## 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, check the buffer, check the network state, preserve pitch or not, check readiness state of video, check the time ranges the user can seek to.
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.
9. Playing videos with transparency is possible since 1.8 across browsers, by using the *ChromaKey* shader under *MarksAssets/VideoPlayerWebGL/Shaders/ChromaKey* for videos with green screen background. Alternatively, there is also the *Alpha Video* shadergraph under *MarksAssets/VideoPlayerWebGL/Shaders/Side-by-Side Alpha Masked*. There are a few important remarks about the latter:

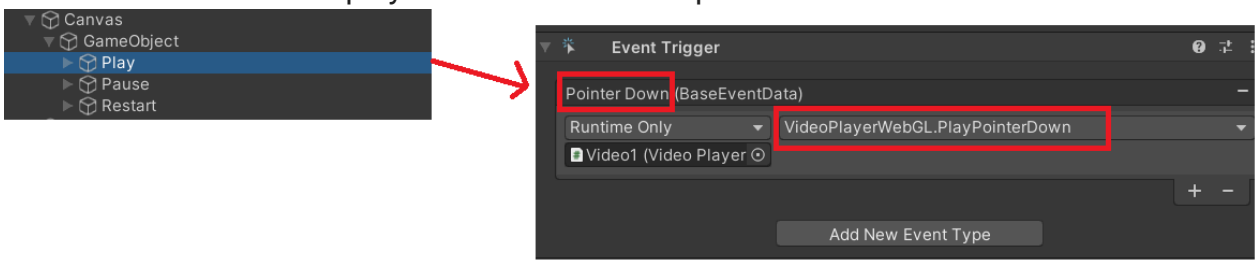
    1-If you use the built-in RP, it only works on Unity 2021.2 or higher. Otherwise, it requires you to use URP.

    2- You must prepare the video, so that one of the sides has the color channels, and the other side has the alpha channel. As a consequence, the prepared video will have twice the width or height of the original video. If your video has a bigger width than height, duplicate the height. If it has a bigger height, duplicate the width. This way, you might not need to compress the video.

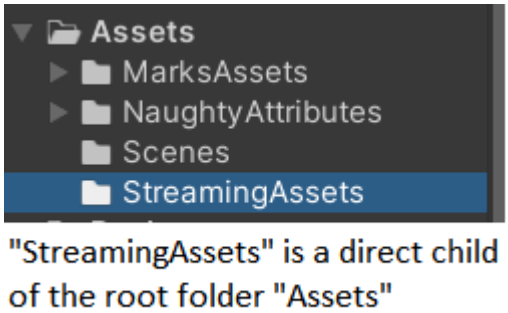
3- In the video preparation, it must be encoded as an h.264 mp4 file.

## 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. If you don't want to listen to any event, pass 0 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()***

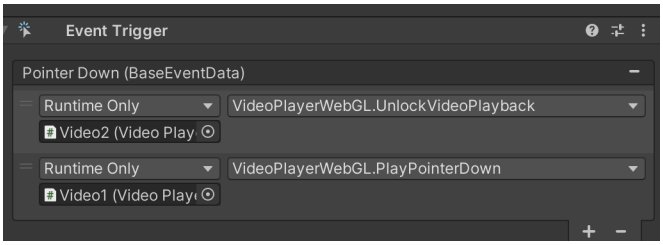
This method just plays and immediately pauses a video. It only works on a pointerdown event. It was made because of Safari. On Safari, you need to either call this method once or *PlayPointerDown* once. And then you can use *Play()* normally. This method also works on Chrome, so you can use *UnlockVideoPlayback* + *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. A typical use case is having a start scene with a start button, where you click on said button and use the pointerdown event with this method to switch scenes and also unlock the video.

