAction voidAction)
- void **SetMethod** (Type newEventType, UnityAction< bool > boolAction)
- virtual void **Update** ()

Public Attributes

- Type **eventType** = Type.Continuous
  The event type for the function call

- bool **eventNetworking** = false
  For future use :-)

- FunctionCall **functionCall**
  The function to call

- bool **boolInverse** = false
  Negate the boolean state before calling event trigger

- OverrideMode **overrideMode**

Protected Member Functions

- override void UpdateBool ()
- override void UpdateInt ()
- override void UpdateFloat ()
- void **UpdateAnimationParameter** ()
- virtual void **UpdateVoid** ()
- virtual void **UpdateString** ()
- virtual void **UpdateString** (string s)
- virtual void **UpdateVector3** ()
- virtual void **UpdateGameObject** ()
- virtual void **UpdateRigidbody** ()
- virtual void **UpdateStringBool** (string s)
- virtual void **UpdateStringFloat** (string s)

- virtual void **UpdateStringInt** (string s)
- bool **CheckCondition** (bool active, bool changed, bool valueChanged)

Protected Attributes

- bool **initialized**
- bool **_boolValue**
- bool **boolChanged** = true
- int **_intValue**
- bool **intChanged**
- float **_floatValue**
- bool **floatChanged**

Properties

- bool??? **value** `[getset]`
  The GameObject value for this event

- virtual bool **boolValue** `[getset]`
- bool isDead `[get]`
  True when the eventHandler is dead and can be removed

Member Enumeration Documentation

enum Passer.EventHandler.Type `[inherited]`

The different types of events when the function is called

Enumerator:

Never	The function is never called.
OnStart	The function is called when the event starts.
OnEnd	The function is called when the event ends.
WhileActive	The function is called every frame while the event is active.
WhileInactive	The function is called every frame while the event is not active.
OnChange	The function is called every time the event changes.
Continuous	The functions is called every frame.

enum Passer.EventHandler.OverrideMode `[inherited]`

Enumerator:

Prepend	Prepend this handler before existing handlers.

Append	Append this handler after existing handlers.
Replace	Replace the topmost handler with this handler.

Member Function Documentation

override void Passer.BoolEvent.UpdateBool () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.BoolEvent.UpdateInt () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.BoolEvent.UpdateFloat () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

Property Documentation

bool Passer.EventHandler.isDead `[get], [inherited]`

True when the eventHandler is dead and can be removed

A function is dead when it does nothing. This is when the functionCall is not defined or when the target of the functionCall is empty

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/BoolEvent.cs

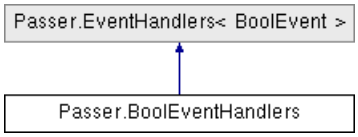
Passer.BoolEventHandlers Class Reference

Description

A list of event handlers with a boolean parameter

This is used to implement a list of functions which should be called with a boolean as parameter

Inheritance diagram for Passer.BoolEventHandlers:



Public Member Functions

- void **Update** ()

Public Attributes

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events**
  The EventHandlers

Properties

- bool **value** `[getset]`

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/BoolEvent.cs

Passer.CollisionEventHandler Class Reference

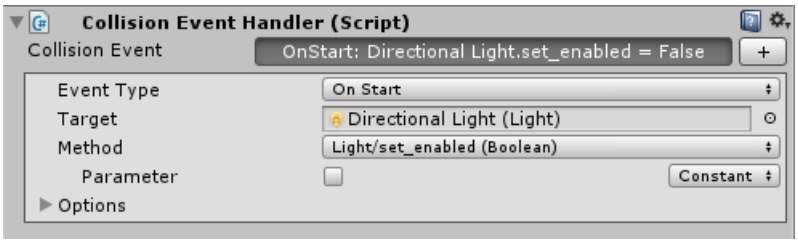
Description

Implements behaviour based on collisions

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

The Event

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



The event type is as follows:

- Never: the event handler is disabled, the Target Method is never called
- On Collision Start: the Target Method is called when the collision starts. This is equivalent to the OnCollisionEnter() function of Unity.
- On Collision End: the Target Method is called when the collision ends. This is equivalent to the OnCollisionExit() function of Unity.
- While Colliding: 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() function of Unity.
- While not Colliding: the Target Method is called while the collider is not colliding. In that case it will be called in every frame.
- On Collision Change: the Target Method is called when the collision starts or ends.
- Always: 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.

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.

Inherits MonoBehaviour.

Public Attributes

- GameObjectEventHandlers collisionHandlers

Protected Member Functions

- virtual void **OnCollisionEnter** (Collision collision)
- virtual void **OnCollisionExit** (Collision collision)

Member Data Documentation

GameObjectEventHandlers Passer.CollisionEventHandler.collisionHandlers

```
Initial value:= new GameObjectEventHandlers() {
            label = "Collision Event",
            tooltip =
                "Call functions using the collider state\n" +
                "Parameter: the GameObject colliding with the collider",
            eventTypeLabels = new string[] {
                "Never",
                "On Collision Start",
                "On Collision End",
                "While Colliding",
                "While not Colliding",
                "On Collision Change",
                "Always"
            },
        }
```

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Physics/CollisionEventHandler.cs

Passer.Controller Class Reference

Description

Controller input for a single controller

Public Types

- enum Side { **Left**, **Right** }
  Identification for the left or right side of the controller
- enum Button { **ButtonOne** = 0, **ButtonTwo** = 1, **ButtonThree** = 2, **ButtonFour** = 3, **Bumper** = 10, **BumperTouch** = 11, **Trigger** = 12, **TriggerTouch** = 13, **StickButton** = 14, **StickTouch** = 15, **Option** = 30, **None** = 9999 }
  Button identification values

Public Member Functions

- void **Update** ()
  Update the current values of the controller input

- **Controller** ()
  Constructor for access to the controller input

- void **Clear** ()
  Reset the values of all buttons

- void **EndFrame** ()
  Called at the end of the frame to indicate that new controller values can be read

- bool GetButton (Side side, Button buttonID)
  Retrieve the pressed state of a button

Public Attributes

- ControllerSide **left**
  The left side of the controller

- ControllerSide **right**
  The right side of the controller

Member Function Documentation

bool Passer.Controller.GetButton (Side *side*, Button *buttonID*)

Retrieve the pressed state of a button

Parameters

side	The identification of the side of the controller
buttonID	The identification of the requested button

The documentation for this class was generated from the following file:

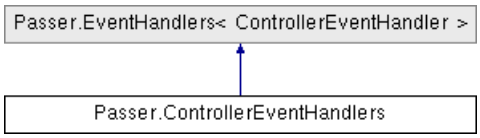
- Assets/Passer/HumanoidControl/Runtime/Tools/Input/Controller.cs

Passer.ControllerEventHandlers Class Reference

Description

A list of EventHandlers for takeing care of controller input

Inheritance diagram for Passer.ControllerEventHandlers:



Public Member Functions

- **ControllerEventHandlers** ()
  Create new ControllerEventHandlers

- void **Clear** ()
- void **SetMethod** (EventHandler.Type eventType, UnityAction voidEvent)
- void **SetMethod** (EventHandler.Type eventType, UnityAction< bool > boolEvent)

Static Public Member Functions

- static void Cleanup (ControllerEventHandlers[] eventHandlers)
  Cleanup the eventHandlers

Public Attributes

- string **defaultParameterProperty**
  For future use...

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events**
  The EventHandlers

Static Protected Attributes

- static string[] controllerEventTypeLabels

Properties

- float **floatValue** `[getset]`
  The float input value for the controller event

Member Function Documentation

static void Passer.ControllerEventHandlers.Cleanup (ControllerEventHandlers[] *eventHandlers*) `[static]`

Cleanup the eventHandlers

This will remove all EventHandlers for which isDead is true.

Parameters

eventHandlers	The array of eventHandlers to clean.

Member Data Documentation

string [] Passer.ControllerEventHandlers.controllerEventTypeLabels `[static], [protected]`

```
Initial value:= new string[] {
            "Never",
            "On Press",
            "On Release",
            "While Down",
            "While Up",
            "On Change",
            "Continuous"
        }
```

The documentation for this class was generated from the following file:

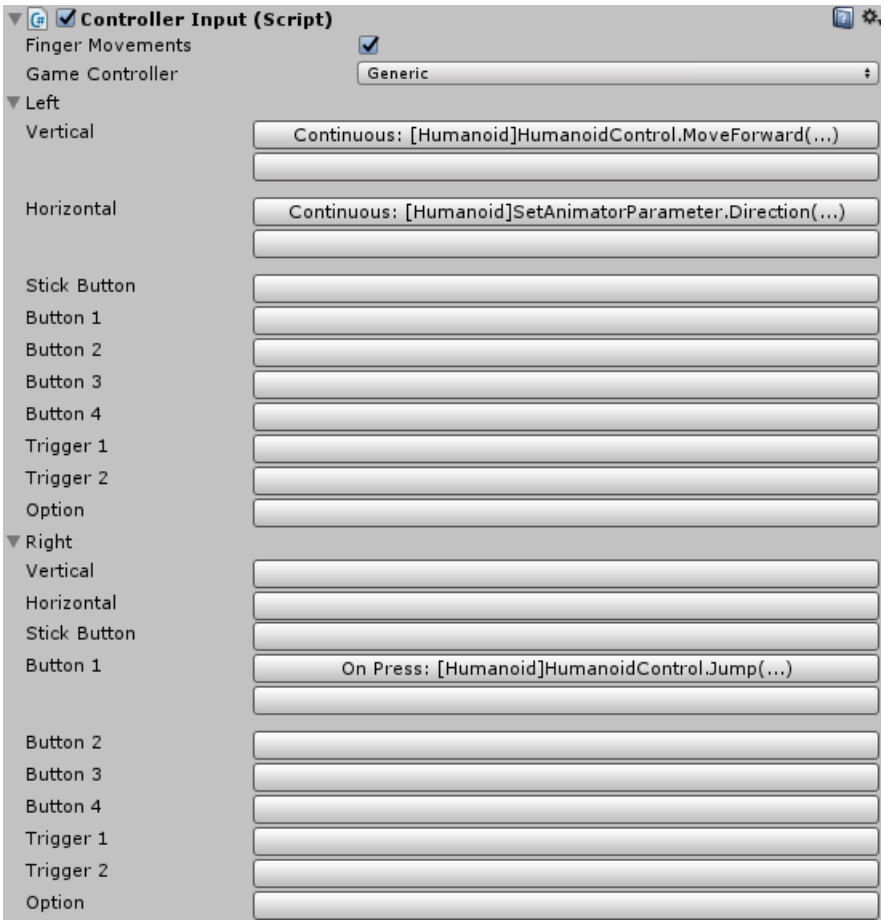
- Assets/Passer/HumanoidControl/Runtime/Tools/Events/ControllerEvent.cs

Passer.ControllerInput Class Reference

Description

A unified method for using controller input

ControllerInput can be used to access the state and change of input buttons, touchpads and thumbsticks on any kind of input controller including game controllers and VR/XR controllers.



- Finger Movements
- Game Controller

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. Event input is using an EventHandler to define its behaviour.

Inherits MonoBehaviour.

Public Types

- enum **SideButton** { **StickVertical**, **StickHorizontal**, **StickButton**, **TouchpadVertical**, **TouchpadHorizontal**, **TouchpadButton**, **Button1**, **Button2**, **Button3**, **Button4**, **Trigger1**, **Trigger2**, **Option** }

Public Member Functions

- void **InitializeLeftInputEvents** ()
- void **InitializeRightInputEvents** ()
- ControllerEventHandlers **GetInputEvent** (bool isLeft, SideButton controllerButton)
- void **SetEventHandler** (bool isLeft, SideButton sideButton, UnityAction< bool > boolEvent)
- void **SetRightTrigger1Value** (float value)

Static Public Member Functions

- static void **SetBoolMethod** (GameObject gameObject, ControllerEventHandlers eventHandlers, EventHandler.Type eventType, UnityAction< bool > boolEvent)

Public Attributes

- HumanoidControl **humanoid**
- bool fingerMovements = true
  Enables a built-in support for finger movements from the controller buttons.

- GameControllers gameController
  Selects which controller type is showed in the Inspector.

- ControllerEventHandlers[] **leftInputEvents**
- ControllerEventHandlers[] **rightInputEvents**

Protected Member Functions

- virtual void **Awake** ()
- virtual void **OnDestroy** ()
- virtual void **Start** ()
- virtual void **Update** ()
- void **UpdateInputList** (ControllerEventHandlers inputEventList, float value)
- void **UpdateFingerMovements** ()
- void **UpdateFingerMovementsSide** (FingersTarget fingers, ControllerSide controllerSide)

Protected Attributes

- HandTarget **leftHandTarget**
- HandTarget **rightHandTarget**
- Controller **controller**

Static Protected Attributes

- static string[] eventTypeLabels

Properties

- float **leftStickVertical** `[get]`
- float **leftStickHorizontal** `[get]`
- float **leftTouchpadVertical** `[get]`
- float **leftTouchpadHorizontal** `[get]`
- float **leftTrigger1** `[get]`
- bool **leftTrigger1Touched** `[get]`
- bool **leftTrigger1Pressed** `[get]`

- float **leftTrigger2** `[get]`
- bool **leftTrigger2Touched** `[get]`
- bool **leftTrigger2Pressed** `[get]`
- bool **leftButton1Pressed** `[get]`
- bool **leftButton2Pressed** `[get]`
- bool **leftButton3Pressed** `[get]`
- bool **leftButton4Pressed** `[get]`
- bool **leftOptionPressed** `[get]`
- float **rightStickVertical** `[get]`
- float **rightStickHorizontal** `[get]`
- float **rightTouchpadVertical** `[get]`
- float **rightTouchpadHorizontal** `[get]`
- float **rightTrigger1** `[get]`
- bool **rightTrigger1Touched** `[get]`
- bool **rightTrigger1Pressed** `[get]`
- float **rightTrigger2** `[get]`
- bool **rightTrigger2Touched** `[get]`
- bool **rightTrigger2Pressed** `[get]`
- bool **rightButton1Pressed** `[get]`
- bool **rightButton2Pressed** `[get]`
- bool **rightButton3Pressed** `[get]`
- bool **rightButton4Pressed** `[get]`
- bool **rightOptionPressed** `[get]`

Member Data Documentation

bool Passer.ControllerInput.fingerMovements = true

Enables a built-in support for finger movements from the controller buttons.

This option is only available when the Controller Input Component is attached to an Humanoid.

GameControllers Passer.ControllerInput.gameController

Selects which controller type is showed in the Inspector.

his 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.

string [] Passer.ControllerInput.eventTypeLabels `[static]`, `[protected]`

```
Initial value:= new string[] {
                "Never",
                "On Press",
                "On Release",
                "While Down",
                "While Up",
                "On Change",
                "Continuous"
        }
```

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Input/ControllerInput.cs

Passer.Controllers Class Reference

Description

Controller input for all controllers

Max 4 controllers are supports

Static Public Member Functions

- static void **Update** ()
  Update the current values of the controller input

- static Controller GetController (int controllerID)
  Retrieves a controller and creates it when it is first accessed

- static void **Clear** ()
  Reset the values of all buttons

- static void **EndFrame** ()
  Called at the end of the frame to indicate that new controller values can be read

Static Public Attributes

- static Controller[] **controllers**
  Array containing all controllers

Member Function Documentation

static Controller Passer.Controllers.GetController (int *controllerID*) `[static]`

Retrieves a controller and creates it when it is first accessed

Parameters

controllerID	The index of the controller

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Input/Controller.cs

Passer.ControllerSide Class Reference

Description

Controller input for the left or right side of the controller (pair)

Public Member Functions

- delegate void OnButtonDown (Controller.Button buttonNr)
  Function for processing button down events

- delegate void OnButtonUp (Controller.Button buttonNr)
  Function for processing button up events

- void **Update** ()
  Update the current values of the controller input

- void **Clear** ()
  Reset the values of all buttons

- bool GetButton (Controller.Button buttonID)
  Retrieve the pressed state of a button

Public Attributes

- float stickHorizontal
  The vertical value of the thumbstick

- float stickVertical
  The horizontal value of the thumbstick

- bool **stickButton**
  The pressed state of the thumbstick

- bool **stickTouch**
  The touched state of the thumbstick

- float touchpadVertical
  The vertical value of the touchpad

- float touchpadHorizontal
  The horizontal value of the touchpad

- bool **touchpadPress**
  The pressed state of the touchpad

- bool **touchpadTouch**

The touched state of the touchpad

- bool[] buttons = new bool[4]
  The pressed state of genertic buttons

- float trigger1
  The value of the first trigger

- float trigger2
  The value of the second trigger

- bool option
  The pressed state of the option button

Events

- OnButtonDown **OnButtonDownEvent**
  Event for handling Button down events

- OnButtonUp **OnButtonUpEvent**
  Event for handling Button up events

Member Function Documentation

delegate void Passer.ControllerSide.OnButtonDown (Controller.Button *buttonNr*)

Function for processing button down events

Parameters

buttonNr	The idetification of the pressed button

delegate void Passer.ControllerSide.OnButtonUp (Controller.Button *buttonNr*)

Function for processing button up events

Parameters

buttonNr	The identification of the released button

bool Passer.ControllerSide.GetButton (Controller.Button *buttonID*)

Retrieve the pressed state of a button

Parameters

buttonID	The identification of the requested button

Member Data Documentation

float Passer.ControllerSide.stickHorizontal

The vertical value of the thumbstick

Values: -1..1

float Passer.ControllerSide.stickVertical

The horizontal value of the thumbstick

Values: -1..1

float Passer.ControllerSide.touchpadVertical

The vertical value of the touchpad

Values: -1..1

float Passer.ControllerSide.touchpadHorizontal

The horizontal value of the touchpad

Values: -1..1

bool [] Passer.ControllerSide.buttons = new bool[4]

The pressed state of genertic buttons

There can be up to 4 generic buttons. buttons[0] usually mathes the default fire button

float Passer.ControllerSide.trigger1

The value of the first trigger

Values: 0..1 The first trigger is normally operated with the index finger

float Passer.ControllerSide.trigger2

The value of the second trigger

Values: 0..1 The second trigger is normally operated with the middle finger

bool Passer.ControllerSide.option

The pressed state of the option button

The option button is usually a special button for accessing a specific menu

The documentation for this class was generated from the following file:

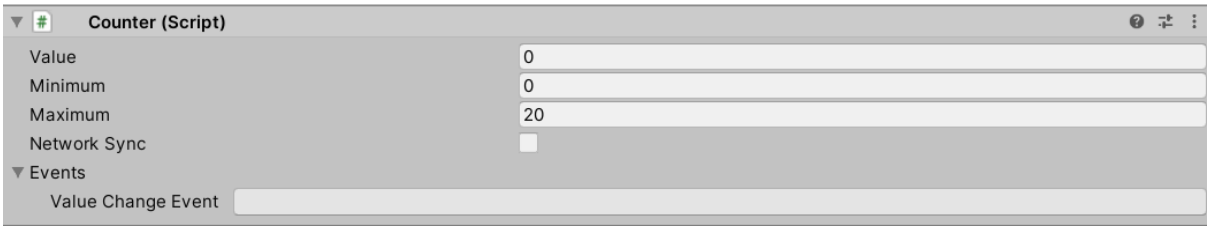
- Assets/Passer/HumanoidControl/Runtime/Tools/Input/Controller.cs

Passer.Counter Class Reference

Description

A Counter can be used to record a integer number.

The Counter can be incremented and decremented and functions cal be called based on the value.



- Value, see Counter::value
- Minimum, see Counter::min
- Maximum, see Counter::max
- Network Sync, see Counter::networking. This property is only available when a supported networking solution is included in the project.
- Events
  - Value Change Event, see Counter::counterEvent

Version

4.0 and higher

Inherits MonoBehaviour.

Public Member Functions

- void Decrement ()
  Decrements the Counter value by 1

- void Increment ()
  Increments the Counter value by 1

- void **SetValueToMin** ()
  Sets the Counter value to the minimum value

- void **SetValueToMax** ()
  Sets the Counter value to the maximum value

- void **SetTimer** (int seconds)
- void **SetTimer** (int seconds, float speed)

Public Attributes

- int **min** = 0

The minimum value for the Counter

- int **max** = 10
  The maximum value for the Counter

- float **timer** = 0
- bool **networking** = false
  Synchronize the value across the network.

- IntEventHandlers counterEvent
  Can be used to call values based on the Counter value

Protected Member Functions

- virtual void **Awake** ()
- virtual void **FixedUpdate** ()
- virtual void **CheckBounds** ()
- void **CallRemote** (string functionName, params object[] parameters)

Protected Attributes

- int **_value**
- PhotonView **photonView**

Properties

- int value `[getset]`
  Sets or gets the value of the Counter

Member Function Documentation

void Passer.Counter.Decrement ()

Decrements the Counter value by 1

If the Counter value is equal or lower than the minimum value, the value is not changed

void Passer.Counter.Increment ()

Increments the Counter value by 1

If the Counter value is equal or higher than the maximum value, the value is not changed
If networking is enabled, the value is incremented on all clients and event handlers are called on all clients

---

Member Data Documentation

IntEventHandlers Passer.Counter.counterEvent

```
Initial value:= new IntEventHandlers() {
            label = "Value Change Event",
            tooltip =
```

```
                "Call functions using counter values\n" +
                "Parameter: the counter value",
            eventTypeLabels = new string[] {
                "Never",
                "On Min",
                "On Max",
                "While Min",
                "While Max",
                "When Changed",
                "Always"
            }
        }
```

Can be used to call values based on the Counter value

Property Documentation

int Passer.Counter.value `[get]`, `[set]`

Sets or gets the value of the Counter

If networking is enabled, the value is synchoronized across the network.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/Counter.cs

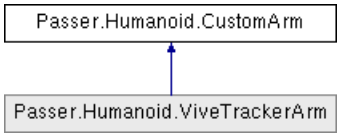
Passer.Humanoid.CustomArm Class Reference

Description

A custom sensor used on the arm of a humanoid

This tracking option supports a custom developed ControllerComponent or HandSkeleton for the hand and/or arm.

Inheritance diagram for Passer.Humanoid.CustomArm:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **SetSensor2Target** ()
  Updates the arm targets based on the current sensor position and rotation

- override void UpdateSensorTransformFromTarget (Transform _)
  Updates the sensor position and rotation based on the current position of the arm targets.

- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the arm for tracking with the tracked controller and/or skeleton

- override void Update ()
  Updates the arm target based on the status of the tracked controller and/or skeleton

Static Public Member Functions

- static Rotation **CalculateUpperArmOrientation** (Vector upperArmPosition, float upperArmLength, Vector forearmUp, float forearmLength, Vector handPosition, bool isLeft)
- static Rotation **CalculateArmOrientation** (Vector joint1Position, Vector joint1Up, Vector joint2Position, bool isLeft)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- ArmBones **attachedBone** = ArmBones.Hand
  The bone on the arm controlled by the sensor

- bool **isLeft**
- TargetData **shoulder**
- TargetData **upperArm**
- TargetData **forearm**
- TargetData **hand**
- Finger **thumb**

- Finger **indexFinger**
- Finger **middleFinger**
- Finger **ringFinger**
- Finger **littleFinger**
- Finger[] **fingers**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- void **UpdateControllerInput** ()
  Updates the Controller Input for this side

- virtual void UpdateControllerInput (ControllerSide controllerSide)
  Updates one side of the ControllerInput from the values of the tracked ControllerComponent

- override void UpdateHandFromSkeleton ()
  This function uses the tracked HandSkeleton to update the pose of the hand

- virtual void **UpdateHand** ()
- void **UpdateSensor** ()

Protected Attributes

- ControllerComponent **controllerComponent**
  The tracker controller to use for this arm

- Controller **controllerInput**
  The controller input for this humanoid

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
  The name of this sensor

- override HumanoidTracker **tracker** `[get]`
  THe tracker for this sensor

- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.CustomArm.UpdateSensorTransformFromTarget (Transform _)

Updates the sensor position and rotation based on the current position of the arm targets.

Parameters

_	Not used

override void Passer.Humanoid.CustomArm.Start (HumanoidControl *_humanoid*, Transform *targetTransform*)

Prepares the arm for tracking with the tracked controller and/or skeleton

Parameters

_humanoid	The humanoid for which this arm is tracked
targetTransform	The transform of the hand target

This will find and initialize the controllerInput for the given humanoid. It will initialize the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking for the controller and/or hand skeleton.

override void Passer.Humanoid.CustomArm.Update () `[virtual]`

Updates the arm target based on the status of the tracked controller and/or skeleton

Reimplemented from Passer.Humanoid.Tracking.Sensor.

virtual void Passer.Humanoid.CustomArm.UpdateControllerInput (ControllerSide *controllerSide*) `[protected], [virtual]`

Updates one side of the ControllerInput from the values of the tracked ControllerComponent

Parameters

controllerSide	The controller side to update

This function does nothing when the controller is not available or not tracking.

override void Passer.Humanoid.CustomArm.UpdateHandFromSkeleton () `[protected]`

This function uses the tracked HandSkeleton to update the pose of the hand

This function does nothing when the hand skeleton is not available or not tracking.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Custom/CustomArm.cs

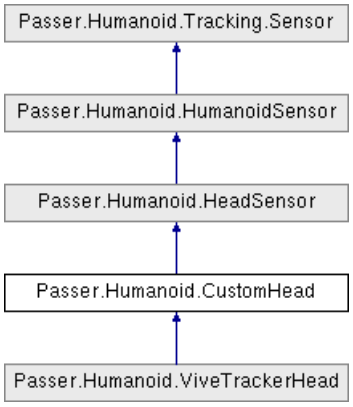
Passer.Humanoid.CustomHead Class Reference

Description

A custom sensor used on the head of a humanoid.

This tracking option supports a custom developed SensorComponent for the head.

Inheritance diagram for Passer.Humanoid.CustomHead:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the head for tracking with the tracked sensor

- override void Update ()
  Updates the head target based on the status of the tracke sensor

- virtual void **CheckSensor** (HeadTarget headTarget)
- virtual void **Init** (HeadTarget headTarget)
- void **InitController** (SerializedProperty sensorProp, HeadTarget target)
- void **RemoveController** (SerializedProperty sensorProp)
- virtual void **CheckSensorTransform** ()
- virtual void **SetSensor2Target** ()
- virtual void **UpdateSensorTransformFromTarget** (Transform targetTransform)
- virtual void **Stop** ()
- virtual void **RefreshSensor** ()
- virtual void **ShowSensor** (HumanoidTarget target, bool shown)

Static Public Member Functions

- static Vector3 **InverseTransformPointUnscaled** (Transform transform, Vector3 position)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- Vector3 **sensor2TargetPosition**
- Quaternion **sensor2TargetRotation**
- DeviceView **device**

The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Static Public Attributes

- const string **_name** = ""

Protected Member Functions

- virtual void **CreateSensorTransform** (string resourceName, Vector3 sensor2TargetPosition, Quaternion sensor2TargetRotation)
- virtual void **CreateSensorTransform** ()
- void **CreateSensorTransform** (Transform targetTransform, string resourceName, Vector3 _sensor2TargetPosition, Quaternion _sensor2TargetRotation)
- virtual void **UpdateNeckTargetFromHead** ()
- void **RemoveSensorTransform** ()
- void **UpdateSensorTransform** (Tracking.Sensor sensor)
- virtual void **UpdateTargetTransform** ()
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, Transform sensorTransform)
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, SensorComponent sensorComponent)
- Vector3 **GetTargetPosition** (Transform sensorTransform)
- Quaternion **GetTargetRotation** (Transform sensorTransform)
- void **UpdateSensor** ()

Static Protected Member Functions

- static Vector3 **TransformPointUnscaled** (Transform transform, Vector3 position)

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- HeadTarget **headTarget** `[get]`
- HumanoidControl **humanoid** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`

- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`

Member Function Documentation

override void Passer.Humanoid.CustomHead.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) `[virtual]`

Prepares the head for tracking with the tracked sensor

Parameters

_humanoid	The humanoid for which this head is tracked
targetTransform	The transform of the head target

It will initialize the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking of the sensor.

Reimplemented from Passer.Humanoid.HeadSensor.

override void Passer.Humanoid.CustomHead.Update () `[virtual]`

Updates the head target based on the status of the tracke sensor

Reimplemented from Passer.Humanoid.HeadSensor.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Custom/CustomHead.cs

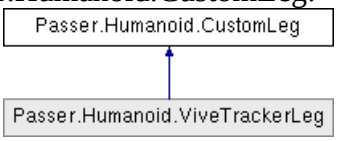
Passer.Humanoid.CustomLeg Class Reference

Description

A custom sensor used on the leg of a humanoid.

This tracking option supports a custom developed SensorComponent on the leg or foot.

Inheritance diagram for Passer.Humanoid.CustomLeg:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **SetSensor2Target** ()
  Updates the leg targets based on the current sensor position and rotation

- override void UpdateSensorTransformFromTarget (Transform _)
  Updates the sensor position and rotation based on the current position of the leg targets

- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the leg for tracking with the sensor

- override void Update ()
  Updates the leg target based on the status of the tracked sensor

Static Public Member Functions

- static Rotation **CalculateLegOrientation** (Vector joint1Position, Vector joint2Position, Rotation hipsRotation)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- LegBones **attachedBone** = LegBones.Foot
  The bone on the leg controlled by the sensor

- bool **isLeft**
- TargetData **upperLeg**
- TargetData **lowerLeg**
- TargetData **foot**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- void **UpdateSensor** ()

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.CustomLeg.UpdateSensorTransformFromTarget (Transform _)

Updates the sensor position and rotation based on the current position of the leg targets

Parameters

_	Not used

override void Passer.Humanoid.CustomLeg.Start (HumanoidControl *_humanoid*, Transform *targetTransform*)

Prepares the leg for tracking with the sensor

Parameters

_humanoid	The humanoid for which this leg is tracked
targetTransform	The transform of the foot target

It will initialze the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking of the sensor.

override void Passer.Humanoid.CustomLeg.Update () `[virtual]`

Updates the leg target based on the status of the tracked sensor

Reimplemented from Passer.Humanoid.Tracking.Sensor.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Custom/CustomLeg.cs

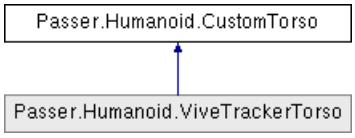
Passer.Humanoid.CustomTorso Class Reference

Description

A custom sensor used on the torso of a humanoid.

This tracking option supports a custom developed SensorComponent for the torso.

Inheritance diagram for Passer.Humanoid.CustomTorso:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **SetSensor2Target** ()
  Updates the torso targets based on the current sensor position and rotation

- override void UpdateSensorTransformFromTarget (Transform _)
  Updates the sensor position and rotation based on the current position of the torso targets

- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the torso for tracking with the sensor

- override void Update ()
  Updates the torso targets based on the status of the tracked sensor

Static Public Member Functions

- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- TorsoBones **attachedBone** = TorsoBones.Hips
  The bone on the torso controlled by the sensor

- TargetData **chest**
- TargetData **spine**
- TargetData **hips**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- void **UpdateSensor** ()

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.CustomTorso.UpdateSensorTransformFromTarget (Transform _)

Updates the sensor position and rotation based on the current position of the torso targets

Parameters

_	Not used

override void Passer.Humanoid.CustomTorso.Start (HumanoidControl *_humanoid*, Transform *targetTransform*)

Prepares the torso for tracking with the sensor

Parameters

_humanoid	The humanoid for which this torso is tracked
targetTransform	The transform of the hips target

It will initialze the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking of the sensor.

override void Passer.Humanoid.CustomTorso.Update () `[virtual]`

Updates the torso targets based on the status of the tracked sensor

Reimplemented from Passer.Humanoid.Tracking.Sensor.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Custom/CustomTorso.cs

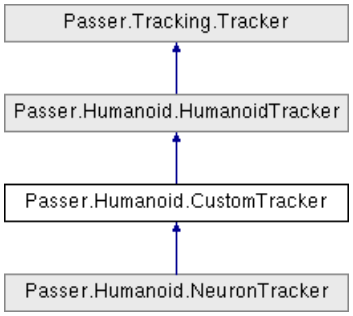
Passer.Humanoid.CustomTracker Class Reference

Description

A tracker wich can be used for custom tracking solutions

THis tracking option supports custom trackers and sensors for a humanoid.

Inheritance diagram for Passer.Humanoid.CustomTracker:



Public Types

- enum Status { Unavailable, Present, Tracking }
  The tracking status

Public Member Functions

- override void UpdateTracker ()
  Update the tracker state

- virtual void CheckTracker (HumanoidControl humanoid)
  Check the status of the tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker)
  Function to check the status of a specific tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker, Vector3 localPosition, Quaternion localRotation)
  Function to check the status of a specific tracker

- delegate TrackerComponent TrackerGetter (Transform transform, Vector3 localPosition, Quaternion localRotation)
  Function delegate for retrieving the tracker

- virtual Vector3 **GetBonePosition** (uint actorId, FacialBone boneId)
- virtual Quaternion **GetBoneRotation** (uint actorId, FacialBone boneId)
- virtual float **GetBoneConfidence** (uint actorId, FacialBone boneId)
- virtual void StartTracker (HumanoidControl _humanoid)
  Start the tracker

- virtual void StartTracker ()
  Optional list of SubTrackers

- virtual void StopTracker ()
  Stop the tracker

- virtual void ShowTracker (bool shown)
  Show or hide the Tracker renderers

- virtual void Calibrate ()
  Calibrate the tracker

- virtual void AdjustTracking (Vector3 positionalDelta, Quaternion rotationalDelta)
  Adjust the position of the tracker by the given delat

Public Attributes

- BodySkeleton bodySkeleton
  A skeleton for the body

- HumanoidControl **humanoid**
  The humanoid for this tracker

- bool **enabled**
  Is this tracker enabled?

- Status **status**
  The tracking Status of the tracker

- TrackerComponent **trackerComponent**
  The tracking device

Protected Member Functions

- void **UpdateBodyFromSkeleton** ()
- virtual void **UpdateTorso** ()
- virtual void **UpdateLeftArm** ()
- virtual void **UpdateRightArm** ()
- virtual void **UpdateLeftLeg** ()
- virtual void **UpdateRightLeg** ()

Properties

- override string name `[get]`
  The name of this tracker

- virtual HeadSensor headSensor `[get]`
  Get the sensor for the head

- virtual ArmSensor leftHandSensor `[get]`
  Get the sensor for the left hand

- virtual ArmSensor rightHandSensor `[get]`
  Get the sensor for the right hand

- virtual TorsoSensor hipsSensor `[get]`
  Get the sensor for the hips

- virtual LegSensor leftFootSensor `[get]`
  Get the sensor for the left foot

- virtual LegSensor rightFootSensor `[get]`
  Get the sensor for the right foot

- virtual HumanoidSensor[] **sensors** `[get]`
  The sensors for this tracker

Member Enumeration Documentation

enum Passer.Tracking.Tracker.Status `[inherited]`

The tracking status

Enumerator:

Unavailable	The tracking device is not available.
Present	The tracking device is available but not tracking.
Tracking	The tracking device is actively tracking.

Member Function Documentation

override void Passer.Humanoid.CustomTracker.UpdateTracker () `[virtual]`

Update the tracker state

Reimplemented from Passer.Tracking.Tracker.

virtual void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*) `[virtual], [inherited]`

Check the status of the tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked

Reimplemented in Passer.Humanoid.UnityXRTracker, Passer.Humanoid.LeapTracker, and Passer.Humanoid.NeuronTracker.

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*) [inherited]

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker

The default position/rotation for the tracker when created will be zero

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*, Vector3 *localPosition*, Quaternion *localRotation*) [inherited]

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

delegate TrackerComponent Passer.Humanoid.HumanoidTracker.TrackerGetter (Transform *transform*, Vector3 *localPosition*, Quaternion *localRotation*) [inherited]

Function delegate for retrieving the tracker

Parameters

transform	The root transform to start the searching of the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

Returns

The tracker component found or created

The default position/rotation is relative to the humanoid's real world.

virtual void Passer.Humanoid.HumanoidTracker.StartTracker (HumanoidControl *_humanoid*) [virtual], [inherited]

Start the tracker

Reimplemented in Passer.Humanoid.LeapTracker, Passer.Humanoid.UnityXRTracker, and Passer.Humanoid.NeuronTracker.

virtual void Passer.Tracking.Tracker.StartTracker () [virtual], [inherited]

Optional list of SubTrackers

Start the tracker

virtual void Passer.Tracking.Tracker.StopTracker () `[virtual], [inherited]`

Stop the tracker

Reimplemented in Passer.Humanoid.LeapTracker.

virtual void Passer.Tracking.Tracker.ShowTracker (bool *shown*) `[virtual], [inherited]`

Show or hide the Tracker renderers

Parameters

shown	Renderers are enabled when shown == true

virtual void Passer.Tracking.Tracker.Calibrate () `[virtual], [inherited]`

Calibrate the tracker

Reimplemented in Passer.Humanoid.UnityXRTracker.

virtual void Passer.Tracking.Tracker.AdjustTracking (Vector3 *positionalDelta*, Quaternion *rotationalDelta*) `[virtual], [inherited]`

Adjust the position of the tracker by the given delat

Parameters

positionalDelta	The positional delta to apply
rotationalDelta	The rotational delta to apply

Member Data Documentation

BodySkeleton Passer.Humanoid.CustomTracker.bodySkeleton

A skeleton for the body

When this is set, the tracking will be taken from this skeleton

---

Property Documentation

override string Passer.Humanoid.CustomTracker.name `[get]`

The name of this tracker

virtual HeadSensor Passer.Humanoid.HumanoidTracker.headSensor `[get]`, `[inherited]`

Get the sensor for the head

Will return null when this sensor is not present

virtual ArmSensor Passer.Humanoid.HumanoidTracker.leftHandSensor `[get]`, `[inherited]`

Get the sensor for the left hand

Will return null when this sensor is not present

virtual ArmSensor Passer.Humanoid.HumanoidTracker.rightHandSensor `[get]`, `[inherited]`

Get the sensor for the right hand

Will return null when this sensor is not present

virtual TorsoSensor Passer.Humanoid.HumanoidTracker.hipsSensor `[get]`, `[inherited]`

Get the sensor for the hips

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.leftFootSensor `[get]`, `[inherited]`

Get the sensor for the left foot

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.rightFootSensor `[get]`, `[inherited]`

Get the sensor for the right foot

Will return null when this sensor is not present

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Custom/CustomTracker.cs

Passer.Destroyer Class Reference

Description

Destroyer of things

Inherits MonoBehaviour.

Public Member Functions

- void **SelfDestroy** ()
  Destroy this GameObject

- void Destroy (GameObject gameObject)
  Destroy the given GameObject

- void **ApplicationQuit** ()
  Exit the application

Member Function Documentation

void Passer.Destroyer.Destroy (GameObject *gameObject*)

Destroy the given GameObject

Parameters

gameObject	The GameObject to Destroy

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Sites/Scripts/Destroyer.cs

Passer.EventHandler Class Reference

Description

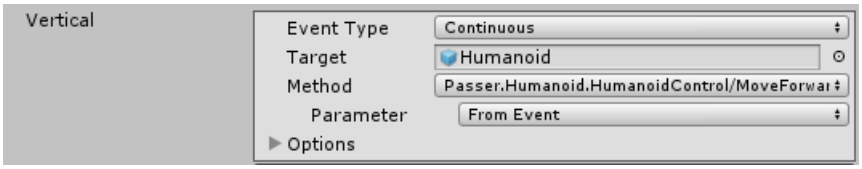
Easy universal way to attach script functions to events and statuses

Event handlers are used in many Passer components like ControllerInput, CollisionEventHandler, TriggetEventHandler, HeadTarget and Handle to call functions based on events.

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.

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



Event Type

The labels of this drop down field can be differen depending on the event, but in general they work as follows:

- /ref Never: the Target Method is never called.
- /ref On Start: the Target Method is called when the event starts.
- /ref On End: the Target Method is called when the event ends.
- /ref While Active: the Target Method is called while the event is active.
- /ref While Inactive: the Target Method is called while the event is not active.
- /ref On Change: the Target Method is called when the event starts of ends.
- /ref Always: the Target Method is called independe from the event state.

Target

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

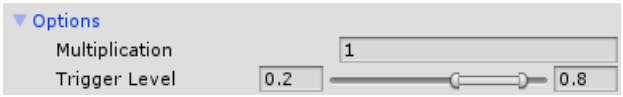
Method

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.

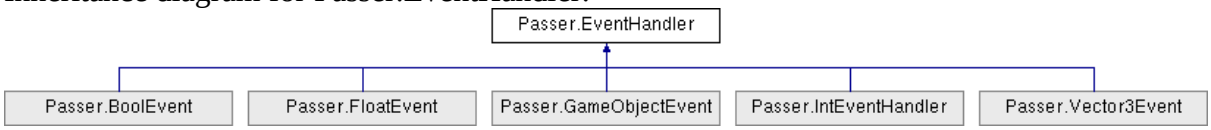
Options



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

- Inverse: the parameter value will be inverted before it is sent to the Method
- Multiplication: the parameter value will be multiplied by the given value before it is sent to the Method
- Trigger Level: 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.

Inheritance diagram for Passer.EventHandler:



Public Types

- enum Type { Never, OnStart, OnEnd, WhileActive, WhileInactive, OnChange, Continuous }
  The different types of events when the function is called
- enum OverrideMode { Prepend, Append, Replace }

Public Member Functions

- virtual void **Update** ()

Public Attributes

- Type **eventType** = Type.Continuous
  The event type for the function call

- bool **eventNetworking** = false
  For future use :-)

- FunctionCall **functionCall**
  The function to call

- bool **boolInverse** = false
  Negate the boolean state before calling event trigger

- OverrideMode **overrideMode**

Protected Member Functions

- void **UpdateAnimationParameter** ()
- virtual void **UpdateVoid** ()
- virtual void **UpdateBool** ()
- virtual void **UpdateInt** ()
- virtual void **UpdateFloat** ()
- virtual void **UpdateString** ()
- virtual void **UpdateVector3** ()
- virtual void **UpdateGameObject** ()

- virtual void **UpdateRigidbody** ()
- virtual void **UpdateString** (string s)
- virtual void **UpdateStringBool** (string s)
- virtual void **UpdateStringFloat** (string s)
- virtual void **UpdateStringInt** (string s)
- bool **CheckCondition** (bool active, bool changed, bool valueChanged)

Protected Attributes

- bool **initialized**
- bool **_boolValue**
- bool **boolChanged** = true
- int **_intValue**
- bool **intChanged**
- float **_floatValue**
- bool **floatChanged**

Properties

- virtual bool **boolValue** `[getset]`
- bool isDead `[get]`
  True when the eventHandler is dead and can be removed

Member Enumeration Documentation

enum Passer.EventHandler.Type

The different types of events when the function is called

Enumerator:

Never	The function is never called.
OnStart	The function is called when the event starts.
OnEnd	The function is called when the event ends.
WhileActive	The function is called every frame while the event is active.
WhileInactive	The function is called every frame while the event is not active.
OnChange	The function is called every time the event changes.
Continuous	The functions is called every frame.

enum Passer.EventHandler.OverrideMode

Enumerator:

Prepend	Prepend this handler before existing handlers.

Append	Append this handler after existing handlers.
Replace	Replace the topmost handler with this handler.

Property Documentation

bool Passer.EventHandler.isDead `[get]`

True when the eventHandler is dead and can be removed

A function is dead when it does nothing. This is when the functionCall is not defined or when the target of the functionCall is empty

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/Event.cs

Passer.EventHandlers< T > Class Template Reference

Description

A list of event Handlers

For each input, one or more events can be defined when the input changes.

Template Parameters

T	The type of event handler

Public Attributes

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events** = new List<T>()
  The EventHandlers

The documentation for this class was generated from the following file:

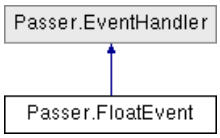
- Assets/Passer/HumanoidControl/Runtime/Tools/Events/Event.cs

Passer.FloatEvent Class Reference

Description

An event handler calling a function with a float parameter

Inheritance diagram for Passer.FloatEvent:



Public Types

- enum Type { Never, OnStart, OnEnd, WhileActive, WhileInactive, OnChange, Continuous }
  The different types of events when the function is called
- enum OverrideMode { Prepend, Append, Replace }

Public Member Functions

- **FloatEvent** (Type newEventType=Type.OnChange)
- virtual void **Update** ()

Public Attributes

- float **floatParameter**
- float **floatTriggerLow** = 0.01F
- float **floatTriggerHigh** = 0.99F
- float **multiplicationFactor** = 1
- int **intTriggerLow** = 0
- int **intTriggerHigh** = 1
- Type **eventType** = Type.Continuous
  The event type for the function call

- bool **eventNetworking** = false
  For future use :-)

- FunctionCall **functionCall**
  The function to call

- bool **boolInverse** = false
  Negate the boolean state before calling event trigger

- OverrideMode **overrideMode**

Protected Member Functions

- override void UpdateInt ()
- override void UpdateFloat ()
- override void UpdateStringInt (string s)
- override void UpdateStringFloat (string s)
- void **UpdateAnimationParameter** ()
- virtual void **UpdateVoid** ()
- virtual void **UpdateBool** ()
- virtual void **UpdateString** ()

- virtual void **UpdateString** (string s)
- virtual void **UpdateVector3** ()
- virtual void **UpdateGameObject** ()
- virtual void **UpdateRigidbody** ()
- virtual void **UpdateStringBool** (string s)
- bool **CheckCondition** (bool active, bool changed, bool valueChanged)

Protected Attributes

- bool **initialized**
- bool **_boolValue**
- bool **boolChanged** = true
- int **_intValue**
- bool **intChanged**
- float **_floatValue**
- bool **floatChanged**

Properties

- virtual float? **value** `[getset]`
- virtual bool **boolValue** `[getset]`
- bool isDead `[get]`
  True when the eventHandler is dead and can be removed

Member Enumeration Documentation

enum Passer.EventHandler.Type `[inherited]`

The different types of events when the function is called

Enumerator:

Never	The function is never called.
OnStart	The function is called when the event starts.
OnEnd	The function is called when the event ends.
WhileActive	The function is called every frame while the event is active.
WhileInactive	The function is called every frame while the event is not active.
OnChange	The function is called every time the event changes.
Continuous	The functions is called every frame.

enum Passer.EventHandler.OverrideMode `[inherited]`

Enumerator:

Prepend	Prepend this handler before existing handlers.
Append	Append this handler after existing handlers.
Replace	Replace the topmost handler with this handler.

Member Function Documentation

override void Passer.FloatEvent.UpdateInt () `[protected]`, `[virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.FloatEvent.UpdateFloat () `[protected]`, `[virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.FloatEvent.UpdateStringInt (string *s*) `[protected]`, `[virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.FloatEvent.UpdateStringFloat (string *s*) `[protected]`, `[virtual]`

Reimplemented from Passer.EventHandler.

Property Documentation

bool Passer.EventHandler.isDead `[get]`, `[inherited]`

True when the eventHandler is dead and can be removed

A function is dead when it does nothing. This is when the functionCall is not defined or when the target of the functionCall is empty

---

The documentation for this class was generated from the following file:

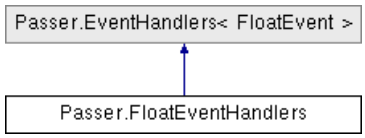
- Assets/Passer/HumanoidControl/Runtime/Tools/Events/FloatEvent.cs

Passer.FloatEventHandlers Class Reference

Description

A list of event handlers with a float parameter

Inheritance diagram for Passer.FloatEventHandlers:



Public Attributes

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events**
  The EventHandlers

Properties

- float **value** `[getset]`

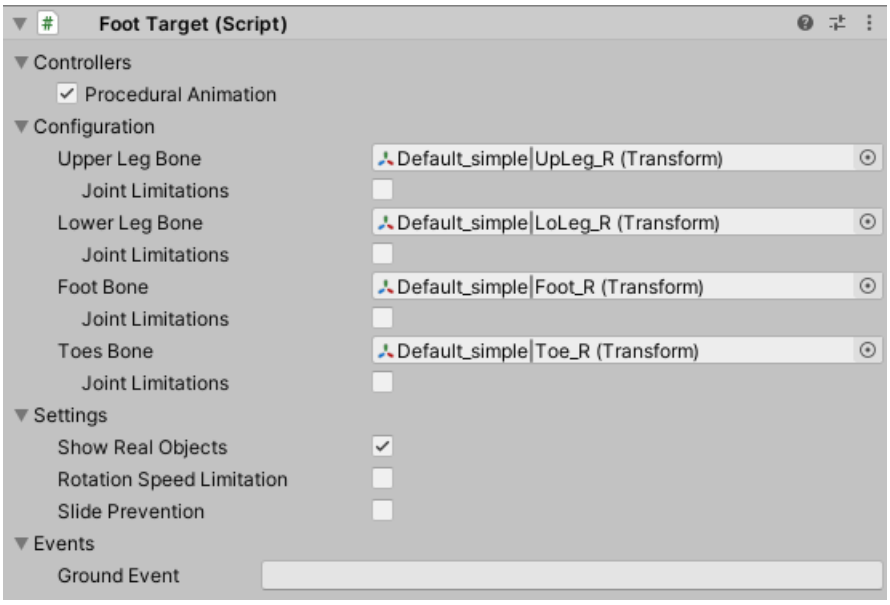
The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/FloatEvent.cs

Passer.Humanoid.FootTarget Class Reference

Description

Humanoid Control options for leg related things



Sensors

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. 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.

