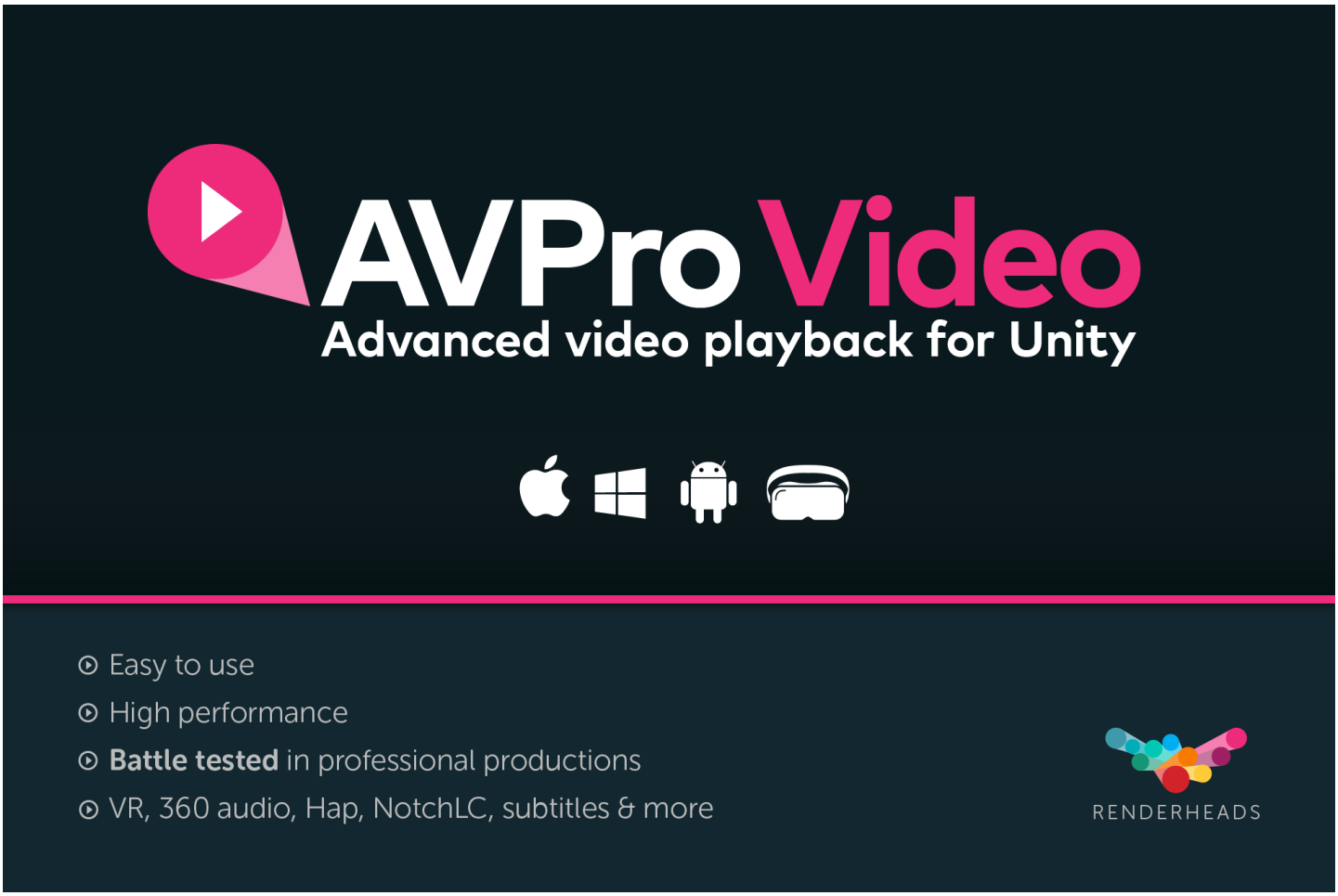


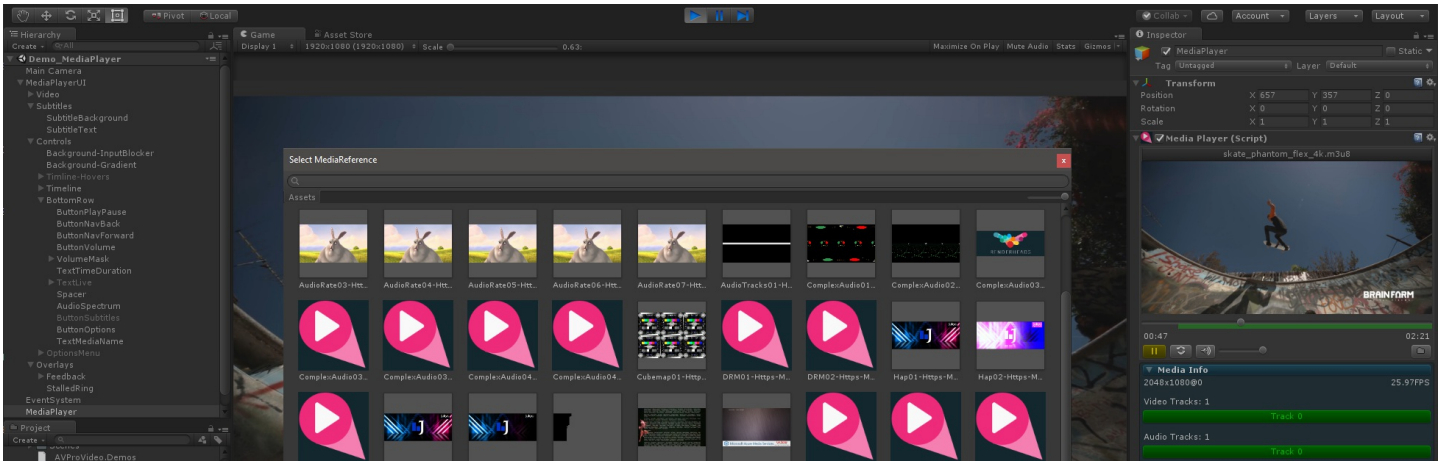
For full documentation, visit the AVPro Video Developer Portal

Table of Contents

- Articles
  - About
    - Introduction
    - What's New
    - Features
    - Requirements
    - Download
    - Asset Files
  - Getting Started
    - Installation
    - Upgrading Projects
    - Quick Start
    - Loading Media
    - Demos
    - Shaders
  - Core Features
    - Events
    - Streaming
    - AR/VR/XR
    - High Res
    - Stereo Video
    - Transparency
    - Subtitles
    - Video Capture
    - Content Protection
    - Seeking & Playback Rate
  - Ultra Features
    - 360 Audio
    - 10-bit Video
    - Hap Codec
    - NotchLC Codec
    - Content Protection
    - Caching

- Smooth Video
- Components
  - Media Player
  - Media Reference
  - Apply To Material
  - Apply To Mesh
  - Update Multipass Stereo
  - Display IMGUI
  - Display UGUI
  - Resolve To RenderTexture
  - Audio Output
  - Audio Channel Mixer
  - Playlist Media Player
  - Subtitles UGUI
- Extension Components
  - Apply To VFX Graph
  - Timeline Playables
- Platform Notes
  - Windows
  - Android
  - macOS
  - iOS/iPadOS/tvOS
  - UWP
  - WebGL
- Media
  - Supported Media
  - Encoding Notes
- Known Issues
- FAQ

Introduction



AVPro Video is a powerful cross-platform video playback plugin for Unity created by RenderHeads

We created this plugin primarily for use with our internal projects. At RenderHeads we create interactive installations for events and educational games for museums, so video playback is an important aspect of our work. We found that the built-in video functionality in Unity didn't cater to many of the use cases we had, so since 2012 we have been building video related plugins for Unity.

AVPro Video is the latest in a series of video playback plugins and it leverages the specific features of the different platforms it supports, while at the same time being easy to use and highly performant.

What's New

What's New in AVPro Video 2.0

For a quick and easy comparison, some of the main differences compared to the previous AVPro Video 1.x plugin are summarised in this table:

	AVPRO VIDEO 2.0	AVPRO VIDEO 1.X
First released	**2021 (January)**	2016 (February)
New Features		
Unity 2020 & 2021 support	**Yes**	No
Direct3D 12 support	**Yes** [1]	No
NotchLC codec support on Windows	**Yes** [2,]	No
Stereo metadata video on Windows	**Yes**	No
Stereo packing detection	**Yes**	No
Unity audio playback on Android	**Yes**	No
Multi-track support	**Yes**	No
Subtitle track support	**Yes**	No
Frame-based time methods	**Yes**	No
Asset-based media (MediaReference)	**Yes**	No
Edit-mode media playback	**Yes**	No
Apple Silicon architecture support	**Yes**	No
10-bit video support	**Yes** [4]	No
Improved Features		
Low overhead macOS/iOS/tvOS plugin	**Yes**	No
Support for modern Unity features (VfxGraph, Timeline etc)	**Yes**	No
Upgraded to latest APIs (ExoPlayer, Audio 360, Hap etc)	**Yes**	No
YCbCr optimisation support on macOS	**Yes**	No
ASync Hap codec on Windows	**Yes** [2]	No

	AVPRO VIDEO 2.0	AVPRO VIDEO 1.X
Adaptive streaming support	**Advanced**	Basic
Content protection support	**Advanced** [2]	Basic
Deprecated Features		
Legacy Windows XP	**No**	Yes
Legacy Windows Phone	**No**	Yes
Legacy Unity 4.x support	**No**	Yes
Legacy Unity 5.x support	**No**	Yes
Legacy Unity 2017.x support	**No**	Yes
WebGL support	**No** [3]	Yes
PlayMaker actions included	**No**	Yes
NGUI component included	**No**	Yes
Asset Changes		
New Ultra Edition for advanced users	**Yes** [2]	No
macOS, iOS and tvOS plugins combined into a single edition	**Yes**	No

Why update to AVPro Video 2.0?

Modernised

- Now with support for D3D12
- Upgraded to using latest APIs for best modern features support, eg ExoPlayer, Hap, Facebook Audio 360
- New WinRT player on Windows
- Shaders and scripts upgraded to use modern Unity features

Workflow improvements

- Edit-mode media playback allows media to be played without running the scene
- New MediaReference asset type allows easy defining of media

Audio improvements

- Android can now play audio back to Unity
- Windows supports 16 channel PCM audio
- Demo MediaPlayerUI includes audio spectrum visualisation

Powerful streaming options

- New option to force adaptive stream to start at the highest bitrate on Android and Windows (using WinRT API)
- New WinRT has improved streaming support
- Android, macOS, iOS and tvOS expose options for specifying the preferred max bitrate and resolution

- Android exposes new options for controlling buffering behaviour

Better integration with Unity features

- Ready-to-use components for integration with the Unity features:
  - Timeline / Playables
  - Shader Graph
  - Visual Effect Graph
- SRP / HDRP / UWP
  - The new texture resolver option makes it easier to integrate with other shaders / features / components

Timed Text

- Subtitles support
- ~~ID3 / metadata support~~ (planned)

Long awaited features

- Stereo videos now work correctly on Windows without having to remove metadata
- Frame-centric methods
- Multiple video and audio track support
- Textures now persist between video loads, allowing for a better visual experience and less resource reallocation (WinRT only currently)

Improved Demos

- Fewer demos, but higher quality
- Focused on helping people get started with common use cases

Better Components

- MediaPlayer can now play media without having to enter Play mode, improving workflow
- Many MediaPlayer UI improvements

Better Code

- The C# code has been refactored to give a cleaner and easier to work with API
- Scripts are now organised using assembly definition (asmdef) files, for improved modularity, better interoperability with modern Unity features, and reduce compile time

Documentation improvements

- Manual and scripting documentation completely updated
- Changed from PDF to HTML

Continued support and development

- Expect the same high level of support and longevity you've come to expect from other RenderHeads products
- New features will continue to be added, with customer feature requests taken into high consideration
- Customer support will continue to be a top priority, just as it always has been
- Just as we have done before, we will continue maintaining the product through to new versions of Unity

Better content protection options [2]

- Encrypted video via AES-128 HLS (now also supported on Windows via WinRT API)
- Custom HTTP headers (now also supported on Windows via WinRT API)

Professional Codec Support [2]

- Hap codec support on Windows now supports Media Foundation API, with faster non-blocking loading
- New 10-bit NotchLC codec support on Windows
- Improved support for 10-bit videos [4]

Legacy Removal

- Legacy support removed we can focus on the modern features
- Removed support for Unity 4.x, Unity 5.x and Unity 2017.x
- Windows
    - Removed support for legacy Windows Metro (Phone and 8.1)
    - Removed support for Windows XP
- macOS
    - Minimum version changed to 10.12
    - Dropped OpenGL support
- iOS / tvOS
    - Dropped 32-bit support
    - Dropped OpenGLES support
    - Minimum version changed to 11.0

[1] Direct3D12 support requires minimum Unity 2019.3

[2] Only in the Ultra Edition

[3] WebGL plugin still included, but we do not support it (see WebGL platform notes)

[4] 10-bit video support has some elements that only work in the Ultra Edition and has other limitations as documented

Features

A summary of the features:

- Compatibility
  - Unity 2018.x - 2021.x and above supported
  - Cross-platform with versions for iOS, tvOS, macOS, Android, Windows, UWP
  - Optional fallback to Unity's VideoPlayer (coming soon)
  - Graceful fallback for unsupported platforms

- Easy to use
  - One API for all supported platforms
  - Easy to use, drag and drop components
  - Scriptable API

- Powerful features
  - Playback of local files, URL files and adaptive streaming formats
  - VR support (mono, stereo, equirectangular and cubemap)
  - Transparency support (native and packed)
  - Linear and Gamma colour spaces supported
  - 8K video support (on supported hardware)
  - Streaming bitrate controls (coming soon)

- Advanced features
  - Encrypted video playback via AES HLS
  - Custom HTTP headers for secure playback
  - Support for industry codecs such as Hap, Hap Alpha, Hap Q and Hap Q Alpha (supporting 16K+)

- Highly optimised
  - GPU video decoding
  - Optimised native Direct3D, OpenGL and Metal support
  - Focus on minimal garbage generation

- Extensible
  - Components for IMGUI, Unity UI (and maybe NGUI later)
  - Components for Timeline/Playables, Shader Graph, Visual Effect Graph
  - Support for SRP (URP / HDRP)

- Well supported
  - Free watermarked trial version available (download here)
  - Good documentation
  - Public issue tracker

Requirements

System Requirements

Unity

- 2018.x, 2019.x, 2020.x, 2021.x, 2022.x

Platforms

- Apple
  - macOS 10.13 and above, 64bit only, Metal only
  - iOS 10.0 and above, 64bit only, Metal only
  - tvOS 11.0 and above, Metal only
- Microsoft
  - Windows 7, 8, 8.1, 10 and above (x86 and x86_64)
  - Universal Windows Platform (UWP) 10 (x86 and x86_64, ARM and ARM64)
  - Direct3D 11 and 12
- Android
  - Android 4.3.0 (Jelly Bean, API level 18) and above (ARM7, ARM64, x86 and x86_64)
  - OpenGLES2 and OpenGLES3

Platforms not Supported

- Linux desktop
- Lumin (Magic Leap)
- TV platforms (Tizen, Samsung TV etc)
- Game Consoles (XBox*, PlayStation, Switch etc)

** XBox One may be supported using the UWP build option. We have not tested this though.*

VR / AR / MR / XR Headsets Supported

Android

- Oculus Rift Go
- Oculus Gear VR
- Oculus Quest
- Oculus Quest 2
- HTC Vive Focus
- HTC Vive Focus Plus
- HTC Vive Focus 3
- Gear VR
- Google Cardboard
- Google Daydream
- Pico Goblin
- Pico G2 4K
- Pico Neo 2
- Pico Neo 3
- Lenovo Mirage Solo

Windows Desktop

- HTC Vive

- HTC Vive Pro
- HTC Vive Cosmos
- Valve Index
- Oculus Rift
- Oculus Rift S
- StarVR

Windows UWP

- Microsoft Hololens
- Microsoft Hololens 2
- Samsung Odyssey
- Asus
- HP
- Acer
- Lenovo Explorer
- Dell Visor

Downloads, Editions & Upgrades

Free Trial Version

1. Fully featured watermarked trial versions can be downloaded here: https://github.com/RenderHeads/UnityPlugin-AVProVideo/releases
2. Once the .unitypackage file has been downloaded, follow the installation instructions.

Purchase

All editions of AVPro Video can be purchased via the Unity Asset Store:

- Core Edition (core features for all platforms)
  - Core Windows Edition (core features for Windows and UWP platform)
  - Core Android Edition (core features for Android platform)
  - Core macOS/iOS/tvOS Edition (core features for Apple platforms)
- Ultra Edition (advanced features for all platforms)
- Enterprise Edition (for larger companies with multiple sites/offices/subsidiaries)

Editions

We've made several editions of AVPro Video available so you can pick the one that's best for your project:

	TRIAL EDITION	CORE WINDOWS EDITION	CORE ANDROID EDITION	CORE MACOS/IOS/TVOS EDITION	CORE EDITION	ULTRA EDITION	ENTERPRISE EDITION
Platforms							
Windows standalone	Watermarked	**Yes**	Watermarked	Watermarked	**Yes**	**Yes**	**Yes**
Windows UWP	Watermarked	**Yes**	Watermarked	Watermarked	**Yes**	**Yes**	**Yes**
macOS standalone	Watermarked	Watermarked	Watermarked	**Yes**	**Yes**	**Yes**	**Yes**
iOS	Watermarked	Watermarked	Watermarked	**Yes**	**Yes**	**Yes**	**Yes**
tvOS	Watermarked	Watermarked	Watermarked	**Yes**	**Yes**	**Yes**	**Yes**
Android	Watermarked	Watermarked	**Yes**	Watermarked	**Yes**	**Yes**	**Yes**
Core Features							
All Core features	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Android OES Texture support	**No**	**No**	**Yes**	**No**	**Yes**	**Yes**	**Yes**

	TRIAL EDITION	CORE WINDOWS EDITION	CORE ANDROID EDITION	CORE MACOS/IOS/TVOS EDITION	CORE EDITION	ULTRA EDITION	ENTERPRISE EDITION
Ultra Features							
Hap Codec [1]	Watermarked	Watermarked	Watermarked	Watermarked	Watermarked	**Yes**	**Yes**
NotchLC Codec [2]	Watermarked	Watermarked	Watermarked	Watermarked	Watermarked	**Yes**	**Yes**
Custom Http Headers	**Yes**	No	No	No	No	**Yes**	**Yes**
AES-128 HLS	**Yes**	No	No	No	No	**Yes**	**Yes**
Spatial Audio [3]	**Yes**	No	No	No	No	**Yes**	**Yes**
10-bit Textures [4]	**Yes**	No	No	No	No	**Yes**	**Yes**
Caching [5]	**Yes**	No	No	No	No	**Yes**	**Yes**
Smooth Video [6]	**Yes**	No	No	No	No	**Yes**	**Yes**
Other							
Support Priority	Normal	Normal	Normal	Normal	Normal	**High**	**Very High**
Multi-site license	N/A	No	No	No	No	No	**Yes**
Price	Free	$200	$200	$200	$400	$800	$5000
Link	Download	Store	Store	Store	Store	Store	Contact Us

[1] Hap Codec only supported on Windows and macOS platforms.

[2] NotchLC Codec only supported on Windows platform.

[3] Spatial Audio only supported on Windows and Android platforms.

[4] 10-bit Textures only supported on Windows, macOS, iOS and tvOS platforms.

[5] Caching only supported on Android and iOS platforms.

[6] Experimental Feature. Only on Windows.

Upgrade Paths

In many cases a developer may own one edition and as their needs increase may need the features of other editions. To cater for this we have set up some upgrade paths on the Unity Asset Store to make this cost effective:

Owners of AVPro Video 1.x

NOTE

Despite AVPro Video 1.x being deprecated, existing owners of this version can still access it for download via the Unity Asset Store. Make sure that you're logged in with the same account that you used to originally purchase it and it should be visible in your list to download the last published version.

IMPORTANT

AVPro Video 2.x has many API changes, so upgrading a project from AVPro Video 1.x is not recommended or should be done carefully at least. There are also feature differences between version 1.x and 2.x which are summarised in the table on the What's New page. If you do plan to upgrade your existing project then do follow the migration guide.

There is **discounted** pricing automatically applied (for a limited time) for owners of the retired product **AVPro Video 1.x**:

Upgrade Pricing

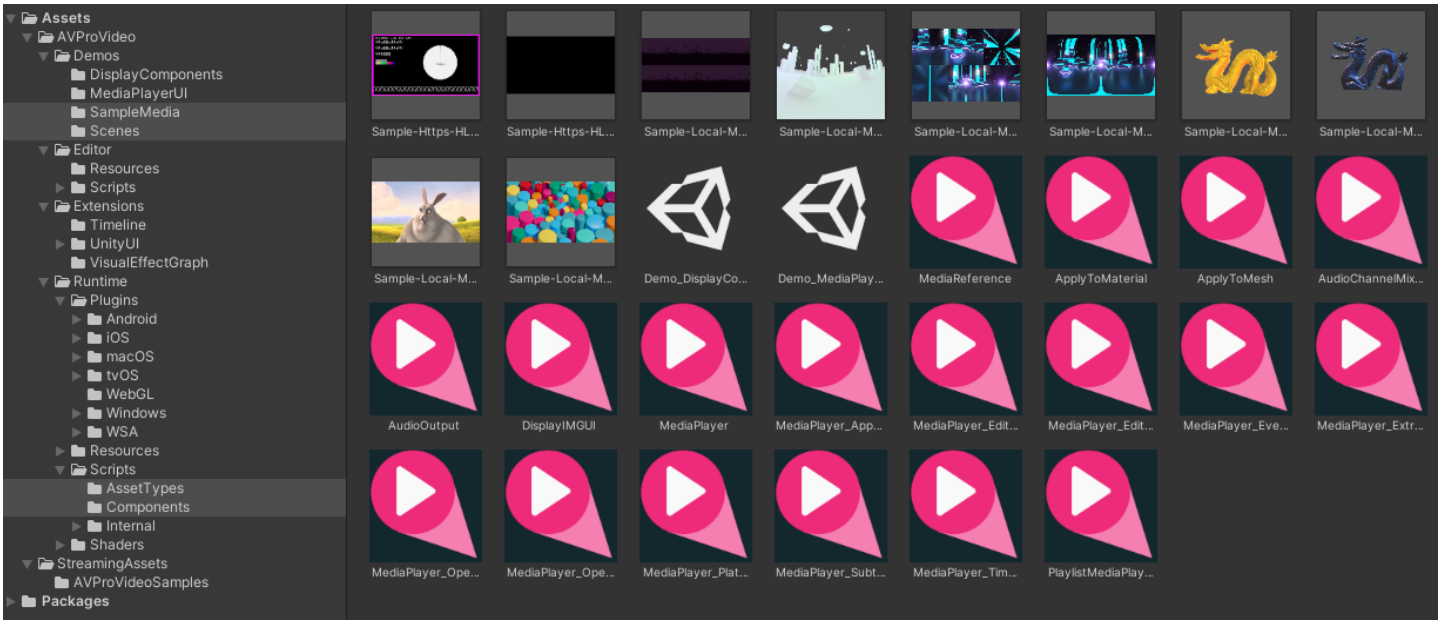
	CORE WINDOWS EDITION	CORE ANDROID EDITION	CORE MACOS/IOS/TVOS EDITION	CORE EDITION	ULTRA EDITION	ENTERPRISE EDITION
Full Price:	**$200**	**$200**	**$200**	**$400**	**$800**	**$5000**
Upgrading from another edition:						
Core Windows Edition	.	.	.	$200	.	.
Core Android Edition	.	.	.	$200	.	.
Core macOS/iOS/tvOS Edition	.	.	.	$200	.	.
Core Edition	.	.	.	.	$400	$4600
Ultra Edition	.	.	.	.	.	$4200
Upgrading from AVPro Video 1.x:						
Windows Edition	$100	.	.	$300	$700	.
Android Edition	.	$100	.	$300	$700	.
macOS Edition	.	.	$100	$300	$700	.
iOS Edition	.	.	$100	$300	$700	.
tvOS Edition	.	.	$100	$300	$700	.
Full Edition	.	.	.	$200	$600	.
Enterprise Edition	.	.	.	.	.	$3000

Future Updates

We plan to continue supporting this product until the next major version (3.0.0). This will include bug fixes, new features and improvements.

Each major release will be actively supported for a minimum of 2 years, and up until the release of the next major version. We feel that being open about our strategy for upgrades is beneficial to developers planning their projects. Owners of the current 2.x version will get a discounted price when upgrading to 3.x

Asset Files



The AVPro Video asset package contains files organised into 5 major folders:

1. Runtime Folder
2. Extensions Folder
3. Demos Folder
4. Editor Folder
5. StreamingAssets Folder

Runtime Folder

This folder contains the primary elements of AVPro Video. It builds to the AVProVideo.Runtime assembly.

- **Plugins/**

  Contains all native plugin files

- **Scripts/**

  - **AssetTypes/**

    ScriptableObject for defining media

  - **Components/**

    Monobehaviour Components

  - **Internal/**

    Internal code that is not usually modified by users

- **Resources/Textures**

  Textures used by the NullMediaPlayer and always included in builds

- **Shaders/**

  This base folder contains all the optional shaders which will be stripped out if they are not used. It also contains the important .cginc file shared by all shaders

- **Resources/**

  Internal shaders that are always included in builds

Extensions Folder

This folder contains files for supporting other packages, Unity extensions or other assets.

- **Timeline/**

  Adds support for the Playables Timeline feature

- **UnityUI/**

  Adds support for displaying video to Unity's UI system

- **VisualEffectGraph/**

  Adds support for sending video to Unity's VFX graph

Demos Folder

This folder contains several examples scenes showing how to use some of the components included with AVPro Video. This folder builds to its own assembly and is optional.

- **MediaPlayerUI/**

  Complete scripts and shaders for a fully functional advanced UI for media playback.

- **SampleMedia/**

  MediaReference asset files for the sample media (local files and streaming URLs)

- **Scenes/Demo_MediaPlayer.unity**

  A comprehensive example scene playing back media with AVPro Video and using the MediaPlayerUI to help demonstrate features like subtitles, seeking, Unity audio, streaming etc.

- **Scenes/Demo_DisplayComponents.unity**

  An example scene showing all of the different components used to display and manipulate videos.

- **Scenes/Demo_360Stereo.unity**

  An example scene showing 360 equirectangular stereo video applied to a sphere.

Editor Folder

This folder contains editor-only scripts which are not usually modified by users. It builds to the AVProVideo.Editor assembly.

StreamingAssets Folder

This folder contains sample media files included for demonstration purposes that the demo scenes rely on. The contained AVProVideoSamples subfolder is optional and only used for our demo scenes.

Installation

Installation Steps

If you are installing from scratch:

1. Either download the latest trial version: https://github.com/RenderHeads/UnityPlugin-AVProVideo/releases
   or
   Purchase the latest version from the Unity Asset Store
2. Open your project in Unity
3. Import the `unitypackage` file into your Unity project by double-clicking the file
4. If prompted to upgrade some scripts click Yes

Upgrade Steps

If you are upgrading to a new version:

1. Close your Unity project
2. Open your project again and proceed to the next step immediately without running the scene or clicking on any AVPro Video components, as this can cause the plugin files to become locked
3. Import the `unitypackage` file into your Unity project by double-clicking the file
4. If prompted to upgrade some scripts click Yes

Watermarked Trial Version

If you are using a trial version of the plugin then you will see a watermark displayed over the video. The watermark is in the form of a "RenderHeads" logo that animates around the screen, or a thick horizontal bar that moves around the screen.

Installation & Watermark Troubleshooting

It's often a good idea to check that the correct version is reported after a plugin upgrade. You can check which version you have installed by adding an MediaPlayer component to your scene and clicking on the "About / Help" button in the Inspector for that component. The version number is displayed in this box.

The Core and Ultra editions of AVPro Video have no watermarks for any platforms. If you use one of the platform specific Core editions (eg AVPro Video - Core Windows Edition) then you will not see the watermark on the platform you purchased for, but you will see the watermark on the other platforms. For example if you purchased AVPro Video - Core Android Edition then you will still see the watermark in the Unity editor as this is running on Windows/macOS, but the videos played back when you deploy to your Android device will be watermark-free.

Installing Multiple Single-Platform Packages

If you are not using the AVPro Video Core or Ultra package and instead have opted to purchase multiple individual Core single-platform packages then the installation must be done carefully, especially when upgrading to a new version.

If you have installed the Core macOS/iOS/tvOS edition package then it will also contain plugins for all of the other platforms but with the watermark enabled. This means that if you then try to install another AVPro Video package it may not override the plugins correctly. Here is how to resolve this using the macOS/iOS/tvOS and Android package as examples:

1. Open a fresh Unity instance (this is important as otherwise Unity may have locked the plugin files which prevents them from being upgraded)
2. Import the macOS/iOS/tvOS package
3. Import the Android package, but make sure that you have the macOS/iOS/tvOS native plugin files unticked (so that it is not

overwritten) A similar process can be applied for other package combinations.

List of native plugin files that need to be selectively chosen when replacing specific platforms:

- Android
  - Plugins/Android/AVProVideo.jar
- macOS
  - Plugins/AVProVideo.bundle
- iOS
  - Plugins/iOS/AVProVideo.framework
- tvOS
  - Plugins/tvOS/AVProVideo.framework
- Windows
  - Plugins/WSA/UWP/ARM64/AVProVideo.dll
  - Plugins/WSA/UWP/ARM64/AVProVideoWinRT.dll
  - Plugins/WSA/UWP/ARM/AVProVideo.dll
  - Plugins/WSA/UWP/ARM/AVProVideoWinRT.dll
  - Plugins/WSA/UWP/x86/AVProVideo.dll
  - Plugins/WSA/UWP/x86/AVProVideoWinRT.dll
  - Plugins/WSA/UWP/x86_64/AVProVideo.dll
  - Plugins/WSA/UWP/x86_64/AVProVideoWinRT.dll
  - Plugins/x86/AVProVideo.dll
  - Plugins/x86/AVProVideoWinRT.dll
  - Plugins/x86_64/AVProVideo.dll
  - Plugins/x86_64/AVProVideoWinRT.dll

Upgrading from AVPro Video 1.x

If you have existing projects based on the legacy AVPro Video 1.x product, it is possible to upgrade them to AVPro Video 2.x using the steps below.

> **NOTE**
>
> It is not possible to have AVPro Video 1.x and 2.x within the same project as there would be conflicts.

Upgrading projects is not something we recommend as there have been a lot of changes between the two API versions. Before starting this process we recommend having a backup or source control for your project.

Scenes, Prefabs and Scripts

Existing scenes and prefabs should mostly upgrade correctly, as we have created serialisation fixups for most properties. Existing scripts will need to be manually updated to the new API using the reference below.

Upgrading

1. Close Unity and open your project up fresh (otherwise some of the plugin files will be locked if you have used the AVPro Video scripts or plugins during your editor session, which will prevent them from being deleted)
2. The old AVPro Video package must be deleted:
    1. Delete the old AVPro Video **plugins** folder (usually found at Assets/Plugins). When you do this you must be careful that you don't have other files in there from other packages.
    2. Delete the old AVPro Video **main** folder (usually found at Assets/AVProVideo)
    3. Optionally delete the old **sample videos** folder (usually found in /Assets/StreamingAssets/AVProVideoSamples)
3. Import the latest AVPro Video 2.x package
4. Fix script compile issues in your project using the list of API changes below

API Changes

A non-exhaustive list of API changes:

LEGACY 1.X	NEW 2.X	NOTES
Types		
class CubemapCube		Removed
class DisplayBackground		Removed
class DisplayBackground		Removed
class DebugOverlay		Removed
class UpdateStereoMaterial	class UpdateMultiPassStereo	
class ApplyToTextureWidgetNGUI		Removed
enum MediaPlayer.FileLocation	enum MediaPathType	

LEGACY 1.X	NEW 2.X	NOTES
Fields		
MediaPlayer.m_VideoPath	MediaPlayer.MediaPath.Path	Loading Notes
MediaPlayer.m_VideoLocation	MediaPlayer.MediaPath.PathType	Loading Notes
MediaPlayer.m_AutoOpen	MediaPlayer.AutoOpen	
MediaPlayer.m_AutoStart	MediaPlayer.AutoStart	
MediaPlayer.m_Loop	MediaPlayer.Loop	
MediaPlayer.m_Volume	MediaPlayer.AudioVolume	
MediaPlayer.m_Balance	MediaPlayer.AudioBalance	
MediaPlayer.m_Muted	MediaPlayer.AudioMuted	
MediaPlayer.m_PlaybackRate	MediaPlayer.PlaybackRate	
MediaPlayer.m_Resample	MediaPlayer.UseResampler	
MediaPlayer.m_ResampleMode	MediaPlayer.ResampleMode	
MediaPlayer.m_ResampleBufferSize	MediaPlayer.ResampleBufferSize	
MediaPlayer.m_StereoPacking		Replaced with FallbackMediaHints
MediaPlayer.m_AlphaPacking		Replaced with FallbackMediaHints
MediaPlayer.m_DisplayDebugStereoColorTint		Removed
MediaPlayer.m_FilterMode	MediaPlayer.TextureFilterMode	
MediaPlayer.m_WrapMode	MediaPlayer.TextureWrapMode	
MediaPlayer.m_AnisoLevel	MediaPlayer.TextureAnisoLevel	
MediaPlayer.m_LoadSubtitles	MediaPlayer.SideloadSubtitles	
DisplayUGUI._mediaPlayer	DisplayUGUI.CurrentMediaPlayer	
DisplayIMGUI._mediaPlayer	DisplayIMGUI.Player	
DisplayIMGUI._color	DisplayIMGUI.Color	
Methods		

LEGACY 1.X	NEW 2.X	NOTES
MediaPlayer.OpenVideoFromFile()	MediaPlayer.OpenMedia()	
MediaPlayer.CloseVideo()	MediaPlayer.CloseMedia()	
MediaPlayer.OpenVideoFromBuffer()	MediaPlayer.OpenMediaFromBuffer()	
MediaPlayer.StartOpenChunkedVideoFromBuffer()	MediaPlayer.StartOpenChunkedMediaFromBuffer()	
IMediaInfo.GetDurationMs()	IMediaInfo.GetDuration()	Time format changes
IMediaInfo.GetAudioTrackCount()	IAudioTracks.GetAudioTracks().Count	Tracks
IMediaControl.GetCurrentTimeMs()	IMediaControl.GetCurrentTime()	Time format changes
IMediaControl.GetBufferingProgress()		No replacement
IMediaControl.GetBufferedTimeRange()		Time Ranges
IMediaControl.GetBufferedTimeRangeCount()	IMediaControl.GetBufferedTimes().Count	Time Ranges
IMedaControl.SetAudioTrack()	IAudioTracks.SetActiveAudioTrack()	Tracks

Time

All time values have changed from 32-bit float milliseconds, to 64-bit double seconds.

