# 实战让你搞懂ADTS!

今天给大家分享的音视频主题是AAC文件格式里面的ADTS,开始随着我的步伐一起探索吧！

## **AAC ADTS格式探索：**

### **1、AAC两种文件格式：**

AAC(Advanced Audio Coding)是一种高级音频编码，也是我们平常用的和见的最多了一种音频编码标准（音频编码标准，暂时看到的有：AAC、AC-3(DolbyDigital AC-3,d杜比数字AC-3是杜比公司开发的一种家庭影院多声道数字音频系统)、MP3(Moving Picture Experts Group Audio Layer-3)、WMA(Window Media Audio)）,而AAC音频编码是一种有MPEG-4标准定义的有损音频压缩格式，同时 它有两种音频文件格式：

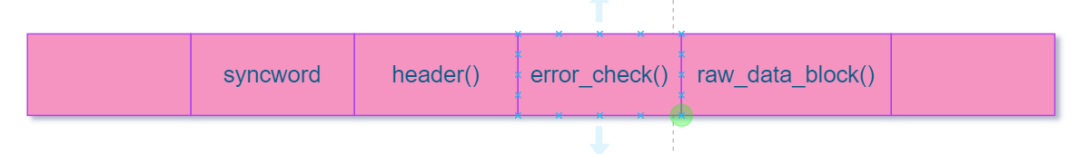
ADIF:音频数据交换格式(Audio Data Interchange Format),它的特征是可以确定的找到这个音频数据的开始，不需要进行在音频数据流中间开始解码，也就是说它的解码必须在明确定义的开始处开始进行。比较常用在磁盘文件中

ADTS:音频数据传输流(Audio Data Transport Stream),它的特征与音频数据交换格式不同，它是一个有同步字的比特流，解码的时候，可以在这个音频流中任何位置开始进行解码。换一句话来说，ADTS可以在在任意帧解码(也就是说它每一帧都有头部信息)；而ADIF只有一个统一的头，所以必须得到所有的数据后解码

下面我们来看一下这两种格式构造图：



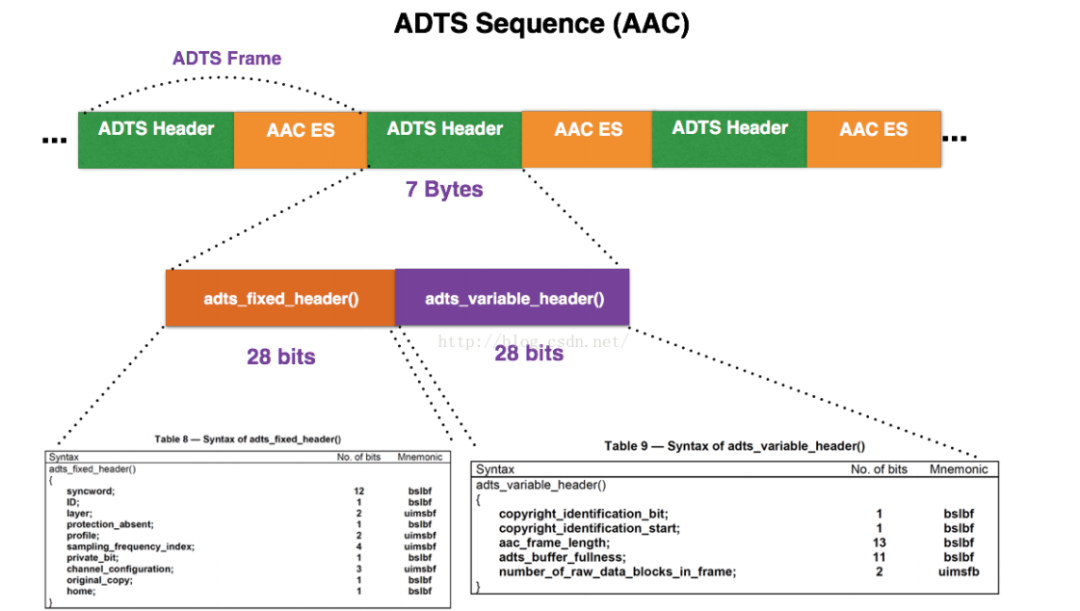
ADIF组成

 ADTS组成

注意：ADTS组成中，最前面的空格和最后的空格表示前后帧。不过目前编码后和抽取出来的都是ADTS格式的音频流

平时如果你遇到了编码AAC裸流的时候，很有可能会出现写出来的AAC文件不能在PC端和手机上进行正常播放，这时候可能的原因是你的AAC文件中的每一帧里面缺少了ADTS头信息文件的包装拼接。解决方法：只需要加⼊头⽂件ADTS即可。⼀个AAC原始数据块⻓度是可变的，对原始帧 加上 ADTS头 进⾏ADTS的封装，就形成了ADTS帧。

