SUMMARY

This asset works in a similar fashion as the official VideoPlayer component, but for WebGL only. It does everything you'd expect from a video player, and is packed with more events than the official API. If you are having issues playing videos with the standard video player on Safari or other browser(s), you came to the right place!

FEATURES

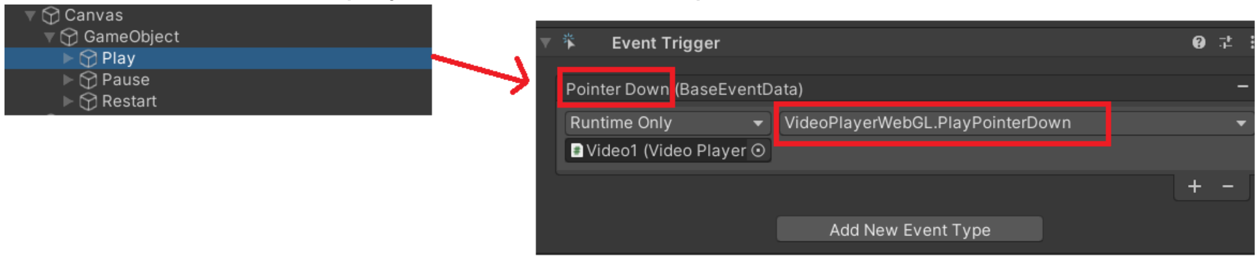
1. You can play, pause, stop, restart, seek, change the volume, the playback rate, mute, check the length, width, height, if it's playing, if it's paused, check the current time.
2. You can have multiple videos playing at the same time.
3. It supports all available video events for the web. There are 20 events available, the official Unity API only supports 8.
4. You can easily configure everything through the inspector. You can also change anything later at runtime if you want to. For example, changing the source of the video to play multiple content using the same video instance.
5. You can autoplay the video.
6. You can play videos from the StreamingAssets folder(local) or from an external URL. Easily configurable through the inspector.
7. It works on basically anything. You can play the video on a plane or a sphere for example. You can play the video on a raw image. My asset works by storing the video feed on a render texture, which you then use to create materials that you apply to your objects.
8. It works on all major browsers: chrome, safari, firefox, edge, opera. Currently just not working on firefox for Android, because of a firefox bug. Nothing I can do about it.
9. **NEW IN 1.6** You can downmix to mono, and control the panning. Check out the compatibility table for these features. Special thanks to @**Matt Foreman,** who created the example script under MarksAssets/VideoPlayerWebGL/Example/Scripts that implements spatial audio using them.

REQUIREMENTS

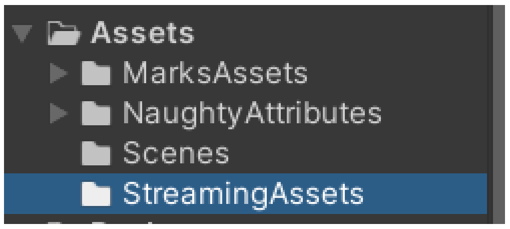
1. Install the NaughtyAttributes free asset first. It works on Unity 2019 or higher.
2. Use mp4 files encoded with H.264 or MPEG-4 for cross-browser compatibility. At

the time of writing, webm isn't supported on Safari yet.

3. If your video's source is an external URL, it must be a direct link pointing to the video, that is, the URL must end with the file name. For example, this video. It works on videos stored in CDNs like AWS or CloudFlare for example. But it does **not** support playing videos from youtube because the URL from youtube videos are not direct links.
4. If your video is not muted and has an audio track, you **must** use the *UnlockVideoPlayback* or *PlayPointerDown* method at least once on a pointerdown event to make the video play on Safari. The example scene shows this.



5. Each VideoPlayerWebGL instance **must** have its own render texture.
6. If you are hosting your video files locally, they must reside in the StreamingAssets folder, and the StreamingAssets folder must be a direct child of the root folder. If the folder doesn't exist, create one.