Tracks

```
int trackCount = mediaPlayer.AudioTracks.GetAudioTracks().Count;
AudioTrack track = mediaPlayer.AudioTracks.GetAudioTracks()[index];
mediaPlayer.AudioTracks.SetActiveAudioTrack(track);
```

Media Hints

```
MediaHints hints = mediaPlayer.FallbackMediaHints;
hints.stereoPacking = StereoPacking.None;
mediaPlayer.FallbackMediaHints = hints;
```

Time Ranges

```
int bufferZones = IMediaControl.GetBufferedTimes().Count
TimeRange range = IMediaControl.GetBufferedTimes()[0];
```

PlayMaker

PlayMaker actions from AVPro Video 1.x are not included in 2.x

Other Changes

Without going into detail, other API changes include:

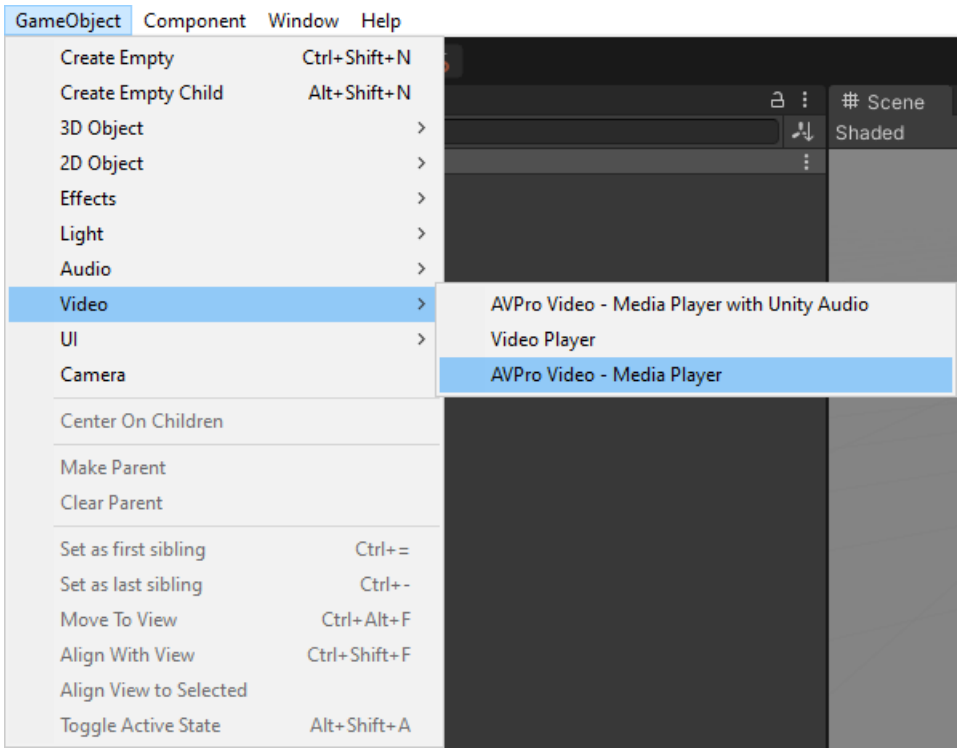
1. Custom HTTP headers
2. Setting audio output mode
3. macOS, iOS and tvOS have been unified, so most mentions of 'MacOSX' have been replaced with 'Apple'

Quick Start

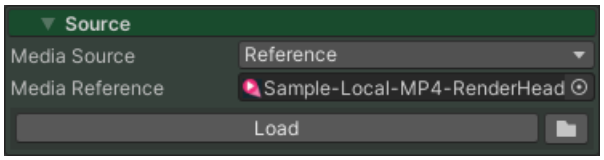
Playing Media

A MediaPlayer component is always required and so should be the first component you add to your scene.

1. Create a GameObject with the MediaPlayer component by selecting from the menu `GameObject > Video > AVPro Video - MediaPlayer`.



2. On the MediaPlayer component, set the media source via the `Settings > Source` section. Press on the folder button to browse for the media you want to play. See the Loading Media section for more information about this.



The MediaPlayer is set up to load and play your video, however it will not display yet (see below).

Displaying Video on the UI

The DisplayUGUI component is used to render the video to the UI.

1. Create your UI canvas by going to the menu `GameObject > UI > Canvas`.
2. Make sure the new Canvas GameObject is selected and then select from the menu `GameObject > UI > AVPro Video uGUI` to add the DisplayUGUI component.
3. Select the new GameObject and assign the `MediaPlayer` property to the MediaPlayer created above.
4. Play the scene to see your media displayed

Displaying Video on a Mesh

The ApplyToMesh component is used to render the video to a mesh.

1. Add a 3D mesh object to your scene (eg Quad) and move it so that it is visible by the main camera.

2. Add the ApplyToMesh components to a GameObject and assign the `Media` property to the MediaPlayer created above and the `Renderer` property to the MeshRenderer component.
3. Create a new material and assign it a suitable AVPro Video shader (eg AVProVideo/Unlit/Opaque).
4. In the MeshRenderer assign this new material to the Materials Element 0 property.
5. Play the scene to see your media displayed on a mesh.

Demo Scenes

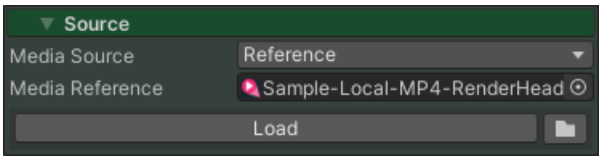
Read about the included demo scenes as these are also a good quick start reference for typical use-cases.

Loading Media

The location of media can be specified in two main ways:

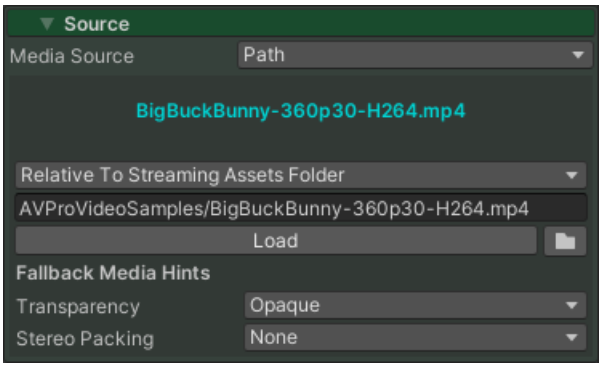
1. Media Reference
2. Path

Media Reference



The MediaReference asset allows media to be specified and stored within Unity, media hints to be set (eg transparency and stereo) and referenced easily. This is the preferred way to specify media if the media is permanent.

Path



Alternatively a file path / URL can be specified directly in the MediaPlayer. This is suitable for media that isn't permanent and therefore isn't worth creating a MediaReference for. With this method the media hints need to be specified in the MediaPlayer.

Scripting

```
// Opening media via a MediaReference
MediaReference mediaReference = _myMediaReference;
bool isOpening = mediaPlayer.OpenMedia(mediaReference, autoPlay:true);

// Opening media URL via a Path
bool isOpening = mediaPlayer.OpenMedia(new MediaPath("https://www.myvideos.com/stream.m3u8",
MediaPathType.AbsolutePathOrURL), autoPlay:true);

// Opening local file media via a Path
bool isOpening = mediaPlayer.OpenMedia(new MediaPath("myvideo.mp4",
MediaPathType.RelativeToStreamingAssetsFolder), autoPlay:true);

// Changing the media hints for content loaded via Path
MediaHints hints = mediaPlayer.FallbackMediaHints;
hints.stereoPacking = StereoPacking.TopBottom;
mediaPlayer.FallbackMediaHints = hints;
```

Android

On Android keeping very large video files in the `StreamingAssets` folder is not a good idea because this folder is converted into a JAR file and uses a lot of memory to open. In these cases it's best to store the videos in `Application.persistentDataPath` (

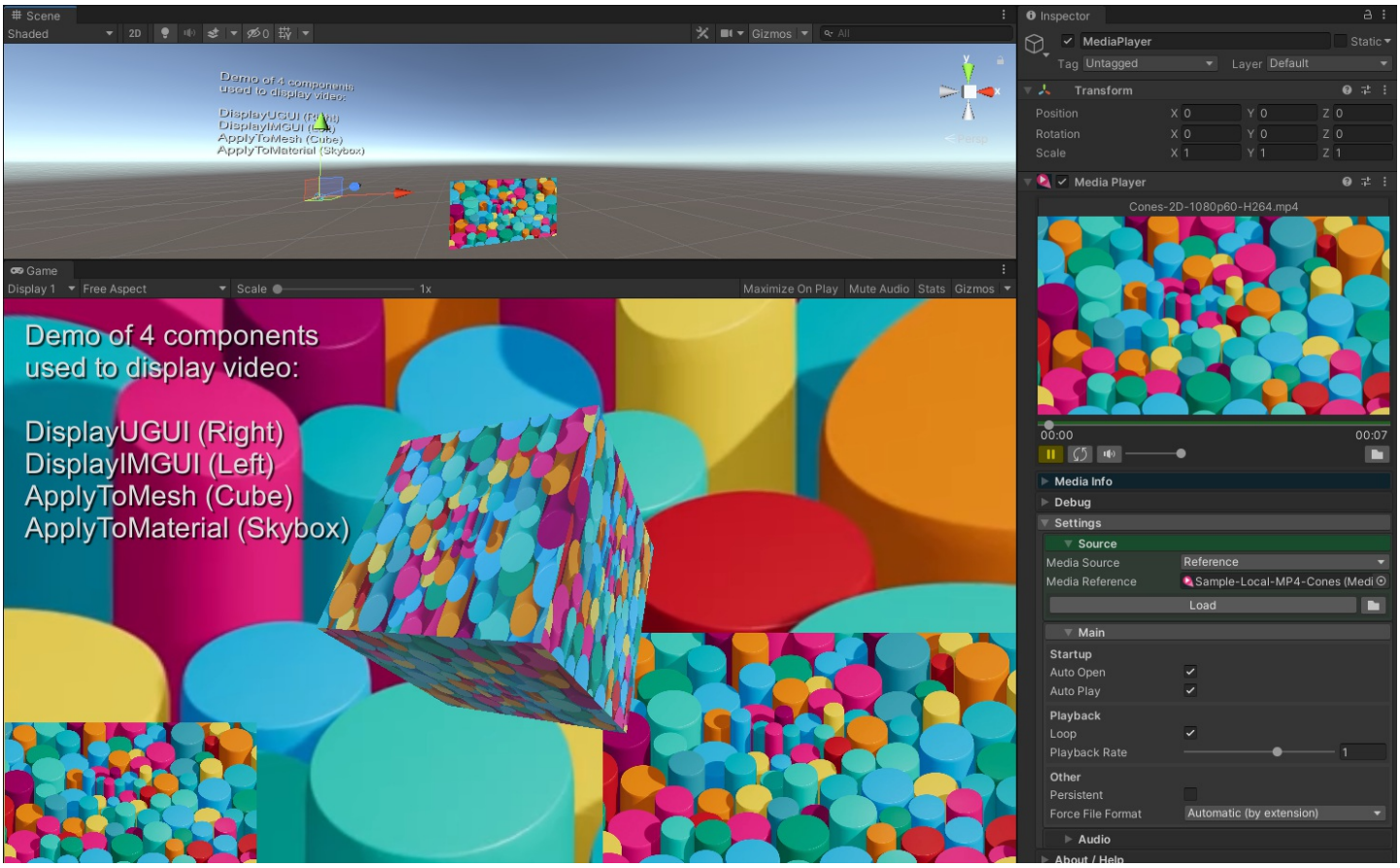
`MediaPathType.RelativeToPersistentDataFolder`), usually by downloading it to this folder. You may also need to set the permissions to access this folder which is in `Player Settings > Android > Write Permission > External (SDCard)`.

Demos

Media Player UI Demo

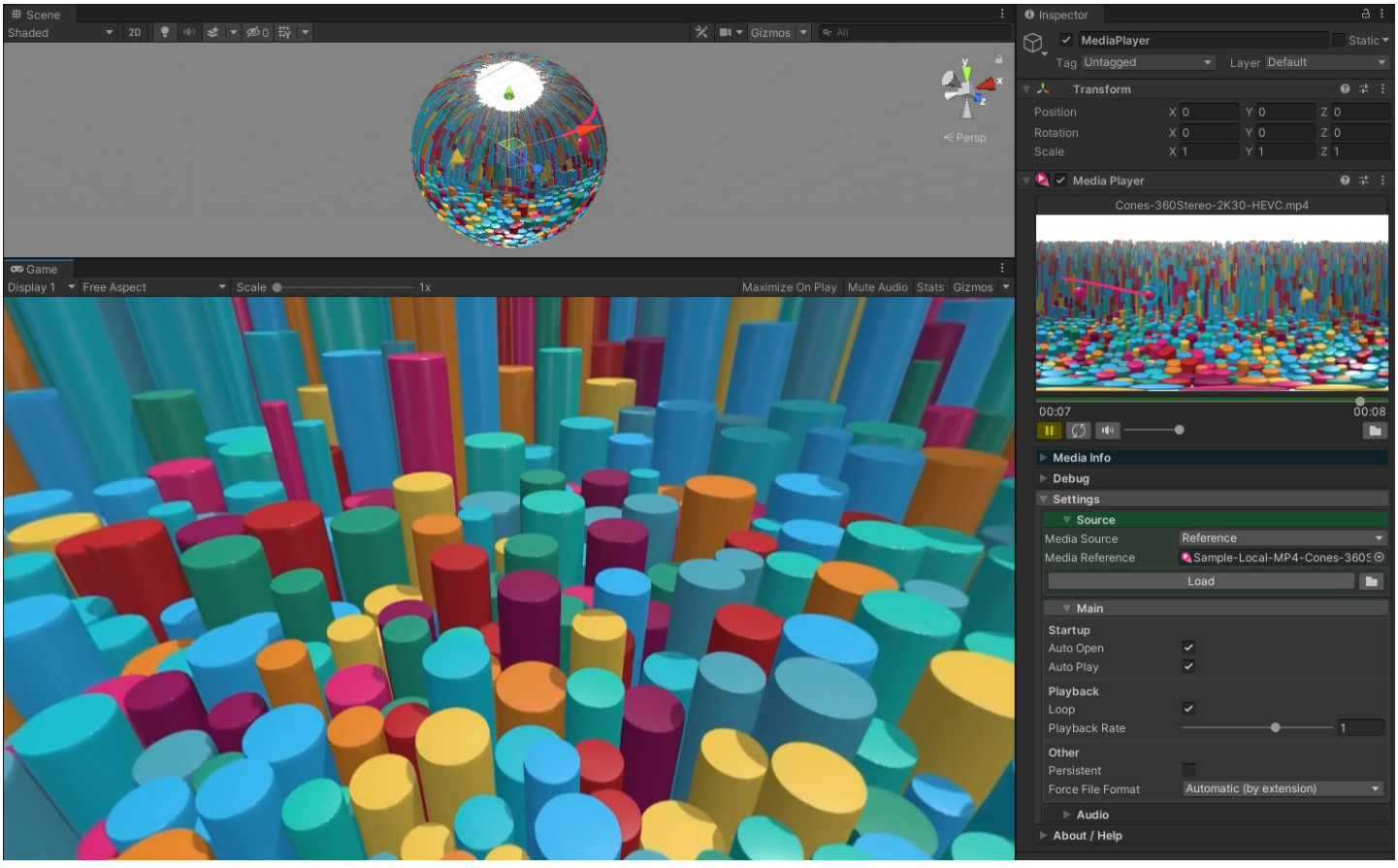
A comprehensive example scene playing back media with AVPro Video and using the MediaPlayerUI to help demonstrate features like subtitles, seeking, Unity audio, streaming etc.

Display Components Demo



An example scene showing all of the different components used to display and manipulate videos (eg ApplyToMesh, DisplayUGUI etc). It also shows how to apply the video to a `Skybox`.

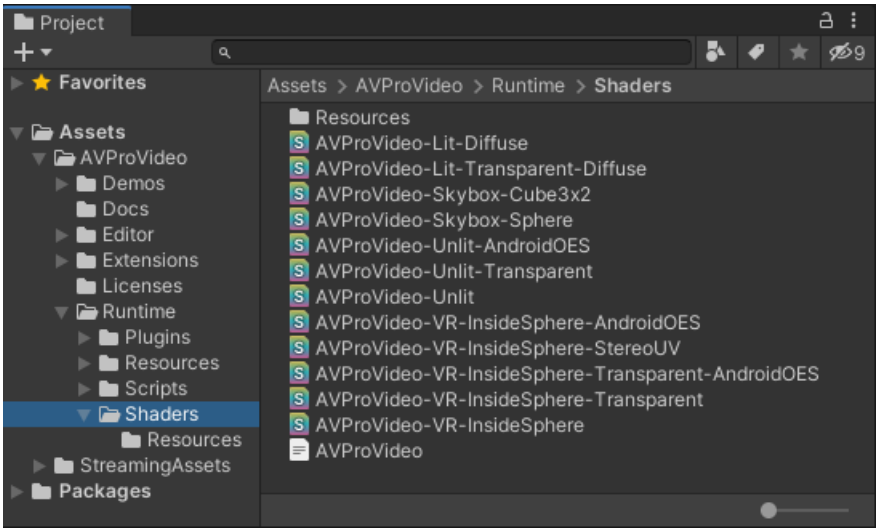
360 Stereo Demo



An example scene showing 360 equi-rectangular stereo video applied to a sphere. If no VR headset is available touch/mouse can

be used to rotate the camera view.

Shaders



AVPro Video includes a number of shaders, most of which are used internally, but in some cases the user is required to use the appropriate shader.

Canvas Display

When displaying video on the Canvas via the DisplayUGUI component you don't have to worry about the shader being used. Internally this component will select the best AVPro Video shader to use.

Mesh Display

When using the ApplyToMesh or ApplyToMaterial component, you should make sure that the material uses one of the AVPro Video shaders. This is because the textures produced by the plugin need transforming to be displayed correctly (eg gamma adjustments, stereo resolving, alpha packing, vertical flipping on some platforms etc). Our shaders handle all of this automatically and without an extra resolve step which would add overhead.

> **NOTE**
>
> In the near future we plan to add a "Resolve" feature which will resolve the textures to a RenderTexture so that our shaders won't be required. This will be very useful for many cases, but it will be a small performance penalty due to the extra resolve step.

Unlit Shaders

For cases where lighting isn't needed:

- Unlit
  - Supports: Stereo, colour tint, fog
  - Unsupported: Lighting, transparency
- Unlit-Transparent
  - Supports: Stereo, colour tint, fog, transparency
  - Unsupported: Lighting
- Unlit-AndroidOES
  - Android OES mode only (it will fall back to the Unlit shader on non-Android platforms)
  - Supports: Stereo, colour tint
  - Unsupported: Lighting, transparency
- Unlit-Transparent-AndroidOES

    - Android OES mode only (it will fall back to the Unlit transparent shader on non-Android platforms)
    - Supports: Stereo, colour tint
    - Unsupported: Lighting

Unlit VR Shaders

Very similar to the previous Unlit shaders, but with some extra functionality often used for 360 / 180 VR videos, and culling reversed to make the mesh (usually a sphere) visible from inside:

- VR-InsideSphere
    - Supports: Stereo, fog, Equi-rectangular 360 and 180 video layout
    - Unsupported: Lighting, transparency
- VR-InsideSphere-Transparent
    - Supports: Stereo, fog, Equi-rectangular 360 and 180 video layout, transparency
    - Unsupported: Lighting
- VR-InsideSphere-StereoUV
    - This is a special case, it's the same as VR-InsideSphere but uses a different texture coordinate set per eye. This is useful fo custom layouts specified via UV coordinates
    - Supports: Stereo, fog, custom video layouts
    - Unsupported: Lighting, transparency
- VR-InsideSphere-AndroidOES
    - Android OES mode only (it will fall back to the VR-InsideSphere shader on non-Android platforms)
    - Supports: Stereo, fog, Equi-rectangular 360 and 180 video layout
    - Unsupported: Lighting, transparency
- VR-InsideSphere-Transparent-AndroidOES
    - Android OES mode only (it will fall back to the VR-InsideSphere shader on non-Android platforms)
    - Supports: Stereo, fog, Equi-rectangular 360 and 180 video layout, transparency
    - Unsupported: Lighting

Skybox Shaders

For cases where the video is to be applied to a Skybox:

- Skybox-Sphere
    - Supports: Stereo, Equi-rectangular 360 video layout
- Skybox-Cube3x2
    - Supports: Stereo, Cubemap 3x2 video layout

Lit Shaders

For cases where the video needs to be lit by the scene:

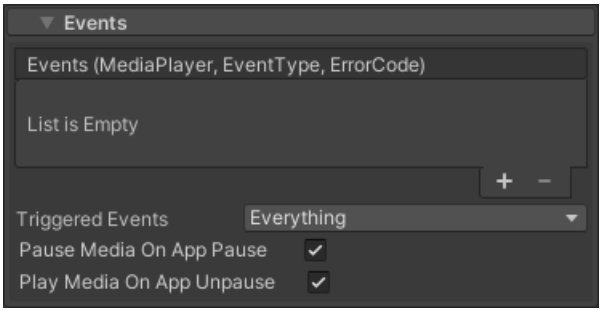
- Lit-Diffuse
    - Supports: Lighting, stereo, Colour tint
    - Unsupported: Transparency, fog
- Lit-Transparent-Diffuse
    - Supports: Lighting, transparency
    - Unsupported: Stereo, fog

> **NOTE**
>
> Materials using these lit shaders will not automatically update when upgrading to HDRP / UWP rendering pipelines.

Events

The MediaPlayer component generates various events which can be subscribed to.

Events Section



The Events section of the `MediaPlayer` component can be usd to assign a method for handling events, however it is usually better to do this via scripting (see below).

Events can also be filtered via the dropdown `Triggered Events` list to prevent certain events from triggering which can save some scripting performance.

Scripting

Here is an example of how to add a listener for events and handle those events:

```
using UnityEngine;
using RenderHeads.Media.AVProVideo;

[RequireComponent(typeof(MediaPlayer))]
public class EventsExample : MonoBehaviour
{
    void Awake()
    {
        // The method HandleEvent will be called whenever an event is triggered
        GetComponent<MediaPlayer>().Events.AddListener(HandleEvent);
    }

    // This method is called whenever there is an event from the MediaPlayer
    void HandleEvent(MediaPlayer mp, MediaPlayerEvent.EventType eventType, ErrorCode code)
    {
        Debug.Log("MediaPlayer " + mp.name + " generated event: " + eventType.ToString());
        if (eventType == MediaPlayerEvent.EventType.Error)
        {
            Debug.LogError("Error: " + code);
        }
    }
}
```

Events

EVENT	DESCRIPTION
MetaDataReady	Triggered when certain data (width, duration etc) is available
ReadyToPlay	Triggered when the video is loaded and ready to play (not triggered when auto-play is enabled)

EVENT	DESCRIPTION
Started	Triggered when the playback starts
FirstFrameReady	Triggered when the first frame has been rendered (not available on some playforms e.g. WebGL)
FinishedPlaying	Triggered when a non-looping video has finished playing
Closing	Triggered when the media is closed
Error	Triggered when an error occurs
SubtitleChange	Triggered when the subtitles change
Stalled	Triggered when media is stalled (e.g. when lost connection to media stream)
Unstalled	Triggered when media is resumed form a stalled state (e.g. when lost connection is re-established)
ResolutionChanged	Triggered when the resolution of the video has changed (including the load). Useful for adaptive streams
StartedSeeking	Triggered when seeking begins
FinishedSeeking	Triggered when seeking has finished
StartedBuffering	Triggered when buffering begins
FinishedBuffering	Triggered when buffering has finished
PropertiesChanged	Triggered when any properties (e.g. stereo packing are changed) - this has to be triggered manually
PlaylistItemChanged	Triggered when the new item is played in the playlist
PlaylistFinished	Triggered when the playlist reaches the end
TextTracksChanged	Triggered when the text tracks are added or removed
TextCueChanged	Triggered when the text to display changes

Streaming

AVPro Video supports several streaming protocol depending on the platform:

	WINDOWS	UWP	ANDROID	MACOS	IOS / IPADOS / TVOS
HTTP Progressive					
MP4	□	□	□	□	□
Adaptive					
HLS (m3u8)	□ [1]	□ [1]	□	□	□
MPEG-DASH (mpd)	□ [1]	□ [1]	□ [4]	.	.
Microsoft Smooth Streaming (ism)	□ [1]	□ [1]	.	.	.
Real-time					
RTSP	~ [2]	.	□ [5]	.	.
RTMP	~ [3]	.	□ [6]	.	.

[1] Requires Windows 10 for native support, or using DirectShow with suitable 3rd party filter (eg LAV Filters).

[2] Limited native support. Read Microsoft notes about support here: https://docs.microsoft.com/en-us/windows/win32/medfound/supported-protocols. Generally only support ASF, MP3 and PCM media types, but support seems improved from Windows 10 build 1803 onwards (as in added H.264 support), but it's not documented (parsing is handled by mfnetsrc.dll).

[3] Only using DirectShow with suitable 3rd party filter (eg LAV Filters).

[4] Using ExoPlayer API only.

[5] Using ExoPlayer API, or MediaPlayer API (but not fully featured).

[6] Using ExoPlayer API only. Known issues surrounding address resolution.

HTTP Progressive Streaming

This form of streaming is probably the most widely supported. It is very similar to playing a local MP4 file, except that it is streamed from a network source. The HTTP server should support features such as byte range requests.

When encoding MP4 videos for streaming make sure they are encoded with the video header data at the beginning of the file. You normally do this by selecting "Fast Start" in QuickTime encoder, or use the "-movflags faststart" option in FFMPEG, Other encoders will have a similar option. To prepare an MP4 for streaming using FFMPEG you can use the following command:

```
ffmpeg -i %1 -acodec copy -vcodec copy -movflags faststart %1-streaming.mp4
```

Adaptive Streaming

Adaptive streaming such as HLS and MPEG-DASH are flexibly formats that support adaptive bit-rate selection and multiple audio, video and subtitle tracks. HLS is by far the most widely supported.

On certain platforms (Android and Windows 10) we allow setting a hint so that streaming will begin with the highest bit-rate. On Apple platforms the player adheres to Apple's standard of starting with the first stream in the manifest file.

AES-128 encrypted HLS streams, and custom HTTP headers are supported in the Ultra Edition.

Real-time Streaming

Formats like RTSP/RTMP are designed for real-time streaming and are popular with live streaming cameras. As shown in the above table AVPro Video doesn't have strong support for these formats as they are not the focus of this plugin and most operating systems do not have good native support for them.

Live Streaming

Live HLS and MPEG-DASH streams are supported and will return a duration of +infinity.

```
// Detect a live stream
double duration = mediaPlayer.Info.GetDuration();
bool isLive = double.IsInfinity(duration);
```

Some live streams contain a seekable range so that the stream can be viewed from an offset from the live time. The seekable range should be queried to determine this.

```
// Get the seekable time range
TimeRanges seekable = mediaPlayer.Control.GetSeekableTimes();
Debug.Log("Seekable time: " + seekable.MinTime + " " + seekable.MaxTime);
```

Some HLS streams contain a program-time via the EXT-X-PROGRAM-DATE-TIME tag. This is supported on macOS, iOS, tvOS, Android (using ExoPlayer) and Windows 10 (using WinRT API). This can be queried in script:

```
// Get the stream date and time
System.DateTime time = mediaPlayer.Control.GetProgramDateTime();
if (time != DateTime.MinValue)
{
    Debug.Log("Valid time: " + time.toString());
}
```

Windows

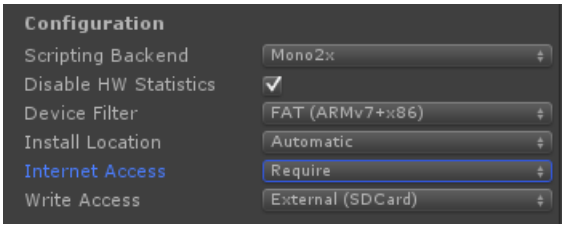
For best results with adaptive streams, select the WinRT video API (requires Windows 10) instead of the default Media Foundation. This will give better streaming performance and compatibility, especially for live streams. This will also allow you to select the option to begin streaming at the highest bit-rate available via the `Platform Specific > Windows` options.

```
// Start adaptive stream using the highest resolution - WinRT only
mediaPlayer.PlatformOptionsWindows.videoApi = Windows.VideoApi.WinRT;
mediaPlayer.PlatformOptionsWindows.startWithHighestBitrate = true;

// Set live stream to use the lowest latency possible for live streams - WinRT only
mediaPlayer.PlatformOptionsWindows.videoApi = Windows.VideoApi.WinRT;
mediaPlayer.PlatformOptionsWindows.useLowLiveLatency = true;
```

Android

Using the ExoPlayer API is recommended for streaming video as it generally has wider support for streaming protocols.

Android streaming requires the Internet Access setting (in Player Settings) to be set to `"Require"`:



ExoPlayer will also allow you to select the option to begin streaming at the highest bit-rate available via the `Platform Specific > Android` options.

```
// Start adaptive stream using the highest resolution - ExoPlayer only
mediaPlayer.PlatformOptionsAndroid.videoApi = Android.VideoApi.ExoPlayer;
mediaPlayer.PlatformOptionsAndroid.startWithHighestBitrate = true;

// Set the maximum adaptive resolution to 1080p - ExoPlayer only
mediaPlayer.PlatformOptionsAndroid.videoApi = Android.VideoApi.ExoPlayer;
mediaPlayer.PlatformOptionsAndroid.preferredMaximumResolution = OptionsAndroid.Resolution._1080p;

// Set the peak adaptive bitrate to 4Mbps - ExoPlayer only
mediaPlayer.PlatformOptionsAndroid.videoApi = Android.VideoApi.ExoPlayer;
mediaPlayer.PlatformOptionsAndroid.preferredPeakBitRate = 4.0f;
mediaPlayer.PlatformOptionsAndroid.preferredPeakBitRateUnits = OptionsAndroid.BitRateUnits.Mbps;
```

ExoPlayer exposes buffering values via the `Platform Specific > Android` options.

```
// Adjust buffering - ExoPlayer only
mediaPlayer.PlatformOptionsAndroid.videoApi = Android.VideoApi.ExoPlayer;
// Set a buffer size of 50 seconds
mediaPlayer.PlatformOptionsAndroid.minBufferMs = 50000;
mediaPlayer.PlatformOptionsAndroid.maxBufferMs = 50000;
// Wait for 2.5 seconds of media to become buffered before playback begins or resumes after a seek
mediaPlayer.PlatformOptionsAndroid.bufferForPlaybackMs = 2500;
// Wait for 5.0 seconds of media to become buffered before playback starts after the buffer runs out due to
bandwidth/connection issues
mediaPlayer.PlatformOptionsAndroid.bufferForPlaybackAfterRebufferMs = 5000;
```

NOTE

Starting with Android 9 (API level 28) cleartext support (unencrypted HTTP connections) is disabled by default, which can cause some HTTP streams to fail. You should be able to resolve this by switching the URL to HTTPS or by adding `android:usesCleartextTraffic="true"` into your AndroidManifest.xml file.

macOS / iOS / tvOS

This platform supports streaming of HLS streams which typically end with the m3u or m3u8 extension.

NOTE

HTTP URLs are no longer supported by default on these platforms, and a secure HTTPS URL should be used.

If you can only use HTTP then your app has to have a special flag set to let it use HTTP connections (this is a security issue for Apple). This setting is exposed in the Unity Player Settings here for iOS and tvOS:

The setting is also exposed in the Unity scripting API here: http://docs.unity3d.com/ScriptReference/PlayerSettings.iOS-allowHTTPDownload.html

If for some reason your version of Unity doesn't expose this then you will have to add it manually. In the Unity editor you need to edit "Unity.app/Contents/Info.plist" and in your built application you would need to edit "your.app/Contents/Info.plist". These files need to have these keys added:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

HLS options are exposed via the Platform Specific options.

```
// Set maximum HLS resolution to 1080p and peak bitrate to 4Mbps - macOS
mediaPlayer.PlatformOptionsMacOSX.preferredMaximumResolution = OptionsApple.Resolution._1080p;
mediaPlayer.PlatformOptionsMacOSX.preferredPeakBitRate = 4.0f;
mediaPlayer.PlatformOptionsMacOSX.preferredPeakBitRateUnits = OptionsApple.BitRateUnits.Mbps;

// Set maximum HLS resolution to 1080p and peak bitrate to 4Mbps - iOS
mediaPlayer.PlatformOptionsIOS.preferredMaximumResolution = OptionsApple.Resolution._1080p;
mediaPlayer.PlatformOptionsIOS.preferredPeakBitRate = 4.0f;
mediaPlayer.PlatformOptionsIOS.preferredPeakBitRateUnits = OptionsApple.BitRateUnits.Mbps;

// Set maximum HLS resolution to 1080p and peak bitrate to 4Mbps - tvOS
mediaPlayer.PlatformOptionsTVOS.preferredMaximumResolution = OptionsApple.Resolution._1080p;
mediaPlayer.PlatformOptionsTVOS.preferredPeakBitRate = 4.0f;
mediaPlayer.PlatformOptionsTVOS.preferredPeakBitRateUnits = OptionsApple.BitRateUnits.Mbps;
```

UWP / Hololens

Make sure to tick the `"InternetClient"` capabilities option in Player Settings.

If you're streaming video from a local server / LAN then you need to enable the `"PrivateNetworkClientServer"` option.

For best results with adaptive streams, select the WinRT video API (requires Windows 10) instead of the default Media Foundation. This will give better streaming performance and compatibility, especially for live streams. This will also allow you to select the option to begin streaming at the highest bit-rate available via the `Platform Specific > Windows` options

```
// Start adaptive stream using the highest resolution - WinRT only
mediaPlayer.PlatformOptionsWindowsUWP.videoApi = WindowsUWP.VideoApi.WinRT;
mediaPlayer.PlatformOptionsWindowsUWP.startWithHighestBitrate = true;

// Set live stream to use the lowest latency possible for live streams - WinRT only
mediaPlayer.PlatformOptionsWindowsUWP.videoApi = WindowsUWP.VideoApi.WinRT;
mediaPlayer.PlatformOptionsWindowsUWP.useLowLiveLatency = true;
```

WebGL

> **IMPORTANT**
>
> **We do not officially support WebGL**, but only include it as it may be useful for some people. We found too many issues with browser

compatibility to continue supporting it

If you are trying to access a URL on another server/port/domain then you need to have CORS (cross-origin resource sharing) configured on that server to allow access. Websites like https://enable-cors.org/ show you how to configure CORS on different web servers. If you are hosting on a S3 bucket there are also ways to configure this. You can also test whether CORS is the issue by installing a browser plugin to toggle CORS, for example this one for Chrome: https://chrome.google.com/webstore/detail/allow-cors-access-control/lhobafahddgcelffkeicbaginigeejlf?hl=en

Better HLS/MPEG-DASH Support

HLS and MPEG-DASH are not natively supported by all browsers. We have added hooks to include third-party javascript libraries to handle these formats. Under the `"Platform Specific > WebGL"` section you can select `"External Library"`. This will force the MediaPlayer to use either hls.js or dash.js. You can also select "custom" if you wish to add support for your own javascript library.

```
// Set the external library to hls.js
mediaPlayer.PlatformOptionsWebGL.externalLibrary = WebGL.ExternalLibrary.HlsJs;
```

To add support or dash.js:

1. Download the latest dash.js release (we last tested with 2.8.0): https://github.com/Dash-Industry-Forum/dash.js/releases
2. Copy "dash.all.min.js" to the Assets/Plugins/WebGL folder and rename it "dash.all.min.jspre" (don't rename it inside the Unity editor as it will not get the correct extension, instead rename it from Explorer or Finder)
3. In the MediaPlayer component set Platform Specific > WebGL > External Library to dash.js
4. Build for WebGL

To add support for hls.js:

1. In the MediaPlayer component set Platform Specific > WebGL > External Library to hls.js
2. Build for WebGL
3. Download the latest hls.js release (we last tested with v1.0.7):

   https://github.com/video-dev/hls.js/releases
4. Once your build is made, copy "hls.min.js" to the TemplateData folder
5. Edit the index.html to add <script src="TemplateData/hls.min.js"></script> in a <head>; section of the HTML, before the UnityLoader.js script is loaded. Ideally you would add this to a new WebGL template so that you don't have to make these changes for each build.
6. If you want to pass in custom config options (see hls.js reference), then you will need to edit AVProVideo.jslib and change the HLS constructor to pass in your config options, eg:

YouTube / Facebook Live / Twitch Support

We get asked a lot about YouTube/Twitch etc support, so we are including this note here. AVPro Video doesn't officially support streaming from such platforms. This is because it is against their terms & conditions, and the streaming URLs are protected by javascript and server obfuscation. We have heard though that some people are using 3rd party tools to extract the streaming URLs. In theory these URLs could be playable in AVPro Video but this is not something we support.

