Transcoding with FFmpeg

Coding Technology Document

- FFMPEG를 활용한 Visual Studio 내 Transcoding–

2017.01.10

작성자 : 김문원, 황진하

목차

[1. 개요 2](#_Toc471981098)

[2. 시스템 환경 2](#_Toc471981099)

[3. 환경 설정 2](#_Toc471981100)

[3.1 FFMpeg Compile (Windows) 2](#_Toc471981101)

[3.2 Visual Studio 환경 설정 방법 4](#_Toc471981102)

[4. 코딩 5](#_Toc471981103)

[4.1 Header Include 5](#_Toc471981104)

[4.2 Library 초기화 하기 5](#_Toc471981105)

[4.3 File Input&Output 6](#_Toc471981106)

[4.4 Stream Demuxing 8](#_Toc471981107)

[4.5 Decoding 11](#_Toc471981108)

[4.6 Encoding 12](#_Toc471981109)

[5. 결과 15](#_Toc471981110)

[5.1 Genenral Result 15](#_Toc471981111)

[5.2 Video Result 16](#_Toc471981112)

[5.3 Audio Result 17](#_Toc471981113)

[6. 참고문헌 17](#_Toc471981114)

[7. 부록 18](#_Toc471981115)

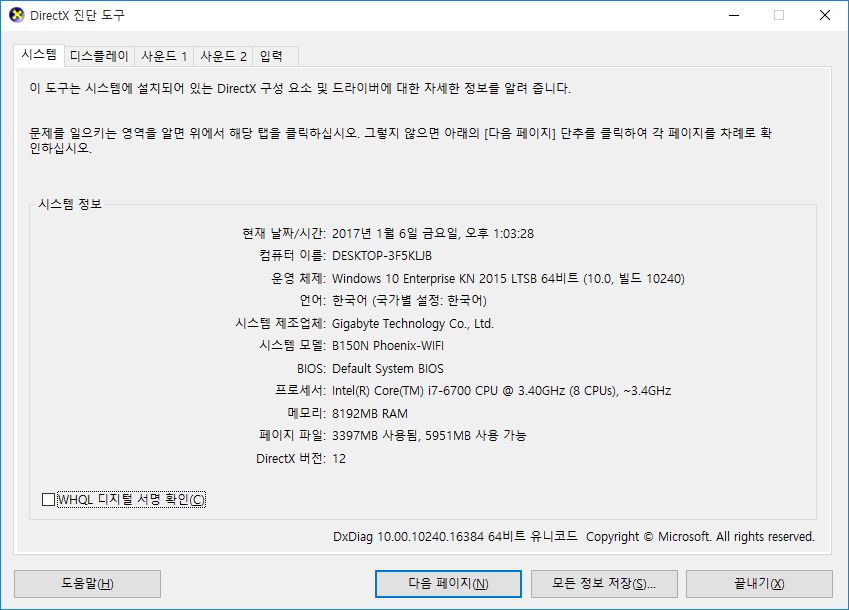
# 개요

본 문서에서는 FFMPEG의 라이브러리를 이용해 Window 환경에서 Media file의 Decoding후, 다시 Encoding을 하는 Transcoding 과정을 다룬다. 과정은 소스코드를 중심으로 설명하며, 자세한 정보와 오류보고는 하단의 부록에 포함된다.

# 시스템 환경

Microsoft Visual Studio 2015 (v140)

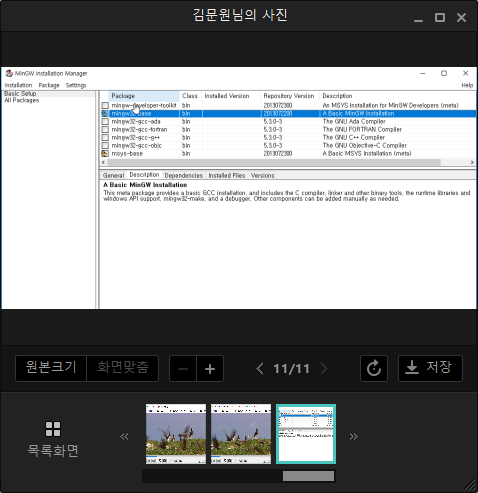
FFMPEG 2.7.1



# 환경 설정

## FFMpeg Compile (Windows)[[1]](#footnote-1)

* + - Windows로 포팅한 GNU 소프트웨어 도구 모음인 MinGW를 설치한다.
* yasm 어셈블러를 windows용으로 다운로드 후 MinGW설치 경로의 bin폴더 “yasm.exe”로 이름 변경 후 저장
* MinGW 설치 매니저 실행 시, “Mingw32-base” 와 “msys-base” 패키지를 체크한 후 설치한다.