"StreamingAssets" is a direct child of the root folder "Assets"

HOW TO USE

Under the *VideoPlayerWebGL->Scripts* folder, there is one file called *VideoPlayerWebGL.cs*. Drag it to any game object. Configure the parameters on the inspector or through code, or both, and you're done.

INSPECTOR

1. Source: a dropdown menu. If you want to play a video that resides inside your project, use the "StreamingAssets" option. If you want to play a video that resides on an external server, use the "External" option.

a. “StreamingAssets”: By selecting this option, the “File Name” field will appear. Just type the name of your video in it(including the file extension). For example, “myvideo.mp4”. You also need to put the file under the StreamingAssets folder. This folder **must** be under the root of your project: *Assets->StreamingAssets*. If the folder doesn’t already exist, create one.

b. “External”: By selecting this option, 2 fields will appear:
   i. “URL” field. Type the full path of the video here. For example: *https://d8d913s460fub.cloudfront.net/videoserver/cat-test-video-320x240.mp4* .
   ii. “Cross Origin” field, a dropdown menu. The option to select here depends on how the response header of the server hosting the video is configured. If it has the header “*Access-Control-Allow-Origin: **”, for example, you need to select the *Anonymous* option. Please note that setting the CORS option here is necessary, but not sufficient to make it work. The server hosting the video needs to set the proper response headers as well.

2. Autoplay: Tick this field to automatically play the video when the video loads. Playing the video automatically always works if the video is **muted,** without any user intervention. If it’s not muted, there are restrictions. On Chrome, it will only play if the user interacts with the document before the video loads. This means simply tapping on the screen anywhere. If you can guarantee that the user will interact with the document before playing the video(maybe there’s a start screen with a start button or something) and your target is just Chrome, you can leave the script disabled, and only enable it once it’s supposed to play(because the video loads on Start), then the autoplay option will work even if the video is not muted. On Safari, it’s simply not possible. It requires a user gesture to play the video, meaning the user needs to tap on something(a button for example) and the play method needs to be called manually. To be honest it’s just simpler that you forget that autoplay even exists if the video is unmuted, because it only works on Chrome under the specific condition described above. The universal solution is to call the *PlayPointerDown* method on a button. You can read more about it later on.
3. Loop: Tick this field to make the video loop infinitely.
4. Muted: Tick this field to mute the video. Necessary to make the video autoplay on Chrome without user intervention, and the **only** way to autoplay on Safari.
5. Volume: A slider to control the volume of the video. From 0 to 1, where 0 there is no volume and 1 is the normal volume. Please note that changing the volume does not work on mobile Safari.
6. Pan: A slider to control the stereo pan of the video. From -1 to 1.
7. Force Mono: If this option is ticked, the audio is downmixed to mono.

8. Playback Speed: A slider to control the speed of the video. From 0 to 10, where 0 the video is effectively paused, 1 is the normal speed, 2 is twice as fast, etc.
9. Target Texture: The render texture where the video feed will be sent to. In the *Textures* folder of this asset you will find the texture *VideoMaterialTexture.renderTexture* that I created for your convenience. The size of the texture you can leave at 256x256, the correct dimensions are set by code internally. Apply the texture to a material, and then apply the material to a Mesh Renderer. Or just drag the material to any object in the scene that works with materials, like a quad or a cube. I also created the material *VideoMaterial.mat* under the *Materials* folder for your convenience. **IMPORTANT: YOU NEED ONE RENDER TEXTURE PER VIDEOPLAYERWEBGL INSTANCE.**
10. Events: A multi selectable dropdown menu. Here you can select the events you are interested in subscribing to. For each event that you select, the corresponding UnityEvent will show up in the inspector, where you can add callbacks that will run when the UnityEvent is invoked. For example, if you select the "*Ended*" event, the "Ended" UnityEvent shows up in the inspector; if you then add a callback to disable a gameobject, when the video finishes playing, on the javascript side the ended event will be called and your corresponding "Ended" UnityEvent will be invoked, and the gameobject disabled.