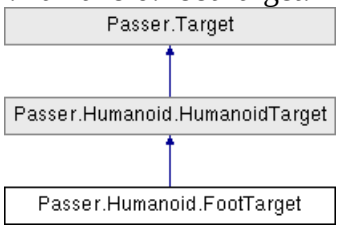
Settings

- Rotation Speed Limitation
- Slide Prevention

Events

- Ground Event

Inheritance diagram for Passer.Humanoid.FootTarget:



Public Member Functions

- override void InitSensors ()
- override void StartSensors ()
- Bone **GetBoneId** (bool isLeft, LegBones armBone)
- TargetedBone **GetTargetBone** (LegBones boneID)
- virtual void **ClearBones** ()
- override Transform GetDefaultTarget (HumanoidControl humanoid)
- void **RetrieveBones** ()
- void **ShowControllers** (bool shown)
- override void InitAvatar ()
- override void NewComponent (HumanoidControl _humanoid)
- override void InitComponent ()
- override void StartTarget ()
- bool **IsInTPose** ()
- override void MatchTargetsToAvatar ()
- override void InitializeTrackingConfidence ()
  Sets the tracking confidence before all tracking information is updated

- override void UpdateTarget ()
- override void UpdateMovements (HumanoidControl humanoid)
- override void CopyTargetToRig ()
- override void CopyRigToTarget ()
- void **UpdateSensorsFromTarget** ()
- void **CheckGrounded** ()
- void **OnDrawGizmos** ()
- void **OnDrawGizmosSelected** ()
- virtual void **DrawTensions** ()
- virtual void **StopSensors** ()

Static Public Member Functions

- static FootTarget **CreateTarget** (FootTarget oldTarget)
- static FootTarget **SetTarget** (HumanoidControl humanoid, Transform targetTransform, bool isLeft)
- static bool **IsInitialized** (HumanoidControl humanoid)
- static void DetermineTarget (HumanoidControl humanoid, bool isLeft)
  Checks whether the humanoid has a FootTarget and adds one if none has been found

- static Vector **ToVector** (Vector3 vector3)
- static Vector3 **ToVector3** (Vector position)
- static Rotation **ToRotation** (Quaternion quaternion)

- static Quaternion **ToQuaternion** (Rotation orientation)
- static void **SetRotation** (Transform transform, Rotation orientation)
- static void **DrawTarget** (Confidence confidence, Transform target, Vector3 direction, float length)
- static void **DrawTargetBone** (TargetedBone bone, Vector3 direction)
- static void **DrawTargetBone** (TargetTransform target, Vector3 direction)
- static void **DrawAvatarBone** (TargetedBone bone, Vector3 direction)
- static void **DrawAvatarBone** (BoneTransform bone, Vector3 direction)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, Bone boneId, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, HumanBodyBones boneID, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, params string[] boneNames)
- static void **GetDefaultTargetBone** (Animator rig, ref Transform boneTransform, Bone boneID, params string[] boneNames)
- static List< Collider > **SetColliderToTrigger** (GameObject obj)
- static List< Collider > **SetColliderToTrigger** (Rigidbody rb)
- static void **UnsetColliderToTrigger** (List< Collider > colliders)
- static void **UnsetColliderToTrigger** (List< Collider > colliders, Collider collider)

Public Attributes

- bool **isLeft**
- Side **side**
- FootTarget **otherFoot**
- LegMovements **legMovements** = new LegMovements()
- bool rotationSpeedLimitation = false
  Limits the maximum rotation speed of the joints.

- LegAnimator **legAnimator** = new LegAnimator()
- ViveTrackerLeg **viveTracker** = new ViveTrackerLeg()
- Kinect1Leg **kinect1** = new Kinect1Leg()
- Kinect2Foot **kinect2** = new Kinect2Foot()
- Kinect4Leg **kinect4** = new Kinect4Leg()
- AstraLeg **astra** = new AstraLeg()
- OptitrackLeg **optitrack** = new OptitrackLeg()
- AntilatencyLeg **antilatency** = new AntilatencyLeg()
- CustomLeg **custom** = new CustomLeg()
- TargetedUpperLegBone **upperLeg**
- TargetedLowerLegBone **lowerLeg**
- TargetedFootBone **foot**
- TargetedToesBone **toes**
- bool **jointLimitations** = true
- bool **slidePrevention** = false
  Prevents movement of the foot when it is on the ground.

- bool **physics** = true
- GameObjectEventHandlers groundEvent
  Use to call functions based on the foot touching the ground.

- Transform **ground**
- Vector3 **groundNormal** = Vector3.up
- Vector3 **groundTranslation** = Vector3.zero
- float **groundDistance** = 0
- float **soleThicknessFoot**
- float **soleThicknessToes**

- HumanoidControl **humanoid**

Static Public Attributes

- const float **maxUpperLegAngle** = 120
- const float **maxLowerLegAngle** = 130
- const float **maxFootAngle** = 50
- const float **maxToesAngle** = 30
- static readonly Vector3 **minLeftUpperLegAngles** = new Vector3(-130, -45, -50)
- static readonly Vector3 **maxLeftUpperLegAngles** = new Vector3(30, 40, 30)
- static readonly Vector3 **minRightUpperLegAngles** = new Vector3(-130, -40, -30)
- static readonly Vector3 **maxRightUpperLegAngles** = new Vector3(30, 45, 50)
- static readonly Vector3 **minLeftLowerLegAngles** = new Vector3(-15, float.NaN, float.NaN)
- static readonly Vector3 **maxLeftLowerLegAngles** = new Vector3(130, float.NaN, float.NaN)
- static readonly Vector3 **minRightLowerLegAngles** = new Vector3(-15, float.NaN, float.NaN)
- static readonly Vector3 **maxRightLowerLegAngles** = new Vector3(130, float.NaN, float.NaN)
- static readonly Vector3 **minLeftFootAngles** = new Vector3(-45, 0, -30)
- static readonly Vector3 **maxLeftFootAngles** = new Vector3(70, 0, 20)
- static readonly Vector3 **minRightFootAngles** = new Vector3(-45, 0, -20)
- static readonly Vector3 **maxRightFootAngles** = new Vector3(50, 0, 30)
- static readonly Vector3 **minLeftToesAngles** = new Vector3(-70, float.NaN, float.NaN)
- static readonly Vector3 **maxLeftToesAngles** = new Vector3(45, float.NaN, float.NaN)
- static readonly Vector3 **minRightToesAngles** = new Vector3(-70, float.NaN, float.NaN)
- static readonly Vector3 **maxRightToesAngles** = new Vector3(45, float.NaN, float.NaN)

Protected Member Functions

- override void UpdateSensors ()
- void **InitSubTargets** ()
- virtual void **UpdateEvents** ()
- override void DrawTargetRig (HumanoidControl humanoid)
- override void DrawAvatarRig (HumanoidControl humanoid)
- virtual void **DrawTensionGizmo** (TargetedBone targetedBone)

Protected Attributes

- bool **_showRealObjects** = true

Properties

- override Passer.Sensor **animator** `[get]`
- override TargetedBone **main** `[get]`
- virtual bool **showRealObjects** `[getset]`

  show the target meshes

Member Function Documentation

override void Passer.Humanoid.FootTarget.InitSensors () `[virtual]`

Implements Passer.Target.

override void Passer.Humanoid.FootTarget.StartSensors () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.FootTarget.UpdateSensors () `[protected], [virtual]`

Reimplemented from Passer.Target.

override Transform Passer.Humanoid.FootTarget.GetDefaultTarget (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.InitAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.NewComponent (HumanoidControl *_humanoid*) `[virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.InitComponent () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.FootTarget.StartTarget () `[virtual]`

Implements Passer.Target.

static void Passer.Humanoid.FootTarget.DetermineTarget (HumanoidControl *humanoid*, bool *isLeft*) `[static]`

Checks whether the humanoid has a FootTarget and adds one if none has been found

Parameters

humanoid	The humanoid to check
isLeft	Is this the left foot?

override void Passer.Humanoid.FootTarget.MatchTargetsToAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.InitializeTrackingConfidence () `[virtual]`

Sets the tracking confidence before all tracking information is updated

Implements Passer.Target.

override void Passer.Humanoid.FootTarget.UpdateTarget () `[virtual]`

Implements Passer.Target.

override void Passer.Humanoid.FootTarget.UpdateMovements (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.CopyTargetToRig () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.CopyRigToTarget () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.DrawTargetRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.FootTarget.DrawAvatarRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

Member Data Documentation

bool Passer.Humanoid.FootTarget.rotationSpeedLimitation = false

Limits the maximum rotation speed of the joints.

This can result in more natural movements with optical tracking solutions.

GameObjectEventHandlers Passer.Humanoid.FootTarget.groundEvent

```
Initial value:= new GameObjectEventHandlers() {
            id = 1,
            label = "Ground Event",
            tooltip =
                "Call function based on ground standing\n" +
                "Parameter: the ground object",
            eventTypeLabels = new string[] {
                "Never",
                "On Grounded",
                "On not Grounded",
                "While Grounded",
                "While not Grounded",
                "When Ground Changes",
                "Always"
            },
            fromEventLabel = "ground.gameObject"
        }
```

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

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Targets/FootTarget.cs

Passer.FunctionCall Class Reference

Description

A function which can be called

Classes

- class Parameter
  Function Parameter

Public Types

- enum **ParameterType** { **Void**, **Float**, **Int**, **Bool**, **Vector3**, **GameObject**, **Rigidbody**, **String** }
- enum Networking { **No**, **Yes** }
  For future use...

Public Member Functions

- FunctionCall (GameObject targetGameObject, string methodName)
  Creates a new FunctionCall

- Parameter AddParameter ()
  Adds a new Parameter to the function call

- virtual void Execute (Networking networking=Networking.No)
  Execute the void function call

- void Execute (bool value, Networking networking=Networking.No)
  Execute a function call with a boolean parameter

- void Execute (int value)
  Call the function with an integer parameter

- virtual void Execute (float value)
  Call the function with a float parameter

- void Execute (Vector3 value)
  Call the function with a Vector3 parameter

- void Execute (GameObject value)
  Call the funtion with a GameObject parameter

- void ExecuteString (string s, bool value)
  Call the function with a string and a boolean parameter

- void ExecuteString (string s, float value)
  Call the function with a string and a float parameter

- void ExecuteString (string s, int value)
  Call the function with a string and a integer parameter

- void **Execute** (string value, bool networkSync=false)
- void **CreateTargetMethodString** (UnityEngine.Object target, string methodName, bool fromEvent, string stringConstant)

Static Public Member Functions

- static Type **ToSystemType** (ParameterType parameterType)
- static void **Execute** (GameObject target, string methodName)
- static void **Execute** (GameObject target, string methodName, bool boolValue)
- static void **Execute** (GameObject target, string methodName, float floatValue)
- static GameObject GetGameObject (UnityEngine.Object obj)
  Gets the GameObject for the Object.

- static void **Execute** (GameObject target, string methodName, string stringValue)

Public Attributes

- GameObject **targetGameObject**
  The target GameObject on which the function should be called.

- string methodName
  The name of the method to call on the GameObject

- Parameter[] **parameters**
  For future use...

Protected Member Functions

- delegate void **Method** ()
- delegate void **MethodBool** (bool value)
- delegate void **MethodFloat** (float value)
- delegate void **MethodInt** (int value)
- delegate void **MethodVector3** (Vector3 value)
- delegate void **MethodGameObject** (GameObject value)
- delegate void **MethodRigidbody** (Rigidbody value)
- delegate void **MethodStringBool** (string s, bool value)
- delegate void **MethodStringFloat** (string s, float value)
- delegate void **MethodStringInt** (string s, int value)
- void Execute (Rigidbody value)
  Call the function with a Rigidbody parameter

- virtual void **GetTargetMethod** ()
- void **CreateAnimationParameterMethod** (GameObject target, string fullMethodName)
- void **CreateTargetMethod** (UnityEngine.Object target, string methodName, bool fromEvent, bool boolConstant)
- void **CreateTargetMethod** (UnityEngine.Object target, string methodName, bool fromEvent, int intConstant)
- void **CreateTargetMethod** (UnityEngine.Object target, string methodName, bool fromEvent, float floatConstant)
- void **CreateTargetMethod** (UnityEngine.Object target, string methodName, bool fromEvent, Vector3 vectorConstant)

- void **CreateTargetMethod** (UnityEngine.Object target, string methodName, bool fromEvent, GameObject gameObjectConstant)
- void **CreateTargetMethod** (UnityEngine.Object target, string methodName, bool fromEvent, Rigidbody rigidbodyConstant)
- delegate void **MethodString** (string value)

Static Protected Member Functions

- static UnityEngine.Object **GetComponent** (GameObject target, string fullMethodName)
- static UnityEngine.Object **GetComponent** (GameObject target, string fullMethodName, out string methodName)
- static Method **CreateMethod** (GameObject target, string fullMethodName)
- static Method **CreateMethod** (UnityEngine.Object target, string methodName)
- static MethodBool **CreateMethod** (GameObject target, string fullMethodName, bool boolConstant)
- static MethodBool **CreateMethodBool** (UnityEngine.Object target, string methodName)
- static MethodFloat **CreateMethodFloat** (UnityEngine.Object target, string methodName)
- static MethodString **CreateMethodString** (UnityEngine.Object target, string methodName)

Protected Attributes

- INetworkObject **networkObject**
- Delegate **targetDelegate**

Constructor & Destructor Documentation

Passer.FunctionCall.FunctionCall (GameObject *targetGameObject*, string *methodName*)

Creates a new FunctionCall

Parameters

targetGameObject	The GameObject on which the functioncall is executed
methodName	The full method name: component/methodName

Member Function Documentation

Parameter Passer.FunctionCall.AddParameter ()

Adds a new Parameter to the function call

Returns

The new Parameter

virtual void Passer.FunctionCall.Execute (Networking *networking* = `Networking.No`)`[virtual]`

Execute the void function call

Parameters

networking	For future use...

void Passer.FunctionCall.Execute (bool *value*, Networking *networking* = `Networking.No`)

Execute a function call with a boolean parameter

Parameters

value	The boolean value to pass to the function
networking	For future use...

void Passer.FunctionCall.Execute (int *value*)

Call the function with an integer parameter

Parameters

value	The integer value to pass to the function

virtual void Passer.FunctionCall.Execute (float *value*) `[virtual]`

Call the function with a float parameter

Parameters

value	The float value to pass to the function

void Passer.FunctionCall.Execute (Vector3 *value*)

Call the function with a Vector3 parameter

Parameters

value	The Vector3 value to pass to the function

void Passer.FunctionCall.Execute (GameObject *value*)

Call the funtion with a GameObject parameter

Parameters

value	The GameObject value to pass to the function

void Passer.FunctionCall.Execute (Rigidbody *value*) `[protected]`

Call the function with a Rigidbody parameter

Parameters

value	The Rigidbody value to pass to the function

void Passer.FunctionCall.ExecuteString (string *s*, bool *value*)

Call the function with a string and a boolean parameter

Parameters

s	The string value to pass to the function as the first parameter
value	The boolean value to pass to the function as the second parameter

void Passer.FunctionCall.ExecuteString (string *s*, float *value*)

Call the function with a string and a float parameter

Parameters

s	The string value to pass to the function as the first parameter
value	The float value to pass to the function as the second parameter

void Passer.FunctionCall.ExecuteString (string *s*, int *value*)

Call the function with a string and a integer parameter

Parameters

s	The string value to pass to the function as the first parameter
value	The integer value to pass to the function as the second parameter

static GameObject Passer.FunctionCall.GetGameObject (UnityEngine.Object *obj*) `[static]`

Gets the GameObject for the Object.

Parameters

obj	

Returns

Member Data Documentation

string Passer.FunctionCall.methodName

The name of the method to call on the GameObject

The format of this string is <fully qualified component type>.<function name>

The documentation for this class was generated from the following files:

- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/FunctionCall.cs
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/FunctionCallString.cs

Passer.GameObjectEvent Class Reference

Description

An EventHandler calling a function with a GameObject type parameter

Inheritance diagram for Passer.GameObjectEvent:

Public Types

- enum Type { Never, OnStart, OnEnd, WhileActive, WhileInactive, OnChange, Continuous }
  The different types of events when the function is called
- enum OverrideMode { Prepend, Append, Replace }

Public Member Functions

- **GameObjectEvent** (Type eventType)
- void **SetMethod** (Type newEventType, UnityAction voidAction)
- void **SetMethod** (Type newEventType, UnityAction< bool > boolAction)
- void **SetMethod** (Type newEventType, UnityAction< GameObject > gameObjectAction)
- virtual void **Update** ()

Public Attributes

- Type **eventType** = Type.Continuous
  The event type for the function call

- bool **eventNetworking** = false
  For future use :-)

- FunctionCall **functionCall**
  The function to call

- bool **boolInverse** = false
  Negate the boolean state before calling event trigger

- OverrideMode **overrideMode**

Protected Member Functions

- override void UpdateGameObject ()
- override void UpdateBool ()
- void **UpdateAnimationParameter** ()
- virtual void **UpdateVoid** ()
- virtual void **UpdateInt** ()
- virtual void **UpdateFloat** ()
- virtual void **UpdateString** ()
- virtual void **UpdateString** (string s)
- virtual void **UpdateVector3** ()
- virtual void **UpdateRigidbody** ()
- virtual void **UpdateStringBool** (string s)

- virtual void **UpdateStringFloat** (string s)
- virtual void **UpdateStringInt** (string s)
- bool **CheckCondition** (bool active, bool changed, bool valueChanged)

Protected Attributes

- GameObject **gameObject**
- bool **objectChanged**
- bool **initialized**
- bool **_boolValue**
- bool **boolChanged** = true
- int **_intValue**
- bool **intChanged**
- float **_floatValue**
- bool **floatChanged**

Properties

- GameObject? **value** `[getset]`
  The GameObject value for this event

- virtual bool **boolValue** `[getset]`
- bool isDead `[get]`
  True when the eventHandler is dead and can be removed

Member Enumeration Documentation

enum Passer.EventHandler.Type `[inherited]`

The different types of events when the function is called

Enumerator:

Never	The function is never called.
OnStart	The function is called when the event starts.
OnEnd	The function is called when the event ends.
WhileActive	The function is called every frame while the event is active.
WhileInactive	The function is called every frame while the event is not active.
OnChange	The function is called every time the event changes.
Continuous	The functions is called every frame.

enum Passer.EventHandler.OverrideMode `[inherited]`

Enumerator:

Prepend	Prepend this handler before existing handlers.
Append	Append this handler after existing handlers.
Replace	Replace the topmost handler with this handler.

Member Function Documentation

override void Passer.GameObjectEvent.UpdateGameObject () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.GameObjectEvent.UpdateBool () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

Property Documentation

bool Passer.EventHandler.isDead `[get], [inherited]`

True when the eventHandler is dead and can be removed

A function is dead when it does nothing. This is when the functionCall is not defined or when the target of the functionCall is empty

---

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/GameObjectEvent.cs

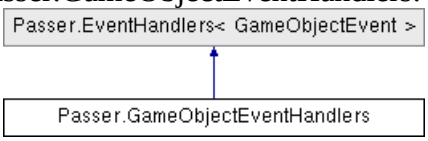
Passer.GameObjectEventHandlers Class Reference

Description

A list of event handlers with GameObject parameters

This is used to implement a list of functions which should be called with a GameObject as parameter.

Inheritance diagram for Passer.GameObjectEventHandlers:



Public Member Functions

- void **SetMethod** (EventHandler.Type eventType, UnityAction voidAction, int index=0)

Public Attributes

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events**
  The EventHandlers

Properties

- GameObject **value** `[getset]`

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/GameObjectEvent.cs

Passer.Handle Class Reference

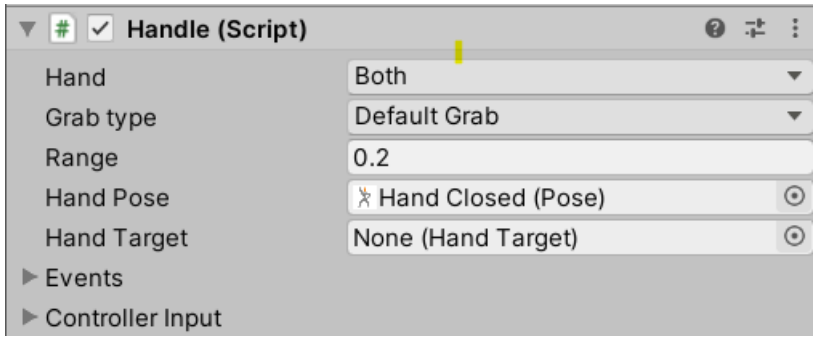
Description

Component to specify behaviour when grabbing a GameObject

A 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 Handle 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.

Sockets

Humanoid hands actually have a socket inside which receives the handle. Sockets can also be used at different places to receive handles.



- Hand
- Grab Type
- Range
- Hand Pose
- Hand Target

Events

- Grabbed Events

Controller Input

If a hand grabs an handle you can override the ControllerInput for the side of the hand which grabbed the handle during the time it is held. This enables you to assign a button to shoot a weapon when it is held for example. The configuration is similar to the normal ControllerInput. The difference is that empty entries do not override the ControllerInput configuration. When the handle is released by the hand, the ControllerInput is restored to the original configuration.

Inherits MonoBehaviour.
Inherited by Passer.Humanoid.BallHandle, Passer.Humanoid.BarHandle, and

Passer.Humanoid.NoGrab.

Public Types

- enum GrabType { DefaultGrab, BarGrab, BallGrab, RailGrab, AnyGrab, NoGrab }
  The way in which the hand can grab the handle
- enum Hand { Both, Left, Right }
  Which hand can pick up this handle?

Public Member Functions

- Vector3 **TranslationTo** (Vector3 position)
- Quaternion **RotationTo** (Quaternion orientation)
- void **ReleaseFromSocket** ()
  Releases this handle from the socket

- virtual void **UpdateGrabbed** ()
- void **OnTriggerEnter** (Collider other)

Static Public Member Functions

- static void **Create** (GameObject gameObject, Humanoid.HandTarget handTarget)
- static Handle **GetClosestHandle** (Transform transform, Vector3 position, Hand hand, float range=float.PositiveInfinity)
  Finds the handle on the transform closest to the given position

- static Handle **GetClosestHandle** (Transform transform, Vector3 position, float range=float.PositiveInfinity)
  Finds the handle on the transform closest to the given position

- static Handle GetClosestHandle (Transform transform, Vector3 position, Hand hand)
  Finds the handle on the transform closest to the given position

Public Attributes

- GrabType **grabType**
  Select how the hand will grab the handle

- bool **sticky** = false
  Sticky handles will not release unless release sticky is used

- float **range** = 0.2f
  The range within the handle will work. Outside this range normal grabbing is used.

- ControllerEventHandlers[] controllerInputEvents
  The Controller input which will be active while the Handle is grabbed.

- Socket socket
  The socket holding the handle

- Hand hand
  Selects which hand can pick up this handle

- Pose pose
  The Hand Pose which will be active while the Handle is grabbed.

- bool **useNearPose**
- int **nearPose**
- Humanoid.HandTarget handTarget
  The hand target which grabbed the handle.

- bool **isHeld**
  The Handle is held by a socket

- GameObjectEventHandlers grabbedEvent

Protected Member Functions

- virtual void **Update** ()

Static Protected Member Functions

- static Handle **GetClosestHandle** (Handle[] handles, Vector3 position, Hand hand)
- static bool **CheckHand** (Handle handle, Hand hand)

Protected Attributes

- Mesh **gizmoMesh**

Member Enumeration Documentation

enum Passer.Handle.GrabType

The way in which the hand can grab the handle

Enumerator:

DefaultGrab	Same as BarGrab.
BarGrab	The hand will grab the handle in the specified position and rotation.
BallGrab	The hand will grab the handle in the specified position, the rotation is free.
RailGrab	The hand will grab the handle along the the specified position, the rotation around the rail is free.
AnyGrab	The hand will grab the handle in any position or rotation.
NoGrab	The hand cannot grab the handle or the gameObject.

enum Passer.Handle.Hand

Which hand can pick up this handle?

Enumerator:

Both	The handle can be picked up by any hand.
Left	The handle can only be grabbed by the left hand.
Right	The handle can only be grabbed by the right hand.

Member Function Documentation

static Handle Passer.Handle.GetClosestHandle (Transform *transform*, Vector3 *position*, Hand *hand*) `[static]`

Finds the handle on the transform closest to the given position

Handles not in socket have lower priority

Member Data Documentation

ControllerEventHandlers [] Passer.Handle.controllerInputEvents

```
Initial value:= {
            new ControllerEventHandlers() { label = "Vertical", id = 0 },
            new ControllerEventHandlers() { label = "Horizontal", id = 1 },
            new ControllerEventHandlers() { label = "Stick Button", id = 2 },
            new ControllerEventHandlers() { label = "Vertical", id = 3 },
            new ControllerEventHandlers() { label = "Horizontal", id = 4 },
            new ControllerEventHandlers() { label = "Stick Button", id = 5 },
            new ControllerEventHandlers() { label = "Button 1", id = 6 },
            new ControllerEventHandlers() { label = "Button 2", id = 7 },
            new ControllerEventHandlers() { label = "Button 3", id = 8 },
            new ControllerEventHandlers() { label = "Button 4", id = 9 },
            new ControllerEventHandlers() { label = "Trigger 1", id = 10 },
            new ControllerEventHandlers() { label = "Trigger 2", id = 11 },
            new ControllerEventHandlers() { label = "Option", id = 12 },
        }
```

The Controller input which will be active while the Handle is grabbed.

Version

v3

Socket Passer.Handle.socket

The socket holding the handle

This parameter contains the socket holding the handle when it is held by a socket.

Hand Passer.Handle.hand

Selects which hand can pick up this handle

Some handles may only be grabbed by the left or right hand.

Pose Passer.Handle.pose

The Hand Pose which will be active while the Handle is grabbed.

See also: Hand Pose

Humanoid.HandTarget Passer.Handle.handTarget

The hand target which grabbed the handle.

When the Handle is grabbed this will contain the HandTarget of the grabbing hand. When the HandTarget of a Handle is set in the editor while editing the scene the applicable hand will try to grab the Handle. This is null when the handle is not grabbed by a hand.

GameObjectEventHandlers Passer.Handle.grabbedEvent

```
Initial value:= new GameObjectEventHandlers() {
            label = "Grab Event",
            tooltip =
                "Call functions using the grabbing status\n" +
                "Parameter: the grabbed object",
            eventTypeLabels = new string[] {
                "Nothing",
                "On Grab Start",
                "On Let Go",
                "While Holding",
                "While Not Holding",
                "On Grab Change",
                "Always"
            },

            fromEventLabel = "socket.gameObject"
        }
```

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/Handle.cs

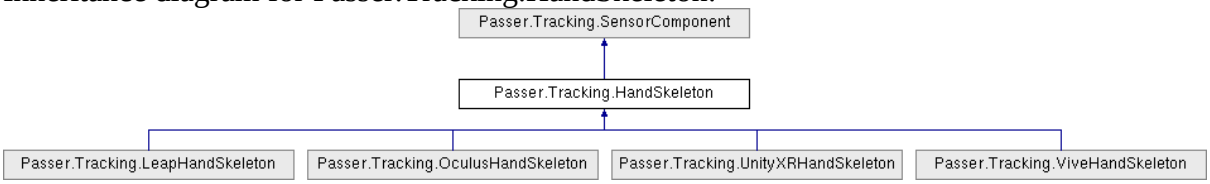
Passer.Tracking.HandSkeleton Class Reference

Description

The representation of a tracked hand

The bone hierarchy will be created below this transform at runtime

Inheritance diagram for Passer.Tracking.HandSkeleton:



Public Types

- enum **BoneId** { **Invalid** = -1, **Hand** = 0, **ThumbProximal**, **ThumbIntermediate**, **ThumbDistal**, **ThumbTip**, **IndexMetaCarpal**, **IndexProximal**, **IndexIntermediate**, **IndexDistal**, **IndexTip**, **MiddleMetaCarpal**, **MiddleProximal**, **MiddleIntermediate**, **MiddleDistal**, **MiddleTip**, **RingMetaCarpal**, **RingProximal**, **RingIntermediate**, **RingDistal**, **RingTip**, **LittleMetacarpal**, **LittleProximal**, **LittleIntermediate**, **LittleDistal**, **LittleTip**, **Forearm**, **Count** }

Public Member Functions

- virtual GameObject **CreateHandSkeleton** (Transform trackerTransform, bool isLeft, bool showRealObjects)
- virtual Transform **GetForearmBone** ()
- virtual Transform **GetWristBone** ()
- virtual Transform GetBoneTransform (Finger finger, FingerBone fingerBone)
  Gets the transform of the tracked bone

- virtual int **GetBoneId** (Finger finger, FingerBone fingerBone)
- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

- virtual void UpdateComponent ()
  Update the component manually

Static Public Member Functions

- static HandSkeleton **FindHandSkeleton** (Transform trackerTransform, bool isLeft)

Public Attributes

- bool **isLeft**
  True: this is a left hand. Fale: this is a right hand

- new bool **show** = false
  Determines whether this skeleton should be rendered

- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- virtual void InitializeSkeleton ()
  This function is used to initialize the tracked bones

- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()
- virtual void Awake ()
  Initializes the sensor.

- virtual void Start ()
  Starts the sensor

Protected Attributes

- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

virtual void Passer.Tracking.HandSkeleton.InitializeSkeleton () `[protected]`, `[virtual]`

This function is used to initialize the tracked bones

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.LeapHandSkeleton, Passer.Tracking.ViveHandSkeleton, and Passer.Tracking.OculusHandSkeleton.

virtual Transform Passer.Tracking.HandSkeleton.GetBoneTransform (Finger *finger*, FingerBone *fingerBone*) `[virtual]`

Gets the transform of the tracked bone

Parameters

finger	The requested finger
fingerBone	The requested bone in the finger

Returns

The tracked bon transform of *null* if it does not exist

Reimplemented in Passer.Tracking.LeapHandSkeleton.

virtual void Passer.Tracking.SensorComponent.Awake () `[protected], [virtual], [inherited]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.Start () `[protected], [virtual], [inherited]`

Starts the sensor

Does nothing at this moment.

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, and Passer.Tracking.HydraController.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual], [inherited]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

virtual void Passer.Tracking.SensorComponent.UpdateComponent () `[virtual], [inherited]`

Update the component manually

This function is meant to be overridden

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.LeapHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent,

Passer.Tracking.HydraController, Passer.Tracking.OculusController, Passer.Tracking.OculusHandSkeleton, and Passer.Tracking.OculusHmd.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true [inherited]

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

The documentation for this class was generated from the following file:

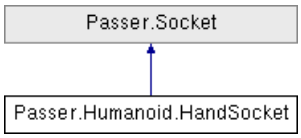
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/HandSkeleton.cs

Passer.Humanoid.HandSocket Class Reference

Description

A Socket attached to a hand

Inheritance diagram for Passer.Humanoid.HandSocket:



Public Types

- enum **AttachMethod** { **Unknown**, **Parenting**, **ReverseParenting**, **Joint**, **StaticJoint**, **ColliderDuplication** }

Public Member Functions

- override bool Attach (Handle handle, bool rangeCheck=true)
  Tries to attach the given Transform to this socket

- override void AttachStaticJoint (Transform objTransform)
- override void Release (bool releaseSticky=false)
  Releases a Transform from the socket

- override void ReleaseStaticJoint ()
- void **Attach** (GameObject objectToAttach)
- bool **Attach** (GameObject objectToAttach, bool rangeCheck)
- virtual bool Attach (Transform transformToAttach, bool rangeCheck=true)
  Tries to attach the given Transform to this socket

- void **OnDrawGizmosSelected** ()

Static Public Member Functions

- static Mesh **GenerateGizmoMesh1** ()
- static Mesh **GenerateGizmoMesh** ()

Public Attributes

- HandTarget handTarget
  The handTarget of the socket

- GameObject **attachedPrefab**
  A prefab which is used to attach to the socket at startup.

- Transform attachedTransform
  The Transform attached to this socket

- Handle **attachedHandle**
- string socketTag
  A tag for limiting which handles can be held by the socket

- AttachMethod **attachMethod** = AttachMethod.Unknown
- bool **destroyOnLoad** = false
- GameObjectEventHandlers attachEvent
  A GameObject Event for triggering changes in the Transform held by the Socket

Protected Member Functions

- override void MoveHandleToSocket (Transform socketTransform, Handle handle)
- override void MoveSocketToHandle (Transform socketTransform, Handle handle)
- override void MassRedistribution (Rigidbody socketRigidbody, Rigidbody objRigidbody)
- override bool AttachRigidbody (Rigidbody objRigidbody, Handle handle, bool rangeCheck=true)
- override void AttachRigidbodyParenting (Rigidbody objRigidbody, Handle handle)
- override void AttachRigidbodyJoint (Rigidbody objRigidbody, Handle handle)
- override void AttachSocketParenting (Rigidbody objRigidbody, Handle handle, Rigidbody socketRigidbody)
- override void ReleaseRigidbodyJoint ()
- override void ReleaseSocketParenting (Rigidbody objRigidbody, Transform socketTransform)
- override void ReleaseStaticObject ()
- override void MassRestoration (Rigidbody socketRigidbody, Rigidbody objRigidbody)
- virtual void **MoveHandleToSocket** (Transform socketTransform, Rigidbody handleRigidbody, Handle handle)
- virtual void **MoveRailToSocket** (Transform socketTransform, Transform railTransform, Handle rail)
- virtual void **MoveSocketToHandle** (Transform socketTransform, Rigidbody socketRigidbody, Handle handle)
- void **AttachTransformParenting** (Transform objTransform, Handle handle)
- virtual void **AttachRigidbodyReverseJoint** (Rigidbody objRigidbody, Handle handle)
  Attach handle to socket using a static joint on the handle

- virtual void **AttachSocketParenting** (Transform objTransform, Handle handle, Rigidbody thisRigidbody)
- virtual bool **AttachStaticObject** (Transform objTransform, Handle handle)
- virtual void **AttachStaticJointRotY** (Transform objTransform)
- virtual void **ReleaseRigidbodyParenting** ()
- void **ReleaseRigidbodyReverseJoint** ()
- void **ReleaseSocketParenting** (Transform objTransform)
- void **ReleaseTransformParenting** ()
- virtual void **Awake** ()
- virtual void **Update** ()
- void **UpdateHolding** ()
- virtual void **OnSceneUnload** (UnityEngine.SceneManagement.Scene _)

Static Protected Member Functions

- static void **DebugLog** (string s)

Protected Attributes

- Transform **releasingTransform**
- Transform attachedTransformParent
  The parent of the attached transform before it was attached

- RigidbodyDisabled **rigidbodyDisabled** = null
- float **originalMass** = 1
- bool **originalUseGravity** = false
- Mesh **gizmoMesh**

Static Protected Attributes

- static string[] attachEventTypeLabels

Properties

- bool **isOccupied** `[get]`
  Does the socket currently have a handle attached?

Member Function Documentation

override void Passer.Humanoid.HandSocket.MoveHandleToSocket (Transform *socketTransform*, Handle *handle*) `[protected], [virtual]`

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.MoveSocketToHandle (Transform *socketTransform*, Handle *handle*) `[protected], [virtual]`

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.MassRedistribution (Rigidbody *socketRigidbody*, Rigidbody *objRigidbody*) `[protected], [virtual]`

Reimplemented from Passer.Socket.

override bool Passer.Humanoid.HandSocket.Attach (Handle *handle*, bool *rangeCheck* = `true`) `[virtual]`

Tries to attach the given Transform to this socket

If the Handle has the right Socket Tag it will be attached to the socket. Static and Kinematic Rigidbodies will be attached by parenting. Non-Kinematic Rigidbodies will be attached using a joint.

Parameters

handle	The Handle to attach to this socket

Returns

Boolean indicating whether attachment succeeded

Reimplemented from Passer.Socket.

override bool Passer.Humanoid.HandSocket.AttachRigidbody (Rigidbody *objRigidbody*, Handle *handle*, bool *rangeCheck* = `true`) `[protected], [virtual]`

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.AttachRigidbodyParenting (Rigidbody *objRigidbody*, Handle *handle*) `[protected], [virtual]`

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.AttachRigidbodyJoint (Rigidbody *objRigidbody*, Handle *handle*) [protected], [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.AttachSocketParenting (Rigidbody *objRigidbody*, Handle *handle*, Rigidbody *socketRigidbody*) [protected], [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.AttachStaticJoint (Transform *objTransform*) [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.Release (bool *releaseSticky* = false) [virtual]

Releases a Transform from the socket

Note that if the Transform is not taken out of the range of the socket or held by another Socket, it will automatically snap back to the Socket.

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.ReleaseRigidbodyJoint () [protected], [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.ReleaseSocketParenting (Rigidbody *objRigidbody*, Transform *socketTransform*) [protected], [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.ReleaseStaticObject () [protected], [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.ReleaseStaticJoint () [virtual]

Reimplemented from Passer.Socket.

override void Passer.Humanoid.HandSocket.MassRestoration (Rigidbody *socketRigidbody*, Rigidbody *objRigidbody*) [protected], [virtual]

Reimplemented from Passer.Socket.

virtual bool Passer.Socket.Attach (Transform *transformToAttach*, bool *rangeCheck* = true) [virtual], [inherited]

Tries to attach the given Transform to this socket

If the Transform has a Handle with the right Socket Tag it will be attached to the socket. Static and Kinematic Rigidbodies will be attached by parenting. Non-Kinematic Rigidbodies will be attached using a joint.

Parameters

transformToAttach	The Transform to attach to this socket

Returns

Boolean indicating whether attachment succeeded

Member Data Documentation

HandTarget Passer.Humanoid.HandSocket.handTarget

The handTarget of the socket

This is the HandTarget of the hand to which the socket is attached

Transform Passer.Socket.attachedTransform [inherited]

The Transform attached to this socket

If the socket holds a Handle, this will contain the Transform of the Handle. It will be null otherwise

Transform Passer.Socket.attachedTransformParent [protected], [inherited]

The parent of the attached transform before it was attached

This is used to restore the parent when the transform is released again.

string Passer.Socket.socketTag [inherited]

A tag for limiting which handles can be held by the socket

If set (not null or empty) only Handles with the given tag will fit in the socket.

string [] Passer.Socket.attachEventTypeLabels [static], [protected], [inherited]

```
Initial value:= {
                "Never",
                "On Attach",
                "On Release",
                "While Attached",
                "While Released",
                "When Changed",
                "Always"
        }
```

GameObjectEventHandlers Passer.Socket.attachEvent [inherited]

```
Initial value:= new GameObjectEventHandlers() {
            label = "Hold Event",
            tooltip =
                "Call functions using what the socket is holding\n" +
```

```
                "Parameter: the GameObject held by the socket",
            eventTypeLabels = attachEventTypeLabels
        }
```

A GameObject Event for triggering changes in the Transform held by the Socket

---

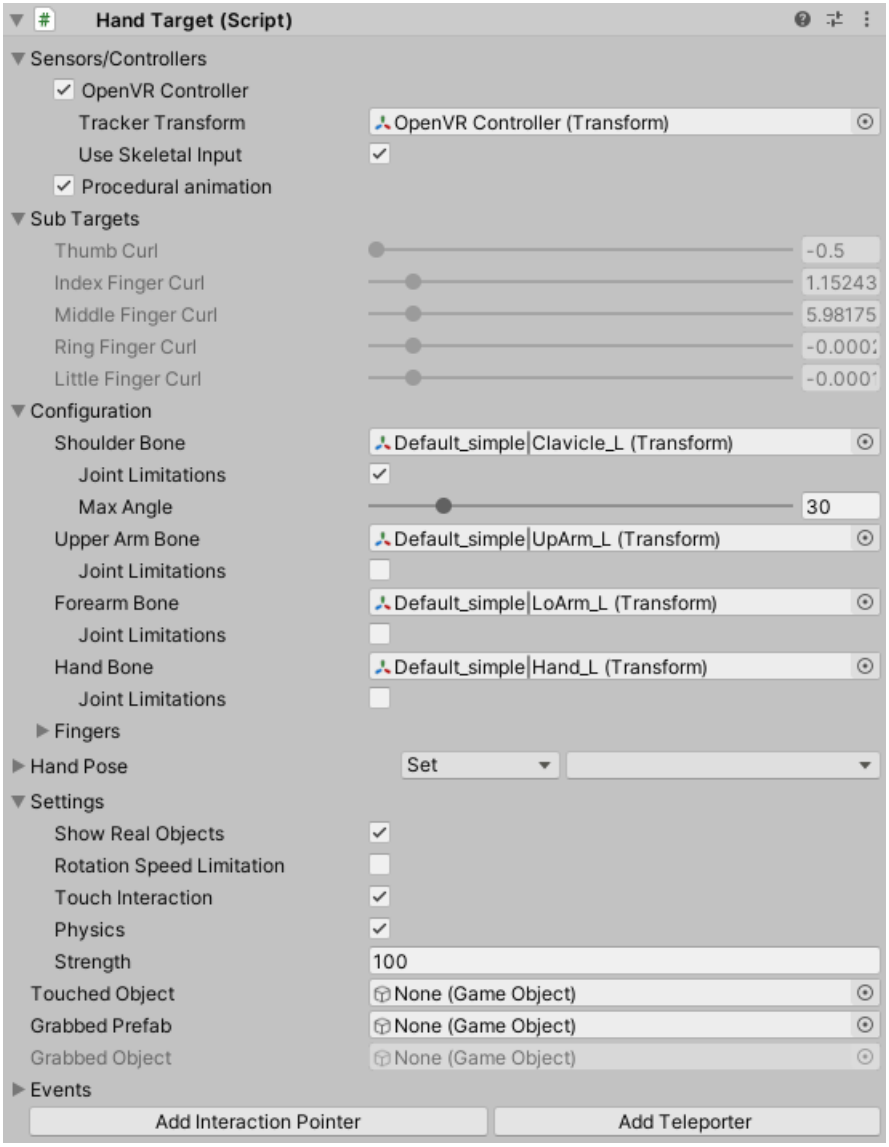
The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Physics/HandSocket.cs

Passer.Humanoid.HandTarget Class Reference

Description

Humanoid Control options for hand-related things



Sensors

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
- Rotation Speed Limitation
- Touch Interaction
- Physics
- Strength

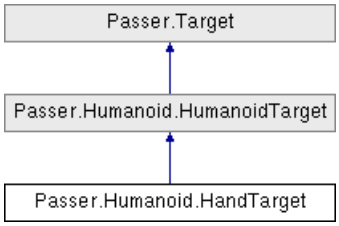
Events

- Pose Event
- Touch Event
- Grab Event

Buttons

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

Inheritance diagram for Passer.Humanoid.HandTarget:



Public Types

- enum **GrabType** { **HandGrab**, **Pinch** }
- enum **PoseMethod** { **Position**, **Rotation** }
- enum GrabbingTechnique { TouchGrabbing, NearGrabbing, NoGrabbing }
  Grabbing techniques

Public Member Functions

- virtual void **StartInteraction** ()
- HandSocket **CreateGrabSocket** ()
- Socket **CreatePinchSocket** ()
- void **OnNearing** (GameObject obj)
- virtual void **OnTouchStart** (GameObject obj, Vector3 contactPoint)
- virtual void **OnTouchEnd** (GameObject obj)
- void **GrabTouchedObject** ()
  Try to grab the object we touch

- void NearGrabCheck ()
  Check the hand for grabbing near objects

- void **GrabNearObject** ()
  Try to grab an object near to the hand

- void GrabCheck (GameObject obj)
  Try to take the object

- virtual bool **CanBeGrabbed** (GameObject obj)
- virtual void Grab (GameObject obj, bool rangeCheck=true)
  Grab and object with the hand (non-networked)

- void **GrabHandle** (Handle handle, bool rangeCheck=false)
- virtual void **LetGo** ()
- void **GrabOrLetGo** (GameObject obj, bool rangeCheck=true)
- override Transform GetDefaultTarget (HumanoidControl humanoid)
- Bone **GetBoneId** (bool isLeft, ArmBones armBone)
- TargetedBone **GetTargetBone** (ArmBones boneID)
- void **RetrieveBones** ()
- void **SetPose1** (Pose pose)
- void **SetPose** (Pose pose, float weight)
- bool **GrabbedStaticObject** ()
- Vector3 **HandBoneRightAxis** ()
- Vector3 **HandBoneOutwardAxis** ()
- void **InitTarget** ()
- override void InitAvatar ()
- override void NewComponent (HumanoidControl _humanoid)
- override void InitComponent ()

- override void StartTarget ()
- override void MatchTargetsToAvatar ()
- void **CopyBoneToTarget** (ArmBones armBone, TargetedBone bone)
- override void InitializeTrackingConfidence ()
  Sets the tracking confidence before all tracking information is updated

- override void UpdateTarget ()
- void **SetTargets** ()
- override void UpdateMovements (HumanoidControl humanoid)
- override void CopyTargetToRig ()
- override void CopyRigToTarget ()
- void **UpdateSensorsFromTarget** ()
- float **GetFingerCurl** (Finger fingerID)
- float **GetFingerCurl** (FingersTarget.TargetedFinger finger)
- void **AddFingerCurl** (Finger fingerID, float curlValue)
- void **SetFingerCurl** (Finger fingerID, float curlValue)
- void **SetFingerGroupCurl** (FingersTarget.FingerGroup fingerGroupID, float curlValue)
- void **DetermineFingerCurl** (Finger fingerID)
- float **HandCurl** ()
- void **Vibrate** (float strength)
- void **OnDrawGizmos** ()
- void **OnDrawGizmosSelected** ()
- virtual void **DrawTensions** ()
- abstract void **InitSensors** ()
- virtual void **StartSensors** ()
- virtual void **StopSensors** ()

Static Public Member Functions

- static HandTarget **CreateTarget** (HandTarget oldTarget)
- static HandTarget **SetTarget** (HumanoidControl humanoid, Transform targetTransform, bool isLeft)
- static void **ClearBones** (HandTarget handTarget)
- static void **TmpDisableCollisions** (HandTarget handTarget, float duration)
- static bool **IsInitialized** (HumanoidControl humanoid)
- static void DetermineTarget (HumanoidControl humanoid, bool isLeft)
  Checks whether the humanoid has an HandTarget and adds one if none has been found

- static Vector **ToVector** (Vector3 vector3)
- static Vector3 **ToVector3** (Vector position)
- static Rotation **ToRotation** (Quaternion quaternion)
- static Quaternion **ToQuaternion** (Rotation orientation)
- static void **SetRotation** (Transform transform, Rotation orientation)
- static void **DrawTarget** (Confidence confidence, Transform target, Vector3 direction, float length)
- static void **DrawTargetBone** (TargetedBone bone, Vector3 direction)
- static void **DrawTargetBone** (TargetTransform target, Vector3 direction)
- static void **DrawAvatarBone** (TargetedBone bone, Vector3 direction)
- static void **DrawAvatarBone** (BoneTransform bone, Vector3 direction)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, Bone boneId, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, HumanBodyBones boneID, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, params string[] boneNames)
- static void **GetDefaultTargetBone** (Animator rig, ref Transform boneTransform, Bone boneID, params string[] boneNames)

- static List< Collider > **SetColliderToTrigger** (GameObject obj)
- static List< Collider > **SetColliderToTrigger** (Rigidbody rb)
- static void **UnsetColliderToTrigger** (List< Collider > colliders)
- static void **UnsetColliderToTrigger** (List< Collider > colliders, Collider collider)

Public Attributes

- bool **isLeft**
  Is this the left hand?

- Side **side**
  Left or right hand side hand

- Vector3 **outward**
  Vector pointing toward the outward direction of the hand. In general this is the index finger pointing direction

- Vector3 **up**
  Vector3 pointing toward the up direction of the hand. This is the opposite side of the palm of the hand

- FingersTarget **fingers** = null
- bool **rotationSpeedLimitation** = false
  Limits the maximum rotation speed of the joints. This can result in more natural movements with optical tracking solutions.

- UnityXRController **unity**
- ArmAnimator **armAnimator** = new ArmAnimator()
- UnityXRHand **unityXR** = new UnityXRHand()
- ViveTrackerArm **viveTracker** = new ViveTrackerArm()
- WindowsMRHand **mixedReality** = new WindowsMRHand()
- WaveVRHand **waveVR** = new WaveVRHand()
- LeapHand **leap** = new LeapHand()
- Kinect1Arm **kinect1** = new Kinect1Arm()
- Kinect2Arm **kinect2** = new Kinect2Arm()
- Kinect4Arm **kinect4** = new Kinect4Arm()
- AstraArm **astra** = new AstraArm()
- RazerHydraHand **hydra** = new RazerHydraHand()
- OptitrackArm **optitrack** = new OptitrackArm()
- AntilatencyHand **antilatency** = new AntilatencyHand()
- Hi5Hand **hi5** = new Hi5Hand()
- CustomArm **custom** = new CustomArm()
- Transform **stretchlessTarget**
- TargetedShoulderBone **shoulder**
- TargetedUpperArmBone **upperArm**
- TargetedForearmBone **forearm**
- TargetedHandBone **hand**
- PoseMethod **poseMethod**
- bool physics = true
  Enables physics for this hand

- AdvancedHandPhysics.PhysicsMode **physicsMode** = AdvancedHandPhysics.PhysicsMode.HybridKinematic
- float **strength** = 100

The strength of the arm when interacting with physics

- GrabbingTechnique **grabbingTechnique** = GrabbingTechnique.TouchGrabbing
  The GrabbingType used to grab objects

- bool touchInteraction = true
  Enables touch interaction with the envrionment

- Pose **detectedPose**
- PoseMixer **poseMixer** = new PoseMixer()
- HandSocket **grabSocket**
  The Socket for holding objects with the whole hand

- Socket **pinchSocket**
  The Socket for holding objects between the thumb and index finger

- InteractionModule **inputModule**
- GameObject **touchedObject** = null
- GameObject **grabbedPrefab**
- GameObject **grabbedObject**
- Handle **grabbedHandle** = null
- Vector3 **targetToHandle**
- bool **grabbedRigidbody**
- bool **grabbedKinematicRigidbody**
- List< Collider > **colliders**
- bool **twoHandedGrab** = false
- Vector3 **targetToSecondaryHandle**
- IntEventHandlers poseEvent
  Use to call functions based on the defined poses of the hand.