< MinGW 설치 매니저 실행 시 체크 설정>

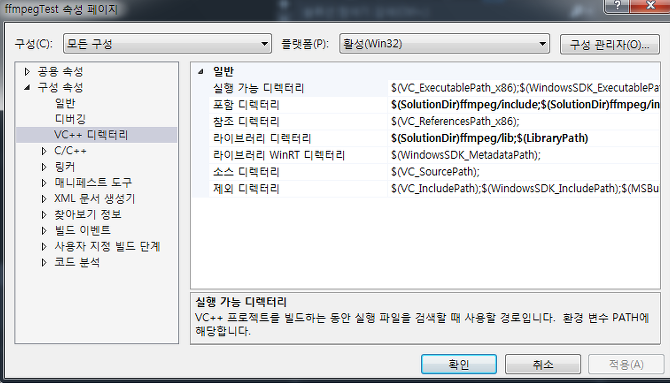
* MSYS (GNU 유틸리티) 설치 후 위와 같은 경로에 “pr.exe” 파일을 저장 “pr.exe”파일이 없으면 configure 시에 오류가 발생한다.
* ※ pr.exe 다운로드 경로https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/\_obsolete/coreutils-5.97-MSYS-1.0.11-2/coreutils-5.97-MSYS-1.0.11-snapshot.tar.bz2/download
* x264 소스코드를 다운로드한다. <http://www.videolan.org/developers/x264.html>
* ffmpeg 공식홈페이지로 가서 소스코드파일을 다운로드 한다.
* 다운받은 소스코드 폴더를 MinGW/msys/1.0 폴더에 저장한다.
* MinGW \ msys \ 1.0 \ msys.bat 실행
* 루트폴더는 MinGW \ msys \ 1.0 을 기준으로 설정되어있다..
* x264에 대한 configure를 실행 명령어 “./configure –enable-shared”
* make, make install 명령을 순서대로 수행하여 x264를 빌드한 후 설치한다
* 루트폴더에 local이라는 폴더가 생겼으면 편의를 위해 폴더이름을 x264로 변경한다

“./configure --enable-shared --enable-static --target-os=mingw32 --enable-gpl --enable-libx264 --extra-cflags=”-I/x264/include” --extra-ldflags=”-L/x264/lib” --enable-gpl”

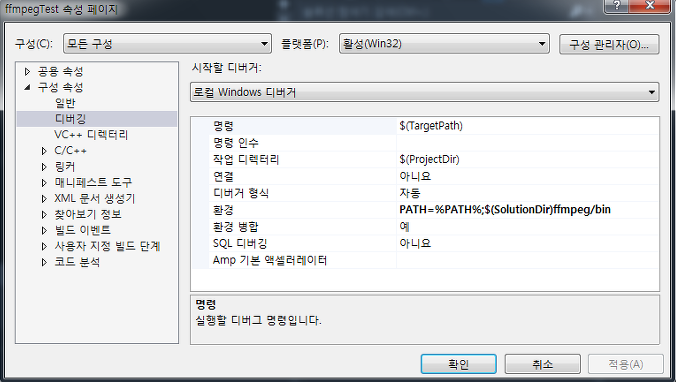
* 마지막에 경고가 뜨지만 pkg-config가 없다는 것이므로 최신 라이브러리에선 없는게 맞다.
* MinGW\include\unistd.h 파일에 “\_cdecl”을 “\_\_cdecl”로 수정한다. 최신버전을 받았다면 수정하지 않아도 될 수 있다.  
  ffmpeg폴더 위치로 cd명령어를 통해 접근한 후 make와 make install을 해준다. 이 작업에서 꽤 많은 시간이 소요될 수 있다.
* 설치가 완료되면 MinGW\msys\1.0\local 폴더에 파일들이 정리되어 있다.
* 이후 외부 lib인 x264를 실행파일 폴더에 복사하여 사용한다.
* ffmpeg 폴더 내의 bin 폴더에서 lib 파일들을 가져와, lib 폴더에 넣는다. (추가 안할시 library 참조 오류 발생)
* 참고 문헌 내의 inttype.h 관련 사항은 수정 할 필요가 없다. (재정의 오류 발생)
* 릴리즈 모드 컴파일시 SAFESEH 오류가 확인되면, 프로젝트 속성 -> 링커 -> 명령줄 에서 “/safeseh:no”를 추가한다.