Vimeo Support

If you are streaming videos from VIMEO as MP4 then you should note that you can replace the ".mp4" part in the URL with ".m3u8" to instead make it an HLS stream. This may be particularly useful if you are developing apps for the Apple's App Store as you would need to use HLS streaming to pass certification (as for April 2016).

There is also an official Unity plugin for Vimeo (by Vimeo) that integrates with the old version of AVPro Video. Unfortunately Vimeo have not maintained their plugin so we can no longer recommend it.

Dropbox / Google Drive / One Drive Support

These services aren't designed for streaming video files, so files hosted here will not work with AVPro Video.

AWS S3 Support

AVPro Video supports streaming video files from AWS S3.

Test Streams

We found these 3rd-party streams handy for testing (but no guarantee that they're still working):

- "Tears of Steel" VOD
  https://stream.mux.com/4XYzhPXzqArkFI8d1vDsScBLD69Gh1b2.m3u8
  HTTPS - HLS - H.264 - AAC - WebVTT Subtitles - 1080p
- "Tears of Steel" LIVE
  https://cph-p2p-msl.akamaized.net/hls/live/2000341/test/master.m3u8
  HTTPS - HLS - H.264 - AAC - 1080p
- "Apple Bip Bop" VOD
  https://devstreaming-cdn.apple.com/videos/streaming/examples/bipbop_16x9/bipbop_16x9_variant.m3u8
  HTTPS - HLS - H.264 - AAC - WebVTT Subtitles - 1080p
- "Skate Phantom" VOD
  http://sample.vodobox.net/skate_phantom_flex_4k/skate_phantom_flex_4k.m3u8
  HTTP - HLS - H.264 - AAC - 4K
- "Llama Drama" VOD
  http://amssamples.streaming.mediaservices.windows.net/634cd01c-6822-4630-8444-8dd6279f94c6/CaminandesLlamaDrama4K.ism/manifest(format=m3u8-aapl)
  HTTP - HLS - H.264 - AAC - 4K

Augmented / Mixed / Virtual Reality

AVPro Video has a number of features useful for XR experiences.

360 and 180 Formats

Three popular spatial mapping formats are supported:

- Equirectangular 360
- Equirectanuglar 180
- Cubemap 3:2 (also known as Facebook Cubemap)

These layouts can be set as a hint on the MediaPlayer via the Inspector, or via scripting:

```
// Set the video layout mapping hint on the MediaPlayer
mediaPlayer.VideoLayoutMapping = VideoMapping.EquiRectangular180;
```

Displaying

Equirectangular videos can be played on a sphere, or assigned to a Skybox material.

The `ApplyToMesh` component can be used to render a video to a sphere with the user camera at the center of the sphere. See the 360 Sphere Demo for an example of this.

Or a `Skybox` component with `ApplyToMaterial` component can be used to render a video to the skybox. See the Display Components Demo for an example of this.

High Resolution Video

360 / 180 VR videos typically require high resolutions. See the High Resolution Video section.

3D Stereo Video

See the 3D Stereo Video section.

180 Video

180 video works similarly to 360 video, but with some differences:

- Make sure to set the spatial mapping format to `Equirectangular 180` either in the `Visual` section of the `MediaPlayer` component in the Inspector, or via scripting.
- If the 180 video displayed (eg using `ApplyToMesh` component) on a sphere then it will be mirrored front and back. Instead a material with the `AVProVideo/VR/InsideSphere Unlit Transparent(stereo+color+fog+alpha)` shader can be applied to the sphere and this will only render the video to the front half of the sphere, and feather the edges to transparency. The amount of feathering can be controlled in the material using the `Edge Feather` property.

Spatial Audio

See the 360 Audio section.

VR / AR / MR / XR Headsets Supported

Android

- Oculus Rift Go
- Oculus Gear VR
- Oculus Quest
- Oculus Quest 2
- HTC Vive Focus
- HTC Vive Focus Plus
- HTC Vive Focus 3
- Gear VR
- Google Cardboard
- Google Daydream
- Pico Goblin
- Pico G2 4K
- Pico Neo 2
- Pico Neo 3
- Lenovo Mirage Solo

Windows Desktop

- HTC Vive
- HTC Vive Pro
- HTC Vive Cosmos
- Valve Index
- Oculus Rift
- Oculus Rift S
- StarVR

Windows UWP

- Microsoft Hololens
- Microsoft Hololens 2
- Samsung Odyssey
- Asus
- HP
- Acer
- Lenovo Explorer
- Dell Visor

High Resolution Video

The video resolution that can be played is only limited by the video decoder capabilities of the hardware. Most modern systems can decode at least 4K H.264 or HEVC(H.265) in hardware. Some systems (especially mobile / untethered) have some custom encoding requirements for higher resolutions and the developer documentation should be researched.

NOTE

In general support for H.264 above 4K resolution is not universally common. For resolutions above 4K the HEVC (H.265) codec should be used unless targeting a specific platform with other capabilities.

WARNING

Hardware has a limit to how much video they can decode simultaneously. It may be able to load 4 1080p videos and decode them, but only 1 4K video. This is especially true on mobile / portable VR platforms. If you use up all the capabilities of the hardware video decoder, then a second video will often not be loadable until you have unloaded the first video.

Windows

On Windows 8K video decoding usually requires a high-end GPUs (NVidia Geforce 10xx series and above, or newer Intel integrated GPUs) with 64-bit builds, using the HEVC codec.

For very high resolutions (eg 16K and beyond), a flexible codec like Hap or NotchLC can be used.

Android

See the performance notes for the Android Platform about using OES mode for best high-resolution video performance.

Oculus Quest

The Quest has special encoding requirements which are detailed in this Blog Post.

macOS / iOS

See the performance notes for the iOS Platform about using YcbCr mode for best high-resolution video performance.

3D Stereo Video

Packed side-by-side or top-bottom stereo video is supported on all platforms.

> **NOTE**
>
> For best results, pad the video size to a multiple to 16 pixels. This will guarantee accurate left-right extraction on all platforms.

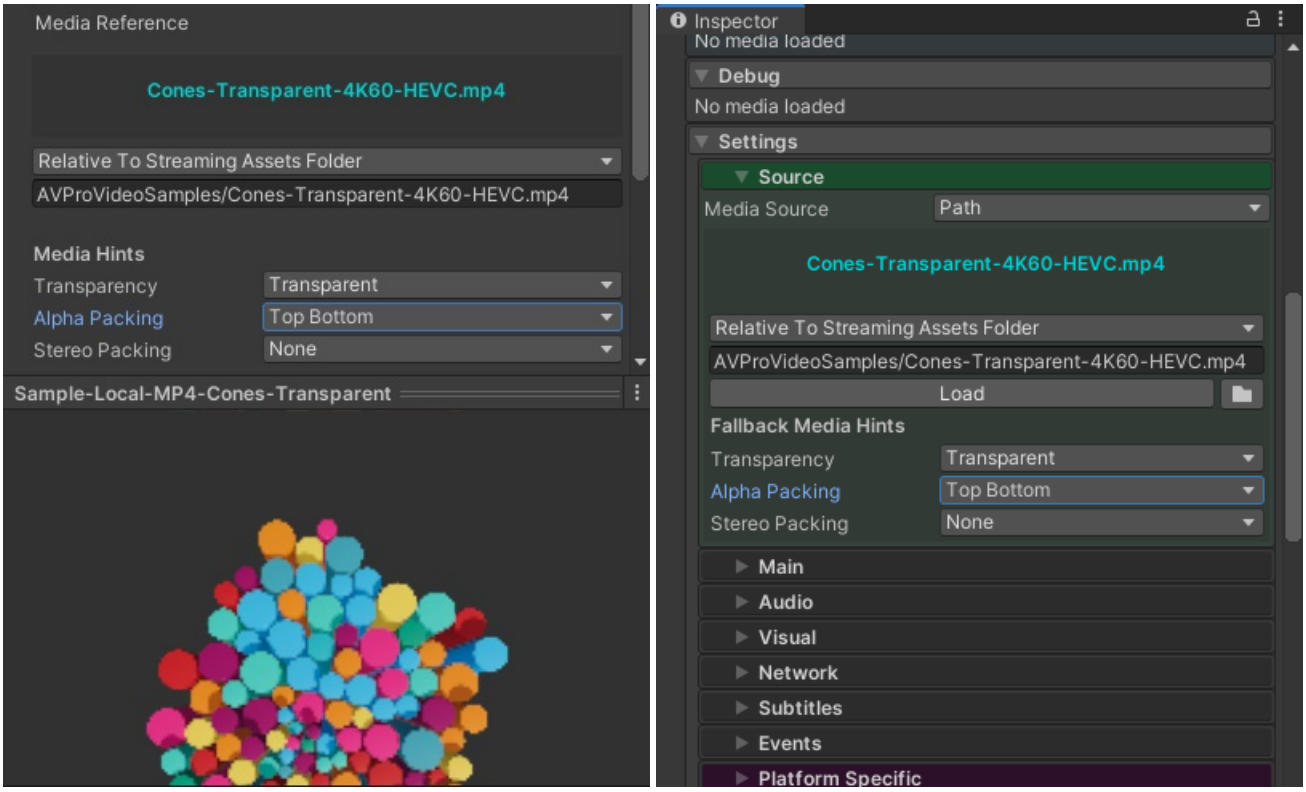
Auto-Detection

The stereo packing format of the video will be detected automatically by AVPro Video if the correct metadata is embedded during encoding. There are two standards for specifying stereo:

- Currently the st3d box / atom is supported on all platforms (except DirectShow API in Windows, and MediaPlayer API on Android). This is the best method to use.
- Stereo videos without the above atom may also have stereo data embedded as "SEI FPA (frame packing arrangement)", however this automatic detection of this is currently only supported on Windows.

Manually Specifying

If the auto-detection is not able to determine the stereo packing format (perhaps because the information is not encoded into the video file), then the packing format can be explicitly set via the MediaReference asset, or as a fallback MediaHint in the MediaPlayer when not loading from a MediaReference.



Stereo packing hints can also be set via scripting:

```
// Setting the fallback stereo packing on the MediaPlayer when loading videos via the Path MediaSource
MediaHints hints = mediaPlayer.FallbackMediaHints;
hints.stereoPacking = StereoPacking.TopBottom;
mediaPlayer.FallbackMediaHints = hints;
```

Getting Stereo Packing Mode

The stereo packing mode that is either automatically detected, or manually specified can be retrieved via scripting:

```
// Get the stereo packing mode that is used
StereoPacking videoStereoPacking = mediaPlayer.TextureProducer.GetTextureStereoPacking();
```

Forcing Stereo Eye Mode

It's possible to force the eye mode as well via scripting:

```
// Use the static VideoRender class to force a stereo eye mode on a material (eg Material used in
ApplyToMesh/ApplyToMaterial)
VideoRender.SetupStereoEyeModeMaterial(material, StereoEye.Left);
```

NOTE

Be sure to include the UpdateMultiPassStereo component if your stereo application will run in multi-pass mode.

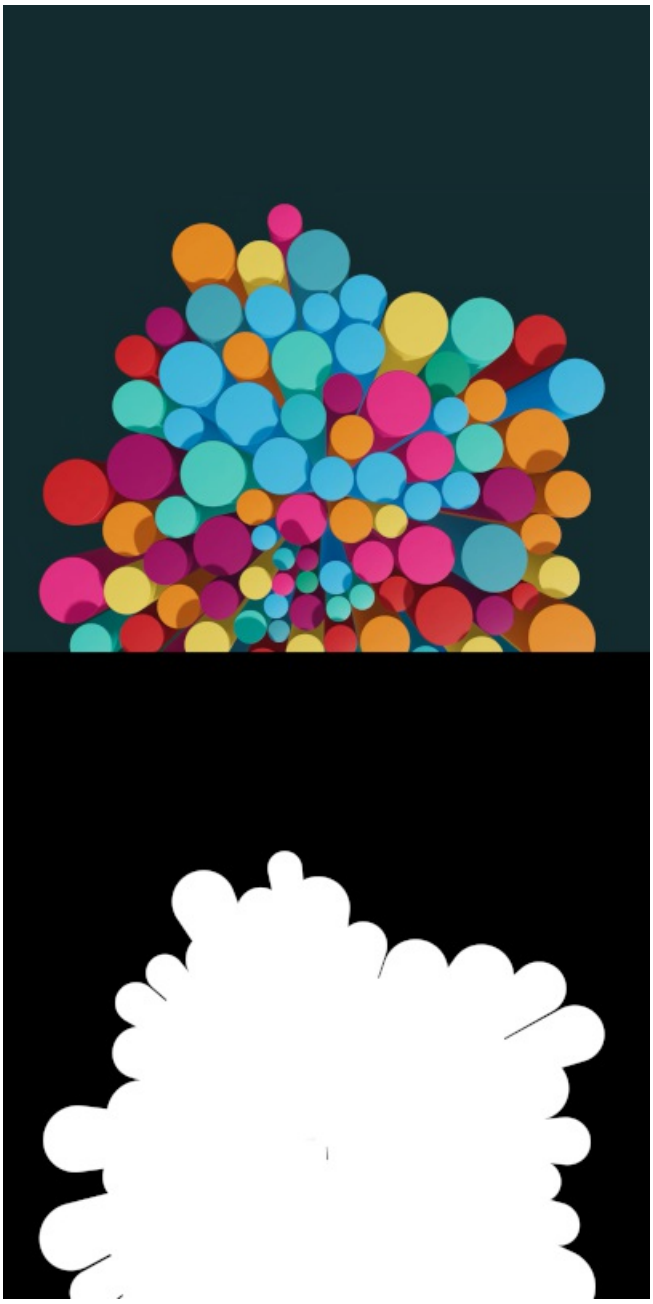
Transparency

Codecs

Not many video codecs have native support for transparency / alpha channels. Formats supported by some platforms of AVPro Video are:

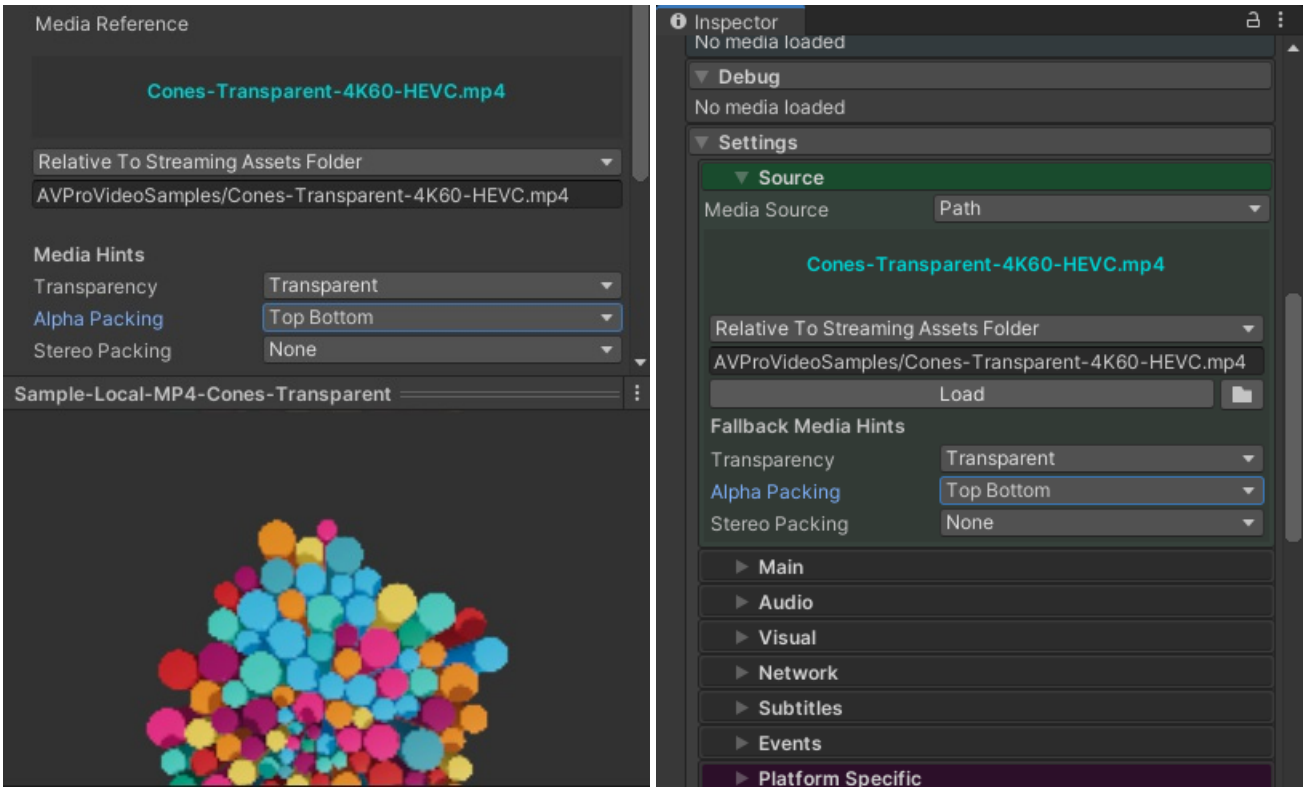
- HEVC+Alpha
  - Requires macOS 10.15, iOS 13.0 or tvOS 13.0
- Hap
  - Only supported in Ultra edition
  - Hap Alpha
    - Great support on Windows and macOS. Fast and low overhead format, though file size can get large depending on the content. Currently this is the format we recommend for transparent video.
  - Hap Q Alpha
    - Great support on Windows and macOS. Slightly higher quality and file size compared to Hap Alpha.
- NotchLC
  - Only supported in Ultra edition
- Uncompressed RGBA / YUVA
  - Uncompressed isn't ideal for file size or disk bandwidth but can still be used as a fallback
- ProRes 4444
  - Best support is on macOS. Files can be huge though.
- VP6
  - Legacy format. We support it only via 3rd party DirectShow plugins for Windows (eg LAV Filters)

Alpha Packing



Perhaps the best option is to encode your videos in video formats that don't support an alpha channel (eg MP4 as H.264 or HEVC) by packing the alpha channel into the same frame. You can double the width for a left-right packing layout, or double the height for a top-bottom packing layout. This packing could be created in software such as AfterEffects, or the command-line FFMPEG tool can be used.

The packing format is then specified either in the MediaReference or MediaPlayer as part of the MediaHints:



Authoring Videos with Alpha Packing

Exporting

> **NOTE**
>
> Export using the "straight" and not "premultiplied" alpha mode. AVPro Video shaders only support straight alpha.

> **NOTE**
>
> For best results, pad the video size to a multiple to 16 pixels. This will guarantee accurate alpha channel extraction on all platforms.

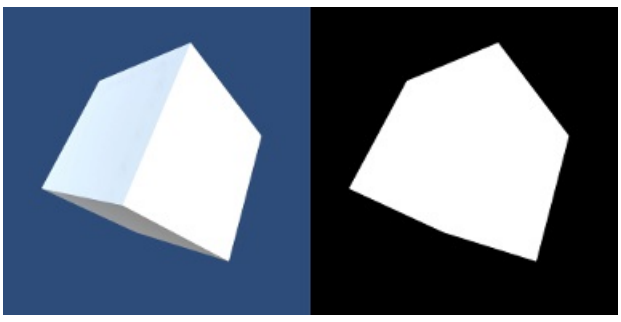
Usually you export your transparent video from your favourite software (eg After Effects). A codec that supports transparency such as ProRes4444 can be used. Make sure to use "straight" alpha mode. Then another tool such as FFMPEG can be used to convert that video into the packed alpha layout.

Alpha Packing with FFMPEG

FFMPEG command-line can be used to convert a source video containing a transparency/alpha channel into an alpha packed format:

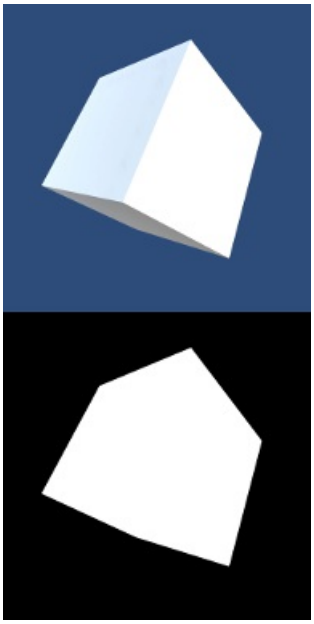
Left-right alpha packing:

```
ffmpeg -i input.mov -vf "split [a], pad=iw*2:ih [b], [a] alphaextract, [b] overlay=w" -y output-lr.mp4
```

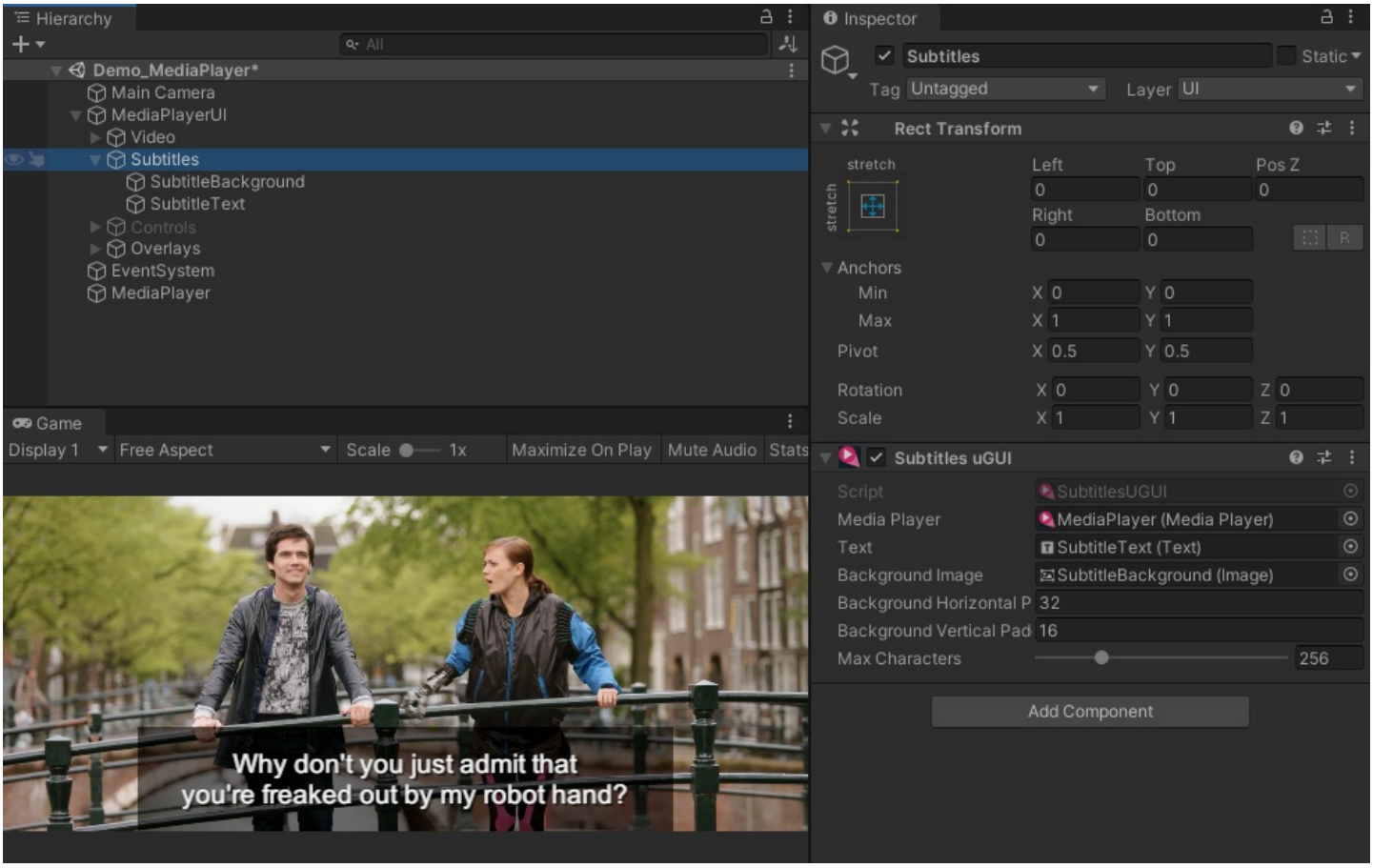


Top-bottom alpha packing:

```
ffmpeg -i input.mov -vf "split [a], pad=iw:ih*2 [b], [a] alphaextract, [b] overlay=0:h" -y output-tb.mp4
```



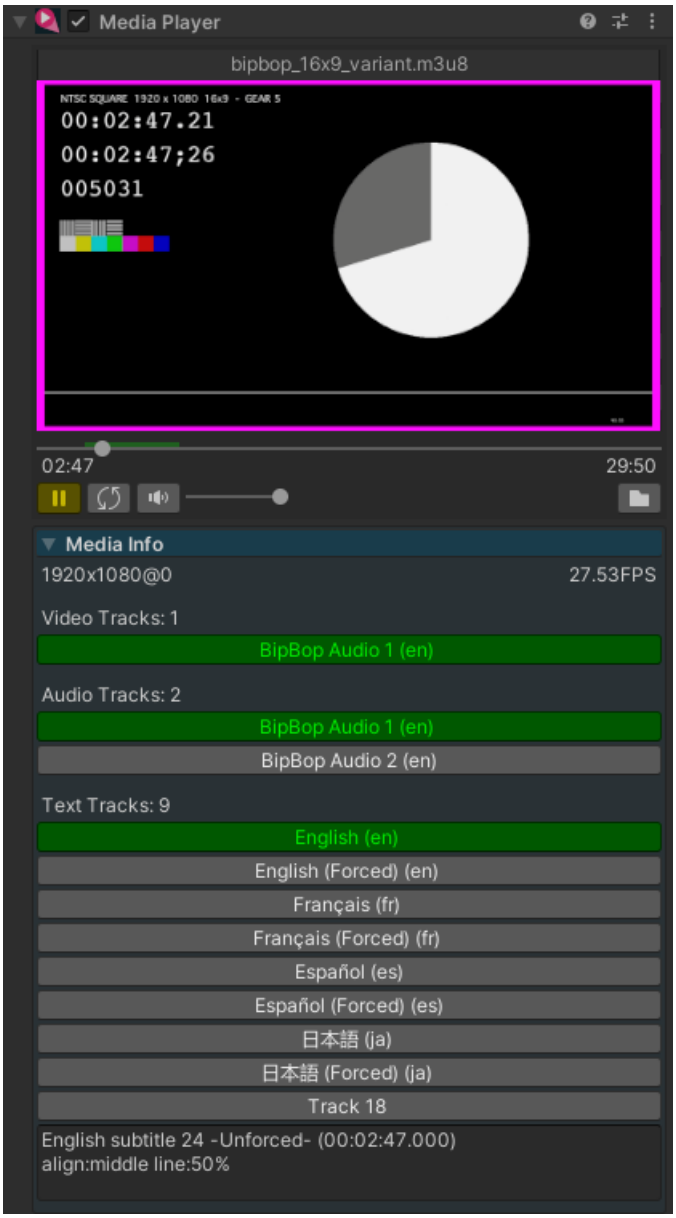
Subtitles



Various subtitles formats are supported:

	WEBVTT IN HLS	CEA / EIA 608 & 708	TX3G IN MP4 / MOV	SRT SIDELOADING
Windows (WinRT / Media Foundation)	□	□	.	□
Windows (DirectShow)	.	.	.	□
UWP (WinRT / Media Foundation)	□	□	.	□
Android (ExoPlayer)	□	.	.	□
Android (MediaPlayer)	.	.	.	□
macOS	□	.	□	□
iOS/iPadOS/tvOS	□	.	.	□
WebGL	.	.	.	□

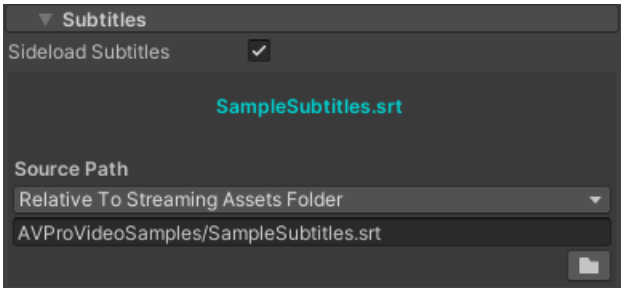
When loading media that contains subtitles tracks, the tracks are displayed in the Inspector allowing tracks to be selected and a preview of the subtitle content to be shown:



Text tracks can be scripted:

```
// Get the number of text tracks
int trackCount = mediaPlayer.TextTracks.GetTextTracks().Count;

// Iterate through the tracks
foreach (TextTrack track in mediaPlayer.TextTracks.GetTextTracks())
{
    Debug.Log(track.DisplayName);
}

// Get information about the active text track
TextTrack track = mediaPlayer.TextTracks.GetActiveTextTrack();
if (track != null)
{
    Debug.Log(string.Format("{0}:{1}", track.Name, track.Language));
}
else
{
    Debug.Log("No active text track");
}

// Set the active text track
mediaPlayer.TextTracks.SetActiveTextTrack(track);

// Get the current text cue
TextCue textCue = mediaPlayer.TextTracks.GetCurrentTextCue();
if (textCue != null)
{
    Debug.Log(textCue.Text);
}
```

SRT Sideloading



AVPro Video supports external subtitles in the SRT format on all platforms.

```
// Load subtitles
mediaPlayer.SideloadSubtitles = true;
mediaPlayer.EnableSubtitles(MediaLocation.RelativeToStreamingAssetsFolder, "subtitles.srt");

// Disable subtitles
mediaPlayer.DisableSubtitles();
```

Displaying

You can create your own system to display subtitles, or use the included SubtitlesUGUI component.

Video Capture

To make a non-realtime video capture of your Unity scenes which include videos, requires the video playback to slow down or speed up to match the video capture rate. AVPro Video supports this through the `“TimeScale Support”` option which is found in the `Global Settings` panel of the Media Player component. This means you can create high quality renders running at 1fps to produce a smooth 60fps video, and any videos in your scene will play back at the correct rate for the recording. Audio is not supported though when using this option.

Content Protection

Most of the content protection features are in the Ultra Edition, but the Core edition does support the file offsetting on Android.

File Offset

On Android (this is the only platform that currently supports this feature) in the Core Edition a file offset (in bytes) can be specified which allows loading of media which is embedded within another file. This is very useful for hiding media.

In Windows you can use the following command via the command-line to easily append your video to a dummy file:

```
copy /b DummyFile.dat + MyVideo.mp4 MysteryFile.dat
```

The offset can be set via the UI in the `Platform Specific > Android` section, or via `PlatformOptionsAndroid` in scripting:

```
// Set the Android file offset to 54321 bytes (the size of the dummy file, or the offset into a file if you have as embedding it within a file)
mediaPlayer.PlatformOptionsAndroid.fileOffset = 54321;
```

An alternative is to use a dummy video file, and then append your real media to this file, as this will then allow the dummy video to play instead of your real media, making it not obvious that you're hiding your video.

In Windows you can use the following command via the command-line to easily append your video to a dummy video:

```
copy /b DummyVideo.mp4 + MyVideo.mp4 MysteryFile.mp4
```

In Windows you can use the following command to create a batch file for converting multiple files easily:

```
copy /b DummyVideo.mp4 + %1 %~n1-hidden.mp4
```

Seeking / Playback Rate

Seeking

All time operations are done in seconds using doubles:

```
// After the video is loaded and metadata event fires you can use these:

// Get the media duration in seconds
double duration = mediaPlayer.Info.GetDuration();

// Get current time in seconds
double time = mediaPlayer.Control.GetCurrentTime();

// Get the ranges of time that can be seeked between
TimeRanges seekRanges = mediaPlayer.Control.GetSeekableTimes();

// Seek to 24 seconds
mediaPlayer.Control.Seek(24.0);

// Seek to nearest keyframe at 24 seconds
mediaPlayer.Control.SeekFast(24.0);

// Seek to closest keyframe allowing keyframe to be either ahead, behind or on both sides of the desired time
// This is only currently available on macOS, iOS, iPadOS and tvOS
mediaPlayer.Control.SeekWithTolerance(24.0, 5.0, 0.0);

// Seek to the current 'live' time for a live stream
TimeRange seekableRange = Helper.GetTimelineRange(mediaPlayer.Info.GetDuration(),
mediaPlayer.Control.GetSeekableTimes());
mediaPlayer.Control.Seek(seekableRange.endTime);
```

Media that has a known constant frame rate can be seeked using frames:

```
// After the video is loaded and metadata event fires you can use these:

// Get the media duration in frames
int durationFrames = mediaPlayer.Info.GetDurationFrames();

// Get the highest frame number you can seek to (the same as durationFrames-1)
int maxFrame = mediaPlayer.Info.GetMaxFrameNumber();

// Seek to frame 60
mediaPlayer.Control.SeekToFrame(60);

// Seek back 10 frames
mediaPlayer.Control.SeekToFrameRelative(-10.0);

// Get current time in frames
int frame = mediaPlayer.Control.GetCurrentTimeFrames();
```

If the frame rate can not be determined (eg in some HLS media the frame rate returns zero) then you can still use the frame-based time methods by manually supplying the frame rate as an optional final parameter to the above methods:

```
// After the video is loaded and metadata event fires you can use these:

// Get the media duration in frames
int durationFrames = mediaPlayer.Info.GetDurationFrames(30f);

// Seek to frame 60
mediaPlayer.Control.SeekToFrame(60, 30f);

// Get current time in frames
int frame = mediaPlayer.Control.GetCurrentTimeFrames(30f);
```

There are some platform differences for seeking behaviour:

	FAST APPROXIMATE KEYFRAME SEEKING	SLOW ACCURATE FRAME SEEKING
Windows (WinRT / Media Foundation)	☐	☐
Windows (DirectShow)	☐	Depends on the codec
Android (ExoPlayer)	☐	☐
Android (MediaPlayer)	☐	API 26 and above
macOS	☐	☐
iOS/iPadOS/tvOS	☐	☐
WebGL	☐	Varies

Playback Rate

Generally we recommend these rates:

```
0.25, 0.5, 1.0, 1.25, 1.5, 1.75, 2.0
```

Going up to 4.0 might be possible depending on your platform, machine specs and the codec used. Increasing playback rate usually places more demand on the video decoder and also on the disk/network source, so these limit how high you can set the playback rate.

