ADM Video Streamer

Overview

ADM Video Streamer is a plugin for Unity that allows you to add video live streaming capabilities to your game or app. It is easy to set up and uses hardware acceleration to minimize impact on the performance. The plugin consists of two parts, a native plugin which does all the video processing and a couple of Unity assets to capture the screen or camera and control the native plugin.

System requirements

Windows
NVIDIA Driver version 522.25 or higher.
AMD and Intel minimum driver version is not yet determined.

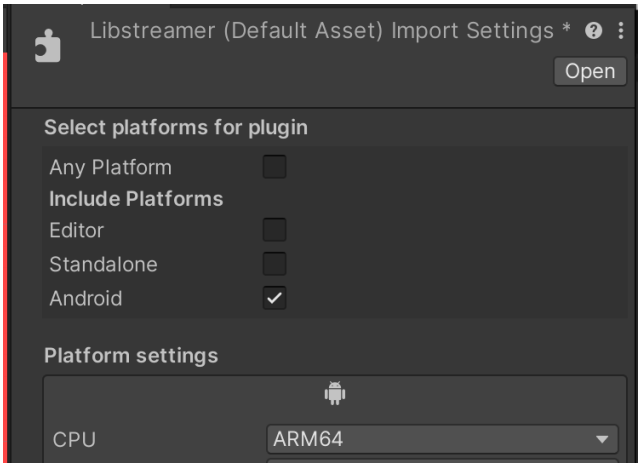
Linux
Vaapi.

Android
ARM64 with MediaCodec support.
Android 10 or higher (this might change with future releases).
Older versions might work, but are not supported.

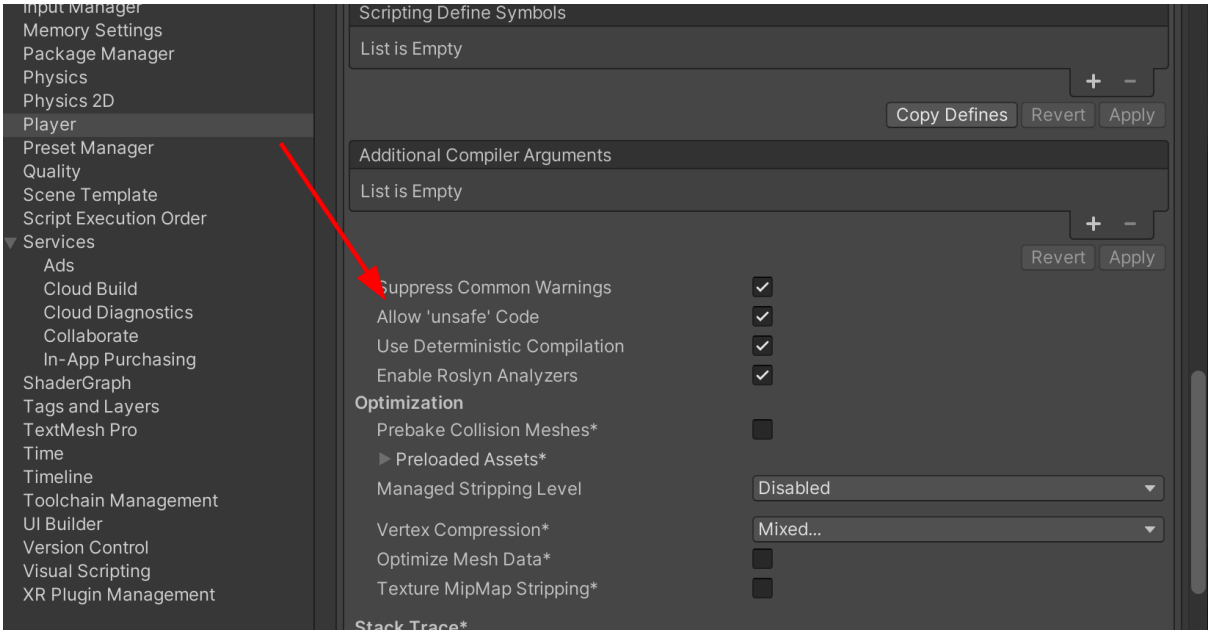
iOS
iOS version 16 or higher (this might change with future releases).
Older versions might work, but are not supported.

