AAC ADTS格式分析

AAC音频格式：Advanced Audio Coding(高级音频解码)，是一种由MPEG-4标准定义的有损音频压缩格式，由Fraunhofer发展，Dolby, Sony和AT&T是主要的贡献者。

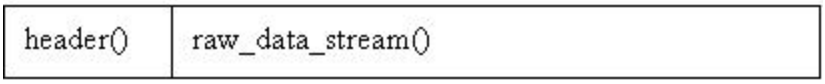
- **ADIF**：Audio Data Interchange Format 音频数据交换格式。这种格式的特征是可以确定的找到这个音频数据的开始，不需进行在音频数据流中间开始的解码，即它的解码必须在明确定义的开始处进行。故这种格式常用在磁盘文件中。

- **ADTS**的全称是Audio Data Transport Stream。是AAC音频的传输流格式。AAC音频格式在MPEG-2（ISO-13318-7 2003）中有定义。AAC后来又被采用到MPEG-4标准中。这种格式的特征是它是一个有同步字的比特流，解码可以在这个流中任何位置开始。它的特征类似于mp3数据流格式。

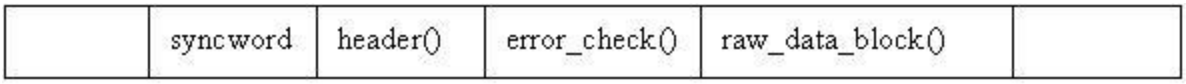
简单说，ADTS可以在任意帧解码，**也就是说它每一帧都有头信息**。ADIF只有一个统一的头，所以必须得到所有的数据后解码。

且这两种的header的格式也是不同的，目前一般编码后的和抽取出的都是ADTS格式的音频流。两者具体的组织结构如下所示：

- AAC的ADIF格式见下图：



- AAC的ADTS的一般格式见下图：



空白处表示前后帧

有的时候当你编码AAC裸流的时候，会遇到写出来的AAC文件并不能在PC和手机上播放，很大的可能就是AAC文件的每一帧里缺少了ADTS头信息文件的包装拼接。
只需要加入头文件ADTS即可。一个AAC原始数据块长度是可变的，对原始帧加上ADTS头进行ADTS的封装，就形成了ADTS帧。