METHODS

First, there are 3 enums that are used as input and output on some of the methods:
public enum cors {anonymous = 1, usecredentials = 2};
public enum srcs {StreamingAssets, External};

[Flags]

public enum evnts {canplay = 1, canplaythrough = 2, complete = 4, durationchange = 8, emptied = 16, ended = 32, loadeddata = 64, loadedmetadata = 128, pause = 256, play = 512, playing = 1024, progress = 2048, ratechange = 4096, seeked = 8192, seeking = 16384, stalled = 32768, suspend = 65536, timeupdate = 131072, volumechange = 262144, waiting = 524288};

Now to the methods:
public void CreateVideo(srcs _source, string _path, cors _crossOrigin, bool _autoplay, bool _loop, bool _muted, double _volume, double _pan, bool _forceMono, double _playbackSpeed, RenderTexture _targetTexture, evnts _events)

This method is called automatically on Start. You only need to call it manually if you destroy the video and want to recreate it. If for some reason you don't want the video to load automatically, disable the script on the inspector. Don't confuse loading with playing. By loading I mean creating the video element on the html side and retrieving the video from the specified source(local or external).

1. _source: use *srcs.StreamingAssets* option for videos in the *Assets/StreamingAssets* folder(local) or *srcs.External* option for videos located on an external server.
2. _path: It depends on the option you selected for *_source:* if you selected *srcs.StreamingAssets*, _path just takes the name of the video file, including the extension. If you selected *srcs.External,* it's the URL of the file.
3. _crossOrigin: *cors.anonymous or cors.usecredentials*. This parameter is only relevant if *_source* is *srcs.External*.
4. _autoplay*:* true if you want to play the video automatically on load, false if not.
5. _loop*:* true if you want the video to loop infinitely, false if not.
6. _muted: true if you want the video to be muted, false if not.
7. _volume: from 0 to 1, where 0 is a video with no volume and 1 is the normal volume.
8. _pan: from -1 to 1, where -1 is full left pan, and 1 is full right pan. See StereoPannerNode
9. _forceMono: If true, the video's audio is downmixed to mono. If false, the number of channels is preserved.
10. _playbackSpeed*:* from 0 to 10, where 0 the video doesn't play, 1 is the normal speed, 2 is twice as fast, etc.
11. _targetTexture: the texture where the video feed will be sent to.
12. _*events*: The events you want to listen to. You can pass a single event or multiple, for example: *evnts.timeupdate | evnts.play*(listen to timeupdate and play events). If you want to listen to all, you can pass *(evnts)(-1)* as input.

public void Play()

Plays the video from its current time. If the current time is the end of the video, it plays from the start. It works no problem on Chrome, but for Safari, see the 2 methods below. This method can be called anywhere you want.

public void PlayPointerDown()

Same as *Play()*, but this method only works if called on a pointerdown event. This method was made because of Safari. On Safari, you need to either call this method once or *UnlockVideoPlayback* once. And then you can use *Play()* normally. Alternatively, you can simply always call *PlayPointerDown()*. This method also works on Chrome, so if you want a universal method for all browsers, use this one.

public void UnlockVideoPlayback()

This method just plays and immediately pauses a video. It only works on a pointerdown event. It was made because of Safari. On Safari, you need to either call this method once or *PlayPointerDown* once. And then you can use *Play()* normally. This method also works on Chrome, so you can use *UnlockVideoPlayback* + *Play* for Chrome and Safari for a universal solution(although unnecessary for Chrome, because on Chrome it works with *Play* alone). The idea of this method is to call it on a button at the start of the experience, together with other permission requests like the gyroscope, so that all permission stuff is done in one place and then you can forget about it.

public void Restart()

Restarts the video. Be sure to have called *PlayPointerDown* or *UnlockVideoPlayback* on Safari at least once, otherwise this method won't work.

public void Pause()

