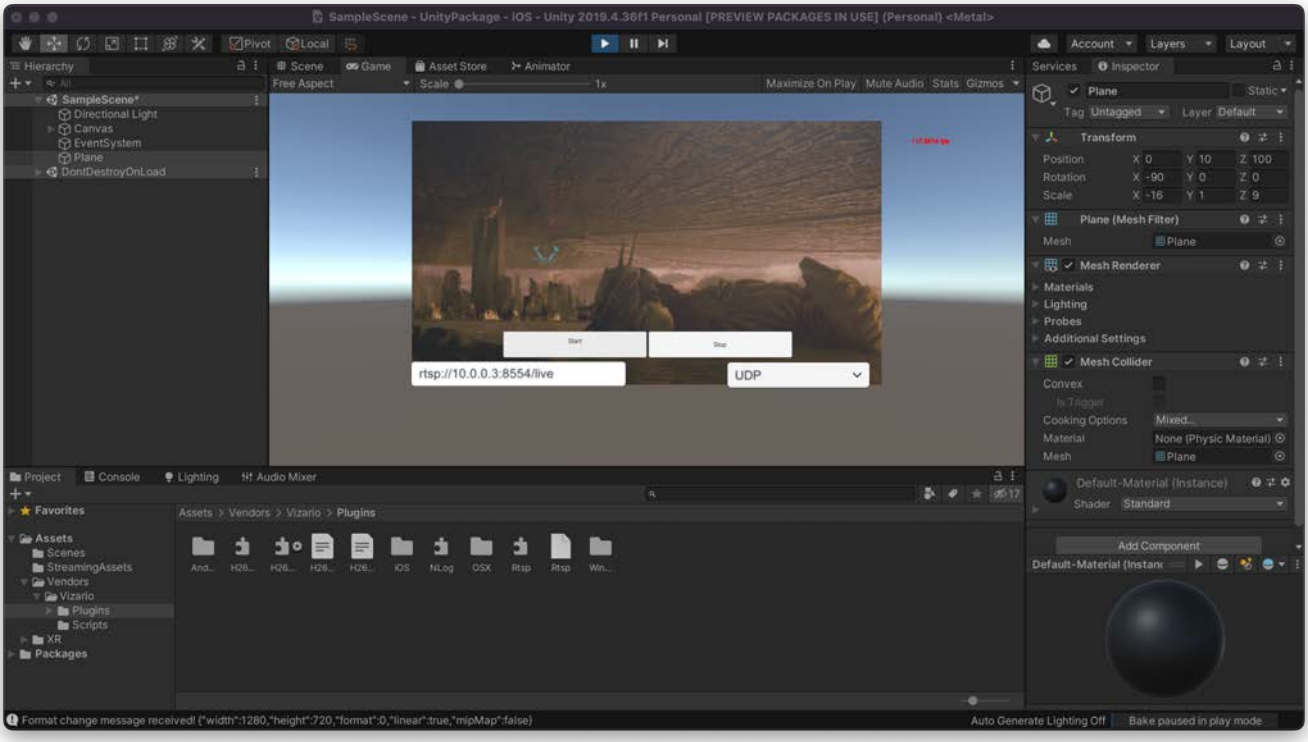



# Vizario H264RTSP Player

Vizario H264RTSP Player is a lightweight library to enable live RTSP streaming of H264/HEVC and AAC bitstreams to iOS or Android devices (respectively standalone apps or Editor Win64 and OSX) from a local PC or remote server. The library allows you to register as an RTSP client to a server and play back the video content within a texture or a Sprite, respectively audio over the native audio device.

The library is developed towards playback of live data and is essentially the playback-counterpart of the video casting asset: https://assetstore.unity.com/packages/tools/video/vizario-h264rtsp-196278

The main purpose of this asset is to enable simple streaming of H264/HEVC/AAC live streams from an RTSP server to a client, decoding und playing back as efficiently as possible.

- It was **NOT** designed to do perfect audio/video synchronization, **NOR** to become a generic player framework like LibVLC. However, the libraries use the native audio devices for audio output (if desired). In other words, the AAC decoder is indeed a full player without the need to mess with device audio setups.
- It has **SOME HEURISTIC BUFFERING**, but this is still experimental. The main purpose of this package is to have the latency to be as low as possible. Without buffering, network congestion is exposed through video stuttering, Audio will likely be unusable. However, latency is essentially ZERO, so you need to decide what makes most sense for you.