同时我们要知道AAC音频文件中的每一帧是由ADTS Header 和 AAC Audio Data组成，结构图如下：



AAC音频文件中的每一帧组成图

注意：ADTS Header的长度可能是7个字节或者9个字节

protection\_absent=0时，则ADTS Header的长度为9个字节；

protection\_absent=1时，则ADTS Header的长度为7个字节。

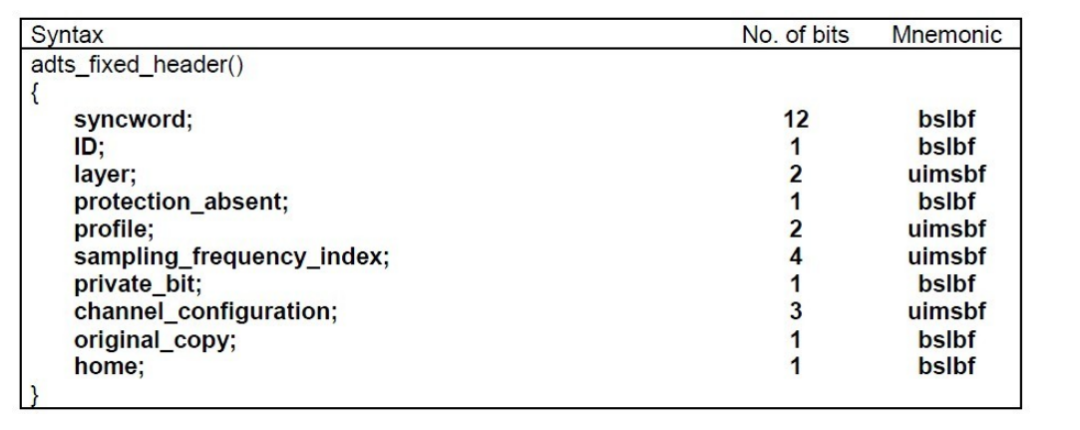
每⼀帧的ADTS的头⽂件都包含了⾳频的采样率，声道，帧⻓度等信息，这样解 码器才能解析读取。⼀般情况下ADTS的头信息都是7个字节，分为2部分：

adts\_fixed\_header();

adts\_variable\_header();

adts\_fixed\_header()为固定头信息，adts\_variable\_header()为可变头信息；固定头信息中的数据每⼀帧都相同，⽽可变头信息则在帧与帧之间可变。

我们先来看固定头信息：

固定头信息

我现在来分析一下：

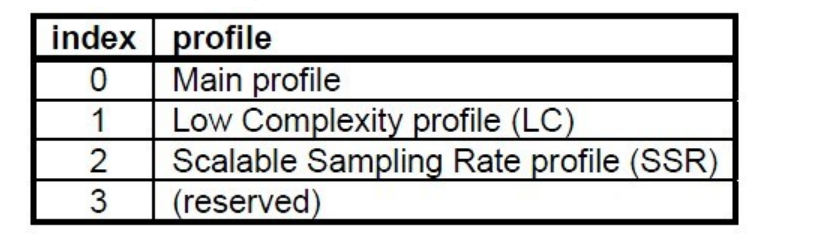
syncword：同步头 总是0xFFF, all bits must be 1，代表着⼀个ADTS帧的开始

ID：MPEG标识符，0标识MPEG-4，1标识MPEG-2

Layer：always: '00'

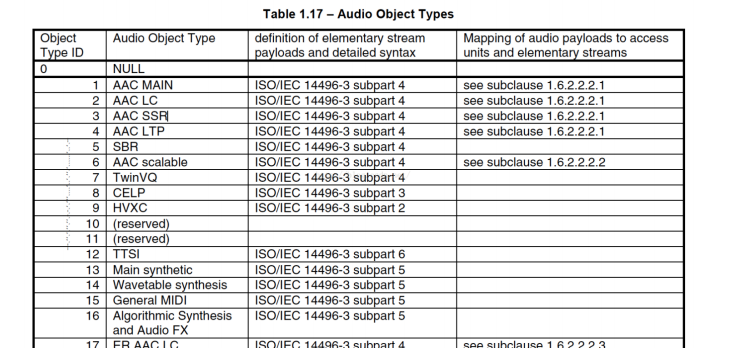
protection\_absent：表示是否误码校验。Warning, set to 1 if there is no CRC and 0 if there is CRC

profile：表示使⽤哪个级别的AAC，如01 Low Complexity(LC)--- AAC LC。有些芯⽚只⽀持AAC LC；在MPEG-2 AAC中定义了3种：