## Visual Studio 환경 설정 방법

* 만들어진 Library[[2]](#footnote-2)를 참고해, 필요한 정보를 추가한다.
* 위 컴파일 과정에서 만들어진 bin, include, lib 폴더를 개발 솔루션 폴더 아래에 ffmpeg폴더로 생성하여 이동한다.
* 모든 구성 속성에 대하여 디렉터리 및 라이브러리 디렉터리에 각각 include폴더와 lib폴더를 입력한다.



* Ffmpeg 실행파일 경로 설정을 해준다.



* Wind32 응용프로그램을 만들어서 간단한 테스트를 진행해본다.[[3]](#footnote-3)

av\_log(NULL, AV\_LOG\_INFO,”TEST!!”);

# 코딩

## 4.1 Header Include

FFMPEG를 사용하기 위해, 필요한 헤더 파일들을 포함한다.

#include <libavformat/avformat.h> // Basic Format manage header

#include <libavcodec/avcodec.h> // For using codec header

#include <libavutil/avutil.h> // AV Convenience header

#include <libavutil\imgutils.h> // Img Convenience header

#include <libswscale/swscale.h> // External API header

#include <libavformat/avio.h> // Input & output header

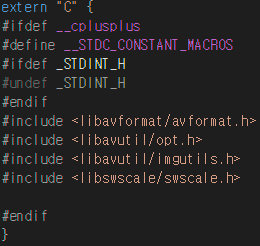
#include <libavutil/opt.h> // option header

#include <libavutil/mathematics.h> // math header

#include <libswresample/swresample.h> // Resample header

## 4.2 Library 초기화 하기

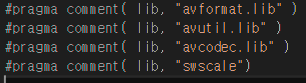
* FFMPEG을 사용하기 전에, 지원하는 모든 Component를 사용 할 수 있게 먼저 등록한다.
* av\_register\_all();
* C형태로 라이브러리가 제작되었기 때문에 헤더들을 include할 때 다음과 같이 입력한다.



* 대부분의 예제코드들이 과거의 라이브러리로 만들어졌기 때문에 deprecated 오류가 생긴다. 원래 경고로 알려주지만 VS2013이후 버전에서는 오류로 취급한다. 다음과 같은 구문을 넣어 오류를 무시해준다.



* 라이브러리가 수많은 패치를 거쳐오면서 함수 이름이 변경되거나 헤더위치가 옮겨지거나해서 중복되서 정의되어있는 경우도 있다. 그래서 외부참조 오류가 날 경우 다음과같이 처리해준다. 직접 링크를 해주는 방법이다.



## 4.3 File Input&Output

[Input File]

* Media 파일을 다루기 위해 먼저, 파일을 Open 한다.
* avformat\_open\_input(AVFormatContext \*\*ps, const char \*filename, AVInputFormat \*fmt, AVdictionary \*\*options);
* AVFormatContext 파라미터는 해당 Media의 정보를 담은 구조체이다.
* Filename 파라미터는 Open할 파일의 이름이다.
* fmt 파라미터는 강제로 InputFormat 을 지정할 경우 사용한다. (보통 NULL)
* options 파라미터는 Demuxer의 추가 옵션을 설정할 때 사용한다. (보통 NULL)

[Output File]

* Media Data를 저장하기 위한, File과 OutputFormat을 설정한다.
* Transcoding 한 Media File을 저장하기 위한 파일 경로를 지정한다.
  + - Ex) const char \*outputfile = "output.mkv";
* OutFile의 ContainerFormat을 정하기 위해, avformat\_alloc\_output\_context2 를 이용한다
  + - Ex) avformat\_alloc\_output\_context2(&ofmt\_ctx, NULL, NULL, outputfile);

위 함수는 outputfile의 Name을 기준으로 Format\_context를 Allocate 한다.

이 문서에서는 container format을 mp4로 설정한다.

avformat\_alloc\_output\_context2(&ofmt\_ctx, NULL, “mp4”, outputfile);

(OutputFormat을 직접 지정하거나, FormatName을 지정 할 수 있다.)