Installing the plugin

This plugin can use the FFmpeg libraries to do RTMP streaming. Please be aware that by including FFmpeg you might need to obtain certain licenses, for example (but not limited to) for the use of AAC. Neither ADMStreamer, AdmiraalIT or FFmpeg are covering those licenses. You can compile FFmpeg yourself but for convenience they are also available on this website. Place the .dll, .so or .dylib files that match your platform in the ADMVideoStreamer\Plugins\{platform} directory. When creating a Unity app for multiple platforms you need to configure the library files so that only the correct files are loaded for the target platform. E.g. don't load the Android libraries on the Windows platform.



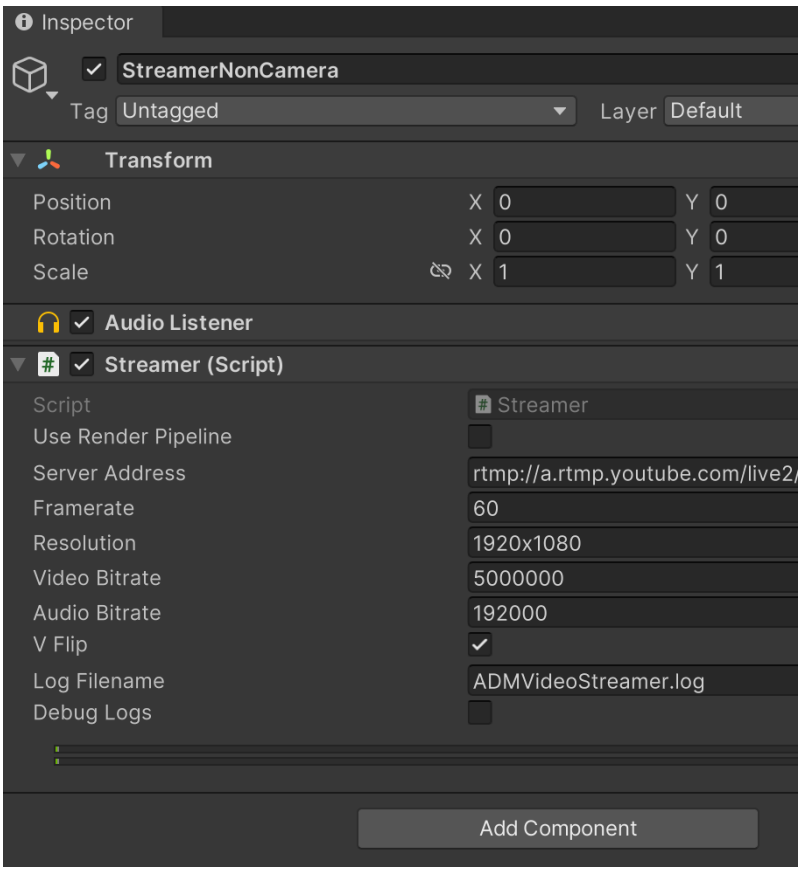
By default Unity does not allow native compiled plugins. You can change this for your Unity project by enabling the ‘Allow unsafe Code’ option in the Player settings.



There are two ways to capture audio and video with the streamer plugin. The following chapters will explain both of them.