AAC等级

并且profile的值等于 Audio Object Type的值减1：

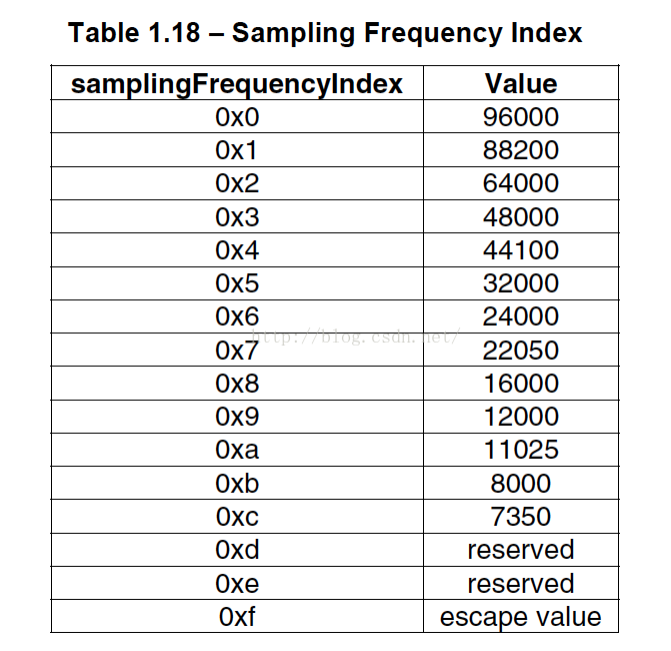
profile = MPEG-4 Audio Object Type - 1



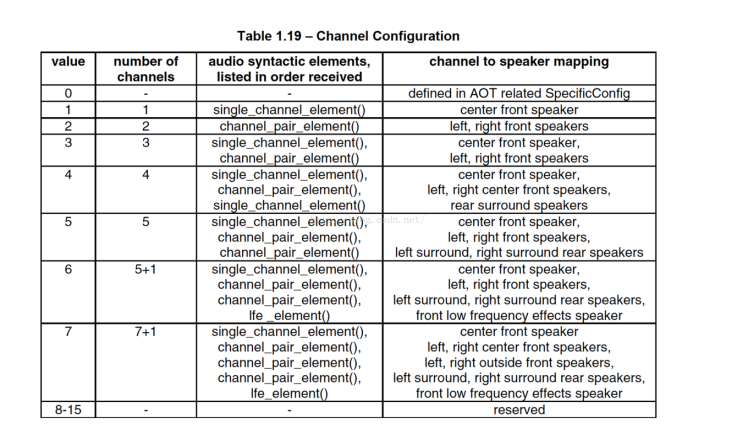
在ffmpeg源码中我们可以找到AAC级别被设成的值，就不用上面的那个公式来计算了：

   /\*\*  
     \* profile  
     \* - encoding: Set by user.  
     \* - decoding: Set by libavcodec.  
     \*/  
     int profile;  
#define FF\_PROFILE\_UNKNOWN -99  
#define FF\_PROFILE\_RESERVED -100  
  
#define FF\_PROFILE\_AAC\_MAIN 0  
#define FF\_PROFILE\_AAC\_LOW  1  
#define FF\_PROFILE\_AAC\_SSR  2  
#define FF\_PROFILE\_AAC\_LTP  3  
#define FF\_PROFILE\_AAC\_HE   4  
#define FF\_PROFILE\_AAC\_HE\_V2 28  
#define FF\_PROFILE\_AAC\_LD   22  
#define FF\_PROFILE\_AAC\_ELD  38  
#define FF\_PROFILE\_MPEG2\_AAC\_LOW 128  
#define FF\_PROFILE\_MPEG2\_AAC\_HE  131

sampling\_frequency\_index：表示使⽤的采样率下标，通过这个下标在 Sampling Frequencies[ ]数组中查找得知采样率的值。

