onAirVR

onAirVR Server for Unity User Manual

Version 1.4.0

TABLE OF CONTENTS

INTRODUCTION

onAirVR makes a mobile VR device act as a wireless VR HMD for a desktop by streaming video/audio which are rendered in realtime on the desktop to onAirVR mobile apps. This document describes how to implement onAirVR desktop contents on Unity game engine using onAirVR Server for Unity.

SYSTEM REQUIREMENTS

Hardware

- Desktop
    - NVIDIA graphics card **(Kepler architecture or later)**
- Mobile
    - Samsung GearVR

Software

- Desktop
    - Windows 7 or later
    - **Unity 5.6.x or higher**
    - NVIDIA CUDA Toolkit 8.0
      https://developer.nvidia.com/cuda-80-ga2-download-archive
    - The latest NVIDIA Graphics Driver
      You SHOULD update the graphics driver after installing CUDA toolkit because CUDA toolkit installation may include an older version of the graphics driver.
- Mobile
    - Android 5.0 (Lollipop) or higher

QUICK START

1. Put an **AirVRCameraRig** prefab onto a suitable place (e.g. the head position of a player character).

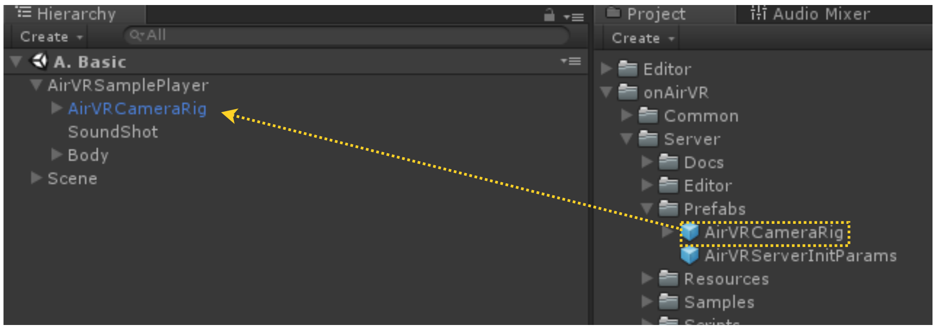


Figure 1. Put an AirVRCameraRig prefab under the player object.

2. Play your project in the editor.
3. Launch onAirVR App on your phone then combine the phone with your GearVR.
4. Enter the IP address of your desktop and click Play button.

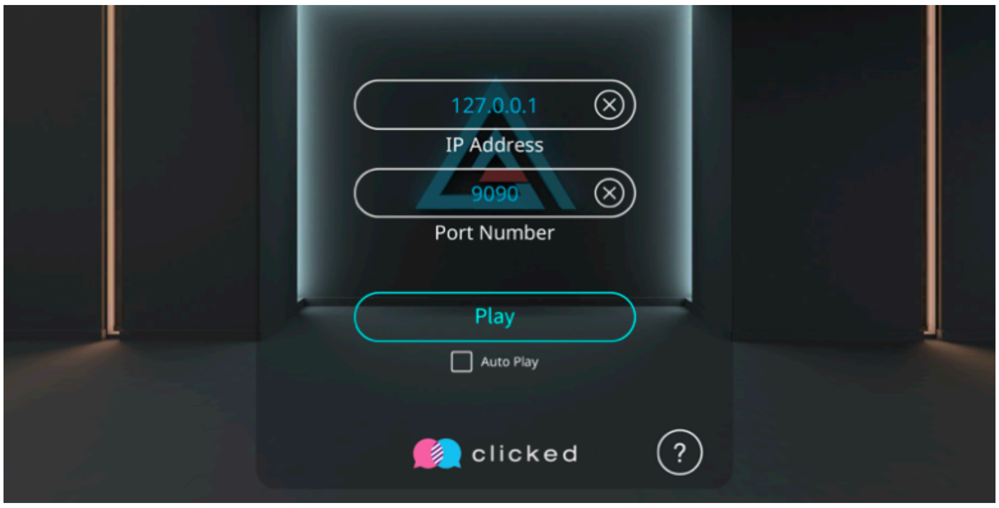


Figure 2. The scene of onAirVR App

5. Enjoy your scene!

PROGRAMMING GUIDE

How onAirVR Server Works

When an onAirVR server application is launched and the first scene is loaded, the first awaken AirVRCameraRig instantiates an AirVRServer instance and starts it up. And each AirVRCameraRig registers itself to AirVRCameraRigManager in the scene (If no AirVRCameraRigManager exists in the scene, one instance of it is automatically instantiated). Then,

1. When the onAirVR App on a mobile VR device connects to the onAirVR server application,
2. AirVRServer establishes a session and informs AirVRCameraRigManager.
3. AirVRCameraRigManager then finds an available AirVRCameraRig, and
4. Binds the AirVRCameraRig to the session.
5. Data from the client - such as the HMD orientation, input device values, etc. - are applied to the AirVRCameraRig through the session.
6. Meanwhile AirVRCameraRig renders video frames using child Unity cameras then encodes and sends the video frames back to the client.

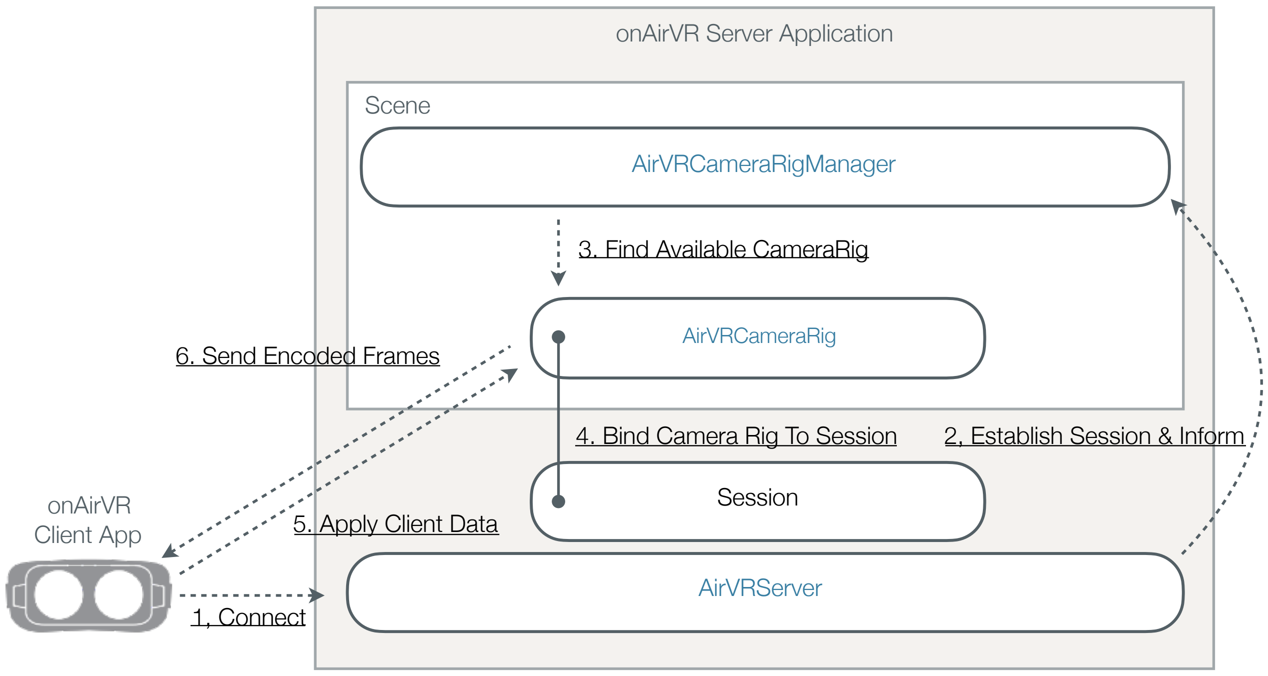


Figure 3. This diagram describes how onAirVR components interact each other.