Example scene

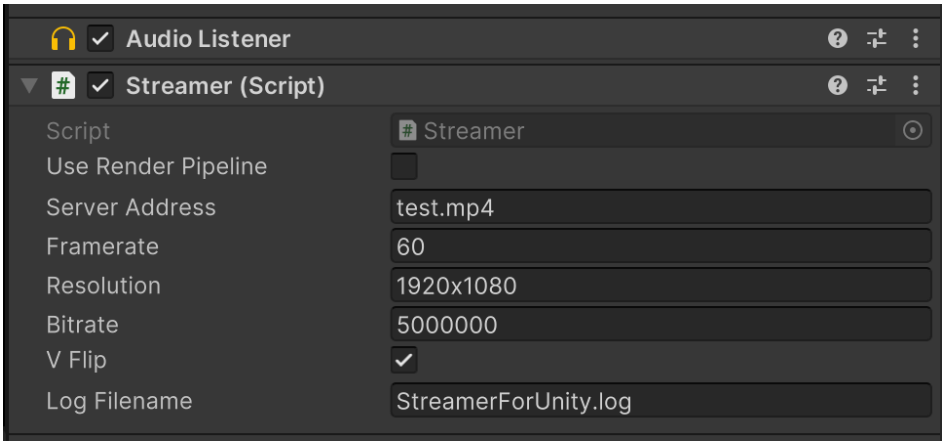
The plugin comes with a tiny example scene. Once the steps from `Installing the plugin` are done, this scene can be used to quickly verify whether the plugin is working correctly. The only thing that needs to be changed in this scene is the `Server Address`. The ADMStreamer plugin is attached to the `StreamerNoCamera` game object.



After setting the Server Address to a valid RTMP source (See the settings chapter for more details), the scene can be run. It will immediately start to stream to the provided server, and it will stop when the scene is closed. Check the `StreamStarter.cs` file for how that is done.

Screen capture

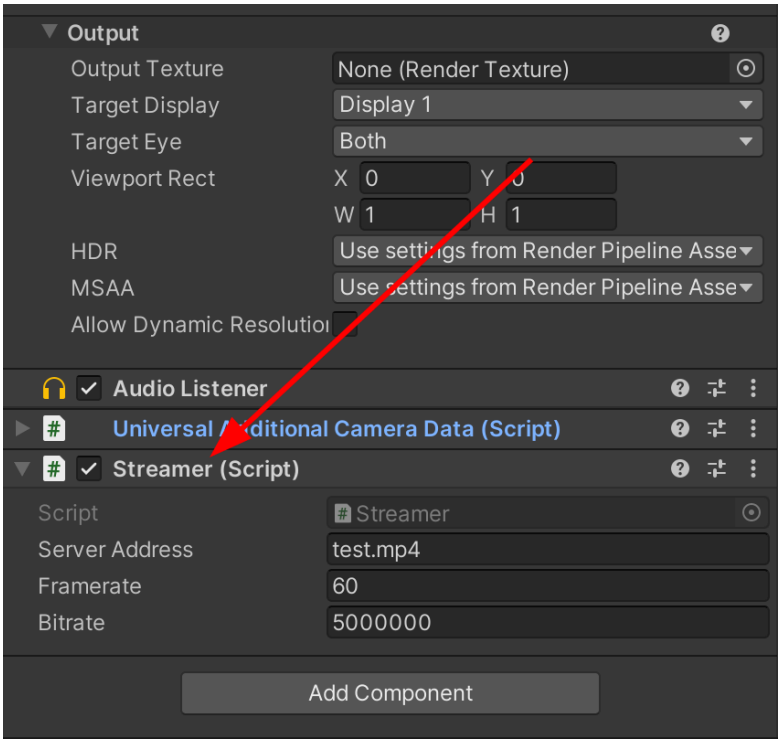
With this option the complete screen is captured. This is the most straightforward method and is a good method to start with. In order to use this you need to add Streamer.cs to the gameObject that has the Audio Listener component attached to it. Usually this will be the main camera.



The settings are explained in the Settings chapter, but other than that the configuration is done. The streamer plugin is now ready to capture video and start streaming.