※ 어떤 샘플의 경우 컨테이너를 여러 가지로 설정이 가능해서 outputfile만 설정해주고 함수를 실행하게되면 제대로된 컨테이너를 입히지 못하고 파일을 실행했을 때 timestamp값이 이상해지거나 재생이 되질 않거나하는 오류가 발생한다. Outputfile에 context에 가능한 컨테이너 값을 직접 구분지어 주거나 선택해주는 예외처리를 통해 이 문제를 해결 한다.

string input\_format = ifmt\_ctx->iformat->name;

if (input\_format.find("mp4") != -1) {

output\_format = "mp4";

}

else if (input\_format.find("avi") != -1) {

output\_format = "avi";

}

else if (input\_format.find("flv") != -1) {

output\_format = "flv";

}

else {

output\_format = "ts";

}

* AVOutputFormat \*ofmt을 allocate한 ofmt\_ctx의 OutputFormat으로 초기화 시킨다.

Ex) ofmt = ofmt\_ctx->oformat;

[수정 전 outputfile 의 문제점]

* 아래 같은 방법으로 파일을 생성하게 된다.

File\* outputfile = fopen(Filename, "wb");

* 인코딩으로 생성된 패킷들을 파일에 입력한다.

fwrite(outpkt.data, 1, outpkt.size, f);

* mpg파일을 테스트로 사용하였기 때문에 end sequence code를 추가한다.

uint8\_t endcode[] = { 0, 0, 1, 0xb7 };

fwrite(endcode, 1, sizeof(endcode), f);

* 하지만 이와 같이 output을 만들경우 여러가지 스트림이 존재할 때 원하는 한가지 스트림만 저장이 가능하다. 그리고 컨테이너에 따라 직접 모두 설정해줘야하기 때문에 라이브러리 내에서 제공해주는 함수를 사용하지 않게 된다.

## 4.4 Stream Demuxing

* 대부분의 Media는 Video Stream 하나와 Audio Stream 하나로 구성 되어있으며, 보통 Video는 0번 Stream\_Index를 가지고, Audio는 1번 Stream\_Index를 가진다.

하지만, 복수의 Stream을 가지는 Media도 있을 수 있기에, 다음의 함수를 사용한다.

* nVSI = av\_find\_best\_stream(f\_ctx, AVMEDIA\_TYPE\_VIDEO, -1, -1, NULL, 0); 사용한다.

위 함수는 현재 FormatContext에서 다음 Stream의 Index를 반환하는 함수이다.

찾은 Stream Index를 이용해, 다음의 변수들을 초기화 한다.

AVCodecContext \*pVCtx = f\_ctx->streams[nVSI]->codec; // 비디오 코덱의 Context

AVCodecContext \*pACtx = f\_ctx->streams[nASI]->codec; // 오디오 코덱의 Context

AVFrame \*pVFrame = avcodec\_alloc\_frame(); // 비디오 Frame (Allocation)

AVFrame \*pAFrame = avcodec\_alloc\_frame(); // 오디오 Frame (Allocation)

int bGotPicture = 0; // Data 획득 구분

int bGotSound = 0;

Media file의 Video Stream 과 Audio Stream을 구분해서 사용할수 있는 것을 확인 할 수 있다.

[Stream 생성]

* Input\_FormatContext의 Stream의 CodecContext를 Output\_FormatContext의 Stream의 CodecContext에 Copy한다.
* Input 할 Stream을 초기화하고 필요한 Stream Data 가져온다

AVStream \*in\_stream = ifmt\_ctx->streams[i]; ( i=0 Video Stream, i=1 Audio Stream)

AVCodecContext\* in\_codec\_ctx = in\_stream->codec;

* Output 할 Stream을 선언하고, Output\_FormatContext에 스트림을 추가하기 위해, avformat\_new\_stream 함수를 이용한다.