To repeat - it is not a regular video player, but it was tested in a variety of scenarios with webcams, DJI drone live video footage and other sources. However, it was built with special care on performance, at the cost of breaking compatibility to other video/audio formats (i.e. MPEG2 or AC3).

The Unity package includes libraries for the following platforms:

- MacOS (intel & silicon)
- Windows x86 and x64
- iOS
- Android (x86-64, armv7, arm64)

Note that the package was tested also on Magic Leap 2, on Meta Quest 2/Pro. It was found to perform well, however, with a plausible performance hit given the effort to decode a video and fill a texture. Here's a matrix on the features tested to work:

| Platform | H264 | HEVC/H265 | AAC |
|---|---|---|---|
| iOS | ✅ | ✅ | ✅ |
| Android | ✅ | ✅ | ✅ |
| MacOS | ✅ | ✅ | ✅ |
| Windows x64 | ✅ | ❌ | ✅ |
| Magic Leap 2 | ✅ | ❓ | ❌ |
| Meta Quest 2/Pro | ✅ | ✅ | ✅ |

Please find an additional FAQ maintained between regular package releases here. Also note that for the examples given here, I use **MY** IP address in my local network or *127.0.0.1*, so the local loopback interface. You need to fix it for your setup.

---

## Example and Getting Started

The best way to get the package to run is to look at the test scenes provided, `SimplePlayer` for a basic example rendering a video on a plane. The most important script used is `H264RTSPPlayerDriver.cs` which essentially contains the entire setup and is just triggered to start and stops the entire engine through button clicks. The entire code should be pretty self-explanatory. You need to enable `unsafe Code` in the Player settings.

There is only one define on top of the script, `LOCAL`. If you are using `#define LOCAL` it just reads a raw H264/HEVC file, respectively an AAC file and passes it on to the decoder - that's it. It is for testing the decoder and display part only. Commenting out `// #define LOCAL` you use the RTSP receiver engine. Therefore you need to enter **YOUR** IP of the caster or server into the URL. If you are receiving data, you should see a Debug message.

It has two modes, which you can see from the Scene. One is using the *Plane* as surface and a *MeshRenderer* component. The other one uses a *SpriteRender* on the *SpriteHolder* game object. You enable or disable those by enabling the GameObject itself, as you will likely only need one of those.

---

## Basic Functionality

This document focuses on the use of the entire engine from the outside, rather than on the internal parts of the library. The assemblies have XML documentation attached with rudimentary info about the intrinsics.

The main class to deal with are `H264Decoder` (and `AACDecoder` for audio) responsible to push the retrieved NAL, SPS/PPC (and AAC) packages to the underlying decoders. Basically these engines can run also with data from local buffers, or even plain files if required. The primary part requiring configuration is the `RTSPClient` which is essentially taken from *SharpRTSP* (see corresponding open source implementation and third-party notes on this).

### RTSP Modes and Requests

Different transport and media request modes are exposed through the aforementioned `RTSPClient`, which should be pretty self explanatory.

```
enum RTP_TRANSPORT
{
UDP,
TCP,
MULTICAST
}
```