Using negative values isn't generally recommended as it isn't as well supported, but if you do have to use a negative rate then also try keeping the numbers small such as:

```
-0.25, -0.5, -1.0
```

Audio also may or may not play when changing the playback rate - this depends on the platform (see table below).

One safe alternative to adjusting rate is to pause the video and fast seek to simulate a change in playback rate. This approach would work on all platforms.

Video encoding can also help the performance of a change in playback rate. Videos with more key-frames (or ideally all key-frames) and with less complex encoding (eg no B frames, CABAC disabled etc) will work better. Alternatively a fast key-frame-only codec could be used, such as Hap.

Scripting playback rate:

```
// After the video is loaded and metadata event fires you can use these:

// Get the current playback rate
float rate = mediaPlayer.PlaybackRate;

// Set the current playback rate
mediaPlayer.PlaybackRate = rate * 2f;
```

There are some platform differences for playback-rate behaviour:

	ADJUST PLAYBACK RATE	NEGATIVE RATES	AUDIO PLAYS
Windows (WinRT / Media Foundation)	□	□	Depends on codec
Windows (DirectShow)	□	.	.
Android (ExoPlayer)	□	?	□
Android (MediaPlayer)	API 23 and above	?	□
macOS	□	Depends on media source	□
iOS/iPadOS/tvOS	□	Depends on media source	□
WebGL	□	.	Depends on browser

macOS/iOS/iPadOS/tvOS Playback Rate

By default playback rates higher than 2.0 causes the player to only show key-frames. If you need to play back a video at rates above 2.0 then you can adjust the `"Max Playback Rate"` slider in the platform specific options to set a higher threshold. Should your playback rate exceed this new maximum rate then the player will drop back to showing just key-frames. This option must be configured prior to creating the player and cannot be changed after the media has been opened.

Optimal Encoding

Most videos are optimally encoded for the typical use case: normal forward playback with approximate seeking.

If you want to start changing the playback rate, play in reverse, allow fast scrubbing, or have fast frame accurate seeking then you may run into issues where the playback becomes extremely slow or the seeking is not accurate. There are several reasons for this, but it mostly is related to how the video is encoded and specifically the key-frame distribution. There are also some platform differences to consider.

Codecs such as H.264 and H.265 generally compress video frames so that they depend on data included with previously decoded frames. These are called P and B frames and seeking to one of these is computationally expensive, as in order to display them the decoder must first decode the other frames that they depend on. The other type of frame is called a key-frame or I-frame and these frames can be decoded immediately because they don't depend on any other frames. Compressing using P And B frames is known as temporal compression and isn't ideal for accurate random seeking or playback rate changes.

For the best results you would encode your video with only key-frames, as then you can seek accurately and quickly anywhere in the video. This is possible, but increases the file size dramatically. Using FFMPEG you can encode a video to use key-frames only using the "-g 1" option. Another option would be to use a codec that only supports key-frames, such as Hap or ProRes - but again these result in large file sizes and limited GPU decoding capabilities.

In most codec with temporal compression the key-frames are spaced every 250 frames. Some platforms can only seek to the key-

frames (see table above), while others can do accurate seeking but this can be very slow if the distances between key-frames is too large. Try reducing the key-frame distance for faster seeking. You can also reduce the decoder complexity by encoding with a `fastdecode` tuning option.

Here is an example FFMPEG command to encode using H.264 codec with all keyframes for very fast seeking:

```
ffmpeg -hide_banner -y -i input.mp4 -pix_fmt yuv420p -c:v libx264 -crf 18 -tune fastdecode -x264-params
"keyint=1" output-h264.mp4
```

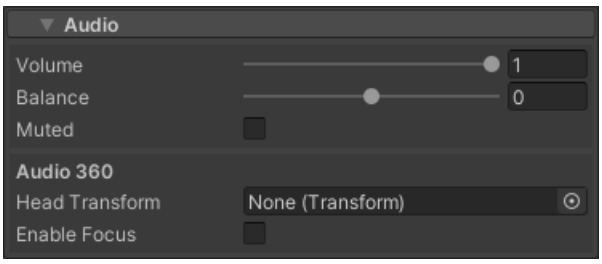
See the section about encoding for further information.

360 Audio

Spatial audio support is currently available using Facebook Audio 360 on Windows desktop, UWP and Android. On Windows, only Windows 10 and above is supported and the Media Foundation video API must be selected. On Android the ExoPlayer video API must be selected. The video files must be using a MKV file container and audio must be using the Opus codec encoded with Facebook Audio 360 tools.

> **NOTE**
>
> Unfortunatly Meta/Facebook are no longer updating their "Facebook Audio 360" technology. The download links for the `Facebook 360 Spatial Workstation` authoring tools are currently broken but apparently will be made live soon.

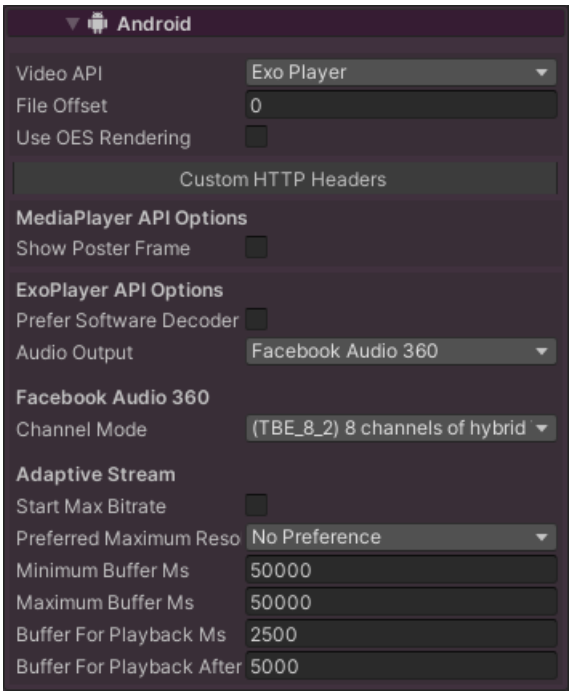
The best way to encode the video is to use the FB360 Encoder tool which comes as part of the FB360 Spatial Workstation. Set Output Format to "FB360 Matroska (Spatial Workstaton 8 channel)" and then set your video and audio source files and encode your video. This should create a MKV file with 10 channels of Opus audio.



The settings are located under the Audio section of the MediaPlayer component. The `"Head Transform"` field must be set to the transform that represents the player's head so that rotation and positional changes affect the audio rendering. Usually this is the main camera.

`"Enable Focus"` can be enabled when a specific region of audio in the 360 field needs to be given focus. The rest of the audio has its volume reduced.

Next the Facebook Audio 360 support must be enabled for each platform that needs it via the `"Platform Specific"` panel. Currently it is only available on Windows desktop and Android.



The `Channel Mode` must be set to the channel encoding mode used when creating the video. Currently this can not be determined automatically. The default is `TBE_8_2` which means 8 channels of hybrid ambisonics and 2 channels of head-locked

stereo audio.

```
// Set the head transform for audio
mediaPlayer.AudioHeadTransform = Camera.main;

// Set the MediaPlayer for spatial audio on Android - ExoPlayer API only
mediaPlayer.PlatformOptionsAndroid.videoApi = Android.VideoApi.ExoPlayer;
mediaPlayer.PlatformOptionsAndroid.audioOutput = Android.AudioOutput.FacebookAudio360;
mediaPlayer.PlatformOptionsAndroid.audio360ChannelMode = Audio360ChannelMode.TBE_8_2;

// Set the MediaPlayer for spatial audio on Windows - Media Foundation API only
mediaPlayer.PlatformOptionsWindows.videoApi = Windows.VideoApi.MediaFoundation;
mediaPlayer.PlatformOptionsWindows.audioOutput = Windows.AudioOutput.FacebookAudio360;
mediaPlayer.PlatformOptionsWindows.audio360ChannelMode = Audio360ChannelMode.TBE_8_2;
```

Encoding with Facebook 360 Spatial WorkStation

More information on encoding etc can be found on the Facebook Audio 360 website at: https://facebookincubator.github.io/facebook-360-spatial-workstation/.

Alternative steps for encoding manually

1. Create a WAV file with the audio format they need (Eg 9 channels ambisonics with 2 channels of head-locked audio will require a 11 channel WAV file with the 2 head-locked channels at the end)
2. Use Opus tools, as described here to convert the WAV file to Opus: https://opus-codec.org/downloads/ https://facebookincubator.github.io/facebook-360-spatial-workstation/Documentation/SDK/Audio360_SDK_GettingStarted.html#encoding-opus-files
3. Use ffmpeg to mux this opus file into the video container (ensure that the video file doesn't have any audio first):

   ```
   ffmpeg -i audio.opus -i video.mp4 -c:a copy -c:v copy audio_video.mkv
   ```

4. In AVPro Video specify the required channel map

Converting existing ambisonic videos

It is also possible to convert existing ambisonic videos so they are compatible. For example if you have an existing MP4 file with 4-channel 1st order ambisonic audio, then it is possible to convert this into the above format (MKV container with Opus audio) using a tool like FFMPEG. Simply put the following command in a .BAT file and then drag your MP4 into the batch file:

```
ffmpeg -y -i input.mp4 -c:v copy -acodec libopus -mapping_family 255 output.mkv
```

This should then generate an MKV file that you can play with AVPro Video. All that remains is to set the channel mapping in the MediaPlayer component to AMBIX_4.

10-Bit Video

AVPro Video has growing support for playback of 10-bit content. Encoding videos as 10-bit can give improved quality for very little extra storage space, especially for gradients.

NOTE

We are not talking about HDR videos here. We're simply talking about videos encoded with a higher per-pixel bit depth.

NOTE

Even if you are displaying the 10-bit video on an 8-bit screen, there are still quality benefits to using 10-bit video.

WARNING

Ability to decode 10-bit depending on your operating system and GPU capabilities. This is still a relatively new technology and shouldn't be used if compatibility is a high priority.

Editions

Ultra Edition

The Ultra edition now supports 10-bit textures on Windows, macOS, iOS and tvOS.

Core Edition

On the Core Edition only 8-bit textures are supported. In this case the 10-bit videos will be rendered to 8-bit, either directly or sometimes the driver will apply some smart dithering.

Platform Specifics

Windows Support

In the Ultra Edition you can specify to use 10-bit textures (in the `Platform Specific` section). This will give the best results when displaying to a 10-bit back buffer and monitor. We have found that using the `WinRT` API setting gives the best quality decode for 10-bit, whereas `Media Foundation` API setting still seems to add some sort of dithering/resolve step.

Currently Windows is only officially supported:

	H.264 MAIN 10 PROFILE	HEVC MAIN 10 PROFILE	VP9	AV1
Windows (WinRT / Media Foundation)	. [1]	□ [2]	?	?
Windows (DirectShow)	□ [3]	□ [3]	?	?

[1] Microsoft's H.264 Decoder doesn't support the Main 10 profile or output to 10-bit (P010) textures

[2] Requires Windows 10 and Microsoft's HEVC Video Extensions. WinRT API gives the best quality results

[3] Only when using a suitable 3rd-party codec such as LAV Filters

iOS / macOS / tvOS

iOS 11.0 and above

Only HEVC Main 10 Profile is supported.

iOS, macOS and tvOS will automatically generate textures capable of supporting 10 bits per component when the source media is 10-bit.

Depending on the texture format chosen this will be:

TEXTURE FORMAT
BGRA
Y CbCr 420

Other platforms

This is a new feature which we hope to expand support and documentation for soon.

Encoding

Using the command-line tool FFMPEG we have found the following command useful in testing:

```
ffmpeg -y -i %1 -pix_fmt yuv420p10le -color_primaries 1 -color_trc 1 -colorspace 1 -color_range 2 -crf 4 -
vcodec libx265 -movflags +write_colr -movflags +faststart %i-hevc-10bit-rec709-high.mp4
```

NOTE

For best results use `Rec709` as we do not yet support other primaries/colorspaces and incorrect rendering of colours will result otherwise.

Hap Codec

The Hap video codec is natively supported by AVPro Video on macOS and Windows in the Ultra Edition, and has the following benefits:

- Low CPU usage
- Low memory usage
- GPU decompression
- Supports very high resolutions
- Supports alpha channel transparency
- Fast seeking and variable playback speed
- Doesn't use GPU video decoding resources

The main downside is:

- Very large files

Windows Support

Hap, Hap Alpha, HapQ, HapQ Alpha and HapR are supported. AVI and MOV containers can both be used in Windows however we recommend the MOV container. Hap currently requires either the “DirectShow” or “Media Foundation” video API to be used.

> **⚠ WARNING**
>
> There is currently a bug in Windows that prevents MOV files with a bitrate of more than 4Gbps from playing. We have created a fix for this which is activated by enabled `"Use Custom MOV Parser"` in Platform Specific >Windows options.

Options

On Windows some options are exposed for Hap and NotchLC decoding:

- Use Custom MOV Parser
    - Enables our custom MOV parser to be used, which is useful for Hap and NotchLC codecs, as a Microsoft parser is not able to open very high bit-rate MOV files. If your files are high bit-rate (above 4Gbps) then use this option otherwise the file will not load.
    - Parallel Frame Count
        - Maximum number of threads to use for parallel frame decoding. Less threads for less latency in playback operations (seeking, playing etc), more threads for better performance.
    - Preroll Frame Count
        - Amount of frames to decoder before starting playback, less frames for less latency in seeking, more frames for less chance of buffer emptying too quickly.

macOS Support

Hap, Hap Alpha, HapQ, HapQ Alpha and HapR are supported.

Encoding

> **NOTE**
>
> Width and height must be multiple of a 4 for Hap encoded videos in AVPro Video. Ideally a width with a multiple of 16 is best for performance

(aligned copy).

JOKYO

In 2020 Jokyo introduced their fast and high quality Hap encoder with plugins for Adobe After Effects, Premiere Pro etc: https://jokyohapencoder.com

FFMPEG

Alternatively you can use a recent build of FFMPEG with the following command-lines:

- `ffmpeg -i input.mov -vcodec hap -format hap output-hap.mov`
- `ffmpeg -i input.mov -vcodec hap -format hap_alpha output-hap.mov`
- `ffmpeg -i input.mov -vcodec hap -format hap_q output-hap.mov`

Notes:

- You can also add the `-chunks 4` option which will encode each frame into 4 chunks so the decoding work can be split across multiple threads, resulting in faster decoding as long as the disk can keep up.
- Hap Alpha requires straight alpha (not pre-multipled).
- Sadly ffmpeg doesn't yet support the HapQ Alpha format.
- We don't support Hap Q Alpha variant in Windows when using the legacy D3D9 graphics API
- See Encoding Notes for more details

NotchLC Codec

NotchLC is a product of 10bitFX Limited (www.notch.one)

The NotchLC video codec is natively supported by AVPro Video on Windows in the Ultra Edition, and has the following benefits:

- Low CPU usage
- Low memory usage
- GPU decompression
- 10-bit colour
- Supports very high resolutions
- Supports alpha channel transparency
- Fast seeking and variable playback speed
- Higher quality than the Hap codec
- Doesn't use GPU video decoding resources

The main downside is:

- Very large files

Windows Support

AVI and MOV containers can both be used in Windows however we recommend the MOV container. NotchLC currently requires the “Media Foundation” video API to be used.

> ⚠ **WARNING**
>
> There is currently a bug in Windows that prevents MOV files with a bitrate of more than 4Gbps from playing. We have created a fix for this which is activated by enabled `"Use Custom MOV Parser"` in Platform Specific >Windows options.

Options

On Windows some options are exposed for Hap and NotchLC decoding:

- Use Custom MOV Parser
    - Enables our custom MOV parser to be used, which is useful for Hap and NotchLC codecs, as a Microsoft parser is not able to open very high bit-rate MOV files. If your files are high bit-rate (above 4Gbps) then use this option otherwise the file will not load.
    - Parallel Frame Count
        - Maximum number of threads to use for parallel frame decoding. Less threads for less latency in playback operations (seeking, playing etc), more threads for better performance.
    - Preroll Frame Count
        - Amount of frames to decoder before starting playback, less frames for less latency in seeking, more frames for less chance of buffer emptying too quickly.

Encoding

Encoding notes are in the Notch user manual

Content Protection

Content protection schemes supported by the plugin are summarised as:

- Windows Desktop & UWP
    - Custom HTTP header fields can be specified which can help with server side validation (requires using the WinRT API). This is only supported for adaptive media (HLS/DASH).
    - HLS with AES-128 clear-key, direct key and key request is supported
- Android
    - Custom HTTP header fields can be specified which can help with server side validation
    - HLS with AES-128 clear-key, direct key and key request is supported (make sure your TS segments are 188 bytes aligned for maximum Android compatibility)
    - A file offset feature allows you to access files hidden within a file at an offset.
- macOS / iOS / tvOS
    - HLS with AES-128 clear-key, direct key and key request using an auth token in the HTTP header ("Authorization" field). More information about HLS encryption can be read in the RFC here: https://tools.ietf.org/html/draft-pantos-http-live-streaming-23
    - Custom HTTP header fields can be specified which can help with server side validation

> **NOTE**
>
> DRM schemes Fairplay, Widevine, PlayReady etc are not yet supported

File Offset

On Android (this is the only platform that currently supports this feature) in the Core Edition a file offset (in bytes) can be specified which allows loading of media which is embedded within another file. This is very useful for hiding media.

In Windows you can use the following command via the command-line to easily append your video to a dummy file:

```
copy /b DummyFile.dat + MyVideo.mp4 MysteryFile.dat
```

The offset can be set via the UI in the `Platform Specific > Android` section, or via `PlatformOptionsAndroid` in scripting:

```
// Set the Android file offset to 54321 bytes (the size of the dummy file, or the offset into a file if you have as embedding it within a file)
mediaPlayer.PlatformOptionsAndroid.fileOffset = 54321;
```

An alternative is to use a dummy video file, and then append your real media to this file, as this will then allow the dummy video to play instead of your real media, making it not obvious that you're hiding your video.

In Windows you can use the following command via the command-line to easily append your video to a dummy video:

```
copy /b DummyVideo.mp4 + MyVideo.mp4 MysteryFile.mp4
```

In Windows you can use the following command to create a batch file for converting multiple files easily:

```
copy /b DummyVideo.mp4 + %1 %~n1-hidden.mp4
```

Custom HTTP Headers

Custom HTTP headers can be specified in the Ultra Edition. Typically we have seen them used for authentication, token exchange and cookies. Here are Some examples of formats we've used in the past:

For authentication the typical HTTP headers are:

```
Authorization Basic <username>:<password>
Authorization Bearer <token>
```

For cookies the typical HTTP headers are:

```
Cookie <cookie-name>=<cookie-value>;<cookie-name2>=<cookie-value2>;
```

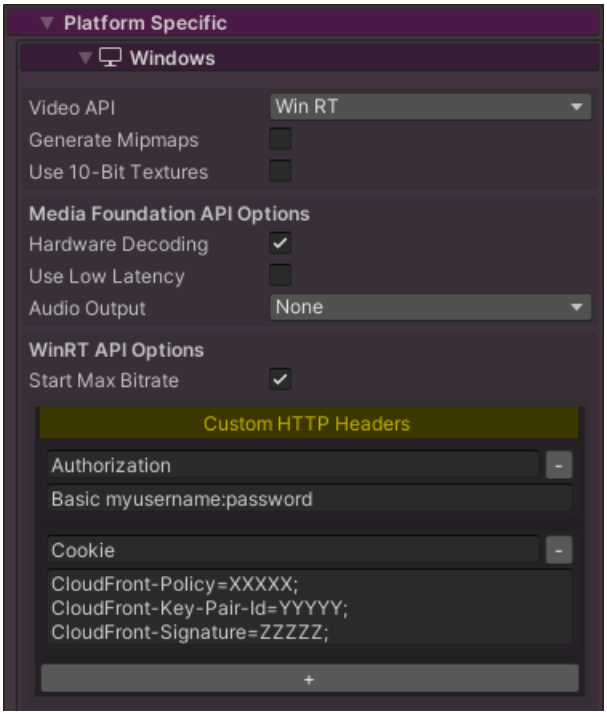
NOTE

On Windows custom HTTP headers are only supported on the WinRT API, and are only supported for adaptive media (HLS and DASH).

NOTE

On Android custom HTTP headers are only supported on the ExoPlayer API, the MediaPlayer API doesn't support this.

In the plugin custom headers can be specified via the component UI or via script. Using the component UI you would specify them like:



NOTE

The fields will turn red in the editor if the format is not correct

Or via scripting:

```
// On Windows the Video API needs to be set to WinRT to use custom HTTP headers
mediaPlayer.PlatformOptionsWindows.videoApi = Windows.VideoApi.WinRT;

// Set custom HTTP headers on Android platform
mediaPlayer.PlatformOptionsAndroid.httpHeaders.Add("Authorization", "Bearer <token>");
mediaPlayer.PlatformOptionsAndroid.httpHeaders.Add("Cookie", "<cookie-name>=<cookie-value>;<cookie-name2>=
<cookie-value2>;");

// For Basic Authorization the <username>:<password> should be base64 encoded:
string username = "user";
string password = "password";
string base64token = System.Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(username + ":" +
password));
mediaPlayer.PlatformOptionsAndroid.httpHeaders.Add("Authorization", "Basic " + base64token);

// Set custom HTTP headers on the currently running platform
mediaPlayer.GetCurrentPlatformOptions().httpHeaders.Add("MyHlsUriToken", "1234567890");
```

NOTE

Basic Authorization headers don't appear to work when using WinRT and RTSP streams

NOTE

Internally custom HTTP headers are passed in this string format: `name1:value1\r\nname2:value2`

HLS AES-128 Encrypted Playback

In the Ultra Edition AES-128 HLS streams are supported. This allows playback of encrypted content with secure key exchange. There is support on macOS, iOS, tvOS, Android (only using Exoplayer API), Windows and UWP (only using WinRT API).

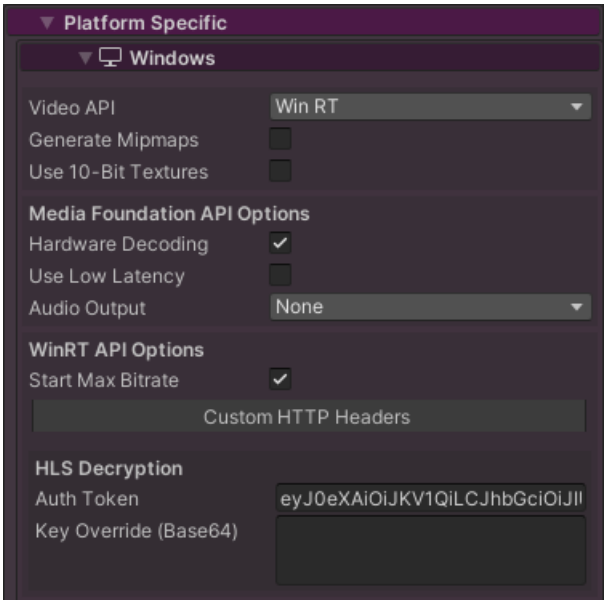
Key retrieval from a server URL usually requires an authentication token, which can be specified using the member `keyServerToken)`, or this can be ignored if your key retrieval server doesn't require any token for key retrieval (clear-key). The auth token is a string that is inserted into the "Authorization" HTTP field when retrieving the decryption key from the server URL specified in the HLS manifest. Sometimes this field has the "Bearer" prefix.

We also added some functionality to specify the decryption key data directly. `overrideDecryptionKey` can be used to specify the key directly as an array of bytes. Using this will bypass any server key retrieval, which can be useful for debugging.

Scripting examples:

```
// On Windows the Video API needs to be set to WinRT to use AES-128 keys
mediaPlayer.PlatformOptionsWindows.videoApi = Windows.VideoApi.WinRT;

// Set the authentication token for the key server to allow access of the decryption key for iOS platform
mediaPlayer.PlatformOptionsIOS.keyAuth.keyServerToken =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1cm46bWljcm9zb2Z0OmF6dXJlOm1lZGlhc2VydmljZXM6Y29udGVudGtleWlkZW50aWZpZXIiOiI5ZGRhMGJjYy01NmZiLTQxNDMtOWQzMi0zYWI5Y2M2ZWE4MGIiLCJpc3MiOiJodHRwOi8vdGVzdGFjcy5jb20vIiwiYXVkIjoidXJuOnRlc3QiLCJleHAiOjE3MTA4MDczODl9.lJXm5hmkp5ArRIAHqVJGefW2bcTzd91iZphoKDwa6w8");

// Just specify a decryption key manually for iOS platform
mediaPlayer.PlatformOptionsIOS.keyAuth.overrideDecryptionKey = new byte[] { 12, 95, 93, 64, 234, 76, 93, 64,
125, 0, 95, 23 };

// Just specify a decryption key manually using base64 for Windows platform
mediaPlayer.PlatformOptionsWindows.keyAuth.overrideDecryptionKey = Convert.FromBase64String(base64Key);

// Just specify a decryption key manually loading from a 16 byte .key file for Windows platform
mediaPlayer.PlatformOptionsWindows.keyAuth.overrideDecryptionKey =
System.IO.File.ReadAllBytes("C:/myfile.key");
```

These options can also be specified in the MediaPlayer inspector UI (in the Platform Specific section) under HLS Decryption:



Caching

You can now cache online media items for future playback on certain platforms. The media player will automatically playback from the cache when a media item is present, otherwise the behaviour will be exactly the same as before with media being played over the internet.

Currently caching is not automatic and you need to specifically request that media is added to the cache.

You can check if the current platform supports caching as follows:

```
if (mediaPlayer.Cache.IsMediaCachingSupported())
{
    // Caching is supported
}
```

In order to add media to the cache, call:

```
mediaPlayer.Cache.AddMediaToCache(url, headers, options);
```

where:

- `url` is the URL of the asset to cache
- `headers` are any HTTP headers required to access the asset
- `options` are any options to configure how the media is cached:
    - `minimumRequiredBitRate` the lowest bitrate that is acceptable to cache
    - `minimumRequiredResolution` the lowest resolution that is acceptable to cache
    - `maximumRequiredBitRate` the highest bitrate that is acceptable to cache (Android only)
    - `maximumRequiredResolution` the highest resolution that is acceptable to cache (Android only)

To remove media from the cache, call:

```
mediaPlayer.Cache.RemoveMediaFromCache(url);
```

where:

- `url` is the URL of the asset to remove

To stop the caching of a media item:

```
mediaPlayer.Cache.CancelDownloadOfMediaToCache(url)
```

where:

- `url` is the URL of the asset to cancel caching

To pause an active caching download of a media item (Android only):

```
mediaPlayer.Cache.PauseDownloadOfMediaToCache(url)
```

where:

- `url` is the URL of the asset to pause caching

To resume a paused caching download of a media item (Android only):

```
mediaPlayer.Cache.ResumeDownloadOfMediaToCache(url)
```

where:

- `url` is the URL of the asset to resume caching

To get the status of media in the cache:

```
float progress = 0.0f;
CachedMediaStatus status = mediaPlayer.Cache.GetCachedMediaStatus(url, ref progress);
```

where:

- `url` is the URL of the asset to cancel caching
- `progress` returns the current progress in the range 0...1 of the download should the returned status be `CachedMediaStatus.Caching` and `status` is one of:
- `CachedMediaStatus.NotCached` the media is not cached
- `CachedMediaStatus.Caching` the media is currently being cached
- `CachedMediaStatus.Cached` the media is cached

To see if the currently open media will play or is playing from the cache:

```
if (mediaPlayer.Cache.IsMediaCached())
{
    // The open media is cached
}
```

Platform specifics

Android

The following limitiations exist for Android:

- Storing and playback of media from cache is currently only supported when using the ExoPlayer API path for playback.
- Cached media items are downloaded to the default external cache folder. They are stored in a proprietary ExoPlayer format and are not readable/usable externally from the ExoPlayer architecture.
- Only remote video files can be cached to the local file system.
- HLS/DASH/SmoothStream and MP4 can all be cached.
- Playback whilst caching is possible but will depend on the available network resources.
- Currently only caching of clear-key (where the key is included in the manifest) AES encrypted streams is supported.
- The following `MediaCachingOptions` will not be supported: `title`, `artwork`.

iOS

Cached media items are listed in the Settings app in the iPhone Storage or iPad Storage page of the General section. By default the name given to the items is taken from the URL so it might not be that user friendly. You can use the `title` and `artwork` members of the `MediaCachingOptions` class to provide more user friendly details as follows:

```
MediaCachingOptions options = new MediaCachingOptions();
options.title = "User Friendly Name";

TextAsset artwork = Resources.Load<TextAsset>("artwork.png");
if (artwork)
{
    options.artwork = artwork.bytes;
}

_mediaPlayer.Cache.AddMediaToCache("https://example.com/media/movie.m3u8", null, options);
```

The following limitiations exist for iOS:

- iOS 11.0 and above
- Only caching of HLS videos is supported, not progressive downloads (i.e. mp4 files).
- Only caching of VOD streams is supported, Live streams will fail unless the stream has already finished and is complete in the manifest.
- Playback whilst caching is possible but will depend on the available network resources. Caching is likely to be paused whilst a video is playing in order to maintain the best playback quality for the viewer.
- Currently only the variants marked as DEFAULT in the playlist will be cached.
- Cached videos may be removed by the system if storage space is low on the device.
- Currently only caching of clear-key (where the key is included in the manifest) AES encrypted streams is supported.
- `MediaCachingOptions.minimumRequiredResolution` is supported on Android and iOS 14.0 and later only.
- `MediaCachingOptions.maximumRequiredBitRate` and `MediaCachingOptions.maximumRequiredResolution` are not supported.
- Currently Pause/Resume of an active video caching download is not supported.

macOS / tvOS

Caching is not currently supported on macOS or tvOS.

Windows / UWP

Caching is not currently supported on Windows or UWP.

Smooth Video (Experimental)

> ⚠ **WARNING**
>
> This is an experimental feature.

This feature allows video playback to be perfectly smooth by not dropping or duplicating frames and presenting frames at the same rate at the display refresh rate.

This is particularly important for very large video walls where a single glitch in playback can be very noticable.

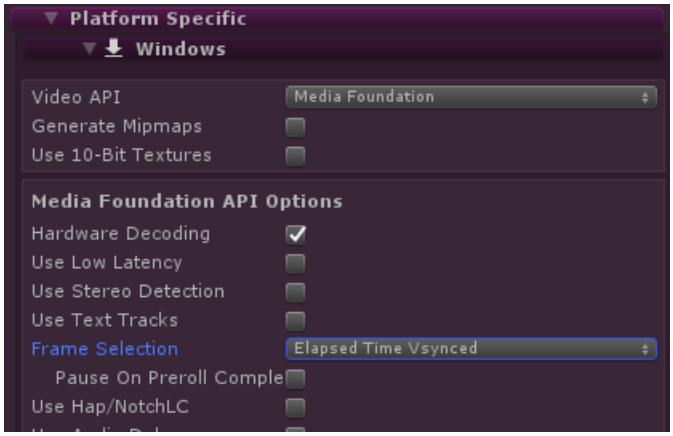
Limitations

1. This feature only works on Windows and uses the Media Foundation API.
2. Hap/NotchLC codecs are not supported

Setup

To use this feature:

1. Add the text `AVPROVIDEO_SUPPORT_BUFFERED_DISPLAY` to `Edit > Project Settings > Player > Scripting Define Symbols`. You can also enable `Developer Mode` in the `Global Section` of the `MediaPlayer` component, and then tick the `Support Buffered Display` option.
2. Now a new `Frame Selection` option will be available in the `MediaPlayer` component . Set this to `Elapsed Time Vsynced`.



3. Ensure that vsync is enabled in your application (`Edit > Project Settings > Quality`). In the Unity editor make sure it's also enabled in the Game View.
4. Ensure that your display vsync rate is a perfect multiple of your media frame rate. For example a display that is 60 FPS is fine for media that is 60 or 30 FPS.
5. For best results your video shouldn't contain audio, but if they do then the audio may slightly ahead of the video. In this case you can enable the option `Use Audio Delay` to counteract this.
6. Don't run any scripts etc that delay the main thread, otherwise sync may be lost

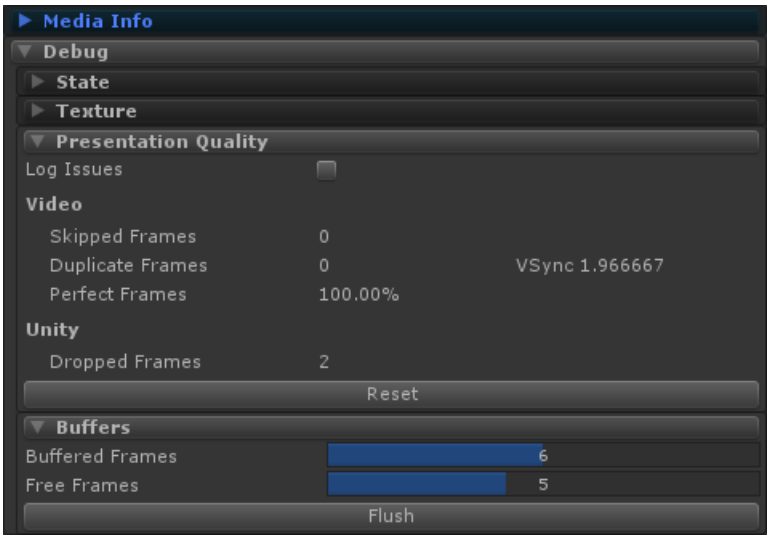
> ℹ **NOTE**
>
> In the Unity editor it is difficult to achieve perfect playback. For best results play using a standalone IL2CPP build (ideally with full-screen mode set to exclusive).

How it works

1. When the video loads it will being playing internally (pre-roll) and buffer those frames.
2. Once pre-roll is complete the video can begin playback.
3. Scripted logic selects which frame to display from the collection of buffered frames, based on vsync rate, Unity dropping frames etc.
4. Meanwhile the buffers are continued to be filled up ahead of what is being displayed
5. As long as the buffers are used and filled at the same rate, smooth playback should continue.