If you load a new scene containing AirVRCameraRigs,

1. AirVRCameraRigs in the old scene are unbound from the current sessions, then
2. AirVRCameraRigManager tries to bind AirVRCameraRigs in the new scene to the sessions.

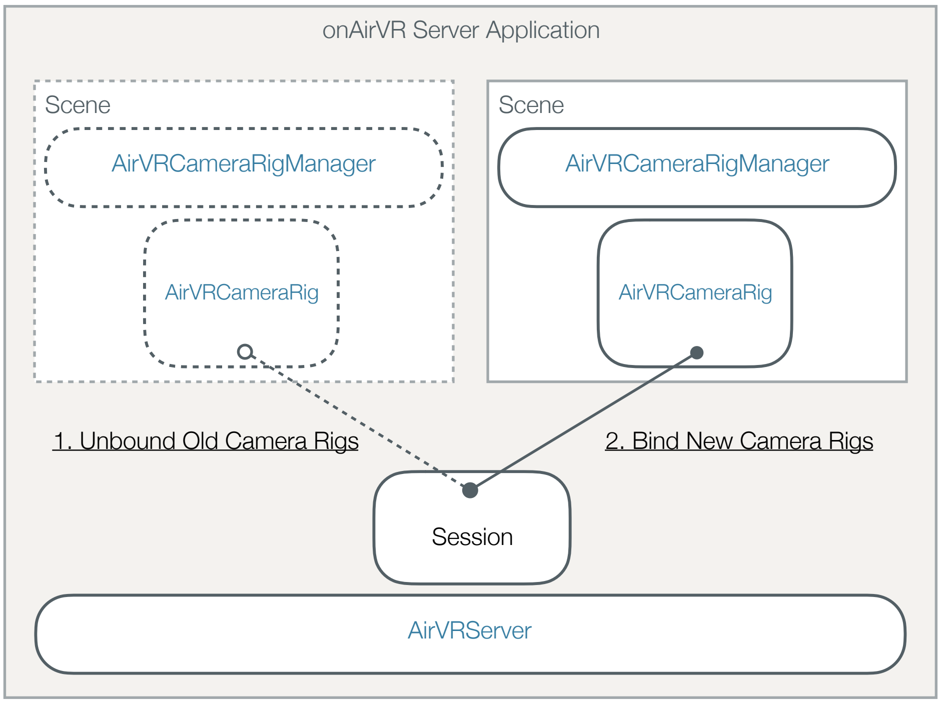
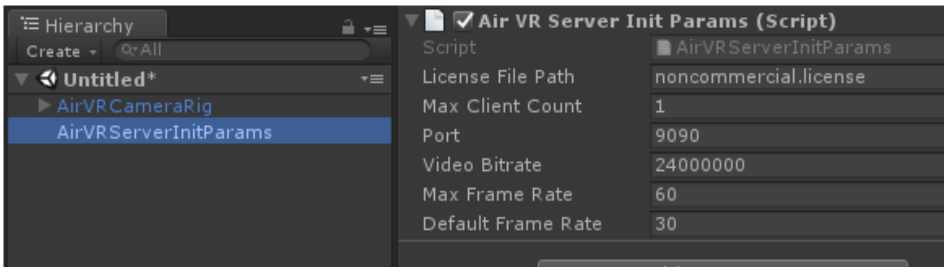


Figure 4. When loading a new scene, AirVRCameraRig bindings are transferred.

Server Configuration

You can override network configuration and video encoding parameters by putting an AirVRServerInitParams on the scene where AirVRServer will be started up. Then the AirVRServer replace the default configuration to one of the AirVRServerInitParams. Please see “References” section for each field for detail.



Script	AirVRServerInitParams
License File Path	noncommercial.license
Max Client Count	1
Port	9090
Video Bitrate	24000000
Max Frame Rate	60
Default Frame Rate	30

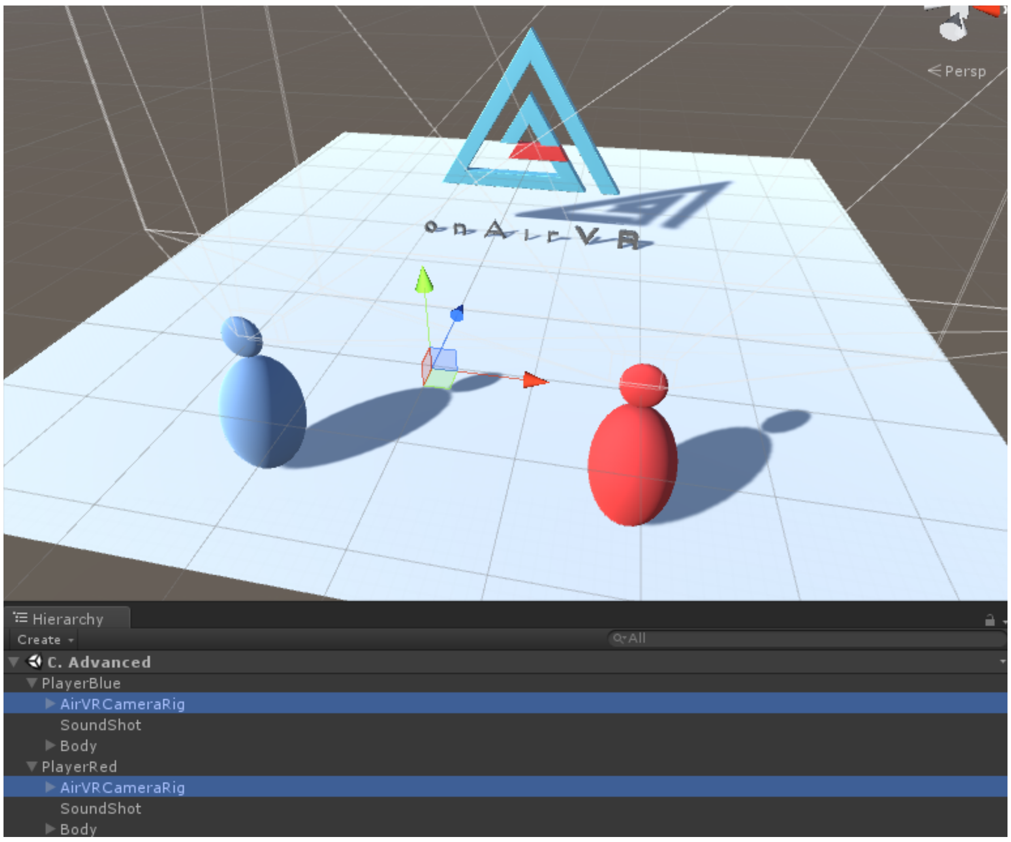
Figure 5. AirVRServerInitParams is put on a scene.

Multiple Players

onAirVR server acts like a video streaming server which streams realtime-rendered video frames to clients. So it's possible that two or more clients are connecting and playing simultaneously. To make a scene with multiple players, you just need to :

1. Put two or more AirVRCameraRig instances in the scene, and
2. Override the maximum client count to two or more using AirVRServerInitParams.

Then when a session is established for a client, AirVRCameraRigManager finds one of available AirVRCameraRigs in the scene randomly then binds it to the session.
Or if you implement AirVRCameraRigManager.EventHandler, AirVRCameraRigManager requests you to select one of AirVRCameraRigs through AirVRCameraRigManager.EventHandler.AirVRCameraRigWillBeBound().



```
C. Advanced
  PlayerBlue
    AirVRCameraRig
    SoundShot
    Body
  PlayerRed
    AirVRCameraRig
    SoundShot
    Body
```

Figure 6. Two AirVRCameraRigs are instantiated in a scene to allow two players.