AAC音频文件的每一帧由ADTS Header和AAC Audio Data组成。结构体如下：

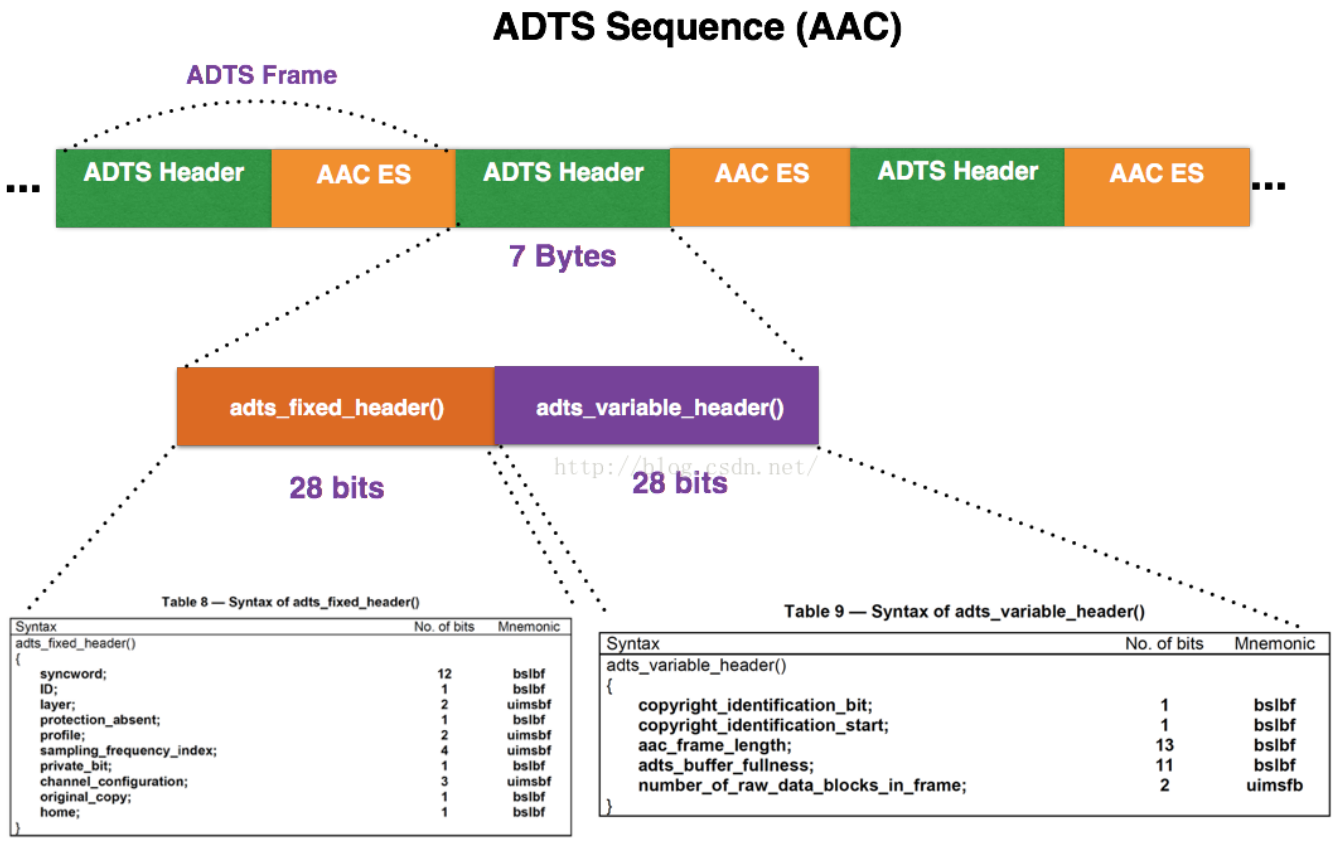


Table 8 — Syntax of adts_fixed_header()

Syntax	No. of bits	Mnemonic
adts_fixed_header()		
{		
syncword;	12	bslbf
ID;	1	bslbf
layer;	2	uimsbf
protection_absent;	1	bslbf
profile;	2	uimsbf
sampling_frequency_index;	4	uimsbf
private_bit;	1	bslbf
channel_configuration;	3	uimsbf
original_copy;	1	bslbf
home;	1	bslbf
}		

Table 9 — Syntax of adts_variable_header()

Syntax	No. of bits	Mnemonic
adts_variable_header()		
{		
copyright_identification_bit;	1	bslbf
copyright_identification_start;	1	bslbf
aac_frame_length;	13	bslbf
adts_buffer_fullness;	11	bslbf
number_of_raw_data_blocks_in_frame;	2	uimsfb
}		

注:ADTS Header的长度可能为7字节或9字节. protection_absent=0时, 9字节. protection_absent=1时, 7字节.

每一帧的ADTS的头文件都包含了音频的采样率，声道，帧长度等信息，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();

其一为**固定头信息**，紧接着是**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。

Syntax	No. of bits	Mnemonic
adts_fixed_header()		
{		
syncword;	12	bslbf
ID;	1	bslbf
layer;	2	uimsbf
protection_absent;	1	bslbf
profile;	2	uimsbf
sampling_frequency_index;	4	uimsbf
private_bit;	1	bslbf
channel_configuration;	3	uimsbf
original_copy;	1	bslbf
home;	1	bslbf
}		