- GameObjectEventHandlers touchEvent
  Use to call functions based on the hand touching objects

- GameObjectEventHandlers grabEvent
  Use to call functions based on the hand grabbing objects

- Transform **handPalm**
- Rigidbody **handRigidbody**
- AdvancedHandPhysics **handPhysics**
- HandMovements **handMovements** = new HandMovements()
- ArmMovements **armMovements** = new ArmMovements()
- bool **grabbedChanged**
- bool **directFingerMovements** = true
- HumanoidControl **humanoid**

Static Public Attributes

- static float **maxGrabbingMass** = 10
  The maximum mass of object you can grab

- const float **maxShoulderAngle** = 30
- const float **maxUpperArmAngle** = 120

- const float **maxForearmAngle** = 130
- const float **maxHandAngle** = 100
- static readonly Vector3 **minLeftShoulderAngles** = new Vector3(0, 0, -45)
- static readonly Vector3 **maxLeftShoulderAngles** = new Vector3(0, 45, 0)
- static readonly Vector3 **minRightShoulderAngles** = new Vector3(0, -45, 0)
- static readonly Vector3 **maxRightShoulderAngles** = new Vector3(0, 0, 45)
- static readonly Vector3 **minLeftUpperArmAngles** = new Vector3(-180, -45, -180)
- static readonly Vector3 **maxLeftUpperArmAngles** = new Vector3(60, 130, 45)
- static readonly Vector3 **minRightUpperArmAngles** = new Vector3(-180, -130, -45)
- static readonly Vector3 **maxRightUpperArmAngles** = new Vector3(60, 45, 180)
- static readonly Vector3 **minLeftForearmAngles** = new Vector3(0, 0, 0)
- static readonly Vector3 **maxLeftForearmAngles** = new Vector3(0, 150, 0)
- static readonly Vector3 **minRightForearmAngles** = new Vector3(0, -150, 0)
- static readonly Vector3 **maxRightForearmAngles** = new Vector3(0, 0, 0)
- static readonly Vector3 **minLeftHandAngles** = new Vector3(-180, -50, -70)
- static readonly Vector3 **maxLeftHandAngles** = new Vector3(90, 20, 90)
- static readonly Vector3 **minRightHandAngles** = new Vector3(-90, -20, -70)
- static readonly Vector3 **maxRightHandAngles** = new Vector3(45, 50, 70)

Protected Member Functions

- void **MoveToPalm** (Transform t)
- void **LerpToGrab** (GameObject obj)
- virtual GameObject **DetermineGrabObject** ()
- bool **GrabbedWithOtherHand** (GameObject obj)
- bool **SecondHandGrab** (GameObject obj, bool rangeCheck)
- virtual bool **GrabHandle** (Rigidbody objRigidbody, Handle handle, bool rangeCheck)
- virtual bool **GrabRigidbody** (Rigidbody objRigidbody, bool rangeCheck=true)
- virtual bool **GrabRigidbodyHandle** (Rigidbody objRigidbody, Handle handle, bool rangeCheck)
- virtual bool **GrabRigidbodyWithoutHandle** (Rigidbody objRigidbody)
- virtual bool **GrabRigidbodyParenting** (Rigidbody objRigidbody)
- virtual bool **GrabStaticWithoutHandle** (GameObject obj)
- virtual void **CheckLetGo** ()
- virtual bool **PulledLoose** ()
  summary>Let go the object the hand is holding (if any)

- void **LetGoRigidbodyWithoutHandle** ()
- void **LetGoStaticWithoutHandle** ()
- void **LetGoRigidbody** (Rigidbody grabbedRigidbody)
- void **LetGoHandle** (Handle handle)
- void **LetGoPinch** ()
- virtual void **UpdateEvents** ()
- void **CheckRigidbody** ()
- void **UpdateUsingTrackedRigidbody** (TrackedRigidbody trackedRigidbody)
- void **UpdateTwoHanded2** ()
- Vector3 **Orthagonal** (Transform primaryTransform, Transform secondaryTransform)
- override void DrawTargetRig (HumanoidControl humanoid)
- override void DrawAvatarRig (HumanoidControl humanoid)
- virtual void **GenerateColliders** ()
- virtual void **DrawTensionGizmo** (TargetedBone targetedBone)
- virtual void **UpdateSensors** ()

Static Protected Member Functions

- static void **DebugLog** (string s)

Protected Attributes

- bool **grabChecking** = false
- bool **letGoChecking** = false
- float **letGoCheckStart**
- ArmSensor[] **sensors**
- List< SensorComponent > **sensorComponents** = new List<SensorComponent>()
- List< TrackedRigidbody > **trackedRigidbodies** = new List<TrackedRigidbody>()
- bool **_showRealObjects** = true

Properties

- override Passer.Sensor **animator** `[get]`
- bool **isTrackingHand** `[getprotected set]`
  Is this hand currently using hand tracking?

- override TargetedBone **main** `[get]`
- int **detectedPoseIx** `[getprotected set]`
- HandTarget? **otherHand** `[get]`
- Vector3 **localPalmPosition** `[get]`
- Vector3 **palmPosition** `[get]`
- Quaternion **palmRotation** `[get]`
- virtual bool **showRealObjects** `[getset]`
  show the target meshes

Member Enumeration Documentation

enum Passer.Humanoid.HandTarget.GrabbingTechnique

Grabbing techniques

Enumerator:

TouchGrabbing	try to grab the object when you touch it and you close the hand
NearGrabbing	try to grab an object near to your hand when you close the hand
NoGrabbing	do not try to grab objects when you close the hand

---

Member Function Documentation

void Passer.Humanoid.HandTarget.NearGrabCheck ()

Check the hand for grabbing near objects

This function will grab a near object if the hand is grabbing

void Passer.Humanoid.HandTarget.GrabCheck (GameObject *obj*)

Try to take the object

Parameters

obj	The object to take

Depending on the hand pose the object may be grabbed, pinched or not

virtual void Passer.Humanoid.HandTarget.Grab (GameObject *obj*, bool *rangeCheck* = `true`) `[virtual]`

Grab and object with the hand (non-networked)

Parameters

handTarget	The hand to grab with
obj	The gameObject to grab
rangeCheck	check wither the hand is in range of the handle

override Transform Passer.Humanoid.HandTarget.GetDefaultTarget (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.InitAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.NewComponent (HumanoidControl *_humanoid*) `[virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.InitComponent () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.HandTarget.StartTarget () `[virtual]`

Implements Passer.Target.

static void Passer.Humanoid.HandTarget.DetermineTarget (HumanoidControl *humanoid*, bool *isLeft*) `[static]`

Checks whether the humanoid has an HandTarget and adds one if none has been found

Parameters

humanoid	The humanoid to check
isLeft	Is this the left hand?

override void Passer.Humanoid.HandTarget.MatchTargetsToAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.InitializeTrackingConfidence () `[virtual]`

Sets the tracking confidence before all tracking information is updated

Implements Passer.Target.

override void Passer.Humanoid.HandTarget.UpdateTarget () `[virtual]`

Implements Passer.Target.

override void Passer.Humanoid.HandTarget.UpdateMovements (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.CopyTargetToRig () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.CopyRigToTarget () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.DrawTargetRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HandTarget.DrawAvatarRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

Member Data Documentation

bool Passer.Humanoid.HandTarget.physics = true

Enables physics for this hand

See Physics.

bool Passer.Humanoid.HandTarget.touchInteraction = true

Enables touch interaction with the envrionment

See Interaction

IntEventHandlers Passer.Humanoid.HandTarget.poseEvent

```
Initial value:= new IntEventHandlers() {
            id = 1,
            label = "Pose Event",
            tooltip =
                "Call functions based on recognized poses" +
                "Parameter: the index of the recognized pose",
            eventTypeLabels = new string[] {
                "Never",
                "On Pose Recognized",
                "On No Pose Recongnized",
                "While Pose Recognized",
                "While No Pose Recognized",
                "When Pose Changes",
                "Always",
            },
            fromEventLabel = "poseMixer.detectedPoseIx",
        }
```

Use to call functions based on the defined poses of the hand.

GameObjectEventHandlers Passer.Humanoid.HandTarget.touchEvent

```
Initial value:= new GameObjectEventHandlers() {
            id = 2,
            label = "Touch Event",
            tooltip =
                "Call funtions based on touched objects" +
                "Parameter: the touched object",
            eventTypeLabels = new string[] {
                "Never",
                "On Touch Start",
                "On Touch End",
                "While Touching",
                "While not Touching",
                "On Touched Object Changes",
                "Always",
            },
            fromEventLabel = "touchedObject",
        }
```

Use to call functions based on the hand touching objects

GameObjectEventHandlers Passer.Humanoid.HandTarget.grabEvent

```
Initial value:= new GameObjectEventHandlers() {
            id = 3,
            label = "Grab Event",
            tooltip =
                "Call functions based on grabbed objects" +
                "Parameter: the grabbed object",
            eventTypeLabels = new string[] {
                "Never",
                "On Grab Start",
                "On Grab End",
                "While Holding Object",
                "While not Holding Object",
                "On Grabbed Object Changes",
                "Always",
            },
            fromEventLabel = "grabbedObject",
        }
```

Use to call functions based on the hand grabbing objects

The documentation for this class was generated from the following files:

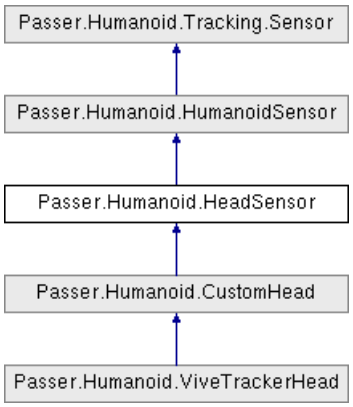
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Interaction/HandInteraction.cs
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Targets/HandTarget.cs

Passer.Humanoid.HeadSensor Class Reference

Description

A sensor used on the head of a humanoid

Inheritance diagram for Passer.Humanoid.HeadSensor:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- virtual void **CheckSensor** (HeadTarget headTarget)
- virtual void **Init** (HeadTarget headTarget)
- override void Start (HumanoidControl _humanoid, Transform targetTransform)
- void **InitController** (SerializedProperty sensorProp, HeadTarget target)
- void **RemoveController** (SerializedProperty sensorProp)
- override void Update ()
  Update the sensor state

- virtual void **CheckSensorTransform** ()
- virtual void **SetSensor2Target** ()
- virtual void **UpdateSensorTransformFromTarget** (Transform targetTransform)
- virtual void **Stop** ()
- virtual void **RefreshSensor** ()
- virtual void **ShowSensor** (HumanoidTarget target, bool shown)

Static Public Member Functions

- static Vector3 **InverseTransformPointUnscaled** (Transform transform, Vector3 position)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- Vector3 **sensor2TargetPosition**
- Quaternion **sensor2TargetRotation**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable

Status of the sensor

Static Public Attributes

- const string **_name** = ""

Protected Member Functions

- virtual void **CreateSensorTransform** (string resourceName, Vector3 sensor2TargetPosition, Quaternion sensor2TargetRotation)
- virtual void **UpdateNeckTargetFromHead** ()
- virtual void **CreateSensorTransform** ()
- void **CreateSensorTransform** (Transform targetTransform, string resourceName, Vector3 _sensor2TargetPosition, Quaternion _sensor2TargetRotation)
- void **RemoveSensorTransform** ()
- void **UpdateSensorTransform** (Tracking.Sensor sensor)
- virtual void **UpdateTargetTransform** ()
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, Transform sensorTransform)
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, SensorComponent sensorComponent)
- Vector3 **GetTargetPosition** (Transform sensorTransform)
- Quaternion **GetTargetRotation** (Transform sensorTransform)
- void **UpdateSensor** ()

Static Protected Member Functions

- static Vector3 **TransformPointUnscaled** (Transform transform, Vector3 position)

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- HeadTarget **headTarget** `[get]`
- HumanoidControl **humanoid** `[get]`
- virtual new HumanoidTracker **tracker** `[get]`
- override string **name** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`

Member Function Documentation

override void Passer.Humanoid.HeadSensor.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) [virtual]

Reimplemented from Passer.Humanoid.HumanoidSensor.

Reimplemented in Passer.Humanoid.CustomHead.

override void Passer.Humanoid.HeadSensor.Update () [virtual]

Update the sensor state

Returns

Status of the sensor after the update

Reimplemented from Passer.Humanoid.Tracking.Sensor.

Reimplemented in Passer.Humanoid.CustomHead.

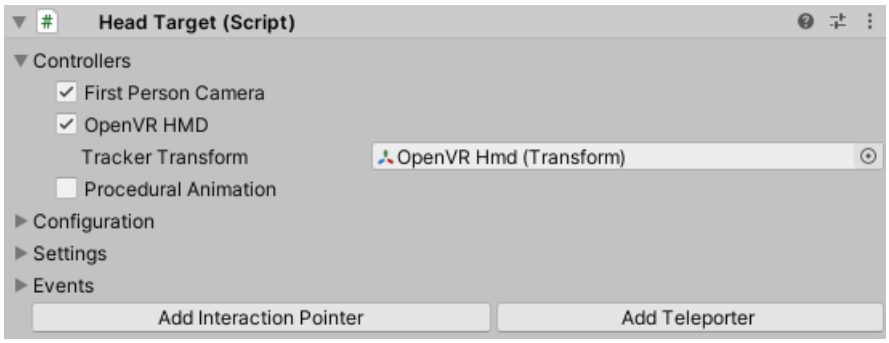
The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/HeadSensor.cs

Passer.Humanoid.HeadTarget Class Reference

Description

Humanoid Control options for head-related things



Sensors

Depending on the selected Inputs in HumanoidControl, 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.

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 (Pro)

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

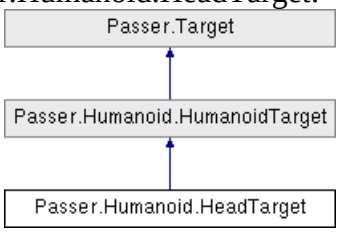
Events

- Tracking Event
- Audio Event
- Focus Event (Pro)
- Blink Event (Pro)
- In Collider Event

Buttons

- Add Interaction Pointer: Adds a gaze interaction pointer to the head target. 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.
-

Inheritance diagram for Passer.Humanoid.HeadTarget:



Public Types

- enum **InteractionType** { **None**, **Gazing** }

Public Member Functions

- override void InitSensors ()
- override void StartSensors ()
- void **TurnTo** (GameObject obj)
- void **TurnTo** (GameObject obj, float confidence)
- void **LookTo** (Vector3 position)
- void **TurnTo** (Vector3 position, float confidence)
- void **SetLookDirection** (Vector3 direction, float confidence)
- override Transform GetDefaultTarget (HumanoidControl humanoid)
- void **RetrieveBones** ()
- override void InitAvatar ()
- override void NewComponent (HumanoidControl _humanoid)
- override void InitComponent ()
- override void StartTarget ()
- override void MatchTargetsToAvatar ()
- override void InitializeTrackingConfidence ()
  Sets the tracking confidence before all tracking information is updated

- override void UpdateTarget ()
  Update all head sensors

- override void UpdateMovements (HumanoidControl humanoid)

Updates the avatar bones based on the current target rig

- override void CopyTargetToRig ()
  Copy the head target to the target rig

- override void CopyRigToTarget ()
  Copy the target rig head bone to the head target

- void **UpdateSensorsFromTarget** ()
  Update the sensor locations based on the head target

- void **RotationX** (float angle)
  Sets the rotation of the head around the X axis

- void **RotationY** (float angle)
  Sets the rotation of the head around the Y axis

- Vector3 **GetEyePosition** ()
  Gets the eye position in world coordinates

- Vector3 **GetNeckEyeDelta** ()
  Gets the local eye position relative to the neck bone

- Vector3 **GetHeadEyeDelta** ()
  Gets the local eye position realtive to the head bone

- Vector3 **GetNeckHeadDelta** ()
  Gets the local head position relative to the neck bone

- void **DisableVR** ()
- void **EnableVR** ()
- void **OnDrawGizmos** ()
- void **OnDrawGizmosSelected** ()
- virtual void **DrawTensions** ()
- virtual void **StopSensors** ()

Static Public Member Functions

- static HeadTarget **CreateTarget** (HumanoidTarget oldTarget)
- static HeadTarget **SetTarget** (HumanoidControl humanoid, Transform targetTransform)
- static void **GetDefaultNeck** (Animator rig, ref Transform boneTransform)
- static void **GetDefaultHead** (Animator rig, ref Transform boneTransform)
- static void **ClearBones** (HeadTarget headTarget)
- static bool **IsInitialized** (HumanoidControl humanoid)
  Is the head target initialized?

- static void DetermineTarget (HumanoidControl humanoid)
  Checks whether the humanoid has an HeadTargetand adds one if none has been found

- static Quaternion **GetRotationAround** (Vector3 axis, Quaternion rotation)

- static Quaternion **Normalize** (Quaternion q)
- static void **GetSwingTwist** (Vector3 axis, Quaternion rotation, out Quaternion swing, out Quaternion twist)
- static SkinnedMeshRenderer[] **FindAvatarMeshes** (HumanoidControl humanoid)
- static string[] **DistillAvatarMeshNames** (SkinnedMeshRenderer[] meshes)
- static int **FindMeshWithBlendshapes** (SkinnedMeshRenderer[] renderers)
- static int **FindBlendshapemesh** (SkinnedMeshRenderer[] renderers, SkinnedMeshRenderer renderer)
- static string[] **GetBlendshapes** (SkinnedMeshRenderer renderer)
- static void **FindBlendshapeWith** (string[] blendshapes, string namepart1, string namepart2, ref int blendshape)
- static Vector **ToVector** (Vector3 vector3)
- static Vector3 **ToVector3** (Vector position)
- static Rotation **ToRotation** (Quaternion quaternion)
- static Quaternion **ToQuaternion** (Rotation orientation)
- static void **SetRotation** (Transform transform, Rotation orientation)
- static void **DrawTarget** (Confidence confidence, Transform target, Vector3 direction, float length)
- static void **DrawTargetBone** (TargetedBone bone, Vector3 direction)
- static void **DrawTargetBone** (TargetTransform target, Vector3 direction)
- static void **DrawAvatarBone** (TargetedBone bone, Vector3 direction)
- static void **DrawAvatarBone** (BoneTransform bone, Vector3 direction)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, Bone boneId, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, HumanBodyBones boneID, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, params string[] boneNames)
- static void **GetDefaultTargetBone** (Animator rig, ref Transform boneTransform, Bone boneID, params string[] boneNames)
- static List< Collider > **SetColliderToTrigger** (GameObject obj)
- static List< Collider > **SetColliderToTrigger** (Rigidbody rb)
- static void **UnsetColliderToTrigger** (List< Collider > colliders)
- static void **UnsetColliderToTrigger** (List< Collider > colliders, Collider collider)

Public Attributes

- bool **tracking**
  Is the Head Target updated using an active tracking device

- Passer.Tracking.UnityXRHmd **unity**
- HeadAnimator **headAnimator** = new HeadAnimator()
  Controls the head when no tracking is active

- UnityXRHead **unityXR** = new UnityXRHead()
- ViveTrackerHead **viveTracker** = new ViveTrackerHead()
- WindowsMRHead **mixedReality** = new WindowsMRHead()
- WaveVRHead **waveVR** = new WaveVRHead()
- Kinect1Head **kinect1** = new Kinect1Head()
- Kinect2Head **kinect2** = new Kinect2Head()
- Kinect4Head **kinect4** = new Kinect4Head()
- AstraHead **astra** = new AstraHead()
- IntelRealsenseHead **realsense** = new IntelRealsenseHead()
- OptitrackHead **optitrack** = new OptitrackHead()
- AntilatencyHead **antilatency** = new AntilatencyHead()
- CustomHead **custom** = new CustomHead()
- MicrophoneHead **microphone** = new MicrophoneHead()

- Kinect2Face **kinectFace**
- TobiiHead **tobiiHead** = new TobiiHead()
- ArKitHead **arkit** = new ArKitHead()
- Tracking.Pupil.Head **pupil** = new Tracking.Pupil.Head()
- DlibHead **dlib** = new DlibHead()
- HeadSensor[] **sensors**
- TargetedHeadBone **head** = null
- TargetedNeckBone **neck** = null
- FaceTarget **face** = null
- float **audioEnergy**
- Vector3 **lookDirection** = Vector3.forward
- Vector3 **localLookDirection** = Vector3.forward
- Vector3 **neck2eyes**
- Vector3 **head2eyes**
- bool **collisionFader** = false
  Adds a screen fader which blacks out the camera when the head enters objects.

- bool **isInsideCollider** = false
- bool **virtual3d** = false
- Transform **screenTransform**
- BoolEventHandlers trackingEvent
  Use to call functions based on the tracking status of the headset.

- FloatEventHandlers audioEvent
  Use to call functions based on the audio level measured with the microphone.

- GameObjectEventHandlers focusEvent
  Use to call functions based on the object in focus

- BoolEventHandlers blinkEvent
  Use to call functions based on eye blinking

- BoolEventHandlers insideColliderEvent
  Use to call functions based on the state of the head being inside colliders.

- SkinnedMeshRenderer **smRenderer**
- Rigidbody **headRigidbody**
- HeadMovements **headMovements** = new HeadMovements()
- HumanoidControl **humanoid**

Static Public Attributes

- const float **maxNeckAngle** = 80
- const float **maxHeadAngle** = 50
- static readonly float **neckTurnRatio** = 0.65F
- static readonly Vector3 **minHeadAngles** = new Vector3(0, 0, 0)
- static readonly Vector3 **maxHeadAngles** = new Vector3(0, 0, 0)
- static readonly Vector3 **minNeckAngles** = new Vector3(-55, -70, -35)
- static readonly Vector3 **maxNeckAngles** = new Vector3(80, 70, 35)
- static readonly Vector **minNeckAngles2** = new Vector(-55, -70, 0)
- static readonly Vector **maxNeckAngles2** = new Vector(80, 70, 0)

Protected Member Functions

- override void UpdateSensors ()
- virtual void **UpdateEvents** ()
- override void DrawTargetRig (HumanoidControl humanoid)
  Draw the target rig

- override void DrawAvatarRig (HumanoidControl humanoid)
  Draw the avatar rig

- virtual void **DrawTensionGizmo** (TargetedBone targetedBone)

Protected Attributes

- bool **_showRealObjects** = true

Properties

- override Passer.Sensor **animator** `[get]`
- override TargetedBone **main** `[get]`
- virtual bool **showRealObjects** `[getset]`
  show the target meshes

Member Function Documentation

override void Passer.Humanoid.HeadTarget.InitSensors () `[virtual]`

Implements Passer.Target.

override void Passer.Humanoid.HeadTarget.StartSensors () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.HeadTarget.UpdateSensors () `[protected]`, `[virtual]`

Reimplemented from Passer.Target.

override Transform Passer.Humanoid.HeadTarget.GetDefaultTarget (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.InitAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.NewComponent (HumanoidControl *_humanoid*) `[virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.InitComponent () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.HeadTarget.StartTarget () `[virtual]`

Implements Passer.Target.

static void Passer.Humanoid.HeadTarget.DetermineTarget (HumanoidControl *humanoid*) `[static]`

Checks whether the humanoid has an HeadTargetand adds one if none has been found

Parameters

humanoid	The humanoid to check

override void Passer.Humanoid.HeadTarget.MatchTargetsToAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.InitializeTrackingConfidence () `[virtual]`

Sets the tracking confidence before all tracking information is updated

Implements Passer.Target.

override void Passer.Humanoid.HeadTarget.UpdateTarget () `[virtual]`

Update all head sensors

Implements Passer.Target.

override void Passer.Humanoid.HeadTarget.UpdateMovements (HumanoidControl *humanoid*) `[virtual]`

Updates the avatar bones based on the current target rig

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.CopyTargetToRig () `[virtual]`

Copy the head target to the target rig

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.CopyRigToTarget () `[virtual]`

Copy the target rig head bone to the head target

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.DrawTargetRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Draw the target rig

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HeadTarget.DrawAvatarRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Draw the avatar rig

Reimplemented from Passer.Humanoid.HumanoidTarget.

Member Data Documentation

BoolEventHandlers Passer.Humanoid.HeadTarget.trackingEvent

```
Initial value:= new BoolEventHandlers() {
            id = 1,
            label = "Tracking Event",
            tooltip =
                "Call functions using the HMD tracking status\n" +
                "Parameter: HMD tracking",
            eventTypeLabels = new string[] {
                "Never",
                "On Tracking Start",
                "On Tracking Stop",
                "While Tracking",
                "While not Tracking",
                "On Tracking Changes",
                "Always",
            },
            fromEventLabel = "tracking",
        }
```

Use to call functions based on the tracking status of the headset.

FloatEventHandlers Passer.Humanoid.HeadTarget.audioEvent

```
Initial value:= new FloatEventHandlers() {
            id = 2,
            label = "Audio Event",
            tooltip =
                "Call functions based on the microphone audio level\n" +
                "Parameter: the audio level",
            eventTypeLabels = new string[] {
                "Never",
                "On Loud Start",
                "On Silence Start",
                "While Noisy",
                "While Silent",
                "On Level Changes",
                "Always",
            },
            fromEventLabel = "audioEnergy",
        }
```

Use to call functions based on the audio level measured with the microphone.

GameObjectEventHandlers Passer.Humanoid.HeadTarget.focusEvent

```
Initial value:= new GameObjectEventHandlers() {
            id = 3,
            label = "Focus Event",
            tooltip =
                "Call functions using the focus\n" +
                "Parameter: the focus object",
            eventTypeLabels = new string[] {
                "Never",
                "On Focus Start",
                "On Focus End",
                "While Focusing",
                "While Nothing in Focus",
                "On Focus Changes",
                "Always",
            },
            fromEventLabel = "Focus Object",
        }
```

Use to call functions based on the object in focus

BoolEventHandlers Passer.Humanoid.HeadTarget.blinkEvent

```
Initial value:= new BoolEventHandlers() {
            id = 4,
            label = "Blink Event",
            tooltip =
                "Call functions using blinking\n" +
                "Parameter: the blinking state",
            eventTypeLabels = new string[] {
                "Never",
                "On Blink Starts",
                "On Blink Ends",
                "While Eyes Closed",
                "While Eyes Open",
                "On Blink Starts or Ends",
                "Always",
            },
            fromEventLabel = "Eyes Closed",
        }
```

Use to call functions based on eye blinking

BoolEventHandlers Passer.Humanoid.HeadTarget.insideColliderEvent

```
Initial value:= new BoolEventHandlers() {
            id = 5,
            label = "In Collider Event",
            tooltip =
                "Call functions using the head being inside Colliders\n" +
                "Parameter: isInsideCollider state",
            eventTypeLabels = new string[] {
                "Never",
                "When Head Enters Collider",
                "When Head Exits Collider",
                "While Head is inside Collider",
                "While Head outside Collider",
                "When Enters/Exists Collider",
                "Always",
            },
            fromEventLabel = "Inside Collider",
        }
```

Use to call functions based on the state of the head being inside colliders.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Targets/HeadTarget.cs

Passer.Humanoid.HipsTarget Class Reference

Description

Humanoid Control options for torso related things

Sensors

See the list of supported devices to get information on the hips 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.

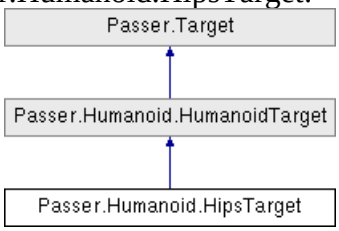
Limits

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

Inheritance diagram for Passer.Humanoid.HipsTarget:



Public Member Functions

- override void InitSensors ()
- override void StartSensors ()
- TargetedBone **GetTargetBone** (TorsoBones boneID)
- override Transform GetDefaultTarget (HumanoidControl humanoid)
- void **RetrieveBones** ()
- override void InitAvatar ()
- override void InitComponent ()
- override void StartTarget ()
- override void MatchTargetsToAvatar ()
- override void InitializeTrackingConfidence ()
  Sets the tracking confidence before all tracking information is updated

- override void UpdateTarget ()
- void **Cerebellum_UpdateTargets** ()

- override void UpdateMovements (HumanoidControl humanoid)
- override void CopyTargetToRig ()
- override void CopyRigToTarget ()
- void **UpdateSensorsFromTarget** ()
- virtual void **NewComponent** (HumanoidControl _humanoid)
- void **OnDrawGizmos** ()
- void **OnDrawGizmosSelected** ()
- virtual void **DrawTensions** ()
- virtual void **StopSensors** ()

Static Public Member Functions

- static HipsTarget **CreateTarget** (HumanoidTarget oldTarget)
- static HipsTarget **SetTarget** (HumanoidControl humanoid, Transform targetTransform, bool isLeft)
- static bool **IsInitialized** (HumanoidControl humanoid)
- static void DetermineTarget (HumanoidControl humanoid)
  Checks whether the humanoid has an HipsTarget and adds one if none has been found

- static Vector **ToVector** (Vector3 vector3)
- static Vector3 **ToVector3** (Vector position)
- static Rotation **ToRotation** (Quaternion quaternion)
- static Quaternion **ToQuaternion** (Rotation orientation)
- static void **SetRotation** (Transform transform, Rotation orientation)
- static void **DrawTarget** (Confidence confidence, Transform target, Vector3 direction, float length)
- static void **DrawTargetBone** (TargetedBone bone, Vector3 direction)
- static void **DrawTargetBone** (TargetTransform target, Vector3 direction)
- static void **DrawAvatarBone** (TargetedBone bone, Vector3 direction)
- static void **DrawAvatarBone** (BoneTransform bone, Vector3 direction)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, Bone boneId, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, HumanBodyBones boneID, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, params string[] boneNames)
- static void **GetDefaultTargetBone** (Animator rig, ref Transform boneTransform, Bone boneID, params string[] boneNames)
- static List< Collider > **SetColliderToTrigger** (GameObject obj)
- static List< Collider > **SetColliderToTrigger** (Rigidbody rb)
- static void **UnsetColliderToTrigger** (List< Collider > colliders)
- static void **UnsetColliderToTrigger** (List< Collider > colliders, Collider collider)

Public Attributes

- float **hipsBaseHeight**
- bool **newSpineIK** = false
- TorsoMovements **torsoMovements** = new TorsoMovements()
- TorsoAnimator **torsoAnimator** = new TorsoAnimator()
  Controls the hips when no tracking is active

- ViveTrackerTorso **viveTracker** = new ViveTrackerTorso()
- Kinect1Torso **kinect1** = new Kinect1Torso()
- Kinect2Torso **kinect2** = new Kinect2Torso()
- Kinect4Torso **kinect4** = new Kinect4Torso()
- AstraTorso **astra** = new AstraTorso()
- OptitrackTorso **optitrack** = new OptitrackTorso()
- CustomTorso **custom** = new CustomTorso()

- TargetedChestBone **chest** = null
- TargetedSpineBone **spine** = null
- TargetedHipsBone **hips** = null
- float **bendingFactor** = 1
- float **torsoLength**
- Quaternion **spine2HipsRotation**
- HumanoidControl **humanoid**

Static Public Attributes

- const float **maxSpineAngle** = 20
- const float **maxChestAngle** = 20

Protected Member Functions

- override void UpdateSensors ()
- Quaternion **DetermineSpine2HipsRotation** ()
- override void DrawTargetRig (HumanoidControl humanoid)
- override void DrawAvatarRig (HumanoidControl humanoid)
- virtual void **DrawTensionGizmo** (TargetedBone targetedBone)

Protected Attributes

- bool **_showRealObjects** = true

Properties

- override Passer.Sensor **animator** `[get]`
- override TargetedBone **main** `[get]`
- virtual bool **showRealObjects** `[getset]`
  show the target meshes

Member Function Documentation

override void Passer.Humanoid.HipsTarget.InitSensors () `[virtual]`

Implements Passer.Target.

override void Passer.Humanoid.HipsTarget.StartSensors () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.HipsTarget.UpdateSensors () `[protected]`, `[virtual]`

Reimplemented from Passer.Target.

override Transform Passer.Humanoid.HipsTarget.GetDefaultTarget (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.InitAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.InitComponent () `[virtual]`

Reimplemented from Passer.Target.

override void Passer.Humanoid.HipsTarget.StartTarget () `[virtual]`

Implements Passer.Target.

static void Passer.Humanoid.HipsTarget.DetermineTarget (HumanoidControl *humanoid*) `[static]`

Checks whether the humanoid has an HipsTarget and adds one if none has been found

Parameters

humanoid	The humanoid to check

override void Passer.Humanoid.HipsTarget.MatchTargetsToAvatar () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.InitializeTrackingConfidence () `[virtual]`

Sets the tracking confidence before all tracking information is updated

Implements Passer.Target.

override void Passer.Humanoid.HipsTarget.UpdateTarget () `[virtual]`

Implements Passer.Target.

override void Passer.Humanoid.HipsTarget.UpdateMovements (HumanoidControl *humanoid*) `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.CopyTargetToRig () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.CopyRigToTarget () `[virtual]`

Implements Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.DrawTargetRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

override void Passer.Humanoid.HipsTarget.DrawAvatarRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented from Passer.Humanoid.HumanoidTarget.

The documentation for this class was generated from the following files:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Movements/TorsoMovements.cs
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Targets/HipsTarget.cs

Passer.HumanoidAttachments Class Reference

Description

Attaches additional functionality to Humanoid on a Site

Inherits MonoBehaviourif hPHOTON2, and IPunPrefabPoolendif.

Public Attributes

- GameObject[] attachments
  The attachements to add to the Humanoid

Member Data Documentation

GameObject [] Passer.HumanoidAttachments.attachments

The attachements to add to the Humanoid

Note that these should be prefab resources (this is not yet checked)

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Sites/Scripts/HumanoidAttachments.cs

Passer.Humanoid.HumanoidButton Class Reference

Description

Unity UI button with information on which humanoid pressed the button

Unity provides an great UI system which includes a Button component which can call functions on objects when it is pressed. 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.

Inherits Button.

Classes

- class HumanoidEvent
  The Event taking a HumanoidControl parameter

Public Member Functions

- override void OnPointerClick (PointerEventData eventData)
  This function is called when the button is clicked

- override void OnSubmit (BaseEventData eventData)
  This function is called when the button is activated with the default button.

Public Attributes

- new HumanoidEvent onClick = new HumanoidEvent()
  The onClick event which replaces the standard onClick event

Protected Member Functions

- void **Press** (BaseEventData eventData)

Member Function Documentation

override void Passer.Humanoid.HumanoidButton.OnPointerClick (PointerEventData *eventData*)

This function is called when the button is clicked

Parameters

eventData	Event payload associated with the humanoid

override void Passer.Humanoid.HumanoidButton.OnSubmit (BaseEventData *eventData*)

This function is called when the button is activated with the default button.

This is not supported by Humanoid Control, but added for completeness ///

Parameters

eventData	Event payload associated with the humanoid

Member Data Documentation

new HumanoidEvent Passer.Humanoid.HumanoidButton.onClick = new HumanoidEvent()

The onClick event which replaces the standard onClick event

This version takes an HumanoidControl parameter The standard does not take a parameter

---

The documentation for this class was generated from the following file:

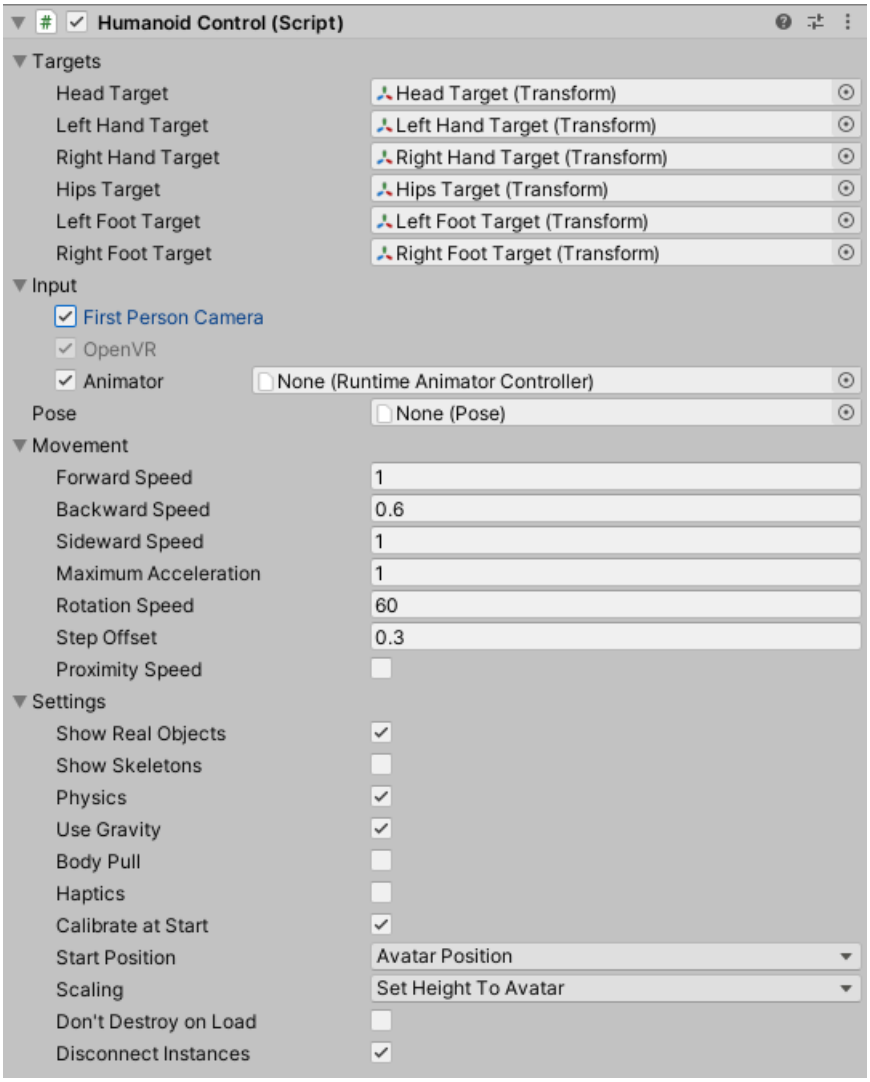
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Tools/HumanoidButton.cs

Passer.Humanoid.HumanoidControl Class Reference

Description

Control avatars using tracking and animation options

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



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
- Left/Right Hand Target
- Hips Target
- Left/Right 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. See the list of supported devices to get information on the settings forf each device.

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
- Backward Speed
- Sideward Speed
- Maximum Acceleration
- Rotation Speed
- Step Ofsset
- Proximity Speed

Settings

- Show Real Objects
- Physics
- Use Gravity
- Body Pull
- Haptics
- Calibrate at Start
- Start Position
- Scaling
- Don't Destroy on Load
- Disconnect Instances

Inherits MonoBehaviour.

Public Types

- enum **PrimaryTarget** { **Head**, **Hips** }
- enum **TargetId** { **Hips**, **Head**, **LeftHand**, **RightHand**, **LeftFoot**, **RightFoot**, **Face** }
- enum StartPosition { **AvatarPosition**, **PlayerPosition** }
  Types of startposition for the Pawn
- enum ScalingType { **None**, **SetHeightToAvatar**, **ScaleTrackingToAvatar**, **ScaleAvatarToTracking** }
  Types of Scaling which can be used to scale the tracking input to the size of the avatar

Public Member Functions

- void **ChangeAvatar** (GameObject fpAvatarPrefab)
- void **ChangeAvatar** (GameObject fpAvatarPrefab, GameObject tpAvatarPrefab)
- void **LocalChangeAvatar** (GameObject avatarPrefab)
- void **InitializeAvatar** ()
- void **InitAvatar** ()
  Analyses the avatar's properties requires for the movements

- void **ScaleAvatarToTracking** ()
- Animator **GetAvatar** (GameObject avatarRoot)
  Retrieve the avatar rig for this humanoid

- void **InitTargets** ()
  Initialize the targets for this humanoid

- void **DetermineTargets** ()
  Checks the humanoid for presence of Targets and adds them if they are not found

- void **MatchTargetsToAvatar** ()
  Changes the target rig transforms to match the avatar rig

- void **UpdateMovements** ()
  Updates the avatar pose based on the targets rig

- void **CopyRigToTargets** ()

Copies the pose of the avatar to the target rig

- void **UpdateSensorsFromTargets** ()
  Updated the sensor transform from the target transforms

- HumanoidTarget.TargetedBone GetBone (Bone boneId)
  Get the Humanoid Bone

- HumanoidTarget.TargetedBone GetBone (Side side, SideBone sideBoneId)
  Get the Humanoid Bone on the incated side of the humanoid

- void **ScaleTrackingToAvatar** ()
- void AdjustTracking (Vector3 translation)
  Adjust the tracking origin of all trackers

- void AdjustTracking (Vector3 translation, Quaternion rotation)
  Adjust the tracking origin of all trackers

- void **RetrieveBones** ()
  Scans the humanoid to retrieve all bones

- void **OnApplicationQuit** ()
- delegate void **OnNewNeckHeight** (float neckHeight)
- void **SetStartPosition** ()
- void **Calibrate** ()
  Calibrates the tracking with the player

- delegate void **OnHumanoidPose** (HumanoidPose pose)
- void MoveForward (float z)
  maximum forward speed in units(meters)/second

- void MoveSideward (float x)
  Moves the humanoid sideward

- virtual void Move (Vector2 velocity)
  Moves the humanoid in 2D space

- virtual void **Move** (Vector3 velocity)
  Moves the humanoid

- void **MoveWorldVector** (Vector3 v)
- void **Stop** ()
- void Rotate (float angularSpeed)
  Rotate the humanoid

- void **Rotation** (float yAngle)
  Set the rotation angle along the Y axis

- void Dash (Vector3 targetPosition)
  Quickly moves this humanoid to the given position

- void Teleport (Vector3 targetPosition)
  Teleports this humanoid to the given position

- void TeleportForward (float distance=1)
  Teleports the humanoid in the forward direction

- void MoveTo (Vector3 position, MovementType movementType=MovementType.Teleport)
  Moves the humanoid to the given position

- IEnumerator RotateTo (Quaternion targetRotation)
  Rotation animation to reach the given rotation

- IEnumerator LookAt (Vector3 targetPosition)
  Rotation animation to look at the given position

- IEnumerator WalkTo (Vector3 targetPosition)
  Movement animation to reach the given position

- IEnumerator WalkTo (Vector3 targetPosition, Quaternion targetRotation)
  Movement animation to reach the given position and rotation

- void Jump (float takeoffVelocity)
  Lets the Humanoid jump up with the given take off velocity

- Vector3 **CheckMovement** ()
- bool **IsMyRigidbody** (Rigidbody rigidbody)
- float **GetDistanceToGroundAt** (Vector3 position, float maxDistance)
- float **GetDistanceToGroundAt** (Vector3 position, float maxDistance, out Transform ground, out Vector3 normal)
- void **SetAnimationParameterBool** (string parameterName, bool boolValue)
- void **SetAnimationParameterFloat** (string parameterName, float floatValue)
- void **SetAnimationParameterInt** (string parameterName, int intValue)
- void **SetAnimationParameterTrigger** (string parameterName)
- Vector3 GetHumanoidPosition ()
  The humanoid can be on a differentlocation than the humanoid.transform because the tracking can move the humanoid around independently This function takes this into account

- delegate void **OnNewHumanoid** (HumanoidControl humanoid)
- bool IsVisible (Camera camera)
  Is this humanoid visible in the given camera?

Static Public Member Functions

- static void **SetControllerID** (HumanoidControl humanoid, int controllerID)
- static void **CheckTargetRig** (HumanoidControl humanoid)
  Match the target rig transform to the humanoid transform

- static GameObject GetRealWorld (Transform transform)
  Gets the Real World GameObject for this Humanoid

- static GameObject FindTrackerObject (GameObject realWorld, string trackerName)
  Tries to find a tracker GameObject by name

- static HumanoidControl[] AllVisibleHumanoids (Camera camera)
  Returns all humanoid which are in the frustum of the camera

Public Attributes

- string **path**
  The path at which the HumanoidControl script is found

- HeadTarget **headTarget**
- HandTarget **leftHandTarget**
  The target script to control the left arm bones

- HandTarget **rightHandTarget**
  The target script to control the right arm bones

- HipsTarget **hipsTarget**
  The target script to control the torso bones

- FootTarget **leftFootTarget**
  The target script to control the left leg bones

- FootTarget **rightFootTarget**
  The target script to control the right leg bones

- PrimaryTarget **primaryTarget**
- Animator targetsRig
  The target bones rig

- bool **showTargetRig** = false
  Draws the target rig in the scene view

- Animator avatarRig
  The avatar Rig

- bool **showAvatarRig** = true
  Draws the avatar rig in the scene view

- float **avatarNeckHeight**
  The neck height of the avatar

- bool **showMuscleTension** = false
  Draws the tension at the joints of the avatar

- bool calculateBodyPose = true
  Calculate the avatar pose

- bool **animatorEnabled** = true
  Enables the animator for this humanoid

- RuntimeAnimatorController **animatorController** = null
  The Animator for this humanoid

- Pose **pose**
  The pose of this humanoid

- bool **editPose**
  Is true when the pose is currently being edited

- IHumanoidNetworking **humanoidNetworking**
  The networking interface

- GameObject **remoteAvatar**
  The remote avatar prefab for this humanoid

- int playerType
  Is true when this is a remote avatar

- bool **syncRootTransform** = true
- string **remoteTrackerIpAddress**
- int **humanoidId** = -1
  The local Id of this humanoid

- Cerebellum.Cerebellum **cerebellum**
- bool **showRealObjects** = true
  If true, real world objects like controllers and cameras are shown in the scene

- bool **physics** = true
  Enables controller physics and collisions during walking.

- bool **useGravity** = true
  If there is not static object below the feet of the avatar the avatar will fall down until it reaches solid ground

- bool **generateColliders** = true
  If true, it wil generate colliders for the avatar where necessary

- bool **haptics** = false
  Will use haptic feedback on supported devices when the hands are colliding or touching objects

- StartPosition **startPosition** = StartPosition.AvatarPosition
  The start position of the humanoid

- bool **calibrateAtStart** = false
  Perform a calibration when the scene starts

- bool **dontDestroyOnLoad** = false
  This option will make sure that the humanoid is not destroyed when the scene is changed.

- bool **disconnectInstances** = false
  (Prefab only) Will disconnect the instance from the prefab when they are included in the scene.

- bool **gameControllerEnabled** = true
  Use game controller input

- Passer.Controller **controller**
  The game controller for this pawn

- int **gameControllerIndex**
  The index of the game controller

- GameControllers **gameController**
- UnityXRTracker unityXR = new UnityXRTracker()
  The Unity XR tracker

- WindowsMRTracker **mixedReality** = new WindowsMRTracker()
  The Windows Mixed Reality tracker

- WaveVRTracker **waveVR** = new WaveVRTracker()
  The Wave VR tracker

- NeuronTracker **neuronTracker** = new NeuronTracker()
  The Perception Neuron tracker

- LeapTracker **leap** = new LeapTracker()
  The Leap Motion tracker

- RealsenseTracker **realsense** = new RealsenseTracker()
  The Intel RealSense tracker

- HydraTracker **hydra** = new HydraTracker()
  The Razer Hydra tracker

- Kinect1Tracker **kinect1** = new Kinect1Tracker()
  The Microsoft Kinect 360/Kinect for Windows tracker

- Kinect2Tracker **kinect2** = new Kinect2Tracker()
  The Microsoft Kinect 2 tracker

- Kinect4Tracker **kinect4** = new Kinect4Tracker()
  Azure Kinect tracker

- AstraTracker **astra** = new AstraTracker()
  The Orbbec Astra tracker

- OptiTracker **optitrack** = new OptiTracker()
  The OptiTrack tracker

- TobiiTracker **tobiiTracker** = new TobiiTracker()
  The Tobii tracker

- ArKit **arkit** = new ArKit()
- Tracking.Pupil.Tracker **pupil** = new Tracking.Pupil.Tracker()
  The Pupil Labs tracker

- DlibTracker **dlib** = new DlibTracker()
  The Dlib tracker

- AntilatencyTracker **antilatency** = new AntilatencyTracker()
- Hi5Tracker **hi5** = new Hi5Tracker()
- CustomTracker **custom** = new CustomTracker()
- float **forwardSpeed** = 1
  maximum forward speed in units(meters)/second

- float **backwardSpeed** = 0.6F
  maximum backward speed in units(meters)/second

- float **sidewardSpeed** = 1
  maximum sideways speed in units(meters)/second

- float **maxAcceleration** = 1
  maximum acceleration in units(meters)/second/second value 0 = no maximum acceleration

- float **rotationSpeed** = 60
  maximum rotational speed in degrees/second

- float **stepOffset** = 0.3F
  The maximum height of objects of the ground which do not stop the humanoid

- bool **proximitySpeed** = false
  Reduces the walking speed of the humanoid when in the neighbourhood of objects to reduce motion sickness.

- float **proximitySpeedRate** = 0.8f

The amount of influence of the proximity speed. 1=No influence, 0 = Maximum

- bool **bodyPull** = false
  Will move the pawn position when grabbing handles on static objects

- Vector3 **velocity**
  The current velocity of the Pawn

- Vector3 **targetVelocity**
- Vector3 **acceleration**
- float **turningVelocity**
- bool **triggerEntered**
- bool **collided**
- Vector3 **hitNormal** = Vector3.zero
- Rigidbody **humanoidRigidbody**
- Rigidbody **characterRigidbody**
- CapsuleCollider **bodyCapsule**
- CapsuleCollider **bodyCollider**
- Transform ground
  The ground Transform on which the pawn is standing

- Vector3 **groundVelocity**
  The velocity of the ground on which the pawn is standing

- float groundAngularVelocity
  The angular velocity of the ground on which the pawn is standing

- bool **useLegLengthCorrection** = false
- bool **floatCorrection** = false
  Correct floating avatars when user is taller than the avatar

- string **animatorParameterForward**
- string **animatorParameterSideward**
- string **animatorParameterRotation**
- string **animatorParameterHeight**
- int **animatorParameterForwardIndex**
- int **animatorParameterSidewardIndex**
- int **animatorParameterRotationIndex**
- int **animatorParameterHeightIndex**
- bool isRemote = false
  Is true when this is a remote pawn

- ulong **nwId**
  The Id of this pawn across the network

- int **id** = -1
  The local Id of this humanoid

Protected Member Functions

- void **Awake** ()
- void **NewTargetComponents** ()
- void **StartTargets** ()
  Start the targets for this humanoid

- void **UpdateTargets** ()
- void **InitTrackers** ()
- void **StartTrackers** ()
- void **UpdateTrackers** ()
- void **InitializeTrackingConfidence** ()
- void **StartSensors** ()
- void **StopSensors** ()
- void SetTrackingHeightToAvatar ()
  Adjust Y position to match the tracking with the avatar

- void **Update** ()
- void **FixedUpdate** ()
- void **LateUpdate** ()
- virtual void **UpdatePose** ()
- virtual void **UpdatePoseEvent** ()
- void **CalculateMovement** ()
- void **CheckGround** ()
- void **CheckGrounded** ()
- void **CheckGroundMovement** ()
- virtual void **CheckBodyPull** ()
- void **KinematicBodyControlOneHanded** (HandTarget handTarget)
- void **KinematicBodyControlTwoHanded** ()
- void **Fall** ()

Protected Attributes

- ScalingType **scaling** = ScalingType.SetHeightToAvatar
  Scale Tracking to Avatar scales the tracking input to match the size of the avatar

- Transform **_realWorld**
- Vector3 **gravitationalVelocity**
- Vector3 **inputMovement** = Vector3.zero
- Transform **lastGround**
- Vector3 **lastGroundPosition** = Vector3.zero
- float **lastGroundAngle** = 0

Properties

- float trackingNeckHeight `[get]`
  The neck height of the target rig

- bool **showSkeletons** `[getset]`
  If enabled, tracking skeletons will be rendered

- Transform **realWorld** `[get]`
  The transform containing all real-world objects

- HumanoidTracker[] **trackers** `[get]`

All available trackers for this humanoid

- Vector3? **up** `[get]`
- static HumanoidControl[] **allHumanoids** `[get]`
  Returns all humanoids in the current scene.