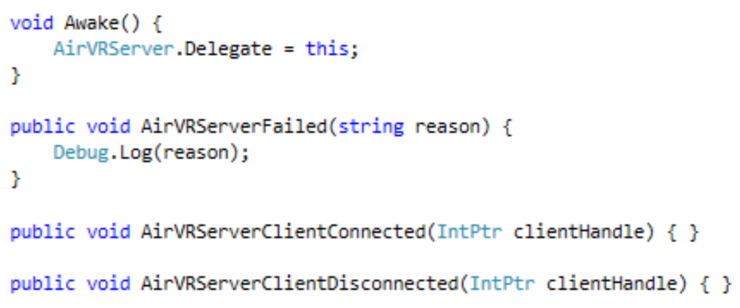
Note
There is a limitation on the number of encoding sessions depending on your graphics card.
For example, NVIDIA Geforce graphics cards allow up to two encoding sessions due to licensing restrictions. In this case you must not make your content available for more than two clients simultaneously.

Event Handling

There are two main components where events you might be interested in are occurred - AirVRServer and AirVRCameraRigManager. If you would like to do something for the events you need to

1. Implement AirVRServer.EventHandler interface and set to AirVRServer.Delegate, and/or
2. Implement AirVRCameraRigManager.EventHandler interface and set to AirVRCameraRigManager.managerOnCurrentScene.Delegate.

Please see “References” section for detail.



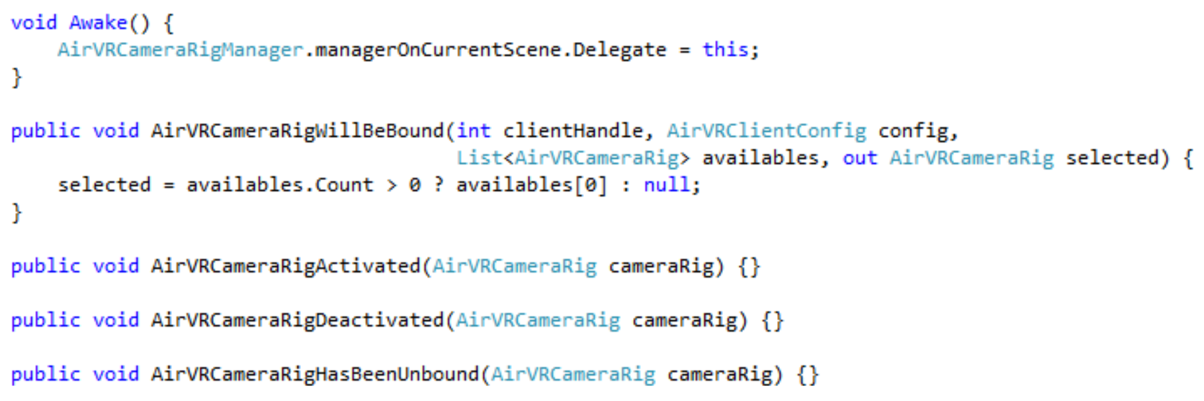
```
void Awake() {
    AirVRServer.Delegate = this;
}

public void AirVRServerFailed(string reason) {
    Debug.Log(reason);
}

public void AirVRServerClientConnected(IntPtr clientHandle) { }

public void AirVRServerClientDisconnected(IntPtr clientHandle) { }
```

Figure 7. An example of AirVRServer.EventHandler implementation



```
void Awake() {
    AirVRCameraRigManager.managerOnCurrentScene.Delegate = this;
}

public void AirVRCameraRigWillBeBound(int clientHandle, AirVRClientConfig config,
                                      List<AirVRCameraRig> availables, out AirVRCameraRig selected) {
    selected = availables.Count > 0 ? availables[0] : null;
}

public void AirVRCameraRigActivated(AirVRCameraRig cameraRig) {}

public void AirVRCameraRigDeactivated(AirVRCameraRig cameraRig) {}

public void AirVRCameraRigHasBeenUnbound(AirVRCameraRig cameraRig) {}
```

Figure 8. An example of AirVRCameraRigManager.EventHandler implementation

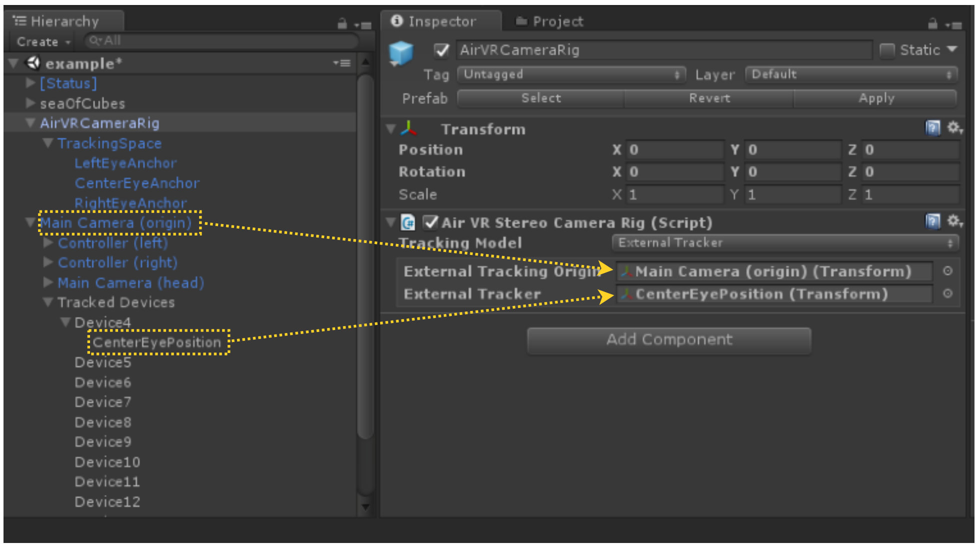
Camera Rig Tracking Models

AirVRStereoCameraRig.TrackingModel defines how the eye cameras of a camera rig move along with the pose of HMD. You can choose one of tracking models for each AirVRStereoCameraRig :

1. Head
   - This tracking model follows the head model of mobile VR of your onAirVR client HMD, which commonly includes eye height (from neck), eye depth and interpupillary distance. You might want to read Oculus developer documentation for the head model of onAirVR App for Oculus Mobile.
2. InterpupillaryDistanceOnly
   - Two eye cameras rotate along with the pose of onAirVR client HMD, while keeping the interpupillary distance. You may consider this as Head model without eye height and eye depth.
3. NoPositionTracking
   - Eye cameras only rotate along with the pose onAirVR client HMD. You can adjust the position of each eye camera manually.

And there is one more model which is specifically designed for external motion tracking system integration like Valve's Lighthouse, OptiTrack, etc. By attaching a tracker device onto your onAirVR client HMD you can achieve 6DOF mobile VR experience.

4. ExternalTracker
   - Eye cameras follow a tracker transform which is tracked by external motion tracking system.
   - Set the tracker transform which tracks the center position of eyes to AirVRStereoCameraRig.externalTracker and the tracking origin transform to AirVRStereoCameraRig.externalTrackingOrigin. The below figure shows an example of how to integrate a Vive tracker with Valve's Lighthouse using SteamVR plugin in Unity.