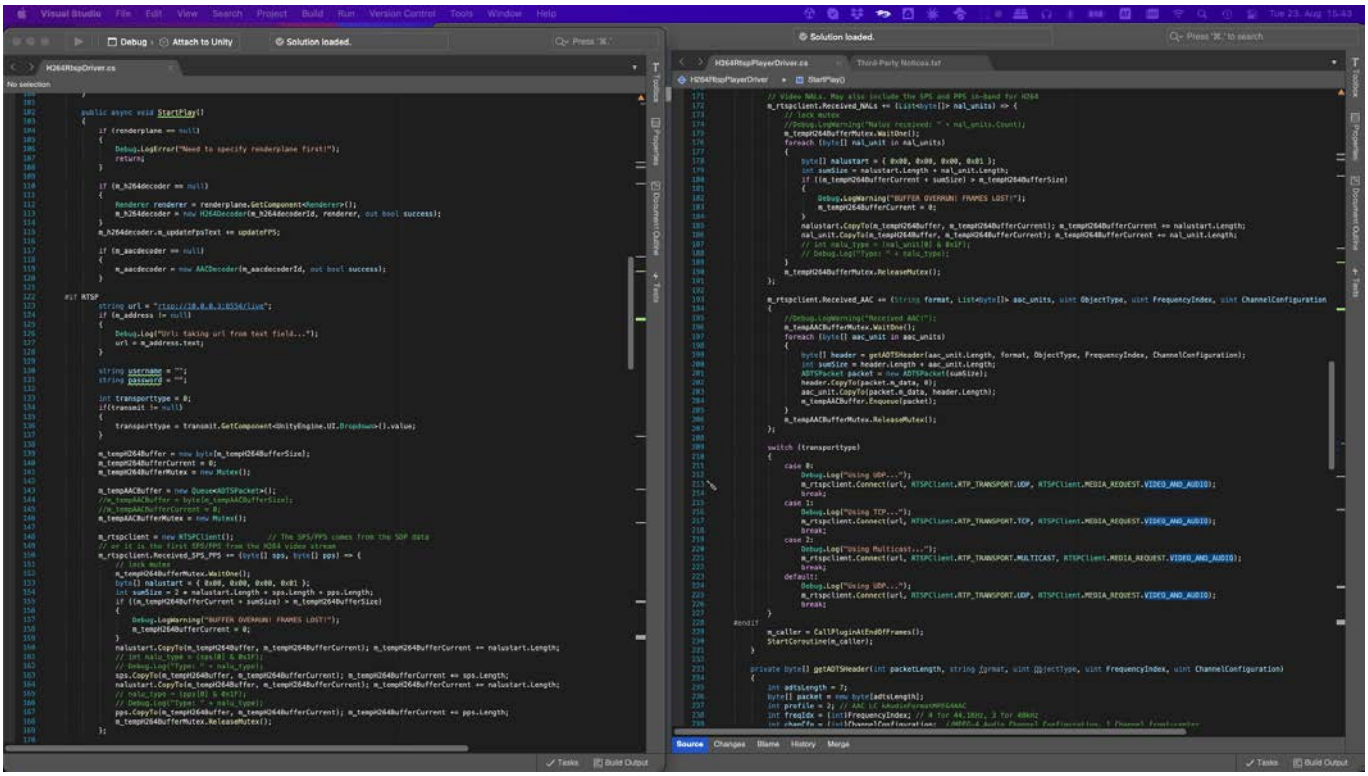
```
enum MEDIA_REQUEST
{
VIDEO_AND_AUDIO,
AUDIO_ONLY,
VIDEO_ONLY
}
```

The primary transmission mode is `UDP` and `VIDEO_AND_AUDIO`. Note that you won't get video or audio, if the stream does not provide either of them.

The main issue with `UDP`is that it cannot traverse network boundaries and there might be several problems regarding the handshake of ports used by the caster and the client. It always has to work with `TCP`, so probably trying with `TCP` first is a good idea.

### Start/Stop/Dispose - Example Code

In order to enable a closer inspection of the workflow, the authors decided not to wrap the functionality of the entire package in a library, but keep the `H264RTSPPlayerDriver.cs` script as is. In the sample scene, this script is attached to the *MainCamera* and triggered by the buttons on the Canvas. The comments in this script should be pretty self-explanatory. As can be seen from the script, callback functions from the `RTSPClient` are used to forward the respective data packets to the individual instances.



## Demo server

A combination of ffmpeg and VLC can create a stream locally like this (watch the IP on your setup!):

```
ffmpeg -i Blindspot.mkv -acodec aac -ar 48000 -vcodec libx264 -preset
ultrafast -crf 20 \
-s hd720 -vf format=yuv420p -profile:v main -f mpegts -
|/Applications/VLC.app/Contents/MacOS/VLC \
-I dummy - --sout='#rtp{sdp=rtsp://192.168.0.40:8554/live.sdp} --sout-all
--sout-keep'
```

For using a dedicated server, we suggest to use a prebuilt version of MediaMTX docker image:

```
docker run --rm -it -e RTSP_PROTOCOLS=tcp -p 8554:8554 -p 1935:1935
aler9/rtsp-simple-server
```

and publish a stream using e.g. ffmpeg. We recently noticed that there might be issues with UDP forwarding using the docker container. If you are