Events

- OnNewNeckHeight **OnNewNeckHeightEvent**
- OnHumanoidPose **onHumanoidPose**
- static OnNewHumanoid **onNewHumanoid**
  Event triggered when a new humanoid enters the current scene

---

Member Enumeration Documentation

enum Passer.Humanoid.HumanoidControl.ScalingType

Types of Scaling which can be used to scale the tracking input to the size of the avatar

SetHeightToAvatar adjusts the vertical tracking to match the avatar size. MoveHeightToAvatar does the same but also resets the tracking origin to the location of the avatar. ScaleTrackingToAvatar scales the tracking space to match the avatar size. ScaleAvatarToTracking resizes the avatar to match the player size.

Member Function Documentation

HumanoidTarget.TargetedBone Passer.Humanoid.HumanoidControl.GetBone (Bone *boneId*)

Get the Humanoid Bone

Parameters

boneId	The identification of the requested bone

HumanoidTarget.TargetedBone Passer.Humanoid.HumanoidControl.GetBone (Side *side*, SideBone *sideBoneId*)

Get the Humanoid Bone on the incated side of the humanoid

Parameters

side	The requested side of the humanoid
sideBoneId	The identification of the requested bone

void Passer.Humanoid.HumanoidControl.SetTrackingHeightToAvatar () `[protected]`

Adjust Y position to match the tracking with the avatar

This function will adjust the vertical position of the tracking origin such that the tracking matches the avatar. This function should preferably be executed when the player is in a base position: either standing upright or sitting upright, depending on the playing pose. This will prevent the avatar being in the air or in a crouching position when the player is taller or smaller than the avatar itself. It retains 1:1 tracking and the X/Z position of the player are not affected.

void Passer.Humanoid.HumanoidControl.AdjustTracking (Vector3 *translation*)

Adjust the tracking origin of all trackers

Parameters

translation	The translation to apply to the tracking origin

void Passer.Humanoid.HumanoidControl.AdjustTracking (Vector3 *translation*, Quaternion *rotation*)

Adjust the tracking origin of all trackers

Parameters

translation	The translation to apply to the tracking origin
rotation	The rotation to apply to the tracking origin

void Passer.Humanoid.HumanoidControl.MoveForward (float *z*)

maximum forward speed in units(meters)/second

maximum backward speed in units(meters)/second

maximum sideways speed in units(meters)/second

maximum acceleration in units(meters)/second/second value 0 = no maximum acceleration

maximum rotational speed in degrees/second

Moves the humanoid forward

Parameters

z	The distance in units(meters) to move forward.

void Passer.Humanoid.HumanoidControl.MoveSideward (float *x*)

Moves the humanoid sideward

Parameters

x	The distance in units(meters) to move sideward.

virtual void Passer.Humanoid.HumanoidControl.Move (Vector2 *velocity*) `[virtual]`

Moves the humanoid in 2D space

Parameters

velocity	The velocity to move with

The velocity axis are mapped as follows: x = left/right movement y = forward/backward movement

void Passer.Humanoid.HumanoidControl.Rotate (float *angularSpeed*)

Rotate the humanoid

Rotates the humanoid along the Y axis

Parameters

angularSpeed	The speed in degrees per second

void Passer.Humanoid.HumanoidControl.Dash (Vector3 *targetPosition*)

Quickly moves this humanoid to the given position

Parameters

targetPosition	The position to move to

void Passer.Humanoid.HumanoidControl.Teleport (Vector3 *targetPosition*)

Teleports this humanoid to the given position

Parameters

targetPosition	The position to move to

void Passer.Humanoid.HumanoidControl.TeleportForward (float *distance* = `1`)

Teleports the humanoid in the forward direction

Parameters

distance	The distance to teleport

The forward direction is determined by the hips target forward.

void Passer.Humanoid.HumanoidControl.MoveTo (Vector3 *position*, MovementType *movementType* = `MovementType.Teleport`)

Moves the humanoid to the given position

Parameters

movementType	The type of movement to use

IEnumerator Passer.Humanoid.HumanoidControl.RotateTo (Quaternion *targetRotation*)

Rotation animation to reach the given rotation

The speed of the rotation is determined by the rotationSpeed

Parameters

targetRotation	The target rotation

IEnumerator Passer.Humanoid.HumanoidControl.LookAt (Vector3 *targetPosition*)

Rotation animation to look at the given position

The speed of the rotation is determined by the rotationSpeed

Parameters

targetPosition	The position to look at

Returns

IEnumerator Passer.Humanoid.HumanoidControl.WalkTo (Vector3 *targetPosition*)

Movement animation to reach the given position

The pawn will walk to the given position. The speed of the movement is determined by forwardSpeed. Any required rotation will be limited by rotationSpeed

Parameters

targetPosition	The position to where the pawn should walk

Returns

IEnumerator Passer.Humanoid.HumanoidControl.WalkTo (Vector3 *targetPosition*, Quaternion *targetRotation*)

Movement animation to reach the given position and rotation

The pawn will walk to the given position and rotation. The speed of the movement is determined by forwardSpeed. Any required rotation will be limited by rotationSpeed

Parameters

targetPosition	The position to where the pawn should walk
targetRotation	The rotation the pawn should have at the end

Returns

void Passer.Humanoid.HumanoidControl.Jump (float *takeoffVelocity*)

Lets the Humanoid jump up with the given take off velocity

Parameters

takeoffVelocity	The vertical velocity to start the jump

static GameObject Passer.Humanoid.HumanoidControl.GetRealWorld (Transform *transform*) `[static]`

Gets the Real World GameObject for this Humanoid

Parameters

transform	The root transform of the humanoid

static GameObject Passer.Humanoid.HumanoidControl.FindTrackerObject (GameObject *realWorld*, string *trackerName*) [static]

Tries to find a tracker GameObject by name

Parameters

realWorld	The Real World GameOject in which the tracker should be
trackerName	The name of the tracker GameObject to find

Vector3 Passer.Humanoid.HumanoidControl.GetHumanoidPosition ()

The humanoid can be on a differentlocation than the humanoid.transform because the tracking can move the humanoid around independently This function takes this into account

Returns

The position of the humanoid

static HumanoidControl[] Passer.Humanoid.HumanoidControl.AllVisibleHumanoids (Camera *camera*) [static]

Returns all humanoid which are in the frustum of the camera

Parameters

camera	The camera in which the humanoids should be visible

Returns

An array of all visible humanoids

bool Passer.Humanoid.HumanoidControl.IsVisible (Camera *camera*)

Is this humanoid visible in the given camera?

Parameters

camera	The camera in which the humaonid may be visible

Returns

true when the humanoid is visible in the camera

Member Data Documentation

Animator Passer.Humanoid.HumanoidControl.targetsRig

The target bones rig

The target bones rig contain the target pose of the avatar The humanoid movements will try to move the avatar such that the target pose is reached as closely as possible

Animator Passer.Humanoid.HumanoidControl.avatarRig

The avatar Rig

This is the rig of the avatar we want to control.

bool Passer.Humanoid.HumanoidControl.calculateBodyPose = true

Calculate the avatar pose

When this option is enabled, the pose of the avatar is updated from the target rig using the Humanoid movements. If you are only interested in the tracking result, you can disable this option and use the target rig to access the tracked pose.

int Passer.Humanoid.HumanoidControl.playerType

Is true when this is a remote avatar

Remote avatars are not controlled locally, but are controlled from another computer. These are copies of the avatar on that other computer and are updated via messages exchanges on a network.

The Id of this humanoid across the network

The Player Type of the humanoid

UnityXRTracker Passer.Humanoid.HumanoidControl.unityXR = new UnityXRTracker()

The Unity XR tracker

\

Transform Passer.Humanoid.HumanoidControl.ground

The ground Transform on which the pawn is standing

When the pawn is not standing on the ground, the value is null

float Passer.Humanoid.HumanoidControl.groundAngularVelocity

The angular velocity of the ground on which the pawn is standing

The velocity is in degrees per second along the Y axis of the pawn

bool Passer.Humanoid.HumanoidControl.isRemote = false

Is true when this is a remote pawn

Remote pawns are not controlled locally, but are controlled from another computer. These are copies of the pawn on that other computer and are updated via messages exchanges on a network.

Property Documentation

float Passer.Humanoid.HumanoidControl.trackingNeckHeight `[get]`

The neck height of the target rig

When head tracking is used, this can be used to estimate the height of the player.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/HumanoidControl.cs

Passer.Humanoid.HumanoidButton.HumanoidEvent Class Reference

Description

The Event taking a HumanoidControl parameter

Inherits UnityEvent< HumanoidControl >.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Tools/HumanoidButton.cs

Passer.Humanoid.HumanoidNetworking Class Reference

Description

Humanoid Networking

Public Types

- enum **Smoothing** { **None**, **Interpolation**, **Extrapolation** }
- enum **DebugLevel** { **Debug**, **Info**, **Warning**, **Error**, **None** }

Public Member Functions

- delegate void **ConnectedToNetwork** (HumanoidControl humanoid)
- delegate void **NewRemoteHumanoidArgs** (HumanoidControl humanoid)

Static Public Member Functions

- static void **Connected** (HumanoidControl humanoid)
- static void **ReceiveInstantiate** (this IHumanoidNetworking networking, byte[] serializedData)
- static void **Receive** (this IHumanoidNetworking receivingNetworking, InstantiateHumanoid msg)
- static HumanoidControl **FindRemoteHumanoid** (List< HumanoidControl > humanoids, ulong nwId, int humanoidId)
- static void **ReceiveDestroy** (this IHumanoidNetworking networking, byte[] serializedData)
- static void **Receive** (this IHumanoidNetworking networking, DestroyHumanoid msg)
- static void **ReceiveHumanoidPose** (this IHumanoidNetworking networking, byte[] data)
- static void **Receive** (this IHumanoidNetworking receivingNetworking, HumanoidPose humanoidPose)
- static void **ReceiveHumanoidPose** (HumanoidControl remoteHumanoid, HumanoidPose humanoidPose, HumanoidPose lastHumanoidPose, HumanoidNetworking.Smoothing smoothing)
- static bool **IsTargetActive** (byte targetMask, HumanoidControl.TargetId targetIndex)
- static void **ReceiveGrab** (this IHumanoidNetworking networking, byte[] serializedData)
- static void **Receive** (this IHumanoidNetworking networking, Grab msg)
- static void **ReceiveLetGo** (this IHumanoidNetworking networking, byte[] serializedData)
- static void **Receive** (this IHumanoidNetworking networking, LetGo msg)
- static void **ReceiveChangeAvatar** (this IHumanoidNetworking networking, byte[] serializedData)
- static void **Receive** (this IHumanoidNetworking receivingNetworking, ChangeAvatar msg)
- static Transform **GetTrackingTransform** (HumanoidControl humanoid)
- static void **ReceiveSyncTrackingSpace** (this IHumanoidNetworking networking, byte[] serializedData)
- static void **Receive** (this IHumanoidNetworking networking, SyncTrackingSpace msg)
- static IHumanoidNetworking **GetHumanoidNetworking** (IHumanoidNetworking networking, ulong nwId)
- static List< HumanoidControl > **FindLocalHumanoids** ()
- static IHumanoidNetworking **GetLocalHumanoidNetworking** ()
- static void **DisableNetworkSync** (GameObject obj)
- static void **ReenableNetworkSync** (GameObject obj)
- static void **TakeOwnership** (GameObject obj)
- static HumanoidControl **FindHumanoid** (List< HumanoidControl > humanoids, int humanoidId)
- static HumanoidControl **FindLocalHumanoid** (List< HumanoidControl > humanoids, int humanoidId)
- static HumanoidControl **FindRemoteHumanoid** (List< HumanoidControl > humanoids, int humanoidId)
- static void **SmoothUpdate** (List< HumanoidControl > humanoids)
- static void **SmoothUpdate** (HumanoidControl humanoid)

Static Public Attributes

- static DebugLevel **debug** = DebugLevel.Error

Events

- static ConnectedToNetwork **OnConnectedToNetwork**
- static NewRemoteHumanoidArgs **OnNewRemoteHumanoid**

The documentation for this class was generated from the following file:

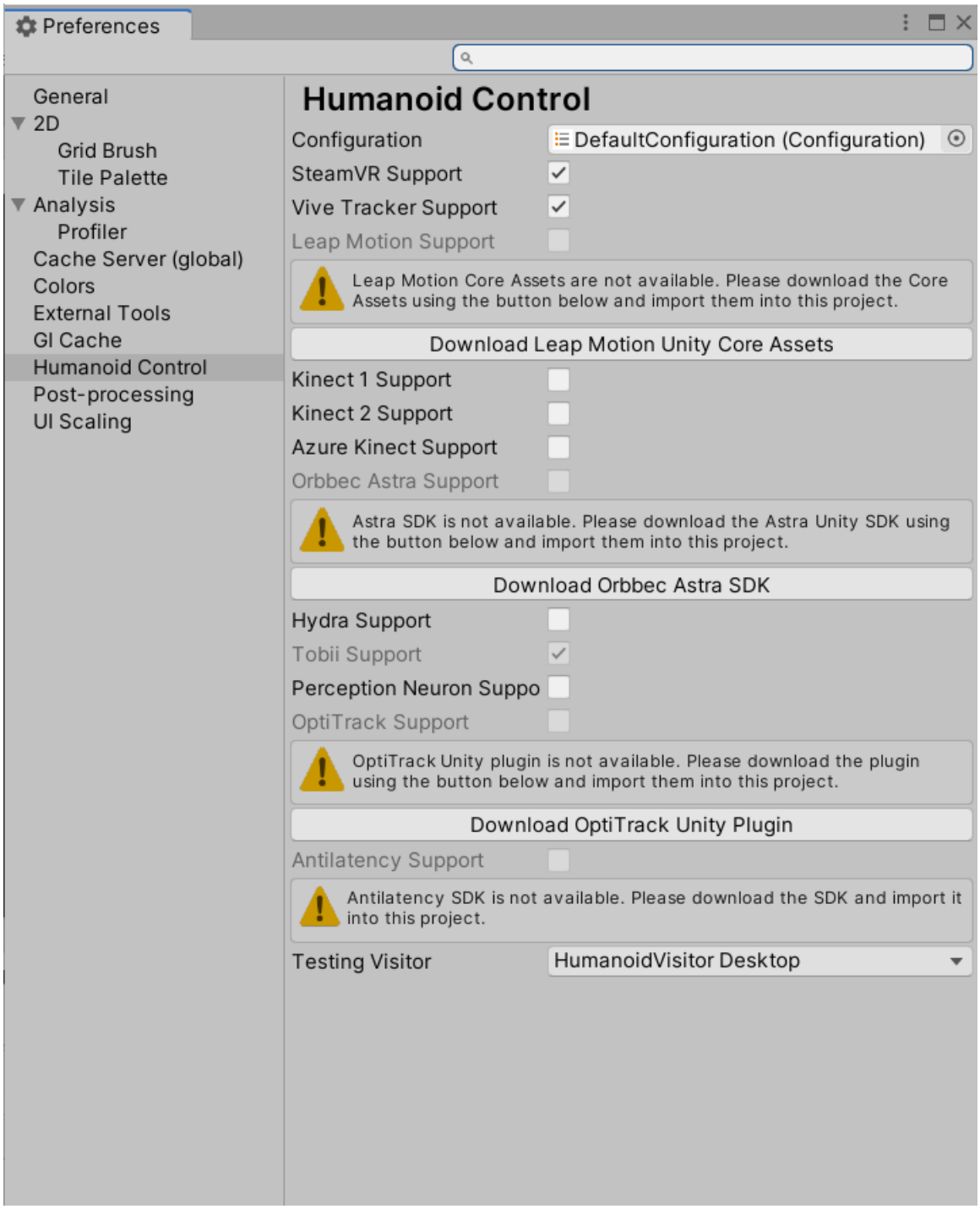
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Networking/HumanoidNetworking.cs

Passer.Humanoid.HumanoidPreferences Class Reference

Description

Sets preferences for Humanoid Control

The preferences can be accessed by the Edit->Preferences... menu.



- SteamVR Support, enables support for SteamVR devices beyound Unity XR (VR, Plus, Pro)
- Vive Tracker Support, enables support for Vive Trackers (Plus, Pro)
- Leap Motion Support, enables support for Leap Motion (Plus, Pro)
- Kinect 1 Support, enables support for Microsoft Kinect 360/Kinect for Windows (Plus, Pro)
- Kinect 2 Support, enables support for Microsoft Kinect 2 (Plus, Pro)
- Azure Kinect Support, enables support for Azure Kinect (Plus, Pro)
- Orbbec Astra Support, enables support for Orbbec Astra (Plus, Pro)
- Hydra Support, enables support for Razer Hydra (Pro)
- Tobii Support, enables support for Tobii desktop eye tracking (Pro)
- Perception Neuron Support, enables support for Perception Neuron (Pro)
- OptiTrack Support, enables support for OptiTrack (Pro)
- Antilatency Support, enables support for Antilatency (Pro)
- The Testing Visitor selects which Visitor Scene will be used when a Site is started in Play mode.

Inherits ScriptableObject.

Public Attributes

- Configuration **configuration**

Static Public Attributes

- static bool **help** = false
- const string **settingsPath** = "Assets/Passer/Humanoid/HumanoidPreferences.asset"

Properties

- static NetworkingSystems **networkingSupport** `[getset]`
- static string **visitorSceneName** `[getset]`

The documentation for this class was generated from the following file:

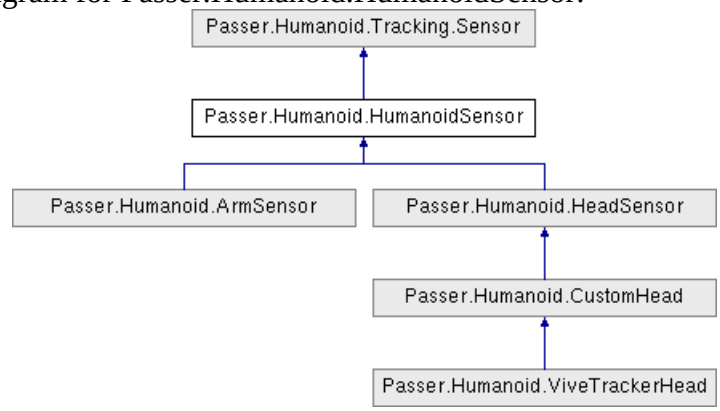
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/HumanoidPreferences.cs

Passer.Humanoid.HumanoidSensor Class Reference

Description

A sensor used to control a humanoid

Inheritance diagram for Passer.Humanoid.HumanoidSensor:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- virtual void Start (HumanoidControl _humanoid, Transform targetTransform)
- virtual void **CheckSensorTransform** ()
- virtual void **SetSensor2Target** ()
- override void Update ()
  Update the sensor state

- virtual void **UpdateSensorTransformFromTarget** (Transform targetTransform)
- virtual void **Stop** ()
- virtual void **RefreshSensor** ()
- virtual void **ShowSensor** (HumanoidTarget target, bool shown)

Static Public Member Functions

- static Vector3 **InverseTransformPointUnscaled** (Transform transform, Vector3 position)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- Vector3 **sensor2TargetPosition**
- Quaternion **sensor2TargetRotation**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Static Public Attributes

- const string **_name** = ""

Protected Member Functions

- virtual void **CreateSensorTransform** ()
- void **CreateSensorTransform** (Transform targetTransform, string resourceName, Vector3 _sensor2TargetPosition, Quaternion _sensor2TargetRotation)
- void **RemoveSensorTransform** ()
- void **UpdateSensorTransform** (Tracking.Sensor sensor)
- virtual void **UpdateTargetTransform** ()
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, Transform sensorTransform)
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, SensorComponent sensorComponent)
- Vector3 **GetTargetPosition** (Transform sensorTransform)
- Quaternion **GetTargetRotation** (Transform sensorTransform)
- void **UpdateSensor** ()

Static Protected Member Functions

- static Vector3 **TransformPointUnscaled** (Transform transform, Vector3 position)

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- virtual new HumanoidTracker **tracker** `[get]`
- override string **name** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`

Member Function Documentation

virtual void Passer.Humanoid.HumanoidSensor.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) `[virtual]`

Reimplemented in Passer.Humanoid.CustomHead.

override void Passer.Humanoid.HumanoidSensor.Update () `[virtual]`

Update the sensor state

Returns

Status of the sensor after the update

Reimplemented from Passer.Humanoid.Tracking.Sensor.

Reimplemented in Passer.Humanoid.CustomHead.

The documentation for this class was generated from the following file:

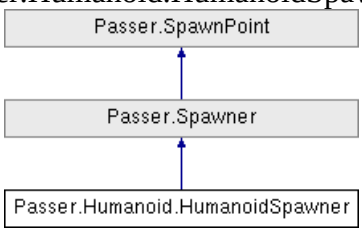
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/HumanoidSensor.cs

Passer.Humanoid.HumanoidSpawner Class Reference

Description

Component for spawning humanoids

Inheritance diagram for Passer.Humanoid.HumanoidSpawner:



Public Types

- enum SpawnMethod { SinglePlayer, Random, RoundRobin }
  Spawning methods.

Public Member Functions

- override GameObject SpawnObject ()
  Spawn one humanoid using the HumanoidSpawner settings.

- virtual HumanoidControl SpawnHumanoid ()
  Spawn one humanoid using the HumanoidSpawner settings.

- virtual HumanoidControl Spawn (HumanoidControl humanoidPrefab)
  Spawn a humanoid

- HumanoidControl Spawn (HumanoidControl humanoidPrefab, SpawnPoint[] spawnPoints, SpawnMethod spawnMethod=SpawnMethod.RoundRobin)
  Spawn a humanoid.

- void **DoSpawn** (GameObject prefab)
- virtual GameObject Spawn (GameObject prefab)
  Spawn a GameObject

- virtual GameObject Spawn (GameObject prefab, SpawnPoint[] spawnPoints, SpawnMethod spawnMethod=SpawnMethod.RoundRobin)
  Spawn a GameObject

Static Public Member Functions

- static GameObject **Spawn** (GameObject prefab, Vector3 position, Quaternion rotation)

Public Attributes

- GameObject **prefab**
  The prefab which will be spawned

- SpawnPoint[] spawnPoints
  The available SpawnPoints.

- SpawnMethod **spawnMethod**
  The SpawnMethod to use for spawning the humanoids.

- bool **spawnAtStart**
  Spawn an object when the scene starts

- bool singleInstance
  Will spawn only one object for this prefab

Protected Member Functions

- override void Start ()
- int **FindSpawnPointIndex** ()
- void **DestroyHumanoid** (HumanoidControl humanoid)
- virtual void **OnEnable** ()
- SpawnPoint **ChooseSpawnPoint** (SpawnPoint[] spawnPoints, SpawnMethod spawnMethod)

Static Protected Member Functions

- static HumanoidControl **AddHumanoidToAvatar** (GameObject avatar)

Protected Attributes

- HumanoidControl[] **spawnedHumanoids**
- int **spawnIndex** = 0

Static Protected Attributes

- static int **nHumanoids**

Member Enumeration Documentation

enum Passer.Spawner.SpawnMethod `[inherited]`

Spawning methods.

Enumerator:

SinglePlayer	Only one humanoid will be spawned. It will be located at the first SpawnPoint.
Random	A SpawnPoint is chosen randomly.
RoundRobin	Spawn points are chosen in round robin manner.

Member Function Documentation

override void Passer.Humanoid.HumanoidSpawner.Start () [protected], [virtual]

Reimplemented from Passer.Spawner.

override GameObject Passer.Humanoid.HumanoidSpawner.SpawnObject () [virtual]

Spawn one humanoid using the HumanoidSpawner settings.

Returns

The HumanoidControl of the spawned humanoid. Will be null when the humanoid could be not spawned.

Reimplemented from Passer.Spawner.

virtual HumanoidControl Passer.Humanoid.HumanoidSpawner.SpawnHumanoid () [virtual]

Spawn one humanoid using the HumanoidSpawner settings.

Returns

The HumanoidControl of the spawned humanoid. Will be null when the humanoid could be not spawned.

virtual HumanoidControl Passer.Humanoid.HumanoidSpawner.Spawn (HumanoidControl *humanoidPrefab*) [virtual]

Spawn a humanoid

Parameters

prefab	The humanoid prefab to spawn.

Returns

The HumanoidControl of the spawned humanoid. Will be null when the humanoid could be not spawned.

HumanoidControl Passer.Humanoid.HumanoidSpawner.Spawn (HumanoidControl *humanoidPrefab*, SpawnPoint[] *spawnPoints*, SpawnMethod *spawnMethod* = SpawnMethod.RoundRobin)

Spawn a humanoid.

Parameters

humanoidPrefab	The humanoid prefab to spawn.
spawnPoints	The array of possible spawn points.
spawnMethod	The SpawnMethod to use for spawning.

Returns

The HumanoidControl of the spawned humanoid. Will be null when the humanoid could be not spawned.

virtual GameObject Passer.Spawner.Spawn (GameObject *prefab*) [virtual], [inherited]

Spawn a GameObject

Parameters

prefab	The GameObject prefab to spawn.

Returns

The instantiated GameObject. Will be null when the prefab could not be spawned.

virtual GameObject Passer.Spawner.Spawn (GameObject *prefab*, SpawnPoint[] *spawnPoints*, SpawnMethod *spawnMethod* = SpawnMethod.RoundRobin) [virtual], [inherited]

Spawn a GameObject

Parameters

prefab	The GameObject prefab to spawn.
spawnPoints	The array of possible spawn points.
spawnMethod	The SpawnMethod to use for spawning.

Returns

The instantiated GameObject. Will be null when the prefab could not be spawned.

Member Data Documentation

SpawnPoint [] Passer.Spawner.spawnPoints [inherited]

The available SpawnPoints.

When the list is empty, the scene will be searched for available spawn points when it becomes enabled. If no spawn points can be found, this transform will be used as a SpawnPoint

bool Passer.Spawner.singleInstance [inherited]

Will spawn only one object for this prefab

When spawn is called a second time for the same prefab, the original spawned object will be teleported to the new spawn point

---

The documentation for this class was generated from the following file:

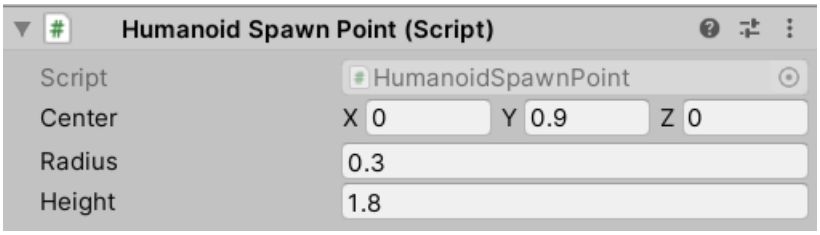
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Multiplayer/HumanoidSpawner.cs

Passer.Humanoid.HumanoidSpawnPoint Class Reference

Description

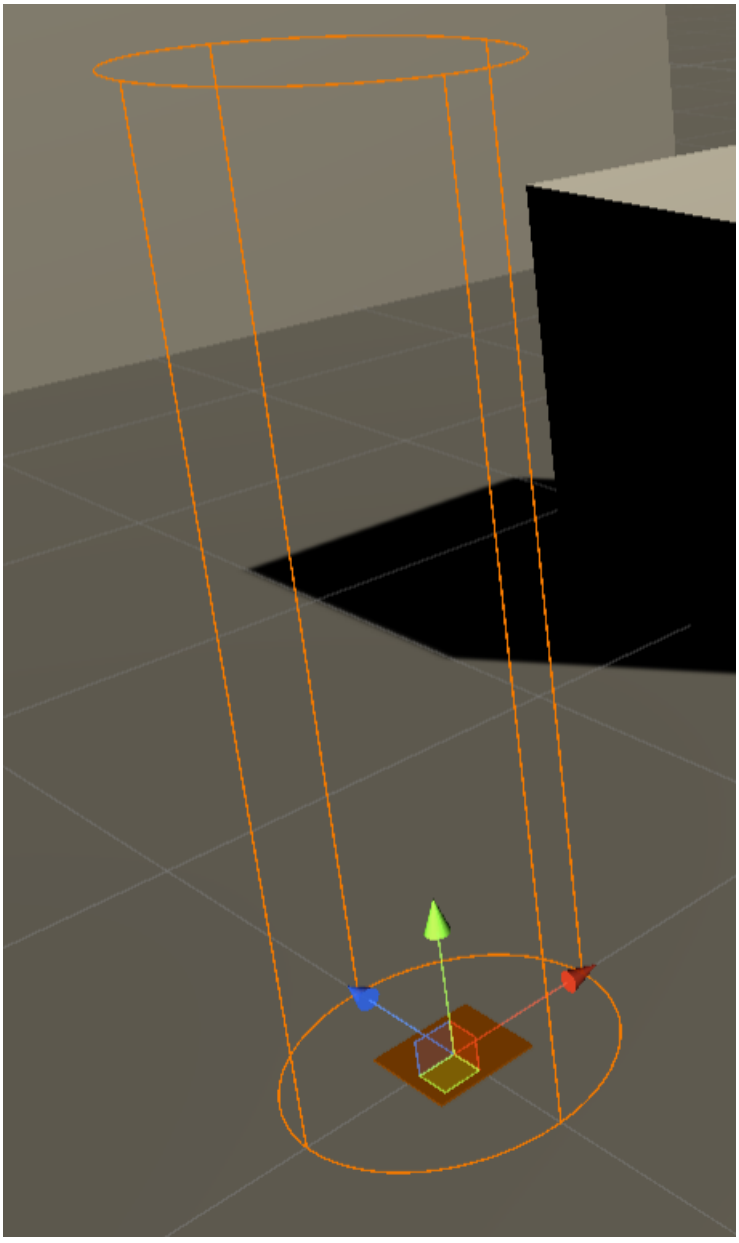
Specifies where a new Humanoid will spawn

At the beginning of the scene, the humanoid in will be moved to this transform if it is free. The check whether the spawn point is free is determined by the center, radius and height parameters which define the capsule which is checked with the isFree function.

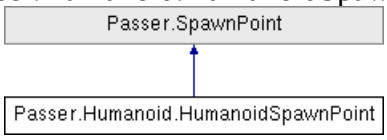


- Center
- Radius
- Height

In the scene view, the capsule is visualized as an orange cylinder when the spawn point is selected:



Inheritance diagram for Passer.Humanoid.HumanoidSpawnPoint:



Public Attributes

- Vector3 **center** = new Vector3(0, 0.9F, 0)
  The center of the capsule which is used to check whether the spawn point is free

- float **radius** = 0.3F
  The radius of the capsule which is used to check whether the spawn point is free

- float **height** = 1.8F
  The height of the capsule which is used to check whether the spawn point is free

Properties

- bool **isFree** `[get]`

The documentation for this class was generated from the following file:

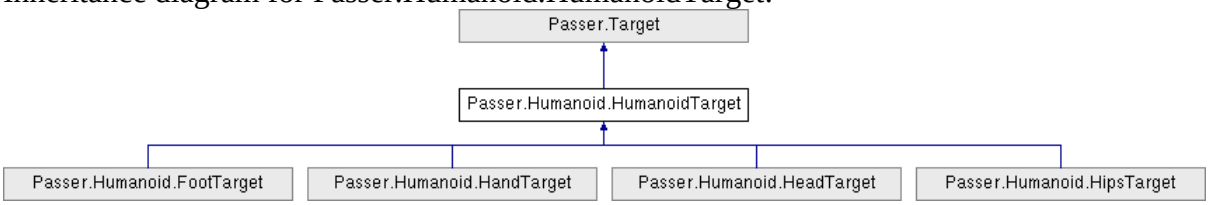
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/HumanoidSpawnPoint.cs

Passer.Humanoid.HumanoidTarget Class Reference

Description

A tracking transform for humanoids

Inheritance diagram for Passer.Humanoid.HumanoidTarget:



Public Member Functions

- abstract Transform **GetDefaultTarget** (HumanoidControl humanoid)
- abstract void **InitAvatar** ()
- virtual void **NewComponent** (HumanoidControl _humanoid)
- void **OnDrawGizmos** ()
- void **OnDrawGizmosSelected** ()
- virtual void **DrawTensions** ()
- abstract void UpdateMovements (HumanoidControl humanoid)
- abstract void **MatchTargetsToAvatar** ()
- abstract void CopyTargetToRig ()
- abstract void CopyRigToTarget ()
- virtual void **InitComponent** ()
- abstract void **StartTarget** ()
- abstract void **InitSensors** ()
- virtual void **StartSensors** ()
- virtual void **StopSensors** ()
- abstract void InitializeTrackingConfidence ()
  Sets the tracking confidence before all tracking information is updated

- abstract void UpdateTarget ()

Static Public Member Functions

- static Vector **ToVector** (Vector3 vector3)
- static Vector3 **ToVector3** (Vector position)
- static Rotation **ToRotation** (Quaternion quaternion)
- static Quaternion **ToQuaternion** (Rotation orientation)
- static void **SetRotation** (Transform transform, Rotation orientation)
- static void **DrawTarget** (Confidence confidence, Transform target, Vector3 direction, float length)
- static void **DrawTargetBone** (TargetedBone bone, Vector3 direction)
- static void **DrawTargetBone** (TargetTransform target, Vector3 direction)
- static void **DrawAvatarBone** (TargetedBone bone, Vector3 direction)
- static void **DrawAvatarBone** (BoneTransform bone, Vector3 direction)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, Bone boneId, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, HumanBodyBones boneID, params string[] boneNames)
- static void **GetDefaultTargetBone** (Animator rig, ref Transform boneTransform, Bone boneID, params string[] boneNames)
- static void **GetDefaultBone** (Animator rig, ref Transform boneTransform, params string[] boneNames)

- static List< Collider > **SetColliderToTrigger** (GameObject obj)
- static List< Collider > **SetColliderToTrigger** (Rigidbody rb)
- static void **UnsetColliderToTrigger** (List< Collider > colliders)
- static void **UnsetColliderToTrigger** (List< Collider > colliders, Collider collider)

Public Attributes

- HumanoidControl **humanoid**

Protected Member Functions

- virtual void DrawTargetRig (HumanoidControl humanoid)
- virtual void DrawAvatarRig (HumanoidControl humanoid)
- virtual void **DrawTensionGizmo** (TargetedBone targetedBone)
- virtual void **UpdateSensors** ()

Protected Attributes

- bool **_showRealObjects** = true

Properties

- abstract TargetedBone **main** `[get]`
- abstract Sensor **animator** `[get]`
- virtual bool **showRealObjects** `[getset]`
  show the target meshes

Member Function Documentation

virtual void Passer.Humanoid.HumanoidTarget.DrawTargetRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented in Passer.Humanoid.HeadTarget.

virtual void Passer.Humanoid.HumanoidTarget.DrawAvatarRig (HumanoidControl *humanoid*) `[protected], [virtual]`

Reimplemented in Passer.Humanoid.HeadTarget.

abstract void Passer.Humanoid.HumanoidTarget.UpdateMovements (HumanoidControl *humanoid*) `[pure virtual]`

Implemented in Passer.Humanoid.HeadTarget.

abstract void Passer.Humanoid.HumanoidTarget.CopyTargetToRig () `[pure virtual]`

Implemented in Passer.Humanoid.HeadTarget.

abstract void Passer.Humanoid.HumanoidTarget.CopyRigToTarget () `[pure virtual]`

Implemented in Passer.Humanoid.HeadTarget.

abstract void Passer.Target.InitializeTrackingConfidence () `[pure virtual], [inherited]`

Sets the tracking confidence before all tracking information is updated

Implemented in Passer.Humanoid.FootTarget, Passer.Humanoid.HandTarget, Passer.Humanoid.HeadTarget, and Passer.Humanoid.HipsTarget.

abstract void Passer.Target.UpdateTarget () `[pure virtual], [inherited]`

Implemented in Passer.Humanoid.HeadTarget.

The documentation for this class was generated from the following file:

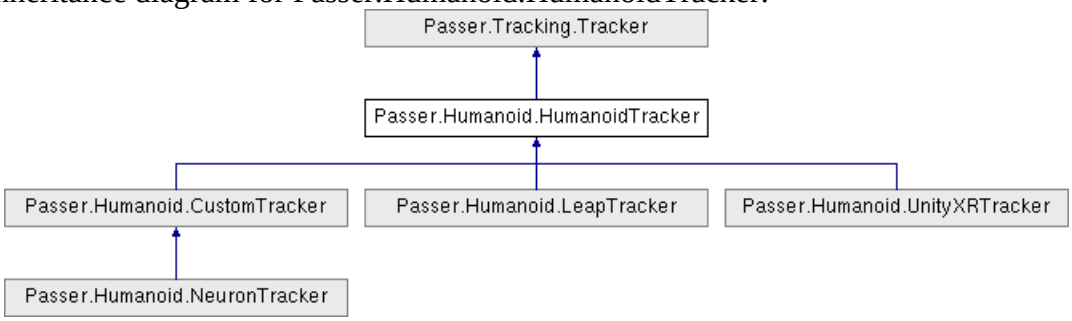
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Targets/HumanoidTarget.cs

Passer.Humanoid.HumanoidTracker Class Reference

Description

A Humanoid tracker

Inheritance diagram for Passer.Humanoid.HumanoidTracker:



Public Types

- enum Status { Unavailable, Present, Tracking }
  The tracking status

Public Member Functions

- virtual void CheckTracker (HumanoidControl humanoid)
  Check the status of the tracker

- delegate TrackerComponent TrackerGetter (Transform transform, Vector3 localPosition, Quaternion localRotation)
  Function delegate for retrieving the tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker)
  Function to check the status of a specific tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker, Vector3 localPosition, Quaternion localRotation)
  Function to check the status of a specific tracker

- virtual Vector3 **GetBonePosition** (uint actorId, FacialBone boneId)
- virtual Quaternion **GetBoneRotation** (uint actorId, FacialBone boneId)
- virtual float **GetBoneConfidence** (uint actorId, FacialBone boneId)
- virtual void StartTracker (HumanoidControl _humanoid)
  Start the tracker

- virtual void StartTracker ()
  Optional list of SubTrackers

- virtual void StopTracker ()
  Stop the tracker

- virtual void UpdateTracker ()

Update the tracker state

- virtual void ShowTracker (bool shown)
  Show or hide the Tracker renderers

- virtual void Calibrate ()
  Calibrate the tracker

- virtual void AdjustTracking (Vector3 positionalDelta, Quaternion rotationalDelta)
  Adjust the position of the tracker by the given delat

Public Attributes

- HumanoidControl **humanoid**
  The humanoid for this tracker

- bool **enabled**
  Is this tracker enabled?

- Status **status**
  The tracking Status of the tracker

- TrackerComponent **trackerComponent**
  The tracking device

Properties

- virtual HeadSensor headSensor `[get]`
  Get the sensor for the head

- virtual ArmSensor leftHandSensor `[get]`
  Get the sensor for the left hand

- virtual ArmSensor rightHandSensor `[get]`
  Get the sensor for the right hand

- virtual TorsoSensor hipsSensor `[get]`
  Get the sensor for the hips

- virtual LegSensor leftFootSensor `[get]`
  Get the sensor for the left foot

- virtual LegSensor rightFootSensor `[get]`
  Get the sensor for the right foot

- virtual HumanoidSensor[] **sensors** `[get]`
  The sensors for this tracker

- virtual string **name** `[get]`
  The name of this tracker

Member Enumeration Documentation

enum Passer.Tracking.Tracker.Status `[inherited]`

The tracking status

Enumerator:

Unavailable	The tracking device is not available.
Present	The tracking device is available but not tracking.
Tracking	The tracking device is actively tracking.

Member Function Documentation

virtual void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*) `[virtual]`

Check the status of the tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked

Reimplemented in Passer.Humanoid.UnityXRTracker, Passer.Humanoid.LeapTracker, and Passer.Humanoid.NeuronTracker.

delegate TrackerComponent Passer.Humanoid.HumanoidTracker.TrackerGetter (Transform *transform*, Vector3 *localPosition*, Quaternion *localRotation*)

Function delegate for retrieving the tracker

Parameters

transform	The root transform to start the searching of the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

Returns

The tracker component found or created

The default position/rotation is relative to the humanoid's real world.

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*)

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker

The default position/rotation for the tracker when created will be zero

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*, Vector3 *localPosition*, Quaternion *localRotation*)

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

virtual void Passer.Humanoid.HumanoidTracker.StartTracker (HumanoidControl *_humanoid*) `[virtual]`

Start the tracker

Reimplemented in Passer.Humanoid.LeapTracker, Passer.Humanoid.UnityXRTracker, and Passer.Humanoid.NeuronTracker.

virtual void Passer.Tracking.Tracker.StartTracker () `[virtual], [inherited]`

Optional list of SubTrackers

Start the tracker

virtual void Passer.Tracking.Tracker.StopTracker () `[virtual], [inherited]`

Stop the tracker

Reimplemented in Passer.Humanoid.LeapTracker.

virtual void Passer.Tracking.Tracker.UpdateTracker () `[virtual], [inherited]`

Update the tracker state

Reimplemented in Passer.Humanoid.UnityXRTracker, Passer.Humanoid.LeapTracker, Passer.Humanoid.NeuronTracker, and Passer.Humanoid.CustomTracker.

virtual void Passer.Tracking.Tracker.ShowTracker (bool *shown*) `[virtual], [inherited]`

Show or hide the Tracker renderers

Parameters

shown	Renderers are enabled when shown == true

virtual void Passer.Tracking.Tracker.Calibrate () `[virtual], [inherited]`

Calibrate the tracker

Reimplemented in Passer.Humanoid.UnityXRTracker.

virtual void Passer.Tracking.Tracker.AdjustTracking (Vector3 *positionalDelta*, Quaternion *rotationalDelta*) `[virtual], [inherited]`

Adjust the position of the tracker by the given delat

Parameters

positionalDelta	The positional delta to apply
rotationalDelta	The rotational delta to apply

Property Documentation

virtual HeadSensor Passer.Humanoid.HumanoidTracker.headSensor `[get]`

Get the sensor for the head

Will return null when this sensor is not present

virtual ArmSensor Passer.Humanoid.HumanoidTracker.leftHandSensor `[get]`

Get the sensor for the left hand

Will return null when this sensor is not present

virtual ArmSensor Passer.Humanoid.HumanoidTracker.rightHandSensor `[get]`

Get the sensor for the right hand

Will return null when this sensor is not present

virtual TorsoSensor Passer.Humanoid.HumanoidTracker.hipsSensor `[get]`

Get the sensor for the hips

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.leftFootSensor `[get]`

Get the sensor for the left foot

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.rightFootSensor `[get]`

Get the sensor for the right foot

Will return null when this sensor is not present

The documentation for this class was generated from the following file:

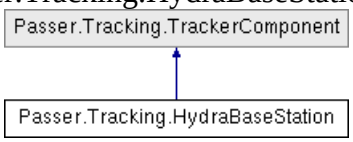
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/HumanoidTracker.cs

Passer.Tracking.HydraBaseStation Class Reference

Description

A Razer Hydra hand tracking device

Inheritance diagram for Passer.Tracking.HydraBaseStation:



Public Member Functions

- HydraController **FindController** (bool isLeft)
- HydraController **GetController** (bool isLeft, Vector3 position, Quaternion rotation)
- virtual void **StartComponent** ()
- virtual void **UpdateComponent** ()
- virtual void **ShowSkeleton** (bool shown)

Static Public Member Functions

- static HydraBaseStation **Find** (Transform parentTransform)
- static HydraBaseStation **Get** (Transform parentTransform, Vector3 position, Quaternion rotation)

Public Attributes

- bool **autoUpdate** = true
- HydraController **leftController**
- HydraController **rightController**
- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- override void Start ()
- override void Update ()

Protected Attributes

- Transform **realWorld**

Member Function Documentation

override void Passer.Tracking.HydraBaseStation.Start () `[protected], [virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

override void Passer.Tracking.HydraBaseStation.Update () `[protected], [virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPro/Runtime/Scripts/Extensions/RazerHydra/HydraBaseStation.cs

Passer.Tracking.HydraController Class Reference

Description

An Razer Hydra controller

Inherits Passer.Tracking.ControllerComponent.

Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- virtual void **StartComponent** (Transform trackerTransform, bool isLeft)
- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Public Attributes

- HydraBaseStation **baseStation**
- bool **isLeft**
- Vector3 **primaryAxis**
- Vector3 **secondaryAxis**
- float **trigger1**
- float **trigger2**
- float **button1**
- float **button2**
- float **button3**
- float **button4**
- float **option**
- float **battery**
- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- override void Start ()
  Starts the sensor

- virtual void Awake ()

Initializes the sensor.

Protected Attributes

- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- virtual bool show `[getset]`
  The render status of the sensor

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.HydraController.Start () `[protected], [virtual]`

Starts the sensor

Does nothing at this moment.

Reimplemented from Passer.Tracking.SensorComponent.

override void Passer.Tracking.HydraController.UpdateComponent () `[virtual]`

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual], [inherited]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

virtual void Passer.Tracking.SensorComponent.Awake () `[protected], [virtual], [inherited]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true `[inherited]`

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

Property Documentation

virtual bool Passer.Tracking.SensorComponent.show `[get]`, `[set]`, `[inherited]`

The render status of the sensor

When enabled, sensors with renderers attached will be rendered. When disabled, sensors will not be rendered.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPro/Runtime/Scripts/Extensions/RazerHydra/HydraController.cs

Passer.ICerebellum Interface Reference

Description

The Cerebellum interface defines the interface to control

Inherited by Passer.Humanoid.Cerebellum.Cerebellum.

Public Member Functions

- ICerebellumTarget GetTarget (sbyte jointId)
  Retrieve the target for the joint

- ICerebellumJoint GetJoint (sbyte jointId)
  Retrieve the joint

Member Function Documentation

ICerebellumTarget Passer.ICerebellum.GetTarget (sbyte *jointId*)

Retrieve the target for the joint

Parameters

jointId	The identification of the joint

Returns

ICerebellumJoint Passer.ICerebellum.GetJoint (sbyte *jointId*)

Retrieve the joint

Parameters

jointId	The identification of the joint

Returns

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Cerebellum/ICerebellum.cs

Passer.ICerebellumJoint Interface Reference

Description

The joint itself

Inherited by Passer.Humanoid.Cerebellum.Cerebellum.Joint.

Properties

- Vector3 **position** `[set]`
  The position of the joint in world space

- Vector3 **localPosition** `[get]`
  The position of the joint in the local space of the parent joint

- Quaternion **orientation** `[getset]`
  The orientation of the joint in world space

- Quaternion **localOrientation** `[getset]`
  The orientation of the joint in the local space of the parent joint

---

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Cerebellum/ICerebellum.cs

Passer.ICerebellumTarget Interface Reference

Description

A joint target which is used to calculate how the joints need to move

Inherited by Passer.Humanoid.Cerebellum.Cerebellum.Target.

Public Member Functions

- void SetPosition (Vector3 position, float confidence)
  Set the position of the target in world space with a confidence value

- void SetOrientation (Quaternion orientation, float confidence)
  Set the orientation of the target in world space with a confidence value

Properties

- Vector3 position `[getset]`
  The position of the target in world space

- Vector3 localPosition `[getset]`
  The position of the target in the local space of the parent joint

- Quaternion orientation `[getset]`
  The orientation of the target in world space

- Quaternion localOrientation `[getset]`
  The orientation fo the target in the local space of the parent joint

Member Function Documentation

void Passer.ICerebellumTarget.SetPosition (Vector3 *position*, float *confidence*)

Set the position of the target in world space with a confidence value

Parameters

position	The position of the target in world space
confidence	The confidence of the position in the range 0..1

void Passer.ICerebellumTarget.SetOrientation (Quaternion *orientation*, float *confidence*)

Set the orientation of the target in world space with a confidence value

Parameters

orientation	The orientation of the target in world space
confidence	The confidence of the orientation in the range 0..1

Property Documentation

Vector3 Passer.ICerebellumTarget.position `[get], [set]`

The position of the target in world space

The confidence will be set to 1 (maximum)

Vector3 Passer.ICerebellumTarget.localPosition `[get], [set]`

The position of the target in the local space of the parent joint

The confidence will be set to 1 (maximum)

Quaternion Passer.ICerebellumTarget.orientation `[get], [set]`

The orientation of the target in world space

The confidence will be set to 1 (maximum)

Quaternion Passer.ICerebellumTarget.localOrientation `[get], [set]`

The orientation fo the target in the local space of the parent joint

The confidence will be set to 1 (maximum)

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Cerebellum/ICerebellum.cs

Passer.Humanoid.IHandCollisionEvents Interface Reference

Description

Interface for handling touch events on objects

This interface can be used to receive touch events on objects If you implement this interface on a script which is connected to the object the On.. functions are called at the appropriate times. This enables you to implement behaviour on the object when it is touched.

Public Member Functions

- void OnHandCollisionStart (GameObject gameObject, Vector3 contactPoint)
  Function is called when the hand starts touching this object

- void OnHandCollisionEnd (GameObject gameObject)
  Function is called when the hand no longer touches this object

Member Function Documentation

void Passer.Humanoid.IHandCollisionEvents.OnHandCollisionStart (GameObject *gameObject*, Vector3 *contactPoint*)

Function is called when the hand starts touching this object

Parameters

gameObject	The gameObject the hand is touching

void Passer.Humanoid.IHandCollisionEvents.OnHandCollisionEnd (GameObject *gameObject*)

Function is called when the hand no longer touches this object

Parameters

gameObject	The gameObject the hand is touching

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Interaction/IHandCollisionEvents.cs

Passer.Humanoid.IHandGrabEvents Interface Reference

Description

Interface for handling grabbing events on objects

This interface can be used to receive grabbing events on grabbable objects If you implement this interface on a script which is connected to the grabbable object the On.. functions are called at the appropriate times. This enables you to implement behaviour on the object when it is grabbed or let go.

Public Member Functions

- void OnHandGrabbed (HandTarget handTarget)
  Function is called when the object is grabbed

- void OnHandLetGo (HandTarget handTarget)
  Function is called when the object is let go

Member Function Documentation

void Passer.Humanoid.IHandGrabEvents.OnHandGrabbed (HandTarget *handTarget*)

Function is called when the object is grabbed

Parameters

handTarget	The hand which grabbed the object

void Passer.Humanoid.IHandGrabEvents.OnHandLetGo (HandTarget *handTarget*)

Function is called when the object is let go

Parameters

handTarget	The hand which let go the object

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Interaction/IHandGrabEvents.cs

Passer.Humanoid.IHandTouchEvents Interface Reference

Description

Interface for handling touch events on objects

This interface can be used to receive touch events on objects If you implement this interface on a script which is connected to the object the On.. functions are called at the appropriate times. This enables you to implement behaviour on the object when it is touched.