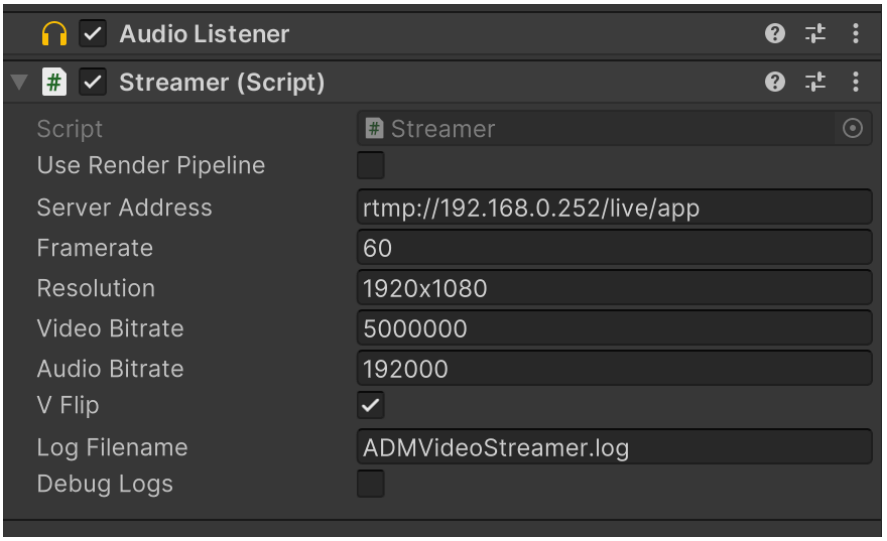
Camera capture with render pipeline

With this capture method a camera is captured by hooking into the render pipeline. There are two use cases for this method. One is to grab the image from a specific camera instead of the image as it's rendered to the screen. If a scene displays multiple cameras at the same time, for example a driving game with a main and rear-mirror camera, it could capture the main camera. Another use case is to capture the image in the middle of the rendering pipeline. For example right before the UI is rendered so the stream contains the full game, without the UI.
This capture method is only available in a project that uses an URP or HDRP render pipeline. In order to use this you can add the Streamer.cs script to a camera and enable the `Use Render Pipeline` checkbox.