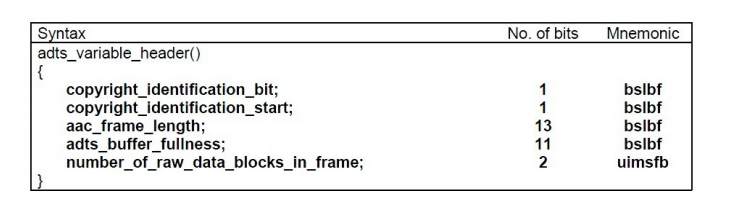
 采样率汇总

channel\_configuration: 表示声道数，⽐如2表示⽴体声双声道



0: Defined **in** AOT Specifc Config   
  
1: 1 channel: front-center   
  
2: 2 channels: front-left, front-right  
  
3: 3 channels: front-center, front-left, front-right   
  
4: 4 channels: front-center, front-left, front-right, back-center   
  
5: 5 channels: front-center, front-left, front-right, back-left, back- right  
  
6: 6 channels: front-center, front-left, front-right, back-left, back- right, LFE-channel   
  
7: 8 channels: front-center, front-left, front-right, side-left, side-right, back-left, back-right, LFE-channel   
  
8-15: Reserved

然后我开始来看可变头信息：

可变头信息

具体解析：

* frame\_length : ⼀个ADTS帧的⻓度包括ADTS头和AAC原始流. frame length, this value must include 7 or 9 bytes of header length: aac\_frame\_length = (protection\_absent == 1 ? 7 : 9) + size(AACFrame)

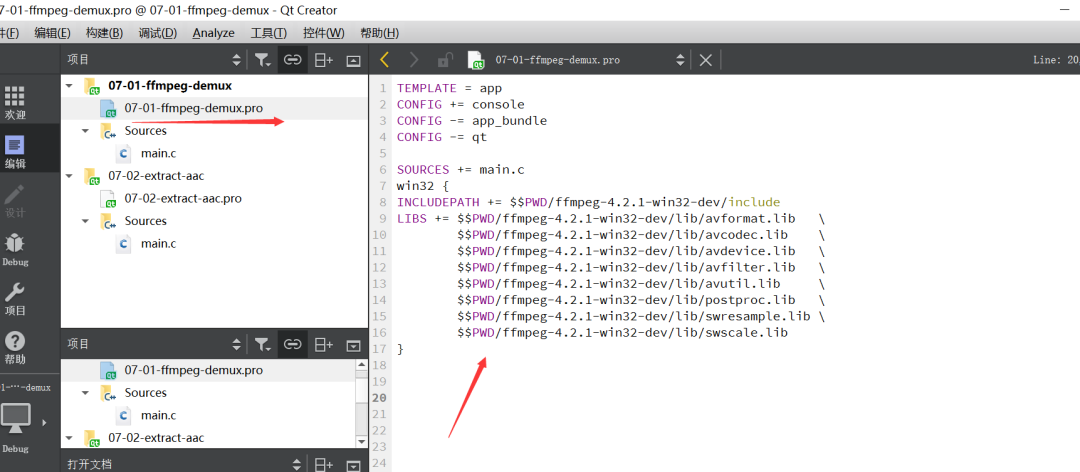
protection\_absent=0时, header length=9bytes

protection\_absent=1时, header length=7bytes

* adts\_buffer\_fullness：0x7FF 说明是码率可变的码流。number\_of\_raw\_data\_blocks\_in\_frame：表示ADTS帧中有 number\_of\_raw\_data\_blocks\_in\_frame + 1个AAC原始帧。所以说number\_of\_raw\_data\_blocks\_in\_frame == 0 表示说ADTS帧中有⼀个 AAC数据块