Public Member Functions

- void OnHandTouchStart (HandTarget handTarget)
  Function is called when the hand starts touching this object

- void OnHandTouchEnd (HandTarget handTarget)
  Function is called when the hand no longer touches this object

Member Function Documentation

void Passer.Humanoid.IHandTouchEvents.OnHandTouchStart (HandTarget *handTarget*)

Function is called when the hand starts touching this object

Parameters

handTarget	The hand touching the object

void Passer.Humanoid.IHandTouchEvents.OnHandTouchEnd (HandTarget *handTarget*)

Function is called when the hand no longer touches this object

Parameters

handTarget	The hand which touched the object

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Interaction/IHandTouchEvents.cs

Passer.Humanoid.IHandTriggerEvents Interface Reference

Description

Interface for handling touch events on objects

This interface can be used to receive touch events on objects If you implement this interface on a script which is connected to the object the On.. functions are called at the appropriate times. This enables you to implement behaviour on the object when it is touched.

Public Member Functions

- void OnHandTriggerEnter (HandTarget handTarget, Collider collider)
  Function is called when the hand starts touching this object

- void OnHandTriggerExit (HandTarget handTarget, Collider collider)
  Function is called when the hand no longer touches this object

Member Function Documentation

void Passer.Humanoid.IHandTriggerEvents.OnHandTriggerEnter (HandTarget *handTarget*, Collider *collider*)

Function is called when the hand starts touching this object

Parameters

handTarget	The hand which touches the object

void Passer.Humanoid.IHandTriggerEvents.OnHandTriggerExit (HandTarget *handTarget*, Collider *collider*)

Function is called when the hand no longer touches this object

Parameters

handTarget	The hand which touched the object

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Interaction/IHandTriggerEvents.cs

Passer.Humanoid.IHumanoidNetworking Interface Reference

Description

Interface for Humanoid Networking functions

Inherited by Passer.Humanoid.HumanoidPlayer, Passer.Humanoid.HumanoidPlayer, and Passer.Humanoid.HumanoidPlayer.

Public Member Functions

- void **Send** (bool b)
- void **Send** (byte b)
- void **Send** (int x)
- void **Send** (float f)
- void **Send** (Vector3 v)
- void **Send** (Quaternion q)
- bool **ReceiveBool** ()
- byte **ReceiveByte** ()
- int **ReceiveInt** ()
- float **ReceiveFloat** ()
- Vector3 **ReceiveVector3** ()
- Quaternion **ReceiveQuaternion** ()
- ulong **GetObjectIdentity** (GameObject obj)
- GameObject **GetGameObject** (ulong objIdentity)
- void **InstantiateHumanoid** (HumanoidControl humanoid)
- void **DestroyHumanoid** (HumanoidControl humanoid)
- void **UpdateHumanoidPose** (HumanoidControl humanoid)
- void **Grab** (HandTarget handTarget, GameObject obj, bool rangeCheck, HandTarget.GrabType grabType=HandTarget.GrabType.HandGrab)
- void **LetGo** (HandTarget handTarget)
- void **ChangeAvatar** (HumanoidControl humanoid, string remoteAvatarName, string possessionLocation=null)
- void **SyncTrackingSpace** (HumanoidControl humanoid)
- void **DebugLog** (string s)
- void **DebugWarning** (string s)
- void **DebugError** (string s)
- void **ReenableNetworkSync** (GameObject obj)
- void **DisableNetworkSync** (GameObject obj)

Properties

- float **sendRate** `[get]`
- HumanoidNetworking.DebugLevel **debug** `[get]`
- HumanoidNetworking.Smoothing **smoothing** `[get]`
- bool **createLocalRemotes** `[getset]`
- bool **isLocal** `[get]`
- ulong **nwId** `[get]`
- bool **syncFingerSwing** `[get]`
- bool **syncTracking** `[getset]`
- bool **fuseTracking** `[get]`
- List< HumanoidControl > **humanoids** `[get]`
- HumanoidNetworking.HumanoidPose **lastHumanoidPose** `[getset]`

The documentation for this interface was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Networking/HumanoidNetworking.cs

Passer.InteractionEventHandler Class Reference

Description

Implements behaviour using interaction

Inherits MonoBehaviour.

Public Attributes

- BoolEventHandlers focusHandlers
- BoolEventHandlers clickHandlers

Protected Member Functions

- virtual void **Update** ()

Member Data Documentation

BoolEventHandlers Passer.InteractionEventHandler.focusHandlers

```
Initial value:= new BoolEventHandlers() {
            id = 0,
            label = "Focus Event",
            tooltip =
                "Call functions using the interaction focus\n" +
                "Parameter: the object having focus",
            eventTypeLabels = new string[] {
                "Never",
                "On get focus",
                "On lost focus",
                "While in focus",
                "While not in focus",
                "On focus change",
                "Always"
            },
        }
```

BoolEventHandlers Passer.InteractionEventHandler.clickHandlers

```
Initial value:= new BoolEventHandlers() {
            id = 1,
            label = "Click Event",
            tooltip =
                "Call functions using the interaction click\n" +
                "Parameter: the click state",
            eventTypeLabels = new string[] {
                "Never",
                "On click down",
                "On click up",
                "While clicking",
                "While not clicking",
                "On click change",
                "Always"
            },
        }
```

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/InteractionEventHandler.cs

Passer.InteractionPointer Class Reference

Description

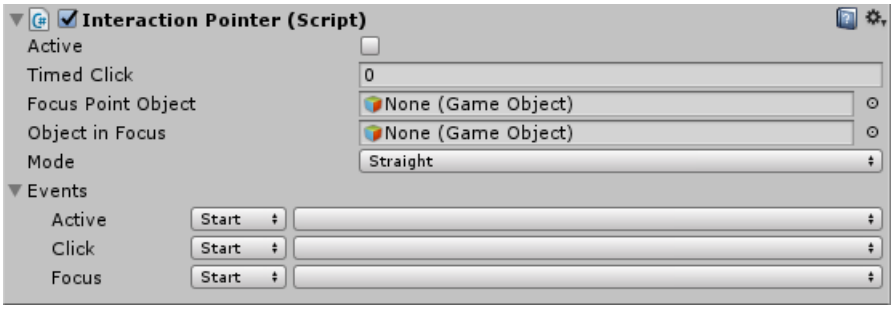
A generic pointer to interact with objects in the scene using the Unity Event system.

The interaction pointer is a generic pointer to interact with objects in the scene using the Unity Event system or InteractionEvents. The objects can receive and react on these interactions when an Unity Event Trigger component has been added to them. The 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



- Active
- Timed Click
- Focus Point Object
- Object in Focus
- Mode

Straight Cast

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

Bezier Cast

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 in the curve. Higher gives a smoother curve but requires more performance.

Gravity Cast

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 Cast

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 case.
- Radius: the radius of the sphere.

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

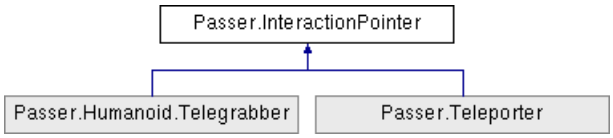
- Active Event
- Click Event
- Focus Event

Event System Compatibility

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

Inheritance diagram for Passer.InteractionPointer:



Public Types

- enum RayType { **Straight**, **Bezier**, **Gravity**, **SphereCast** }
  The ray modes for interaction pointers.
- enum PointerType { **FocusPoint**, **Ray** }
  The type of interaction pointer

Public Member Functions

- void **PointTo** (GameObject obj)
- void **SetDirection** (Vector3 lookDirection)
- void **LaunchRigidbody** (Rigidbody rigidbody)
- void **LaunchPrefab** (GameObject prefab)
- void SpawnOnObjectInFocus (GameObject prefab)
  Spawns a prefab as a child of the objectInFocus

- void **ApplyForce** (float magnitude)
- void FullClick ()
  Click on the objectInFocus

- virtual void Click (bool clicking=true)
  Change the click status on the objectInFocus

- virtual void Activation (bool _active)
  Change the active status of the InteractionPointer

- virtual void **ActivateClick** (bool _active)

Static Public Member Functions

- static InteractionPointer Add (Transform parentTransform, PointerType pointerType=PointerType.Ray)
  Adds a default InteractionPointer to the transform

Public Attributes

- bool active = true
  Is the interaction pointer active?

- float timedClick
  Automatically initiates a click when the objectInFocus does not change for the set amount of seconds.

- GameObject focusPointObj
  The GameObject which represents the focus point of the Interaction Pointer when it is active.

- GameObject objectInFocus
  The object to which the Interaction Pointer is pointing at.

- float focusDistance
  Distance to the objectInFocus

- RayType **rayType** = RayType.Straight
  The ray mode for this interaction pointer.

- float maxDistance = 10
  The maximum length of the curve in units(meters)

- float resolution = 0.2F
  The size of a segment in the curve

- float speed = 3
  The horizontal speed for a gravity curve.

- float **radius** = 0.1F
  The radius of the sphere in a SphereCast

- GameObjectEventHandlers focusEvent
  Event based on the current focus.

- Vector3EventList focusPointEvent
  Event based on the current focus.

- GameObjectEventHandlers clickEvent
  Event based on the clicking status

- BoolEvent activeEvent = new BoolEvent()
  Event based on the active status

Protected Member Functions

- virtual void **Awake** ()
- virtual InteractionModule **CreateInteractionModule** ()
- virtual EventSystem **CreateEventSystem** ()
- virtual void **Start** ()
- virtual void **Update** ()
- virtual void **UpdateStraight** ()
- virtual void **UpdateBezier** ()
- virtual Vector3[] **UpdateBezierCurve** (Transform transform, float maxDistance, out Vector3 normal, out GameObject focusObject)
- Vector3 **GetPoint** (float t, Transform transform)
- Vector3 **GetVelocity** (float t, Transform transform)
- virtual void **UpdateGravity** ()
- virtual void **UpdateGravityCurve** (Transform transform, float forwardSpeed, out Vector3 normal, out GameObject hitObject)
- virtual void **UpdateSpherecast** ()

- void **UpdateFocus** ()
- void **UpdateFocusPoint** ()
- virtual void **OnDrawGizmosSelected** ()

Static Protected Member Functions

- static void **DebugLog** (string s)

Protected Attributes

- bool **hasClicked** = false
- InteractionModule **interactionModule**
- int **interactionID**
- InteractionModule.InteractionPointer **interactionPointer**
- LineRenderer **lineRenderer**
  The LineRender for this pointer when available.

- float **focusTimeToTouch** = 0
- float **focusStart** = 0
- float **heightLimitAngle** = 100f
- Vector3 **startPosition** = Vector3.zero
- Vector3 **intermediatePosition**
- Vector3 **endPosition**
- GameObject **previousObjectInFocus**

Member Function Documentation

static InteractionPointer Passer.InteractionPointer.Add (Transform *parentTransform*, PointerType *pointerType* = `PointerType.Ray`)`[static]`

Adds a default InteractionPointer to the transform

Parameters

parentTransform	The transform to which the Teleporter will be added
pointerType	The interaction pointer type for the Teleporter.

void Passer.InteractionPointer.SpawnOnObjectInFocus (GameObject *prefab*)

Spawns a prefab as a child of the objectInFocus

Parameters

prefab	The prefab to spawn

The spawn position and rotation will match the focusPoint transform. When the objectInFocus is null, the prefab will be spawned as a root transform.

void Passer.InteractionPointer.FullClick ()

Click on the objectInFocus

This function will do a full click: a button down followed by a button up.

virtual void Passer.InteractionPointer.Click (bool *clicking* = `true`) `[virtual]`

Change the click status on the objectInFocus

Parameters

clicking	Indicates if the button is down

Reimplemented in Passer.Teleporter.

virtual void Passer.InteractionPointer.Activation (bool *_active*) `[virtual]`

Change the active status of the InteractionPointer

Parameters

_active	Indicates if the InteractionPointer is active

Member Data Documentation

bool Passer.InteractionPointer.active = true

Is the interaction pointer active?

When an interaction pointer is active, it will actively point to objects. The objectInFocus will be updated based on the pointer. When a focusPointObj is set, this object will be set active. When a Line Renderer is set, the line renderer will be updated according to the properties of the pointer.

float Passer.InteractionPointer.timedClick

Automatically initiates a click when the objectInFocus does not change for the set amount of seconds.

The value 0 disables this function.

GameObject Passer.InteractionPointer.focusPointObj

The GameObject which represents the focus point of the Interaction Pointer when it is active.

If this is set and the pointer is active, the object will be objected based on the pointer's properties This GameObject will be disabled when the pointer is not active.

GameObject Passer.InteractionPointer.objectInFocus

The object to which the Interaction Pointer is pointing at.

This is updated at runtime while the pointer is active. The value is null when the pointer is not reaching any object.

float Passer.InteractionPointer.focusDistance

Distance to the objectInFocus

This is float.infintiy when no object is in focus

float Passer.InteractionPointer.maxDistance = 10

The maximum length of the curve in units(meters)

This value is used for Straight, SphereCast and Bezier RayTypes

float Passer.InteractionPointer.resolution = 0.2F

The size of a segment in the curve

Lower values will give a smoother curve, but requires more performance This value is used for Bezier and Gravity RayTypes

float Passer.InteractionPointer.speed = 3

The horizontal speed for a gravity curve.

A higher speed will reach further positions.

GameObjectEventHandlers Passer.InteractionPointer.focusEvent

```
Initial value:= new GameObjectEventHandlers() {
            label = "Focus Event",
            tooltip =
                "Call functions using the current focus\n" +
                "Parameter: the Object in Focus"
        }
```

Event based on the current focus.

This event is generated from the objectInFocus. It has the objectInFocus as parameter. It can be used to execute functions when the focus changes between objects.

Vector3EventList Passer.InteractionPointer.focusPointEvent

```
Initial value:= new Vector3EventList() {
            label = "Focus Point Event",
            tooltip =
                "Call functions using the current focus point\n" +
                "Parameter: the position of the focus point"
        }
```

Event based on the current focus.

This event is generated from the objectInFocus. It has the objectInFocus as parameter. It can be used to execute functions when the focus changes between objects.

GameObjectEventHandlers Passer.InteractionPointer.clickEvent

```
Initial value:= new GameObjectEventHandlers() {
            label = "Click Event",
            tooltip =
                "Call functions using the clicking status\n" +
                "Parameter: the Object in Focus when clicking"
        }
```

Event based on the clicking status

This event is generated from the clicking boolean. It has the objectInFocus as parameter. It can be used to exectue functions when the user has clicked on an object.

BoolEvent Passer.InteractionPointer.activeEvent = new BoolEvent()

Event based on the active status

This event is genereated from the active boolean Is has the active boolean as parameter. It can be used to execute functions when the interaction pointer is activated.

The documentation for this class was generated from the following file:

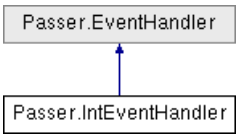
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/InteractionPointer.cs

Passer.IntEventHandler Class Reference

Description

An event handler calling a function with an integer parameter

Inheritance diagram for Passer.IntEventHandler:



Public Types

- enum Type { Never, OnStart, OnEnd, WhileActive, WhileInactive, OnChange, Continuous }
  The different types of events when the function is called
- enum OverrideMode { Prepend, Append, Replace }

Public Member Functions

- **IntEventHandler** (Type eventType)
- virtual void **Update** ()

Public Attributes

- int **minValue**
- int **maxValue**
- int **intTriggerLow** = 0
- int **intTriggerHigh** = 1
- int **multiplicationFactor** = 1
- Type **eventType** = Type.Continuous
  The event type for the function call

- bool **eventNetworking** = false
  For future use :-)

- FunctionCall **functionCall**
  The function to call

- bool **boolInverse** = false
  Negate the boolean state before calling event trigger

- OverrideMode **overrideMode**

Protected Member Functions

- override void UpdateVoid ()
- override void UpdateBool ()
- override void UpdateInt ()
- override void UpdateFloat ()
- void **UpdateAnimationParameter** ()
- virtual void **UpdateString** ()
- virtual void **UpdateString** (string s)
- virtual void **UpdateVector3** ()
- virtual void **UpdateGameObject** ()

- virtual void **UpdateRigidbody** ()
- virtual void **UpdateStringBool** (string s)
- virtual void **UpdateStringFloat** (string s)
- virtual void **UpdateStringInt** (string s)
- bool **CheckCondition** (bool active, bool changed, bool valueChanged)

Protected Attributes

- bool **initialized**
- bool **_boolValue**
- bool **boolChanged** = true
- int **_intValue**
- bool **intChanged**
- float **_floatValue**
- bool **floatChanged**

Properties

- virtual int? **value** `[getset]`
- virtual bool **boolValue** `[getset]`
- bool isDead `[get]`
  True when the eventHandler is dead and can be removed

Member Enumeration Documentation

enum Passer.EventHandler.Type `[inherited]`

The different types of events when the function is called

Enumerator:

Never	The function is never called.
OnStart	The function is called when the event starts.
OnEnd	The function is called when the event ends.
WhileActive	The function is called every frame while the event is active.
WhileInactive	The function is called every frame while the event is not active.
OnChange	The function is called every time the event changes.
Continuous	The functions is called every frame.

enum Passer.EventHandler.OverrideMode `[inherited]`

Enumerator:

Prepend	Prepend this handler before existing handlers.

Append	Append this handler after existing handlers.
Replace	Replace the topmost handler with this handler.

Member Function Documentation

override void Passer.IntEventHandler.UpdateVoid () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.IntEventHandler.UpdateBool () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.IntEventHandler.UpdateInt () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

override void Passer.IntEventHandler.UpdateFloat () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

Property Documentation

bool Passer.EventHandler.isDead `[get], [inherited]`

True when the eventHandler is dead and can be removed

A function is dead when it does nothing. This is when the functionCall is not defined or when the target of the functionCall is empty

The documentation for this class was generated from the following file:

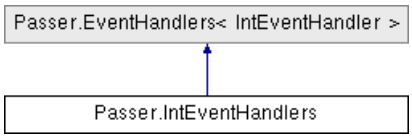
- Assets/Passer/HumanoidControl/Runtime/Tools/Events/IntEvent.cs

Passer.IntEventHandlers Class Reference

Description

A list of event handlers with an integer parameter

Inheritance diagram for Passer.IntEventHandlers:



Public Attributes

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events**
  The EventHandlers

Properties

- int **value** `[getset]`

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/IntEvent.cs

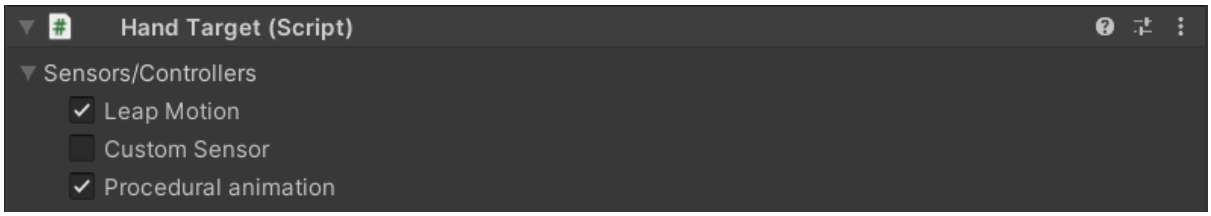
Passer.Humanoid.LeapHand Class Reference

Description

Leap Motion Hand to on the arm of a Humanoid

Unity Inspector

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.



Inherits Passer.Humanoid.Tracking.ArmSensor.

Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **CheckSensor** (HandTarget handTarget)
- override void **Start** (HumanoidControl _humanoid, Transform targetTransform)
- override void Update ()
  Update the sensor state

Static Public Member Functions

- static Rotation **CalculateUpperArmOrientation** (Vector upperArmPosition, float upperArmLength, Vector forearmUp, float forearmLength, Vector handPosition, bool isLeft)
- static Rotation **CalculateArmOrientation** (Vector joint1Position, Vector joint1Up, Vector joint2Position, bool isLeft)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- bool **isLeft**
- TargetData **shoulder**
- TargetData **upperArm**
- TargetData **forearm**
- TargetData **hand**
- Finger **thumb**
- Finger **indexFinger**

- Finger **middleFinger**
- Finger **ringFinger**
- Finger **littleFinger**
- Finger[] **fingers**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- virtual void **UpdateHand** ()
- void **UpdateSensor** ()

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- LeapTracker **leapTracker** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.LeapHand.Update () `[virtual]`

Update the sensor state

Returns

Status of the sensor after the update

Reimplemented from Passer.Humanoid.Tracking.Sensor.

The documentation for this class was generated from the following file:

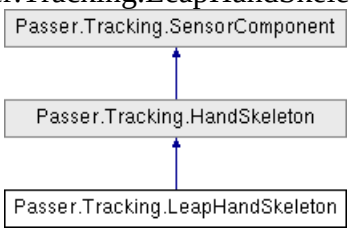
- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/LeapMotion/LeapHand.cs

Passer.Tracking.LeapHandSkeleton Class Reference

Description

A hand tracked by LeapMotion

Inheritance diagram for Passer.Tracking.LeapHandSkeleton:



Public Types

- enum **BoneId** { **Invalid** = -1, **Hand** = 0, **ThumbProximal**, **ThumbIntermediate**, **ThumbDistal**, **ThumbTip**, **IndexMetaCarpal**, **IndexProximal**, **IndexIntermediate**, **IndexDistal**, **IndexTip**, **MiddleMetaCarpal**, **MiddleProximal**, **MiddleIntermediate**, **MiddleDistal**, **MiddleTip**, **RingMetaCarpal**, **RingProximal**, **RingIntermediate**, **RingDistal**, **RingTip**, **LittleMetacarpal**, **LittleProximal**, **LittleIntermediate**, **LittleDistal**, **LittleTip**, **Forearm**, **Count** }

Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- override Transform GetBoneTransform (Finger finger, FingerBone fingerBone)
  Gets the transform of the tracked bone

- virtual GameObject **CreateHandSkeleton** (Transform trackerTransform, bool isLeft, bool showRealObjects)
- virtual Transform **GetForearmBone** ()
- virtual Transform **GetWristBone** ()
- virtual int **GetBoneId** (Finger finger, FingerBone fingerBone)
- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Static Public Member Functions

- static LeapHandSkeleton **Find** (Transform trackerTransform, bool isLeft)
- static LeapHandSkeleton **Get** (Transform trackerTransform, bool isLeft)
- static HandSkeleton **FindHandSkeleton** (Transform trackerTransform, bool isLeft)

Public Attributes

- bool **isLeft**
  True: this is a left hand. Fale: this is a right hand

- new bool **show** = false
  Determines whether this skeleton should be rendered

- Tracker.Status **status**

The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- override void InitializeSkeleton ()
  This function is used to initialize the tracked bones

- virtual void **OnDestroy** ()
- void **UpdateSkeletonBones** (Leap.Hand leapHand)
- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()
- virtual void Awake ()
  Initializes the sensor.

- virtual void Start ()
  Starts the sensor

Protected Attributes

- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.LeapHandSkeleton.InitializeSkeleton () [protected], [virtual]

This function is used to initialize the tracked bones

Reimplemented from Passer.Tracking.HandSkeleton.

override void Passer.Tracking.LeapHandSkeleton.UpdateComponent () [virtual]

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

override Transform Passer.Tracking.LeapHandSkeleton.GetBoneTransform (Finger *finger*, FingerBone *fingerBone*) [virtual]

Gets the transform of the tracked bone

Parameters

finger	The requested finger
fingerBone	The requested bone in the finger

Returns

The tracked bon transform of *null* if it does not exist

Reimplemented from Passer.Tracking.HandSkeleton.

virtual void Passer.Tracking.SensorComponent.Awake () [protected], [virtual], [inherited]

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.Start () [protected], [virtual], [inherited]

Starts the sensor

Does nothing at this moment.

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, and Passer.Tracking.HydraController.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) [virtual], [inherited]

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true [inherited]

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

The documentation for this class was generated from the following file:

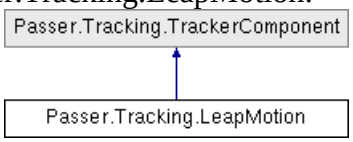
- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/LeapMotion/LeapHandSkeleton.cs

Passer.Tracking.LeapMotion Class Reference

Description

A representation of the real-world LeapMotion camera

Inheritance diagram for Passer.Tracking.LeapMotion:



Public Member Functions

- virtual void **ShowSkeleton** (bool shown)

Static Public Member Functions

- static LeapMotion Find (Transform parentTransform)
  Find a Leap Motion tracker

- static LeapMotion **Get** (Transform parentTransform, Vector3 localPosition, Quaternion localRotation)
  Create a new Leap Motion Tracker

Public Attributes

- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- LeapHandSkeleton **FindSkeleton** (bool isLeft)
- override void Update ()
- virtual void **Start** ()

Static Protected Member Functions

- static void **AddModel** (Transform trackerTransform)

Protected Attributes

- LeapHandSkeleton **_leftSkeleton**
- LeapHandSkeleton **_rightSkeleton**
- Transform **realWorld**

Properties

- LeapHandSkeleton **leftSkeleton** `[get]`
- LeapHandSkeleton **rightSkeleton** `[get]`

Member Function Documentation

static LeapMotion Passer.Tracking.LeapMotion.Find (Transform *parentTransform*) `[static]`

Find a Leap Motion tracker

Parameters

parentTransform	The parent transform of the tracker

Returns

The tracker

override void Passer.Tracking.LeapMotion.Update () `[protected]`, `[virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/LeapMotion/LeapMotion.cs

Passer.Humanoid.LeapTracker Class Reference

Description

Leap Motion enables 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

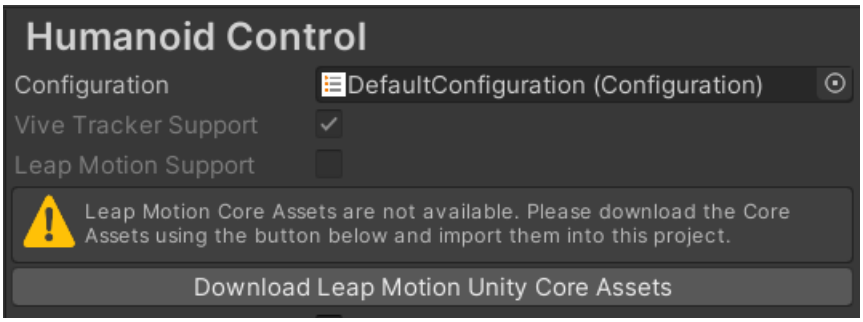
Leap Motion is supported on Microsoft Windows 10.

Runtime

Leap Motion software version 4 or higher is required

Setup

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

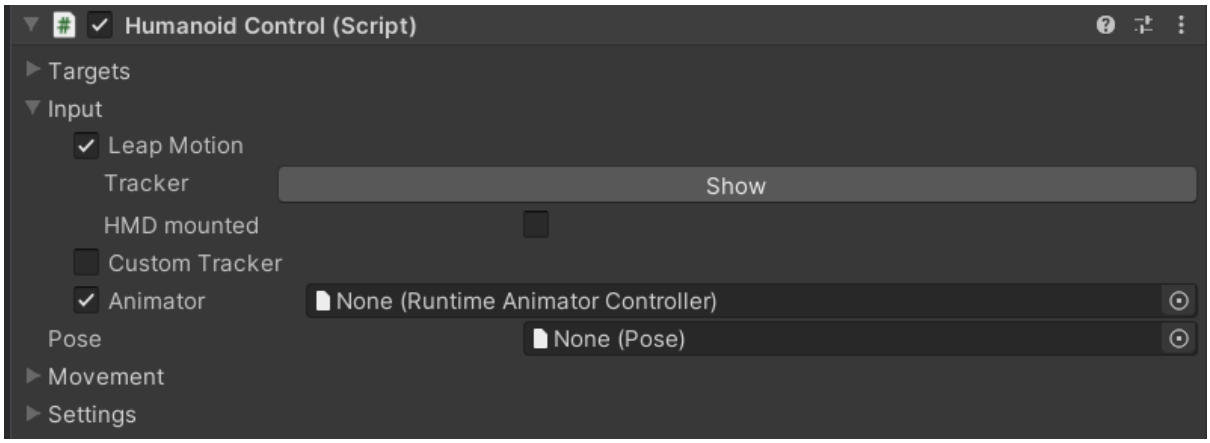


Click the button to go to the download page for the Leap Motion assets. From the SDK, you only need to import the *core.unitypackage* .

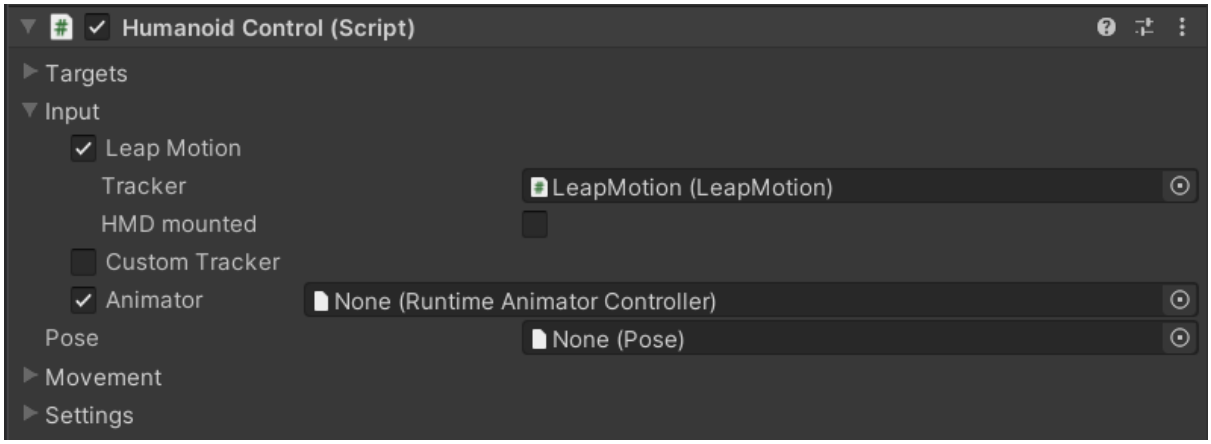
After the assets have been imported, *Leap Motion Support* will be enabled automatically in the preferences.

Inspector

To enable tracking with Leap Motion for a humanoid, *Leap Motion* needs to be enabled in the HumanoidControl Component:



By default the LeapMotion object is not visible in the scene, but it will be created automatically when the scene starts. If the button *Show* is pressed, the Leap Motion object will be created in the *Real World* object.

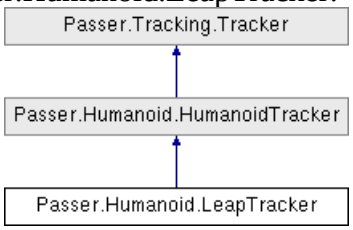


Leap Motion (LeapMotion) is a reference to the object in the scene representing the Leap Motion Sensor. This GameObject is found as a child of the *Real World* GameObject. It is only rendered in the scene when HumanoidControl->Settings->Show Real Objects has been enabled. The LeapMotion GameObject can be used to set the position of the tracking relative to the player in the scene.

See also

- Hand Target: LeapHand
- Passer::Tracking::LeapMotion
- Passer::Tracking::LeapHandSkeleton

Inheritance diagram for Passer.Humanoid.LeapTracker:



Public Types

- enum Status { Unavailable, Present, Tracking }
  The tracking status

Public Member Functions

- override void CheckTracker (HumanoidControl humanoid)
  Check the presence of a LeapMotion object and creates one if it does not exist

- void CheckTrackerOnHmd (HumanoidControl humanoid, TrackerGetter getTracker, Vector3 localPosition, Quaternion localRotation)
  Function to check the status of a specific tracker

- override void StartTracker (HumanoidControl _humanoid)
  Start the tracker

- override void StopTracker ()
  Stop the tracker

- override void UpdateTracker ()
  Update the tracker state

- void **PlaceTrackerTransform** (bool isHeadMounted)
- void **SetTrackerToTarget** ()
- void **UpdateTrackerFromTarget** (bool isHeadMounted)
- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker)
  Function to check the status of a specific tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker, Vector3 localPosition, Quaternion localRotation)
  Function to check the status of a specific tracker

- delegate TrackerComponent TrackerGetter (Transform transform, Vector3 localPosition, Quaternion localRotation)
  Function delegate for retrieving the tracker

- virtual Vector3 **GetBonePosition** (uint actorId, FacialBone boneId)
- virtual Quaternion **GetBoneRotation** (uint actorId, FacialBone boneId)
- virtual float **GetBoneConfidence** (uint actorId, FacialBone boneId)
- virtual void StartTracker ()
  Optional list of SubTrackers

- virtual void ShowTracker (bool shown)
  Show or hide the Tracker renderers

- virtual void Calibrate ()
  Calibrate the tracker

- virtual void AdjustTracking (Vector3 positionalDelta, Quaternion rotationalDelta)
  Adjust the position of the tracker by the given delat

Public Attributes

- LeapDevice **device**
- bool isHeadMounted = true
  Is the LeapMotion mounted on an HMD?

- HumanoidControl **humanoid**
  The humanoid for this tracker

- bool **enabled**
  Is this tracker enabled?

- Status **status**
  The tracking Status of the tracker

- TrackerComponent **trackerComponent**
  The tracking device

Protected Attributes

- bool **useLeapPackage** = false
- Vector3 **headTrackerPosition**
- Quaternion **headTrackerRotation**

Properties

- override string name `[get]`
  The name of this tracker

- override ArmSensor leftHandSensor `[get]`
  Get the sensor for the left hand

- override ArmSensor rightHandSensor `[get]`
  Get the sensor for the right hand

- override HumanoidSensor[] **sensors** `[get]`
- virtual HeadSensor headSensor `[get]`
  Get the sensor for the head

- virtual TorsoSensor hipsSensor `[get]`
  Get the sensor for the hips

- virtual LegSensor leftFootSensor `[get]`
  Get the sensor for the left foot

- virtual LegSensor rightFootSensor `[get]`
  Get the sensor for the right foot

Member Enumeration Documentation

enum Passer.Tracking.Tracker.Status [inherited]

The tracking status

Enumerator:

Unavailable	The tracking device is not available.
Present	The tracking device is available but not tracking.
Tracking	The tracking device is actively tracking.

Member Function Documentation

override void Passer.Humanoid.LeapTracker.CheckTracker (HumanoidControl *humanoid*) [virtual]

Check the presence of a LeapMotion object and creates one if it does not exist

Parameters

humanoid	The humanoid for which the tracker needs to be checked

Reimplemented from Passer.Humanoid.HumanoidTracker.

void Passer.Humanoid.LeapTracker.CheckTrackerOnHmd (HumanoidControl *humanoid*, TrackerGetter *getTracker*, Vector3 *localPosition*, Quaternion *localRotation*)

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

override void Passer.Humanoid.LeapTracker.StartTracker (HumanoidControl *_humanoid*) [virtual]

Start the tracker

Reimplemented from Passer.Humanoid.HumanoidTracker.

override void Passer.Humanoid.LeapTracker.StopTracker () [virtual]

Stop the tracker

Reimplemented from Passer.Tracking.Tracker.

override void Passer.Humanoid.LeapTracker.UpdateTracker () [virtual]

Update the tracker state

Reimplemented from Passer.Tracking.Tracker.

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*) [inherited]

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker

The default position/rotation for the tracker when created will be zero

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*, Vector3 *localPosition*, Quaternion *localRotation*) [inherited]

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

delegate TrackerComponent Passer.Humanoid.HumanoidTracker.TrackerGetter (Transform *transform*, Vector3 *localPosition*, Quaternion *localRotation*) [inherited]

Function delegate for retrieving the tracker

Parameters

transform	The root transform to start the searching of the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

Returns

The tracker component found or created

The default position/rotation is relative to the humanoid's real world.

virtual void Passer.Tracking.Tracker.StartTracker () [virtual], [inherited]

Optional list of SubTrackers

Start the tracker

virtual void Passer.Tracking.Tracker.ShowTracker (bool *shown*) [virtual], [inherited]

Show or hide the Tracker renderers

Parameters

shown	Renderers are enabled when shown == true

virtual void Passer.Tracking.Tracker.Calibrate () [virtual], [inherited]

Calibrate the tracker

Reimplemented in Passer.Humanoid.UnityXRTracker.

virtual void Passer.Tracking.Tracker.AdjustTracking (Vector3 *positionalDelta*, Quaternion *rotationalDelta*) [virtual], [inherited]

Adjust the position of the tracker by the given delat

Parameters

positionalDelta	The positional delta to apply
rotationalDelta	The rotational delta to apply

Member Data Documentation

bool Passer.Humanoid.LeapTracker.isHeadMounted = true

Is the LeapMotion mounted on an HMD?

When enabled, the LeapMotion device will follow the movement of the hmd. And hand tracking will be relative to the hmd.

Property Documentation

override string Passer.Humanoid.LeapTracker.name [get]

The name of this tracker

override ArmSensor Passer.Humanoid.LeapTracker.leftHandSensor [get]

Get the sensor for the left hand

Will return null when this sensor is not present

override ArmSensor Passer.Humanoid.LeapTracker.rightHandSensor [get]

Get the sensor for the right hand

Will return null when this sensor is not present

virtual HeadSensor Passer.Humanoid.HumanoidTracker.headSensor `[get]`, `[inherited]`

Get the sensor for the head

Will return null when this sensor is not present

virtual TorsoSensor Passer.Humanoid.HumanoidTracker.hipsSensor `[get]`, `[inherited]`

Get the sensor for the hips

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.leftFootSensor `[get]`, `[inherited]`

Get the sensor for the left foot

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.rightFootSensor `[get]`, `[inherited]`

Get the sensor for the right foot

Will return null when this sensor is not present

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/LeapMotion/LeapTracker.cs

Passer.MechanicalJoint Class Reference

Description

Mechanical Joints can be used to limit the movements of a Kinematic Rigidbody in local space.

Inherits MonoBehaviour.

Public Types

- enum **RotationMethod** { **AngleDifference**, **PositionDifference** }

Public Member Functions

- void LerpX (float f)
  Linearly interpolates between min and maxLocalPostion.x

- void LerpY (float f)
  Linearly interpolates between min and maxLocalPostion.y

- void LerpZ (float f)
  Linearly interpolates between min and maxLocalPostion.z

- Vector3 **GetCorrectionVector** ()
- Quaternion **GetCorrectionRotation** ()
- Quaternion **GetCorrectionAxisRotation** ()
- Quaternion **GetCorrectionAxisRotation_old** ()
- Quaternion **GetCorrectionAxisRotation2** ()
- void **MoveForward** (float speedZ)
- void **MoveSideward** (float speedX)
- void **MoveUpward** (float speedY)
- void **SetSpeed** (Vector3 speed)
- void **Rotate** (float speed)

Public Attributes

- Transform **parent**
- bool limitX = true
  When true, the local X-position is limited

- bool limitY = true
  When true, the local Y-position is limited

- bool limitZ = true
  When true, the local Z-position is limited

- Vector3 basePosition
  The base position for the bounds

- Vector3 minLocalPosition

The minimum local position.

- Vector3 maxLocalPosition
  The maximum local position.

- Quaternion **baseRotation** = Quaternion.identity
  The base rotation for the bounds

- bool limitAngle = true
  When the the local rotation is limited

- float minLocalAngle
  The maximum angle around the limitAngleAxis

- float maxLocalAngle
  The maximum angle around the limitAngleAxis

- Vector3 **limitAngleAxis** = Vector3.up
  The axis around which the Rigidbody can rotate

- RotationMethod **rotationMethod**
- GameObjectEventHandlers gameObjectEvent
- FloatEventHandlers xSliderEvents
- FloatEventHandlers ySliderEvents
- FloatEventHandlers zSliderEvents
- FloatEventHandlers angleEvents

Protected Member Functions

- Vector3 **GetLocalPosition** ()
- Vector3 **GetWorldPosition** (Vector3 localPosition)
- virtual void **Awake** ()
- virtual void **Start** ()
- virtual void **FixedUpdate** ()
- virtual void **OnDestroy** ()
- void **UpdateEvents** ()
- virtual void **OnDrawGizmosSelected** ()
- Vector3 **ProjectPointOnPlane** (Vector3 point, Vector3 planeOrigin, Vector3 planeNormal)

Protected Attributes

- Rigidbody **rb**
- Transform **t**
- float **xValue**
- float **yValue**
- float **zValue**
- float **angleValue**
- Vector3 **speed**
- float **rotationSpeed**

Static Protected Attributes

- static string[] sliderEventTypeLabels

Member Function Documentation

void Passer.MechanicalJoint.LerpX (float *f*)

Linearly interpolates between min and maxLocalPostion.x

0 matches the minLocalPosition on the X axis 1 matches the maxLocalPosition on the X axis

void Passer.MechanicalJoint.LerpY (float *f*)

Linearly interpolates between min and maxLocalPostion.y

0 matches the minLocalPosition on the Y axis 1 matches the maxLocalPosition on the Y axis

void Passer.MechanicalJoint.LerpZ (float *f*)

Linearly interpolates between min and maxLocalPostion.z

0 matches the minLocalPosition on the Z axis 1 matches the maxLocalPosition on the Z axis

---

Member Data Documentation

bool Passer.MechanicalJoint.limitX = true

When true, the local X-position is limited

The limits are set by the X member of the minLocalPosition and maxLocalPosition When the bounds are both zero, the local X position is locked.

bool Passer.MechanicalJoint.limitY = true

When true, the local Y-position is limited

The limits are set by the Y member of the minLocalPosition and maxLocalPosition When the bounds are both zero, the local Y position is locked.

bool Passer.MechanicalJoint.limitZ = true

When true, the local Z-position is limited

The limits are set by the Z member of the minLocalPosition and maxLocalPosition When the bounds are both zero, the local Z position is locked.

Vector3 Passer.MechanicalJoint.basePosition

The base position for the bounds

The minLocalPosition and maxLocalPosition are relative to the basePosition.

Vector3 Passer.MechanicalJoint.minLocalPosition

The minimum local position.

This localPosition is relative to the basePosition.

Vector3 Passer.MechanicalJoint.maxLocalPosition

The maximum local position.

This localPosition is relative to the basePosition.

bool Passer.MechanicalJoint.limitAngle = true

When the the local rotation is limited

The limitation is determined by a maxLocalAngle around the limitAnglesAxis. With this limitation, the rigidbody can only rotated around the limmitAngleAxis.

float Passer.MechanicalJoint.minLocalAngle

The maximum angle around the limitAngleAxis

When this is zero, the local rotation is locked.

float Passer.MechanicalJoint.maxLocalAngle

The maximum angle around the limitAngleAxis

When this is zero, the local rotation is locked.

GameObjectEventHandlers Passer.MechanicalJoint.gameObjectEvent

```
Initial value:= new GameObjectEventHandlers() {
            label = "GameObject Event",
            id = 0,
            tooltip = "Call functions based on the GameObject life cycle",
            eventTypeLabels = new string[] {
                "Never",
                "Start",
                "On Destroy",
                "Update",
                " ",
                " ",
                " "
            }
        }
```

string [] Passer.MechanicalJoint.sliderEventTypeLabels `[static], [protected]`

```
Initial value:= new string[] {
                "Never",
                "On Min",
                "On Max",
                "While Min",
                "While Max",
                "On Change",
                "Continuous"
        }
```

FloatEventHandlers Passer.MechanicalJoint.xSliderEvents

```
Initial value:= new FloatEventHandlers() {
            label = "X Axis",
            id = 1,
```

```
            eventTypeLabels = sliderEventTypeLabels,
            tooltip =
                "Call function using the X axis range value\n" +
                "Parameter: the range along the X axis (-1..1)"
        }
```

FloatEventHandlers Passer.MechanicalJoint.ySliderEvents

```
Initial value:= new FloatEventHandlers() {
            label = "Y Axis",
            id = 2,
            eventTypeLabels = sliderEventTypeLabels,
            tooltip =
                "Call function using the Y axis range value\n" +
                "Parameter: the range along the Y axis (-1..1)"
        }
```

FloatEventHandlers Passer.MechanicalJoint.zSliderEvents

```
Initial value:= new FloatEventHandlers() {
            label = "Z Axis",
            id = 3,
            eventTypeLabels = sliderEventTypeLabels,
            tooltip =
                "Call function using the Z axis range value\n" +
                "Parameter: the range along the Z axis (-1..1)"
        }
```

FloatEventHandlers Passer.MechanicalJoint.angleEvents

```
Initial value:= new FloatEventHandlers() {
            label = "Angle",
            id = 4,
            eventTypeLabels = sliderEventTypeLabels,
        }
```

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Physics/MechanicalJoint.cs

Passer.MenuManager Class Reference

Description

The Menu Manager uses two Interaction Pointers for each hand:

- An (straight) interaction pointer to interact with the menu when it is active
- An (curved) interaction pointer to do teleporting when the menu is not active. Both pairs of interaction pointers needs to be set in the Inspector to work correctly.

The Menu Manager uses a Trigger Sphere Collider to detect if the user has moved too far the the menu to interact with it.

Inherits MonoBehaviour.

Public Member Functions

- void SetMenuActive (bool active)
  Activates or deactivates the menu and updates the interaction pointer behaviour

Public Attributes

- HumanoidControl **humanoid**
- float **menuDistance** = 0.5F
- InteractionPointer **leftMenuPointer**
- InteractionPointer **rightMenuPointer**
- InteractionPointer **leftTeleporter**
- InteractionPointer **rightTeleporter**

Protected Member Functions

- virtual void **Awake** ()
  Initialization

- HumanoidControl FindHumanoid ()
  Tries to find the local Humanoid

- void InitInteractionPointers (HumanoidControl humanoid)
  Detects the Teleporter and Menu Interaction pointer on the humanoid

- InteractionPointer GetInteractionPointer (HandTarget handTarget, InteractionPointer invalidPointer=null)
  Find interaction pointer on the hand

- void **SetControllerInput** (HumanoidControl humanoid)
- void **OnTriggerExit** (Collider other)
  Trigger handler for when an object moves out of the sphere collider

- void **ShowMenu** ()

Show the menu at 'distance' from the players' head. But take core it is not placed inside objects. If this happens, the menu is places closer to the humanoid.

- void **HideMenu** ()
  Hide the menu

- bool MenuActive ()
  Is the menu currently visible?

- void EnableMenuPointer (ControllerInput controllerInput)
  Enable the Menu Pointer. This will also set the controller input.

- void DisableMenuPointer (ControllerInput controllerInput)
  Disable the Menu Pointer. This will also disable the controller input.

- void EnableTeleporter (ControllerInput controllerInput)
  Enable the teleporter. This will also set the controller input.

- void DisableTeleporter (ControllerInput controllerInput)
  Disable the Teleporter. This will also disable the controller input

- void **AdjustOutOfRangeDistance** ()
  Adjust trigger sphere collider radius for out-of-range

Member Function Documentation

HumanoidControl Passer.MenuManager.FindHumanoid () `[protected]`

Tries to find the local Humanoid

Returns

The found humanoid, null if no local humanoid has been found

void Passer.MenuManager.InitInteractionPointers (HumanoidControl *humanoid*) `[protected]`

Detects the Teleporter and Menu Interaction pointer on the humanoid

Parameters

humanoid	The humanoid for which the interaction pointers need to be found

InteractionPointer Passer.MenuManager.GetInteractionPointer (HandTarget *handTarget*, InteractionPointer *invalidPointer* = `null`) `[protected]`

Find interaction pointer on the hand

Parameters

handTarget	The hand to which the Interaction Pointer should be attached
invalidPointer	(optional) when give, the interaction pointer should be not euqual to the invalidPointer

Returns

The found interaction pointer

void Passer.MenuManager.SetMenuActive (bool *active*)

Activates or deactivates the menu and updates the interaction pointer behaviour

Parameters

active	Indication whether the menu has to be active

bool Passer.MenuManager.MenuActive () `[protected]`

Is the menu currently visible?

Returns

boolean indicating whether the menu is visible

void Passer.MenuManager.EnableMenuPointer (ControllerInput *controllerInput*) `[protected]`

Enable the Menu Pointer. This will also set the controller input.

Parameters

controllerInput	The ControllerInput to update

void Passer.MenuManager.DisableMenuPointer (ControllerInput *controllerInput*) `[protected]`

Disable the Menu Pointer. This will also disable the controller input.

Parameters

controllerInput	The ControllerInput to update

void Passer.MenuManager.EnableTeleporter (ControllerInput *controllerInput*) `[protected]`

Enable the teleporter. This will also set the controller input.

Parameters

controllerInput	The ControllerInput to update

void Passer.MenuManager.DisableTeleporter (ControllerInput *controllerInput*) `[protected]`

Disable the Teleporter. This will also disable the controller input

Parameters

controllerInput	The ControllerInput to update

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Tools/MenuManager.cs

Passer.NetworkingStarter Class Reference

Description

Setup and start of networking

To make a networked multiplayer environment you can use the Networking Starter component. It is still possible to use dedicated code and components for the chosen networking solution instead which will give you more control of the setup of the network.

The easiest way to turn a single-user scene in a multi-user environment is to add the NetworkingStarter prefab to the scene. You can find this in the Humanoid->Prefabs->Networking folder.

Inherits Photon.Pun.MonoBehaviourPunCallbackselse public class NetworkingStarter Photon.PunBehaviourendifelif hNW_BOLT public class NetworkingStarter Bolt.GlobalEventListenerelse public class NetworkingStarter MonoBehaviourendif. Inherited by Passer.Humanoid.NetworkingSpawner.

Public Types

- enum ServerType { **CloudServer**, **OwnServer** }
  Server types
- enum Role { **Host**, **Client** }
  Network Role

Public Member Functions

- void **StartHost** ()
  Start local networking with Host role

- void **StartClient** ()
  Start local networking with Client role

- void **StartClient** (string roomName, int gameVersion)
  Start cloud networking with Client role

- override void **OnConnectedToMaster** ()
- override void **OnJoinRoomFailed** (short returnCode, string message) public override void OnPhotonJoinRoomFailed(object[] codeAndMsg)
- override void **OnJoinedRoom** ()
- override void **BoltStartDone** ()
- override void **SessionListUpdated** (Map< Guid, UdpSession > sessionList)

Public Attributes

- bool **autoStart** = true
- string **serverIpAddress** = "127.0.0.1"
  The IP address of the host

- string **roomName** = "default"
  The name of the environment shared by all the users

- int **gameVersion** = 1
  The version of the environment shared by all the users

- GameObject **playerPrefab**
  The player prefab which will be spawned across the network.