Example:

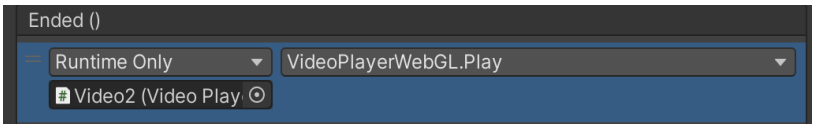
```
//some script attached to a gameobject on the first scene
private VideoPlayerWebGL vp;
    [SerializeField]
    private RenderTexture rt;//render texture, which also persists
across scenes.

    private void Awake() {
        var vps = FindObjectsOfType<VideoPlayerWebGL>();
        if (vps is null || vps.Length == 0) {//if there's no video
yet, create one
```

```
            vp = gameObject.AddComponent<VideoPlayerWebGL>();
            vp.CreateVideo(VideoPlayerWebGL.srcs.StreamingAssets,
"video.mp4", VideoPlayerWebGL.cors.anonymous, false, false, false,
1.0, 0, false, 1.0, rt, 0);
        } else {//otherwise got back to this scene, get video that is
already available
            vp = vps[0];
        }
        DontDestroyOnLoad(vp);//we want the video to persist across
scenes, because on this scene it will just be unlocked.
    }

    public void UnlockVideo() {//call this on pointerdown event of
//button that switches scenes
        vp.UnlockVideoPlayback();
    }

//now on some script attached to a gameobject on some other scene
//that uses the same video from the first scene.
void Awake() {
    vp = FindObjectsOfType<VideoPlayerWebGL>()[0];
}
```

Another common use case is when you have a first video that has to play, and a second one that should play after the first finishes. You can play the first video and unlock the second on the same button, like so (you **must always** call UnlockVideoPlayback **before** PlayPointerDown, otherwise it won't work on mobile Safari, the order is important here)




On the first video's end event, you can simply call Play on the second video now




***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 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;*

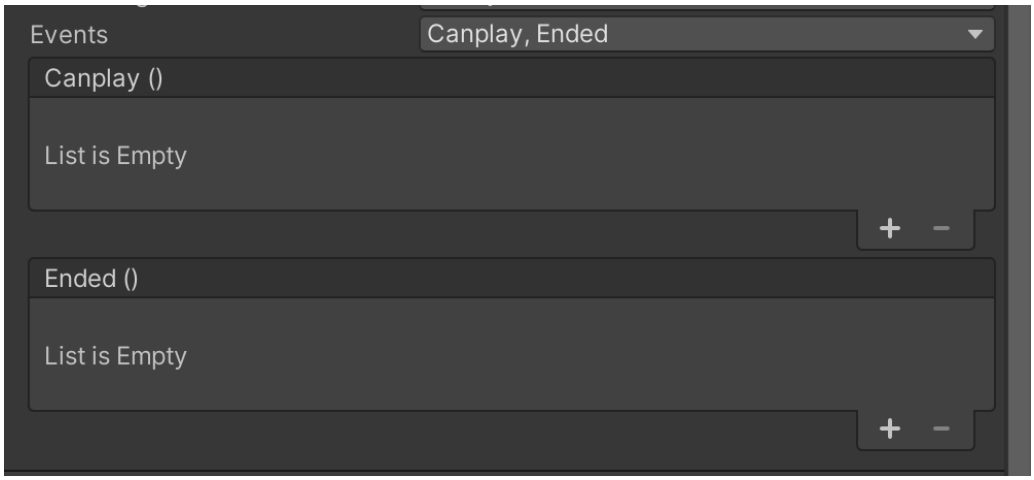
*play.AddListener(myAction);//accessing the play UnityEvent and adding a callback* then you call *RegisterEvent(evnts.play)* to make your play UnityEvent be invoked(and all of its callbacks) when the video plays .

The events are normally registered on *Start*, subscribing to the events you set up in the inspector because *Start* calls *CreateVideo,* and the last parameter of *CreateVideo* can be

used to subscribe to all the events you want.

*RegisterEvent* is not normally necessary to be used, but in a couple of situations:

1. If you don't subscribe to events in the inspector. For example, the following image shows a video subscribed to *ended* and *canplay*, even though there are no callbacks attached to them. In this case, the *RegisterEvent* method is **not** necessary for *ended* and *canplay*.




2. If you called the *UnregisterEvent* method with the event you previously registered, or called the *Destroy* method.
3. If you create the video manually with *CreateVideo* but don't subscribe to the event(s) you want on the last parameter.