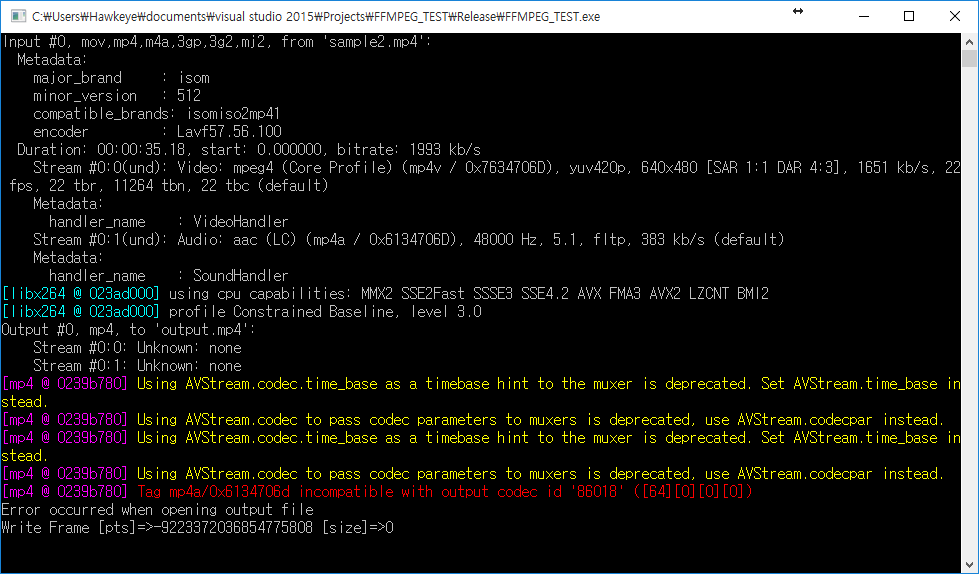
Ex) AVStream \*out\_stream = avformat\_new\_stream(ofmt\_ctx, in\_stream->codec->codec);

* Codec\_Context를 복사하기 위해 avcodec\_copy\_context을 이용한다.

Ex) ret = avcodec\_copy\_context(out\_stream->codec, c);

* 저장하고자 하는 File 형식에 맞게 CodexContext를 설정하기 위해, codec\_tag를 0으로 초기화 한다.

\*codec\_tag를 초기화 해주지않으면 제대로 된 컨테이너를 씌울수가 없다. 다음과 같은 오류가 발생한다. Codec\_tag를 0으로 설정하면 ffmpeg에서 default값으로 설정한다.

Codec\_tag Error 예시)

* 구현된 코드는 다음과 같다.

for (int i = 0; i < ifmt\_ctx->nb\_streams; i++) {

/\*Video Context Copy\*/

AVStream \*in\_stream = ifmt\_ctx->streams[i];

AVCodecContext\* in\_codec\_ctx = in\_stream->codec;

AVStream \*out\_stream = avformat\_new\_stream(ofmt\_ctx, in\_stream->codec->codec);

if (ifmt\_ctx->streams[i]->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO) {

ret = avcodec\_copy\_context(out\_stream->codec, c);

out\_stream->codec->codec\_tag = 0;

}

/\*Audio Context Copy\*/

else if (ifmt\_ctx->streams[i]->codec->codec\_type == AVMEDIA\_TYPE\_AUDIO) {

ret = avcodec\_copy\_context(out\_stream->codec, in\_stream->codec);

out\_stream->codec->codec\_tag = 0;

}

}

* 이후 OutputFile의 Header를 저장한다.

av\_dump\_format(ofmt\_ctx, 0, outputfile, 1); // Output\_Format의 Detail을 출력

* if (!(ofmt->flags & AVFMT\_NOFILE)) {
* ret = avio\_open(&ofmt\_ctx->pb, outputfile, AVIO\_FLAG\_WRITE); // File\_Open
* if (ret < 0) {
* fprintf(stderr, "Could not open output file '%s'", outputfile);
* }
* }
* ret = avformat\_write\_header(ofmt\_ctx, NULL); // Header 작성
* if (ret < 0) {
* fprintf(stderr, "Error occurred when opening output file\n");
* }

## 4.5 Decoding[[4]](#footnote-4)

* Decoding 하기 위해, 사용 할 Decoder를 설정한다.
* CODEC\_ID에 맞는 디코더를 찾기 위해, avcodec\_find\_decoder 함수를 사용한다.

이를 통해 Codec을 초기화 한다.

AVCodec \*pVideoCodec = avcodec\_find\_decoder(f\_ctx->streams[nVSI]->codec->codec\_id);

Avcodec\_find\_decoder는 등록된 디코더들을 순차적으로 검색해, 반환하는 함수이다.

(목록은 FFMPEG –Decoders 명령으로 확인 할 수 있다.)

특정 디코더를 할당하려면 avcodec\_find\_decoder\_by\_name(const char\* name) 을 이용한다.

