Piping to ffmpeg.exe with C#

28 Jul 2017 - 5 min read

ffmpeg

FFMpeg is a great tool for doing all kinds of stuff with media. In this post, I will demonstrate how images and audio can be piped to **ffmpeg.exe** from C#.

We would run **ffmpeg.exe** using the `System.Diagnostics.Process` class. We would use parameters `UseShellExecute = false` and `CreateNoWindow = true` so that the command line window does not show up.

Piping Images

When we pipe images, we use the `image2pipe` format. Input Framerate is required to make things work expected.

Piping Audio

When piping audio specify a format like `s16le` which stands for 16-bit Stereo Low-endian PCM data. Input Frequency and Channels can be specified using `-ar` and `-ac` respectively.

Piping Raw Video

When piping raw video, i.e. pixel data of images, we use the `rawvideo` format. Now, more things need to be done here. You need to specify the Pixel Format, e.g. `-pix_fmt bgr32`. Also, since you are piping raw data, ffmpeg cannot figure out the video size by itself. So, you need to specify that like `-video_size WIDTHxHEIGHT` replacing WIDTH and HEIGHT with the respective values. Also, input framerate is required to make things work expected.

Piping single stream

Suppose you have just a single stream to pipe, either audio or video. We can redirect the Standard Input to get the job done. And in the ffmpeg arguments, we use `-i -` for input to indicate standard input.

Let’s see an example encoding Raw Video to Mp4 (x264).

```
using System.Diagnostics;

var inputArgs = "-framerate 20 -f rawvideo -pix_fmt rgb32 -video_size 1920x1080 -i -";
var outputArgs = "-vcodec libx264 -crf 23 -pix_fmt yuv420p -preset ultrafast -r 20
out.mp4";

var process = new Process
{
    StartInfo =
    {
        FileName = "ffmpeg.exe",
        Arguments = $"{inputArgs} {outputArgs}",
        UseShellExecute = false,
        CreateNoWindow = true,
        RedirectStandardInput = true
    }
};

process.Start();

var ffmpegIn = process.StandardInput.BaseStream;

// Write Data
ffmpegIn.Write(Data, Offset, Count);

// After you are done
ffmpegIn.Flush();
ffmpegIn.Close();

// Make sure ffmpeg has finished the work
process.WaitForExit();
```

Piping multiple streams

While piping multiple streams, things get a bit complicated. We have to create Named Pipes using `System.IO.Pipes.NamedPipeServerStream` as standard input can only be used if we have to pipe only a single input. ffmpeg reads all inputs one by one. So, writing of the streams should remain independent of each other or else ffmpeg might freeze. An easy way to do this is to male the pipes asynchronous and write asynchronously into them.

Example for creating a named pipe, Let's name it ffpipe.

```
// Make it asynchronous. 10,000 is buffer size, make sure it is big enough for your
requirement.
var pipe = new NamedPipeServerStream("ffpipe", PipeDirection.Out, 1,
PipeTransmissionMode.Byte, PipeOptions.Asynchronous, 10000, 10000);
```

For the input, we use `-i \\.\pipe\ffpipe` in context of the above example. Creating the process is same as for single stream.

Before you write to a pipe, make sure it is connected.

```
pipe.WaitForConnection();
```

Create as many pipes as neccessary. Write to them asynchronously.

```
pipe.WriteAsync(Buffer, Offset, Count);
```

After you are done, dispose the pipe.

```
pipe.Flush();
pipe.Dispose();
```

Logging

Sometimes, things might not work as expected. In those cases it is useful to see the output ffmpeg shows when used on the command line. It can be accessed by redirected the standard error.

Here's an example which reads the output using events.

```
var process = new Process
{
    StartInfo =
    {
        FileName = "ffmpeg.exe",

        // Replace Command line arguments here.
        Arguments = Arguments,

        UseShellExecute = false,
        CreateNoWindow = true,
        RedirectStandardInput = true,

        // Redirect FFMpeg output.
        RedirectStandardError = true
    },

    // Get notified when ffmpeg writes to error stream.
    EnableRaisingEvents = true
};

// Event handler to receive written data.
process.ErrorDataReceived += (s, e) => ProcessTheErrorData();

process.Start();

// Start reading error stream.
process.BeginErrorReadLine();
```

Share this:

tomaszzmuda • 2 years ago

I strongly discourage using ProcessInfo to use FFmpeg. It seems to be the easiest way but if you need additional stuff like output only valuable information or catch errors you will have a problem. Take a look on https://xabe.net/product/xa.... It is free (for non commercial use) wrapper for C# allowing to use almost all FFmpeg functionality.

1 | 1 • Reply • Share ›

ayyD33p tomaszzmuda • a year ago

ffmpeg returns perfectly understandable error on closing the redirected pipe if there was any using pipes is pretty valid way of calling ffmpeg if you know your parameters/what you're doing

you also "forgot" to mention that you are the author is the above product which would also add a little bit of credibility to your advertisement

this blog helped me a great deal

1 | • Reply • Share ›

alexrainman • 5 months ago

I know the post is old but i am lost in this part ffmpegIn.Write(Data, Offset, Count);

| • Reply • Share ›

alexrainman alexrainman • 5 months ago

any help?

| • Reply • Share ›

Mohammad Kamrul Hasssan • a year ago

i need some help on the piping issue, can you please help me out? if so then how can i contact you via email so that i can share my code with you?

| • Reply • Share ›