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

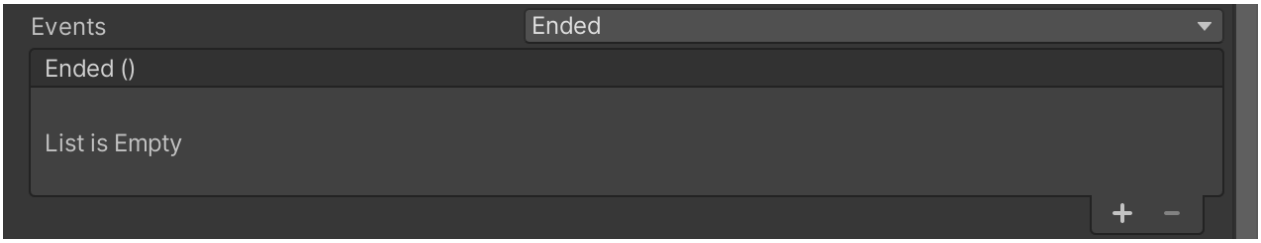
If you are creating the video through code, you need to assign a new *UnityEvent* before calling the *CreateVideo* method, and you must add the listener itself **after** the creation. Example:

```
void Start() {
        vp = gameObject.AddComponent<VideoPlayerWebGL>();
        vp.ended = new UnityEvent();
        vp.CreateVideo(VideoPlayerWebGL.srcs.StreamingAssets,
"myvideo.mp4", VideoPlayerWebGL.cors.anonymous, true, false, true,
1.0, 0, false, 1.0, rt, VideoPlayerWebGL.evnts.ended);
        vp.ended.AddListener(() => Debug.Log("ended"));
```

```
    }
```

If you already have the video player attached to a gameobject, make sure the event is already registered in the inspector, then just add the listener in the script. You should always wait a frame though, otherwise it might not work.

Example




```
IEnumerator Start() {
    videoPlayer = GetComponent<VideoPlayerWebGL>();
    yield return null;
    videoPlayer.ended.AddListener(() => Debug.Log("ended"));
}
```

***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.

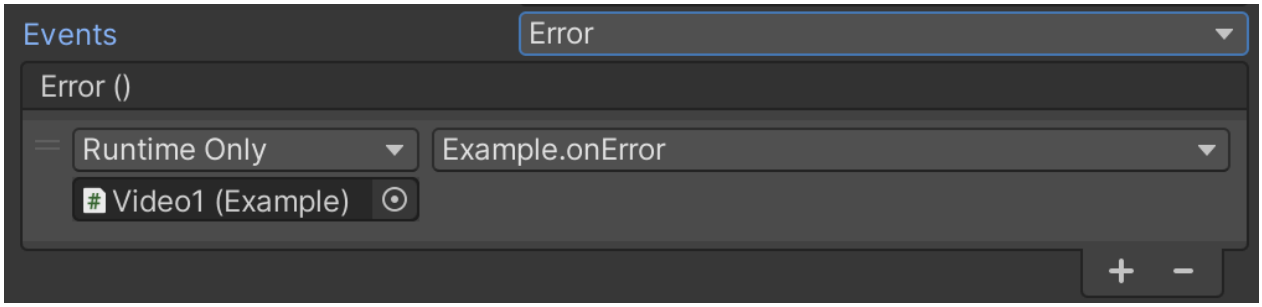
Callbacks added from the inspector are persistent and not removed if the second argument is true.

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

***public MediaError GetError()***

Returns the most recent error that occurred when attempting to play the video, or null if there hasn't been any error. Use this in conjunction with the error event to handle video playback issues. Example usage:

```
public class Example : MonoBehaviour {
    public VideoPlayerWebGL vp;

    public void onError() {
        Debug.Log("errorCode: " + vp.GetError().code);
    }
}
```




**PUBLIC PROPERTIES**

1. RenderTexture targetTexture
2. UnityEvents corresponding to the video events.
3. double length (corresponds to duration and Unity's Video Player length). Call this at least after the loadedmetadata event fires, otherwise it will return -1.
4. TimeRanges buffered (see VideoPlayerWebGLSeekControlSampleScene)
5. TimeRanges seekable
6. NETWORKSTATE networkState
7. READYSTATE readyState
8. bool preservesPitch (need to wait a frame before setting the value on Start)

**HLS SUPPORT**

This plugin doesn't directly support HLS, and if you use it without any third party plugins, your success will depend on whether the browser supports it or not. The good news is, there's a third party library called hls.js which you can easily integrate with my plugin. There's only 2(or 3) steps