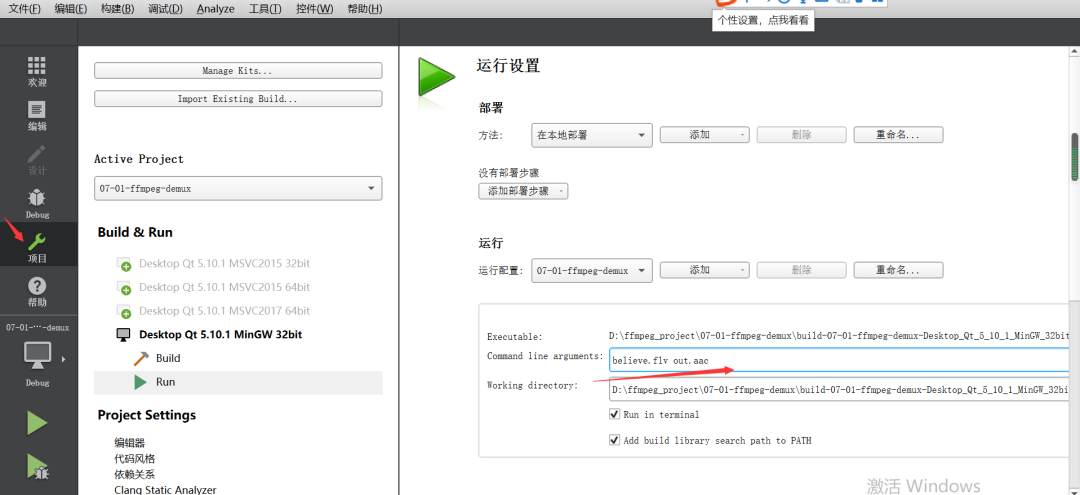
## **代码实战：**

昨天的工程，在qt里面也是要先配置一下，不然你光看我给的代码，不知道怎样配置工程的话，这样工程是跑不起来的哈：

工程配置文件

TEMPLATE = app  
CONFIG += console  
CONFIG -= app\_bundle  
CONFIG -= qt  
  
SOURCES += main.c  
win32 {  
INCLUDEPATH += $$PWD/ffmpeg-4.2.1-win32-dev/include  
LIBS += $$PWD/ffmpeg-4.2.1-win32-dev/lib/avformat.lib   \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/avcodec.lib    \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/avdevice.lib   \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/avfilter.lib   \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/avutil.lib     \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/postproc.lib   \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/swresample.lib \  
        $$PWD/ffmpeg-4.2.1-win32-dev/lib/swscale.lib  
}

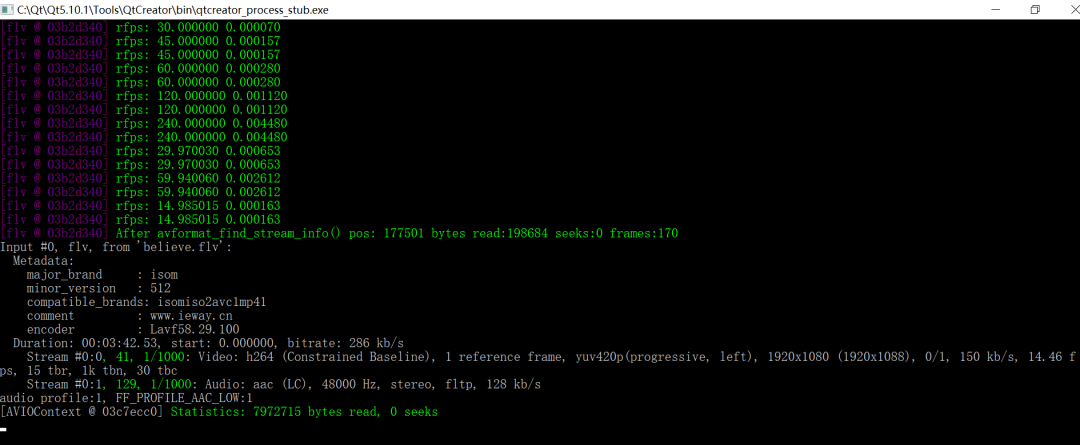
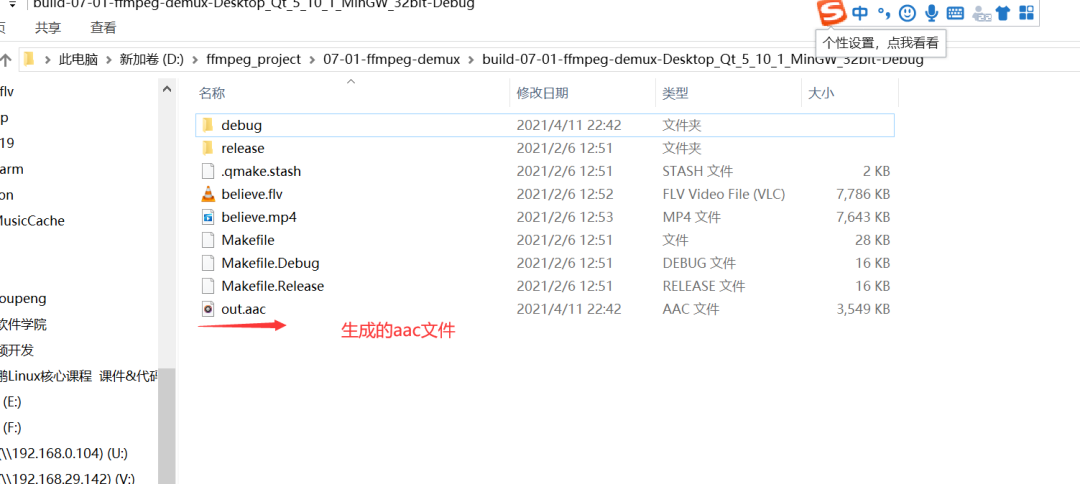
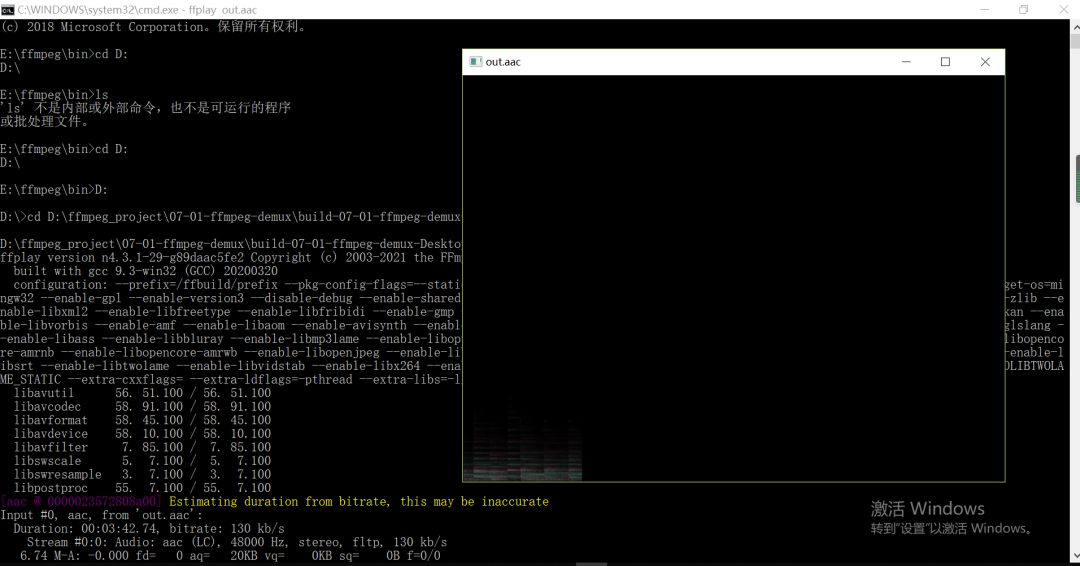
进行工程配置：