Hierarchy
Create
example*
[Status]
seaOfCubes
AirVRCameraRig
TrackingSpace
LeftEyeAnchor
CenterEyeAnchor
RightEyeAnchor
Main Camera (origin)
Controller (left)
Controller (right)
Main Camera (head)
Tracked Devices
Device4
CenterEyePosition
Device5
Device6
Device7
Device8
Device9
Device10
Device11
Device12

Inspector | Project
AirVRCameraRig | Static
Tag: Untagged | Layer: Default
Prefab: Select | Revert | Apply
Transform
Position: X 0 Y 0 Z 0
Rotation: X 0 Y 0 Z 0
Scale: X 1 Y 1 Z 1
Air VR Stereo Camera Rig (Script)
Tracking Model: External Tracker
External Tracking Origin: Main Camera (origin) (Transform)
External Tracker: CenterEyePosition (Transform)
Add Component

Figure 9. Setting up ExternalTracker tracking model with a Vive tracker in Unity

Input

Using AirVRInput class, you can get the values of input devices of a connected mobile HMD bound to an AirVRCameraRig. As you can see in "References" section, you can use AirVRInput class in the same manner with UnityEngine.Input except that AirVRInput methods require an AirVRCameraRig as an argument. Currently onAirVR mobile app supports GearVR Touchpad, Xbox Controller and GearVR Controller and the below figures describe input bindings of each input device. Also note that you can get the pose of a device tracked by sensor - HMD and GearVR Controller - using AirVRInput.GetTrackedDevicePositionAndOrientation().

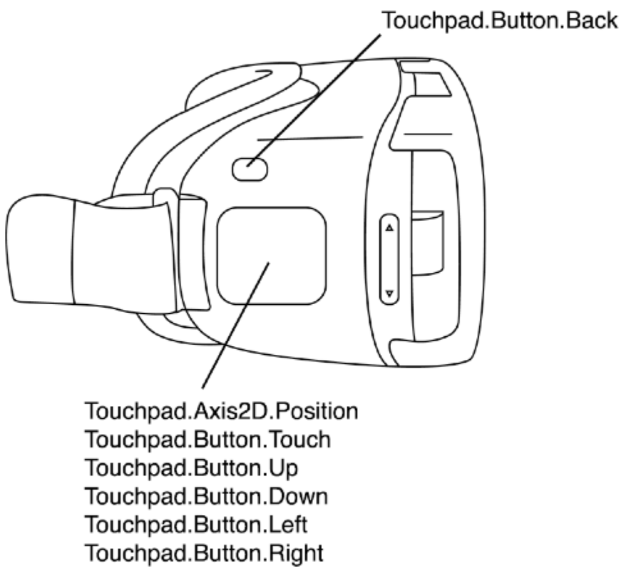


Figure 10. Axes and buttons of GearVR Touchpad

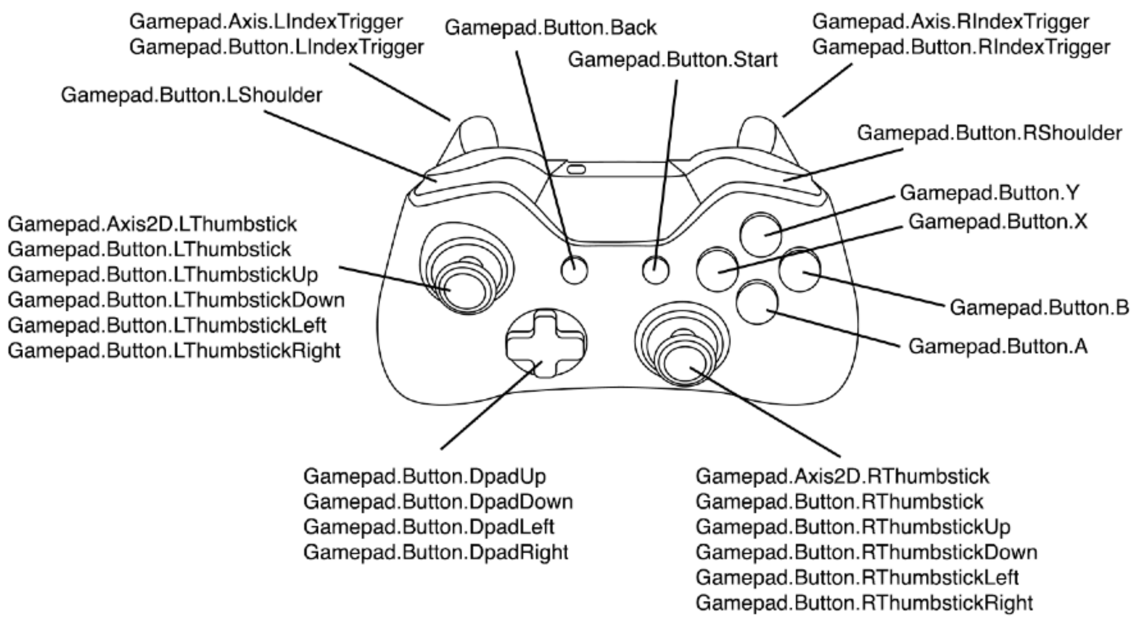

Gamepad.Axis.LIndexTrigger
Gamepad.Button.LIndexTrigger
Gamepad.Button.Back
Gamepad.Button.Start
Gamepad.Axis.RIndexTrigger
Gamepad.Button.RIndexTrigger
Gamepad.Button.LShoulder
Gamepad.Button.RShoulder
Gamepad.Button.Y
Gamepad.Button.X
Gamepad.Axis2D.LThumbstick
Gamepad.Button.LThumbstick
Gamepad.Button.LThumbstickUp
Gamepad.Button.LThumbstickDown
Gamepad.Button.LThumbstickLeft
Gamepad.Button.LThumbstickRight
Gamepad.Button.B
Gamepad.Button.A
Gamepad.Button.DpadUp
Gamepad.Button.DpadDown
Gamepad.Button.DpadLeft
Gamepad.Button.DpadRight
Gamepad.Axis2D.RThumbstick
Gamepad.Button.RThumbstick
Gamepad.Button.RThumbstickUp
Gamepad.Button.RThumbstickDown
Gamepad.Button.RThumbstickLeft
Gamepad.Button.RThumbstickRight

Figure 11. Axes and buttons of Xbox Controller

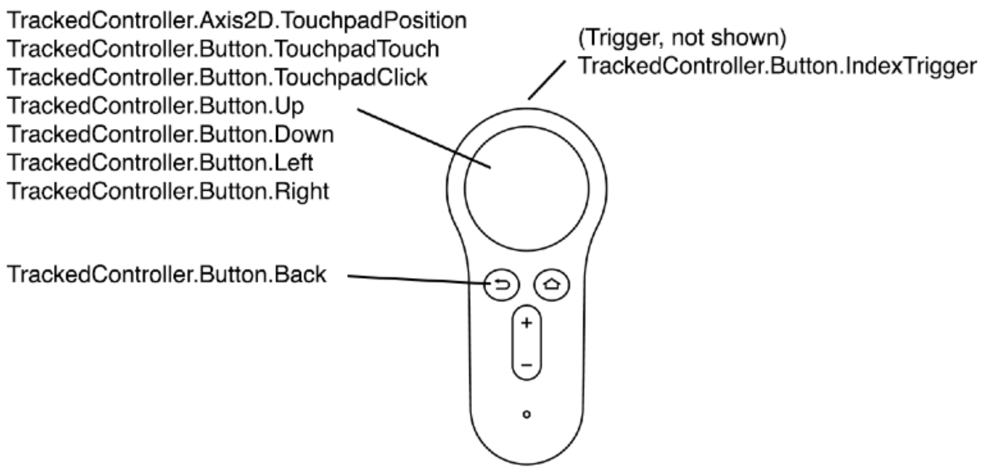

TrackedController.Axis2D.TouchpadPosition
TrackedController.Button.TouchpadTouch
TrackedController.Button.TouchpadClick
TrackedController.Button.Up
TrackedController.Button.Down
TrackedController.Button.Left
TrackedController.Button.Right
(Trigger, not shown)
TrackedController.Button.IndexTrigger
TrackedController.Button.Back

Figure 12. Axes and buttons of GearVR controller

Unity Event System Integration For Tracked Devices (experimental)

Like camera view point or mouse pointer in desktop UI environment, tracked devices - HMD or tracked controller - are usually used as pointing devices in mobile VR UI. onAirVR Event System integrates such tracked devices into Unity Event System and you can use it in the same manner with Unity Event System. The below table describes which onAirVR Event System component corresponds to which Unity Event System component. Please see “B. Event System (experimental)” sample scene for an example of how to use onAirVR Event System in detail.