* avcodec\_open2를 사용해 CodecContext를 Codec으로 초기화한다.

if (avcodec\_open2(f\_ctx->streams[nVSI]->codec, pVideoCodec, NULL) < 0)

* Decoder 설정이 완료 되면, Decoding을 한다.

av\_read\_frame 함수를 이용해 AVFormatContext와 Packet을 전달, frame을 읽어온다.

이후 Packet의 Stream을 구분하여 Video, Audio를 나눈다.

각각의 Packet에 Decode 함수인 avcodec\_decode\_video2를 사용한다.

다음은 선언과 초기화를 제외한 Decoding 과정의 코드 예시이다.

while (av\_read\_frame(f\_ctx, &packet) >= 0) {

if (packet.stream\_index == nVSI) {

// Decode Video

avcodec\_decode\_video2(dec\_ctx, pVFrame, &bGotPicture, &packet);

if (bGotPicture) {

// Encode Video (Example)

encoding(pVFrame, packet);

}

}

}

## 4.6 Encoding

* Encoding을 하기에 앞서 필요한 변수들을 초기화 해준다.

AVCodec \*codec; // Codec for encode

AVCodecContext\* c = NULL; // Codec Context

* Decoding과 마찬가지로
* AVCodec 에 원하는 Encoder를 찾기 위해, avcodec\_find\_encoder 함수를 사용한다.

Ex) codec = avcodec\_find\_encoder(Codec\_id);

( avcodec\_find\_decoder\_by\_name 함수와 마찬 가지로

avcodec\_find\_encoder\_by\_name 으로 원하는 인코더를 따로 할당 할 수 있다. )

* AVCodecContext 를 AVCodec 에 맞게 Allocation한다.

c = avcodec\_alloc\_context3(codec);

* AVCodecContext c 에 원하는 인코딩 정보를 입력한다.

(유저가 수정 할 수 있는 Encoding 정보는 아래 부록을 참고한다.)

예시)

c->bit\_rate = 500000;

c->width = 1280;

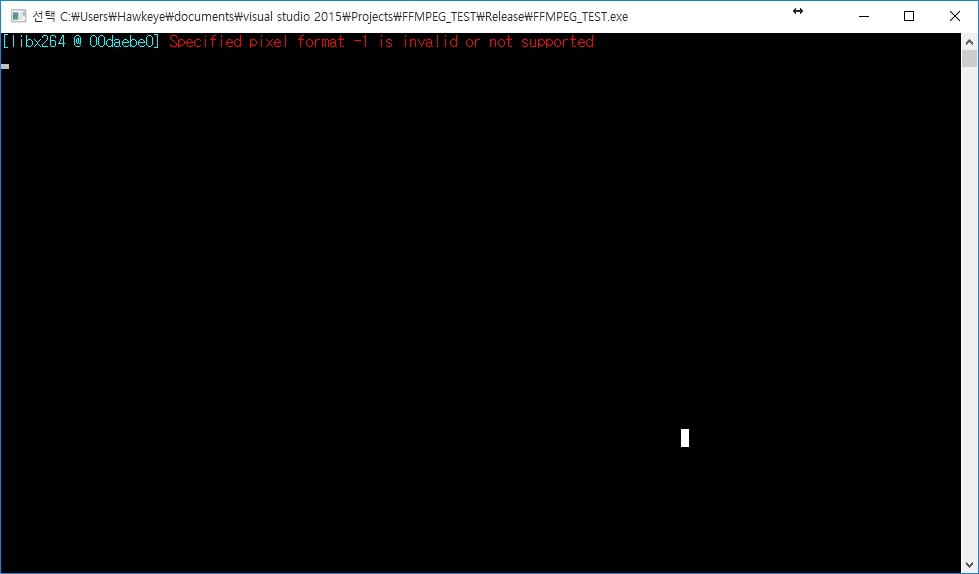
c->height = 528;