Debugging

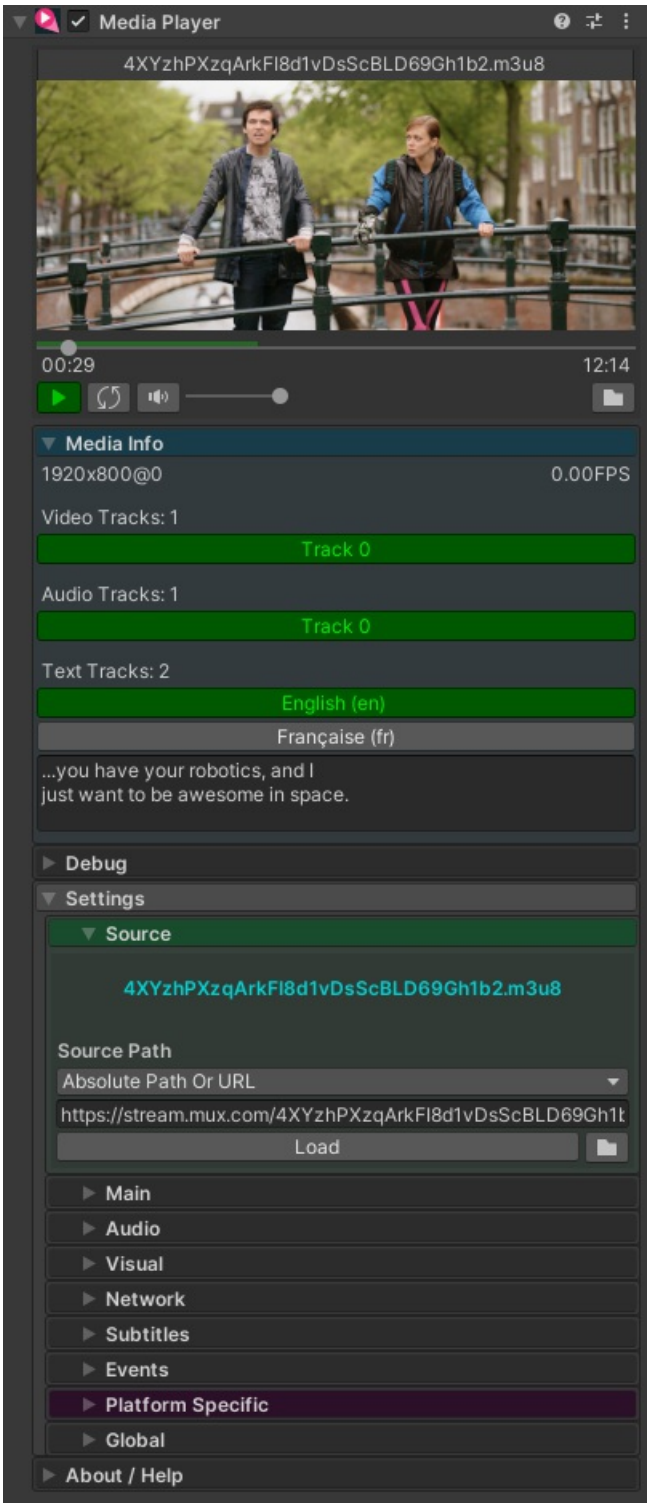
We added some monitors to help understand how the buffers are logic are performing:



The `Buffers` section shows the health of the buffers. For best results `Buffered Frames` should be at least 4 and `Free Frames` should be at least 3.

It can be difficult to spot dropped or duplicated frames, so the `Presentation Quality` section tries to detect these and give you an overview on the smoothness.

Media Player

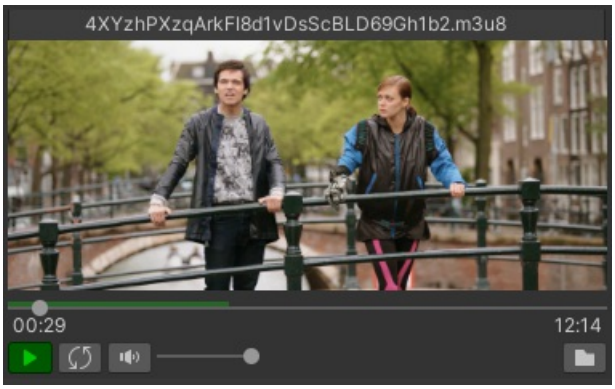


The MediaPlayer component is the primary component of AVPro Video. It handles loading of media, setting playback options, playback and interacting with the other AVPro Video components.

This component does not display the video in the Unity scene. For this you need to use one of these components: ApplyToMesh, ApplyToMaterial, DisplayUGUI, DisplayIMGUI or your own custom script.

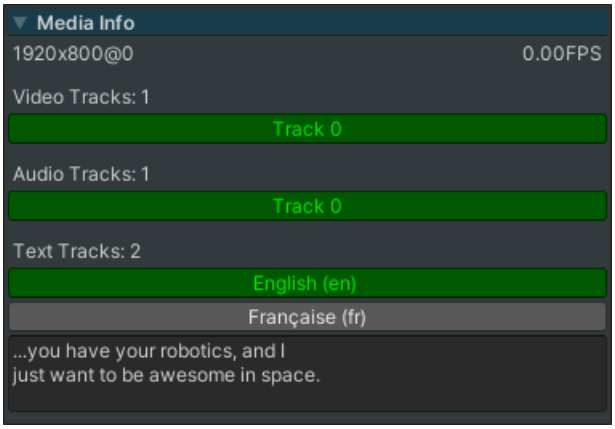
This is a complex component and is split into several sections.

Preview



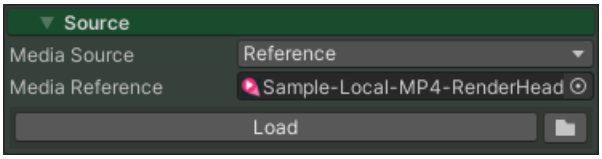
Shows a preview of the playing media and allows control of playback and loading of media. The currently buffered range is shown in green on the timeline control.

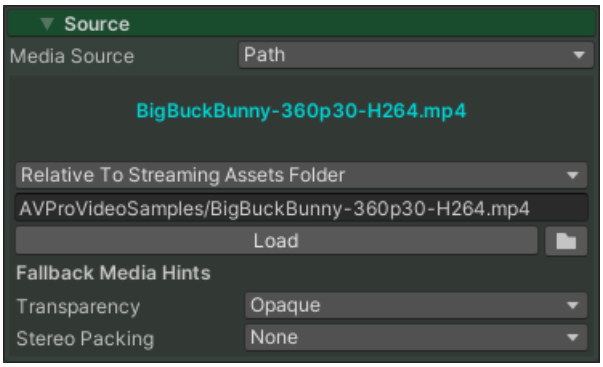
Media Info



Shows information about the media that is currently loaded, including resolution, frame rate (if known), current playback frame rate and track information. Tracks can be switched between and the active text track will show a preview of the current text cue.

Source





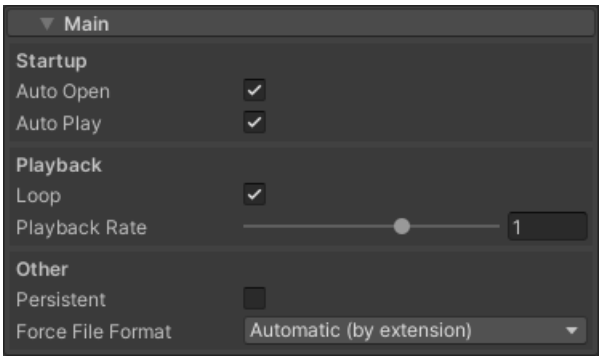
Specifies the location of the media to load. The Load button will load the media immediately. The folder button will show a popup menu allowing file browsing, or shortcuts to recently loaded media or MediaReference assets. See the Loading Media section for more information.

Properties

PROPERTY	FUNCTION
Media Source	Specifies the type of source - either Media Reference or Path. Media Reference allows selection of an existing media asset. Path allows specifying media direct into the MediaPlayer.
Media Reference	The MediaReference asset to use for loading.

PROPERTY	FUNCTION
Source Path	The location of the media (URL or file path).
Fallback Media Hints	
Transparency	This hint specified whether the media contains any transparency
Alpha Packing	If the transparency hint is enabled then an optional Alpha Packing hint can be specified.
Stereo Packing	The packing layout for stereo video

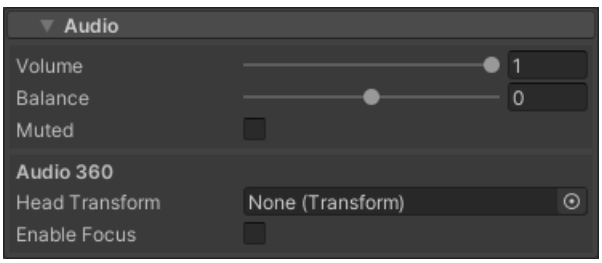
Main



Properties

PROPERTY	FUNCTION
Auto Open	Open/Load the media on Start
Auto Play	Play the media as soon as it has loaded
Loop	Loop the playback
Playback Rate	Speed up or slow down playback by this rate
Persistent	Uses Unity's `Object.DontDestroyOnLoad()` to preserve this GameObject during scene loads
Force File Format	Allows a format to be specified when the correct file extension is not used. This is only supported on Android (using ExoPlayer).

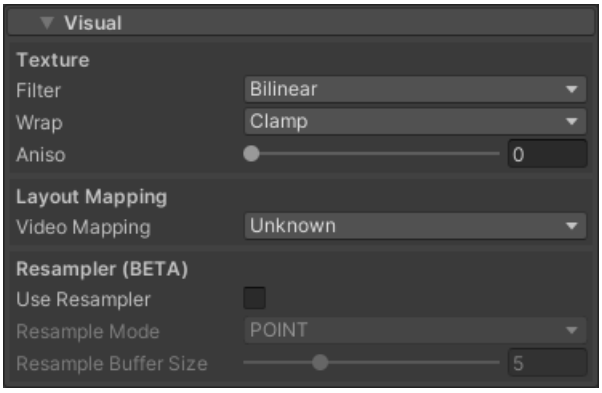
Audio



Properties

PROPERTY	FUNCTION
Volume	Volume in 0..1 range
Balance	Stereo balance in -1..1 range
Muted	Whether to mute audio playback
Audio 360	
Head Transform	Set to the transform that represents the player's head so that rotation and positional changes affect the audio rendering. Usually this is the main camera
Enable Focus	Enable when a specific region of audio in the 360 field needs to be given focus. The rest of the audio has its volume reduced.
Off Focus Level DB	-24..0 range in decibels. How much to reduce the volume by for out of focus audio
Focus Width Degrees	40..120 range in degrees. The angle range for in focus audio
Focus Transform	Transform to use for focus if different from the head transform.

Visual

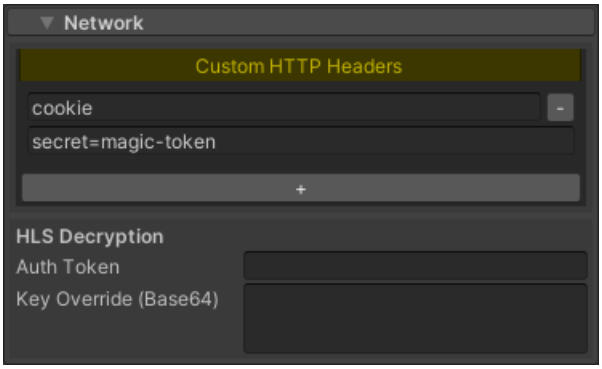


Properties

PROPERTY	FUNCTION
Filter	The texture filter mode to set on the video frame texture
Wrap	The texture wrap mode to set on the video frame texture
Aniso	The anisotropic filter level to set on the video frame texture
Video Mapping	A hint to specify the layout of the video (eg equirectangular, or 180 degree). This is used by some AVPro Video shaders to determine how to display the video
Use Resampler	Enable the video frame resampler. This is useful to smooth out rendering by buffering frame textures and showing them at the correct time, or even blending frames together

PROPERTY	FUNCTION
Resample Mode	POINT selects the best buffered frame texture to display. BILINEAR allows blending between the two best frames

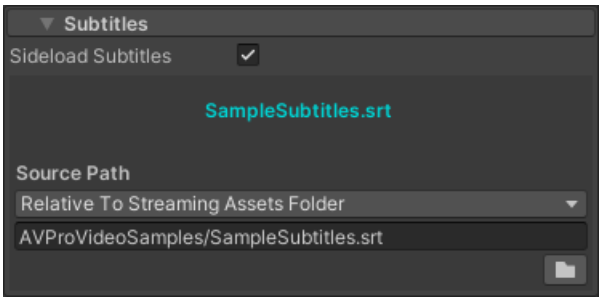
Network



Properties

PROPERTY	FUNCTION
Custom HTTP Headers	Set multiple custom HTTP headers by specifying the header key and value pair.
Auth Token	The authorisation token to pass to the key server for HLS AES-128 decryption.
Key Override (Base64)	The authorisation key to use for HLS AES-128 decryption. This allows a key to be forced in debugging cases where the key server is offline.

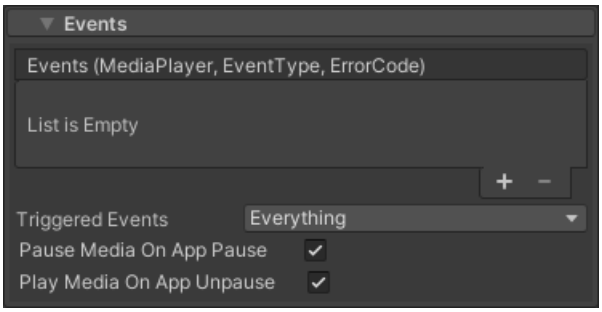
Subtitles



Properties

PROPERTY	FUNCTION
Sideload Subtitles	Enable sideloading of subtitles
Source Path	The location of the subtitle SRT file to sideload

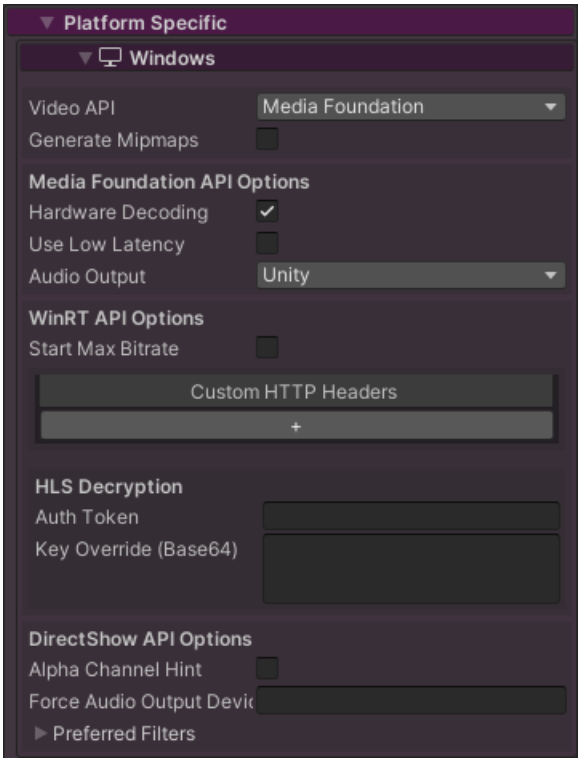
Events



Properties

PROPERTY	FUNCTION
Events	Specify methods to call for triggered events
Triggered Events	A mask to allow events to be disabled for performance reasons
Pause Media On App Pause	Pause media playback when the application is paused (eg switched to the background)
Play Media On App Unpause	Resumed media playback when the application is unpaused (eg switched to foreground)

Platform Specific - Windows



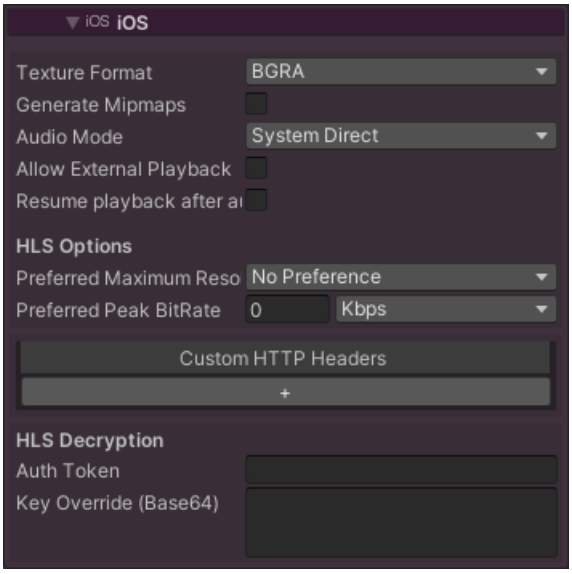
Properties

PROPERTY	FUNCTION
Video API	Select the video API to use. Media Foundation is the default. DirectShow is a legacy API but can be useful for loading many file formats when a codec pack is installed (eg LAV Filters). WinRT is the new Windows API and has better support for adaptive streaming media, but doesn't support many other features (eg Unity audio)
Media Foundation API Options	

PROPERTY	FUNCTION
Hardware Decoding	Enable hardware decoding
Use Low Latency	Enable low latency mode (not recommended as it degrades playback performance)
Use Stereo Detection	Disable for extra performance if stereo packed videos are not required or the stereo packing mode is manually specified
Use Text Tracks	Disable if text track support is not required as this can improve loading time. Enabled by default.
Frame Selection	Experimental feature to allow prebuffering of frames and then logic to decide which frame to display, allowing for smoother playback and easier syncing of multiple videos.
Frame Selection Options	
Pause After Preroll Complete	Whether to pause playback automatically once prerolling has completed. Disabled by default.
Hap / NotchLC Options	
Use Hap/NotchLC	Disable for extra performance if Hap/NotchLC are not required
Use Custom MOV Parser	Enabled our custom MOV parser to be used, which is useful for Hap and NotchLC codecs, as a Microsoft parser is not able to open very high bit-rate MOV files.
Parallel Frame Count	Maximum number of threads to use for parallel frame decoding. Less threads for less latency in playback operations (seeking, playing etc), more threads for better performance.
Preroll Frame Count	Amount of frames to decoder before starting playback, less frames for less latency in seeking, more frames for less chance of buffer emptying too quickly.
Use Facebook Audio 360	Disable if Facebook Audio 360 support is not required as this can improve loading time. Enabled by default.
Audio Output	The audio output mode. System Direct (default): Plays the audio directly to the hardware bypassing Unity. Unity: Sends the audio to Unity for playback via the `AudioOutput` component. Facebook Audio 360: Supports playing MKV files with spatial audio encoded using Facebook Audio 360.
Facebook Audio 360 Options	
Channel Mode	The channel layout to use, usually TBE_8_2 or AMBIX_4
WinRT API Options	
Start Max Bitrate	Forces adaptive streams to begin at the highest bitrate available
Use Low Live Latency	Use the lowest latency possible when playing a live stream
Custom HTTP Headers	Set multiple custom HTTP headers by specifying the header key and value pair.
Auth Token	The authorisation token to pass to the key server for HLS AES-128 decryption.

PROPERTY	FUNCTION
Key Override (Base64)	The authorisation key to use for HLS AES-128 decryption. This allows a key to be forced in debugging cases where the key server is offline.
DirectShow API Options	
Force Audio Output Device	Specify name of the audio output device to use if using the default device is not desired
Preferred Filters	Force named DirectShow filters to be used as first priority. For example, "LAV Video Decoder" could be specified here to prefer it over the Microsoft decoders. Use "GDCL-MPEG4" to force internal MP4 demuxer which can be needed using `OpenMediaFromBuffer()` as some codec packs (eg StarCodec64) cause problems opening MP4 files into a buffer.

Platform Specific - iOS / macOS / tvOS

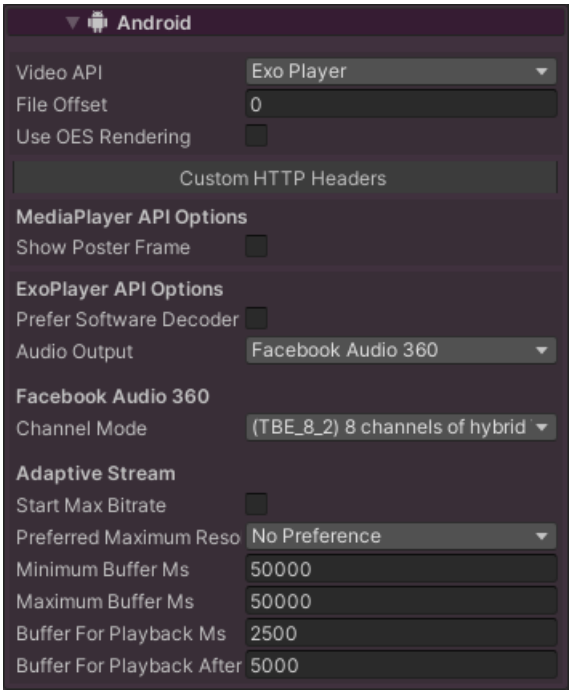


Properties

PROPERTY	FUNCTION
Texture Format	BGRA is the default. YCbCr420 can be specified to save memory and potentially improve performance. With YCbCr420 the AVPro Video shaders are required for display, unless the texture resolve option is used.
Generate Mipmaps	Enable generation of texture mipmaps which is useful to improve filtering quality when the video texture is scaled down on screen.
Audio Mode	**System Direct**: Audio is played directly by the device bypassing Unity. This is the default mode. **Unity**: Audio is played by Unity via the AudioOutput component. HLS media is not supported and will play as if in system direct mode. **System Direct With Capture**: Audio is played directly by the hardware bypassing Unity with PCM data being available via `MediaPlayer.Control.GrabAudio()`. HLS media is not supported.
Allow External Playback	Enable playback on external devices via Airplay
Resume Playback After Audio Session Route Change	The default behaviour is for playback to pause when the audio route changes, for instance when disconnecting headphones.
Max Playback Rate	Set the maximum playback rate that you expect to use

NETWORK	
Preferred Maximum Resolution	Limits the maximum resolution the video will playback at (HLS only).
Preferred Peak Bitrate	Puts an upper limit on the network bandwidth used for playback. Use 0 for no limit. Defaults to 0.
Preferred Forward Buffer Duration	The preferred duration in seconds to buffer in advance of the playhead position in order to minimise stalls. Use 0 to let the system decide based on the current network conditions. Defaults to 0.
Play Without Buffering	Reduces latency when starting playback from a network source at the risk of an increased chance of playback stalling
HLS DECRYPTION	
Auth Token	The authorisation token to pass to the key server for HLS AES-128 decryption.
Key Override (Base64)	The authorisation key to use for HLS AES-128 decryption. This allows a key to be forced in debugging cases where the key server is offline.
CUSTOM HTTP HEADERS	
Set multiple custom HTTP headers by specifying the header key and value pair.	

Platform Specific - Android

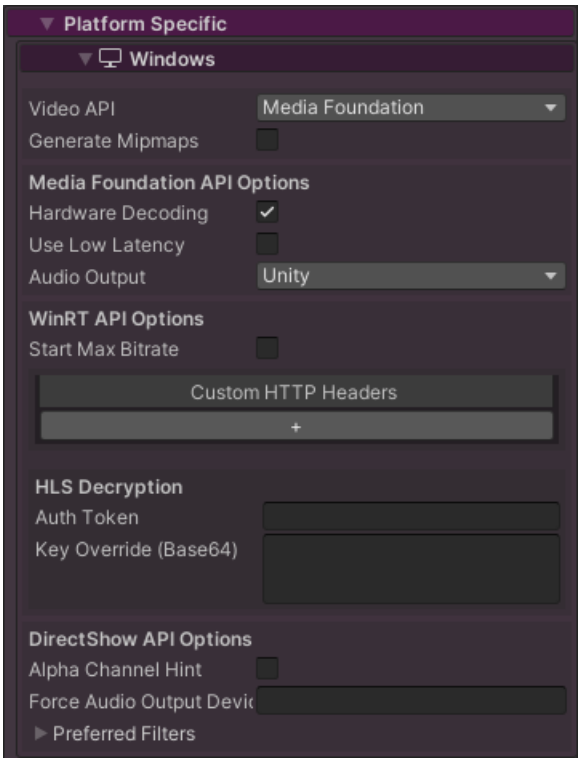


Properties

PROPERTY	FUNCTION
Video API	Select the media API to use. ExoPlayer is the default and is the most flexible option. MediaPlayer uses the built-in Android API.
Use OES Rendering	Enables the OES rendering optimisation. This saves memory and improves performance and is most useful for very high resolution video playback. See Android platform notes for more details.

PROPERTY	FUNCTION
Custom HTTP Headers	Set multiple custom HTTP headers by specifying the header key and value pair.
MediaPlayer API Options	
Show Poster Frame	If the video is not set to auto play, then when it is loaded this option will force a play and then pause so that the first (poster) frame is rendered.
ExoPlayer API Options	
Prefer Software Decoder	Use the software video decoder when possible. This is mostly for internal debugging and should never be used for production.
Force Rtp TCP	Force using TCP as the default RTP transport.
Audio Output	System Direct (default): Audio is played directly to the hardware, bypassing Unity. Unity: Audio is played by Unity via the `AudioOutput` component. Facebook Audio 360: Supports playing MKV files with spatial audio encoded using Facebook Audio 360.
Facebook Audio 360 Options	
Channel Mode	The channel layout to use, usually TBE_8_2 or AMBIX_4
Audio Latency (ms)	Audio latency to add (-ve will play audio sooner, +ve later)
Adaptive Stream Options	
Start Max Bitrate	Start an adaptive stream (eg HLS) at the highest bit-rate possible
Preferred Maximum Resolution	Specify the maximum resolution to limit bandwidth usage
Preferred Peak Bitrate	For HLS videos, puts an upper limit on the network bandwidth used for playback.
Minimum Buffer Ms	The minimum duration of media that the player will attempt to ensure is buffered at all times, in milliseconds.
Maximum Buffer Ms	The maximum duration of media that the player will attempt to buffer, in milliseconds.
Buffer for Playback Ms	The duration of media that must be buffered for playback to start or resume following a user action such as a seek, in milliseconds.
Buffer for Playback After Rebuffer Ms	The default duration of media that must be buffered for playback to resume after a rebuffer, in millisecond.

Platform Specific - Windows UWP

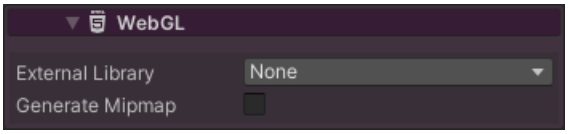


Properties

PROPERTY	FUNCTION
Video API	Select the video API to use. WinRT is the default. WinRT is the new Windows API and has better support for adaptive streaming media, but doesn't support many other features (eg Unity audio) which Media Foundation API does support.
Media Foundation API Options	
Hardware Decoding	Enable hardware decoding
Generate Mipmaps	Generate textures with mip maps
Use Low Latency	Enable low latency mode (not recommended as it degrades playback performance)
Audio Output	The audio output mode. System Direct (default): Plays the audio directly to the hardware bypassing Unity. Unity: Sends the audio to Unity for playback via the `AudioOutput` component. Facebook Audio 360: Supports playing MKV files with spatial audio encoded using Facebook Audio 360.
Facebook Audio 360 Options	
Channel Mode	The channel layout to use, usually TBE_8_2 or AMBIX_4
WinRT API Options	
Start Max Bitrate	Forces adaptive streams to begin at the highest bitrate available
Use Low Live Latency	Use the lowest latency possible when playing a live stream
Custom HTTP Headers	Set multiple custom HTTP headers by specifying the header key and value pair.

PROPERTY	FUNCTION
Auth Token	The authorisation token to pass to the key server for HLS AES-128 decryption.
Key Override (Base64)	The authorisation key to use for HLS AES-128 decryption. This allows a key to be forced in debugging cases where the key server is offline.

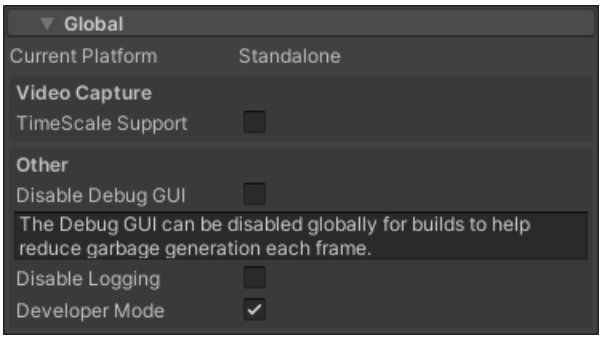
Platform Specific - WebGL



Properties

PROPERTY	FUNCTION
External Library	Whether to use any external JS library for video decoding. This requires that the javascript is included in the HTML. Currently Dash.JS and HLS.JS are supported. A custom library can also be specified and would require editing of the AVProVideo.js file to add support for it. See WebGL streaming information for more details.
Generate Mipmaps	Enable generation of texture mipmaps which is useful to improve filtering quality when the video texture is scaled down on screen.

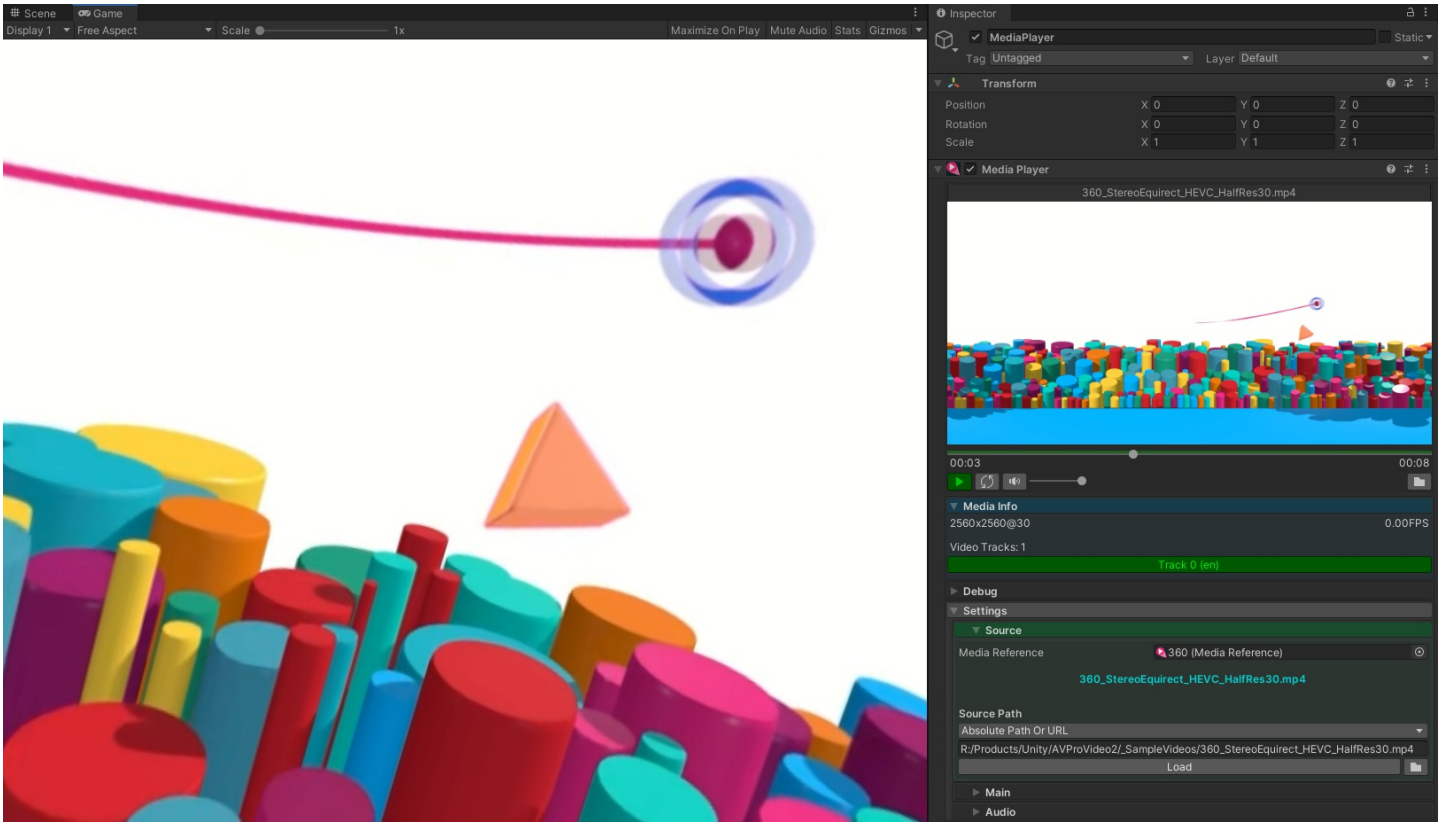
Global



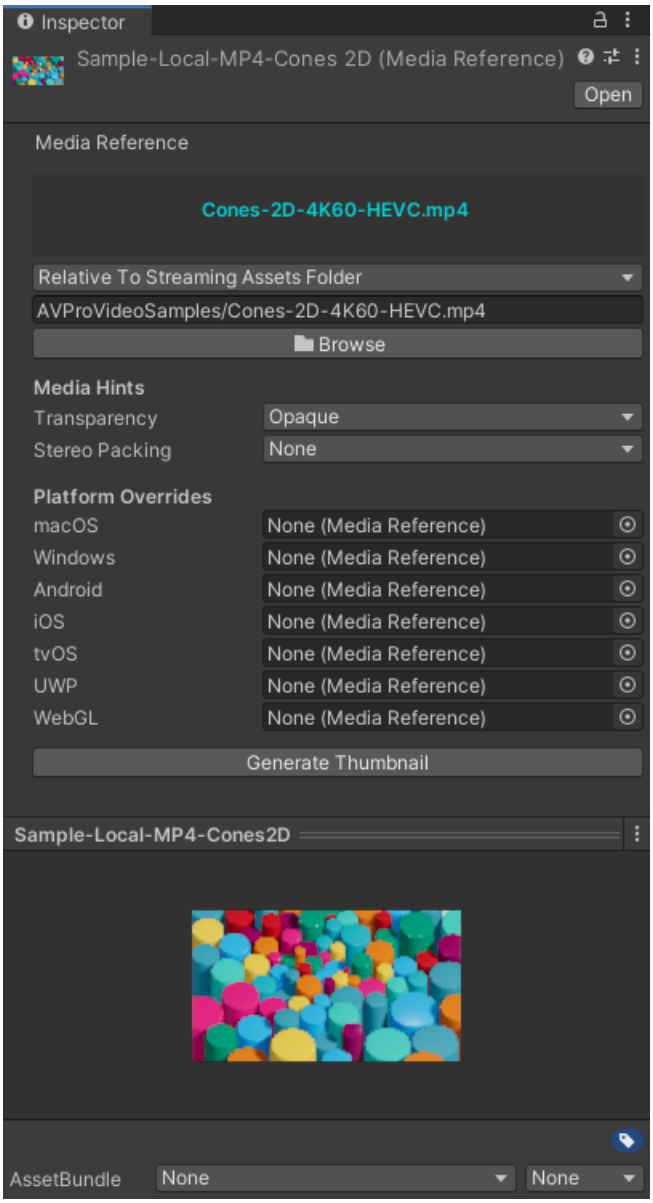
Global per-platform settings. These options need to be set for each platform (via Build Settings > Switch).