Component	UnityEngine.EventSystems	onAirVR
event system	EventSystem	AirVREventSystem
input module	StandaloneInputModule, TouchInputModule, etc.	AirVRInputModule
raycaster on Canvas	GraphicRaycaster	AirVRGraphicRaycaster (working on Canvas in world space only)
raycaster in physics	PhysicsRaycaster	AirVRPhysicsRaycaster
pointer	UnityEngine.Camera, mouse pointer, etc.	AirVRGazePointer (HMD) AirVRTrackedControllerPointer (tracked controller) AirVRAutoSelectPointer (tracks tracked controller if available; else does HMD)

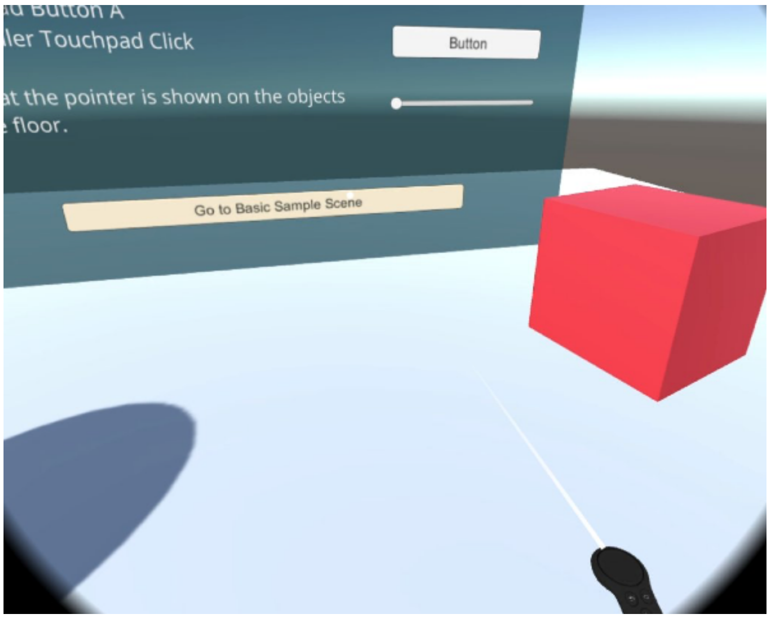


Figure 13. Tracked controller working with onAirVR Event System in onAirVR App

Audio

AirVRServerAudioOutputRouter routes "stereo" audio rendered by Unity audio engine to connected clients. It must be attached to an GameObject on which a Unity's AudioListener is attached.
(Please see AirVRServerAudioOutputRouter component in **CenterEyeAnchor** of **AirVRCameraRig** prefab.)

There are several options for input and output.

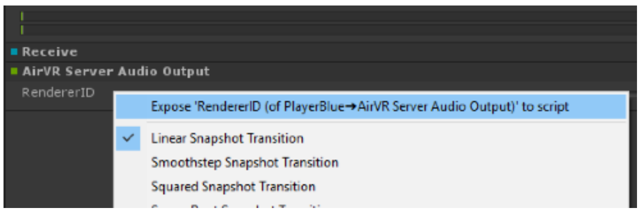
- Input
  - AudioListener : routes stereo audio heard by Unity's AudioListener to clients
  - AudioPlugin : routes stereo audio passing through "AirVR Server Audio Output" audio plugin in a Unity's AudioMixer to clients
- Output
  - All : broadcast audio to all of connected clients
  - One : routes audio to a specific client

In **AirVRCameraRig** prefab, AirVRServerAudioOutputRouter routes AudioListener input to All output by default. Please see "C. Advanced" sample scene for an example which uses AudioPlugin as input.

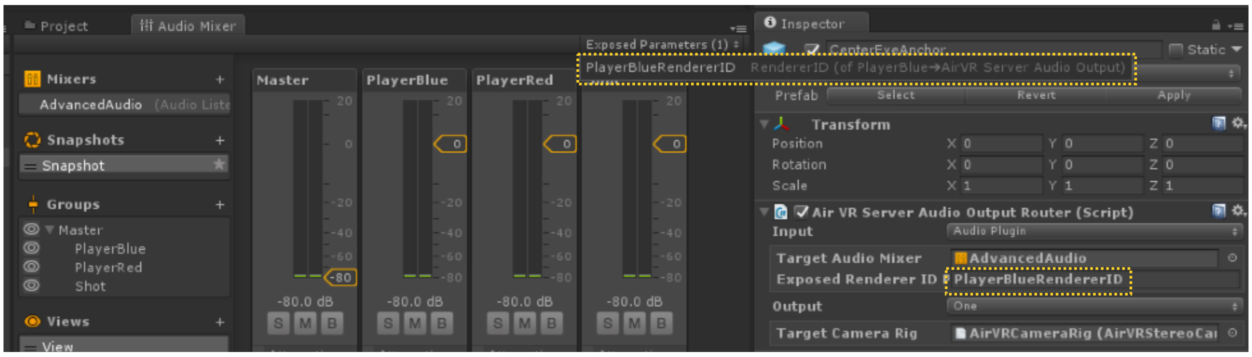
For advanced audio application, you might want to see the code of AirVRServerAudioOutputRouter, in which it takes raw audio data from Unity audio engine then send them to clients using AirVRServer.SendAudioFrame(). You can use this method if you want to send your own raw audio directly, for example in the case you are using a 3rd-party or your own audio engine.

Note
"AirVR Server Audio Output" audio plugin uses "audio renderer ID" to send audio data to AirVRCameraRig. When you set AudioPlugin as input, you must (1) expose RendererID parameter of the plugin then (2) set it to exposedRendererIDParameterName field of AirVRServerAudioOutputRouter.

(1)


Receive
AirVR Server Audio Output
RendererID
Expose 'RendererID (of PlayerBlue→AirVR Server Audio Output)' to script
Linear Snapshot Transition
Smoothstep Snapshot Transition
Squared Snapshot Transition

(2)



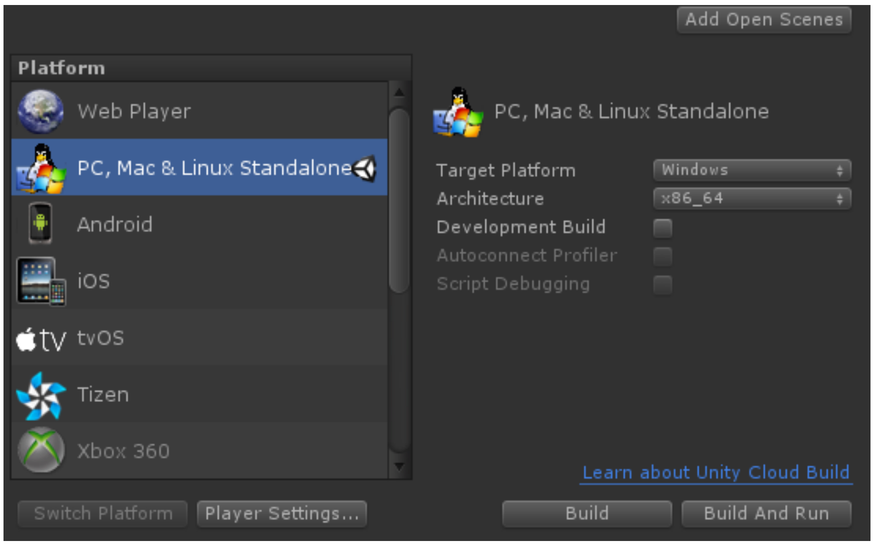
PlayerBlueRendererID RendererID (of PlayerBlue→AirVR Server Audio Output)

Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Air VR Server Audio Output Router (Script)
Input: Audio Plugin
Target Audio Mixer: AdvancedAudio
Exposed Renderer ID P: PlayerBlueRendererID
Output: One
Target Camera Rig: AirVRCameraRig (AirVRStereoCa

BUILD

onAirVR Server for Unity supports only 64bit architecture. So you must set ‘Architecture’ in Build Settings to ‘x86_64’ before you build.



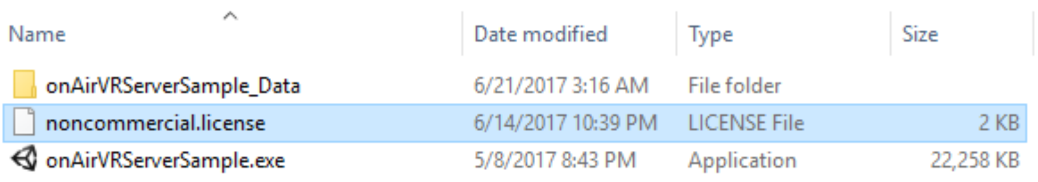
Target Platform Windows
Architecture x86_64
Development Build
Autoconnect Profiler
Script Debugging

Figure 14. ‘Architecture’ must be set to x86_64 in Build Setting.

And you have to distribute the built application with dependent libraries as below :

- **cudart64_80.dll**
  - *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\bin\cudart64_80.dll*

Finally, an onAirVR server license file must be in the directory the executable is in. You can use *noncommercial.license* - which is in *Assets/onAirVR/Server/Editor/Misc/* - included in our package, only for noncommercial purpose.



Name	Date modified	Type	Size
onAirVRServerSample_Data	6/21/2017 3:16 AM	File folder	
noncommercial.license	6/14/2017 10:39 PM	LICENSE File	2 KB
onAirVRServerSample.exe	5/8/2017 8:43 PM	Application	22,258 KB

Figure 15. You can copy *noncommercial.license* file into the directory the executable is in.

BEST PRACTICES

Setting User ID In onAirVR App

There might be cases where an onAirVR server application need to know actually what onAirVR client is connected. For example if you plan to integrate an outside-in positional tracking system into your onAirVR contents for multiple players, you need to match an onAirVR client to the rigid body which tracks the HMD of that client in the positional tracking system. We provide the way to assign a User ID to an onAirVR client to solve such problems.

This is how to assign a User ID in onAirVR App :

1. Swipe up Touchpad while focusing IP Address input field, then User ID input field appears.
2. Enter a number as a User ID which you want to assign, then click Play button.
3. You can get the User ID from AirVRClientConfig.userID property.

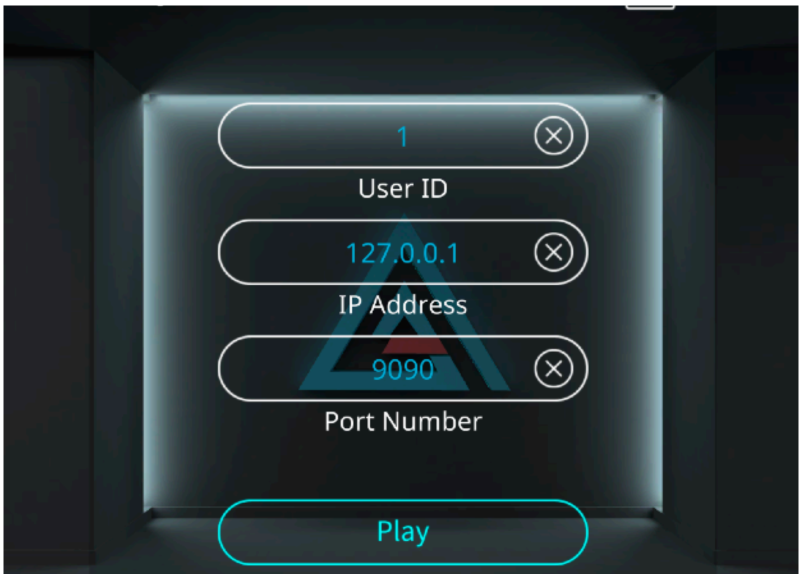


Figure 16. User ID input field in onAirVR App

REFERENCES

AirVRServer

Fundamental component responsible for onAirVR server startup/shutdown, video encoder and network configuration and client connection management. It also includes methods to send audio.

Static Variables		
Delegate	AirVRServer.EventHandler	instance of interface through which events are notified

Static Functions

```
void SendAudioFrame(AirVRCameraRig cameraRig, float[] data, int sampleCount, int channels, double timestamp)
void SendAudioFrameToAllCameraRigs(float[] data, int sampleCount, int channels, double timestamp)
```

sends raw audio data directly to the client of a camera rig.

- Parameters
  - cameraRig : camera rig to which you want to send audio data
  - data : raw audio data in which each sample is represented in float
  - sampleCount : the number of samples in data
  - channels : the number of channels. the first two channels are considered as left and right channels.
  - timestamp : the timestamp of audio data (in seconds).

AirVRServerInitParams

Overrides onAirVR server configuration if this component exists in the scene where AirVRServer is started up. Please read “Programming Guide - Server Configuration” section to see how to use in detail.

Variables		
licenseFilePath	string	the path of an onAirVR server license file which the built executable will use. (In editor, *Assets/onAirVR/Server/Editor/Misc/noncommercial.license* is used always. See “Build” section for detail.)
maxClientCount	int	the maximum number of clients which can be connected simultaneously
port	int	onAirVR server port number
videoBitrate	int	bitrate in which to encode video for each client (bps)
maxFrameRate	float	maximum frame rate in which to encode video for each client (fps)
defaultFrameRate	float	default frame rate in which to encode video for each client (fps)

AirVRServer.EventHandler

Interface through which AirVRServer events are notified. You might implement this interface and set to AirVRServer.Delegate to do something for events on server errors or client connections. The below figure describes what events are occurred when in AirVRServer.

