# LEARNING ffmpeg

#ffmpeg

# Table of Contents

About 1

Chapter 1: Getting started with ffmpeg 2

Remarks 2

Examples 2

Installation or Setup 2

OS X 2

Windows 2

Unix 3

What is FFmpeg? 3

Chapter 2: Decoding 4

Introduction 4

Examples 4

Find a stream 4

Open a codec context 4

Decode frames 5

Chapter 3: Ffmpeg Restream 7

Examples 7

Simple Device Restream 7

Chapter 4: Reading Media 8

Introduction 8

Examples 8

Reading from memory 8

Reading from a file 9

Reading from a format context 9

Reading from an IStream in a IOContext 9

Seeking within an IStream in an IOContext 10

Credits 11

# About

# Chapter 1: Getting started with ffmpeg

## Remarks

FFMpeg This section provides an overview of what ffmpeg is, and why a developer might want to use it.

It should also mention any large subjects within ffmpeg, and link out to the related topics. Since the Documentation for ffmpeg is new, you may need to create initial versions of those related topics.

## Examples

### Installation or Setup

FFmpeg can be installed on a mixture of Operating Systems, including Unix and OS X. Using a command line extension, this utility can also be installed on Windows using a dll.

## OS X

To install this utility on OS X, just head over to ffmpeg.org, download the release relative to your Macs architecture (instructions on finding this can be found here). Then put the application into an accessible directory and run it from command line.

Another way is using HomeBrew: https://www.howtogeek.com/211541/homebrew-for-os-x-easily-installs-desktop-apps-and-terminal-utilities/

For example

```
brew install ffmpeg --with-fdk-aac --with-ffplay --with-libass --with-libvorbis --with-libvpx
--with-rtmpdump --with-openh264 --with-tools
```

## Windows

To install this utility on Windows, head over to [ffmpeg.org](https://www.ffmpeg.org/download.html#build-windows) and follow the download link, using your architecture. Instructions on finding this can be seen [here](http://answers.microsoft.com/en-us/windows/forum/windows_7-hardware/i-need-to-know-how-to-determine-my-processors/3ede9c69-25f5-427b-8e8d-e9dd2d032d22). Then place the downloaded software into an accessible directory and run from command line.

## Unix

To install this utility on Unix, just follow the instructions found at [ffmpeg.org](https://www.ffmpeg.org/download.html#build-linux)

To check if ffmpeg is installed correctly and see a list of available commands try running the following command in the command line:

```
ffmpeg -help
```

### What is FFmpeg?

FFmpeg (or "Fast Forward MPEG") is a simple yet feature rich command line utility to allow the manipulation (including decoding, encoding, transcoding, muxing, demuxing, streaming, filtering, and playing) of media and video files. This utility differs from other GUI orientated software as it employs the WYSIWYG methodology (What You See Is What You Get). Instead of hidden away menus and features, everything can be found by just typing `ffmpeg -h` when set up, or following the comprehensive documentation. In addition to the command line tool there are a number of C libraries (which it uses), that can be used to bring the functionality of FFmpeg into other projects. The documentation for the libraries includes many examples to help get you started.

Read Getting started with ffmpeg online: https://riptutorial.com/ffmpeg/topic/4935/getting-started-with-ffmpeg

# Chapter 2: Decoding

## Introduction

Getting the raw video/audio from encoded media streams.

## Examples

### Find a stream

Media stream containers usually have a several streams, such as a video stream and an audio stream. For example, you can get the audio stream using the following:

```
// A Format Context - see Reading Data for more info
AVFormatContext *formatContext;

// Inspect packets of stream to determine properties
if (avformat_find_stream_info(formatContext, NULL) < 0){
    // Error finding info
}

// Find the stream and its codec
AVCodec* audioCodec;
int audioStreamIndex = av_find_best_stream(
    formatContext,        // The media stream
    AVMEDIA_TYPE_AUDIO,   // The type of stream we are looking for - audio for example
    -1,                   // Desired stream number, -1 for any
    -1,                   // Number of related stream, -1 for none
    &audioCodec,          // Gets the codec associated with the stream, can be NULL
    0                     // Flags - not used currently
);
if(audioStreamIndex = AVERROR_STREAM_NOT_FOUND || !audioCodec){
    // Error finding audio (ie. no audio stream?)
}
```

To get other types of streams, you just need to replace the type of stream. The following are valid types:

```
AVMEDIA_TYPE_VIDEO,
AVMEDIA_TYPE_AUDIO,
AVMEDIA_TYPE_SUBTITLE,
AVMEDIA_TYPE_DATA,        // Usually continuous
AVMEDIA_TYPE_ATTACHMENT,  // Usually sparse
```