Properties

PROPERTY	FUNCTION
TimeScale Support	Enable support for video playback to react to changes in Time.TimeScale. This is mostly useful for offline video capture where capturing is not running in real-time. Audio playback will not work during captures as the video is paused and simply seeks to the desired position.
Disable Debug GUI	Deprecated
Disable Logging	Disables all the "[AVProVideo]" debug logging
Developer Mode	Enables the Developer section on the MediaPlayer inspector view which shows internal state


Scene
Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
Inspector
MediaPlayer
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
Media Player
360_StereoEquirect_HEVC_HalfRes30.mp4
00:03
00:08
Media Info
2560x2560@30
0.00FPS
Video Tracks: 1
Track 0 (en)
Debug
Settings
Source
Media Reference
360 (Media Reference)
360_StereoEquirect_HEVC_HalfRes30.mp4
Source Path
Absolute Path Or URL
R:/Products/Unity/AVProVideo2/_SampleVideos/360_StereoEquirect_HEVC_HalfRes30.mp4
Load
Main
Audio

Media Reference



Media Reference is a `ScriptableObject` asset used to define media location and properties. These are used with the MediaPlayer to load new media in a convenient manner. Media Reference is similar to `VideoClip` asset in Unity.

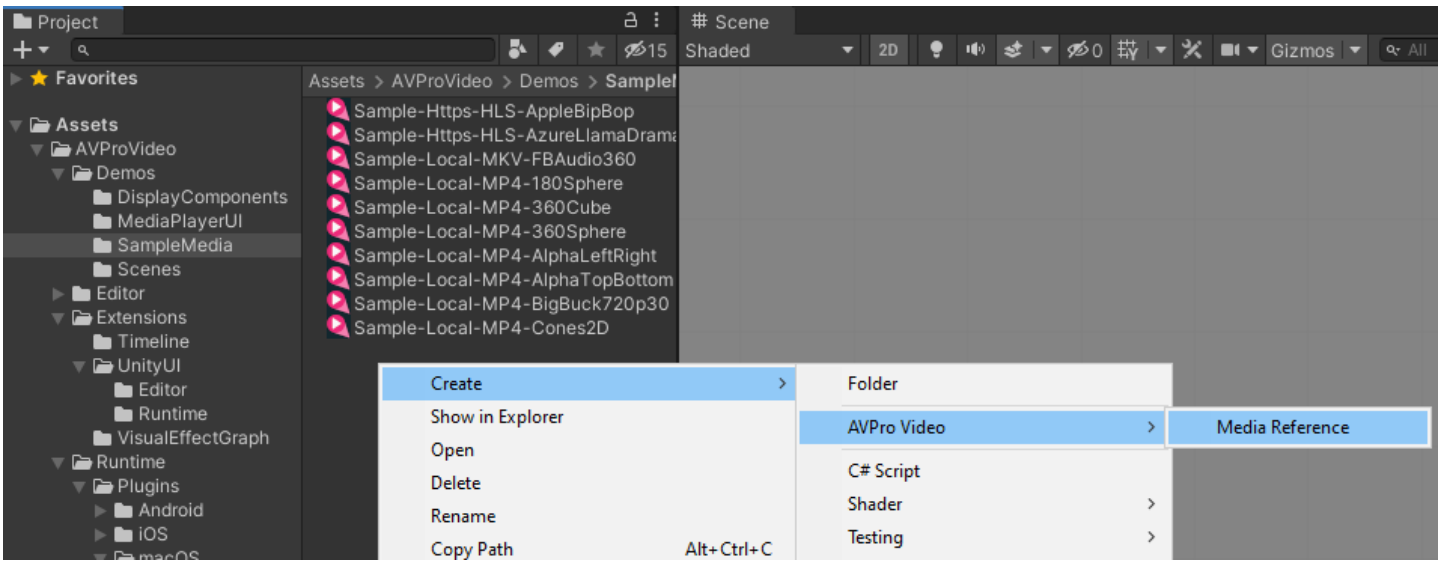
The "Generate Thumbnail" button will generate a previous thumbnail which can make finding media easier. Note these thumbnails are editor only and are not included in builds. The horizontal slider allows setting the time in the video at which to capture the thumbnail. The toggle "Zoom To Fill" allows thumbnail image to fill the entire area of the square thumbnail, regardless of aspect-ratio.

Properties

PROPERTY	FUNCTION
Media Location	The path or URL to the media
Transparency	This hint specified whether the media contains any transparency
Alpha Packing	If the transparency hint is enabled then an optional Alpha Packing hint can be specified.
Stereo Packing	The packing layout for stereo video

PROPERTY	FUNCTION
Platform Override	Allows specifying an alterative MediaReference to load for that platform. If none is specified then the default one specified here is used

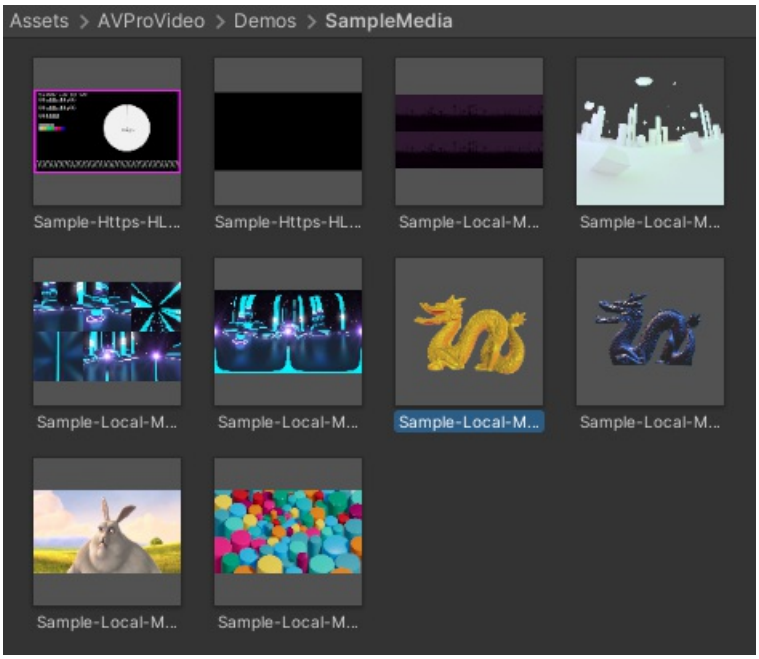
Create



A Media Reference asset can be created in the Project window by right-clicking and selecting:

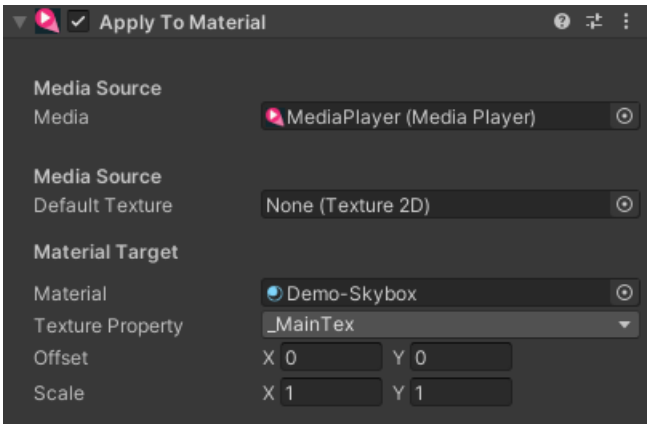
```
Create > AVPro Video > Media Reference
```

Select



Media References display a thumbnail of the video in the Project window and they can be selected from the Browse button on the Media Player.

Apply To Material



Sets up a material to display video from the MediaPlayer component. This material can then be used on 3D meshes or other renderers.

Not only is the texture from the video applied, but also certain material keywords and properties are adjusted depending on the requirements to display the video texture. For example on some platforms the video texture is flipped vertically, or in a different colour space, so the shader is required to support these conversions. The AVPro Video shaders support these conversions and used be used in most cases. If you choose to 'resolve' the textures in the MediaPlayer then these adjustments are already made and so any material can be used.

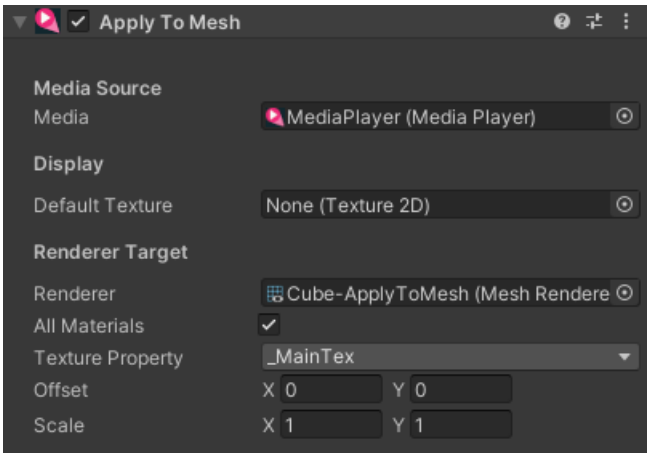
Properties

PROPERTY	FUNCTION
Media	The MediaPlayer component to apply to the material
Default Texture	The texture to display during times when there is no video texture to display (eg during video loading)
Material	The target Material to apply the video texture to
Texture Property	The name of the Material texture to set. The default texture property depends on the render pipeline used. Standard shaders use _MainTex. URP shaders use _Basemap. HDRP shaders use _BaseColorMap.
Offset	The normalised X, Y offset to apply to the texture (if shader supports it)
Scale	The normalised X, Y scale to apply to the texture (if shader supports it)

TIP

This component can be used to render a video to a material which can then be assigned to the `Skybox` component.

Apply To Mesh



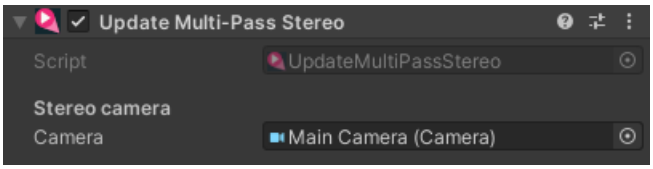
Sets up the materials used by a Renderer (eg MeshRenderer) to display video from the MediaPlayer component.

Not only is the texture from the video applied, but also certain material keywords and properties are adjusted depending on the requirements to display the video texture. For example on some platforms the video texture is flipped vertically, or in a different colour space, so the shader is required to support these conversions. The AVPro Video shaders support these conversions and used be used in most cases. If you choose to 'resolve' the textures in the MediaPlayer then these adjustments are already made and so any material can be used.

Properties

PROPERTY	FUNCTION
Media	The MediaPlayer component to apply to the mesh
Default Texture	The texture to display during times when there is no video texture to display (eg during video loading)
Renderer	The target Renderer (eg MeshRenderer) to apply the video texture to
All Materials	For renderers with multiple materials, either apply to all materials or a specific one
Material Index	Assign to a specific material index
Texture Property	The name of the Material texture to set. The default texture property depends on the render pipeline used. Standard shaders use _MainTex. URP shaders use _Basemap. HDRP shaders use _BasecolorMap.
Offset	The normalised X, Y offset to apply to the texture (if shader supports it)
Scale	The normalised X, Y scale to apply to the texture (if shader supports it)

Update Multi-Pass Stereo



This component is used for stereo rendering VR devices to update AVPro Video shaders so that stereo videos have their left/right frame rendered to the correct eye.

Usually this component isn't required as Unity internally sets up the correct shader variables, however there are some situations where this doesn't happen and this component must be used.

Situations that require this component:

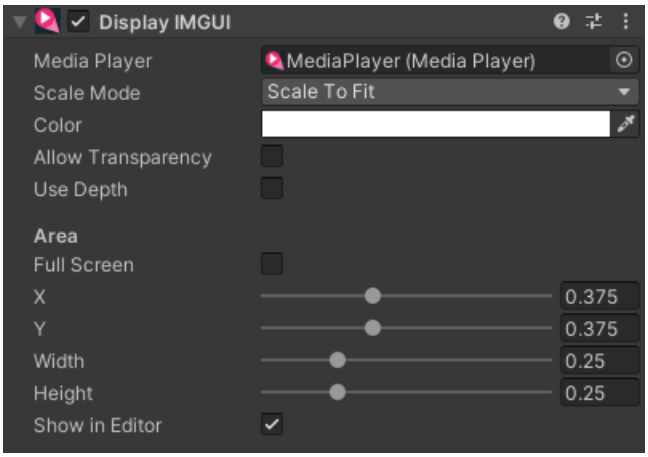
- On some devices when multi-pass stereo rendering is used (e.g. on Pico G2 4K)
- When OES video rendering mode (Android only) is used with multi-pass stereo rendering

> **NOTE**
>
> When using URP/HDRP this component must be on the same GameObject as the camera specified as the view head position.

Properties

PROPERTY	FUNCTION
Camera	The camera representing the viewers head position. If you are using a camera per-eye, then this should still be set to the camera between the two eyes.

Display IMGUI

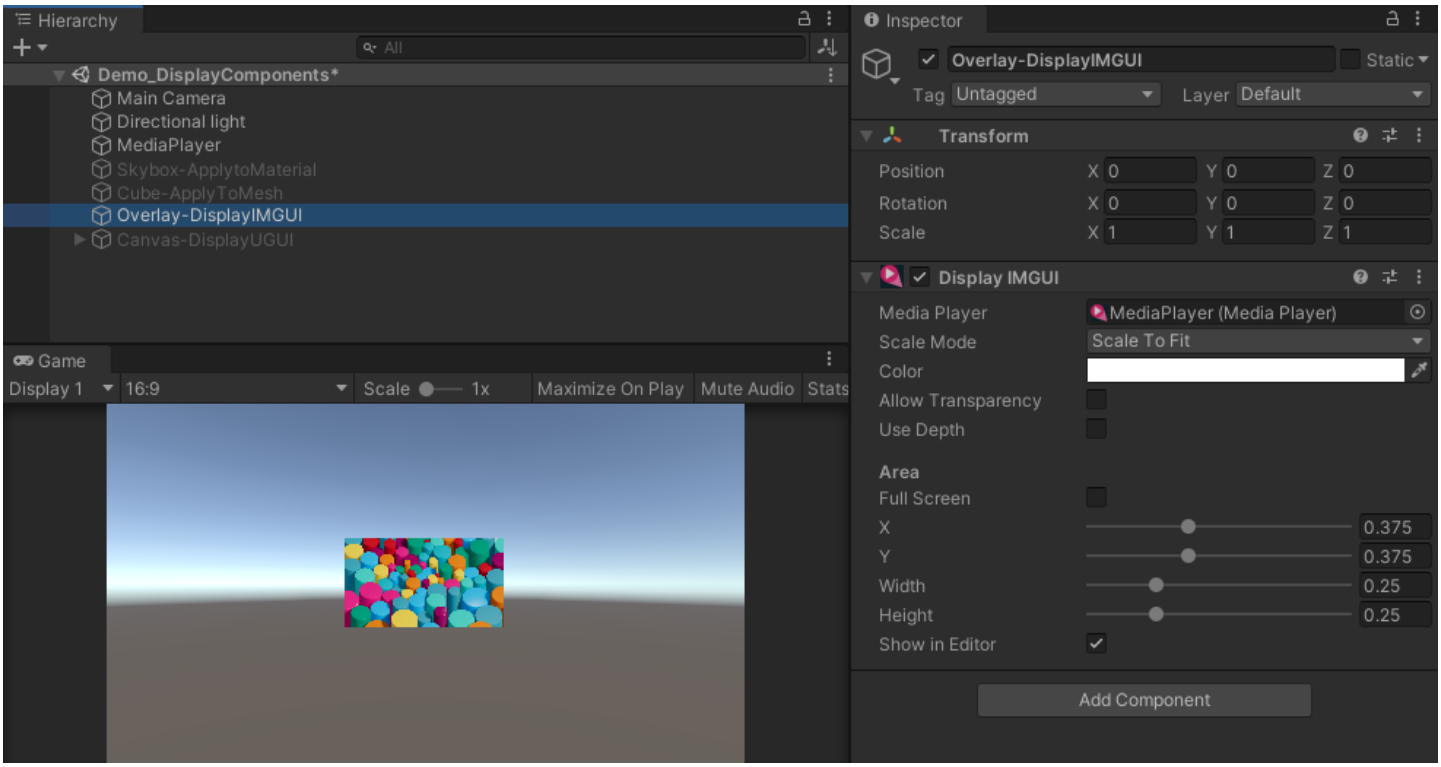


Displays a video on screen from the MediaPlayer component using the legacy IMGUI rendering system. IMGUI is always the rendered last and an top of all other rendering. IMGUI is not supported in VR/AR headsets.

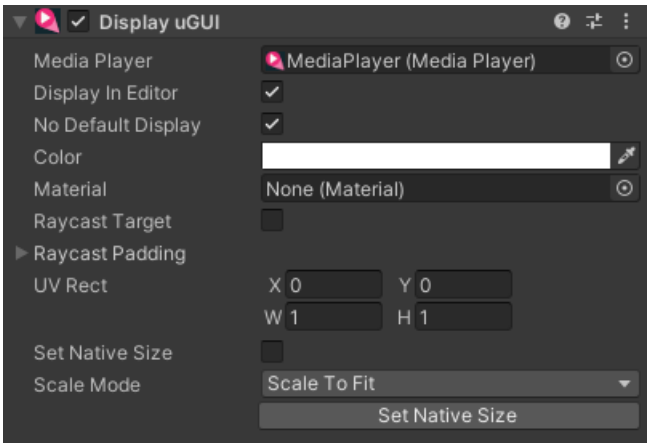
This is the easiest and quickest way to render video to the screen.

Properties

PROPERTY	FUNCTION
Media Player	The MediaPlayer component to display
Scale Mode	Aspect ratio fitting mode to use
Color	The color to multiply the video by. Useful for fading to black or fading to transparent
Allow Transparency	Performance option (disabled is faster) to allow transparent rendering for videos with transparency
Use Depth	Performance option (disabled is faster) to allow specifying the IMGUI depth to value
Depth	Depth value the affects the render order when used with other IMGUI components
Full Screen	Whether to display across the entire screen or allow a rectangle area to be specified
X	Normalised X position for the top-left corner of the video rectangle
Y	Normalised Y position for the top-left corner of the video rectangle
Width	Normalised width of the video rectangle
Height	Normalised height of the video rectangle
Show in Editor	Display a texture in the editor so that rectangle area can be visualised


Hierarchy
All
Demo_DisplayComponents*
Main Camera
Directional light
MediaPlayer
Skybox-ApplytoMaterial
Cube-ApplyToMesh
Overlay-DisplayIMGUI
Canvas-DisplayUGUI
Game
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Inspector
Overlay-DisplayIMGUI
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
X 0
Y 0
Z 0
X 1
Y 1
Z 1
Display IMGUI
Media Player
MediaPlayer (Media Player)
Scale Mode
Scale To Fit
Color
Allow Transparency
Use Depth
Area
Full Screen
X
Y
Width
Height
0.375
0.375
0.25
0.25
Show in Editor
Add Component

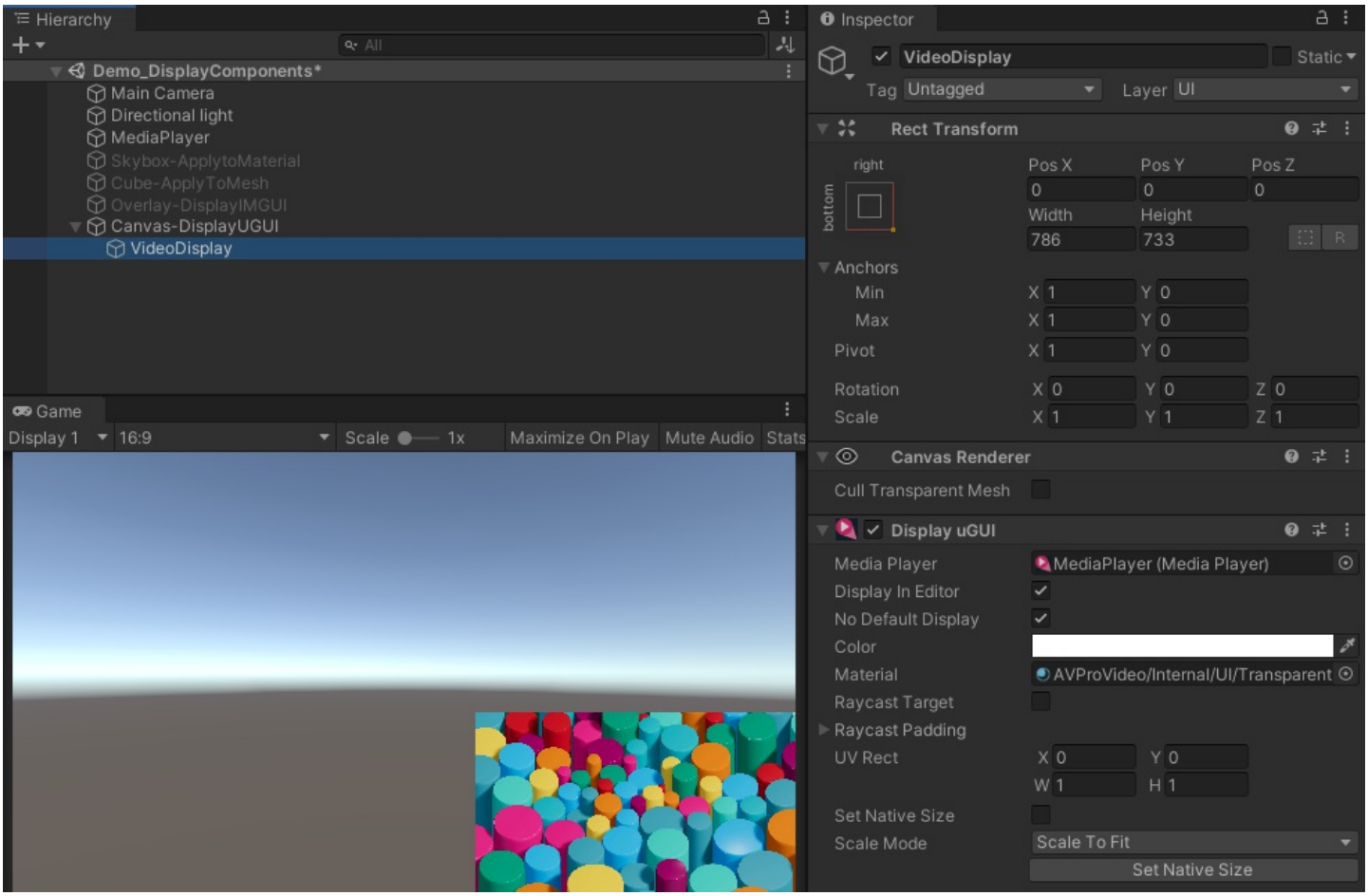
Display UGUI



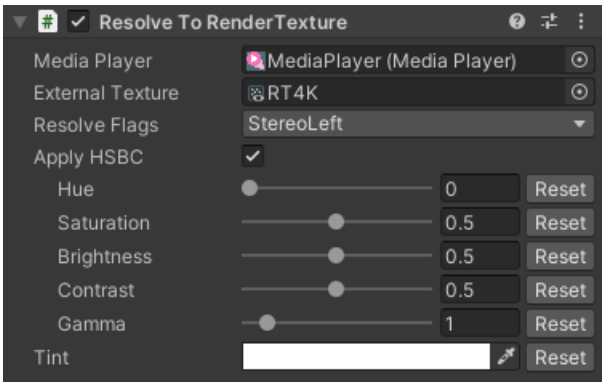
Displays a video on screen from the MediaPlayer component using the standard Unity UI system (also called uGUI).

Properties

PROPERTY	FUNCTION
Media Player	The MediaPlayer component to display
Display in Editor	Display a texture in the editor so that rectangle area can be visualised
No Default Display	Whether or not to display the default texture when no texture is being generated by the MediaPlayer (eg when loading)
Default Texture	The texture to display when no texture is being generated by the MediaPlayer (eg when loading)
Color	The color to multiply the video by. Useful for fading to black or fading to transparent
Material	The material to use for rendering. We recommend this is left empty as generally videos require our own AVPro Video shaders to be assigned at runtime to render the video correctly. When using custom materials, the materials shader needs to support the AVPro Video shader keywords, or the MediaPlayer needs to be set to resolve the textures.
Raycast Target	Whether this element is hitable by the interaction raycaster
UV Rect	Scaling and offset to apply to the texture
Set Native Size	Adjust the size of the RectTransform to match the resolution of the video texture
Scale Mode	Aspect ratio fitting mode to use


Hierarchy
All
Demo_DisplayComponents*
Main Camera
Directional light
MediaPlayer
Skybox-ApplytoMaterial
Cube-ApplyToMesh
Overlay-DisplayIMGUI
Canvas-DisplayUGUI
VideoDisplay
Game
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Inspector
VideoDisplay
Static
Tag
Untagged
Layer
UI
Rect Transform
right
bottom
Pos X
Pos Y
Pos Z
0
0
0
Width
Height
786
733
Anchors
Min
X 1
Y 0
Max
X 1
Y 0
Pivot
X 1
Y 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Canvas Renderer
Cull Transparent Mesh
Display uGUI
Media Player
MediaPlayer (Media Player)
Display In Editor
No Default Display
Color
Material
AVProVideo/Internal/UI/Transparent
Raycast Target
Raycast Padding
UV Rect
X 0
Y 0
W 1
H 1
Set Native Size
Scale Mode
Scale To Fit
Set Native Size

Resolve To RenderTexture

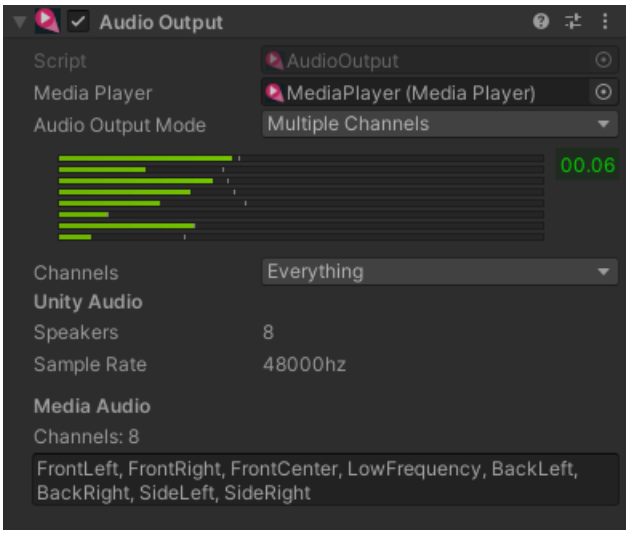


Renders a video from the MediaPlayer component to a Unity `RenderTexture`, optionally adjusting the colours and resolving the texture.

Properties

PROPERTY	FUNCTION
Media Player	The MediaPlayer component to display
Resolve Flags	Optional flags for how to resolve the texture. Stereo eye can be selected, mip-maps can be generated, alpha can be unpacked.
External Texture	Optional RenderTexture to render into. This will cause an extra texture copy to occur. This can also be set using the `ExternalTexture` API property. If no external texture is set then the component generates a RenderTexture the same dimensions as the source video and this is available via the API property `TargetTexture`.
Apply HSBC	Whether to apply HSBC colour adjustments (Hue, Saturation, Brightness, Contast, Gamma)
Hue	Hue adjustment
Saturation	Saturation adjustment
Brightness	Brightness adjustment
Contrast	Contrast adjustment
Gamma	Gamma adjustment
Tint	Tint the colour and transparency

Audio Output



This component is required when the audio output mode is set to Unity. It handles retrieving the audio data from the plugin via `OnAudioFilterRead()` and playing it in Unity through an `AudioSource` component. The number of audio channels that are processed will depend on the Audio settings in Unity and the number of channels in the media. For best performance and latency the sample rate of your audio should match that of Unity.

Properties

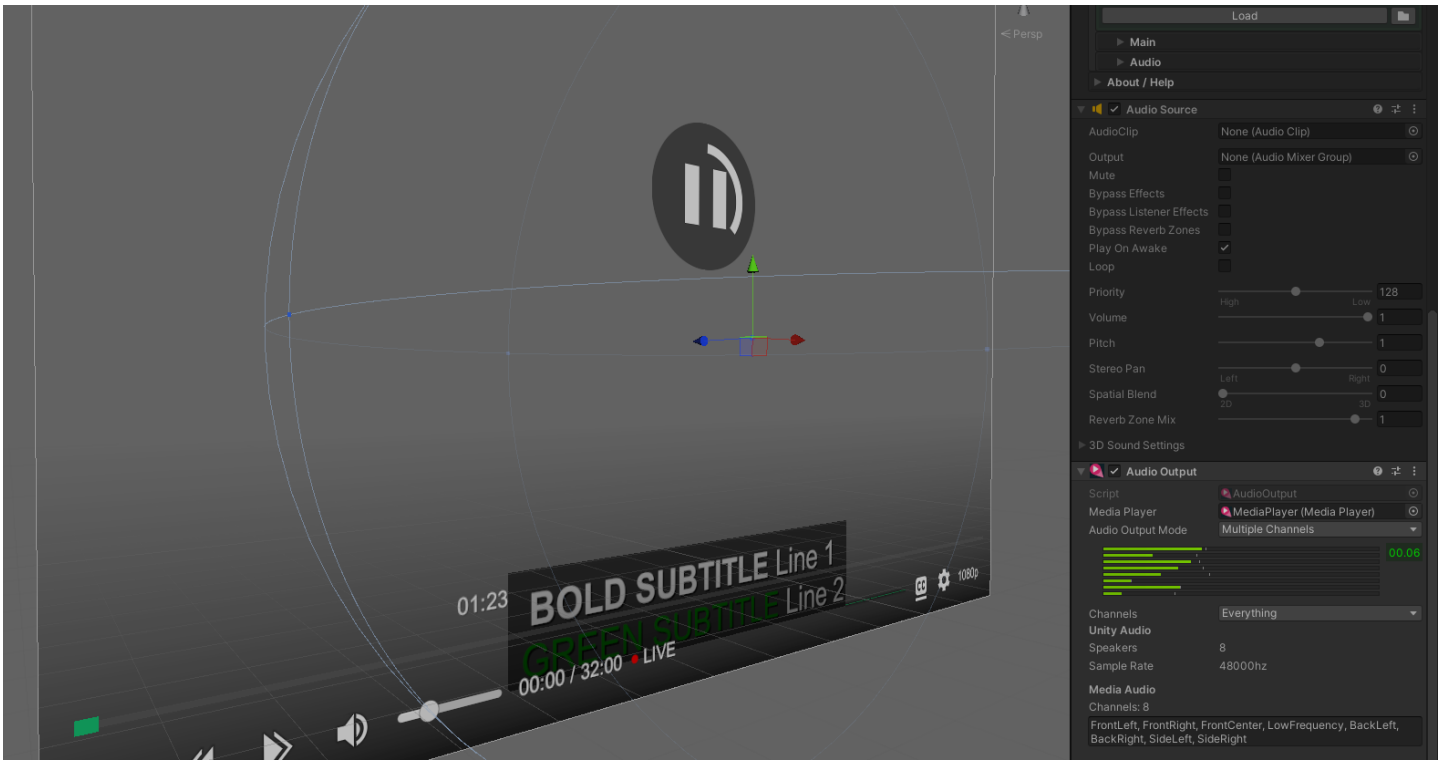
PROPERTY	FUNCTION
MediaPlayer	The MediaPlayer component to retrieve audio from
AudioOutputMode	Selects the mode for how audio channels are rendered. OneToAllChannels: will take a single audio channel and copy it to all the output audio channels. This can be useful when used with the AudioChannelMixer to pan audio across speakers. MultipleChannels: Allows selection of which channels to play back (default is all).
Support Positional Audio	When the `AudioSource` has the `Spatial Blend` set then the position of the audio Transform relative to the `AudioListener` transform is used to attenuate the audio based on distance. This can also apply the doppler effect as the audio moves relative to the listener.

TIP

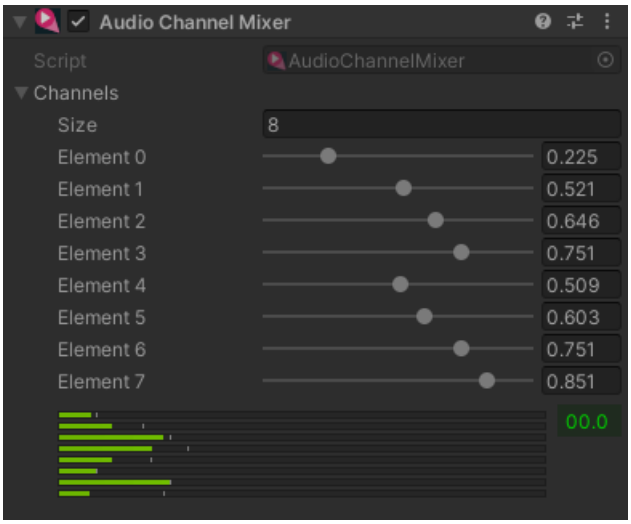
You can retrieve the `AudioSource` component that is being used by the MediaPlayer by using the `AudioSource` property in the MediaPlayer. From the AudioSource you can use methods such as `GetSpectrumData()` to create audio visualisations.