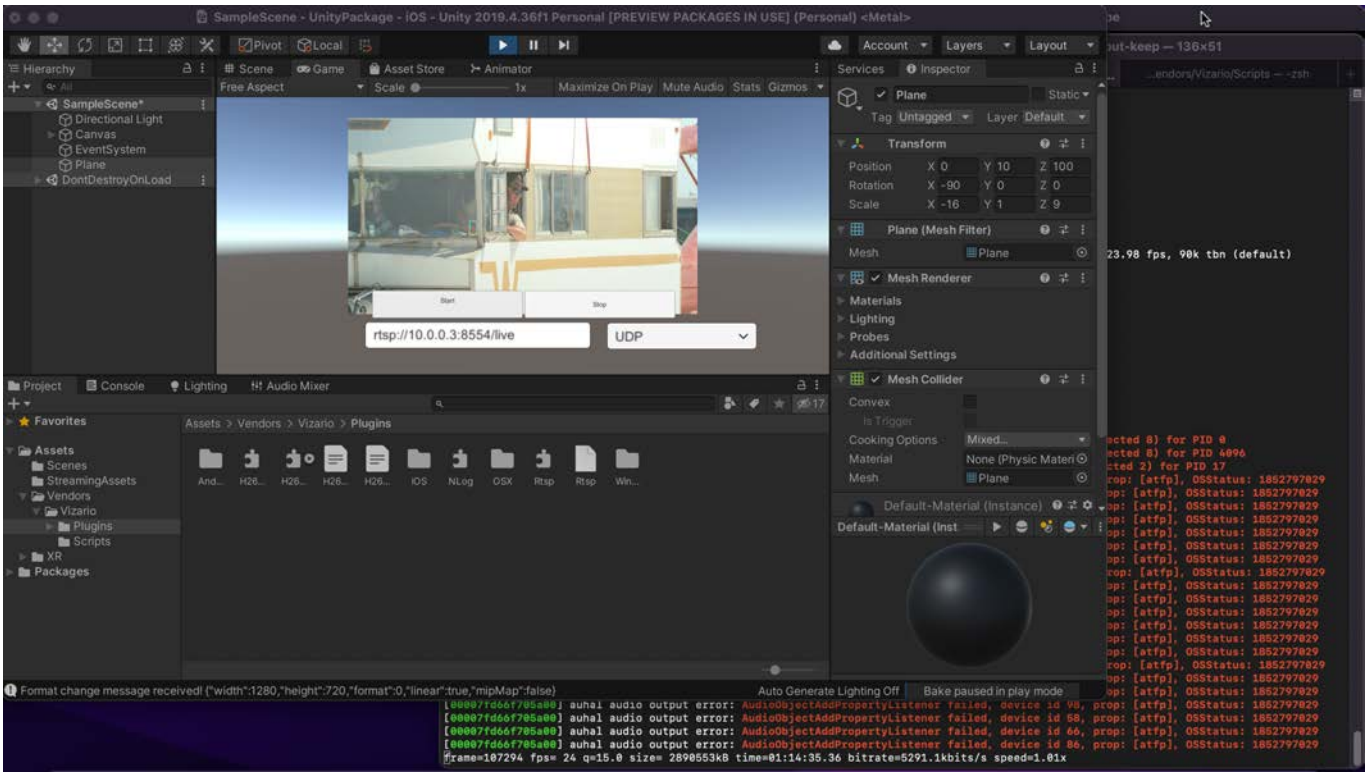
1. in the same subnet
2. want to use UDP

you should probably use the bare-metal executable rather than Docker.

Here's an exemplary call to publish a proper file to MediaMTX:

```
ffmpeg -re -stream_loop -1 -i file.ts -c copy -f rtsp
rtsp://localhost:8554/mystream
```

Make sure that your stream is in the correct `yuv420p` color space, otherwise convert it to be so. Please read the infos here carefully to shape out any problems related to the RTSP protocols, ports or enabled transmission modes.




In order to use H265/HEVC, you need to just set a flag in the `H264RTSPPlayerDriver.cs` script and test it streaming it over this. Note that this call discards B-frames, as they will disturb the received stream.

```
ffmpeg -re -stream_loop -1 -i Cat.mp4 -c:v libx265 -preset medium -x265-
params crf=23:bframes=0 -c:a aac -ar 44100 -b:a 128k -f rtsp
rtsp://localhost:8554/live
```

## Odd Cases

This package provides an **RTSP** streamer and that's it. Therefore please check first that you provide RTSP streams and not RTMP or plain RTP. There is always a way to make things work, but it might require proxying your stream through an engine line MediaMTX.