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：

index	profile
0	Main profile
1	Low Complexity profile (LC)
2	Scalable Sampling Rate profile (SSR)
3	(reserved)

profile的值等于 Audio Object Type的值减1
profile = MPEG-4 Audio Object Type – 1

Table 1.17 – Audio Object Types

Object Type ID	Audio Object Type	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
0	NULL		
1	AAC MAIN	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1
2	AAC LC	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1
3	AAC SSR	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1
4	AAC LTP	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1
5	SBR	ISO/IEC 14496-3 subpart 4	
6	AAC scalable	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.2
7	TwinVQ	ISO/IEC 14496-3 subpart 4	
8	CELP	ISO/IEC 14496-3 subpart 3	
9	HVXC	ISO/IEC 14496-3 subpart 2	
10	(reserved)		
11	(reserved)		
12	TTSI	ISO/IEC 14496-3 subpart 6	
13	Main synthetic	ISO/IEC 14496-3 subpart 5	
14	Wavetable synthesis	ISO/IEC 14496-3 subpart 5	
15	General MIDI	ISO/IEC 14496-3 subpart 5	
16	Algorithmic Synthesis and Audio FX	ISO/IEC 14496-3 subpart 5	
17	ER AAC LC	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.3

```
#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。

Table 1.18 – Sampling Frequency Index

samplingFrequencyIndex	Value
0x0	96000
0x1	88200
0x2	64000
0x3	48000
0x4	44100
0x5	32000
0x6	24000
0x7	22050
0x8	16000
0x9	12000
0xa	11025
0xb	8000
0xc	7350
0xd	reserved
0xe	reserved
0xf	escape value

channel_configuration: 表示声道数，比如2表示立体声双声道

Table 1.19 – Channel Configuration

value	number of channels	audio syntactic elements, listed in order received	channel to speaker mapping
0	-	-	defined in AOT related SpecificConfig
1	1	single_channel_element()	center front speaker
2	2	channel_pair_element()	left, right front speakers
3	3	single_channel_element(), channel_pair_element()	center front speaker, left, right front speakers
4	4	single_channel_element(), channel_pair_element(), single_channel_element()	center front speaker, left, right center front speakers, rear surround speakers
5	5	single_channel_element(), channel_pair_element(), channel_pair_element()	center front speaker, left, right front speakers, left surround, right surround rear speakers
6	5+1	single_channel_element(), channel_pair_element(), channel_pair_element(), lfe _element()	center front speaker, left, right front speakers, left surround, right surround rear speakers, front low frequency effects speaker
7	7+1	single_channel_element(), channel_pair_element(), channel_pair_element(), channel_pair_element(), lfe_element()	center front speaker left, right center front speakers, left, right outside front speakers, left surround, right surround rear speakers, front low frequency effects speaker
8-15	-	-	reserved

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

接下来看下adts_variable_header();

Syntax	No. of bits	Mnemonic
adts_variable_header()		
{		
copyright_identification_bit;	1	bslbf
copyright_identification_start;	1	bslbf
aac_frame_length;	13	bslbf
adts_buffer_fullness;	11	bslbf
number_of_raw_data_blocks_in_frame;	2	uimsfb
}		

frame_length：一个ADTS帧的长度包括**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。

下面是ADTS的AAC文件部分：
高字节开始算

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	FF	F1	4C	40	20	FF	FC	01	48	20	06	FE	C0	00	00	00
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000100	00	00	00	00	00	00	0E	FF	F1	4C	40	20	FF	FC	01	48
00000110	20	06	FE	C0	00	00	00	00	00	00	00	00	00	00	00	00

第一帧的帧头7个字节为：0xFF 0xF1 0x4C 0x40 0x20 0xFF 0xFC
 分析各个关键数值：
111111111111
0
00
1
01
0011
0
001
0
0
0
0
0000100000111(帧长度)
11111111111
00
计算帧长度：将二进制 0000100000111 转换成十进制为263。观察第一帧的长度确实为263个字节。
计算方法:（帧长度为13位，使用unsigned int来存储帧长数值）

```
unsigned int getFrameLength(unsigned char* str)
{
    if ( !str )
    {
        return 0;
    }
    unsigned int len = 0;
    int f_bit = str[3];
    int m_bit = str[4];
    int b_bit = str[5];
    len += (b_bit>>5);
    len += (m_bit<<3);
    len += ((f_bit&3)<<11);
    return len;
```

```
15 }
```