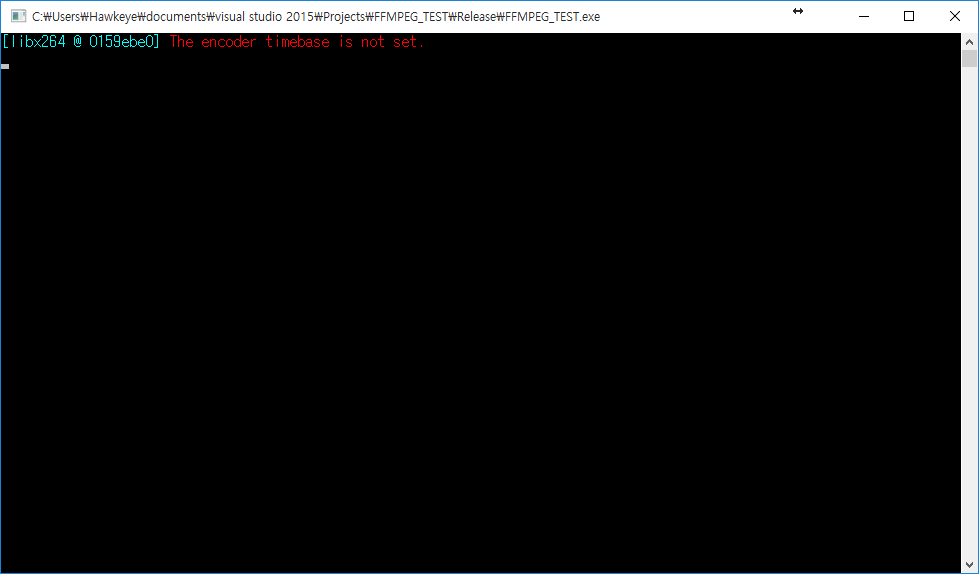
c->time\_base = { 1,25 };

c->pix\_fmt = AV\_PIX\_FMT\_YUV420P;

* 주의 )

Encoding 정보를 입력 시 반드시 width, height, time\_base, pix\_fmt 에 대한 정보는 사용자가 직접 입력해 줘야한다.





* Packet을 사용하기 전 초기화 해준다.

av\_init\_packet(&packet);

* Encoding 함수인 avcodec\_encode\_video2 를 사용한다.

Source Data인 pVFrame을 입력해, 인코딩 후 packet으로 반환한다.

avcodec\_encode\_video2(c, &pkt, NULL, &got\_output);

* Encoding 된 Packet을 본 문서에서는 Media File로 저장한다.

fwrite를 이용해 file로 Encoded Packet을 저장한다.

if (got\_output) {

printf("Write frame %3d (size=%5d)\n", i, packet.size);

fwrite(packet.data, 1, packet.size, f);

if (!ret && got\_output && packet.size) {

packet.stream\_index = 0;

/\* Write the compressed frame to the media file. \*/

//ret = av\_interleaved\_write\_frame(oc, &packet);

}

av\_free\_packet(&packet);

}

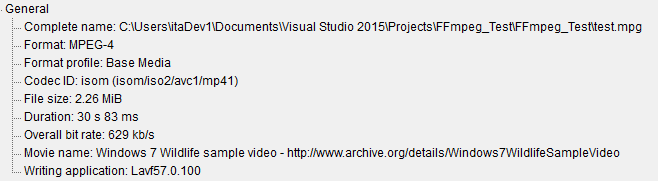
av\_interleaved\_write\_frame를 이용해 파라미터로 지정된 FormatContext에 맞게 파일을 쓸수 있다.

av\_free\_packet을 이용해, 사용 후, 남은 패킷은 해제한다.

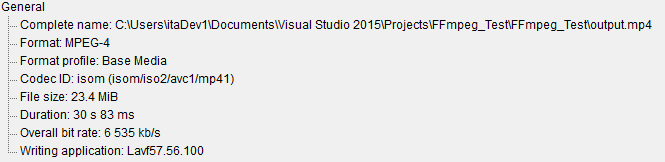
# 결과

변경 된 Media File의 정보를 확인 할 수 있다.

## 5.1 Genenral Result

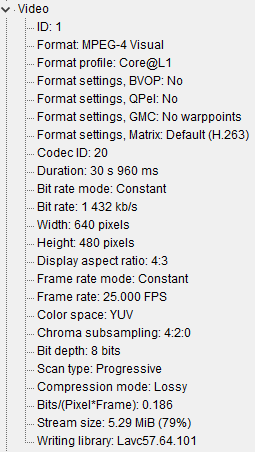
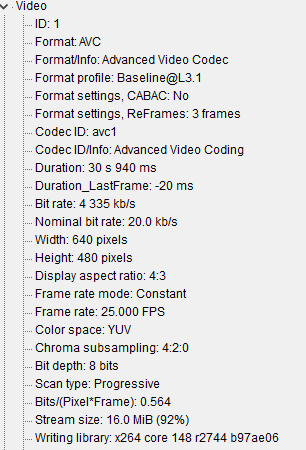