1. Modify the generated HTML to be like this (or create a custom webgl template. You'll add the hls library from CDN)

```
<!DOCTYPE html>
<html lang="en-us">
<head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Unity WebGL Player | VideoPlayerWebGL</title>
    <link rel="shortcut icon" href="TemplateData/favicon.ico">
    <link rel="stylesheet" href="TemplateData/style.css">
    <script src="https://cdn.jsdelivr.net/npm/hls.js@1"></script>
</head>
  <body>
    <div id="unity-container" class="unity-desktop">
      <canvas id="unity-canvas" width=960 height=600></canvas>
      <div id="unity-loading-bar">
        <div id="unity-logo"></div>
        <div id="unity-progress-bar-empty">
          <div id="unity-progress-bar-full"></div>
        </div>
      </div>
      <div id="unity-warning"> </div>
      <div id="unity-footer">
        <div id="unity-webgl-logo"></div>
        <div id="unity-fullscreen-button"></div>
        <div id="unity-build-title">VideoPlayerWebGL</div>
      </div>
    </div>
    <script>
      var container = document.querySelector("#unity-container");
      var canvas = document.querySelector("#unity-canvas");
      var loadingBar = document.querySelector("#unity-loading-bar");
      var progressBarFull = document.querySelector("#unity-progress-bar-full
      var fullscreenButton = document.querySelector("#unity-fullscreen-butto
      var warningBanner = document.querySelector("#unity-warning");
      window.hls = new Hls();

      // Shows a temporary message banner/ribbon for a few seconds, or
```

2. Modify the CreateVideo function from VideoPlayerWebGL.jspre by commenting out the *video.setAttribute('src', url);* line and adding 2 lines `hls.loadSource(url);` `hls.attachMedia(video);` to the file like so

```
        video.VideoPlayerWebGL.useMipMap = useMipMap;
        events && events.forEach((function(event) {
            video.VideoPlayerWebGL.events[event] = (function() {
                Module['VideoPlayerWebGL'][event](id)
            });
            video.addEventListener(event, video.VideoPlayerWebGL.events[event])
        }));
        video.setAttribute('id', 'VideoPlayerWebGL_' + id);
        video.setAttribute('name', /[^/]*$/.exec(url)[0].replace('#t=0.0000001', ''));
        //video.setAttribute('src', url);
        hls.loadSource(url);
        hls.attachMedia(video);
        video.setAttribute('playsinline', '');
        video.defaultPlaybackRate = playbackSpeed;
        video.playbackRate = playbackSpeed;
```

3. If you have to change the video source to another .m3u8 file, you have to modify the SourceVideo function following the same logic

```
Module['VideoPlayerWebGL'].VideoPlayerWebGL_SourceVideo = function(id, url, cors) {
    const video = document.getElementById('VideoPlayerWebGL_' + UTF8ToString(id));
    if (!video) return;
    url = UTF8ToString(url);
    cors = UTF8ToString(cors);
    //video.setAttribute('src', url);
    hls.loadSource(url);
    hls.attachMedia(video);
    video.setAttribute('name', /[^/]*$/.exec(url));
    video.VideoPlayerWebGL.createdVideoFlag = video.VideoPlayerWebGL.playingFlag = video.VideoPlayerWebGL.timeupdateFlag = false;
    cors ? video.setAttribute('crossorigin', cors) : video.removeAttribute('crossorigin');
    video.VideoPlayerWebGL.playedFirstTimeAfterLoading = false;
    video.load();
    video.addEventListener('loadeddata', function() {video.currentTime = 0.0001;}, {once: true});
};
```

And that's it…you should be good to go and be able to play m3u8 files!

## PLAYING ALPHA VIDEOS

The easiest and more efficient way is to use a video with a green background and use the *ChromaKey* shader. There are cases where the *ChromaKey* can be insufficient though. If you can't achieve a clear enough separation, or if you have a dynamic alpha channel (e.g places with 50% transparency, like smoke etc) for example. If the ChromaKey solution doesn't cut it for you, you can use the *Alpha Video* shadergraph. This one isn't as straightforward, but it's still quite simple. Below are the instructions for it.