WARNING

If you use the option to support positional audio then reading back the audio via methods such as `GetSpectrumData()` will not work


Persp
Load
Main
Audio
About / Help
Audio Source
AudioClip
None (Audio Clip)
Output
None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop
Priority
128
High
Low
Volume
1
Pitch
1
Stereo Pan
0
Left
Right
Spatial Blend
0
2D
3D
Reverb Zone Mix
1
3D Sound Settings
Audio Output
Script
AudioOutput
Media Player
MediaPlayer (Media Player)
Audio Output Mode
Multiple Channels
00.06
Channels
Everything
Unity Audio
Speakers
8
Sample Rate
48000hz
Media Audio
Channels: 8
FrontLeft, FrontRight, FrontCenter, LowFrequency, BackLeft, BackRight, SideLeft, SideRight
01:23
BOLD SUBTITLE Line 1
GREEN SUBTITLE Line 2
00:00 / 32:00
LIVE
1080p

Audio Channel Mixer



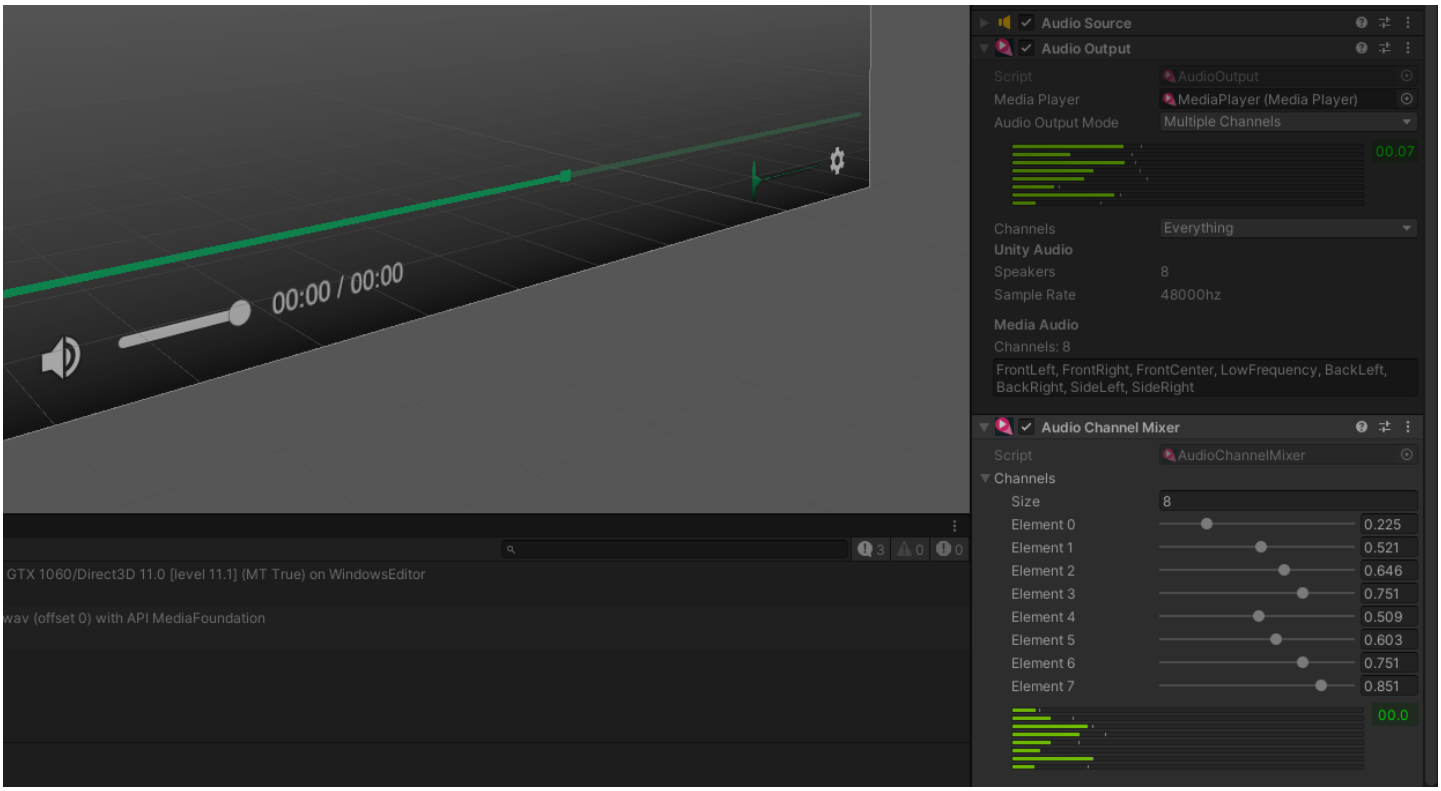
This component allows the volume of each audio channel (up to 8) to be adjusted. Value range from 0.0 to 1.0. This component is used together with the AudioOutput component for audio that is directed to Unity's AudioSource for playback. It must be placed below the AudioSource and AudioOutput components.

Properties

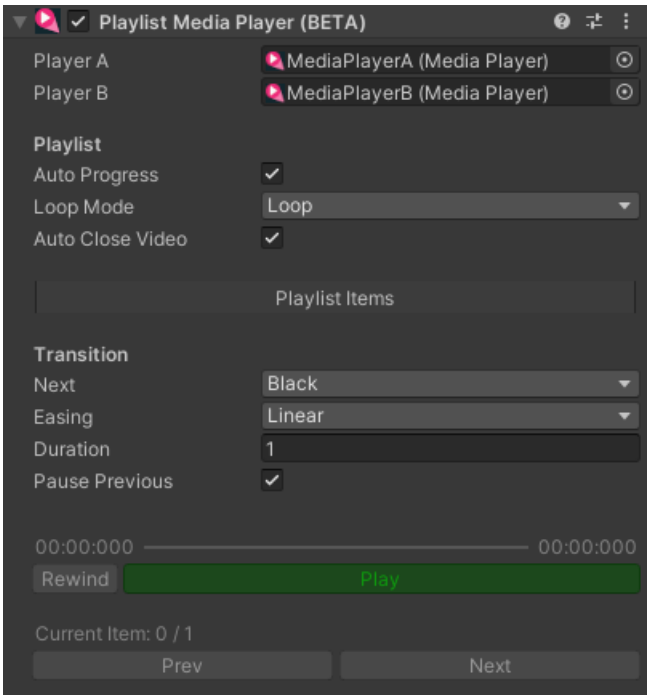
PROPERTY	FUNCTION
Channels	8 values with range 0.0 to 1.0 used to modulate the volume of that audio channel.

TIP

If you have audio where all 8 channels are the same audio then by fading down the appropriate channels you can move that sound between the 8 speakers. If you have the 8 speakers set up in a physical space in a row, then you could move the audio in real world space very easily.


00:00 / 00:00
GTX 1060/Direct3D 11.0 [level 11.1] (MT True) on WindowsEditor
wav (offset 0) with API MediaFoundation
Audio Source
Audio Output
Script
AudioOutput
Media Player
MediaPlayer (Media Player)
Audio Output Mode
Multiple Channels
00.07
Channels
Everything
Unity Audio
Speakers
8
Sample Rate
48000hz
Media Audio
Channels: 8
FrontLeft, FrontRight, FrontCenter, LowFrequency, BackLeft, BackRight, SideLeft, SideRight
Audio Channel Mixer
Script
AudioChannelMixer
Channels
Size
8
Element 0
0.225
Element 1
0.521
Element 2
0.646
Element 3
0.751
Element 4
0.509
Element 5
0.603
Element 6
0.751
Element 7
0.851
00.0

Playlist Media Player



The PlaylistMediaPlayer component extends the MediaPlayer by allowing a playlist of multiple media to be played. Optionally visual transitions can be specified between the media items. Audio is also cross-faded when transitioning.

This component uses two MediaPlayer instances, so it is more expensive than simply using a single MediaPlayer.

NOTE

When playing very high resolution videos (eg in VR), this component may not work on mobile systems as they usually have limited hardware video decoders and memory and this component requires two videos to be in memory at once.

Scripting

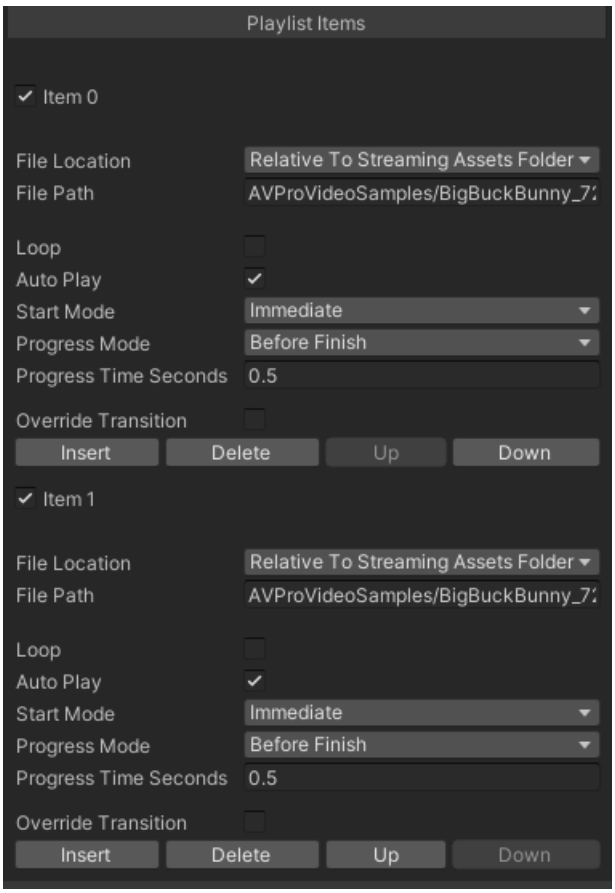
```
PlaylistMediaPlayer pmp;

// Setup
pmp.LoopMode = PlaylistMediaPlayer.PlaylistLoopMode.Loop;
pmp.AutoCloseVideo = true;
pmp.AutoProgress = true;
pmp.DefaultTransition = PlaylistMediaPlayer.Transition.Zoom;
pmp.DefaultTransitionDuration = 2.0f;
pmp.DefaultTransitionEasing = PlaylistMediaPlayer.Easing.Preset.InOutCubic;

// Query the playlist
int playlistSize = pmp.Playlist.Items.Count();
MediaPlaylist.MediaItem currentItem = pmp.PlaylistItem;
if (currentItem != null)
{
    Debug.Log("Current media is at " + currentItem.mediaPath.GetResolvedFullPath());
}
int playlistItemIndex = pmp.PlaylistIndex;

// Build the playlist
pmp.PlaylistItems.Clear();
MediaPlaylist.MediaItem item = new MediaPlaylist.MediaItem();
item.mediaPath = new MediaPath("myvideo.mp4", MediaPathType.RelativeToStreamingAssetsFolder);
item.progressMode = PlaylistMediaPlayer.ProgressMode.OnFinish;
item.isOverrideTransition = true;
item.overrideTransition = PlaylistMediaPlayer.Transition.Black;
item.overrideTransitionDuration = 1.0f;
item.overrideTransitionEasing = PlaylistMediaPlayer.Easing.Preset.Linear;
pmp.Playlist.Items.Add(item);

// Manipulate volume
pmp.AudioVolume = 0.5;
pmp.AudioMuted = true;

// Control playback
pmp.Pause();
pmp.Play();
if (pmp.CanJumpToItem(2))
{
    pmp.JumpToItem(2);
}
if (!pmp.NextItem())
{
    Debug.Log("Can't change item");
}
if (!pmp.PrevItem())
{
    Debug.Log("Can't change item");
}
```

Properties

PROPERTY	FUNCTION
Player A	The first MediaPlayer component
Player B	The second MediaPlayer component
Auto Progress	Enable the playlist to progress to the next item automatically, or wait for manual trigger via scripting
Loop Mode	None: Do not loop the playlist when the end is reached. Loop: Rewind the playlist and play again when the each is reached

PROPERTY	FUNCTION
Auto Close Video	Closes videos that aren't playing. This will save memory but adds extra overhead
Next	The transition to use when progressing to the next video
Easing	The type of easing to use for the transition
Duration	The duration of the transition in seconds
Pause Previous	Causes the previously playing video to pause at the end of the transition which improves performance

Playlist Items



Each item in the playlist needs to be specified.

PROPERTY	FUNCTION
File Location	Location of the file
File Path	Path or URL of the file
Loop	Loop the playback
Start Mode	Immediate (default): start playback immediately Manual: require API or GUI trigger to start playback

PROPERTY	FUNCTION
Progress Mode	On Finish (default): the playlist will progress when the media reaches the end Before Finish: the playlist will progress at some interval before the end of the media is reached. This is useful when cross-fading Manual: the playlist will only progress by the API
Progress Time Seconds	When Progress Mode is set to Before Finish this duration in seconds is used to specify how long before the end of the media duration the playlist will progress to the next item.
Override Transition	Enables a custom transition to be specified for this media item
Transition	The custom transition to use for this media item
Duration	The duration in seconds for the custom transition
Easing	The easing type for the custom transition

Subtitles UGUI

Assigns the current text cue from subtitles active on the MediaPlayer to a uGUI `Text` component.

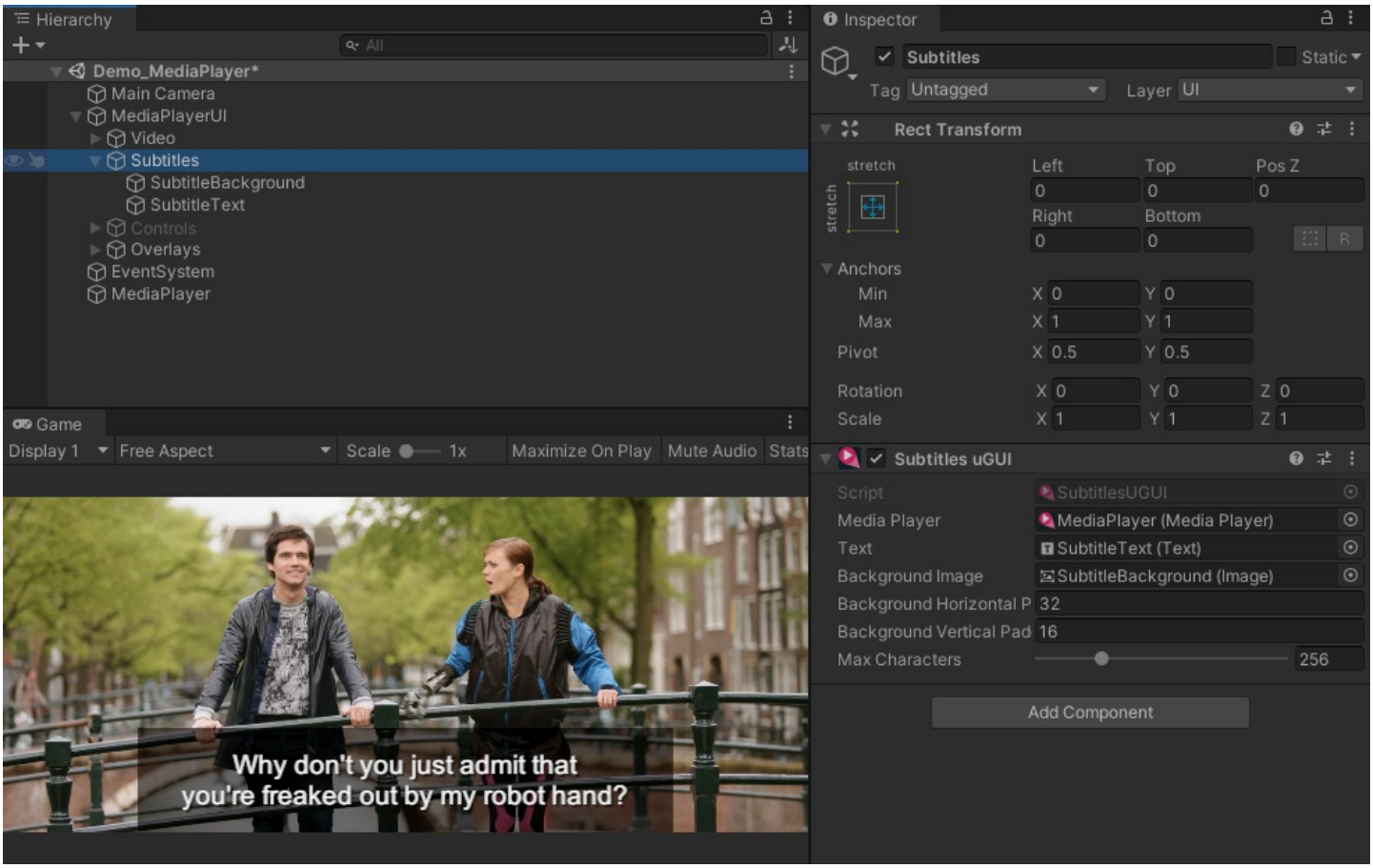
An optional background `Image` component can be adjusted to fit the size of the text with some padding.

Properties

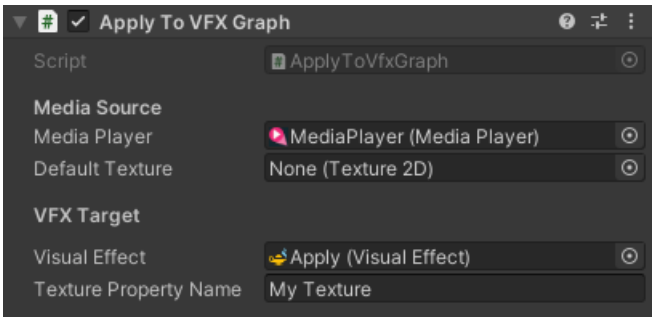
PROPERTY	FUNCTION
Media Player	The MediaPlayer component used to retrieve the subtitle text cues
Text	The Text component to update with subtitle text
Background Image	An optional Image component which is resized to fit the text
Background Horizontal Padding	Number of pixels to use to pad the background image horizontally
Background Vertical Padding	Number of pixels to use to pad the background image vertically
Max Characters	The maximum number of characters to display. Text strings longer than this will be truncated. Useful to prevent unexpected very long text displaying.

TIP

A version of this component for TextMeshPro could easily be made


Hierarchy
All
Demo_MediaPlayer*
Main Camera
MediaPlayerUI
Video
Subtitles
SubtitleBackground
SubtitleText
Controls
Overlays
EventSystem
MediaPlayer
Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Why don't you just admit that
you're freaked out by my robot hand?
Inspector
Subtitles
Static
Tag
Untagged
Layer
UI
Rect Transform
stretch
Left
Top
Pos Z
Right
Bottom
Anchors
Min
Max
Pivot
Rotation
Scale
Subtitles uGUI
Script
SubtitlesUGUI
Media Player
MediaPlayer (Media Player)
Text
SubtitleText (Text)
Background Image
SubtitleBackground (Image)
Background Horizontal P
32
Background Vertical Pad
16
Max Characters
256
Add Component

Apply To VFX Graph



This component allows for a MediaPlayer video to be used as input to a Visual Effect Graph (requires the `Visual Effect Graph` package).

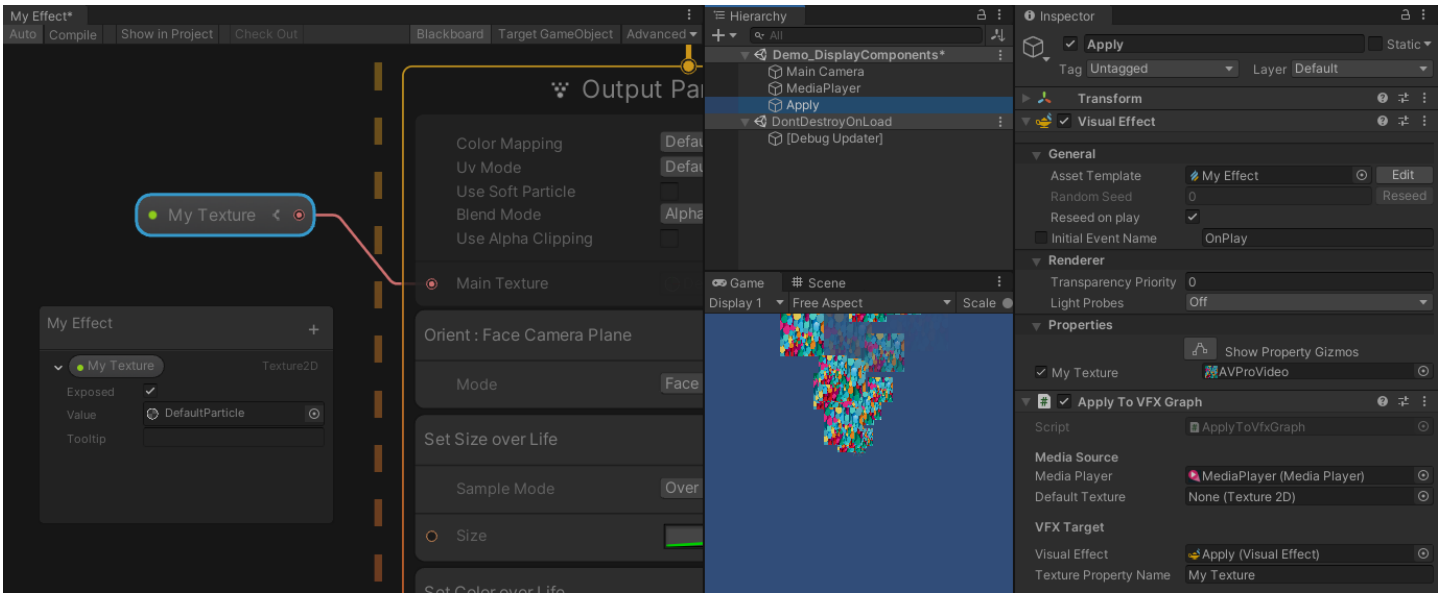
Properties

PROPERTY	FUNCTION
Media Player	The MediaPlayer component to apply to the material
Default Texture	The texture to display during times when there is no video texture to display (eg during video loading)
Visual Effect	The `VisualEffect` component to apply the video texture to
Texture Property Name	The name of the exposed texture variable to set.

Installing

This is an extension as it requires the optional Visual Effect Graph package from the Package Manager, so it requires some setup the first time you use it:

1. Add the Visual Effect Graph package to your project from the Package Manager.
2. Add the Visual Effect Graph package assembly `Unity.VisualEffectGraph.Runtime` to the `_AVProVideo.Extensions.VisualEffectGraph` assembly definition references.
3. Add `AVPRO_PACKAGE_VFXGRAPH` to your player Preprocessor Defines to enable the scripts to be compiled (you can also edit ApplyToVfxGraph.cs and comment in the `#define AVPRO_PACKAGE_VFXGRAPH` line at the top).

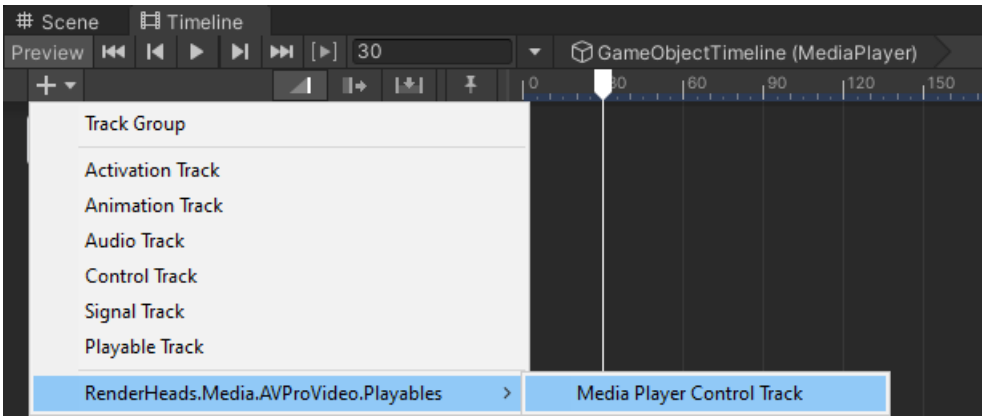


Timeline Playables

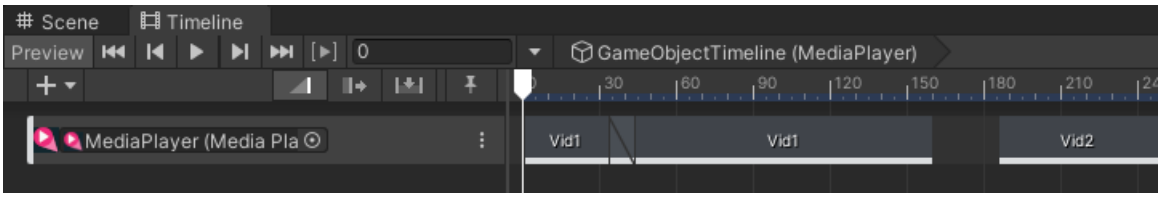
This component allows some basic control over the MediaPlayer component via the Timeline in the `Timeline Package`.

Setup

First the Track is added to the timeline:



Next clips can be added to track by right-clicking on the track and selecting `"Add Media Player Control Asset"`. Clips can overlap to allow some properties to animate. Currently only Audio Volume will animate.



Each clip can then be set up with properties for controlling the MediaPlayer.

Properties

PROPERTY	FUNCTION
Media Reference	The media to load
Audio Volume	0 to 1 range of audio volume
Start Time	The time in seconds to start from when new media is loaded. Negative values are ignored.
Pause On End	Whether to pause playback when the clips ends

Installing

This is an extension as it requires the optional Timeline package from the Package Manager, so it requires some setup the first time you use it:

1. Add the Timeline package to your project from the Package Manager.
2. Add the Timeline package assembly `Unity.Timeline` to the `_AVProVideo.Extensions.Timeline` assembly definition references.
3. Add `AVPRO_PACKAGE_TIMELINE` to your player Preprocessor Defines to enable the scripts to be compiled.

Windows Desktop Platform

Plugin Specs

- Compatibility
  - Unity 2018.x - 2022.x are supported
  - Supported CPU architectures are 32-bit and 64-bit x86
  - Windows 7 - 10 are supported, however some versions of Windows allow for more features than others

- Rendering
  - For rendering we support Direct3D 11, Direct3D 12 (requires minimum Unity 2019.3), and also have limited support for legacy Direct3D 9 and OpenGL.
  - Multi-threaded rendering is supported.

- Internals
  - Under the hood we're using the WinRT, Media Foundation and DirectShow API's. WinRT is supported on Windows 10 and above, Media Foundation supported on Windows 8 and above, and DirectShow is supported from Windows 7.
  - The only 3rd-party libraries used in the Windows Desktop binaries are:
    - Hap https://github.com/Vidvox/hap
    - Google Snappy https://github.com/google/snappy
    - GDCL Mpeg-4 https://github.com/roman380/gdcl.co.uk-mpeg4
    - GLEW http://glew.sourceforge.net/
    - Facebook Audio 360 1.7.12 https://facebookincubator.github.io/facebook-360-spatial-workstation/

Supported Media

See the Supported Media section for more information.

Streaming

See the Streaming section for more information.

Platform-Specific Options

See the Platform-Specific Options section for more information.

Troubleshooting

Windows N / KN editions

- There are some editions of Windows (N and KN) that ship with greatly reduced built-in media playback capabilities.
- It seems like these editions don't include MFPlat.DLL, but do include some basic DirectShow components. This means the Media Foundation playback path will not work.
- These editions of Windows require either a 3rd party codec installed (such as the LAV Filters for DirectShow), or the Microsoft Media Feature Pack:
  - Media Feature Pack for Windows 7 SP1 https://docs.microsoft.com/en-us/troubleshoot/windows-client/shell-experience/windows-media-feature-pack-for-windows-7
  - Media Feature Pack for Windows 8.1 https://www.microsoft.com/en-gb/download/details.aspx?id=40744
  - Media Feature Pack for Windows 10 https://www.microsoft.com/en-gb/download/details.aspx?id=48231

- We found that MJPEG DirectShow codec still works on these editions without installing the Media Feature Pack, so this the best choice for maximum compatibility

Android Platform

Plugin Specs

- Compatibility
  - Unity 2018.x - 2022.x are supported (see below for build notes)
  - Supported CPU architectures are arm-v7a, arm64-v8a, x86 and x86-64
  - This plugin requires a minimum Android API level of 15 when using the MediaPlayer API, and API level 16 when using ExoPlayer (due to its use of MediaCodec).

- Rendering
  - For rendering OpenGL ES 2.0 and OpenGL ES 3.0 are supported, and multi-threaded rendering is supported on both.
  - Vulkan graphics API is not yet supported

- Internals
  - Under the hood we're using the Android MediaPlayer API and ExoPlayer 2.18.0 API
  - The only 3rd-party libraries used are:
    - ExoPlayer 2.19.1 https://github.com/google/ExoPlayer
    - Facebook Audio 360 1.7.12 https://facebookincubator.github.io/facebook-360-spatial-workstation/
    - Secret Rabbit Code (aka libsamplerate) 0.1.9 http://www.mega-nerd.com/SRC/license.html

Supported Media

See the Supported Media section for more information.

Streaming

See the Streaming section for more information.

Platform-Specific Options

See the Platform-Specific Options section for more information.

Build Notes

Gradle must be used as the build environment. This was introduced in Unity 2017, but different versions of Unity use different versions of Gradle (as outlined here: https://docs.unity3d.com/Manual/android-gradle-overview.html). We have found that it is not possible to build an application that incorporates AVPro Video using some of the older versions of Gradle found in many of the 2017/2018 Unity releases. The following list will guide you as to what versions we have observed working, and if there are any special requirements.

```
2017.1               - requires gradle change*
2017.2               - requires gradle change*
2017.3               - requires gradle change*
2017.4 to 2017.4.16  - requires gradle change*
2017.4.17 onwards    - works directly from Unity
2018.1               - requires gradle change*
2018.2               - requires gradle change*
2018.3               - works directly from Unity
2018.4               - works directly from Unity
2019.1               - works directly from Unity
2019.2               - works directly from Unity
2019.3               - works directly from Unity
2019.4               - works directly from Unity
2020.1               - works directly from Unity
2020.2 onwards       - see notes below
```

- old gradle needs a toolchain in older NDK (<= 16b), old NDK does not support Java 1.8 which ExoPlayer requires compilation with.

NOTE

It has been seen that Unity can sometimes want to use a version of the build tools that are not on a given system. To overcome this you have to tell gradle to use a version you have on your system. In `Player Settings|Publisher Settings`, enable the `Custom Gradle Template` option. This will allow you to adjust the buildToolsVersion to one you have on your system. The template is found at: `/Assets/Plugins/Android/mainTemplate.gradle`

Unity 2020.1 and above

When using Unity 2020.1 and above it has been seen that the build process overly optimises which causes issues in the ExoPlayer library. To overcome this issue, you will need to add a custom gradle property. In `Player Settings|Publisher Settings`, enable the `Custom Gradle Properties Template` option. The template is found at: `/Assets/Plugins/Android/gradleTemplate.properties` and it should be edited adding the following line to the end of the file: `android.enableDexingArtifactTransform=false`

NOTE

As of v2.0.9 the addition of this property has been automated in a post process build script.

Unity 2022.1 and above

Exoplayer requires Android SDK 33, which does not come with this version of Unity. In `Player Settings|Other Settings`, set `Target API Level` to `Android 13.0 (API level 33)` and Unity should download the correct SDK when you build. The SDK will be downloaded to `Editor/Data/PlaybackEngines/AndroidPlayer/SDK/platforms/`.

If you need to target a lower version of Android, then you need to add a custom gradle property. In `Player Settings|Publisher Settings`, enable the `Custom Main Gradle Template` and `Custom Launcher Gradle Template` options. The templates are found at: `/Assets/Plugins/Android/mainTemplate.gradle` and `/Assets/Plugins/Android/launcherTemplate.gradle` and they should be edited by adding the following lines:

- minSdkVersion 22
- targetSDKversion 32
- compileSDKversion 33

Proguard Minify

If you use Android minify options, AVPro Video will be stripped out and will not work. The following options need to be added to your proguard configuration file (in `Player Settings > Android > Publishing Settings > Build > Custom Proguard File`)

to prevent AVPro Video classes from being removed:

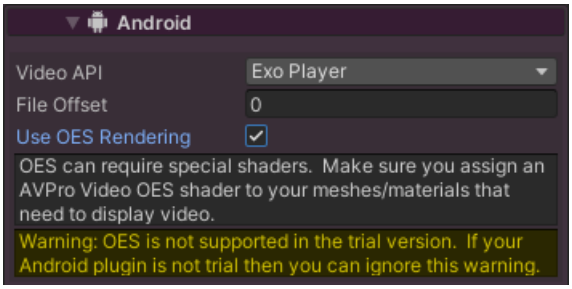
```
-keep class com.renderheads.AVPro.Video.** { *; }
-keep class com.google.android.exoplr2avp.** { *; }
-keep class com.twobigears.audio360.** { *; }
```

The above is included in `/Assets/AVProVideo/Runtime/Plugins/Android/proguard-avprovideo.txt`

OES Playback Path

> **NOTE**
>
> OES playback is not available in the trial version

For Android there is a special playback option called "Use Fast OES Path". This option caters especially for VR where users are trying to get the highest possible frame rate and resolution out of the device (without it overheating at the same time). The option is available in the Platform Specific section of the MediaPlayer component:



The OES path is not enabled by default because it requires some special care to be taken and can be tricky for beginners. When this option is enabled the Android GPU returns special OES textures (see EGL extension OES_EGL_image_external) that are hardware specific. Unfortunately Unity isn't able to use these textures directly, so you can't just map them to a material or UI. To use the texture a GLSL shader must be used. Unfortunately Unity's GLSL support isn't as good as its CG shader support so again this makes things more tricky. The GLSL compiler only happens on the device (not inside Unity) so errors in the shader can be difficult to debug.

We have included a version of the VR sphere shader that supports stereo videos as an example. Hopefully in the future we can improve the integration of these shaders so they aren't such special cases. This playback path is much faster though, so is definitely worth exploring. Note that for VR stereo rendering, OES only currently supports multi-pass rendering path, and not single-pass or single-pass instanced.

Troubleshooting

Streaming

- If you want to support streaming from URL don't forget to set the "Internet Access" option in Player Settings to "require"
- If you're streaming on Android 9 or above and connecting to an HTTP stream, then you either need to switch to HTTPS stream, or enable cleartext support in the AndroidManifest.xml

java.lang.ClassNotFoundException / java.lang.NoSuchMethodError

- Check whether you have Proguard minification enabled (see notes on Minify above) which may be over-optimising and stripping AVPro Video files
- Check the Gradle build notes above

Collecting Logs

- We often need to see the device logs to work out why something isn't working. For this the device should be connected via USB.

- If you're using Android Studio then you can click on the `Logcat` tab and choose `No Filters` in the bar on the top right. You should see logs being produced and can copy-paste all of them into a text file.
- Another useful tool on Windows is called mLogcat and is a GUI tool for monitoring and capturing logs from Android.
- Alternatively the command-line tool "adb logcat" can be used from the Android SDK.

macOS Platform

Plugin Specs

- Compatibility
  - Unity 2018.x - 2022.x are supported
  - macOS Mojave (10.14) and later are supported
  - Only 64-bit (x86_64, arm64) builds are supported
- Rendering
  - Only the Metal rendering API is supported
  - Multi-threaded rendering is supported
- Internals
  - Under the hood we're using Apple's AVFoundation API
  - The only 3rd-party libraries used in the macOS binaries are:
    - HapInAVFoundation https://github.com/Vidvox/hap-in-avfoundation

Supported Media

See the Supported Media section for more information.

Streaming

See the Streaming section for more information.

Platform-Specific Options

See the Platform-Specific Options section for more information.

Troubleshooting

Rendering

- Lack of video output is usually due to the OpenGLCore renderer being selected. Make sure you're using the Metal renderer and have enabled "Metal Editor Support" in the "Other Settings" section of the macOS Player Settings inspector.

Notarising

- We notarise the plugin bundle so you shouldn't have to code-sign it prior to building your package for submission to the App Store.

Plugin fails to load when using macOS versions older than 10.14.4

Significant portions of AVPro Video are now written using Swift. Unfortunately the Swift 5 standard libraries were not added to macOS until 10.14.4 and when running under older version of macOS you will encounter an error similar to the following:

```
dyld: Library not loaded: @rpath/libswiftCore.dylib
```

To be able to use AVPro Video in the Unity editor with older versions of macOS please install the "Swift 5 Runtime Support for Command Line Tools" package available from Apple's developer site here.

If you just want standalone builds to be able to run on older versions of macOS, you need to do the following:

- Make sure you're using Unity 2019 or later
- Make sure the iOS Build Support module is installed
- In the Build Settings dialog, make sure "Create Xcode Project" is checked

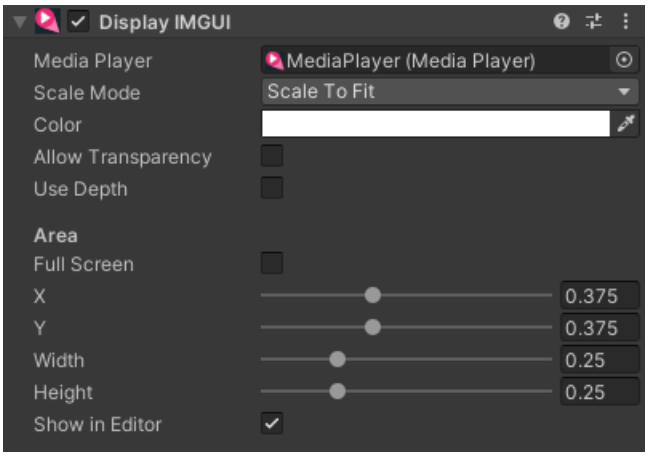
- In the Player Settings inspector:
  - In the Other Settings section
    - Add `AVPROVIDEO_SUPPORT_MACOSX_10_14_3_AND_OLDER` to "Scripting Define Symbols"

- Build the project
- Check that "Always Embed Swift Standard Libraries" is set to "YES" in the main target's build settings.