### Open a codec context

Once you have a stream Format Context and its respective Codec, you can open it for decoding using the following code:

```
// The format context and codec, given - see Find a stream for how to get these
```

```
AVFormatContext *formatContext;
AVCodec* codec;
int streamIndex;

// Get the codec context
AVCodecContext *codecContext = avcodec_alloc_context3(codec);
if (!codecContext){
    // Out of memory
    avformat_close_input(&formatContext);
}

// Set the parameters of the codec context from the stream
int result = avcodec_parameters_to_context(
    codecContext,
    formatContext->streams[streamIndex]->codecpar
);
if(result < 0){
    // Failed to set parameters
    avformat_close_input(&formatContext);
    avcodec_free_context(&codecContext);
}

// Ready to open stream based on previous parameters
// Third parameter (NULL) is optional dictionary settings
if (avcodec_open2(codecContext, codec, NULL) < 0){
    // Cannot open the video codec
    codecContext = nullptr;
}

// Do something with the opened codec context... (ie decode frames through the context)
```

## Decode frames

Given a codec context and encoded packets from a media stream, you can start decoding media into raw frames. To decode a single frame, you can use the following code:

```
// A codec context, and some encoded data packet from a stream/file, given.
AVCodecContext *codecContext;  // See Open a codec context
AVPacket *packet;              // See the Reading Media topic


// Send the data packet to the decoder
int sendPacketResult = avcodec_send_packet(codecContext, packet);
if (sendPacketResult == AVERROR(EAGAIN)){
    // Decoder can't take packets right now. Make sure you are draining it.
}else if (sendPacketResult < 0){
    // Failed to send the packet to the decoder
}

// Get decoded frame from decoder
AVFrame *frame = av_frame_alloc();
int decodeFrame = avcodec_receive_frame(codecContext, frame);

if (decodeFrame == AVERROR(EAGAIN)){
    // The decoder doesn't have enough data to produce a frame
    // Not an error unless we reached the end of the stream
    // Just pass more packets until it has enough to produce a frame
    av_frame_unref(frame);
    av_freep(frame);
}else if (decodeFrame < 0){
    // Failed to get a frame from the decoder
    av_frame_unref(frame);
    av_freep(frame);
}

// Use the frame (ie. display it)
```

If you want to decode *all* frames, you can simply place the previous code in a loop, feeding it consecutive packets.

Read Decoding online: https://riptutorial.com/ffmpeg/topic/10090/decoding

# Chapter 3: Ffmpeg Restream

## Examples

### Simple Device Restream

Ffmpeg is a swiss knife for streaming project. For any kind of device streaming you only need to get the specification of device. To list the device

```
ffmpeg -f dshow -list_devices true  -i dummy
```

Command prompt will list all the aviable device on machine.

```
[dshow @ 0000000004052420] DirectShow video devices
[dshow @ 0000000004052420]  "ManyCam Virtual Webcam"
[dshow @ 0000000004052420]  "UScreenCapture"
[dshow @ 0000000004052420] DirectShow audio devices
```

For restreaming the audio&video device,

```
ffmpeg -f dshow -i video="DirectShow video devices":audio="DirectShow audio devices"
-vcodec libx264 -acodec aac -strict experimental 2 -tune zerolatency -f flv
rmtp://WOWZA_IP/WOWZA_APP/STREAMNAME
```

This can be extended all kind of device like medical devices or video hardware.

Read Ffmpeg Restream online: https://riptutorial.com/ffmpeg/topic/9517/ffmpeg-restream

# Chapter 4: Reading Media

## Introduction

There are a few ways to read Audio/Video into FFmpeg.