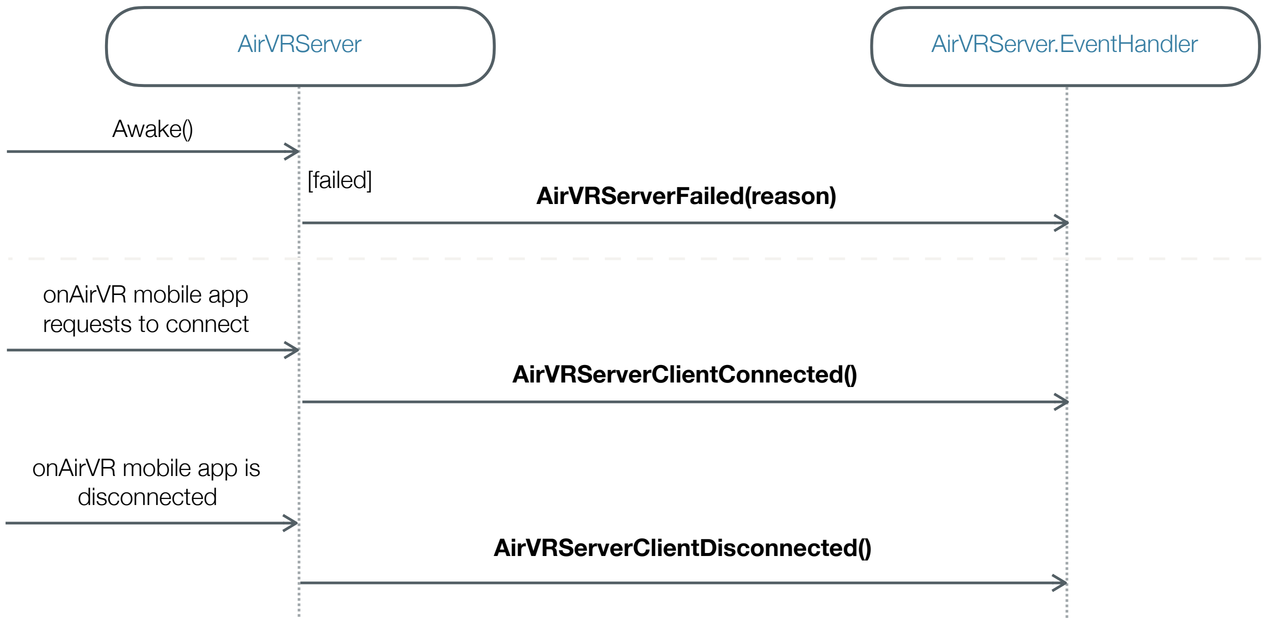

AirVRServer
AirVRServer.EventHandler
Awake()
[failed]
AirVRServerFailed(reason)
onAirVR mobile app requests to connect
AirVRServerClientConnected()
onAirVR mobile app is disconnected
AirVRServerClientDisconnected()

Figure 17. This sequence diagram describes what AirVRServer events are occurred when, through AirVRServer.EventHandler.

Public Functions	
void AirVRServerFailed(string reason)	called when onAirVR server fails to startup • Parameters - reason : reason why startup is failed
void AirVRServerClientConnected (int clientHandle)	called when a client is connected. • Parameters - clientHandle : the connection handle of the connected client
void AirVRServerClientDisconnected (int clientHandle)	called when a connected client is disconnected. • Parameters - clientHandle : the connection handle of the client

AirVRCameraRigManager

Is responsible for binding AirVRCameraRigs in a scene to the sessions of connections to onAirVR clients. It is guaranteed that there is an AirVRCameraRigManager instance in a scene as long as the scene contains more than one AirVRCameraRig.

Static Variables		
Delegate	AirVRCameraRigManager.EventHandler	instance of interface through which events are notified
managerOnCurrentScene	AirVRCameraRigManager	the instance on the current scene

AirVRCameraRigManager.EventHandler

Interface through which AirVRCameraRigManager events are notified. You might implement this interface and set to AirVRCameraRigManager.Delegate to do something for events occurred for AirVRCameraRig. The below figure describes what events are occurred in AirVRCameraRigManager.

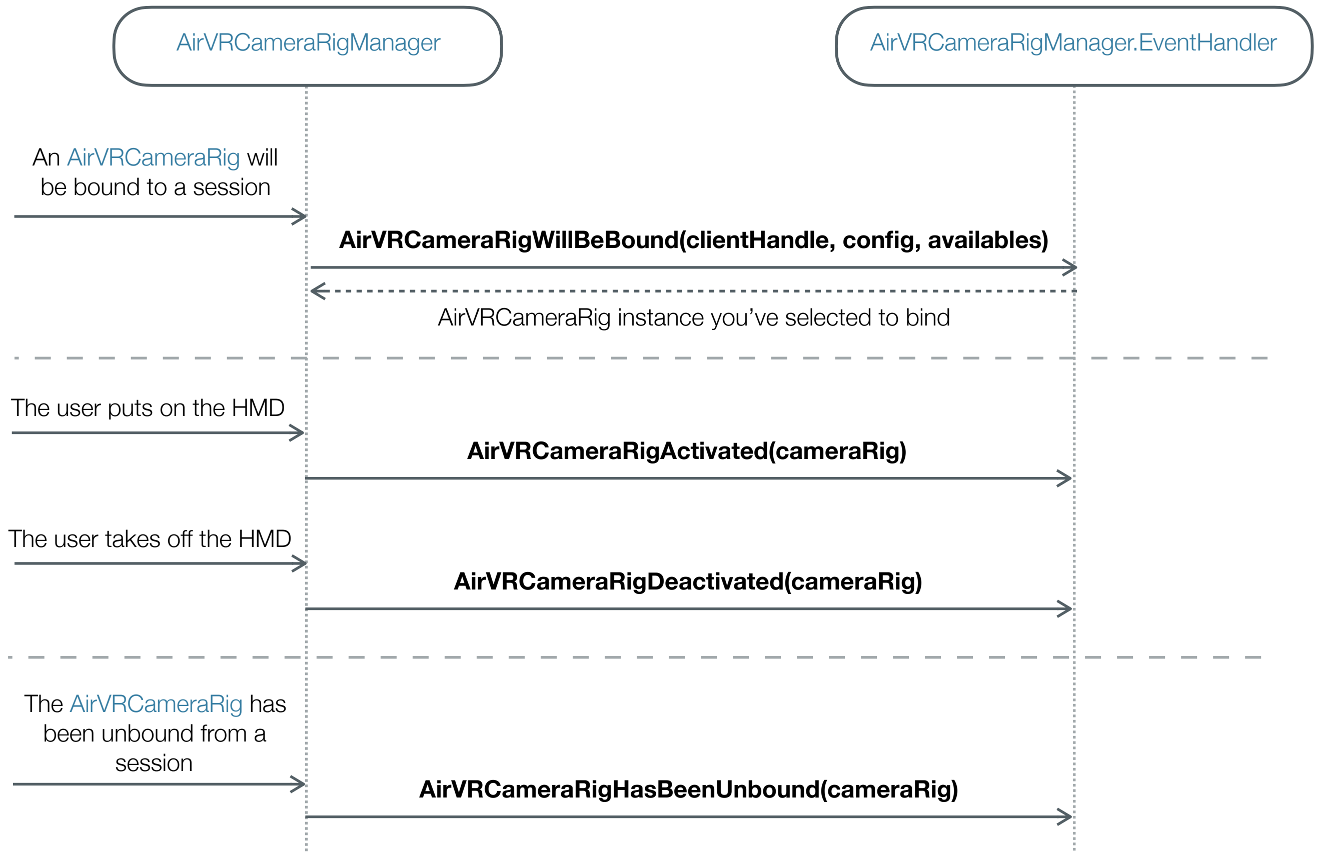

AirVRCameraRigManager
AirVRCameraRigManager.EventHandler
An AirVRCameraRig will be bound to a session
AirVRCameraRigWillBeBound(clientHandle, config, availables)
AirVRCameraRig instance you've selected to bind
The user puts on the HMD
AirVRCameraRigActivated(cameraRig)
The user takes off the HMD
AirVRCameraRigDeactivated(cameraRig)
The AirVRCameraRig has been unbound from a session
AirVRCameraRigHasBeenUnbound(cameraRig)

Figure 18. This sequence diagram describes what AirVRCameraRigManager events are occurred when, through AirVRCameraRigManager.EventHandler.