工程配置输出为out.aac

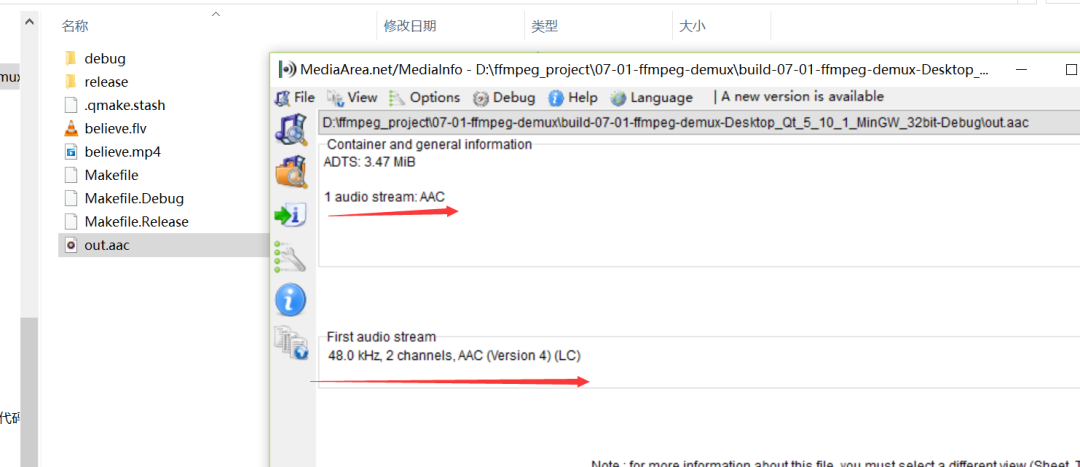
整个工程代码：

#include <stdio.h>  
#include <libavutil/log.h>  
#include <libavformat/avio.h>  
#include <libavformat/avformat.h>  
  