---

## Compatibility

The libraries should generally run on:

- NET 4.0 / IL2CPP

- iOS Version 13 and above (NO Bitcode support)
- Android API 24 and above
- OSX Unity Editor 2019.4 and standalone
- Win64 Unity Editor 2019.4 and standalone

## FAQs

- Q: The texture is not updated, respectively there is no video played back.
- A: Make sure that the color space used in the video is *yuv420*. The decoders do not support other color spaces.
- Q: The video playback starts late and just becomes *nice* over time.
- A: The decoder needs to wait for the first key frame from the H264 stream. Depending on the stream originator, keyframes are only inserted every few seconds.
- Q: Using HEVC/H265, the video seems to jump forth and back.
- A: Your stream provides B-frames, which are *backward*-correction frames. As the decoder is meant to be run in live mode, you can't play back streams with B-frames (this could become a feature some time, but there was no demand yet).
- Q: Audio and video is not perfectly in sync.
- A: The decoders are individual instances and the streams do not automatically provide timing information, so the decoders take timestamps at the arrival of packets over the network. Depending on the chosen setup, video and audio packets might arrive at slightly different times, so that the output is out of sync slightly.
- Q: Can I use TCP over UDP for my RTSP connection?
- A: Yes you can, but beware that using TCP adds considerably more overhead to the network and is not guaranteed to work (or you might hear audio cracks or see slips in the video to playback slower or faster occasionally, if you don't use buffering). The other aspect is: you MUST use TCP when working across subnets, as there is no routing of UDP packets in RTSP.
- Q: There is no audio.
- A: Make sure that the audio format is indeed *aac* and not *ac3* or anything else. The decoders do not support other audio streams.
- Q: XCode does not find the libraries during compilation for iOS.
- A: Since the switch to UnityFramework compilation for iOS in recent versions of Unity, you need to add all the ***.framework** files to the list of embedded binaries.
- Q: Does it run on the Editor or only on a device?
- A: Yes, it should also work on Windows x64 and OSX Editor, respectively after deploying OSX or Win64 apps (no UWP yet).
- Q: Can I have multiple instances running side by side?
- A: Theoretically yes, but this is largely untested and actually not the intended use right now. Two instance will basically use twice the amount of resources and therefore it will slow down. Make sure that you use a unique ID for each of the *AACDecoder* and *H264Decoder* instances in the respective *H264RTSPDriver* scripts, otherwise it will certainly crash.
- Q: Audio takes a long time to start on OSX and iOS - SHOULD BE MOSTLY FIXED NOW!
- A: This is a problem in the native audio playback. Setting up the routes on the OS level takes that amount of time. If you have audio set up correctly, it should become audible after 1-3 seconds. The setup on the OS level cannot be interrupted! If you stop the stream again, be prepared to wait until the setup is complete and teared down again (in other words, don't stop it if you requested Audio until you hear Audio).

- Additional findings and FAQs can be found on

## Known issues

- The decoding works basically the same on iOS and OSX. There is a set of *dylib* libraries for OSX Editor provided that also enable building a Unity experience for standalone OSX. Unfortunately the memory management (as discussed e.g. here and here) does work differently in the Editor, such that the necessary memory allocation across *C++/C#* boundaries fails, which makes the entire engine partially buggy in the Editor (milage may vary depending on Unity 2019.4.X and 2020.3.X version). However, we are aware of this issue and it should be largely solved.
- There is buffering implemented, but it does not perfectly sync audio and video. I'm working on this right now, but beware that for real-time operation (and I really mean *as fast as possible*) buffering is actually counterproductive and should not be used at all.
- Android is adopted by lots of vendors. Unfortunately, the level of *care* to have hardware-accelerated decoders conforming to certain standards varies a lot. After decoding the first frame from a H264 or HEVC/H265 package the decoder should provide information about the color format. However, this is not always the case, so the output can be anything. There is a flag to enforce a particular format, which you can set in case you only see a gray image instead of a colored image (or don't see anything at all).
- If you are building for a newer Android SDK or iOS SDK, you might run into some trouble with Audio. There have been several API changes in both that require a declaration of the *aim* of using Audio in the C++ level, which was not implemented yet. I did only experience this once building with an iOS16 toolchain, but never saw it again. If you encounter it, please let me know.
- If you build for a more *exotic* platform like Magic Leap 2, it might not work to use the provided *AAR* file, which contains all the libraries for all Android devices. In this case

  1. move the AAR file out to a temporary location, rename it to ZIP and unpack:
  2. copy back the *h264decoder.so* file from the respective platform into the Android folder
  3. configure the platforms in Unity for this file and try to rebuild.

  The same applies if you run into trouble with other Android devices.

## Attribution

This software uses compiled and modified versions of the *SharpRTSP* library to be found here.

## Version History

### 1.0

- Initial version (built with Unity 2019.4.36f1)

### 1.1

- Added Vulkan Rendering for Android

### 1.2

- Fixed AAC stuttering issue on Android
- Added Android x86_64 libraries for Magic Leap
- Added H265 support (OSX/iOS and Android only)

## Support requests

Send any requests to our support team.

---

Apr 7, 2023