## Examples

### Reading from memory

libavformat usually takes in a file name and reads media directly from the filesystem. If you want to read from memory (such as streams), do the following:

```
// Define your buffer size
const int FILESTREAMBUFFERSZ = 8192;


// A IStream - you choose where it comes from
IStream* fileStreamData;

// Alloc a buffer for the stream
unsigned char* fileStreamBuffer = (unsigned char*)av_malloc(FILESTREAMBUFFERSZ);
if (fileStreamBuffer == nullptr){
    // out of memory
}

// Get a AVContext stream
AVIOContext* ioContext = avio_alloc_context(
    fileStreamBuffer,    // Buffer
    FILESTREAMBUFFERSZ,  // Buffer size
    0,                   // Buffer is only readable - set to 1 for read/write
    fileStreamData,      // User (your) specified data
    FileStreamRead,      // Function - Reading Packets (see example)
    0,                   // Function - Write Packets
    FileStreamSeek       // Function - Seek to position in stream (see example)
);
if(ioContext == nullptr){
    // out of memory
}

// Allocate a AVContext
AVFormatContext *formatContext = avformat_alloc_context();

// Set up the Format Context
formatContext->pb = ioContext;
formatContext->flags |= AVFMT_FLAG_CUSTOM_IO; // we set up our own IO

// Open "file" (open our custom IO)
// Empty string is where filename would go. Doesn't matter since we aren't reading a file
// NULL params are format and demuxer settings, respectively
if (avformat_open_input(&formatContext, "", nullptr, nullptr) < 0){
    // Error opening file
}
```

```
// Do something with the formatContext

// Free resources!
avformat_close_input(&formatContext);
av_free(ioContext);
```

## Reading from a file

Opening a media file from the local file system.

```
AVFormatContext *formatContext;

// Open the file
if(avformat_open_file(&formatContext, "path/to/file.ogg", NULL, NULL) < 0){
    // Error opening file
}

// Do something with the file

// Free resources
avformat_close_input(&formatContext);
```

## Reading from a format context

Formats contain one or more encoded and muxed streams. We usually read these in chunks, which are often called frames (though in certain cases, FFmpeg refers exclusively to decoded, raw media chunks as frames, and encoded chunks as packets, which may be confusing). To read a single frame from a format, use the following:

```
// A Format Context - see other examples on how to create it
AVFormatContext *formatContext;

// Initialize the AVPacket manually
AVPacket avPacket;
av_init_packet(&avPacket); // set fields of avPacket to default.
avPacket.data = NULL;
avPacket.size = 0;

// Read from the context into the packet
if(av_read_frame(formatContext, &avPacket) == 0){
    // nothing read
}

// Use the packet (such as decoding it and playing it)

// Free packet
av_packet_unref(&avPacket);
```

## Reading from an IStream in a IOContext

The API call `avio_alloc_context`, which sets up a custom IO context, takes in a pointer to a Read function. If you are reading from an IStream, you can use the following:

```
/**
 * Reads from an IStream into FFmpeg.
 *
 * @param ptr       A pointer to the user-defined IO data structure.
 * @param buf       A buffer to read into.
 * @param buf_size  The size of the buffer buff.
 *
 * @return The number of bytes read into the buffer.
 */
int FileStreamRead(void* ptr, uint8_t* buf, int buf_size)
{
    // This is your IStream
    IStream* stream = reinterpret_cast<IStream*>(ptr);

    ULONG bytesRead = 0;
    HRESULT hr = stream->Read(buf, buf_size, &bytesRead);
    if(hr == S_FALSE)
        return AVERROR_EOF;  // End of file

    if(FAILED(hr))
        return -1;
    return bytesRead;
}
```

## Seeking within an IStream in an IOContext

The API call `avio_alloc_context`, which sets up a custom IO context, takes in a pointer to a **Seek** function. If you are reading from an IStream, you can use the following:

```
/**
 * Seeks to a given position on an IStream.
 *
 * @param ptr     A pointer to the user-defined IO data structure.
 * @param pos     The position to seek to.
 * @param origin  The relative point (origin) from which the seek is performed.
 *
 * @return  The new position in the IStream.
 */
int64_t FileStreamSeek(void* ptr, int64_t pos, int origin){
    // Your custom IStream
    IStream* stream = reinterpret_cast<IStream*>(ptr);

    // Prevent overflows
    LARGE_INTEGER in = { pos };
    ULARGE_INTEGER out = { 0 };

    // Origin is an item from STREAM_SEEK enum.
    //   STREAM_SEEK_SET - relative to beginning of stream.
    //   STREAM_SEEK_CUR - relative to current position in stream.
    //   STREAM_SEEK_END - relative to end of stream.
    if(FAILED(stream->Seek(in, origin, &out)))
         return -1;

    // Return the new position
    return out.QuadPart;
}
```

Read Reading Media online: https://riptutorial.com/ffmpeg/topic/10089/reading-media

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with ffmpeg | Abex, Chris, Community, drumkruk, Olga Khylkouskaya, RhysO |
| 2 | Decoding | JCOC611 |
| 3 | Ffmpeg Restream | Emre Karataşoğlu |
| 4 | Reading Media | JCOC611 |