#define ADTS\_HEADER\_LEN  7;  
  
const int sampling\_frequencies[] = {  
    96000,  // 0x0  
    88200,  // 0x1  
    64000,  // 0x2  
    48000,  // 0x3  
    44100,  // 0x4  
    32000,  // 0x5  
    24000,  // 0x6  
    22050,  // 0x7  
    16000,  // 0x8  
    12000,  // 0x9  
    11025,  // 0xa  
    8000   // 0xb  
    // 0xc d e f是保留的  
};  
  
int adts\_header(char \* const p\_adts\_header, const int data\_length,  
                const int profile, const int samplerate,  
                const int channels)  
{  
  
    int sampling\_frequency\_index = 3; // 默认使用48000hz  
    int adtsLen = data\_length + 7;  
  
    int frequencies\_size = sizeof(sampling\_frequencies) / sizeof(sampling\_frequencies[0]);  
    int i = 0;  
    **for**(i = 0; i < frequencies\_size; i++)  
    {  
        **if**(sampling\_frequencies[i] == samplerate)  
        {  
            sampling\_frequency\_index = i;  
            break;  
        }  
    }  
    **if**(i >= frequencies\_size)  
    {  
        printf("unsupport samplerate:%d\n", samplerate);  
        return -1;  
    }  
  
    p\_adts\_header[0] = 0xff;         //syncword:0xfff                          高8bits  
    p\_adts\_header[1] = 0xf0;         //syncword:0xfff                          低4bits  
    p\_adts\_header[1] |= (0 << 3);    //MPEG Version:0 **for** MPEG-4,1 **for** MPEG-2  1bit  
    p\_adts\_header[1] |= (0 << 1);    //Layer:0                                 2bits  
    p\_adts\_header[1] |= 1;           //protection absent:1                     1bit  
  
    p\_adts\_header[2] = (profile)<<6;            //profile:profile               2bits  
    p\_adts\_header[2] |= (sampling\_frequency\_index & 0x0f)<<2; //sampling frequency index:sampling\_frequency\_index  4bits  
    p\_adts\_header[2] |= (0 << 1);             //private bit:0                   1bit  
    p\_adts\_header[2] |= (channels & 0x04)>>2; //channel configuration:channels  高1bit  
  
    p\_adts\_header[3] = (channels & 0x03)<<6; //channel configuration:channels 低2bits  
    p\_adts\_header[3] |= (0 << 5);               //original：0                1bit  
    p\_adts\_header[3] |= (0 << 4);               //home：0                    1bit  
    p\_adts\_header[3] |= (0 << 3);               //copyright id bit：0        1bit  
    p\_adts\_header[3] |= (0 << 2);               //copyright id start：0      1bit  
    p\_adts\_header[3] |= ((adtsLen & 0x1800) >> 11);           //frame length：value   高2bits  
  
    p\_adts\_header[4] = (uint8\_t)((adtsLen & 0x7f8) >> 3);     //frame length:value    中间8bits  
    p\_adts\_header[5] = (uint8\_t)((adtsLen & 0x7) << 5);       //frame length:value    低3bits  
    p\_adts\_header[5] |= 0x1f;                                 //buffer fullness:0x7ff 高5bits  
    p\_adts\_header[6] = 0xfc;      //‭11111100‬       //buffer fullness:0x7ff 低6bits  
    // number\_of\_raw\_data\_blocks\_in\_frame：  
    //    表示ADTS帧中有number\_of\_raw\_data\_blocks\_in\_frame + 1个AAC原始帧。  
  
    return 0;  
}  
  
int main(int argc, char \*argv[])  
{  
    int ret = -1;  
    char errors[1024];  
  
    char \*in\_filename = NULL;  
    char \*aac\_filename = NULL;  
  
    FILE \*aac\_fd = NULL;  
  
    int audio\_index = -1;  
    int len = 0;  
  
  
    AVFormatContext \*ifmt\_ctx = NULL;  
    AVPacket pkt;  
  
    // 设置打印级别  
    av\_log\_set\_level(AV\_LOG\_DEBUG);  
  
    **if**(argc < 3)  
    {  
        av\_log(NULL, AV\_LOG\_DEBUG, "the count of parameters should be more than three!\n");  
        return -1;  
    }  
  
    in\_filename = argv[1];      // 输入文件  
    aac\_filename = argv[2];     // 输出文件  
  
    **if**(in\_filename == NULL || aac\_filename == NULL)  
    {  
        av\_log(NULL, AV\_LOG\_DEBUG, "src or dts file is null, plz check them!\n");  
        return -1;  
    }  
  
    aac\_fd = fopen(aac\_filename, "wb");  
    **if** (!aac\_fd)  
    {  
        av\_log(NULL, AV\_LOG\_DEBUG, "Could not open destination file %s\n", aac\_filename);  
        return -1;  
    }  
  