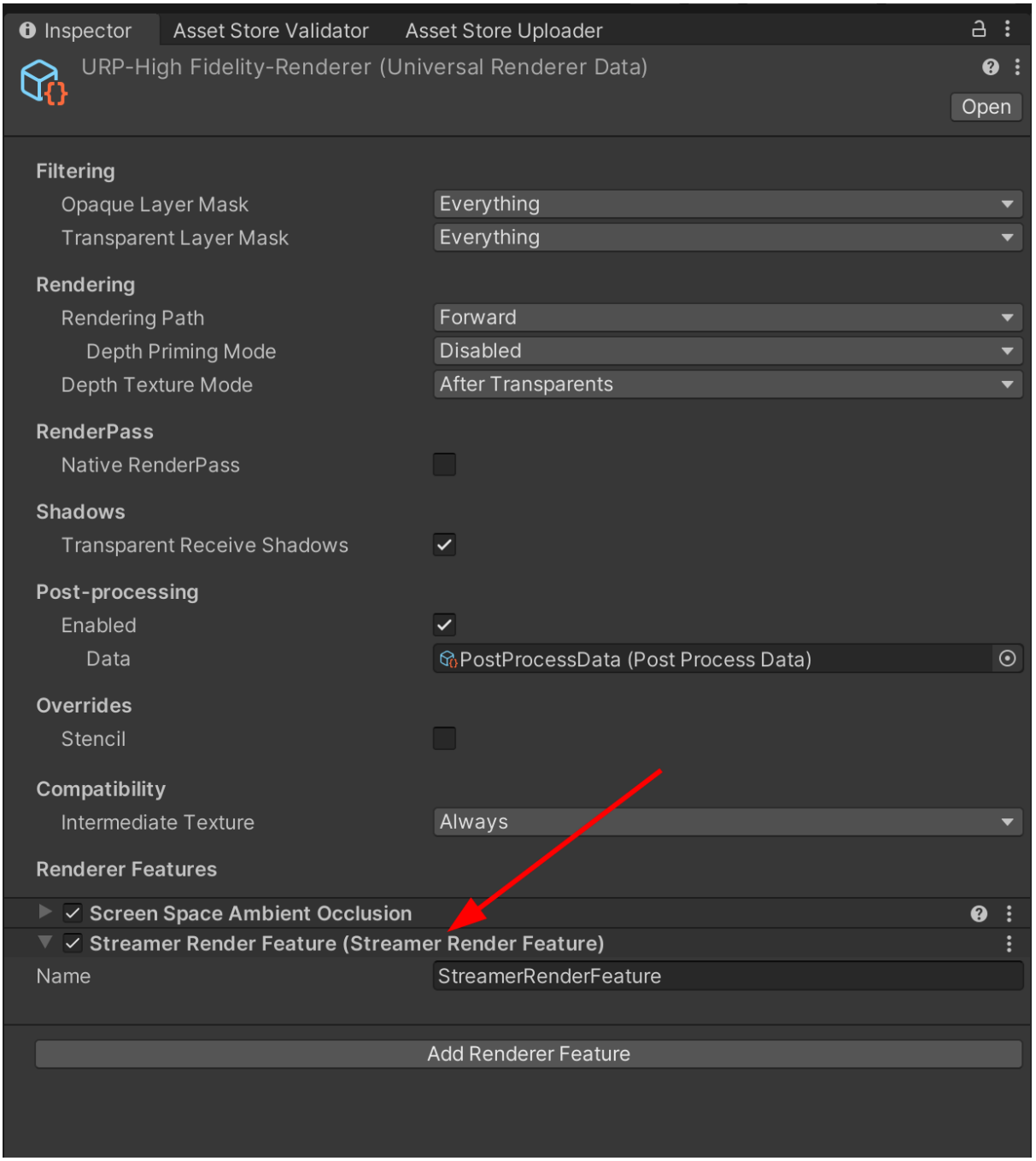
Settings

In this chapter we explain all settings that are supported by Streamer.cs. They can be changed programmatically or via the Inspector:



Use Render Pipeline

When this setting is enabled, the plugin will capture the video during one of the rendering phases of the URP or the HDRP. If this is disabled the Screen capture method will be used. At the time of writing the RP capture will take place after the post processing effects are applied. To make this work you need to add the Streamer render feature to the renderer.



Server Address

The filename or URL to which the plugin will write the video stream. It can be either an MP4 filename, or an RTMP URL. You can provide the stream key as part of the RTMP URL. For example a Youtube RTMP URL will look like this:
`rtmp://a.rtmp.youtube.com/live2/<streamkey>`

Framerate

The target frame rate for the video stream. If the game runs at a higher framerate, the plugin will skip capturing some frames in order to meet the target framerate. This means no unnecessary CPU and GPU cycles are wasted. If the game runs at a lower frame rate, the video stream will use the game framerate.

Resolution

The output resolution for the video stream. The captured video will be scaled to this resolution.

Bitrate

The number of bits per second which is used by the video stream. Setting this number higher will result in better image quality, but requires an internet connection with a higher upload speed.

V Flip

Flips the image vertically. Needed because the Vertical Texture coordinate conventions differ between two types of platforms: Direct3D-like and OpenGL-like. Set this to true if the stream is upside down.

Log filename

The filename to which the plugin will write its debug logs. Please include this file when reporting bugs. When this field is empty the plugin will not write any logs.

Debug Logs

When enabled, the logs will be flushed to disk after each line. This can have a negative impact on performance, but is very useful when debugging crashes.