Assuming you have already prepared the video, here are a few configurations for the alpha video shader where C= color channels, A = Alpha channel, L = left, R = right, T = top, B = bottom. For example CL/AR means that you prepared the video with the color channels on the left side, and the alpha channel to the right side. The shader's default values are for this case.

CL/AR (default)

Multiplier x: 0.5, y: 1

BaseColorOffset: x: 0, y: 0
AlphaOffset: x: 0.5, y: 0

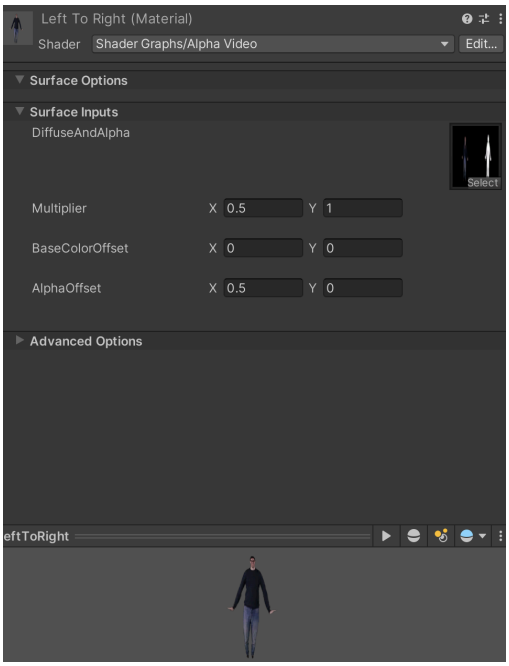
CR/AL
Multiplier x: 0.5, y: 1
BaseColorOffset: x: 0.5, y: 0
AlphaOffset: x: 0, y: 0

CT/AB
Multiplier x: 1.0, y: 0.5
BaseColorOffset: x: 0, y: 0.5
AlphaOffset: x: 0, y: 0

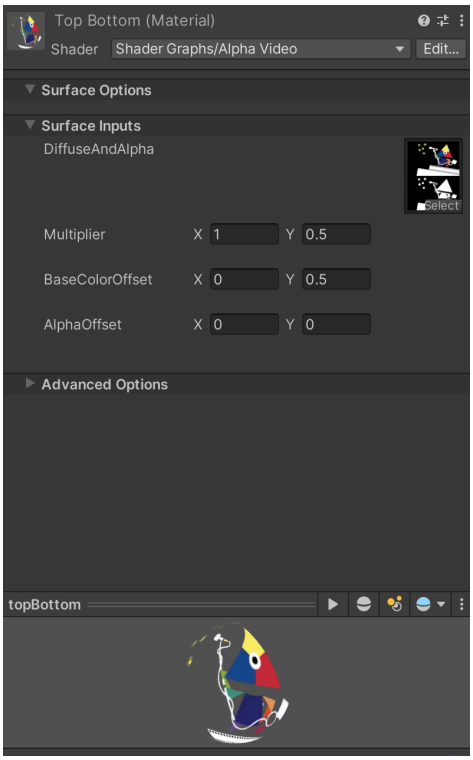
CB/AT
Multiplier x: 1.0, y: 0.5
BaseColorOffset: x: 0, y: 0
AlphaOffset: x: 0, y: 0.5

It helps to take a snapshot of the first video frame and use the texture in the inspector to see if the video will display correctly without having to build. You may have to play a little bit with the offsets.

For example, here's a CL/AR prepared video in the inspector.




And here's a CT/AB one



**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."*, or this error: "*NotSupportedError: The operation is not supported*" This is likely because you didn't follow requirement number 4.
7. You can not change the volume on mobile Safari.
8. Firefox has an issue that prevents my plugin from working on Android.
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.
15. The video color is washed out: There are 3 ways you can solve this

15a) *Edit->Project Settings...->Player->Other Settings->Rendering->Color Space->* Select Gamma