- ServerType **serverType**
  The type of server used for the network

- bool **useRoleFile** = false
  Enables the use of a role file which determines whether the application is a Host or a Client

- string **roleFileName** = "Role.txt"
  The filename of the role file

- Role **role**
  The Role of this application

- int sendRate = 25
  The rate at which humanoid pose messages are sent

Protected Member Functions

- virtual void **Awake** ()
- virtual void **Start** ()

Protected Attributes

- INetworkingStarter **starter** = new UnetStarter()

Properties

- bool **connected** `[getprotected set]`
  Indication whether the application is connected to the network

- bool **connecting** `[getprotected set]`
  Indication whether the application is trying to connect to the network

---

Member Data Documentation

int Passer.NetworkingStarter.sendRate = 25

The rate at which humanoid pose messages are sent

In messages per second

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Networking/NetworkingStarter.cs

Passer.Humanoid.NeuronTracker Class Reference

Description

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

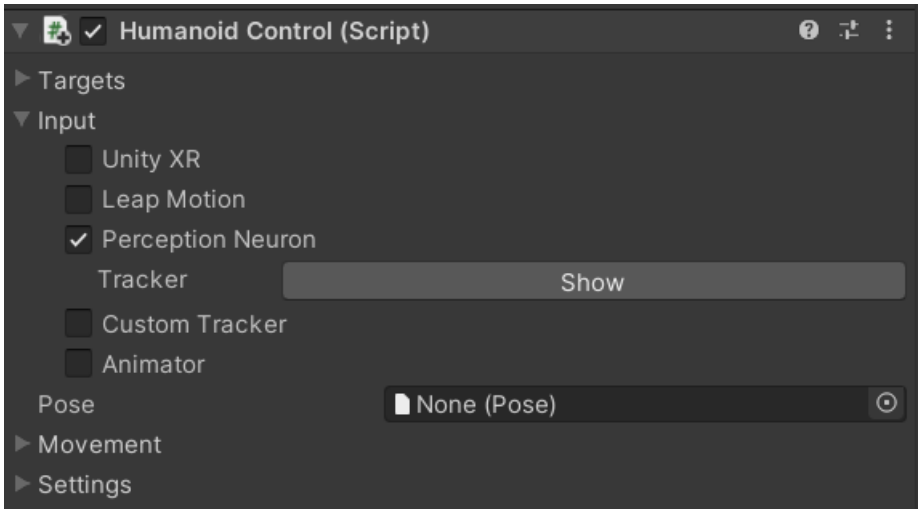
Setup

Axis Neuron

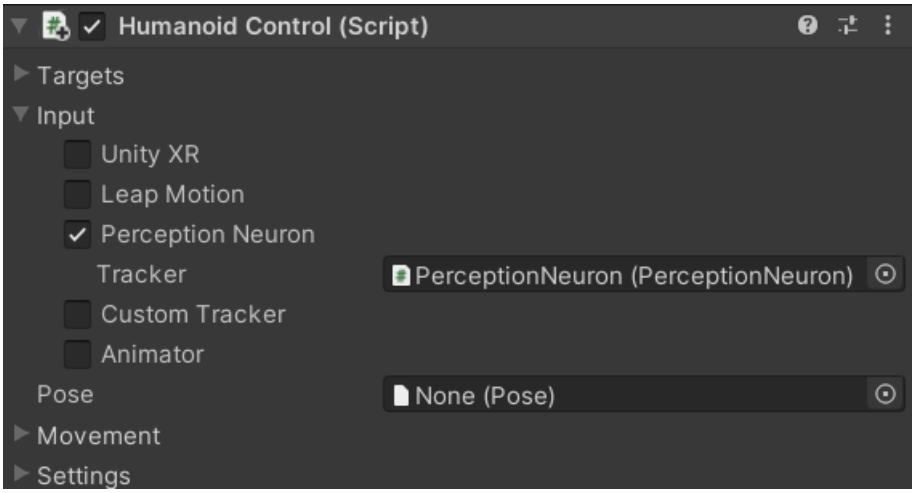
Axis Neuron software is required to support Perception Neuron. In the Settings-Broadcasting, make sure that BVH is enabled in Binary format, without Use old header. ServerPort is default set to 7001, but can be changed. The protocol needs to be set to TCP.

Inspector

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

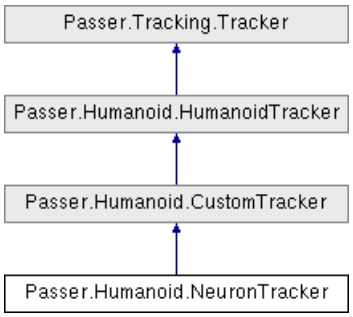


By default, the PerceptionNeuron objectis not visible in the scene, but it will be created automatically with the default settings when the scene starts. If the button *Show* is pressed, the PerceptionNeuron object will be created in the *Real World* object.



PerceptionNeuron (PerceptionNeuron) is a reference to the object in the scene representing the Perception Neuron tracking system. This GameObject is found as a child of the *Real World* GameObject. The PerceptionNeuron component can be used to set the position of the tracking system in the scene. It is also used to change connection settings like the ip address and port on which the Axis Neuron is broadcasting.

Inheritance diagram for Passer.Humanoid.NeuronTracker:



Public Types

- enum Status { Unavailable, Present, Tracking }
  The tracking status

Public Member Functions

- override void CheckTracker (HumanoidControl humanoid)
  Check the presence of a PerceptionNeuron object and creates one if it does not exist

- override void StartTracker (HumanoidControl humanoid)
  Starts the tracker

- override void UpdateTracker ()
  Update the tracker state

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker)
  Function to check the status of a specific tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker, Vector3 localPosition, Quaternion localRotation)
  Function to check the status of a specific tracker

- delegate TrackerComponent TrackerGetter (Transform transform, Vector3 localPosition, Quaternion localRotation)
  Function delegate for retrieving the tracker

- virtual Vector3 **GetBonePosition** (uint actorId, FacialBone boneId)
- virtual Quaternion **GetBoneRotation** (uint actorId, FacialBone boneId)
- virtual float **GetBoneConfidence** (uint actorId, FacialBone boneId)
- virtual void StartTracker ()
  Optional list of SubTrackers

- virtual void StopTracker ()
  Stop the tracker

- virtual void ShowTracker (bool shown)
  Show or hide the Tracker renderers

- virtual void Calibrate ()
  Calibrate the tracker

- virtual void AdjustTracking (Vector3 positionalDelta, Quaternion rotationalDelta)
  Adjust the position of the tracker by the given delat

Public Attributes

- BodySkeleton bodySkeleton
  A skeleton for the body

- HumanoidControl **humanoid**
  The humanoid for this tracker

- bool **enabled**
  Is this tracker enabled?

- Status **status**
  The tracking Status of the tracker

- TrackerComponent **trackerComponent**
  The tracking device

Protected Member Functions

- void **UpdateBodyFromSkeleton** ()
- virtual void **UpdateTorso** ()
- virtual void **UpdateLeftArm** ()
- virtual void **UpdateRightArm** ()
- virtual void **UpdateLeftLeg** ()
- virtual void **UpdateRightLeg** ()

Properties

- override string name `[get]`
  The name of this tracker

- virtual HeadSensor headSensor `[get]`
  Get the sensor for the head

- virtual ArmSensor leftHandSensor `[get]`
  Get the sensor for the left hand

- virtual ArmSensor rightHandSensor `[get]`
  Get the sensor for the right hand

- virtual TorsoSensor hipsSensor `[get]`
  Get the sensor for the hips

- virtual LegSensor leftFootSensor `[get]`
  Get the sensor for the left foot

- virtual LegSensor rightFootSensor `[get]`
  Get the sensor for the right foot

- virtual HumanoidSensor[] **sensors** `[get]`
  The sensors for this tracker

Member Enumeration Documentation

enum Passer.Tracking.Tracker.Status `[inherited]`

The tracking status

Enumerator:

Unavailable	The tracking device is not available.
Present	The tracking device is available but not tracking.
Tracking	The tracking device is actively tracking.

Member Function Documentation

override void Passer.Humanoid.NeuronTracker.CheckTracker (HumanoidControl *humanoid*) `[virtual]`

Check the presence of a PerceptionNeuron object and creates one if it does not exist

Parameters

humanoid	The humanoid for which the tracker needs to be checked

Reimplemented from Passer.Humanoid.HumanoidTracker.

override void Passer.Humanoid.NeuronTracker.StartTracker (HumanoidControl *humanoid*) `[virtual]`

Starts the tracker

Parameters

humanoid	The humanoid for which the tracker needs to be started

It will check the presence of a tracker using CheckTracker first.

Reimplemented from Passer.Humanoid.HumanoidTracker.

override void Passer.Humanoid.NeuronTracker.UpdateTracker () `[virtual]`

Update the tracker state

Reimplemented from Passer.Tracking.Tracker.

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*) `[inherited]`

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker

The default position/rotation for the tracker when created will be zero

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*, Vector3 *localPosition*, Quaternion *localRotation*) `[inherited]`

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

delegate TrackerComponent Passer.Humanoid.HumanoidTracker.TrackerGetter (Transform *transform*, Vector3 *localPosition*, Quaternion *localRotation*) `[inherited]`

Function delegate for retrieving the tracker

Parameters

transform	The root transform to start the searching of the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

Returns

The tracker component found or created

The default position/rotation is relative to the humanoid's real world.

virtual void Passer.Tracking.Tracker.StartTracker () `[virtual], [inherited]`

Optional list of SubTrackers

Start the tracker

virtual void Passer.Tracking.Tracker.StopTracker () `[virtual], [inherited]`

Stop the tracker

Reimplemented in Passer.Humanoid.LeapTracker.

virtual void Passer.Tracking.Tracker.ShowTracker (bool *shown*) `[virtual], [inherited]`

Show or hide the Tracker renderers

Parameters

shown	Renderers are enabled when shown == true

virtual void Passer.Tracking.Tracker.Calibrate () `[virtual], [inherited]`

Calibrate the tracker

Reimplemented in Passer.Humanoid.UnityXRTracker.

virtual void Passer.Tracking.Tracker.AdjustTracking (Vector3 *positionalDelta*, Quaternion *rotationalDelta*) `[virtual], [inherited]`

Adjust the position of the tracker by the given delat

Parameters

positionalDelta	The positional delta to apply
rotationalDelta	The rotational delta to apply

Member Data Documentation

BodySkeleton Passer.Humanoid.CustomTracker.bodySkeleton `[inherited]`

A skeleton for the body

When this is set, the tracking will be taken from this skeleton

---

Property Documentation

override string Passer.Humanoid.NeuronTracker.name [get]

The name of this tracker

virtual HeadSensor Passer.Humanoid.HumanoidTracker.headSensor [get], [inherited]

Get the sensor for the head

Will return null when this sensor is not present

virtual ArmSensor Passer.Humanoid.HumanoidTracker.leftHandSensor [get], [inherited]

Get the sensor for the left hand

Will return null when this sensor is not present

virtual ArmSensor Passer.Humanoid.HumanoidTracker.rightHandSensor [get], [inherited]

Get the sensor for the right hand

Will return null when this sensor is not present

virtual TorsoSensor Passer.Humanoid.HumanoidTracker.hipsSensor [get], [inherited]

Get the sensor for the hips

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.leftFootSensor [get], [inherited]

Get the sensor for the left foot

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.rightFootSensor [get], [inherited]

Get the sensor for the right foot

Will return null when this sensor is not present

The documentation for this class was generated from the following file:

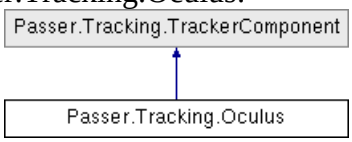
- Assets/Passer/HumanoidControlPro/Runtime/Scripts/Extensions/PerceptionNeuron/PerceptionNeuronTracker.cs

Passer.Tracking.Oculus Class Reference

Description

The Oculus Device

Inheritance diagram for Passer.Tracking.Oculus:



Public Member Functions

- HandSkeleton **FindHandTrackingSkeleton** (bool isLeft)
- override void ShowSkeleton (bool shown)
- OculusHmd **GetHmd** (Vector3 position, Quaternion rotation)
- OculusController FindController (bool isLeft)
  Find an OpenVR Controller

- OculusController **GetController** (bool isLeft, Vector3 position, Quaternion rotation)

Static Public Member Functions

- static Oculus Find (Transform parentTransform)
  Find an Oculus tracker

- static Oculus Get (Transform parentTransform, Vector3 position, Quaternion rotation)
  Find or create a new SteamVR tracker

Public Attributes

- bool **persistentTracking**
- RealWorldConfiguration **realWorldConfiguration**
- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- override void Start ()
- virtual void **Update** ()

Static Protected Member Functions

- static void **AddScreenFader** (Camera camera)

Protected Attributes

- OculusHmd **_hmd**
- Transform **realWorld**

Properties

- HandSkeleton **leftHandSkeleton** `[get]`
- HandSkeleton **rightHandSkeleton** `[get]`

- OculusHmd **hmd** `[get]`

Member Function Documentation

override void Passer.Tracking.Oculus.ShowSkeleton (bool *shown*) `[virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

static Oculus Passer.Tracking.Oculus.Find (Transform *parentTransform*) `[static]`

Find an Oculus tracker

Parameters

parentTransform	The parent transform of the tracker

Returns

The tracker

static Oculus Passer.Tracking.Oculus.Get (Transform *parentTransform*, Vector3 *position*, Quaternion *rotation*) `[static]`

Find or create a new SteamVR tracker

Parameters

parentTransform	The parent transform for the tracker
position	The world position of the tracker
rotation	The world rotation of the tracker

Returns

The tracker

OculusController Passer.Tracking.Oculus.FindController (bool *isLeft*)

Find an OpenVR Controller

Parameters

isLeft	Looking for the left-handed controller?

Returns

The controller or null when it has not been found

override void Passer.Tracking.Oculus.Start () `[protected]`, `[virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Oculus/Oculus.cs

Passer.Tracking.OculusController Class Reference

Description

An Oculus controller

Inherits Passer.Tracking.ControllerComponent.

Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- void **Vibrate** (float length, float strength)
- virtual void **StartComponent** (Transform trackerTransform, bool isLeft)
- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Public Attributes

- TrackerComponent **tracker**
- bool **positionalTracking** = true
- bool **isLeft**
- Vector3 **primaryAxis**
- Vector3 **secondaryAxis**
- float **trigger1**
- float **trigger2**
- float **button1**
- float **button2**
- float **button3**
- float **button4**
- float **option**
- float **battery**
- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- virtual void Awake ()
  Initializes the sensor.

- virtual void Start ()
  Starts the sensor

Protected Attributes

- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- override bool **show** `[getset]`
- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.OculusController.UpdateComponent () `[virtual]`

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual]`, `[inherited]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

virtual void Passer.Tracking.SensorComponent.Awake () `[protected]`, `[virtual]`, `[inherited]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.Start () `[protected]`, `[virtual]`, `[inherited]`

Starts the sensor

Does nothing at this moment.

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, and Passer.Tracking.HydraController.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true [inherited]

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

The documentation for this class was generated from the following file:

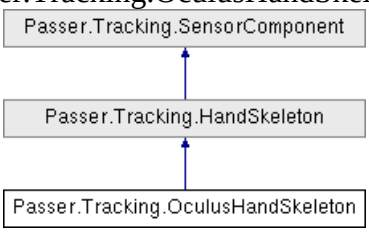
- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Oculus/OculusController.cs

Passer.Tracking.OculusHandSkeleton Class Reference

Description

A hand from Oculus hand tracking

Inheritance diagram for Passer.Tracking.OculusHandSkeleton:



Public Types

- enum **BoneId** { **Invalid** = -1, **Hand** = 0, **ThumbProximal**, **ThumbIntermediate**, **ThumbDistal**, **ThumbTip**, **IndexMetaCarpal**, **IndexProximal**, **IndexIntermediate**, **IndexDistal**, **IndexTip**, **MiddleMetaCarpal**, **MiddleProximal**, **MiddleIntermediate**, **MiddleDistal**, **MiddleTip**, **RingMetaCarpal**, **RingProximal**, **RingIntermediate**, **RingDistal**, **RingTip**, **LittleMetacarpal**, **LittleProximal**, **LittleIntermediate**, **LittleDistal**, **LittleTip**, **Forearm**, **Count** }

Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- override int GetBoneId (Finger finger, FingerBone fingerBone)
- virtual GameObject **CreateHandSkeleton** (Transform trackerTransform, bool isLeft, bool showRealObjects)
- virtual Transform **GetForearmBone** ()
- virtual Transform **GetWristBone** ()
- virtual Transform GetBoneTransform (Finger finger, FingerBone fingerBone)
  Gets the transform of the tracked bone

- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Static Public Member Functions

- static OculusHandSkeleton **Find** (Transform trackerTransform, bool isLeft)
- static OculusHandSkeleton **Get** (Transform trackerTransform, bool isLeft)
- static HandSkeleton **FindHandSkeleton** (Transform trackerTransform, bool isLeft)

Public Attributes

- bool **isLeft**
  True: this is a left hand. Fale: this is a right hand

- new bool **show** = false
  Determines whether this skeleton should be rendered

- Tracker.Status **status**

The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- override void InitializeSkeleton ()
  This function is used to initialize the tracked bones

- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()
- virtual void Awake ()
  Initializes the sensor.

- virtual void Start ()
  Starts the sensor

Protected Attributes

- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.OculusHandSkeleton.InitializeSkeleton () [protected], [virtual]

This function is used to initialize the tracked bones

Reimplemented from Passer.Tracking.HandSkeleton.

override void Passer.Tracking.OculusHandSkeleton.UpdateComponent () [virtual]

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

override int Passer.Tracking.OculusHandSkeleton.GetBoneId (Finger *finger*, FingerBone *fingerBone*) [virtual]

Reimplemented from Passer.Tracking.HandSkeleton.

virtual Transform Passer.Tracking.HandSkeleton.GetBoneTransform (Finger *finger*, FingerBone *fingerBone*) [virtual], [inherited]

Gets the transform of the tracked bone

Parameters

finger	The requested finger
fingerBone	The requested bone in the finger

Returns

The tracked bon transform of *null* if it does not exist

Reimplemented in Passer.Tracking.LeapHandSkeleton.

virtual void Passer.Tracking.SensorComponent.Awake () [protected], [virtual], [inherited]

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.Start () [protected], [virtual], [inherited]

Starts the sensor

Does nothing at this moment.

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, and Passer.Tracking.HydraController.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual], [inherited]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true `[inherited]`

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

The documentation for this class was generated from the following file:

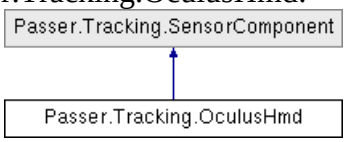
- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Oculus/OculusHandSkeleton.cs

Passer.Tracking.OculusHmd Class Reference

Description

The Oculus Head Mounted Device

Inheritance diagram for Passer.Tracking.OculusHmd:



Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Static Public Member Functions

- static OculusHmd FindHmd (Transform oculusTransform)
  Find the Oculus Hmd

- static OculusHmd **Get** (Transform oculusTransform, Vector3 position, Quaternion rotation)
  Find or Create a new Antilatency Sensor

Public Attributes

- Camera **unityCamera**
- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- virtual void **FuseWithUnityCamera** ()
- virtual void **ResetTrackingPosition** ()
- virtual void Awake ()
  Initializes the sensor.

- virtual void Start ()
  Starts the sensor

Protected Attributes

- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- virtual bool show `[getset]`
  The render status of the sensor

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

static OculusHmd Passer.Tracking.OculusHmd.FindHmd (Transform *oculusTransform*) `[static]`

Find the Oculus Hmd

Parameters

oculusTransform	

Returns

override void Passer.Tracking.OculusHmd.UpdateComponent () `[virtual]`

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

virtual void Passer.Tracking.SensorComponent.Awake () `[protected]`, `[virtual]`, `[inherited]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.Start () `[protected]`, `[virtual]`, `[inherited]`

Starts the sensor

Does nothing at this moment.

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, and Passer.Tracking.HydraController.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual], [inherited]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true `[inherited]`

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

Property Documentation

virtual bool Passer.Tracking.SensorComponent.show `[get], [set], [inherited]`

The render status of the sensor

When enabled, sensors with renderers attached will be rendered. When disabled, sensors will not be rendered.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/Oculus/OculusHmd.cs

Passer.FunctionCall.Parameter Class Reference

Description

Function Parameter

Public Attributes

- bool fromEvent = false
  The parameter value comes from the event

- string **localProperty**
  For future use...

- ParameterType type
  The type of the parameter

- float **floatConstant**
  The constant float value when the parameter type is float and fromEvent is false.

- int **intConstant**
  The constant integer value when the parameter type is int and fromEvent is false.

- bool **boolConstant**
  The constant boolean value when the parameter type is bool and fromEvent is false.

- string **stringConstant**
  The constant string value when the parameter type is string and fromEvent is false.

- Vector3 **vector3Constant**
  The constant Vector3 value when the parameter type is Vector3 and fromEvent is false.

- GameObject **gameObjectConstant**
  The constant GameObject value when the parameter type is GameObject and fromEvent is false.

- Rigidbody **rigidbodyConstant**
  The constant Rigidbody value when the parameter type is Rigidbody and fromEvent is false.

Member Data Documentation

bool Passer.FunctionCall.Parameter.fromEvent = false

The parameter value comes from the event

When false one of the constant values is used, based on the type of the parameter

ParameterType Passer.FunctionCall.Parameter.type

The type of the parameter

May be converted to a System.Type later...

The documentation for this class was generated from the following file:

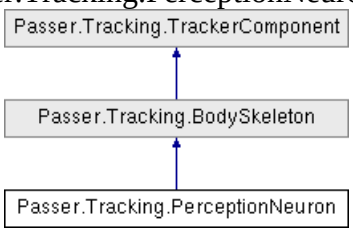
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/FunctionCall.cs

Passer.Tracking.PerceptionNeuron Class Reference

Description

Component to receive tracking information from Axis Neuron

Inheritance diagram for Passer.Tracking.PerceptionNeuron:



Public Member Functions

- Transform GetBoneTransform (Humanoid.Tracking.Bone boneId)
  Gets the transform of the tracked bone

- TrackedBone GetBone (Humanoid.Tracking.Bone boneId)
  Gets the tracked bone

- virtual void **ShowSkeleton** (bool shown)

Static Public Member Functions

- static PerceptionNeuron Find (Transform parentTransform)
  Find a Perception Neuron tracker

- static PerceptionNeuron **Get** (Transform parentTransform, Vector3 localPosition, Quaternion localRotation)
  Create a new Perception Neuron Tracker

Public Attributes

- string **ipAddress** = "127.0.0.1"
  The ip adress on which Axis Neuron is running

- int **port** = 7001
  The port on which Axis Neuron is broadcasting the tracking data.

- bool **show** = false
  Determines whether the skeleton should be rendered

- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- override void Start ()

- override void InitializeSkeleton ()
  This function is used to intialize the tracked bones.

- void **InitializeBone** (Bone boneId, Transform parent, bool addRenderer=true)
- void **OnDisable** ()
- override void Update ()
- void **UpdateSkeletonBones** ()
- void **UpdateTorso** ()
- void **UpdateLeftArm** ()
- void **UpdateRightArm** ()
- void **UpdateLeftLeg** ()
- void **UpdateRightLeg** ()
- void **UpdateBonePosition** (Bone boneId)
- void **UpdateBoneLocalPosition** (Bone boneId)
- void **UpdateBoneRotation** (Bone boneId)
- void **UpdateBoneRotation** (Bone boneId, Side side, SideBone sideBoneId)
- void **UpdateBoneLocalPosition** (Bone boneId, Side side, SideBone sideBoneId)
- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()

Protected Attributes

- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **realWorld**

Member Function Documentation

static PerceptionNeuron Passer.Tracking.PerceptionNeuron.Find (Transform *parentTransform*) `[static]`

Find a Perception Neuron tracker

Parameters

parentTransform	The parent transform of the tracker

Returns

The tracker

override void Passer.Tracking.PerceptionNeuron.Start () `[protected]`, `[virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

override void Passer.Tracking.PerceptionNeuron.InitializeSkeleton () `[protected]`, `[virtual]`

This function is used to intialize the tracked bones.

Implements Passer.Tracking.BodySkeleton.

override void Passer.Tracking.PerceptionNeuron.Update () `[protected], [virtual]`

Reimplemented from Passer.Tracking.TrackerComponent.

Transform Passer.Tracking.BodySkeleton.GetBoneTransform (Humanoid.Tracking.Bone *boneId*) `[inherited]`

Gets the transform of the tracked bone

Parameters

boneId	The requested bone

Returns

The tracked bone transform or *null* if it does not exist

TrackedBone Passer.Tracking.BodySkeleton.GetBone (Humanoid.Tracking.Bone *boneId*) `[inherited]`

Gets the tracked bone

Parameters

boneId	The requested bone

Returns

The tracked bone or *null* if it does not exist

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPro/Runtime/Scripts/Extensions/PerceptionNeuron/PerceptionNeuron.cs

Passer.PlayMakerHumanoidHand Class Reference

Description

Script for automatic sending of Grabbing and LettingGo events

Inherits MonoBehaviour.

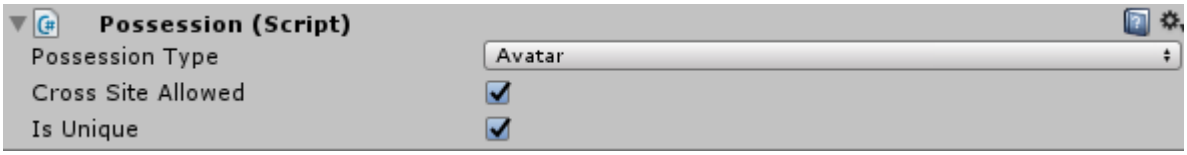
The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlVR/Runtime/Scripts/Extensions/PlayMaker/PlayMakerHumanoidHand.cs

Passer.Possessable Class Reference

Description

Possessions can be owned by an Humanoid



- Possession type, see Possession::possessionType
- Cross site allowed, see Possession::crossSite
- Is unique, see Possession::isUnique

Version

4

Inherits MonoBehaviour.

Public Types

- enum Type { Generic, Avatar }
  The possession type

Public Attributes

- Type **possessionType**
  The Type of Possession

- bool crossSite = true
  If true, this Posession can be taken to other Sites.

- bool **isUnique** = false
  An unique Possession can be possessed only once.

- string **assetPath**

Protected Member Functions

- virtual void **Awake** ()

Properties

- string **siteLocation** `[get]`

Member Enumeration Documentation

enum Passer.Possessable.Type

The possession type

Enumerator:

Generic	A generic Possession.
Avatar	An avatar can be worn by a Humanoid.

Member Data Documentation

bool Passer.Possessable.crossSite = true

If true, this Posession can be taken to other Sites.

Non cross site possessions will be removed from the Humanoid's possessions when they leave the site.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Sites/Scripts/Possessable.cs

Passer.SceneManager Class Reference

Description

The scene manager synchronizes scene changes with humanoids across a network.

Inherits MonoBehaviourPunCallbackselse public class SceneManager Photon.MonoBehaviourendifelif hNW_UNETpragma warning disableRequireComponenttypeofNetworkIdentity public class SceneManager NetworkBehaviourpragma warning restoreelse public class SceneManager MonoBehaviourendif.

Public Member Functions

- void LoadScene (int sceneId)
  Load the scene and causes a scene change.

- void NextScene ()
  Changes the scene to the next scene in the list

- void PreviousScene ()
  Changes the scene to the previous scene in the list

Static Public Member Functions

- static int **mod** (int k, int n)

Public Attributes

- int **currentScene** = 0
  The index of the current scene.

- string[] **sceneNames**
  The list of scenes from the Build Settings.

- bool **dontDestroyOnLoad** = false
  Will prevent the scene manager from being destroyed when the scene changes.

Protected Member Functions

- virtual void **Awake** ()

Member Function Documentation

void Passer.SceneManager.LoadScene (int *sceneId*)

Load the scene and causes a scene change.

Parameters

sceneId	The index of the new scene in the list of scenes

void Passer.SceneManager.NextScene ()

Changes the scene to the next scene in the list

This will wrap around when the last scene in the list has been reached.

void Passer.SceneManager.PreviousScene ()

Changes the scene to the previous scene in the list

This will wrap aorund when the first scene in the list has been reached.

The documentation for this class was generated from the following file:

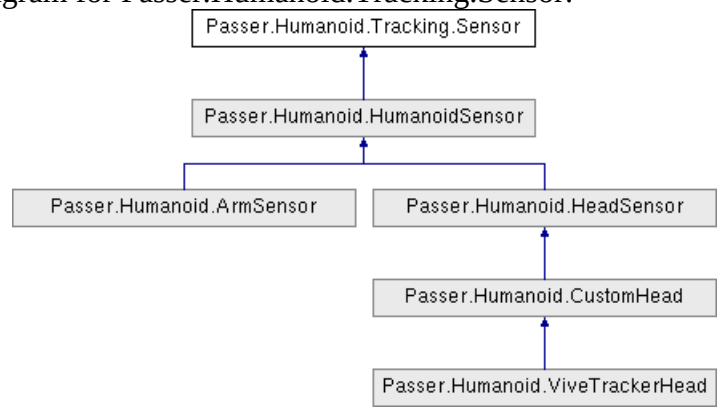
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/SceneManager.cs

Passer.Humanoid.Tracking.Sensor Class Reference

Description

Humanoid Tracking sensor

Inheritance diagram for Passer.Humanoid.Tracking.Sensor:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- Sensor (DeviceView _device)
  Create new sensor the the device

- virtual Tracker.Status Update ()
  Update the sensor state

Static Public Member Functions

- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- void **UpdateSensor** ()

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**

- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Constructor & Destructor Documentation

Passer.Humanoid.Tracking.Sensor.Sensor (DeviceView *_device*)

Create new sensor the the device

Parameters

_device	

Member Function Documentation

virtual Tracker.Status Passer.Humanoid.Tracking.Sensor.Update () `[virtual]`

Update the sensor state

Returns

Status of the sensor after the update

Reimplemented in Passer.Humanoid.HeadSensor, Passer.Humanoid.HumanoidSensor, Passer.Humanoid.LeapHand, Passer.Humanoid.CustomArm, Passer.Humanoid.CustomHead, Passer.Humanoid.CustomLeg, and Passer.Humanoid.CustomTorso.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Libraries/Core/Sensors/Sensor.cs

Passer.Sensor Class Reference

Description

A sensor used to track an object

Public Member Functions

- virtual void **Start** (Transform targetTransform)
- virtual void **Update** ()
- virtual void **ShowSensor** (bool shown)

Public Attributes

- bool **enabled**
- Target **target**
- Tracker **tracker**
- SensorComponent **sensorComponent**
  The sensor used for tracking

Properties

- virtual Tracker.Status **status** `[getset]`
- virtual string **name** `[get]`

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/Sensor.cs

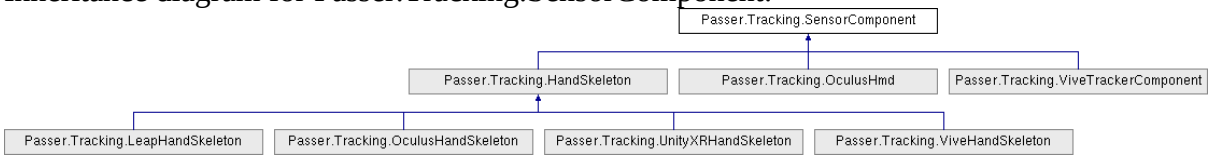
Passer.Tracking.SensorComponent Class Reference

Description

A sensor component is used to add tracking to a transform

Custom sensor implementation can be made by deriving from this class.

Inheritance diagram for Passer.Tracking.SensorComponent:



Public Member Functions

- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

- virtual void UpdateComponent ()
  Update the component manually

Public Attributes

- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- virtual void Awake ()
  Initializes the sensor.

- virtual void Start ()
  Starts the sensor

Protected Attributes

- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- virtual bool show `[getset]`
  The render status of the sensor

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

virtual void Passer.Tracking.SensorComponent.Awake () `[protected], [virtual]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.Start () `[protected], [virtual]`

Starts the sensor

Does nothing at this moment.

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, and Passer.Tracking.HydraController.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

virtual void Passer.Tracking.SensorComponent.UpdateComponent () `[virtual]`

Update the component manually

This function is meant to be overridden

Reimplemented in Passer.Tracking.UnityXRHandSkeleton, Passer.Tracking.LeapHandSkeleton, Passer.Tracking.ViveHandSkeleton, Passer.Tracking.ViveTrackerComponent, Passer.Tracking.HydraController, Passer.Tracking.OculusController, Passer.Tracking.OculusHandSkeleton, and Passer.Tracking.OculusHmd.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

Property Documentation

virtual bool Passer.Tracking.SensorComponent.show `[get], [set]`

The render status of the sensor

When enabled, sensors with renderers attached will be rendered. When disabled, sensors will not be rendered.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/SensorComponent.cs

Passer.Site Class Reference

Description

With this component you can make a Humanoid Site which can be build using SiteBuilder.

This component has currently no parameters. To make a scene into a site, just add an (empty) GameObject with the Site component attached into your scene. Make sure there are no camera's in the scene, because the camera of the Visitor will be used.

When used in the editor, you can just start the scene with the Play button at the top. A Visitor will then be launched for the site so that you can test it. The type of Visitor is selected in the Humanoid::HumanoidPreferences.

You can build a site by selecting the File->Build Sites menu. This will launch the SiteBuilder.

It is possible to determine the place where the Visitor will appear by adding a Humanoid::HumanoidSpawnPoint component to the scene

By default sites are single-user. This means that when a Visitor visits a site, other visitors on the same site at the same time will not be visible. It is possible to make a multi-user site by adding a NetworkingStarter component to the site. In that case visitors will see all other visitors on the same site at that moment. When Photon Voice is used, visitors will also be able to talk to each other.

Example sites can be found in Assets/Passer/Sites/

Version

4.0 and higher

Inherits MonoBehaviour.

The documentation for this class was generated from the following file:

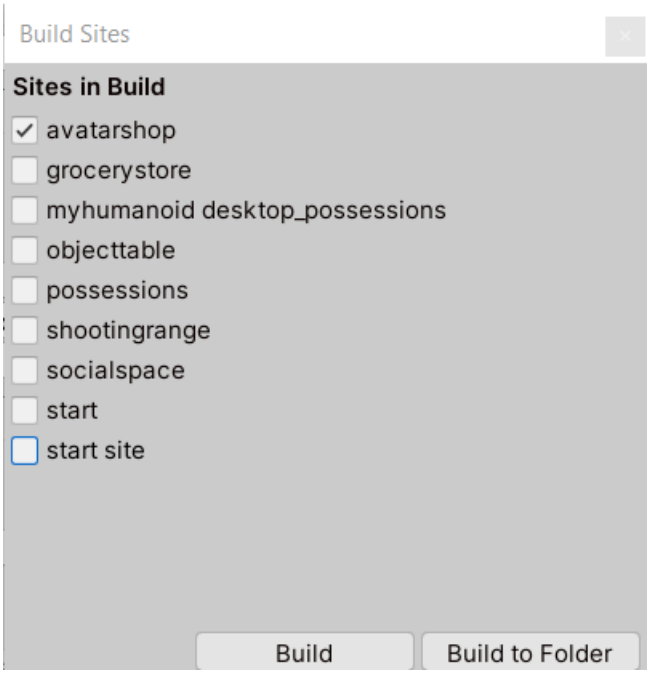
- Assets/Passer/HumanoidControl/Runtime/Sites/Scripts/Site.cs

Passer.SiteBuilder Class Reference

Description

A component for building Humanoid Sites

Sites can be built by selecting the *File->Build Sites* menu. After this you will be able to select which sites will be built by selecting the appropriate sites in the dialog:



When the Build button is pressed, all sites will be build and become available in the Assets/SiteBuilds folder with a submap for each platform (like Windows, Android...)

When the Build to Folder is pressed, the built sites will additionally be copied to the selected target folder. This enables you to directly publish to a web site for instance.

Version

4 and higher

Static Public Member Functions

- static void **GenerateSiteBuilds** ()
- static void BuildSites (string savePath)
  Build all sites in this project and save them in the indicated location

- static void BuildWindowsSites (string savePath)
  Build all sites for the Windows Standalone platform in this project and save them in the indicated location

- static void BuildAndroidSites (string savePath)
  Build all sites for the Android platform in this project and save them in the indicated location

- static void BuildWebGLSites (string savePath)
  Build all sites for the WebGL platform in this project and save them in the indicated location

Static Public Attributes

- static Sites **siteBuilds** = null

Static Protected Member Functions

- static AssetBundleBuild[] **GetBuildmap** ()

Member Function Documentation

static void Passer.SiteBuilder.BuildSites (string *savePath*) [static]

Build all sites in this project and save them in the indicated location

Currently this function will build all sites for Windows and Android platforms only.

Parameters

savePath	The full path to the folder in which the sites should be saved

static void Passer.SiteBuilder.BuildWindowsSites (string *savePath*) [static]

Build all sites for the Windows Standalone platform in this project and save them in the indicated location

Parameters

savePath	The full path to the folder in which the sites should be saved

static void Passer.SiteBuilder.BuildAndroidSites (string *savePath*) [static]

Build all sites for the Android platform in this project and save them in the indicated location

Parameters

savePath	The full path to the folder in which the sites should be saved

static void Passer.SiteBuilder.BuildWebGLSites (string *savePath*) [static]

Build all sites for the WebGL platform in this project and save them in the indicated location

Parameters

savePath	The full path to the folder in which the sites should be saved

---

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Editor/Sites/SiteBuilder.cs

Passer.SiteNavigator Class Reference

Description

The component which takes care of site navigation

Inherits MonoBehaviour.

Public Member Functions

- void **GoHome** ()
- void LoadSiteFromURL (string siteLocation)
  Change the current Site

- void **ReloadSite** ()
- void **GoBack** ()
- void **Quit** ()

Public Attributes

- bool **loadSiteAtStart** = false
- string startSite = ""
  The URL of the Site which should be visited at start

- string startScene
  The name of the scene which should be visited at start

Static Public Attributes

- static Stack< HistoryEntry > **history** = new Stack<HistoryEntry>()
- static HistoryEntry **currentSite** = null

Protected Member Functions

- virtual void **Awake** ()
- void **LoadStartSiteFromStartupJSON** ()
- virtual void **Start** ()
- void **Update** ()

Static Protected Member Functions

- static IEnumerator **LoadSite** (string siteLocation)
- static IEnumerator **UnloadLastAssetBundle** (AssetBundle newAssetBundle)

Static Protected Attributes

- static List< HistoryEntry > **cache** = new List<HistoryEntry>()
- static AssetBundle **currentAssetBundle** = null
- static AssetBundle **loadedAssetBundle** = null

Properties

- static SiteNavigator **instance** `[get]`

Member Function Documentation

void Passer.SiteNavigator.LoadSiteFromURL (string *siteLocation*)

Change the current Site

Parameters

siteLocation	The URL of the site to visit

The URL should not include a protocol (like https://) or an extension (like .site or .windows.site)

Member Data Documentation

string Passer.SiteNavigator.startSite = ""

The URL of the Site which should be visited at start

The URL should not include a protocol (like https://) or an extension (like .site or .windows.site)

string Passer.SiteNavigator.startScene

The name of the scene which should be visited at start

This overrides the startSite but only works in the editor

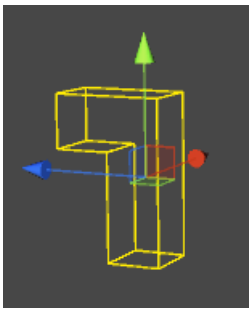
The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Sites/Scripts/SiteNavigator.cs

Passer.Socket Class Reference

Description

Sockets can hold a Handles.



Attaching and Releasing

Attaching a Handle using Scripting

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 a Prefab with a Handle 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 a Handle 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.

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

- Attach Events

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.

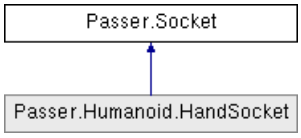
...with a Kinematic Rigidbody Handle

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.

Inheritance diagram for Passer.Socket:



Public Types

- enum **AttachMethod** { **Unknown**, **Parenting**, **ReverseParenting**, **Joint**, **StaticJoint**, **ColliderDuplication** }

Public Member Functions

- void **Attach** (GameObject objectToAttach)
- bool **Attach** (GameObject objectToAttach, bool rangeCheck)
- virtual bool Attach (Transform transformToAttach, bool rangeCheck=true)
  Tries to attach the given Transform to this socket

- virtual bool Attach (Handle handle, bool rangeCheck=true)
  Tries to attach the given Transform to this socket

- virtual void **AttachStaticJoint** (Transform objTransform)
- virtual void Release (bool releaseSticky=false)
  Releases a Transform from the socket

- virtual void **ReleaseStaticJoint** ()
- void **OnDrawGizmosSelected** ()

Static Public Member Functions

- static Mesh **GenerateGizmoMesh1** ()
- static Mesh **GenerateGizmoMesh** ()

Public Attributes

- GameObject **attachedPrefab**
  A prefab which is used to attach to the socket at startup.

- Transform attachedTransform
  The Transform attached to this socket

- Handle **attachedHandle**
- string socketTag
  A tag for limiting which handles can be held by the socket

- AttachMethod **attachMethod** = AttachMethod.Unknown
- bool **destroyOnLoad** = false
- GameObjectEventHandlers attachEvent
  A GameObject Event for triggering changes in the Transform held by the Socket

Protected Member Functions

- virtual void **MoveHandleToSocket** (Transform socketTransform, Rigidbody handleRigidbody, Handle handle)
- virtual void **MoveRailToSocket** (Transform socketTransform, Transform railTransform, Handle rail)
- virtual void **MoveHandleToSocket** (Transform socketTransform, Handle handle)
- virtual void **MoveSocketToHandle** (Transform socketTransform, Handle handle)
- virtual void **MoveSocketToHandle** (Transform socketTransform, Rigidbody socketRigidbody, Handle handle)
- virtual bool **AttachRigidbody** (Rigidbody objRigidbody, Handle handle, bool rangeCheck=true)
- void **AttachTransformParenting** (Transform objTransform, Handle handle)
- virtual void **AttachRigidbodyParenting** (Rigidbody objRigidbody, Handle handle)
- virtual void **AttachRigidbodyJoint** (Rigidbody objRigidbody, Handle handle)
- virtual void **AttachRigidbodyReverseJoint** (Rigidbody objRigidbody, Handle handle)
  Attach handle to socket using a static joint on the handle

- virtual void **AttachSocketParenting** (Rigidbody objRigidbody, Handle handle, Rigidbody socketRigidbody)
- virtual void **MassRedistribution** (Rigidbody socketRigidbody, Rigidbody objRigidbody)
- virtual bool **AttachStaticObject** (Transform objTransform, Handle handle)
- virtual void **AttachSocketParenting** (Transform objTransform, Handle handle, Rigidbody thisRigidbody)
- virtual void **AttachStaticJointRotY** (Transform objTransform)
- virtual void **ReleaseRigidbodyParenting** ()
- virtual void **ReleaseRigidbodyJoint** ()
- void **ReleaseRigidbodyReverseJoint** ()
- virtual void **ReleaseSocketParenting** (Rigidbody objRigidbody, Transform socketTransform)
- virtual void **MassRestoration** (Rigidbody socketRigidbody, Rigidbody objRigidbody)
- virtual void **ReleaseStaticObject** ()
- void **ReleaseTransformParenting** ()
- void **ReleaseSocketParenting** (Transform objTransform)
- virtual void **Awake** ()

- virtual void **Update** ()
- void **UpdateHolding** ()
- virtual void **OnSceneUnload** (UnityEngine.SceneManagement.Scene _)

Static Protected Member Functions

- static void **DebugLog** (string s)

Protected Attributes

- Transform **releasingTransform**
- Transform attachedTransformParent
  The parent of the attached transform before it was attached

- RigidbodyDisabled **rigidbodyDisabled** = null
- float **originalMass** = 1
- bool **originalUseGravity** = false
- Mesh **gizmoMesh**

Static Protected Attributes

- static string[] attachEventTypeLabels

Properties

- bool **isOccupied** `[get]`
  Does the socket currently have a handle attached?

Member Function Documentation

virtual bool Passer.Socket.Attach (Transform *transformToAttach*, bool *rangeCheck* = `true`) `[virtual]`

Tries to attach the given Transform to this socket

If the Transform has a Handle with the right Socket Tag it will be attached to the socket. Static and Kinematic Rigidbodies will be attached by parenting. Non-Kinematic Rigidbodies will be attached using a joint.

Parameters

transformToAttach	The Transform to attach to this socket

Returns

Boolean indicating whether attachment succeeded

virtual bool Passer.Socket.Attach (Handle *handle*, bool *rangeCheck* = `true`) `[virtual]`

Tries to attach the given Transform to this socket

If the Handle has the right Socket Tag it will be attached to the socket. Static and Kinematic Rigidbodies will be attached by parenting. Non-Kinematic Rigidbodies will be attached using a joint.

Parameters

handle	The Handle to attach to this socket

Returns

Boolean indicating whether attachment succeeded

Reimplemented in Passer.Humanoid.HandSocket.

virtual void Passer.Socket.Release (bool *releaseSticky* = `false`) `[virtual]`

Releases a Transform from the socket

Note that if the Transform is not taken out of the range of the socket or held by another Socket, it will automatically snap back to the Socket.

Reimplemented in Passer.Humanoid.HandSocket.

Member Data Documentation

Transform Passer.Socket.attachedTransform

The Transform attached to this socket

If the socket holds a Handle, this will contain the Transform of the Handle. It will be null otherwise

Transform Passer.Socket.attachedTransformParent `[protected]`

The parent of the attached transform before it was attached

This is used to restore the parent when the transform is released again.

string Passer.Socket.socketTag

A tag for limiting which handles can be held by the socket

If set (not null or empty) only Handles with the given tag will fit in the socket.

string [] Passer.Socket.attachEventTypeLabels `[static]`, `[protected]`

```
Initial value:= {
                "Never",
                "On Attach",
                "On Release",
                "While Attached",
                "While Released",
                "When Changed",
                "Always"
        }
```

GameObjectEventHandlers Passer.Socket.attachEvent

```
Initial value:= new GameObjectEventHandlers() {
            label = "Hold Event",
            tooltip =
                "Call functions using what the socket is holding\n" +
                "Parameter: the GameObject held by the socket",
            eventTypeLabels = attachEventTypeLabels
        }
```

A GameObject Event for triggering changes in the Transform held by the Socket

---

The documentation for this class was generated from the following file:

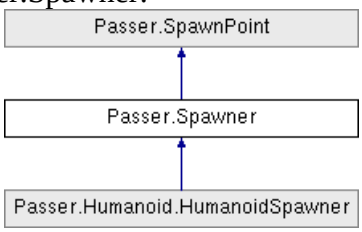
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/Socket.cs

Passer.Spawner Class Reference

Description

Component for spawning objects

Inheritance diagram for Passer.Spawner:



Public Types

- enum SpawnMethod { SinglePlayer, Random, RoundRobin }
  Spawning methods.

Public Member Functions

- virtual GameObject SpawnObject ()
  Spawn one GameObject using the Spawner settings.

- void **DoSpawn** (GameObject prefab)
- virtual GameObject Spawn (GameObject prefab)
  Spawn a GameObject

- virtual GameObject Spawn (GameObject prefab, SpawnPoint[] spawnPoints, SpawnMethod spawnMethod=SpawnMethod.RoundRobin)
  Spawn a GameObject

Static Public Member Functions

- static GameObject **Spawn** (GameObject prefab, Vector3 position, Quaternion rotation)

Public Attributes

- GameObject **prefab**
  The prefab which will be spawned

- SpawnPoint[] spawnPoints
  The available SpawnPoints.

- SpawnMethod **spawnMethod**
  The SpawnMethod to use for spawning the humanoids.

- bool **spawnAtStart**
  Spawn an object when the scene starts

- bool singleInstance

Will spawn only one object for this prefab

Protected Member Functions

- virtual void **OnEnable** ()
- virtual void **Start** ()
- SpawnPoint **ChooseSpawnPoint** (SpawnPoint[] spawnPoints, SpawnMethod spawnMethod)

Protected Attributes

- int **spawnIndex** = 0

Member Enumeration Documentation

enum Passer.Spawner.SpawnMethod

Spawning methods.

Enumerator:

SinglePlayer	Only one humanoid will be spawned. It will be located at the first SpawnPoint.
Random	A SpawnPoint is chosen randomly.
RoundRobin	Spawn points are chosen in round robin manner.

Member Function Documentation

virtual GameObject Passer.Spawner.SpawnObject () `[virtual]`

Spawn one GameObject using the Spawner settings.

Returns

The instantiated GameObject. Will be null when the prefab could not be instantiated.

Reimplemented in Passer.Humanoid.HumanoidSpawner.

virtual GameObject Passer.Spawner.Spawn (GameObject *prefab*) `[virtual]`

Spawn a GameObject

Parameters

prefab	The GameObject prefab to spawn.

Returns

The instantiated GameObject. Will be null when the prefab could not be spawned.

virtual GameObject Passer.Spawner.Spawn (GameObject *prefab*, SpawnPoint[] *spawnPoints*, SpawnMethod *spawnMethod* = `SpawnMethod.RoundRobin`)`[virtual]`

Spawn a GameObject

Parameters

prefab	The GameObject prefab to spawn.
spawnPoints	The array of possible spawn points.
spawnMethod	The SpawnMethod to use for spawning.

Returns

The instantiated GameObject. Will be null when the prefab could not be spawned.

Member Data Documentation

SpawnPoint [] Passer.Spawner.spawnPoints

The available SpawnPoints.