Pauses the video

public void Stop()

Pauses the video, and sets its time to 0.

public double Time()

Returns the current time of the video, in seconds.

public void Time(double time)

Seeks the video, pass the time you want the video to jump to, in seconds. If time is < 0 it becomes 0. If it's > than the video's length, it becomes the video's length.

public double Length()

Returns the length of the video, in seconds.

public bool IsSetToLoop()

returns true if the video is set to loop, false if not.
public void Loop(bool lp)

set true if you want the video to loop, false if not.

public bool IsMuted()
Returns true if the video is muted, false if not.

public void Muted(bool mute)
set true if you want to mute the video, false if not.

public bool IsSetToAutoPlay()
Returns true if the video was set to autoplay, false if not.

public void Autoplay(bool autoplay)
Set true if you want the video to autoplay, false if not. It's only useful to use this if you change the source of the video at runtime, and want to autoplay a video that previously was set to not autoplay, and vice versa.

public double PlaybackSpeed()
Returns the current playback speed of the video, default is 1.

public void PlaybackSpeed(double pbspd)
Sets the playback speed of the video. It can be anything from 0.0(inclusive) to 10.0(inclusive). Values outside that range will be clamped to the nearest valid value.

public string Source()
Returns the source of the video.

public void Source(srcs src, string path, cors crossorigin = cors.anonymous) Sets the source of the video
Example external source: *'Source(srcs.External,"https://d8d913s460fub.cloudfront.net/videoserver/cat-test-video-320x240.mp4")'*
Example StreamingAssets(local) source: *'Source(srcs.StreamingAssets, "cat-test-video-320x240.mp4")'*
The third argument is only relevant if the source is external.

public bool IsPlaying()

Returns true if the video is currently playing, false if not.

public bool IsPaused()

Returns true if the video is currently paused, false if not.

public uint Width()

Returns the width of the video, in pixels.

public uint Height()

Returns the height of the video, in pixels.

public double Volume()

Returns the current volume of the video.

public void Volume(double vol)

Sets the volume of the video. It can be anything from 0.0(inclusive) to 1.0(inclusive). Values outside that range will be clamped to the nearest valid value. It doesn't work on mobile Safari.

public double Pan()

Returns the current pan of the video

public void Pan(double _pan)

Sets the pan of the video. It can be anything from -1.0(inclusive) to 1.0(inclusive). Values outside that range will be clamped to the nearest valid value. Requires *PlayPointerDown* or *UnlockVideoPlayback* to be used at least once.

public bool ForceMono()

Returns true if the video is currently being downmixed to mono, false if not.

public void ForceMono(bool forceMono)

If the parameter is true, the video is downmixed to mono. If false, the original number of channels is restored. You can use this method to downmix from stereo to mono at runtime and later on restore to stereo, for example. This method doesn't take effect immediately. It only takes effect after you start playing the video using the PlayPointerDown method. This means that if you call this method while the video is playing, you will only notice the change after it pauses/finishes and you call PlayPointerDown again. The normal Play method doesn't work here, ever.

public cors CORS()

Returns the current crossorigin configuration. It can either be cors.anonymous or cors.usecredentials

public srcs SourceType()

Returns the current source type. It can either be srcs.StreamingAssets or srcs.External

public void CORS(cors crossorigin)

Sets the cross origin attribute on the video. Irrelevant if the video is local(StreamingAssets folder), but required if the source is external. It can be cors.anonymous or cors.usecredentials.

public bool IsReady()

Returns true if the video is ready to be played, false if not.

public void Destroy()

Destroys the video from the html side, releases the target texture, unregisters all Unity Events and removes all non persistent(runtime) callbacks from them. This method is automatically called on OnDestroy.

public void RegisterEvent(evnts evt)

Registers one event, or multiple events. For example: *RegisterEvent(evnts.timeupdate|evnts.play)* will register *timeupdate* and *play* events. For example

first you add the callbacks that you want

myAction += myCallback;

play.AddListener(myAction);//accessing the play UnityEvent and adding a callback then you call *RegisterEvent(evnts.play)* to make your play UnityEvent be invoked(and all of its callbacks) when the video plays

You only need to call this method if you destroyed the video and recreated it without registering the event again, or if you didn't subscribe to the event in the inspector, or if you unregistered the event using the UnregisterEvent method.