<인코딩 전 저장된 Media의 General Info>



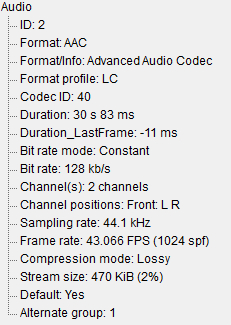
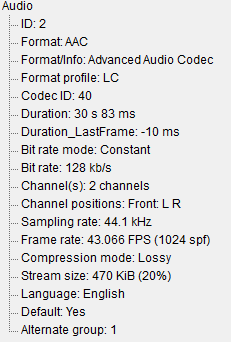
<인코딩 후 저장된 Media의 General Info>

## 5.2 Video Result

<인코딩 전 저장된 Video의 Info> <인코딩 후 저장된 Video의 Info>

## 5.3 Audio Result



<인코딩 전 저장된 Audio의 Info> <인코딩 후 저장된 Audio의 Info>

# 참고문헌

FFMPEG 활용 코딩시에 참고했던 자료의 링크를 첨부한다.

|  |  |
| --- | --- |
| 1 | Google, “FFMPEG VisualStudio” <http://aslike.egloos.com/3081447> (2017.01.10) |
| 기본적인 FFMPEG 코딩의 기초를 확인 할 수 있다. |
| 2 | Google, “FFMPEG Lib” <https://www.ffmpeg.org/doxygen/2.1/index.html> (2017.01.10) |
| FFMPEG 라이브러리의 Document를 확인 할 수 있다. |
| 3 | Google, “FFMPEG Windows” <http://dg087.tistory.com/61> (2017.01.10) |
| MinGW를 이용한, 윈도우용 라이브러리 제작을 확인 할수 있다. |
| 4 | Google, “FFMPEG Decoding & Encoding”  <http://ffmpeg.org/doxygen/trunk/decoding__encoding_8c-source.html> (2017.01.10) |
| Decoding & Encoding의 예제 소스를 참고 할 수 있다. |
| 5 | 이기곤, 『FFmpeg 라이브러리 코덱과 영상 변환을 중심으로』, 한빛미디어(2015) |
| 전반적인 FFMPEG Library와 Coding Guide 를 확인 할 수 있다. |

본 Document의 Source code는

Github에서 다운로드 할수 있다.

[https://github.com/gomyk/FFmpeg\_Transcoding](https://github.com/gomyk/FFmpeg_Transcoding%20)

# 부록

[Encoding Option 목록]

|  |  |  |
| --- | --- | --- |
| 이름 | 변수 형식 | 설명 |
| width | Int | Picture의 가로 크기를 저장한다. |
| height | int | Picture의 세로 크기를 저장한다. |
| time\_base | AVRational | Frame Time Stamp에 대한 내용을 저장한다. |
| bit\_rate | int64 | 평균적인 비트 전송률을 저장한다. |
| profile | int | Profile을 저장한다. (Main, Baseline[[5]](#footnote-5), high 4:4:4 profile[[6]](#footnote-6)) |
| pix\_fmt | AVPixelFormat | Pixel의 Format을 저장한다. AV\_PIX\_FMT\_XXX |
| sample\_aspect\_ratio | AVRational | Video의 종횡비를 저장한다. |
| gop\_size | int | gop의 size를 저장한다. |

1. Google, “FFMPEG Windows” <http://dg087.tistory.com/61> (2017.01.10) [↑](#footnote-ref-1)
2. Google, “FFMPEG Lib” <https://www.ffmpeg.org/doxygen/2.1/index.html> (2017.01.10) [↑](#footnote-ref-2)
3. Google, “FFMPEG VisualStudio” <http://aslike.egloos.com/3081447> (2017.01.10) [↑](#footnote-ref-3)
4. Google, “FFMPEG Decoding & Encoding”, <http://ffmpeg.org/doxygen/trunk/decoding__encoding_8c-source.html> (2017.01.10) [↑](#footnote-ref-4)
5. b\_frame을 사용하지않음, wpredp를 사용하지않음, CABAC를 지원하지않음 [↑](#footnote-ref-5)
6. CRF, CQP mode를 사용하지않음 [↑](#footnote-ref-6)