Methods

Starting the stream

There is a StartStreaming and StopStreaming method available on Streamer.cs to control the start and stop of the video stream. This is a small example on how you could use these methods:

```
private AdmiraalIT.Video.Streamer streamer;
void OnEnable()
{
  streamer = FindObjectOfType<AdmiraalIT.Video.Streamer>();
  streamer.StartStreaming();
}


void OnDisable()
{
  streamer.StopStreaming();
}
```

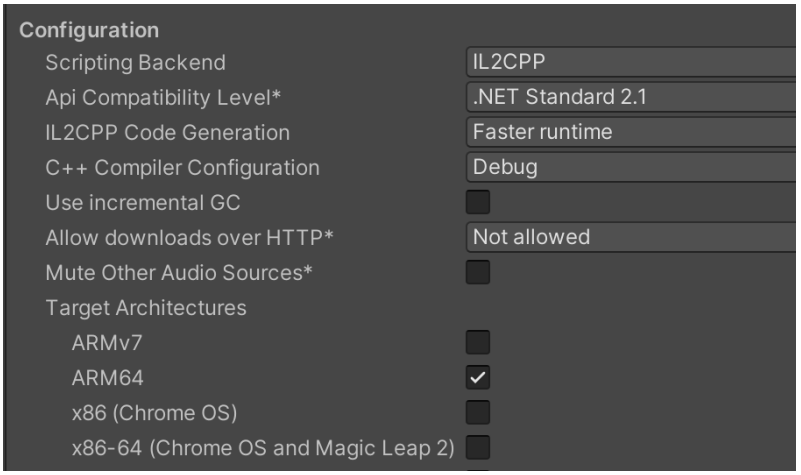
Getting status information

There is a GetStatus method available on Streamer.cs which allows you to retrieve the current status of the native dll. It will return the connection state and the current bitrate. The actual bitrate will be slightly different from the bitrate that is configured due to how H264 video encoders work.

Platform specific

Android

Currently the Android version of the plugin is only compatible with the ARM64 architecture. Please contact me if you need a different architecture. The minimum version which is tested is Android 10. Lower versions might work but are not officially supported as I cannot test them. Unity only supports IL2CPP when compiling for ARM64, so please configure the scripting backend and target architectures like this:



Internet Access

In order for the app or game to be able to stream over the internet, it needs to have the internet permission configured. This can be done by creating an AndroidManifest.xml file in the Assets/Plugins/Android directory. This is an example of what that file could contain:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.yourcompany.yourapp">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application>
        <!-- Your other configurations -->
    </application>
</manifest>
```

Common issues

DllNotFoundException

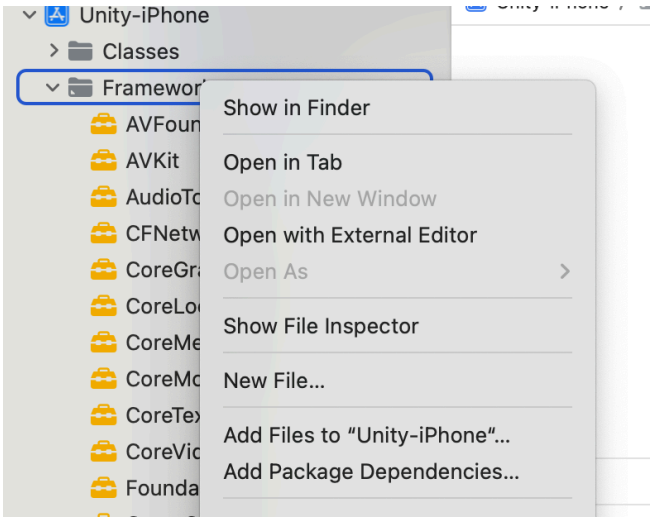
```
DllNotFoundException: Unable to load DLL 'streamer'. Tried the load the
following dynamic libraries: Unable to load dynamic library 'streamer'
because of 'Failed to open the requested dynamic library (0x06000000)
dlerror() = dlopen failed: library "streamer" not found
```

This error can have many causes.
Check whether libstreamer.so and the FFmpeg .so files are placed in the Plugins directory.
Check the Platform settings of the files in the Plugins directory.
Check whether the target architecture is set to ARM64 in the Player settings.

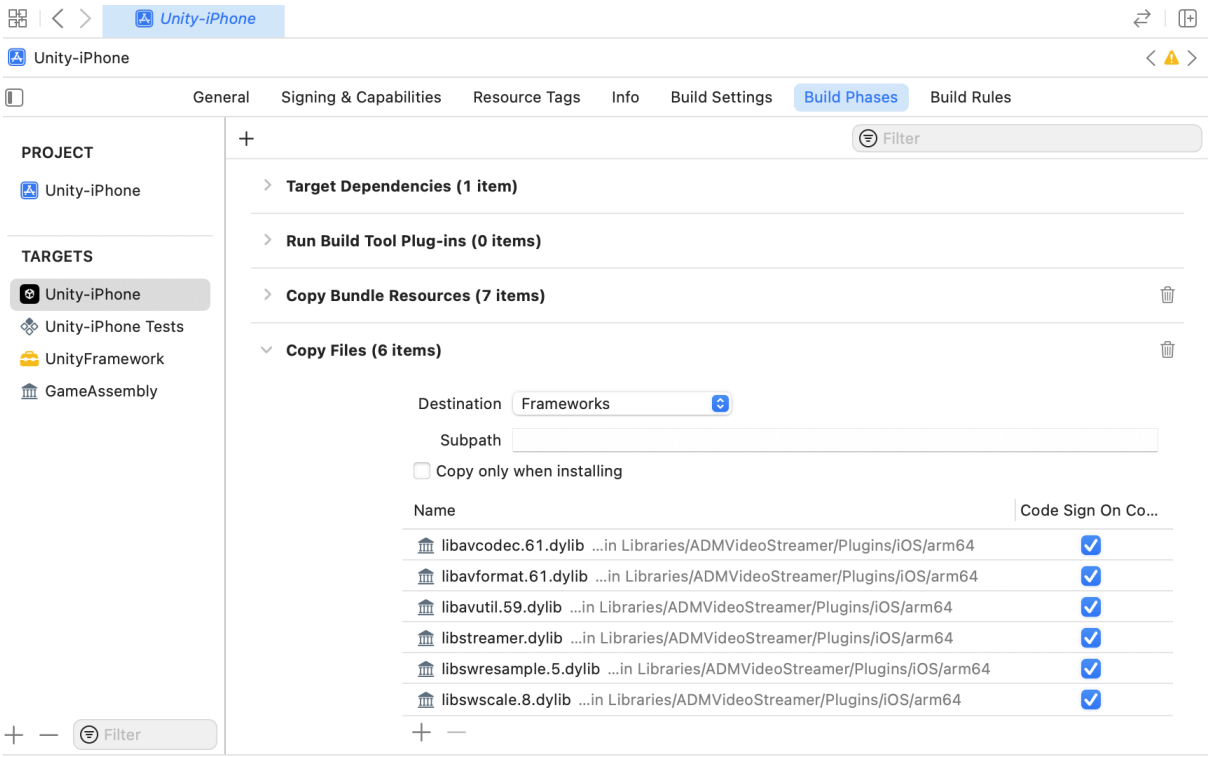
If the problem is still there, trigger a clean build in the build settings.

iOS

Currently there is a bug in Unity versions earlier than v6: UUM-72675. Because of this, the .dylib files need to be added to the Frameworks folder manually:



They also need to be added to a ‘Copy Files’ build phase:



Knows issues

- When streaming to Youtube, the stream sometimes does not start correctly. Improvements will be made in a future version.
- iOS bitrate is fluctuating a lot. I need to investigate if there is anything I can do about this as it seems other apps have the same behavior.