15b) If you really need Linear Color space, go to *Assets->MarksAssets->VideoPlayerWebGL->Plugins->VideoPlayerWebGL.jspre*. Open the file and go to the function *Module['VideoPlayerWebGL'].VideoPlayerWebGL_UpdateTexture*. There are **2 lines** (one inside an *if*, another inside an *else*) like this *GLctx.texImage2D(GLctx.TEXTURE_2D, 0, GLctx.RGBA, GLctx.RGBA, GLctx.UNSIGNED_BYTE, video);*.

```
VideoPlayerWebGL_UpdateTexture: function(id) {
    const video = document.getElementById("VideoPlayerWebGL_" + id);
    if (!video) return false;
    if (!video.VideoPlayerWebGL.canUpdateTexture) return false;
    if (!(video.videoWidth > 0 && video.videoHeight > 0)) return false;

    if (video.previousUploadedWidth != video.videoWidth || video.previousUploadedHeight != video.videoHeight) {
        GLctx.deleteTexture(GL.textures[video.VideoPlayerWebGL.texturePtr]);
        GL.textures[video.VideoPlayerWebGL.texturePtr] = GLctx.createTexture();
        GL.textures[video.VideoPlayerWebGL.texturePtr].name = video.VideoPlayerWebGL.texturePtr;
        var prevTex = GLctx.getParameter(GLctx.TEXTURE_BINDING_2D);
        GLctx.bindTexture(GLctx.TEXTURE_2D, GL.textures[video.VideoPlayerWebGL.texturePtr]);
        GLctx.pixelStorei(GLctx.UNPACK_FLIP_Y_WEBGL, true);
        GLctx.texParameteri(GLctx.TEXTURE_2D, GLctx.TEXTURE_WRAP_S, GLctx.CLAMP_TO_EDGE);
        GLctx.texParameteri(GLctx.TEXTURE_2D, GLctx.TEXTURE_WRAP_T, GLctx.CLAMP_TO_EDGE);
        !video.VideoPlayerWebGL.useMipMap ? GLctx.texParameteri(GLctx.TEXTURE_2D, GLctx.TEXTURE_MIN_FILTER, GLctx.LINEAR) : GLctx.te
        GLctx.texImage2D(GLctx.TEXTURE_2D, 0, GLctx.RGBA, GLctx.RGBA, GLctx.UNSIGNED_BYTE, video);
        video.VideoPlayerWebGL.useMipMap && GLctx.generateMipmap(GLctx.TEXTURE_2D);
        GLctx.pixelStorei(GLctx.UNPACK_FLIP_Y_WEBGL, false);
        GLctx.bindTexture(GLctx.TEXTURE_2D, prevTex);
        video.previousUploadedWidth = video.videoWidth;
        video.previousUploadedHeight = video.videoHeight
    } else {
        GLctx.pixelStorei(GLctx.UNPACK_FLIP_Y_WEBGL, true);
        var prevTex = GLctx.getParameter(GLctx.TEXTURE_BINDING_2D);
        GLctx.bindTexture(GLctx.TEXTURE_2D, GL.textures[video.VideoPlayerWebGL.texturePtr]);
        GLctx.texImage2D(GLctx.TEXTURE_2D, 0, GLctx.RGBA, GLctx.RGBA, GLctx.UNSIGNED_BYTE, video);
        video.VideoPlayerWebGL.useMipMap && GLctx.generateMipmap(GLctx.TEXTURE_2D);
        GLctx.pixelStorei(GLctx.UNPACK_FLIP_Y_WEBGL, false);
        GLctx.bindTexture(GLctx.TEXTURE_2D, prevTex);
    }
},
```

Replace them with *GLctx.texImage2D(GLctx.TEXTURE_2D, 0, GLctx.SRGB8_ALPHA8, GLctx.RGBA, GLctx.UNSIGNED_BYTE, video);*. This second option has a huge performance cost though, it is significantly slower. You probably won't notice if you only have to play one video, but if you have to play several you will feel the FPS dipping.

15c) Since version 1.8, if you are using URP, or if you're using Unity 2021.2 or higher with the built-in RP, you can use the *RGB to Linear* shadergraph under the *Assets->MarksAssets->VideoPlayerWebGL->Shaders* folder. If you are using the built-in RP on a Unity version lower than 2021.2, you can use the *RGB to Linear* shader instead, and delete the shadergraph version. This fixes the issue without the performance penalty of the previous solution. Solution 15b) is only necessary if you must use Unity's standard materials for some reason, on top of having to use linear color space.

16. If you are experiencing lag on your video, or if it freezes, it might be due to hardware acceleration. Try disabling it on your browser.