When the list is empty, the scene will be searched for available spawn points when it becomes enabled. If no spawn points can be found, this transform will be used as a SpawnPoint

bool Passer.Spawner.singleInstance

Will spawn only one object for this prefab

When spawn is called a second time for the same prefab, the original spawned object will be teleported to the new spawn point

---

The documentation for this class was generated from the following file:

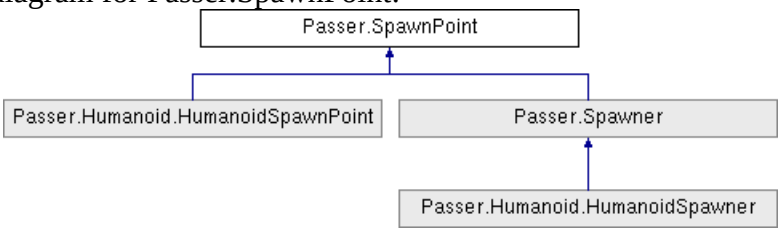
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/Spawner.cs

Passer.SpawnPoint Class Reference

Description

A Transform represeting a place where objects can spawn

Inheritance diagram for Passer.SpawnPoint:



The documentation for this class was generated from the following file:

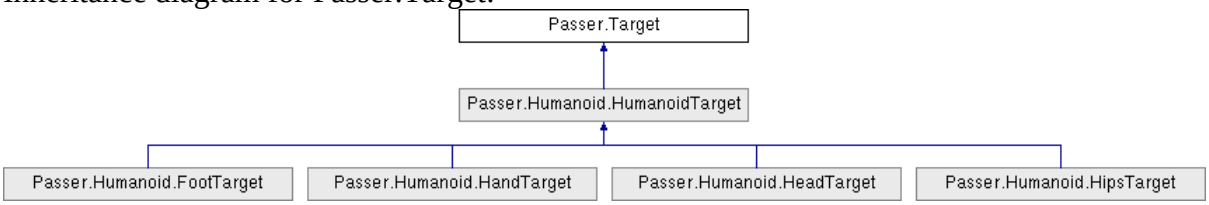
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/SpawnPoint.cs

Passer.Target Class Reference

Description

A main tracking transform

Inheritance diagram for Passer.Target:



Public Member Functions

- virtual void **InitComponent** ()
- abstract void **StartTarget** ()
- abstract void **InitSensors** ()
- virtual void **StartSensors** ()
- virtual void **StopSensors** ()
- abstract void InitializeTrackingConfidence ()
  Sets the tracking confidence before all tracking information is updated
- abstract void UpdateTarget ()

Static Public Member Functions

- static List< Collider > **SetColliderToTrigger** (GameObject obj)
- static List< Collider > **SetColliderToTrigger** (Rigidbody rb)
- static void **UnsetColliderToTrigger** (List< Collider > colliders)
- static void **UnsetColliderToTrigger** (List< Collider > colliders, Collider collider)

Protected Member Functions

- virtual void **UpdateSensors** ()

Protected Attributes

- bool **_showRealObjects** = true

Properties

- virtual bool **showRealObjects** `[getset]`
  show the target meshes

Member Function Documentation

abstract void Passer.Target.InitializeTrackingConfidence () `[pure virtual]`

Sets the tracking confidence before all tracking information is updated

Implemented in Passer.Humanoid.FootTarget, Passer.Humanoid.HandTarget, Passer.Humanoid.HeadTarget, and Passer.Humanoid.HipsTarget.

abstract void Passer.Target.UpdateTarget () `[pure virtual]`

Implemented in Passer.Humanoid.HeadTarget.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Targets/Target.cs

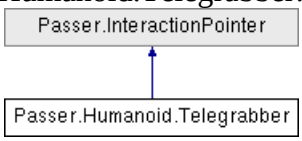
Passer.Humanoid.Telegrabber Class Reference

Description

Have a humaonid grab objects from a distance

The Humanoid Telegrabber enables humanoids to grab objects which are normally out of reach for the hands. This Interaction Pointer should be used on a child of a HandTarget.

Inheritance diagram for Passer.Humanoid.Telegrabber:



Public Types

- enum RayType { **Straight**, **Bezier**, **Gravity**, **SphereCast** }
  The ray modes for interaction pointers.
- enum PointerType { **FocusPoint**, **Ray** }
  The type of interaction pointer

Public Member Functions

- void GrabObject ()
  Grab the object currently in focus of the InteractionPointer

- void **PointTo** (GameObject obj)
- void **SetDirection** (Vector3 lookDirection)
- void **LaunchRigidbody** (Rigidbody rigidbody)
- void **LaunchPrefab** (GameObject prefab)
- void SpawnOnObjectInFocus (GameObject prefab)
  Spawns a prefab as a child of the objectInFocus

- void **ApplyForce** (float magnitude)
- void FullClick ()
  Click on the objectInFocus

- virtual void Click (bool clicking=true)
  Change the click status on the objectInFocus

- virtual void Activation (bool _active)
  Change the active status of the InteractionPointer

- virtual void **ActivateClick** (bool _active)

Static Public Member Functions

- static InteractionPointer Add (Transform parentTransform, PointerType pointerType=PointerType.Ray)
  Adds a default InteractionPointer to the transform

Public Attributes

- bool active = true
  Is the interaction pointer active?

- float timedClick
  Automatically initiates a click when the objectInFocus does not change for the set amount of seconds.

- GameObject focusPointObj
  The GameObject which represents the focus point of the Interaction Pointer when it is active.

- GameObject objectInFocus
  The object to which the Interaction Pointer is pointing at.

- float focusDistance
  Distance to the objectInFocus

- RayType **rayType** = RayType.Straight
  The ray mode for this interaction pointer.

- float maxDistance = 10
  The maximum length of the curve in units(meters)

- float resolution = 0.2F
  The size of a segment in the curve

- float speed = 3
  The horizontal speed for a gravity curve.

- float **radius** = 0.1F
  The radius of the sphere in a SphereCast

- GameObjectEventHandlers focusEvent
  Event based on the current focus.

- Vector3EventList focusPointEvent
  Event based on the current focus.

- GameObjectEventHandlers clickEvent
  Event based on the clicking status

- BoolEvent activeEvent = new BoolEvent()
  Event based on the active status

Protected Member Functions

- override void Awake ()

- virtual InteractionModule **CreateInteractionModule** ()
- virtual EventSystem **CreateEventSystem** ()
- virtual void **Start** ()
- virtual void **Update** ()
- virtual void **UpdateStraight** ()
- virtual void **UpdateBezier** ()
- virtual Vector3[] **UpdateBezierCurve** (Transform transform, float maxDistance, out Vector3 normal, out GameObject focusObject)
- Vector3 **GetPoint** (float t, Transform transform)
- Vector3 **GetVelocity** (float t, Transform transform)
- virtual void **UpdateGravity** ()
- virtual void **UpdateGravityCurve** (Transform transform, float forwardSpeed, out Vector3 normal, out GameObject hitObject)
- virtual void **UpdateSpherecast** ()
- void **UpdateFocus** ()
- void **UpdateFocusPoint** ()
- virtual void **OnDrawGizmosSelected** ()

Static Protected Member Functions

- static void **DebugLog** (string s)

Protected Attributes

- bool **hasClicked** = false
- InteractionModule **interactionModule**
- int **interactionID**
- InteractionModule.InteractionPointer **interactionPointer**
- LineRenderer **lineRenderer**
  The LineRender for this pointer when available.

- float **focusTimeToTouch** = 0
- float **focusStart** = 0
- float **heightLimitAngle** = 100f
- Vector3 **startPosition** = Vector3.zero
- Vector3 **intermediatePosition**
- Vector3 **endPosition**
- GameObject **previousObjectInFocus**

Member Function Documentation

override void Passer.Humanoid.Telegrabber.Awake () `[protected], [virtual]`

Reimplemented from Passer.InteractionPointer.

void Passer.Humanoid.Telegrabber.GrabObject ()

Grab the object currently in focus of the InteractionPointer

If current objectInFocus is a Rigidbody, this function will have the Humanoid hand try to grab the object.

static InteractionPointer Passer.InteractionPointer.Add (Transform *parentTransform*, PointerType *pointerType* = `PointerType.Ray`) `[static], [inherited]`

Adds a default InteractionPointer to the transform

Parameters

parentTransform	The transform to which the Teleporter will be added
pointerType	The interaction pointer type for the Teleporter.

void Passer.InteractionPointer.SpawnOnObjectInFocus (GameObject *prefab*) `[inherited]`

Spawns a prefab as a child of the objectInFocus

Parameters

prefab	The prefab to spawn

The spawn position and rotation will match the focusPoint transform. When the objectInFocus is null, the prefab will be spawned as a root transform.

void Passer.InteractionPointer.FullClick () `[inherited]`

Click on the objectInFocus

This function will do a full click: a button down followed by a button up.

virtual void Passer.InteractionPointer.Click (bool *clicking* = `true`) `[virtual], [inherited]`

Change the click status on the objectInFocus

Parameters

clicking	Indicates if the button is down

Reimplemented in Passer.Teleporter.

virtual void Passer.InteractionPointer.Activation (bool *_active*) `[virtual], [inherited]`

Change the active status of the InteractionPointer

Parameters

_active	Indicates if the InteractionPointer is active

Member Data Documentation

bool Passer.InteractionPointer.active = true `[inherited]`

Is the interaction pointer active?

When an interaction pointer is active, it will actively point to objects. The objectInFocus will be updated based on the pointer. When a focusPointObj is set, this object will be set active. When a Line Renderer is set, the line renderer will be updated according to the properties of the pointer.

float Passer.InteractionPointer.timedClick `[inherited]`

Automatically initiates a click when the objectInFocus does not change for the set amount of seconds.

The value 0 disables this function.

GameObject Passer.InteractionPointer.focusPointObj `[inherited]`

The GameObject which represents the focus point of the Interaction Pointer when it is active.

If this is set and the pointer is active, the object will be objected based on the pointer's properties This GameObject will be disabled when the pointer is not active.

GameObject Passer.InteractionPointer.objectInFocus `[inherited]`

The object to which the Interaction Pointer is pointing at.

This is updated at runtime while the pointer is active. The value is null when the pointer is not reaching any object.

float Passer.InteractionPointer.focusDistance `[inherited]`

Distance to the objectInFocus

This is float.infintiy when no object is in focus

float Passer.InteractionPointer.maxDistance = 10 `[inherited]`

The maximum length of the curve in units(meters)

This value is used for Straight, SphereCast and Bezier RayTypes

float Passer.InteractionPointer.resolution = 0.2F `[inherited]`

The size of a segment in the curve

Lower values will give a smoother curve, but requires more performance This value is used for Bezier and Gravity RayTypes

float Passer.InteractionPointer.speed = 3 `[inherited]`

The horizontal speed for a gravity curve.

A higher speed will reach further positions.

GameObjectEventHandlers Passer.InteractionPointer.focusEvent `[inherited]`

```
Initial value:= new GameObjectEventHandlers() {
            label = "Focus Event",
            tooltip =
```

```
                "Call functions using the current focus\n" +
                "Parameter: the Object in Focus"
        }
```

Event based on the current focus.

This event is generated from the objectInFocus. It has the objectInFocus as parameter. It can be used to execute functions when the focus changes between objects.

Vector3EventList Passer.InteractionPointer.focusPointEvent [inherited]

```
Initial value:= new Vector3EventList() {
            label = "Focus Point Event",
            tooltip =
                "Call functions using the current focus point\n" +
                "Parameter: the position of the focus point"
        }
```

Event based on the current focus.

This event is generated from the objectInFocus. It has the objectInFocus as parameter. It can be used to execute functions when the focus changes between objects.

GameObjectEventHandlers Passer.InteractionPointer.clickEvent [inherited]

```
Initial value:= new GameObjectEventHandlers() {
            label = "Click Event",
            tooltip =
                "Call functions using the clicking status\n" +
                "Parameter: the Object in Focus when clicking"
        }
```

Event based on the clicking status

This event is generated from the clicking boolean. It has the objectInFocus as parameter. It can be used to exectue functions when the user has clicked on an object.

BoolEvent Passer.InteractionPointer.activeEvent = new BoolEvent() [inherited]

Event based on the active status

This event is genereated from the active boolean Is has the active boolean as parameter. It can be used to execute functions when the interaction pointer is activated.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Tools/Telegrabber.cs

Passer.Teleporter Class Reference

Description

The teleporter is a simple tool to teleport transforms

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.

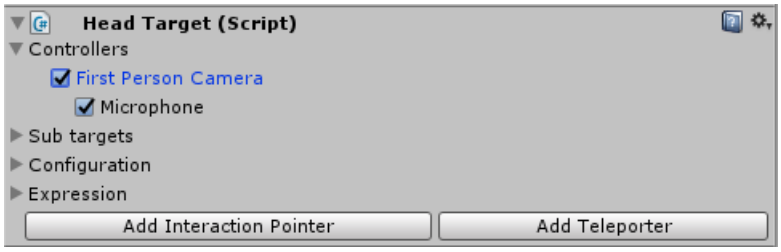
Setup

Two types of teleportation are supported:

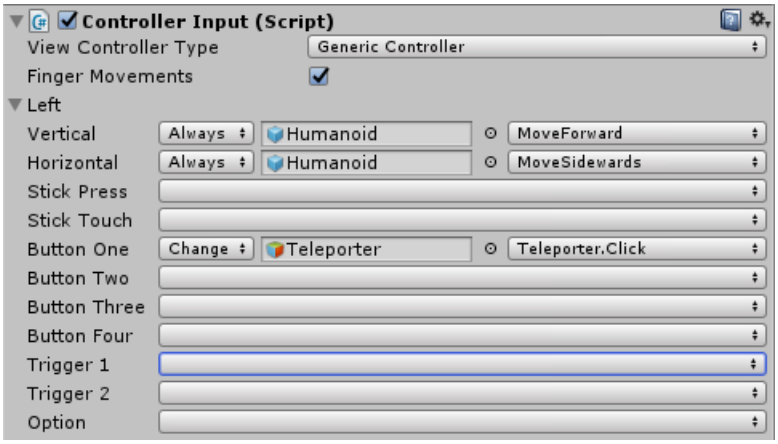
- Gaze based teleportation
- Hand pointing based teleportation

Gaze Teleportation

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



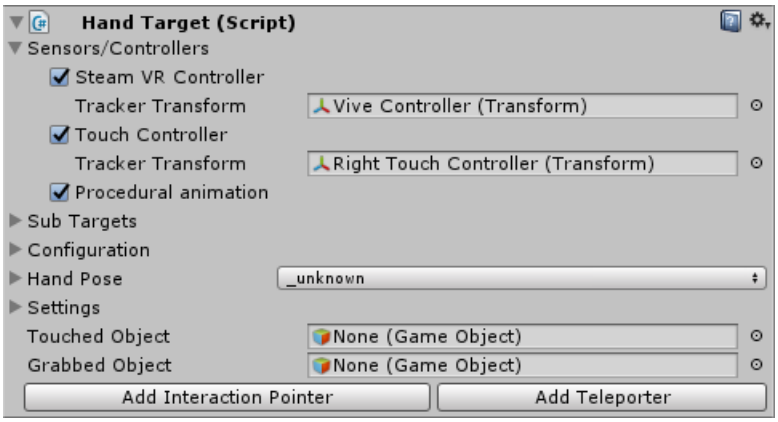
When this button is pressed, a Teleporter GameObject (see below) will be attached to the Head Target. It will be active by default so that you will see the focus point continuously when running the scene. The focus object is a simple sphere, while no line renderer is present. Additionally, the Left Button One of the controller will be set to the Click event of the Teleport which will teleport the humanoid to the location in focus:



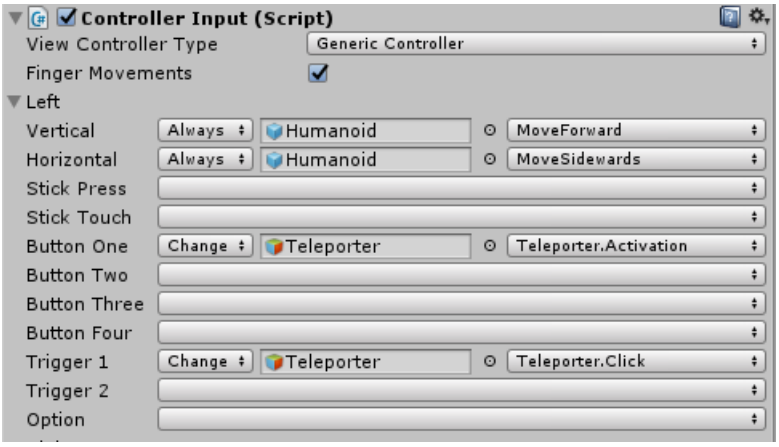
The Left Button One will match the Menu Button on the SteamVR Controller and the X button of the Left Oculus Touch Controller.

Pointing Teleportation

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

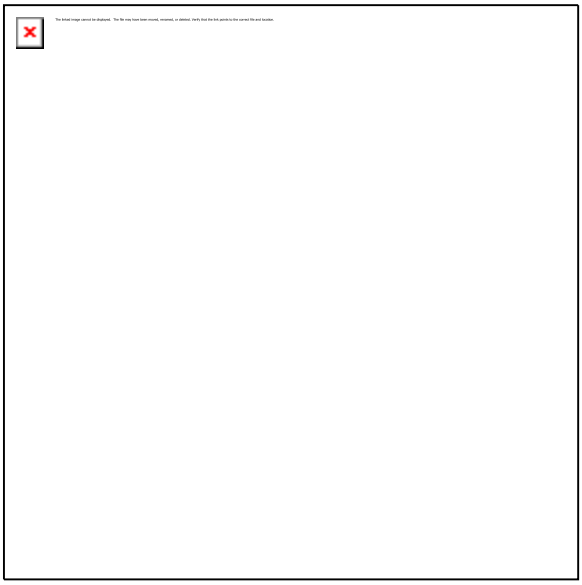


This will add a Teleporter GameObject to the Hand Target. In this case, no focus object is used, but an line renderer is used to show the pointing direction. This is visible as a standard pink ray on the hand. Pointing teleporting is activated when the 'One' button is pressed. While the Click event is 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.



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



- Active
- Timed teleport
- Target Point Object
- Mode
- Transport Type

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

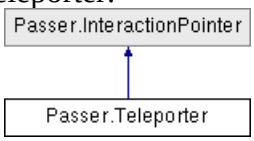
Target Point Object

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.

Inheritance diagram for Passer.Teleporter:



Public Types

- enum TransportType { **Teleport**, **Dash** }
  Determines how the Transform is moved to the Target Point.
- enum RayType { **Straight**, **Bezier**, **Gravity**, **SphereCast** }
  The ray modes for interaction pointers.
- enum PointerType { **FocusPoint**, **Ray** }
  The type of interaction pointer

Public Member Functions

- virtual void **TeleportTransform** ()
  Teleport the transform

- override void Click (bool clicking)
  Change the click status on the objectInFocus

- void **PointTo** (GameObject obj)
- void **SetDirection** (Vector3 lookDirection)
- void **LaunchRigidbody** (Rigidbody rigidbody)
- void **LaunchPrefab** (GameObject prefab)
- void SpawnOnObjectInFocus (GameObject prefab)
  Spawns a prefab as a child of the objectInFocus

- void **ApplyForce** (float magnitude)
- void FullClick ()
  Click on the objectInFocus

- virtual void Activation (bool _active)
  Change the active status of the InteractionPointer

- virtual void **ActivateClick** (bool _active)

Static Public Member Functions

- static new Teleporter Add (Transform parentTransform, PointerType pointerType=PointerType.Ray)
  Adds a default Teleporter to the transform

Public Attributes

- TransportType **transportType** = TransportType.Teleport
  The TransportType to use when teleporting.

- Transform **transformToTeleport**
  The transform which will be teleported

- bool active = true
  Is the interaction pointer active?

- float timedClick
  Automatically initiates a click when the objectInFocus does not change for the set amount of seconds.

- GameObject focusPointObj
  The GameObject which represents the focus point of the Interaction Pointer when it is active.

- GameObject objectInFocus
  The object to which the Interaction Pointer is pointing at.

- float focusDistance
  Distance to the objectInFocus

- RayType **rayType** = RayType.Straight
  The ray mode for this interaction pointer.

- float maxDistance = 10
  The maximum length of the curve in units(meters)

- float resolution = 0.2F
  The size of a segment in the curve

- float speed = 3
  The horizontal speed for a gravity curve.

- float **radius** = 0.1F
  The radius of the sphere in a SphereCast

- GameObjectEventHandlers focusEvent
  Event based on the current focus.

- Vector3EventList focusPointEvent
  Event based on the current focus.

- GameObjectEventHandlers clickEvent
  Event based on the clicking status

- BoolEvent activeEvent = new BoolEvent()
  Event based on the active status

Protected Member Functions

- override void Awake ()
- virtual InteractionModule **CreateInteractionModule** ()
- virtual EventSystem **CreateEventSystem** ()
- virtual void **Start** ()
- virtual void **Update** ()
- virtual void **UpdateStraight** ()
- virtual void **UpdateBezier** ()
- virtual Vector3[] **UpdateBezierCurve** (Transform transform, float maxDistance, out Vector3 normal, out GameObject focusObject)
- Vector3 **GetPoint** (float t, Transform transform)
- Vector3 **GetVelocity** (float t, Transform transform)
- virtual void **UpdateGravity** ()
- virtual void **UpdateGravityCurve** (Transform transform, float forwardSpeed, out Vector3 normal, out GameObject hitObject)
- virtual void **UpdateSpherecast** ()
- void **UpdateFocus** ()
- void **UpdateFocusPoint** ()

- virtual void **OnDrawGizmosSelected** ()

Static Protected Member Functions

- static void **DebugLog** (string s)

Protected Attributes

- HumanoidControl **humanoid**
- bool **hasClicked** = false
- InteractionModule **interactionModule**
- int **interactionID**
- InteractionModule.InteractionPointer **interactionPointer**
- LineRenderer **lineRenderer**
  The LineRender for this pointer when available.

- float **focusTimeToTouch** = 0
- float **focusStart** = 0
- float **heightLimitAngle** = 100f
- Vector3 **startPosition** = Vector3.zero
- Vector3 **intermediatePosition**
- Vector3 **endPosition**
- GameObject **previousObjectInFocus**

Member Function Documentation

override void Passer.Teleporter.Awake () `[protected], [virtual]`

Reimplemented from Passer.InteractionPointer.

static new Teleporter Passer.Teleporter.Add (Transform *parentTransform*, PointerType *pointerType* = `PointerType.Ray`) `[static]`

Adds a default Teleporter to the transform

Parameters

parentTransform	The transform to which the Teleporter will be added
pointerType	The interaction pointer type for the Teleporter

override void Passer.Teleporter.Click (bool *clicking*) `[virtual]`

Change the click status on the objectInFocus

Parameters

clicking	Indicates if the button is down

Reimplemented from Passer.InteractionPointer.

void Passer.InteractionPointer.SpawnOnObjectInFocus (GameObject *prefab*) `[inherited]`

Spawns a prefab as a child of the objectInFocus

Parameters

prefab	The prefab to spawn

The spawn position and rotation will match the focusPoint transform. When the objectInFocus is null, the prefab will be spawned as a root transform.

void Passer.InteractionPointer.FullClick () [inherited]

Click on the objectInFocus

This function will do a full click: a button down followed by a button up.

virtual void Passer.InteractionPointer.Activation (bool *_active*) [virtual], [inherited]

Change the active status of the InteractionPointer

Parameters

_active	Indicates if the InteractionPointer is active

Member Data Documentation

bool Passer.InteractionPointer.active = true [inherited]

Is the interaction pointer active?

When an interaction pointer is active, it will actively point to objects. The objectInFocus will be updated based on the pointer. When a focusPointObj is set, this object will be set active. When a Line Renderer is set, the line renderer will be updated according to the properties of the pointer.

float Passer.InteractionPointer.timedClick [inherited]

Automatically initiates a click when the objectInFocus does not change for the set amount of seconds.

The value 0 disables this function.

GameObject Passer.InteractionPointer.focusPointObj [inherited]

The GameObject which represents the focus point of the Interaction Pointer when it is active.

If this is set and the pointer is active, the object will be objected based on the pointer's properties This GameObject will be disabled when the pointer is not active.

GameObject Passer.InteractionPointer.objectInFocus [inherited]

The object to which the Interaction Pointer is pointing at.

This is updated at runtime while the pointer is active. The value is null when the pointer is not reaching any object.

float Passer.InteractionPointer.focusDistance [inherited]

Distance to the objectInFocus

This is float.infintiy when no object is in focus

float Passer.InteractionPointer.maxDistance = 10 [inherited]

The maximum length of the curve in units(meters)

This value is used for Straight, SphereCast and Bezier RayTypes

float Passer.InteractionPointer.resolution = 0.2F [inherited]

The size of a segment in the curve

Lower values will give a smoother curve, but requires more performance This value is used for Bezier and Gravity RayTypes

float Passer.InteractionPointer.speed = 3 [inherited]

The horizontal speed for a gravity curve.

A higher speed will reach further positions.

GameObjectEventHandlers Passer.InteractionPointer.focusEvent [inherited]

```
Initial value:= new GameObjectEventHandlers() {
            label = "Focus Event",
            tooltip =
                "Call functions using the current focus\n" +
                "Parameter: the Object in Focus"
        }
```

Event based on the current focus.

This event is generated from the objectInFocus. It has the objectInFocus as parameter. It can be used to execute functions when the focus changes between objects.

Vector3EventList Passer.InteractionPointer.focusPointEvent [inherited]

```
Initial value:= new Vector3EventList() {
            label = "Focus Point Event",
            tooltip =
                "Call functions using the current focus point\n" +
                "Parameter: the position of the focus point"
        }
```

Event based on the current focus.

This event is generated from the objectInFocus. It has the objectInFocus as parameter. It can be used to execute functions when the focus changes between objects.

GameObjectEventHandlers Passer.InteractionPointer.clickEvent [inherited]

```
Initial value:= new GameObjectEventHandlers() {
            label = "Click Event",
            tooltip =
                "Call functions using the clicking status\n" +
                "Parameter: the Object in Focus when clicking"
        }
```

Event based on the clicking status

This event is generated from the clicking boolean. It has the objectInFocus as parameter. It can be used to exectue functions when the user has clicked on an object.

BoolEvent Passer.InteractionPointer.activeEvent = new BoolEvent() `[inherited]`

Event based on the active status

This event is genereated from the active boolean Is has the active boolean as parameter. It can be used to execute functions when the interaction pointer is activated.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/Teleporter.cs

Passer.TeleportTarget Class Reference

Description

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

The 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.

Inherits EventTrigger.

Public Types

- enum **TargetPosRot** { **Pointer**, **Transform** }

Public Member Functions

- void TeleportToHere (Transform t)
  Teleport the transform to this Teleport Target.

- void TeleportToHere (HumanoidControl humanoid)
  Teleport the Humanoid to this Teleport Target.

- override void **OnPointerDown** (PointerEventData eventData)
  Teleporting initialized by an Unity UI Event.

Public Attributes

- bool teleportRoot = false
  Teleport the root transform instead of the transform itself.

- bool checkCollision = true
  Check the target location for collisions.

- MovementType movementType
  The type of movement used to reacht the teleport target.

- TargetPosRot targetPosRot = TargetPosRot.Pointer
  Determines the target position and rotation.

- Transform targetTransform
  The target transform for the teleport.

- Pose pose
  The pose of a humanoid after it has been teleported.

- bool enableAnimators = true
  Enables humanoid animators after teleporting.

- UnityEvent **unityEvents**

Protected Member Functions

- virtual void Teleport (Transform transform, Vector3 position, Quaternion rotation, MovementType movementType=MovementType.Teleport, Transform newParent=null)
  Teleport the transform to a position and rotation.

- virtual bool **IsValid** (Vector3 position)
  Check is the target position is valid.

Member Function Documentation

void Passer.TeleportTarget.TeleportToHere (Transform *t*)

Teleport the transform to this Teleport Target.

This function will teleport the given transform to this teleport target using the teleport target settings. If the targetPosRot is set to TargetPosRot.Pointer, the transform will be transported using the transform of the gameObject as there is no pointer information.

Parameters

t	The transform to teleport

void Passer.TeleportTarget.TeleportToHere (HumanoidControl *humanoid*)

Teleport the Humanoid to this Teleport Target.

This function will teleport the given humanoid to this teleport target using the teleport target settings. If the targetPosRot is set to TargetPosRot.Pointer, the humanoid will be transported using the transform of the gameObject as there is no pointer information.

Parameters

humanoid	The humanoid to teleport

virtual void Passer.TeleportTarget.Teleport (Transform *transform*, Vector3 *position*, Quaternion *rotation*, MovementType *movementType* = `MovementType.Teleport`, Transform *newParent* = `null`) `[protected], [virtual]`

Teleport the transform to a position and rotation.

This function will teleport the given transform to this teleport target using the teleport target settings. If the targetPosRot is set to TargetPosRot.Pointer, the transform will be transported using the transform of the gameObject as there is no pointer information.

Parameters

transform	The transform to teleport
position	The new world position of the transform after teleporting
rotation	The new world rotation of the transform after teleporting
movementType	The movement type to use to get to the target
newParent	The new parent of the transform after teleporting

Member Data Documentation

bool Passer.TeleportTarget.teleportRoot = false

Teleport the root transform instead of the transform itself.

If this is enabled the root of the transform (the topmost transform) will be teleported instead of the transform itself.

bool Passer.TeleportTarget.checkCollision = true

Check the target location for collisions.

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.

MovementType Passer.TeleportTarget.movementType

The type of movement used to reacht the teleport target.

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.

TargetPosRot Passer.TeleportTarget.targetPosRot = TargetPosRot.Pointer

Determines the target position and rotation.

TargetPosRot.Pointer = The interaction pointer location is the target position and rotation. TargetPosRot.Transform = The targetTransform determines the target position and rototation.

Transform Passer.TeleportTarget.targetTransform

The target transform for the teleport.

If targetPosRot is set to TargetPosRot.Transform the targetTransform will determine the position and rotation of the transform to teleport. Next to that, the teleported transform will become a child of this teleportTarget after teleporting

Pose Passer.TeleportTarget.pose

The pose of a humanoid after it has been teleported.

This 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.

bool Passer.TeleportTarget.enableAnimators = true

Enables humanoid animators after teleporting.

This will enable or disable the hips and 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.

The documentation for this class was generated from the following file:

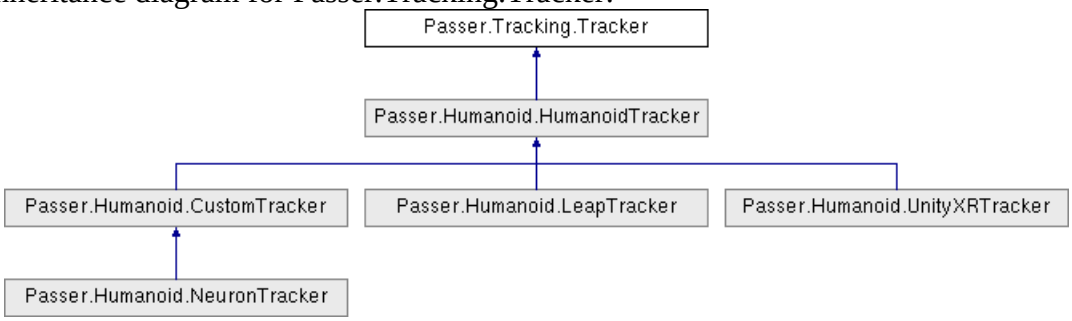
- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/TeleportTarget.cs

Passer.Tracking.Tracker Class Reference

Description

A tracker

Inheritance diagram for Passer.Tracking.Tracker:



Public Types

- enum Status { Unavailable, Present, Tracking }
  The tracking status

Public Member Functions

- virtual void StartTracker ()
  Optional list of SubTrackers

- virtual void StopTracker ()
  Stop the tracker

- virtual void UpdateTracker ()
  Update the tracker state

- virtual void ShowTracker (bool shown)
  Show or hide the Tracker renderers

- virtual void Calibrate ()
  Calibrate the tracker

- virtual void AdjustTracking (Vector3 positionalDelta, Quaternion rotationalDelta)
  Adjust the position of the tracker by the given delat

Public Attributes

- bool **enabled**
  Is this tracker enabled?

- Status **status**
  The tracking Status of the tracker

- TrackerComponent **trackerComponent**
  The tracking device

Properties

- virtual string **name** `[get]`
  The name of this tracker

Member Enumeration Documentation

enum Passer.Tracking.Tracker.Status

The tracking status

Enumerator:

Unavailable	The tracking device is not available.
Present	The tracking device is available but not tracking.
Tracking	The tracking device is actively tracking.

Member Function Documentation

virtual void Passer.Tracking.Tracker.StartTracker () `[virtual]`

Optional list of SubTrackers

Start the tracker

virtual void Passer.Tracking.Tracker.StopTracker () `[virtual]`

Stop the tracker

Reimplemented in Passer.Humanoid.LeapTracker.

virtual void Passer.Tracking.Tracker.UpdateTracker () `[virtual]`

Update the tracker state

Reimplemented in Passer.Humanoid.UnityXRTracker, Passer.Humanoid.LeapTracker, Passer.Humanoid.NeuronTracker, and Passer.Humanoid.CustomTracker.

virtual void Passer.Tracking.Tracker.ShowTracker (bool *shown*) `[virtual]`

Show or hide the Tracker renderers

Parameters

shown	Renderers are enabled when shown == true

virtual void Passer.Tracking.Tracker.Calibrate () `[virtual]`

Calibrate the tracker

Reimplemented in Passer.Humanoid.UnityXRTracker.

virtual void Passer.Tracking.Tracker.AdjustTracking (Vector3 *positionalDelta*, Quaternion *rotationalDelta*) `[virtual]`

Adjust the position of the tracker by the given delat

Parameters

positionalDelta	The positional delta to apply
rotationalDelta	The rotational delta to apply

The documentation for this class was generated from the following file:

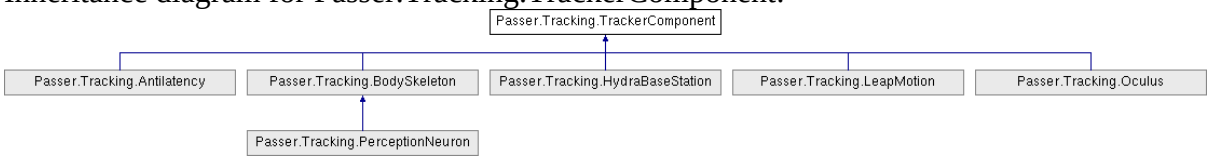
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/Tracker.cs

Passer.Tracking.TrackerComponent Class Reference

Description

Generic Tracking device

Inheritance diagram for Passer.Tracking.TrackerComponent:



Public Member Functions

- virtual void **ShowSkeleton** (bool shown)

Public Attributes

- Tracker.Status **status**
  The status of the tracking device

Protected Member Functions

- virtual void **Start** ()
- virtual void **Update** ()

Protected Attributes

- Transform **realWorld**

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/TrackerComponent.cs

Passer.Transportation Class Reference

Description

Utility functions to move transformes in the world.

Inherits MonoBehaviour.

Public Member Functions

- virtual void MoveForward (float z)
  Moves the transform along the local Z-axis with the indicated speed

- virtual void MoveSideward (float x)
  Moves the transform along the local X-axis with the indicated speed

- virtual void Rotate (float angularSpeed)
  Rotates this transform with the indicated speed around the Y axis.

- void Dash (Vector3 targetPosition)
  Quickly moves this transform to the targetPosition

- void Teleport (Vector3 targetPosition)
  Teleport this transform to the targetPosition

- void **Teleport** ()
  Teleport this transform forward by 1 meter

Public Attributes

- float **forwardSpeed** = 1
- float **sidewardSpeed** = 1
- float **rotationalSpeed** = 60

Member Function Documentation

virtual void Passer.Transportation.MoveForward (float *z*) `[virtual]`

Moves the transform along the local Z-axis with the indicated speed

Should be called every frame for continuous movement.

Parameters

z	

virtual void Passer.Transportation.MoveSideward (float *x*) `[virtual]`

Moves the transform along the local X-axis with the indicated speed

Should be called every frame for continuous movement.

Parameters

x	

virtual void Passer.Transportation.Rotate (float *angularSpeed*) [virtual]

Rotates this transform with the indicated speed around the Y axis.

Should be called every frame for continuous rotation.

Parameters

angularSpeed	The speed in degrees per second

void Passer.Transportation.Dash (Vector3 *targetPosition*)

Quickly moves this transform to the targetPosition

This is a coroutine which will move the transform using DashCoroutine It does not need to be called every frame.

Parameters

targetPosition	

void Passer.Transportation.Teleport (Vector3 *targetPosition*)

Teleport this transform to the targetPosition

Parameters

targetPosition	The target position

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Scripts/Transportation.cs

Passer.TriggerEventHandler Class Reference

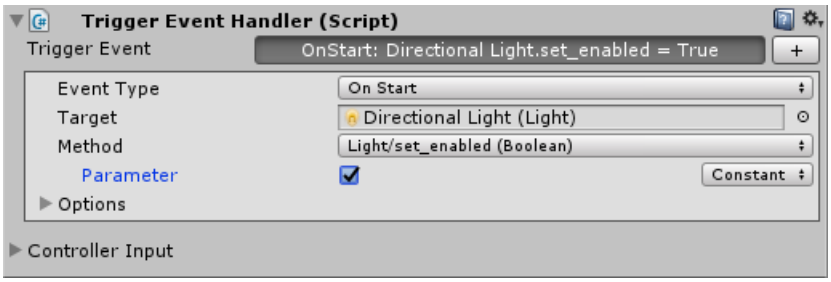
Description

Implements input behaviour using Trigger Colliders

The Trigger Event Handler is a convenience component to act on trigger events without programmer. It is a specific implementation of an EventHandler. It requires a Collider to be attached to the GameObject. It differs from the Collision Event Handler in that it uses trigger events and can also change ControllerInput.

The Event

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



The event type is as follows:

- Never: the event handler is disabled, the Target Method is never called.
- On Trigger Enter: the Target Method is called when the trigger collider is entered by a collider. This is equivalent to the OnTriggerEnter() function of Unity.
- On Trigger Exit: the Target Method is called when the trigger is exited by a collider. This is equivalent to the OnTriggerExit() function of Unity.
- On Trigger Stay: 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 **Error! Hyperlink reference not valid.** function of Unity.
- On Trigger Empty: The Target Method is called while no collider is in the trigger collider. In that case it will be called in every frame.
- On Trigger Change: The Target Method is called when a collider enters or exits the trigger collider.
- Always 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 integer is 1 when the collider is touching the trigger collider and 0 when not.

Integer

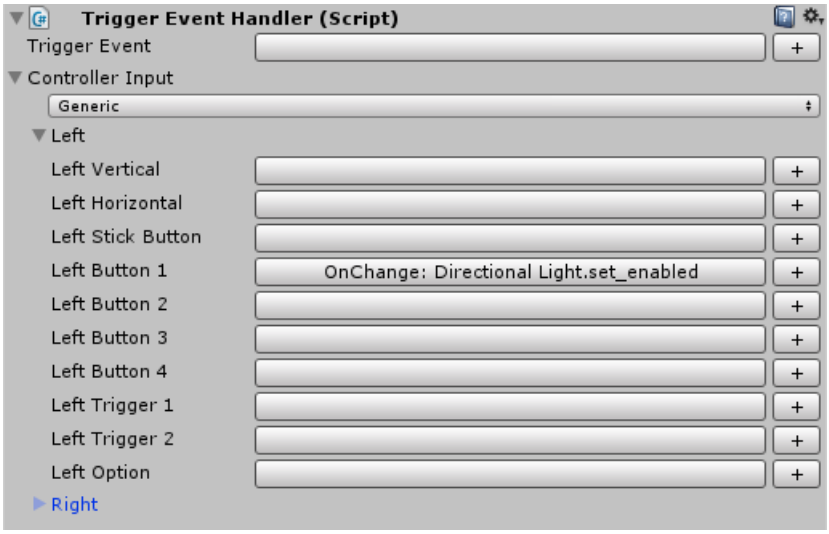
When the Target Method takes an integer (Int32) parameter, a Constant value can be used or the parameter can be set to From Event. When the parameter is from the event, the integer is 1 when the collider is touching the trigger collider and 0 when not.

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.



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.

Inherits MonoBehaviour.

Public Attributes

- GameObjectEventHandlers triggerHandlers
  Trigger Event Handles

- ControllerEventHandlers[] leftInputEvents
- ControllerEventHandlers[] rightInputEvents

Protected Member Functions

- virtual void **OnTriggerEnter** (Collider other)
- virtual void **OnTriggerExit** (Collider other)

Protected Attributes

- bool **entered** = false

Member Data Documentation

GameObjectEventHandlers Passer.TriggerEventHandler.triggerHandlers

```
Initial value:= new GameObjectEventHandlers() {
            label = "Trigger Event",
            tooltip =
                "Call functions using the trigger collider state\n" +
                "Parameter: the GameObject entering the trigger",
            eventTypeLabels = new string[] {
                "Never",
                "On Trigger Enter",
                "On Trigger Exit",
                "On Trigger Stay",
                "On Trigger Empty",
                "On Trigger Change",
                "Always"
            },
        }
```

Trigger Event Handles

Let you execute function calls based on the Trigger Events

ControllerEventHandlers [] Passer.TriggerEventHandler.leftInputEvents

```
Initial value:= {
            new ControllerEventHandlers() { label = "Left Vertical", id = 0 },
            new ControllerEventHandlers() { label = "Left Horizontal", id = 1 },
            new ControllerEventHandlers() { label = "Left Stick Button", id = 2 },
            new ControllerEventHandlers() { label = "Left Button 1", id = 3 },
            new ControllerEventHandlers() { label = "Left Button 2", id = 4 },
            new ControllerEventHandlers() { label = "Left Button 3", id = 5 },
            new ControllerEventHandlers() { label = "Left Button 4", id = 6 },
            new ControllerEventHandlers() { label = "Left Trigger 1", id = 7 },
            new ControllerEventHandlers() { label = "Left Trigger 2", id = 8 },
            new ControllerEventHandlers() { label = "Left Option", id = 9 },
        }
```

ControllerEventHandlers [] Passer.TriggerEventHandler.rightInputEvents

```
Initial value:= {
            new ControllerEventHandlers() { label = "Right Vertical", id = 0 },
            new ControllerEventHandlers() { label = "Right Horizontal", id = 1 },
            new ControllerEventHandlers() { label = "Right Stick Button", id = 2 },
            new ControllerEventHandlers() { label = "Right Button 1", id = 3 },
            new ControllerEventHandlers() { label = "Right Button 2", id = 4 },
            new ControllerEventHandlers() { label = "Right Button 3", id = 5 },
            new ControllerEventHandlers() { label = "Right Button 4", id = 6 },
            new ControllerEventHandlers() { label = "Right Trigger 1", id = 7 },
            new ControllerEventHandlers() { label = "Right Trigger 2", id = 8 },
            new ControllerEventHandlers() { label = "Right Option", id = 9 },
        }
```

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Physics/TriggerEventHandler.cs

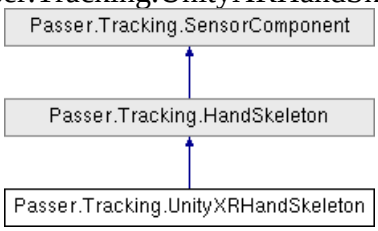
Passer.Tracking.UnityXRHandSkeleton Class Reference

Description

HandSkeleton component for hand tracking with UnityXR

Important note: although the Unity XR SDK implements this interface no known device is supporting it at the moment. The added value of this component is therefore limited. Also: because of this limitation this component is untested! See: https://forum.unity.com/threads/oculus-quest-hand-tracking-data.853579/

Inheritance diagram for Passer.Tracking.UnityXRHandSkeleton:



Public Types

- enum **BoneId** { **Invalid** = -1, **Hand** = 0, **ThumbProximal**, **ThumbIntermediate**, **ThumbDistal**, **ThumbTip**, **IndexMetaCarpal**, **IndexProximal**, **IndexIntermediate**, **IndexDistal**, **IndexTip**, **MiddleMetaCarpal**, **MiddleProximal**, **MiddleIntermediate**, **MiddleDistal**, **MiddleTip**, **RingMetaCarpal**, **RingProximal**, **RingIntermediate**, **RingDistal**, **RingTip**, **LittleMetacarpal**, **LittleProximal**, **LittleIntermediate**, **LittleDistal**, **LittleTip**, **Forearm**, **Count** }

Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- virtual GameObject **CreateHandSkeleton** (Transform trackerTransform, bool isLeft, bool showRealObjects)
- virtual Transform **GetForearmBone** ()
- virtual Transform **GetWristBone** ()
- virtual Transform GetBoneTransform (Finger finger, FingerBone fingerBone)
  Gets the transform of the tracked bone

- virtual int **GetBoneId** (Finger finger, FingerBone fingerBone)
- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Static Public Member Functions

- static HandSkeleton **FindHandSkeleton** (Transform trackerTransform, bool isLeft)

Public Attributes

- TrackerComponent **tracker**
- bool **isLeft**
  True: this is a left hand. Fale: this is a right hand

- new bool **show** = false

Determines whether this skeleton should be rendered

- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- override void Start ()
  Starts the sensor

- virtual void OnDeviceConnected (InputDevice device)
  Hand is detected?

- virtual void OnDeviceDisconnected (InputDevice device)
  Hand is lost

- override void InitializeSkeleton ()
  This function is used to initialize the tracked bones

- void **UpdateFinger** (ref int boneIx, Hand handData, HandFinger fingerId)
- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()
- virtual void Awake ()
  Initializes the sensor.

Protected Attributes

- InputDevice **device**
- XRNode **xrNode**
- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.UnityXRHandSkeleton.Start () `[protected], [virtual]`

Starts the sensor

Does nothing at this moment.

Reimplemented from Passer.Tracking.SensorComponent.

virtual void Passer.Tracking.UnityXRHandSkeleton.OnDeviceConnected (InputDevice *device*) `[protected], [virtual]`

Hand is detected?

Parameters

device	The InputDevice of the hand

virtual void Passer.Tracking.UnityXRHandSkeleton.OnDeviceDisconnected (InputDevice *device*) `[protected], [virtual]`

Hand is lost

This also happens when the hand is no longer tracked.

Parameters

device	The InputDevice of the hand

override void Passer.Tracking.UnityXRHandSkeleton.InitializeSkeleton () `[protected], [virtual]`

This function is used to initialize the tracked bones

Reimplemented from Passer.Tracking.HandSkeleton.

override void Passer.Tracking.UnityXRHandSkeleton.UpdateComponent () `[virtual]`

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

virtual Transform Passer.Tracking.HandSkeleton.GetBoneTransform (Finger *finger*, FingerBone *fingerBone*) `[virtual], [inherited]`

Gets the transform of the tracked bone

Parameters

finger	The requested finger
fingerBone	The requested bone in the finger

Returns

The tracked bon transform of *null* if it does not exist

Reimplemented in Passer.Tracking.LeapHandSkeleton.

virtual void Passer.Tracking.SensorComponent.Awake () `[protected], [virtual], [inherited]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) `[virtual], [inherited]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true `[inherited]`

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/UnityXR/UnityXRHandSkeleton.cs

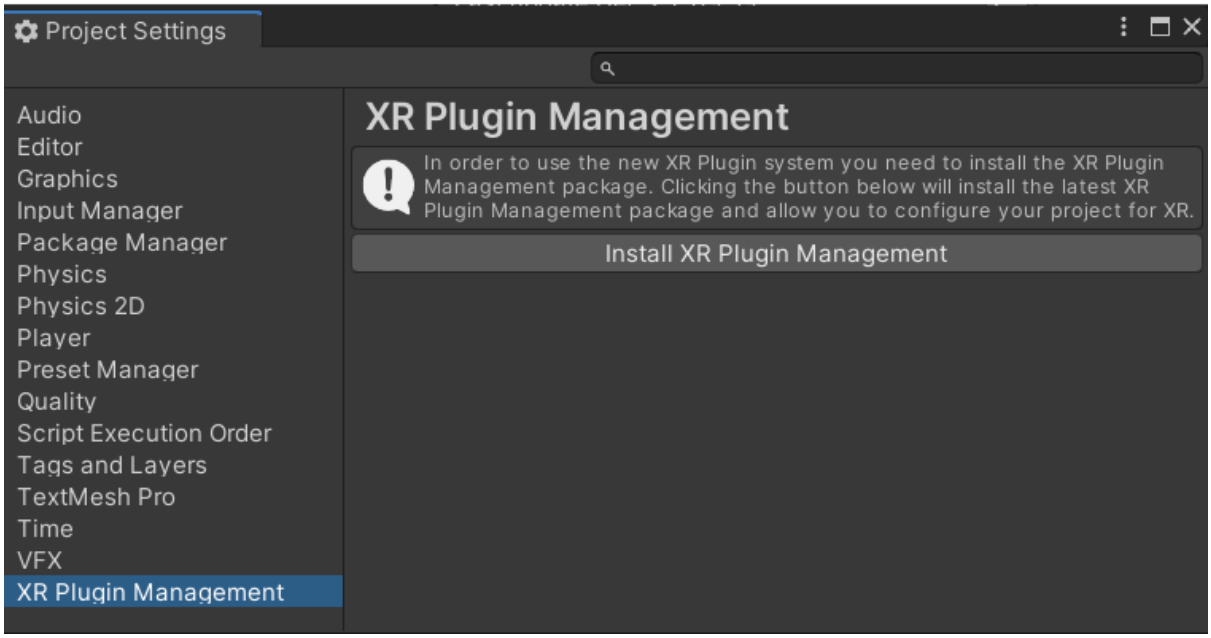
Passer.Humanoid.UnityXRTracker Class Reference

Description

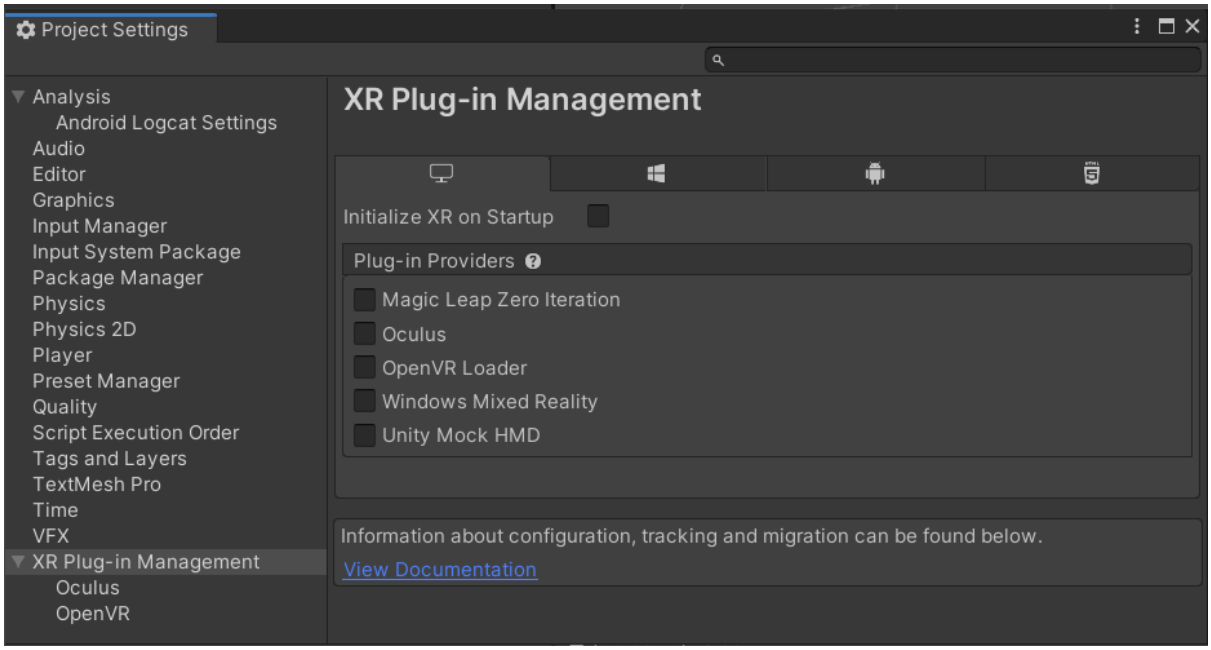
Universal API for tracking XR devices.