Public Functions	
void AirVRCameraRigWillBeBound (int clientHandle, AirVRClientConfig config, List<AirVRCameraRig> availables, out AirVRCameraRig selected)	called just before an AirVRCameraRig is bound to a session of AirVRServer. You must choose an instance of AirVRCameraRig and assign it to parameter *selected*, which might be one in parameter *availables* or one you've just instantiated in this method. • Parameters - clientHandle : the connection handle of the connected client - config : configuration for the connected client - availables : AirVRCameraRig instances in your scene which are not bound to any session yet - selected : the AirVRCameraRig instance you've selected to bind. Assign *null* if you want to reject this binding.
void AirVRCameraRigActivated (AirVRCameraRig cameraRig)	called when a user puts on the HMD. • Parameters - cameraRig : AirVRCameraRig instance bound to the client of the user
void AirVRCameraRigDeactivated (AirVRCameraRig cameraRig)	called when a user takes off the HMD. • Parameters - cameraRig : AirVRCameraRig instance bound to the client of the user
void AirVRCameraRigHasBeenUnbound (AirVRCameraRig cameraRig)	called just after an AirVRCameraRig has been unbound from a session. • Parameters - cameraRig : AirVRCameraRig instance which was bound to the session

AirVRClientConfig

Configuration values requested from a connected client, such as the arrangement of eyes, field of view (FOV), frame rate of video, etc.

Variables		
type	AirVRClientType	the type of the client (read only). Always "Stereoscopic" currently.
videoWidth	int	the width of video to send to the client
videoHeight	int	the height of video to send to the client
framerate	float	frame rate of encoded video to send to the client
fov	float	vertical field of view of cameras
eyeCenterPosition	Vector3	offset from the root to CenterEyeAnchor in AirVRCameraRig
ipd	float	distance between the left and right eyes
userID	string	user ID which is specified in onAirVR App. Read "Best Practice" section for detail.

AirVRCameraRig

Abstract base class for manipulating cameras which render video frames to send to onAirVR clients.

Variables		
type	AirVRClientType	the type of AirVRCameraRig (read only). Always “Stereoscopic” currently.
isBoundToClient	bool	true if this is bound to a client through a session currently (read only).

Public Functions	
AirVRClientConfig GetConfig()	returns the configuration values requested from the client this camera rig is bound to.
void AdjustBitrate(uint bitrateInKbps)	adjusts encoding bit rate of video frames to send to the client, in “kbps”. (1 kbps = 1000 bps)
void RecenterPose()	recenters the pose of the client.
void Disconnect()	disconnects from the connected client this camera rig is bound to.

AirVRStereoCameraRig (inherits from AirVRCameraRig)

A set of cameras which render stereoscopic video frames to send to a client of “Stereoscopic” type.

Variables		
leftEyeCamera	Camera	Camera which acts as the left eye. This renders the left side of a video frame for stereoscopic video type.
rightEyeCamera	Camera	Camera which acts as the right eye. This renders the right side of a video frame for stereoscopic video type.
leftEyeAnchor	Transform	Transform of leftEyeCamera
centerEyeAnchor	Transform	Transform of centerEyeCamera
rightEyeAnchor	Transform	Transform of rightEyeCamera
trackingModel	TrackingModel	defines how eye cameras are tracked along with HMD pose. See “Programming Guide - Camera Rig Tracking Models” section for detail.
externalTrackingOrigin	Transform	the origin of external motion tracking system. If it is null, the world origin is used. Valid only if *trackingModel* is ExternalTracker.
externalTracker	Transform	a tracker transform in external motion tracking system. Used only if *trackingModel* is ExternalTracker.

AirVRInput

Provides methods to get values of input devices - including GearVR Touchpad, Xbox Controller and GearVR Controller - of the client an AirVRCameraRig are bound to. Please see "Programming Guide - Input" section to check what each button or axis is bound to a control in an input device.

Static Functions	
bool IsDeviceAvailable(AirVRCameraRig cameraRig, AirVRInput.Device device)	returns true if an input device is available on the client which cameraRig is bound to. • Parameters - cameraRig : AirVRCameraRig instance bound to a client - device : the type of the input device ✓ AirVRInput.Device - HeadTracker (just tracks the head pose) - Touchpad - Gamepad - TrackedController
void GetTrackedDevicePositionAndOrientation (AirVRCameraRig cameraRig, AirVRInput.Device device, out Vector3 worldPosition, out Quaternion worldOrientation)	gets the "world" position and orientation of a tracked device. • Parameters - cameraRig : AirVRCameraRig instance bound to a client - device : the type of the input device. HeadTracker and TrackerController are available only. - worldPosition / worldOrientation : out parameters for world position and orientation of the input device
Vector2 Get(AirVRCameraRig cameraRig, AirVRInput.Touchpad.Axis2D axis) bool Get(AirVRCameraRig cameraRig, AirVRInput.Touchpad.Button button) Vector2 Get(AirVRCameraRig cameraRig, AirVRInput.Gamepad.Axis2D axis) float Get(AirVRCameraRig cameraRig, AirVRInput.Gamepad.Axis axis) bool Get(AirVRCameraRig cameraRig, AirVRInput.Gamepad.Button button) Vector2 Get(AirVRCameraRig cameraRig, AirVRInput.TrackedController.Axis2D axis) bool Get(AirVRCameraRig cameraRig, AirVRInput.TrackedController.Button button)	returns the value of an axis or a button of an input device of the client which *cameraRig* is bound to. • Parameters - cameraRig : AirVRCameraRig instance bound to a client - axis/button : one of axes/buttons of an input device

Static Functions	
bool GetDown(AirVRCameraRig cameraRig, AirVRInput.Touchpad.Button button) bool GetDown(AirVRCameraRig cameraRig, AirVRInput.Gamepad.Button button) bool GetDown(AirVRCameraRig cameraRig, AirVRInput.TrackedController.Button button)	returns true if a button of an input device of the client which *cameraRig* is bound to is pressed during the frame. • Parameters - cameraRig : AirVRCameraRig instance bound to a client - button : one of buttons of an input device
bool GetUp(AirVRCameraRig cameraRig, AirVRInput.Touchpad.Button button) bool GetUp(AirVRCameraRig cameraRig, AirVRInput.Gamepad.Button button) bool GetUp(AirVRCameraRig cameraRig, AirVRInput.TrackedController.Button button)	returns true if a button of an input device of the client which *cameraRig* is bound to is released during frame. • Parameters - cameraRig : AirVRCameraRig instance bound to a client - button : one of buttons of an input device

AirVRServerAudioOutputRouter

Routes “stereo” audio rendered by Unity audio engine to connected clients. Must be attached to a GameObject on which Unity’s AudioListener is attached. Please read “Programming Guide - Audio” section how it works.

Variables		
input	Input	audio source to send to connected clients • Input - AudioListener · takes stereo audio heard by Unity’s AudioListener as input - AudioPlugin · takes stereo audio passing through “AirVR Server Audio Output” audio plugin as input
output	Output	destination to which onAirVR server will send audio • Output - All · broadcast audio to all connected clients - One · send audio only to a specific client
targetAudioMixer	UnityEngine.Audio.AudioMixer	Unity’s AudioMixer containing “AirVR Server Audio Output” audio plugin. Used only when input is “AudioPlugin”.
exposedRendererIDParameterName	string	exposed parameter name of RendererID parameter of “AirVR Server Audio Output” audio plugin in targetAudioMixer. Used only when input is “AudioPlugin”.
targetCameraRig	AirVRCameraRig	AirVRCameraRig instance bound to a client to which want to send audio. Used only when output is “One”.

CONTACT

If you are looking for answers, would like to solve a problem, or just want to let us know how we can cooperate, please contact us via the below channels.

- onAirVR website : http://onairvr.io
- Email for support : contact@onairvr.io
- We are Clicked : http://www.clicked.co.kr