If you want to register all events, pass *(evnts)(-1)* as input

public void UnregisterEvent(evnts evt, bool removeAllNonPersistentListeners = false)

For example:

UnregisterEvent(evnts.timeupdate | evnts.play) will unregister *timeupdate* and *play* events.

So in this example it means that when the video plays or the *timeupdate* event is fired from javascript, the Unity Events subscribed to them won't be invoked. if the second argument is true, in addition to unregistering the event, its runtime callbacks will be removed as well.

Callbacks added from the inspector are persistent and not removed if the second argument is true.

if you want to unregister all events, pass *(evnts)(-1)* as input

PROPERTIES

Everything is accessible through methods, except the *targetTexture(RenderTexture)* and the UnityEvents corresponding to the video events. These are public properties that can be accessed directly.

Are there known issues?

Yes, and they are all browser restrictions unless said otherwise:

1. Only one unmuted video can play at any time on Safari. Multiple videos will play at the same time only if they are muted or don't have an audio track. On chrome there is no problem.
2. Autoplaying videos has restrictions. It works on Safari and Chrome if the video is muted, no need for user intervention. However, If the video is not muted it doesn't work on Safari at all, and on chrome it works only if the user taps anywhere on the screen(doesn't need to be a button) before the video loads.
3. Chrome can fail to seek a video. This is a browser bug. It happened to me on

Chrome for Windows, but didn't happen on Chrome for Android.

4. When trying to play a video from an external URL, you can get an error like *"Access to video at 'urlVideo' from origin 'urlOrigin' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource."*. This error is caused because the server providing the video did not set the proper CORS response header.
5. When trying to play a video on Safari, even if it's from the StreamingAssets folder, you can get this error: *"Failed to load resource: the server responded with a Status of 404()"*. This happens if the web server you are using doesn't support the Range request header.
6. When trying to play a video on Safari, you get this error: *"NotAllowedError: The play method is not allowed by the user agent or the platform in the current context, possibly because the user denied permission."*. This is likely because you didn't follow requirement number 5.
7. You can not change the volume on mobile Safari.
8. Firefox has an issue that prevents my plugin from working on Android. Nothing that I can do about this.
9. It might take a long time to load a video on Safari if it's too large in size. If this happens, try compressing it.You can find more information here.
10. You get the error *NotSupportedError: The media resource indicated by the src attribute or assigned media provider object was not suitable.* Or *Access Denied.* Or *Failed to load resource: the server responded with a status of 404 (Not Found).* Please make sure that if you are using the StreamingAssets folder, it's directly under the root Assets folder. It **must** be *Assets > StreamingAssets* and **not** something like *Assets > Assets > StreamingAssets*.
11. You get the error *SecurityError: The operation is insecure* only on iOS Safari. Please see this SO post. It's likely that your CDN is using redirection and Safari can't handle it. Check the network tab and use the final url after the redirection.
12. My asset does **not** work in play mode, only in the actual build. This is true for all .jslib plugins. You can run the plugin in play mode but **it won't do anything.**
13. When trying to get the width and/or height of the video, it returns 0. This can happen if you try to retrieve information from the video before it has properly loaded. Try waiting for the canplay or loadeddata event to fire first.
14. You get the following error(s), or similar one(s):
    "Failed to create RenderTexture with RGBA16 UNorm (24) format. The platform doesn't support that format, and it doesn't have a compatible format"
    "RenderTexture.Create failed: format unsupported - None (24)."
    Please take a look at this forum thread. Your device doesn't support the render

texture used in the example. You can try another kind of render texture.