Setup

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

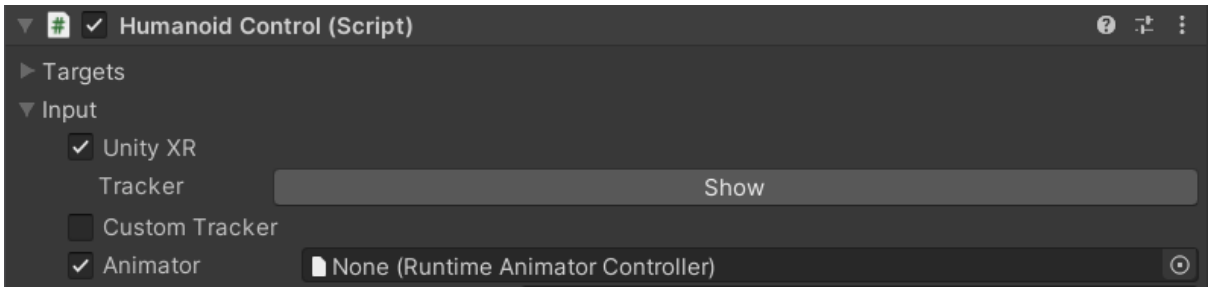


Then enable the desired XR Plugin. For example the Oculus plugin for Android: Note that if you want to test Oculus Quest in the editor, the Oculus Plugin for Standalone needs to be enabled too:

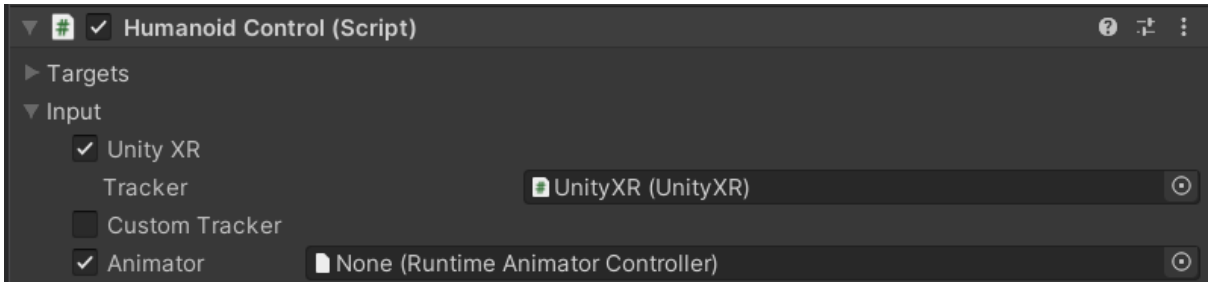


Configuration

To enable body tracking with Unity XR for a humanoid, *Unity XR* needs to be enabled in the HumanoidControl component:



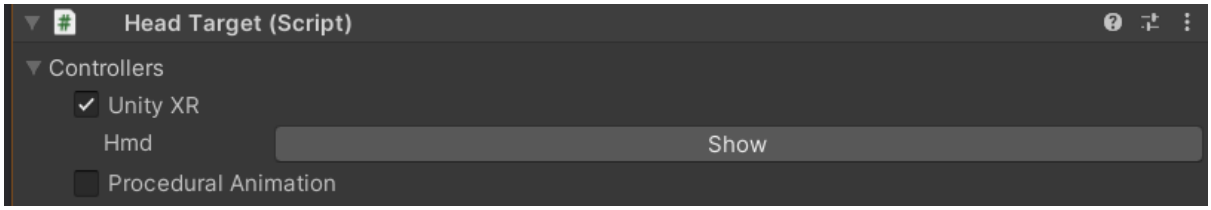
By default the UnityXR object is not visible in the scene, but it will be created automatically when the scene starts. If the button *Show* is pressed, the UnityXR object will be created in the *Real World* object.



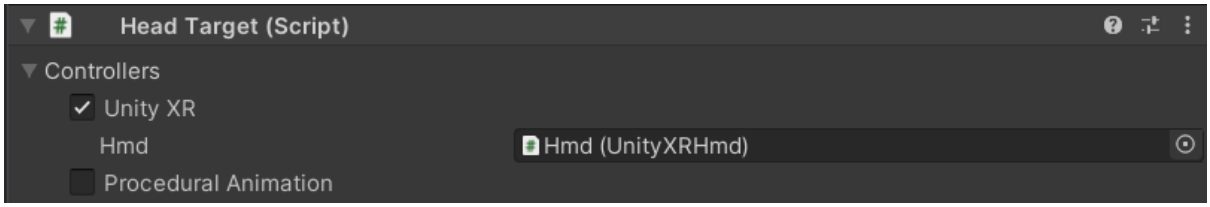
UnityXR (UnityXR) is a reference to the object in the scene representing the root of the Unit XR tracking space. This GameObject is found as a child of the *Real World* GameObject. The UnityXR GameObject can be used to adjust the origin of the Unity XR tracking space.

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:

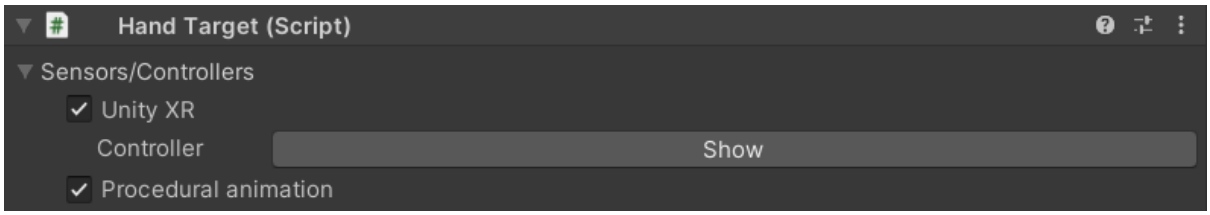


By default the UnityXRHmd object is not visible in the scene, but it will be created automatically when the scene starts. The UnityXRHmd will also have the main camera attached. You can disable the camera attached to the *UnityXRHead* object if needed, for example when you want to see the avatar's movements in third person view. If the button *Show* is pressed, the UnityXRHead object will be created as a child of the *UnityXR* object in the *Real World* .

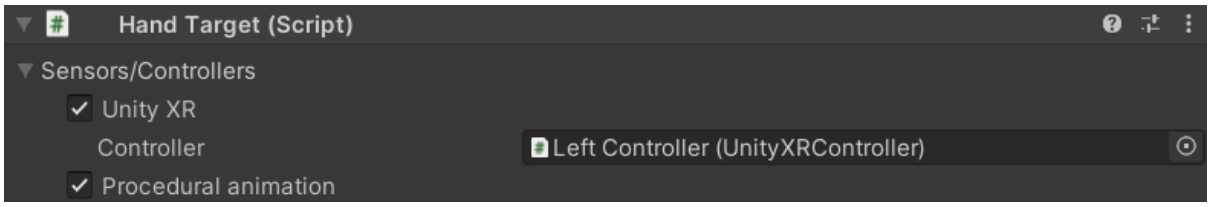


Hand Target

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



Like with the Head Target, the UnityXRController object is not visible in the editor scene by default but it will be created automatically when the scene starts. If the button *Show* is pressed, the UnityXRController object will be creates as a child of the *UnityXR* object in the *Real World* .

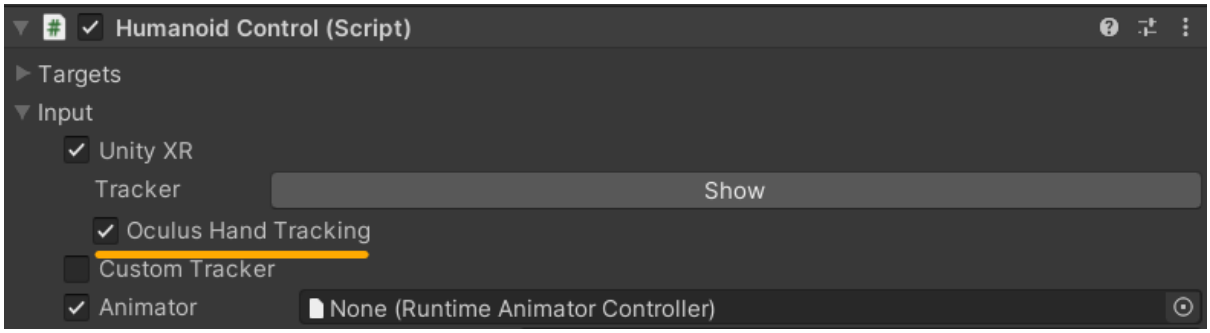


Hand Tracking (Plus & Pro)

Some devices support hand tracking in combination with UnityXR. Native hand tracking with Unity XR is still not possible, so we provide specific implementations for the following hand tracking options

Oculus Quest

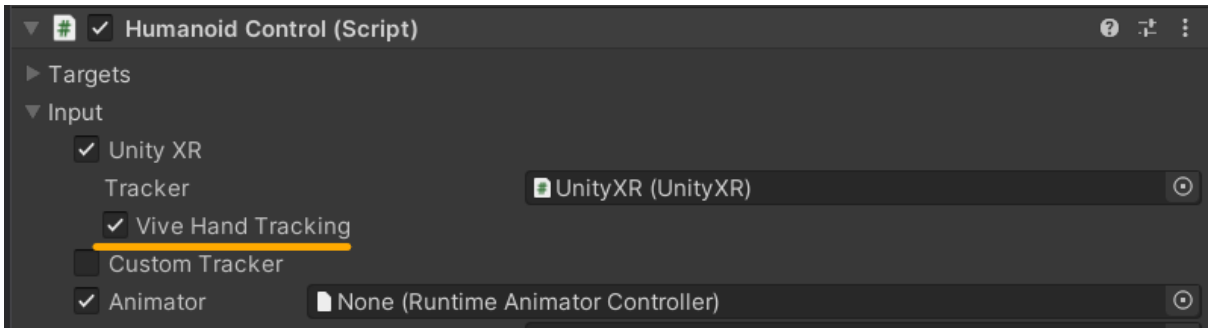
When the Android platform is selected in Unity, an additional option for Oculus Hand Tracking is shown:



When enabled, hand tracking will be possible for this humanoid.

HTC Vive

For Vive hand tracking, the Vive Hand Tracking SDK version 0.9 or higher is required. You can download it here: Vive Hand Tracking SDK. When the HTC Vive Hand Tracking SDK is imported in the project on the Windows Standalone platform, an option for HTC Vive Hand Tracking is shown:



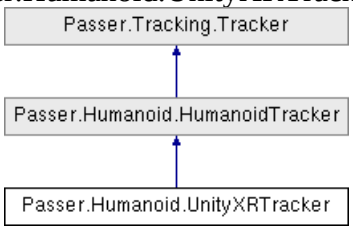
For Vive Hand tracking, the OpenVR Plugin for UnityXR is needed. OpenXR does not work with Vive Hand Tracking in version 1.0.0 of the SDK. This may be fixed in more recent versions. The OpenVR Plugin is automatically installed when the SteamVR Plugin from the Asset Store is imported in the project. HTC Vive headsets like Vive Pre, Vive Pro and normal Vive are supported. HTC Vive Pro 2 is untested, but may work.

Note: if you get an error in the *ViveSkeleton.cs* code stating that *ViveHandTracking* is not defined, you need to import the *ViveHandTracking/ViveHandTracking.asmdef* from the Humanoid Control package.

See also

UnityXRHead UnityXRHand Tracking::UnityXR Tracking::UnityXRHmd Passer::Tracking::UnityXRController

Inheritance diagram for Passer.Humanoid.UnityXRTracker:



Public Types

- enum Status { Unavailable, Present, Tracking }
  The tracking status

Public Member Functions

- override void CheckTracker (HumanoidControl humanoid)
  Check the status of the tracker

- override void StartTracker (HumanoidControl humanoid)
  Start the tracker

- override void UpdateTracker ()
  Update the tracker state

- override void Calibrate ()
  Calibrate the tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker)
  Function to check the status of a specific tracker

- void CheckTracker (HumanoidControl humanoid, TrackerGetter getTracker, Vector3 localPosition, Quaternion localRotation)
  Function to check the status of a specific tracker

- delegate TrackerComponent TrackerGetter (Transform transform, Vector3 localPosition, Quaternion localRotation)
  Function delegate for retrieving the tracker

- virtual Vector3 **GetBonePosition** (uint actorId, FacialBone boneId)
- virtual Quaternion **GetBoneRotation** (uint actorId, FacialBone boneId)
- virtual float **GetBoneConfidence** (uint actorId, FacialBone boneId)
- virtual void StartTracker ()
  Optional list of SubTrackers

- virtual void StopTracker ()
  Stop the tracker

- virtual void ShowTracker (bool shown)
  Show or hide the Tracker renderers

- virtual void AdjustTracking (Vector3 positionalDelta, Quaternion rotationalDelta)
  Adjust the position of the tracker by the given delat

Public Attributes

- bool **oculusHandTracking** = true
  Enables hand tracking on the Oculus Quest

- bool **viveHandTracking** = true
  Enables hand tracking on the HTC Vive

- HumanoidControl **humanoid**

The humanoid for this tracker

- bool **enabled**
  Is this tracker enabled?

- Status **status**
  The tracking Status of the tracker

- TrackerComponent **trackerComponent**
  The tracking device

Properties

- override string name `[get]`
  The name of this tracker

- override HeadSensor headSensor `[get]`
  Get the sensor for the head

- override ArmSensor leftHandSensor `[get]`
  Get the sensor for the left hand

- override ArmSensor rightHandSensor `[get]`
  Get the sensor for the right hand

- override HumanoidSensor[] **sensors** `[get]`
- virtual TorsoSensor hipsSensor `[get]`
  Get the sensor for the hips

- virtual LegSensor leftFootSensor `[get]`
  Get the sensor for the left foot

- virtual LegSensor rightFootSensor `[get]`
  Get the sensor for the right foot

Member Enumeration Documentation

enum Passer.Tracking.Tracker.Status `[inherited]`

The tracking status

Enumerator:

Unavailable	The tracking device is not available.

Present	The tracking device is available but not tracking.
Tracking	The tracking device is actively tracking.

Member Function Documentation

override void Passer.Humanoid.UnityXRTracker.CheckTracker (HumanoidControl *humanoid*) `[virtual]`

Check the status of the tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked

Reimplemented from Passer.Humanoid.HumanoidTracker.

override void Passer.Humanoid.UnityXRTracker.StartTracker (HumanoidControl *_humanoid*) `[virtual]`

Start the tracker

Reimplemented from Passer.Humanoid.HumanoidTracker.

override void Passer.Humanoid.UnityXRTracker.UpdateTracker () `[virtual]`

Update the tracker state

Reimplemented from Passer.Tracking.Tracker.

override void Passer.Humanoid.UnityXRTracker.Calibrate () `[virtual]`

Calibrate the tracker

Reimplemented from Passer.Tracking.Tracker.

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*) `[inherited]`

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker

The default position/rotation for the tracker when created will be zero

void Passer.Humanoid.HumanoidTracker.CheckTracker (HumanoidControl *humanoid*, TrackerGetter *getTracker*, Vector3 *localPosition*, Quaternion *localRotation*) `[inherited]`

Function to check the status of a specific tracker

Parameters

humanoid	The humanoid for which the tracker needs to be checked
getTracker	Function delegate to retrieve the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

delegate TrackerComponent Passer.Humanoid.HumanoidTracker.TrackerGetter (Transform *transform*, Vector3 *localPosition*, Quaternion *localRotation*) `[inherited]`

Function delegate for retrieving the tracker

Parameters

transform	The root transform to start the searching of the tracker
localPosition	The default local position of the tracker
localRotation	The default local rotation of the tracker

Returns

The tracker component found or created

The default position/rotation is relative to the humanoid's real world.

virtual void Passer.Tracking.Tracker.StartTracker () `[virtual], [inherited]`

Optional list of SubTrackers

Start the tracker

virtual void Passer.Tracking.Tracker.StopTracker () `[virtual], [inherited]`

Stop the tracker

Reimplemented in Passer.Humanoid.LeapTracker.

virtual void Passer.Tracking.Tracker.ShowTracker (bool *shown*) `[virtual], [inherited]`

Show or hide the Tracker renderers

Parameters

shown	Renderers are enabled when shown == true

virtual void Passer.Tracking.Tracker.AdjustTracking (Vector3 *positionalDelta*, Quaternion *rotationalDelta*) `[virtual], [inherited]`

Adjust the position of the tracker by the given delat

Parameters

positionalDelta	The positional delta to apply

rotationalDelta	The rotational delta to apply

Property Documentation

override string Passer.Humanoid.UnityXRTracker.name [get]

The name of this tracker

override HeadSensor Passer.Humanoid.UnityXRTracker.headSensor [get]

Get the sensor for the head

Will return null when this sensor is not present

override ArmSensor Passer.Humanoid.UnityXRTracker.leftHandSensor [get]

Get the sensor for the left hand

Will return null when this sensor is not present

override ArmSensor Passer.Humanoid.UnityXRTracker.rightHandSensor [get]

Get the sensor for the right hand

Will return null when this sensor is not present

virtual TorsoSensor Passer.Humanoid.HumanoidTracker.hipsSensor [get], [inherited]

Get the sensor for the hips

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.leftFootSensor [get], [inherited]

Get the sensor for the left foot

Will return null when this sensor is not present

virtual LegSensor Passer.Humanoid.HumanoidTracker.rightFootSensor [get], [inherited]

Get the sensor for the right foot

Will return null when this sensor is not present

The documentation for this class was generated from the following file:

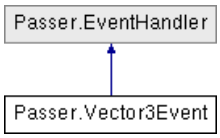
- Assets/Passer/HumanoidControl/Runtime/HumanoidControl/Scripts/Extensions/UnityXR/UnityXRTracker.cs

Passer.Vector3Event Class Reference

Description

An event Handler calling a function with a Vector3 parameter

Inheritance diagram for Passer.Vector3Event:



Public Types

- enum Type { Never, OnStart, OnEnd, WhileActive, WhileInactive, OnChange, Continuous }
  The different types of events when the function is called
- enum OverrideMode { Prepend, Append, Replace }

Public Member Functions

- **Vector3Event** (Type newEventType=Type.OnChange)
- virtual void **Update** ()

Public Attributes

- Type **eventType** = Type.Continuous
  The event type for the function call

- bool **eventNetworking** = false
  For future use :-)

- FunctionCall **functionCall**
  The function to call

- bool **boolInverse** = false
  Negate the boolean state before calling event trigger

- OverrideMode **overrideMode**

Protected Member Functions

- override void UpdateVector3 ()
- void **UpdateAnimationParameter** ()
- virtual void **UpdateVoid** ()
- virtual void **UpdateBool** ()
- virtual void **UpdateInt** ()
- virtual void **UpdateFloat** ()
- virtual void **UpdateString** ()
- virtual void **UpdateString** (string s)
- virtual void **UpdateGameObject** ()
- virtual void **UpdateRigidbody** ()
- virtual void **UpdateStringBool** (string s)
- virtual void **UpdateStringFloat** (string s)
- virtual void **UpdateStringInt** (string s)
- bool **CheckCondition** (bool active, bool changed, bool valueChanged)

Protected Attributes

- Vector3 **_vectorValue**
- bool **vectorChanged**
- bool **initialized**
- bool **_boolValue**
- bool **boolChanged** = true
- int **_intValue**
- bool **intChanged**
- float **_floatValue**
- bool **floatChanged**

Properties

- virtual Vector3 **value** `[getset]`
- virtual bool **boolValue** `[getset]`
- bool isDead `[get]`
  True when the eventHandler is dead and can be removed

Member Enumeration Documentation

enum Passer.EventHandler.Type `[inherited]`

The different types of events when the function is called

Enumerator:

Never	The function is never called.
OnStart	The function is called when the event starts.
OnEnd	The function is called when the event ends.
WhileActive	The function is called every frame while the event is active.
WhileInactive	The function is called every frame while the event is not active.
OnChange	The function is called every time the event changes.
Continuous	The functions is called every frame.

enum Passer.EventHandler.OverrideMode `[inherited]`

Enumerator:

Prepend	Prepend this handler before existing handlers.
Append	Append this handler after existing handlers.

Replace	Replace the topmost handler with this handler.

Member Function Documentation

override void Passer.Vector3Event.UpdateVector3 () `[protected], [virtual]`

Reimplemented from Passer.EventHandler.

Property Documentation

bool Passer.EventHandler.isDead `[get], [inherited]`

True when the eventHandler is dead and can be removed

A function is dead when it does nothing. This is when the functionCall is not defined or when the target of the functionCall is empty

The documentation for this class was generated from the following file:

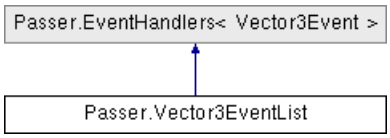
- Assets/Passer/HumanoidControl/Runtime/Tools/Events/Vector3Event.cs

Passer.Vector3EventList Class Reference

Description

A list of event handlers with a Vector3 parameter

Inheritance diagram for Passer.Vector3EventList:



Public Attributes

- int **id**
  The id of the event handler

- string **label**
  The label or name of the Event Handlers

- string **tooltip**
  The tooltip text for the Event Handlers

- string[] **eventTypeLabels**
  The labels for the EventHandler.Type to use in the GUI

- string **fromEventLabel**
  For future use...

- List< T > **events**
  The EventHandlers

Properties

- Vector3 **value** `[getset]`

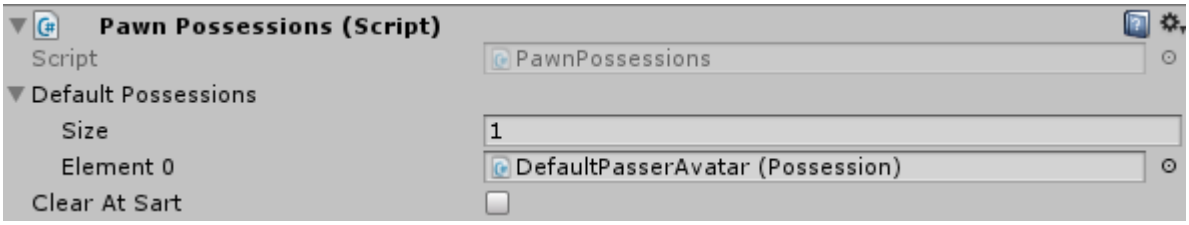
The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControl/Runtime/Tools/Events/Vector3Event.cs

Passer.VisitorPossessions Class Reference

Description

The Possession of a Humanoid Visitor



- Default possessions, see PawnPossessions::defaultPossessions
- Clear at start, see PawnPossessions::clearAtStart
- **Version**

  4.0 and higher

Inherits MonoBehaviour.

Public Member Functions

- void TryAddGameObject (GameObject gameObject)
  Try to add the gameObject to this humanoidpossessions. This only succeeds whent the hameObject has a Possessable component.

- void **AddPossessions** (Possessable[] scenePossessions, bool removable=true)
- Possession Add (Possessable possessable, bool preserved=false)
  Add the possessable object to the humanoidPossessions

- void **DeletePossession** (Possession possession)
- void **UseNextAvatar** ()
- void **UseAvatar** (int avatarIndex)

Static Public Member Functions

- static void **DestroyScenePossessions** ()
- static IEnumerator **RetrievePossessionAsync** (Possession possession, System.Action< GameObject > callback)
- static IEnumerator **RetrievePossessableAsync** (string possessableLocation, string possessablePath, System.Action< GameObject > callback)
- static void **UnloadPossession** ()
- static int **mod** (int k, int n)

Public Attributes

- List< Possession > **possessions**
- Possessable[] **defaultPossessions**
  The possessions which are available from the start

- bool clearOnAwake
  Clear the possessions when the application is started

Protected Member Functions

- virtual void **Awake** ()
- virtual void **OnDestroy** ()

Static Protected Attributes

- static List< CachedPossessionBundle > **bundleCache** = new List<CachedPossessionBundle>()
- static List< CachedPossession > **cache** = new List<CachedPossession>()

Properties

- List< Possession > **myPossessions** `[get]`

Member Function Documentation

void Passer.VisitorPossessions.TryAddGameObject (GameObject *gameObject*)

Try to add the gameObject to this humanoidpossessions. This only succeeds whent the hameObject has a Possessable component.

Parameters

gameObject	The GameObject to add

Possession Passer.VisitorPossessions.Add (Possessable *possessable*, bool *preserved* = `false`)

Add the possessable object to the humanoidPossessions

Parameters

possessable	The possessable object to add

Returns

The persistent possession

Member Data Documentation

bool Passer.VisitorPossessions.clearOnAwake

Clear the possessions when the application is started

Default possessions will not be cleared. This works only on real Humanoids, not on HumanoidInterfaces/Sites

---

The documentation for this class was generated from the following file:

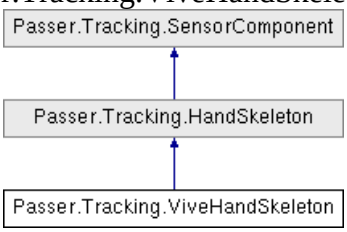
- Assets/Passer/HumanoidControl/Runtime/Visitors/Scripts/VisitorPossessions.cs

Passer.Tracking.ViveHandSkeleton Class Reference

Description

A HTC Vive hand tracking skeleton

Inheritance diagram for Passer.Tracking.ViveHandSkeleton:



Public Types

- enum **BoneId** { **Invalid** = -1, **Hand** = 0, **ThumbProximal**, **ThumbIntermediate**, **ThumbDistal**, **ThumbTip**, **IndexMetaCarpal**, **IndexProximal**, **IndexIntermediate**, **IndexDistal**, **IndexTip**, **MiddleMetaCarpal**, **MiddleProximal**, **MiddleIntermediate**, **MiddleDistal**, **MiddleTip**, **RingMetaCarpal**, **RingProximal**, **RingIntermediate**, **RingDistal**, **RingTip**, **LittleMetacarpal**, **LittleProximal**, **LittleIntermediate**, **LittleDistal**, **LittleTip**, **Forearm**, **Count** }

Public Member Functions

- override void UpdateComponent ()
  Update the component manually

- override int GetBoneId (Finger finger, FingerBone fingerBone)
- virtual GameObject **CreateHandSkeleton** (Transform trackerTransform, bool isLeft, bool showRealObjects)
- virtual Transform **GetForearmBone** ()
- virtual Transform **GetWristBone** ()
- virtual Transform GetBoneTransform (Finger finger, FingerBone fingerBone)
  Gets the transform of the tracked bone

- virtual void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

Static Public Member Functions

- static void **CheckGestureProvider** (TrackerComponent tracker)
- static ViveHandSkeleton **Find** (Transform trackerTransform, bool isLeft)
- static ViveHandSkeleton **Get** (Transform trackerTransform, bool isLeft)
- static HandSkeleton **FindHandSkeleton** (Transform trackerTransform, bool isLeft)

Public Attributes

- bool **isLeft**
  True: this is a left hand. Fale: this is a right hand

- new bool **show** = false
  Determines whether this skeleton should be rendered

- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**
  The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- override void Start ()
  Starts the sensor

- override void InitializeSkeleton ()
  This function is used to initialize the tracked bones

- void **UpdateSkeleton** (GestureResult gestureResult)
- void **UpdateSkeletonRender** ()
  Updates the rendering of the bones

- void **EnableRenderer** ()
- void **DisableRenderer** ()
- virtual void Awake ()
  Initializes the sensor.

Protected Attributes

- List< TrackedBone > **bones**
  The list of tracked bones

- bool **rendered**
- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.ViveHandSkeleton.Start () [protected], [virtual]

Starts the sensor

Does nothing at this moment.

Reimplemented from Passer.Tracking.SensorComponent.

override void Passer.Tracking.ViveHandSkeleton.InitializeSkeleton () [protected], [virtual]

This function is used to initialize the tracked bones

Reimplemented from Passer.Tracking.HandSkeleton.

override void Passer.Tracking.ViveHandSkeleton.UpdateComponent () [virtual]

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

override int Passer.Tracking.ViveHandSkeleton.GetBoneId (Finger *finger*, FingerBone *fingerBone*) [virtual]

Reimplemented from Passer.Tracking.HandSkeleton.

virtual Transform Passer.Tracking.HandSkeleton.GetBoneTransform (Finger *finger*, FingerBone *fingerBone*) [virtual], [inherited]

Gets the transform of the tracked bone

Parameters

finger	The requested finger
fingerBone	The requested bone in the finger

Returns

The tracked bon transform of *null* if it does not exist

Reimplemented in Passer.Tracking.LeapHandSkeleton.

virtual void Passer.Tracking.SensorComponent.Awake () [protected], [virtual], [inherited]

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

virtual void Passer.Tracking.SensorComponent.StartComponent (Transform *trackerTransform*) [virtual], [inherited]

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented in Passer.Tracking.ViveTrackerComponent.

Member Data Documentation

bool Passer.Tracking.SensorComponent.autoUpdate = true `[inherited]`

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/ViveHandTracking/ViveHandSkeleton.cs

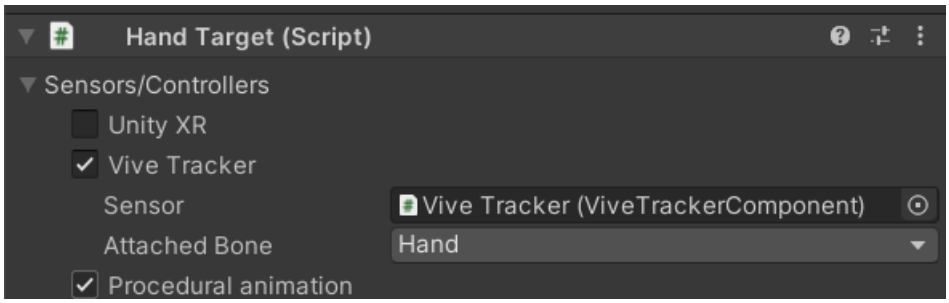
Passer.Humanoid.ViveTrackerArm Class Reference

Description

Vive Tracker used on the leg of a Humanoid

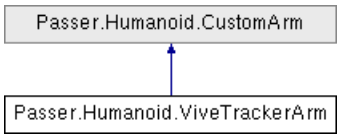
Unity Inspector

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 UnityXR controller and a Vive Tracker on the hand.



- *Show (Button)* : Press to create an Gameobject representing the real world Vive tracker.
- **Sensor** : A reference to the the real world Vive tracker.
- Attached Bone

Inheritance diagram for Passer.Humanoid.ViveTrackerArm:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **CheckSensor** (HandTarget handTarget)
- override void **SetSensor2Target** ()
- override void **UpdateSensorTransformFromTarget** (Transform _)
- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the arm for tracking with the tracked controller and/or skeleton

- override void Update ()
  Updates the arm target based on the status of the tracked controller and/or skeleton

Static Public Member Functions

- static Rotation **CalculateUpperArmOrientation** (Vector upperArmPosition, float upperArmLength, Vector forearmUp, float forearmLength, Vector handPosition, bool isLeft)
- static Rotation **CalculateArmOrientation** (Vector joint1Position, Vector joint1Up, Vector joint2Position, bool isLeft)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- ArmBones **attachedBone** = ArmBones.Hand
  The bone on the arm controlled by the sensor

- bool **isLeft**
- TargetData **shoulder**
- TargetData **upperArm**
- TargetData **forearm**
- TargetData **hand**
- Finger **thumb**
- Finger **indexFinger**
- Finger **middleFinger**
- Finger **ringFinger**
- Finger **littleFinger**
- Finger[] **fingers**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- HumanoidTarget.TargetedBone **GetTargetBone** ()
- override void **CreateSensorTransform** ()
- void **UpdateControllerInput** ()
  Updates the Controller Input for this side

- virtual void UpdateControllerInput (ControllerSide controllerSide)
  Updates one side of the ControllerInput from the values of the tracked ControllerComponent

- override void UpdateHandFromSkeleton ()
  This function uses the tracked HandSkeleton to update the pose of the hand

- virtual void **UpdateHand** ()
- void **UpdateSensor** ()

Protected Attributes

- ControllerComponent **controllerComponent**
  The tracker controller to use for this arm

- Controller **controllerInput**
  The controller input for this humanoid

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- ViveTrackerComponent **viveTracker** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.CustomArm.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) `[inherited]`

Prepares the arm for tracking with the tracked controller and/or skeleton

Parameters

_humanoid	The humanoid for which this arm is tracked
targetTransform	The transform of the hand target

This will find and initialize the controllerInput for the given humanoid. It will initialize the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking for the controller and/or hand skeleton.

override void Passer.Humanoid.CustomArm.Update () `[virtual], [inherited]`

Updates the arm target based on the status of the tracked controller and/or skeleton

Reimplemented from Passer.Humanoid.Tracking.Sensor.

virtual void Passer.Humanoid.CustomArm.UpdateControllerInput (ControllerSide *controllerSide*) `[protected], [virtual], [inherited]`

Updates one side of the ControllerInput from the values of the tracked ControllerComponent

Parameters

controllerSide	The controller side to update

This function does nothing when the controller is not available or not tracking.

override void Passer.Humanoid.CustomArm.UpdateHandFromSkeleton () `[protected], [inherited]`

This function uses the tracked HandSkeleton to update the pose of the hand

This function does nothing when the hand skeleton is not available or not tracking.

The documentation for this class was generated from the following file:

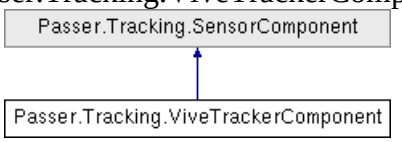
- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/ViveTrackers/ViveTrackerArm.cs

Passer.Tracking.ViveTrackerComponent Class Reference

Description

A representation of a real-world Vive Tracker

Inheritance diagram for Passer.Tracking.ViveTrackerComponent:



Public Member Functions

- override void StartComponent (Transform trackerTransform)
  Start the manual updating of the sensor.

- override void UpdateComponent ()
  Update the component manually

Static Public Member Functions

- static ViveTrackerComponent **Get** (Transform realWorld, Vector3 position, Quaternion rotation)
- static Quaternion **GetRotation** (Matrix4x4 matrix)
- static Vector2 **GetPosition** (Matrix4x4 matrix)

Public Attributes

- int trackerId = -1
  The tracker id of this tracker

- TrackerComponent tracker
  The tracker component for this Vive Tracker

- string **hardwareId**
  The unique hardware identification of this tracker

- bool **useForBodyTracking** = true
  Is this tracker used for body tracking (true) or generic object tracking (false)

- SteamVR_Input_Sources inputSource = SteamVR_Input_Sources.Any
  The Vive Tracker role for this tracker

- Tracker.Status **status**
  The tracking status of the sensor

- float **rotationConfidence**
  The confidence (0..1) of the tracked rotation

- float **positionConfidence**

The confidence (0..1) of the tracked position

- bool autoUpdate = true
  Is used to set whether the sensor updates itself

Protected Member Functions

- override void Start ()
  Starts the sensor

- virtual void **OnEnable** ()
- virtual void **OnDisable** ()
- virtual void **OnDeviceConnectedChanged** (SteamVR_Action_Pose changedAction, SteamVR_Input_Sources changedSource, bool connected)
- virtual void **OnUpdate** (SteamVR_Action_Pose fromAction, SteamVR_Input_Sources fromSource)
- virtual void Awake ()
  Initializes the sensor.

Protected Attributes

- SteamVR_Action_Pose **poseAction** = SteamVR_Input.GetAction<SteamVR_Action_Pose>("Pose")
- Transform **trackerTransform**
  The transform which is used as the root of the tracking space

- bool **_show**

Properties

- virtual bool show `[getset]`
  The render status of the sensor

- bool **renderController** `[set]`
  Enable or disable the renderers for this sensor.

Member Function Documentation

override void Passer.Tracking.ViveTrackerComponent.StartComponent (Transform *trackerTransform*) `[virtual]`

Start the manual updating of the sensor.

Parameters

trackerTransform	

When this function has been called, autoUpdate will be disabled and the sensor will no longer update from Unity Updates. Instead, UpdateComponent needs to be called to update the sensor data

Reimplemented from Passer.Tracking.SensorComponent.

override void Passer.Tracking.ViveTrackerComponent.Start () `[protected], [virtual]`

Starts the sensor

Does nothing at this moment.

Reimplemented from Passer.Tracking.SensorComponent.

override void Passer.Tracking.ViveTrackerComponent.UpdateComponent () `[virtual]`

Update the component manually

This function is meant to be overridden

Reimplemented from Passer.Tracking.SensorComponent.

virtual void Passer.Tracking.SensorComponent.Awake () `[protected], [virtual], [inherited]`

Initializes the sensor.

When trackerTransform is null, it will be set automatically to the parent of this transform.

Member Data Documentation

int Passer.Tracking.ViveTrackerComponent.trackerId = -1

The tracker id of this tracker

This value can change between runs

TrackerComponent Passer.Tracking.ViveTrackerComponent.tracker

The tracker component for this Vive Tracker

This usually is an UnityXR object

SteamVR_Input_Sources Passer.Tracking.ViveTrackerComponent.inputSource = SteamVR_Input_Sources.Any

The Vive Tracker role for this tracker

/sa ViveTacker

bool Passer.Tracking.SensorComponent.autoUpdate = true `[inherited]`

Is used to set whether the sensor updates itself

When enabled, the sensor will update itself. When disabled, StartComponent and UpdateComponent need to be called to update the tracking status.

Property Documentation

virtual bool Passer.Tracking.SensorComponent.show `[get]`, `[set]`, `[inherited]`

The render status of the sensor

When enabled, sensors with renderers attached will be rendered. When disabled, sensors will not be rendered.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/ViveTrackers/ViveTrackerComponent.cs

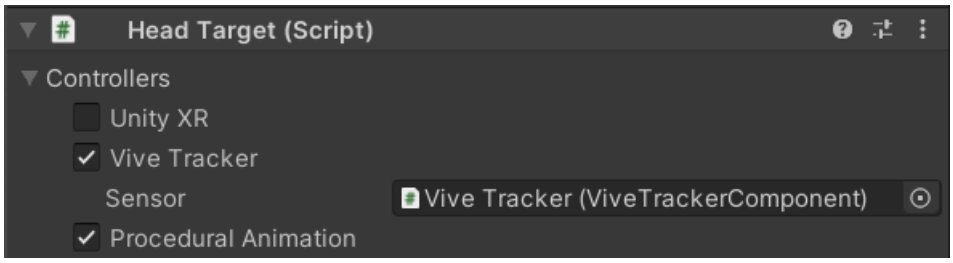
Passer.Humanoid.ViveTrackerHead Class Reference

Description

Vive Tracker used on the head of a Humanoid

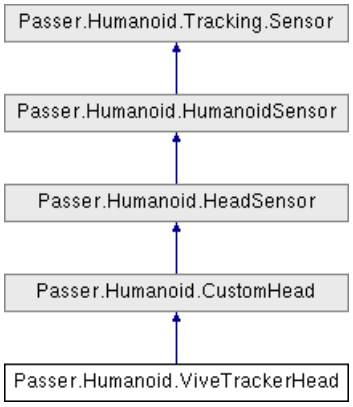
Unity Inspector

A Vive Tracker can be used on the head instead of an HMD. There is no need to have a UnityXR controller or any camera at all in the scene. It is not possible to have both a Unity XR and Vive Tracker enabled on the head.



- **Show (Button)** : Press to create an Gameobject representing the real world Vive tracker.
- **Sensor** : A reference to the the real world Vive tracker.

Inheritance diagram for Passer.Humanoid.ViveTrackerHead:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void CheckSensor (HeadTarget headTarget)
- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the head for tracking with the tracked sensor

- override void Update ()
  Updates the head target based on the status of the tracke sensor

- virtual void **Init** (HeadTarget headTarget)

- void **InitController** (SerializedProperty sensorProp, HeadTarget target)
- void **RemoveController** (SerializedProperty sensorProp)
- virtual void **CheckSensorTransform** ()
- virtual void **SetSensor2Target** ()
- virtual void **UpdateSensorTransformFromTarget** (Transform targetTransform)
- virtual void **Stop** ()
- virtual void **RefreshSensor** ()
- virtual void **ShowSensor** (HumanoidTarget target, bool shown)

Static Public Member Functions

- static Vector3 **InverseTransformPointUnscaled** (Transform transform, Vector3 position)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- Vector3 **sensor2TargetPosition**
- Quaternion **sensor2TargetRotation**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Static Public Attributes

- const string **_name** = ""

Protected Member Functions

- virtual void **CreateSensorTransform** (string resourceName, Vector3 sensor2TargetPosition, Quaternion sensor2TargetRotation)
- virtual void **CreateSensorTransform** ()
- void **CreateSensorTransform** (Transform targetTransform, string resourceName, Vector3 _sensor2TargetPosition, Quaternion _sensor2TargetRotation)
- virtual void **UpdateNeckTargetFromHead** ()
- void **RemoveSensorTransform** ()
- void **UpdateSensorTransform** (Tracking.Sensor sensor)
- virtual void **UpdateTargetTransform** ()
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, Transform sensorTransform)
- virtual void **UpdateTarget** (HumanoidTarget.TargetTransform target, SensorComponent sensorComponent)
- Vector3 **GetTargetPosition** (Transform sensorTransform)
- Quaternion **GetTargetRotation** (Transform sensorTransform)
- void **UpdateSensor** ()

Static Protected Member Functions

- static Vector3 **TransformPointUnscaled** (Transform transform, Vector3 position)

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**

Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- ViveTrackerComponent **viveTracker** `[get]`
- HeadTarget **headTarget** `[get]`
- HumanoidControl **humanoid** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`

Member Function Documentation

override void Passer.Humanoid.ViveTrackerHead.CheckSensor (HeadTarget *headTarget*) `[virtual]`

Reimplemented from Passer.Humanoid.HeadSensor.

override void Passer.Humanoid.CustomHead.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) `[virtual]`, `[inherited]`

Prepares the head for tracking with the tracked sensor

Parameters

_humanoid	The humanoid for which this head is tracked
targetTransform	The transform of the head target

It will initialize the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking of the sensor.

Reimplemented from Passer.Humanoid.HeadSensor.

override void Passer.Humanoid.CustomHead.Update () `[virtual]`, `[inherited]`

Updates the head target based on the status of the tracke sensor

Reimplemented from Passer.Humanoid.HeadSensor.

The documentation for this class was generated from the following file:

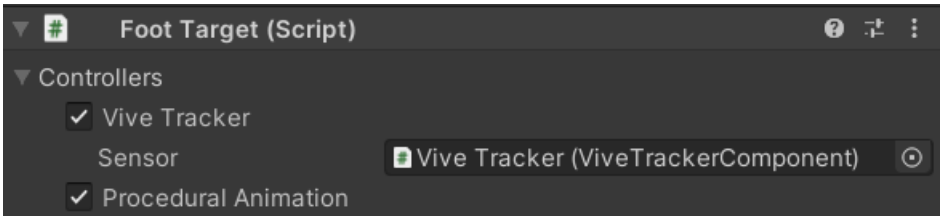
- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/ViveTrackers/ViveTrackerHead.cs

Passer.Humanoid.ViveTrackerLeg Class Reference

Description

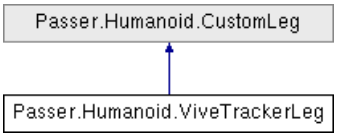
Vive Tracker used on the leg of a Humanoid

Unity Inspector



- **Show (Button)** : Press to create an Gameobject representing the real world Vive tracker.
- **Sensor** : A reference to the the real world Vive tracker.

Inheritance diagram for Passer.Humanoid.ViveTrackerLeg:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **CheckSensor** (FootTarget footTarget)
- override void **SetSensor2Target** ()
  Updates the leg targets based on the current sensor position and rotation

- override void UpdateSensorTransformFromTarget (Transform _)
  Updates the sensor position and rotation based on the current position of the leg targets

- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the leg for tracking with the sensor

- override void Update ()
  Updates the leg target based on the status of the tracked sensor

Static Public Member Functions

- static Rotation **CalculateLegOrientation** (Vector joint1Position, Vector joint2Position, Rotation hipsRotation)
- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- LegBones **attachedBone** = LegBones.Foot
  The bone on the leg controlled by the sensor

- bool **isLeft**
- TargetData **upperLeg**
- TargetData **lowerLeg**
- TargetData **foot**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- override void **CreateSensorTransform** ()
- void **UpdateSensor** ()

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- ViveTrackerComponent **viveTracker** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.CustomLeg.UpdateSensorTransformFromTarget (Transform _) [inherited]

Updates the sensor position and rotation based on the current position of the leg targets

Parameters

_	Not used

override void Passer.Humanoid.CustomLeg.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) [inherited]

Prepares the leg for tracking with the sensor

Parameters

_humanoid	The humanoid for which this leg is tracked
targetTransform	The transform of the foot target

It will initialze the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking of the sensor.

override void Passer.Humanoid.CustomLeg.Update () [virtual], [inherited]

Updates the leg target based on the status of the tracked sensor

Reimplemented from Passer.Humanoid.Tracking.Sensor.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/ViveTrackers/ViveTrackerLeg.cs

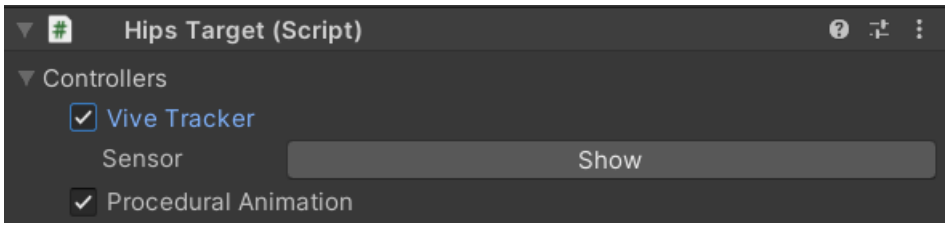
Passer.Humanoid.ViveTrackerTorso Class Reference

Description

Vive Tracker used on the torso of a Humanoid

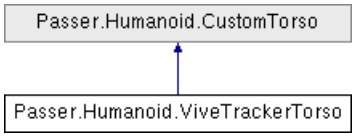
Unity Inspector

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).



- **Show (Button)** : Press to create an Gameobject representing the real world Vive tracker.
- **Sensor** : A reference to the the real world Vive tracker.

Inheritance diagram for Passer.Humanoid.ViveTrackerTorso:



Public Types

- enum **ID** { **Head**, **LeftHand**, **RightHand**, **Hips**, **LeftFoot**, **RightFoot**, **Tracker1**, **Tracker2**, **Tracker3**, **Tracker4**, **Count** }

Public Member Functions

- override void **CheckSensor** (HipsTarget hipsTarget)
- override void **SetSensor2Target** ()
  Updates the torso targets based on the current sensor position and rotation

- override void UpdateSensorTransformFromTarget (Transform _)
  Updates the sensor position and rotation based on the current position of the torso targets

- override void Start (HumanoidControl _humanoid, Transform targetTransform)
  Prepares the torso for tracking with the sensor

- override void Update ()
  Updates the torso targets based on the status of the tracked sensor

Static Public Member Functions

- static Rotation **CalculateBoneRotation** (Vector bonePosition, Vector parentBonePosition, Vector upDirection)

Public Attributes

- TorsoBones **attachedBone** = TorsoBones.Hips
  The bone on the torso controlled by the sensor

- TargetData **chest**
- TargetData **spine**
- TargetData **hips**
- DeviceView **device**
  The device to which the sensor belongs

- Tracker.Status **status** = Tracker.Status.Unavailable
  Status of the sensor

Protected Member Functions

- override void **CreateSensorTransform** ()
- override void **UpdateTarget** (HumanoidTarget.TargetTransform target, SensorComponent sensorComponent)
- void **UpdateSensor** ()

Protected Attributes

- Vector **_localSensorPosition**
- Rotation **_localSensorRotation**
- Vector **_sensorPosition**
- Rotation **_sensorRotation**
- float **_positionConfidence**
  Tracking confidence

- float **_rotationConfidence**
- Vector **_sensor2TargetPosition** = Vector.zero
  The position of the tracker relative to the origin of the object it is tracking

- Rotation **_sensor2TargetRotation** = Rotation.identity

Properties

- override string **name** `[get]`
- override HumanoidTracker **tracker** `[get]`
- Vector **localSensorPosition** `[get]`
- Rotation **localSensorRotation** `[get]`
- Vector **sensorPosition** `[get]`
- Rotation **sensorRotation** `[get]`
- float **positionConfidence** `[get]`
- float **rotationConfidence** `[get]`
- Vector **sensor2TargetPosition** `[getset]`
- Rotation **sensor2TargetRotation** `[getset]`

Member Function Documentation

override void Passer.Humanoid.CustomTorso.UpdateSensorTransformFromTarget (Transform _) [inherited]

Updates the sensor position and rotation based on the current position of the torso targets

Parameters

_	Not used

override void Passer.Humanoid.CustomTorso.Start (HumanoidControl *_humanoid*, Transform *targetTransform*) [inherited]

Prepares the torso for tracking with the sensor

Parameters

_humanoid	The humanoid for which this torso is tracked
targetTransform	The transform of the hips target

It will initialze the sensor2TargetPosition and sensor2TargetRotation values. It will determine whether the sensor should be shown and rendered. It will start the tracking of the sensor.

override void Passer.Humanoid.CustomTorso.Update () [virtual], [inherited]

Updates the torso targets based on the status of the tracked sensor

Reimplemented from Passer.Humanoid.Tracking.Sensor.

The documentation for this class was generated from the following file:

- Assets/Passer/HumanoidControlPlus/Runtime/Scripts/Extensions/ViveTrackers/ViveTrackerTorso.cs

Index

INDEX