    // 打开输入文件  
    **if**((ret = avformat\_open\_input(&ifmt\_ctx, in\_filename, NULL, NULL)) < 0)  
    {  
        av\_strerror(ret, errors, 1024);  
        av\_log(NULL, AV\_LOG\_DEBUG, "Could not open source file: %s, %d(%s)\n",  
               in\_filename,  
               ret,  
               errors);  
        return -1;  
    }  
  
    // 获取解码器信息  
    **if**((ret = avformat\_find\_stream\_info(ifmt\_ctx, NULL)) < 0)  
    {  
        av\_strerror(ret, errors, 1024);  
        av\_log(NULL, AV\_LOG\_DEBUG, "failed to find stream information: %s, %d(%s)\n",  
               in\_filename,  
               ret,  
               errors);  
        return -1;  
    }  
  
    // dump媒体信息  
    av\_dump\_format(ifmt\_ctx, 0, in\_filename, 0);  
  
    // 初始化packet  
    av\_init\_packet(&pkt);  
  
    // 查找audio对应的steam index  
    audio\_index = av\_find\_best\_stream(ifmt\_ctx, AVMEDIA\_TYPE\_AUDIO, -1, -1, NULL, 0);  
    **if**(audio\_index < 0)  
    {  
        av\_log(NULL, AV\_LOG\_DEBUG, "Could not find %s stream in input file %s\n",  
               av\_get\_media\_type\_string(AVMEDIA\_TYPE\_AUDIO),  
               in\_filename);  
        return AVERROR(EINVAL);  
    }  
  
    // 打印AAC级别  
    printf("audio profile:%d, FF\_PROFILE\_AAC\_LOW:%d\n",  
           ifmt\_ctx->streams[audio\_index]->codecpar->profile,  
           FF\_PROFILE\_AAC\_LOW);  
  
    **if**(ifmt\_ctx->streams[audio\_index]->codecpar->codec\_id != AV\_CODEC\_ID\_AAC)  
    {  
        printf("the media file no contain AAC stream, it's codec\_id is %d\n",  
               ifmt\_ctx->streams[audio\_index]->codecpar->codec\_id);  
        goto failed;  
    }  
    // 读取媒体文件，并把aac数据帧写入到本地文件  
    **while**(av\_read\_frame(ifmt\_ctx, &pkt) >=0 )  
    {  
        **if**(pkt.stream\_index == audio\_index)  
        {  
            char adts\_header\_buf[7] = {0};  
            adts\_header(adts\_header\_buf, pkt.size,  
                        ifmt\_ctx->streams[audio\_index]->codecpar->profile,  
                        ifmt\_ctx->streams[audio\_index]->codecpar->sample\_rate,  
                        ifmt\_ctx->streams[audio\_index]->codecpar->channels);  
            fwrite(adts\_header\_buf, 1, 7, aac\_fd);  // 写adts header , ts流不适用，ts流分离出来的packet带了adts header  
            len = fwrite( pkt.data, 1, pkt.size, aac\_fd);   // 写adts data  
            **if**(len != pkt.size)  
            {  
                av\_log(NULL, AV\_LOG\_DEBUG, "warning, length of writed data isn't equal pkt.size(%d, %d)\n",  
                       len,  
                       pkt.size);  
            }  
        }  
        av\_packet\_unref(&pkt);  
    }  
  
failed:  
    // 关闭输入文件  
    **if**(ifmt\_ctx)  
    {  
        avformat\_close\_input(&ifmt\_ctx);  
    }  
    **if**(aac\_fd)  
    {  
        fclose(aac\_fd);  
    }  
  
    return 0;  
}

最终效果：

运行结果aac文件生成利用ffplay进行播放out.aac

生成的out.aac文件，我们可以用MediaArea进行查看详细信息：

out.aac文件信息

## **总结：**

今天的分享就到这里了，如果你也喜欢音视频开发，可以加我微信，相互交流，一起进步；同时如果大家看到我没更新，不是在偷懒，一般是在给自己疯狂输入；好了，我们下期见

可以扫码加我

音视频开发14

音视频开发 · 目录

上一篇ffmpeg实战实现音视频解封装！下一篇畅聊直播技术！

阅读 1021

​