Unfortunately we don't have a solution for projects built with older versions of Unity.

Y'CbCr Video Output Mode

Apple platforms have support for Y'CbCr textures which has lower memory overheads and is slightly more performant when compared with standard BGRA32 textures. This option is disabled by default and can be enabled on the MediaPlayer in at Platform Specific section:



The DisplayIMGUI and DisplayUGUI components automatically detect the use of Y'CbCr mode and switch to a suitable shader. ApplyToMesh/ApplyToMaterial also detect this setting and try to set up the shader on the material to the correct settings, however it requires the shader to have the correct properties. The AVPro Video shaders support this, so if you want to use this on a mesh then make sure you're using these shaders.

iOS / tvOS Platforms

Plugin Specs

- Compatibility
  - Unity 2018.x - 2022.x are supported
  - iOS 10.0, iPadOS 13.1, tvOS 11.0 and later are supported
- Rendering
  - Only the Metal rendering API is supported
  - Multi-threaded rendering is supported
- Internals
  - Under the hood we're using Apple's AVFoundation API

Supported Media

See the Supported Media section for more information.

Streaming

See the Streaming section for more information.

Platform-Specific Options

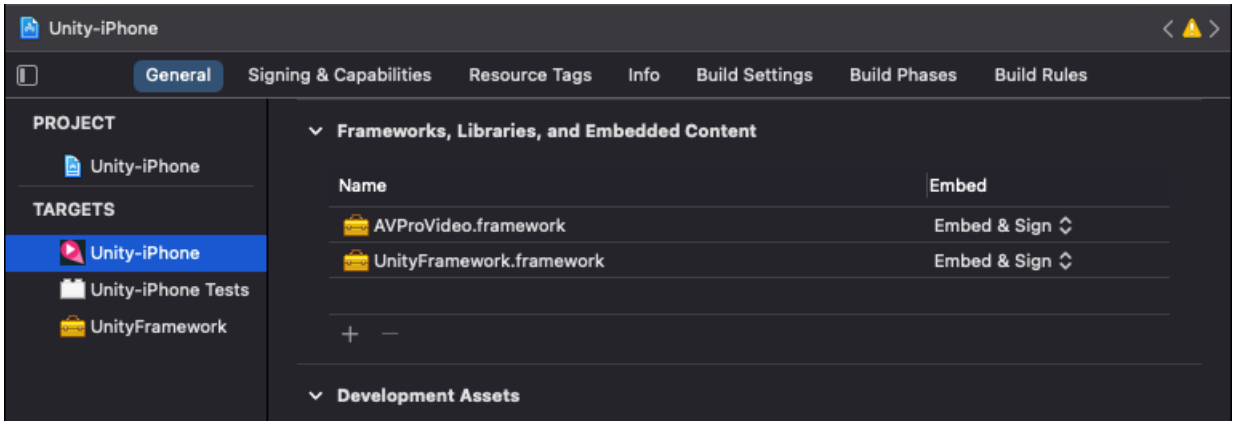
See the Platform-Specific Options section for more information.

Troubleshooting

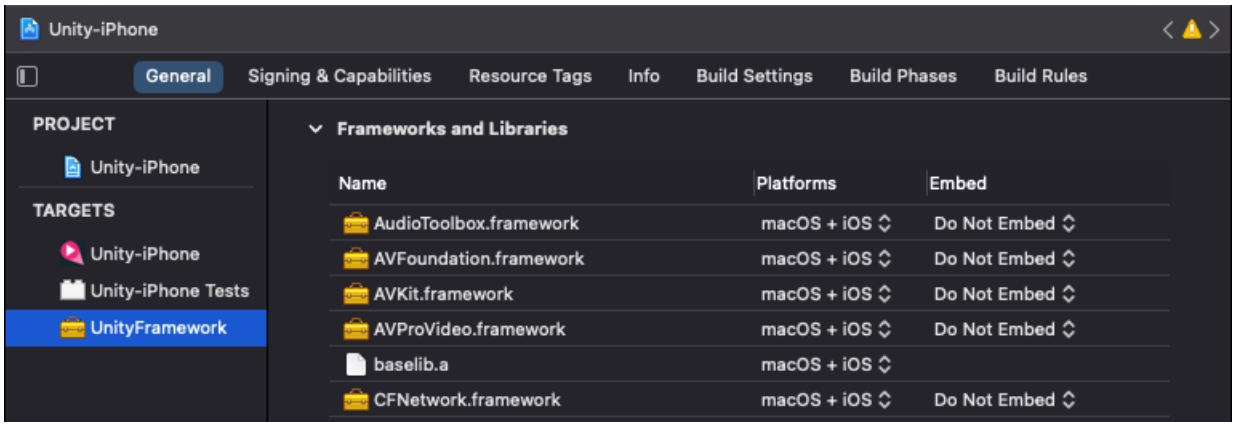
- My app terminates with the following error in the log:

```
dyld: Library not loaded: @rpath/AVProVideo.framework/AVProVideo
```

You need to make sure the **AVProVideo.framework** is set to **Embed & Sign** in the **Frameworks, Libraries and Embedded Content** section of the **General** pane for the **Unity-iPhone** target of the generated Xcode project as shown below:



You will also need to make sure that **AVProVideo.framework** is set to **Do Not Embed** in the **Frameworks, Libraries and Embedded Content** section of the **General** pane for the **UnityFramework** target of the generated Xcode project as shown below:



The PostProcessBuild_iOS script should do this automatically for you. It has been known to fail when:

- building from the command line and not passing the `-buildTarget iOS` option
- the path to the framework cannot be identified. This is usually due to either:
  - The plugin being moved from its default installed location
    Whilst relocating the entire plugin folder is fine, the contents should be left in their respective locations
  - The plugin was installed into a project with version 1.x already present
    Remove version 1.x of the plugin and then reinstall version 2.x

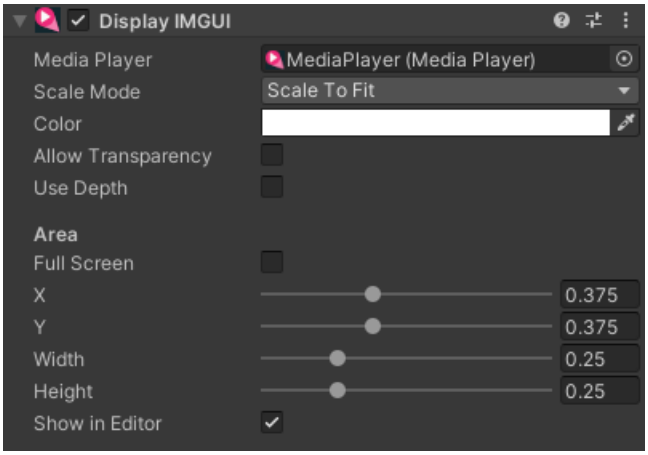
- you are integrating Unity into a native iOS application as outlined here
  - add **AVProVideo.framework** to **Frameworks, Libraries and Embedded Content** section of the **General** pane for your application target and make sure it is set to **Embed & Sign**
  - make sure that **AVProVideo.framework** is set to **Do Not Embed** in the **Frameworks, Libraries and Embedded Content** section of the **General** pane for the **UnityFramework** target

Streaming

- By default streaming requires that the HTTPS protocol is used. If you want to support streaming via HTTP you need to enable this explicitly. Look for the "Allow downloads over HTTP" option in the "Other Settings" pane of the iOS and tvOS player settings inspector.

Y'CbCr Video Output Mode

Apple platforms have support for Y'CbCr textures which has lower memory overheads and is slightly more performant when compared with standard BGRA32 textures. This option is disabled by default and can be enabled on the MediaPlayer in at Platform Specific section:



The DisplayIMGUI and DisplayUGUI components automatically detect the use of Y'CbCr mode and switch to a suitable shader. ApplyToMesh/ApplyToMaterial also detect this setting and try to set up the shader on the material to the correct settings, however it requires the shader to have the correct properties. The AVPro Video shaders support this, so if you want to use this on a mesh then make sure you're using these shaders.

Windows UWP Platform

Plugin Specs

- Compatibility
  - Unity 2018.x - 2022.x are supported
  - Supported CPU architectures are x86, x86_64, ARM and ARM64
- Rendering
  - For rendering we support Direct3D 11 and Direct3D 12 (requires minimum Unity 2019.3)
  - Multi-threaded rendering is supported.
- Internals
  - Under the hood we're using the WinRT and Media Foundation
  - The only 3rd-party libraries used in the UWP binaries (x86 32-bit and 64-bit only, not ARM) are:
    - Facebook Audio 360 1.7.12 https://facebookincubator.github.io/facebook-360-spatial-workstation/

Supported Media

See the Supported Media section for more information.

Streaming

See the Streaming section for more information.

Platform-Specific Options

See the Platform-Specific Options section for more information.

Troubleshooting

Streaming

- For streaming video don't forget to enable the "InternetClient" capability option in Unity's Player Settings.
- If you're streaming video from a local server / LAN then you need to enable the "PrivateNetworkClientServer" option.

Performance

- For best compatibility and performance add

```
appCallbacks.AddCommandLineArg("-force-d3d11-no-singlethreaded");
```

  to your MainPage.xaml.cs/cpp or MainPage.cs/cpp. You should call this before the appCallbacks.Initialize() function.

Certification

- If you use Windows App Certification Kit and you get error messages related to the Audio360.dll then you can simply delete this DLL if you are not using the Facebook Audio 360 feature.

WebGL Platform

> 👉 **IMPORTANT**
>
> **We do not officially support WebGL**, but still include it as it may be useful for some people. We found too many issues with browser compatibility to continue supporting it

Plugin Specs

- Compatibility
  - We have had especially troubling times with the Safari browser as it has very strict rules about content playback, and also mobile browsers
  - Web security features like CORS also make this platform difficult to support
  - The supported formats and features is dependant on the web browser capabilities
  - We have used it for playback of MP4 files, HLS and MPEG-DASH streams
  - The plugin supports extensions via the hls.js and dash.js libraries for browsers that do not have native support for these streaming formats
- Rendering
  - The plugin supports both WebGL 1.0 and 2.0, however some browsers (notably Safari) do not support WebGL 2.0
- Internals
  - Under the hood we're just using the <video> browser tag
  - No 3rd-party libraries are used, however there is support for adding specific 3rd-party libraries (hls.js and dash.js)

Supported Media

See the Supported Media section for more information.

Streaming

See the Streaming section for more information.

Platform-Specific Options

See the Platform-Specific Options section for more information.

Troubleshooting

Compatibility

- The `Run in Background` setting in Player Settings is best enabled as browsers can sometimes stop updating the Unity context which can lead to problems with state not being updated.
- For best compatibility you can always force WebGL 1.0 instead of 2.0 (which is the default). This is done by going to `Player Settings > Other Settings > Auto Graphics API` and removing WebGL 2.0. We have tested successfully with the following browsers
  - macOS
    - Mozilla Firefox
    - Google Chrome
  - Windows

        - Microsoft Edge 38.14393.0.0
        - Mozilla Firefox 51.0
        - Google Chrome 56.0 - 62.0

- The following browsers are not supported:
    - Internet Explorer 11 (any version), instead use the Microsoft Edge browser
- iOS
    - All browsers on iOS are required to use WebKit and as a result exhibit the same behaviour as Safari, which at this time is not supported.

Supported Media

In general the most common format that is supported are MP4 files with H.264 encoding for video and AAC encoding for audio. This format is supported across all platforms though not necessarily all bit-rates and profiles.

AVPro Video doesn't include native support for any codecs (except for Hap and NotchLC) and relies on codecs that are natively supported by the operating system. The tables below give a fairly accurate idea of what we expect to be supported. On Windows 3rd party codecs can be installed via DirectShow and Media Foundation and are supported.

Container Formats

Container formats are file formats that contain audio, video, text or metadata tracks. An important distinction to realise is that these file formats and are separate for the audio and video codecs. It is not enough to say a video is in 'MP4' format as this format contains tracks which are encoded using different codecs.

	WINDOWS	ANDROID	MACOS	IOS / IPADOS / TVOS
MP4	□	□	□	□
MOV	□	.	□	□
MKV	□ [1]	□	.	.
WebM	□ [2]	□	.	.
AVI	□	.	.	.
MP3	□	□	□	□
AAC	□	□	□	□
WAV	□	.	.	.
CAF	.	.	□	?

[1] Requires Windows 10 for native support. Otherwise DirectShow API can be used with LAV Filters.

[2] Requires Windows 10 (1607 Anniversary and above) for native support. Otherwise DirectShow API can be used with LAV Filters.

Streaming Formats

AVPro Video supports several streaming protocol depending on the platform:

	WINDOWS	UWP	ANDROID	MACOS	IOS / IPADOS / TVOS
HTTP Progressive					
MP4	□	□	□	□	□
Adaptive					

	WINDOWS	UWP	ANDROID	MACOS	IOS / IPADOS / TVOS
HLS (m3u8)	□ [1]	□ [1]	□	□	□
MPEG-DASH (mpd)	□ [1]	□ [1]	□ [4]	.	.
Microsoft Smooth Streaming (ism)	□ [1]	□ [1]	.	.	.
Real-time					
RTSP	~ [2]	.	□ [5]	.	.
RTMP	~ [3]	.	□ [6]	.	.

[1] Requires Windows 10 for native support, or using DirectShow with suitable 3rd party filter (eg LAV Filters).

[2] Limited native support. Read Microsoft notes about support here: https://docs.microsoft.com/en-us/windows/win32/medfound/supported-protocols. Generally only support ASF, MP3 and PCM media types, but support seems improved from Windows 10 build 1803 onwards (as in added H.264 support), but it's not documented (parsing is handled by mfnetsrc.dll).

[3] Only using DirectShow with suitable 3rd party filter (eg LAV Filters).

[4] Using ExoPlayer API only.

[5] Using ExoPlayer API, or MediaPlayer API (but not fully featured).

[6] Using ExoPlayer API only. Known issues surrounding address resolution.

Audio Codecs

	WINDOWS	ANDROID	MACOS	IOS / IPADOS / TVOS
AAC	□	□	□	□
MP3	□	□	□	□ [3]
FLAC	~ [1]	□	□	□
AC3	□	?	□	□
WMA	□	.	.	.
MIDI	□	?	□	□
Vorbis	.	□	.	.
Opus	~ [2]	□	.	.
ALAC (Apple Lossless)	.	.	□	□
µLAW	□	.	□	□

	WINDOWS	ANDROID	MACOS	IOS / IPADOS / TVOS
ADPCM	□	.	□	□
Linear PCM	□	□	□	□

[1] Requires Windows 10 for native support. Otherwise DirectShow API can be used with LAV Filters.

[2] Requires Windows 10 Windows 10 1607 Anniversary and above.

[3] Audio files only, not supported as audio tracks inside MP4 files

Video Codecs

	WINDOWS	ANDROID	MACOS	IOS / IPADOS / TVOS
HEVC / H.265	□ [1]	□	□	□
H.264	□ [5]	□	□	□
H.263 (DivX/XVid)	□	□	.	?
MJPEG	□	.	□	□
WMV	□	.	.	.
VP8	□ [2 4]	□ [4]	.	.
VP9	□ [2 4]	□ [4]	.	.
Hap	□	.	□	.
NotchLC	□	.	.	.
ProRes 422	.	.	□	.
ProRes 4444	.	.	□	.
DV	□	□	.	.
Lagarith	□ [3]	.	.	.
Huffyuv	□ [3]	.	.	.
Uncompressed RGBA	□ [3]	?	□	?
Uncompressed YUV	□ [3]	?	?	?
Uncompressed R10K	.	□	.	.
Uncompressed V210	?	□	.	.

	WINDOWS	ANDROID	MACOS	IOS / IPADOS / TVOS
Uncompressed 2VUY	?	□	.	.

[1] HEVC requires Windows 10 for native support, but the codec no longer ships with Windows and requires a download from Microsoft Store 12. Otherwise DirectShow API can be used with LAV Filters.

[2] Yes, only in Windows 10 and only 4:2:0. Native VP9 support only comes in Yes in Windows 10 1607 Anniversary Update and above, but it may be available before that via Intel GPU drivers. If you use DirectShow and 3rd party filter then 4:4:4 can be supported. Using Media Foundation no audio codecs (Vorbis or Opus) are supported and will cause the video to fail to load if included.

[3] Using DirectShow API and with codec installed.

[4] Transparency is not supported, unless packing is used.

[5] H.264 on Windows only supports 4:2:0 and 8-bit (unless using DirectShow with 3rd party decoder such as LAV Filters).

Android

Android supports many media formats. For a complete list check the Android MediaPlayer documentation here: https://developer.android.com/guide/appendix/media-formats.html and the ExoPlayer documentation here: https://google.github.io/ExoPlayer/supported-formats.html

HEVC (H.265) support was officially added in Android 5.0 (Lollipop) but only as a software decoding implementation on older devices.

We have found that using GearVR on Samsung Galaxy S6 and S7 that H.265 codec works best, with a resolution of 3840x1920 at 30fps, or 2048x2048 at 60fps.

A list of media-player related Android chipsets and which formats they support for hardware decoding: http://kodi.wiki/view/Android_hardware

iOS / iPadOS / tvOS

Many media formats are supported by iOS including H.264. iOS 11 adds support for H.265 (HEVC). The iPhone 7 is the first device with the hardware to support H.265.

iOS doesn't support MP3 audio tracks in a video file, so best to use AAC instead.

It has proven difficult getting the true video decoding capabilities of iOS devices. Apple's website has information, but we found it to be slightly inaccurate (for example we can decode 4K video on iPhone5s, which apparently can only do 1080p). It seems that if your device has a 64-bit processor then it will be able to decode 4K H.264, but older devices with 32-bit processors will not.

macOS

Many media formats are supported by macOS including H.264, HEVC, ProRes 422 and ProRes 4444.

macOS Yosemite (10.10) added support for

- DV
- Uncompressed R10k
- Uncompressed v210
- Uncompressed 2vuy

macOS High Sierra (10.13) added support for

- HEVC (H.265)
- Flac
- Opus (only as a .caf file)

Windows

A full list of natively supported formats can be found here: https://msdn.microsoft.com/en-us/library/windows/desktop/dd757927(v=vs.85).aspx

https://msdn.microsoft.com/en-us/windows/uwp/audio-video-camera/supported-codecs

H.264 decoder supports up to profile L5.1, but Windows 10 supports above L5.1 profile: https://msdn.microsoft.com/en-us/library/windows/desktop/dd797815(v=vs.85).aspx

H.265 decoder specs are here: https://msdn.microsoft.com/en-us/library/windows/desktop/mt218785(v=vs.85).aspx

Windows 10 adds native support for the following formats:

- H.265 / HEVC
- MKV
- FLAC
- HLS Adaptive Streaming
- MPEG-DASH

Windows 10 and UWP HLS features supported: https://docs.microsoft.com/en-us/windows/uwp/audio-video-camera/hls-tag-support

Windows 10 and UWP MPEG-DASH features supported: https://docs.microsoft.com/en-us/windows/uwp/audio-video-camera/dash-profile-support

Windows 10 Fall Update seems to remove native H.265 / HEVC support for some users and requires them to download the free HEVC Video Extension. Before update KB4056892 (4 Jan 2018), users also had to open a H.265 video in the Films & TV app after a restart before AVPro Video could play H.265 videos. This update seems to fix that however.

Windows UWP

Details on media supported by this platform can be found is platform are here:

https://msdn.microsoft.com/library/windows/apps/ff462087(v=vs.105).aspx

https://msdn.microsoft.com/en-us/windows/uwp/audio-video-camera/supported-codecs

WebGL

Support for WebGL platform is still varied and depends on the platform and browser support. Some formats such as AVI file container are not supported at all. As with all other platforms, H.264 video in an MP4 container is the most widely supported format.

Adaptive streaming (such as HLS and MPEG-DASH) is not supported natively by all browsers, but we have seen it working in the Microsoft Edge and Safari browsers. For the best compatibility we have added the ability to include 3rd party javascript libraries to handle these (dash.js and hls.js). See the streaming section for how to implement these.

For best compatibility make sure to force WebGL 1.0 by going to Player Settings > Other Settings > Auto Graphics API and removing WebGL 2.0. Failure to do so can make videos on Chrome not render.

On newer versions of Safari videos are not allows to auto-play unless given permission by the user (in the preferences menu). This doesn't affect videos that have no audio track so this may be a workaround. More details can be found here:

https://webkit.org/blog/7734/auto-play-policy-changes-for-macos/

Some resources about the supported formats:

- https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats
- https://en.wikipedia.org/wiki/HTML5_video#Browser_support

Encoding Notes

Correct video encoding is important to get the best results for different use cases. Mostly videos are encoded for the typical case: forward playback with support for slow random seeking. Alternatively you may want to be able to seek through the video quickly, so you would need to encode with different parameters for that use case.

There is a wide range video encoding software but we shall focus on the command-line tool FFMPEG as it is very mature and flexible, however it should be possible to apply the same concepts with others encoders.

Resources

- FFMPEG
  - H.264 Codec
  - H.265 / HEVC Codec
  - VP9 Codec
  - Filters
  - Colorspaces
- Headjack Encoding Blog

FMPEG Basics

Below is a basic FFMPEG command. This will convert input.mov to output.mp4. The video track will be compressed using the H.264 video codec (libx264 is the name of this encoder), with frames encoded in the 4:2:0 format which is the most common. Since no other options are specified it will use some default settings for the encoder and based on the input video properties. Any audio in the source video will be compressed with the default audio compression for MP4 which is AAC.

```
ffmpeg -i input.mov -c:v libx264 -pix_fmt yuv420p -y output.mp4
```

You can add many parameters to control how videos are encoded. These paramters will limit the output video to 4 seconds, remove the audio track, and change the frame rate to 30 FPS, the size to 1280x702 and with a quality factor of 14 which is quite high quality and will result in a slightly larger file.

```
ffmpeg -i input.mov -c:v libx264 -pix_fmt yuv420p -t 4 -an -r 30 -s 1280x720 -crf 14 -y output.mp4
```

Windows Batch Files

You can create .bat files with FFMPEG commands so that videos can be processed by dragging the files into the batch files.

Using %1 for the input and output file names, this will become the path of the source file:

```
ffmpeg -i %1 -c:v libx264 -pix_fmt yuv420p -y %1-output.mp4
```

Fast Seeking

For a use case of fast seeking/scrubbing through the video, it's important to simplify the encoding so that it favours fast decoding over small file size. This is usually done by increasing the number of key frames and disabling some of the more complex encoding settings (eg B-frames).

Convert a video to H.264 with only key-frames and tuned to decode quickly at the expense of file size:

```
ffmpeg -i input.mov -pix_fmt yuv420p -c:v libx264 -crf 18 -tune fastdecode -x264-params "keyint=1" output.mp4
```

Convert a video to HEVC with only key-frames and tuned to decode quickly at the expense of file size:

```
ffmpeg -i input.mov -pix_fmt yuv420p -c:v libx265 -crf 18 -tune fastdecode -g 1 output.mp4
```

See the section about seeking for further information.

Transparency Packing

Convert a video with alpha channel (eg using ProRes4444) to a left-right alpha packing video:

```
ffmpeg -i input.mov -c:v libx264 -pix_fmt yuv420p -vf "split [a], pad=iw*2:ih [b], [a] alphaextract, [b]
overlay=w" -y output.mp4
```

Convert a video with alpha channel (eg using ProRes4444) to a top-bottom alpha packing video:

```
ffmpeg -i input.mov -c:v libx264 -pix_fmt yuv420p -vf "split [a], pad=iw:ih*2 [b], [a] alphaextract, [b]
overlay=0:h" -y output.mp4
```

Stereo Packing

Convert two videos, each representing one eye into a packed left-right stereo video (without stereo metadata):

```
ffmpeg -i left.mp4 -c:v libx264 -pix_fmt yuv420p -vf "[in] pad=2*iw:ih [left]; movie=right.mp4 [right];[left]
[right] overlay=main_w/2:0 [out]" output.mp4
```

H.265 / HEVC

Convert a video to HEVC (note that the `-tag:v hvc1` is specified here for compatiblity with Apple platforms):

```
ffmpeg -i input.mov -pix_fmt yuv420p -crf 18 -vcodec libx265 -tag:v hvc1 -movflags faststart output.mp4
```

Frame Rate

The frame rate of your video should ideal match the refresh rate of your target display device. If that's not possible then it shoudl be a multiple of the display device rate. For example if you intend to display on a 60hz screen, then ideally your video frame rate should be 60 or 30. This will result in the smoothest playback.

VR / 360 / High Resolution Video

When encoding high resolution video (for example for 360 VR) you should usually consult the developer documentation for the target platform / device. Usually though a codec like VP9 or HEVC (H.265) is required instead of the standard H.264 as HEVC supports higher resolutions.

With FFMPEG to use HEVC codec:

```
ffmpeg -i input.mov -pix_fmt yuv420p -crf 18 -vcodec libx265 -tag:v hvc1 -movflags faststart output.mp4
```

or the VP9 codec:

```
ffmpeg -i input.mov -pix_fmt yuv420p -crf 18 -libvpx-vp9  -movflags faststart output.mp4
```

Colour-space Profiles

When encoding videos there are different standard for colour encoding. They can broken up into 4 main properties with these common values:

- Range

    - `-color_range tv` (limited range)
    - `-color_range pc` (full range)

- Colour transfer
    - `-color_trc bt709`
    - `-color_trc bt2020-10` (for 10-bit)
    - `-color_trc smpte2084` (PQ)
    - `-color_trc arib-std-b67` (HLG)

- Colorspace
    - `-colorspace bt709`
    - `-colorspace bt2020nc`

- Colour primaries
    - `-color_primaries bt709`
    - `-color_primaries bt2020`

The most common encoding for videos is to use the Rec709 standard. Convert a video to 10-Bit HEVC using Rec709 standard with full range:

```
ffmpeg -i input.mov -pix_fmt yuv420p10le -color_primaries bt709 -color_trc bt709 -colorspace bt709 -
color_range pc -crf 18 -vcodec libx265 -tag:v hvc1 -movflags +write_colr -movflags faststart output.mp4
```

10-Bit Videos

Convert a video to 10-Bit HEVC using Rec709 standard with full range:

```
ffmpeg -i input.mov -pix_fmt yuv420p10le -color_primaries 1 -color_trc 1 -colorspace 1 -color_range 2 -crf 18
-vcodec libx265 -tag:v hvc1 -movflags +write_colr -movflags +faststart output.mp4
```

Quality and File Size - Constant Rate Factor

The constant rate factor will try to give you a target quality without caring about the final file size or bit rate. You can control quality using the `-crf` option where the lower the number [1..51] the higher the quality is. Also the `-preset` can be set to the slowest preset you can handle to get the best quality

Convert a video to HEVC with high quality and long encoding time:

```
ffmpeg -i input.mov -pix_fmt yuv420p -crf 8 -vcodec libx265 -tag:v hvc1 -preset slower output.mp4
```

Convert a video to H.264 with normal quality and short encoding time:

```
ffmpeg -i input.mov -pix_fmt yuv420p -crf 28 -vcodec libx264 -preset ultrafast output.mp4
```

Quality and File Size - Variable Bit Rates

If you have a target file size, then you can use 2-pass encoding to achieve the best quality for that size. First you need to caculate the maximum bit rate for your target file size:

```
BitRateInMbps = (TargetFileSizeInMegaBytes x 8) / VideoDurationInSeconds
```

You can then pass this bit rate into the `-b:v` parameter:

```
ffmpeg -i input.mov -c:v libx264 -pix_fmt yuv420p -b:v 3.5M -pass 1 -an -f mp4 NUL
ffmpeg -i input.mov -c:v libx264 -pix_fmt yuv420p -b:v 3.5M -pass 2 -c:a aac output.mp4
```

Progressive MP4 Streaming

A video file can be prepared for streaming by applying the 'fast start' option to an existing video:

```
ffmpeg -i input.mp4 -acodec copy -vcodec copy -movflags faststart output.mp4
```

Hap Codec

FFMPEG doesn't support all the Hap format variants, but you can encode these types:

Convert video to basic Hap:

```
ffmpeg -i input.mov -vcodec hap -format hap output.mov
```

Convert video to basic Hap with alpha channel:

```
ffmpeg -i input.mov -vcodec hap -format hap_alpha output.mov
```

Convert video to basic HapQ for higher quality:

```
ffmpeg -i input.mov -vcodec hap -format hap_q output.mov
```

Convert video to basic HapQ for higher quality with 8 chunks for better multi-threaded decoding, and force using the snappy compressor for smaller file size:

```
ffmpeg -i input.mov -vcodec hap -format hap_q -chunks 8 -compressor snappy output.mov
```

For encoding other newer Hap formats you can find the plugins online including Jokyo, Original Quicktime plugin, Original AV Foundation tool.

Oculus Quest / Go

The Oculus Quest / Go have special encoding requirements which are detailed in this Blog Post.

Facebook Audio 360

Typically the Facebook Audio 360 Sparial Workstation would be used to create compatible MKV files with Opus audio. But in some cases ambisonic encoded videos can be converted using FFMPEG.

Convert a video with ambisonic audio (1st, 2nd or 3rd order) into a MKV file suitable for Facebook Audio 360 decoding and copying the existing video tracks:

```
ffmpeg -y -i input.mov -c:v copy -sample_fmt s16 -acodec libopus -mapping_family 255 output.mkv
```

Removing Audio

If audio is not needed, it's best to strip it out using the `-an` option:

```
ffmpeg -i input.mp4 -an -vcodec copy output.mp4
```

Remove Rotation

Rotation metadata can be removed:

```
ffmpeg -i input.mp4 -metadata:s:v:0 rotate=0 output.mp4
```

Known Issues

AVPro Video is built on top of operating system APIs, and sometimes these APIs have bugs that we cannot easily resolve.

Most of these are edge cases, but here is a list of known issues that we're unlikely to be able to fix:

1. Interlaced video doesn't play back smoothly on Windows using Media Foundation / WinRT APIs

Interlaced videos play back with some stuttering. The workaround is to use progressive video only, or use DirectShow API for interlaced videos.

2. H.264 on Windows only supports 4:2:0 8-bit using Media Foundation / WinRT APIs

Videos with other chroma sub-sampling profiles (eg 4:2:2 or 4:4:4) or 10-bit formats can cause the app/editor to freeze. The workaround is to use 4:2:0 8-bit videos only for H.264 on Windows, or use DirectShow API with LAV Filters installed.

3. Videos with a very large timescale (TBN) can play back skipping frames when using DirectShow API

DirectShow doesn't seem to be able to handle really massive timescale values. This issue is mostly caused by dubious encoding sources. The workaround is to make sure your video is encoded with sensible timescale values. You can also re-encode the video to correct the timescale using FFMPEG command:

```
ffmpeg -i input.mp4 -an -pix_fmt yuv420p -crf 18 -video_track_timescale 90000 -movflags faststart -y output.mp4
```

4. Windows Media Foundation sometimes fails to stream multiple instances of the same MP4 URL concurrently

If you have 3 or more MP4 streams (remote) to the same file, some of the streams will not being playback. The workaround is to switch to the WinRT API. Issue

5. Windows Media Foundation sometimes stops buffering HLS when playback rate is above 1.0

Issue

Frequently Asked Questions

Top

1. Why doesn't my video play?

Typically when media fails to load you will see this generic error message:

```
Loading failed. File not found, codec not supported, video resolution too high or insufficient system resources
```

This is a complicated question to answer, as there are many possible reasons why the media can't be played, and it's not always easy to determine the reason - hence the generic error message.

In order for media to play, let's examine all of the conditions that need be true:

1. **The file must exist at the specified location**
    - Check that the file is at the location specified.
    - The file cannot be in the /Assets folder unless it is within the /StreamingAssets subfolder.

2. **The Unity application must have permission to access this file**
    - For streaming:
        - Platforms often prefer a HTTPS URL and will fail to load a HTTP URL unless forced, see platform notes (Android, iOS) for details.
        - On Android make sure you have the `Internet Access` option in `Player Settings` set to `require`.
        - On UWP make sure to enable the `InternetClient` capability option in `Player Settings`
        - On WebGL it's best to use one of the javascript libraries (eg hls.js) for adaptive streaming
        - On WebGL make sure there is not a CORS permission issue
        - For MP4 files it's best if they're encoded with 'fast start' so they begin streaming immediately
    - For local files:
        - On Android make sure `Write Permissions` in `Player Settings` is set to `External (SDCard)`
        - On Android 11 there are changes to folder permissions - see Android article here
        - On Android if the file is inside the application JAR (from `StreamingAssets`) and is very large, it may fail to decode because the JAR must be opened and uses a lot of memory. For large files we recommend instead streaming from a URL, or downloading it to the persistent storage path and playing from there.

3. **Unity player must be set correctly**
    - The correct graphics API must be used, check the platform requirements. For example on Vulkan is not a supported graphics API on Android.

4. **The system must have sufficient resources to open and decode the media**
    - This is particularly true for mobile / tablet / portable VR platforms which have more resource constraints than desktops
    - Memory and video decode resources are typical bottlenecks. Say your mobile platform supports up to 4K30 video, then you probably need to completely unload the video before playing the next one, as there are limited video decode resources.

5. **The container format (mp4 / mov / mkv / m3u8 etc) and audio-video codecs (h.264 / hevc / aac etc) must be supported**
    - See the section on Supported Media for different platforms

6. **The media codec specs (eg high resolution / frame-rate, 10-bit, 4:4:4, profile/level) must be supported by the hardware**
    - See the section on Supported Media for different platforms
    - If your video is failing, check whether another simpler video plays (perhaps one of our included sample videos), it's possible that your system doesn't support a feature of the media
    - If you're playing high resolution video, check the notes about this

7. **The `MediaPlayer` component must be configured correctly**
    - If you're playing MKV files with 360 audio, then the Facebook 360 Audio options need to be set correctly
    - You may need to select a specific API for video playback in the `Platform Specific` section depending on the media features

> **NOTE**
>
> Just because it plays in the editor doesn't mean it will play when you deploy to another platform. The editor is running on Windows / macOS which has completely different hardware capabilities to mobile devices.

2. How do I troubleshoot my issue?

Whether your video is failing to load, or there is a problem with the display, there are several common steps to help diagnose the cause of the issue:

1. Check the log file - the log file may contain useful error messages which will explain why the media failed to play. See Android notes on how to get the log file.

2. Check the conditions carefully in FAQ question 1.

3. Try loading one of our simple sample videos instead - eg `BigBuckBunny-360p30-H264.mp4` from `StreamingAssets/AVProVideoSamples` or streaming from our test URL: https://rh-testmedia.s3-eu-west-1.amazonaws.com/Samples/BigBuckBunny-360p30-H264.mp4