FLV文件格式

原创 学不可以已 于 2021-05-03 18:29:41 发布 237 收藏 版权

文章标签： 程序人生

FLV =

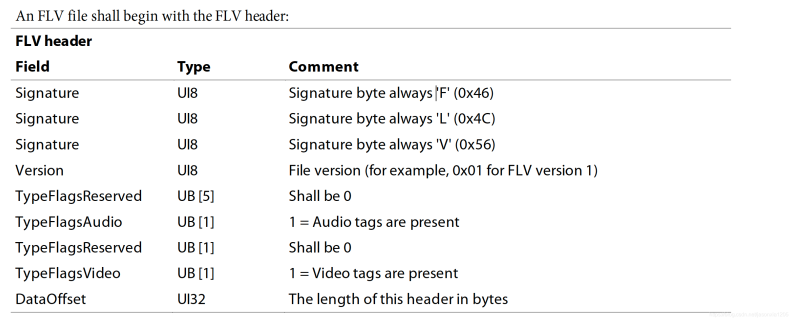
FLV header

+

FLV body

常用的直播协议 RTMP 和HTTP-FLV都使用了FLV文件格式作为媒体流容器。FLV文件格式在Adobe Flash Video File Format Specification的Annex E中定义，可以从这里https://www.adobe.com/devnet/f4v.html下载。

这里主要记录下规范中FLV文件格式相关说明：FlV文件由 header 和body两部分组成。Header由文件签名，版本，音视频存在标记，以及header 长度组成，详细如下图：



An FLV file shall begin with the FLV header:

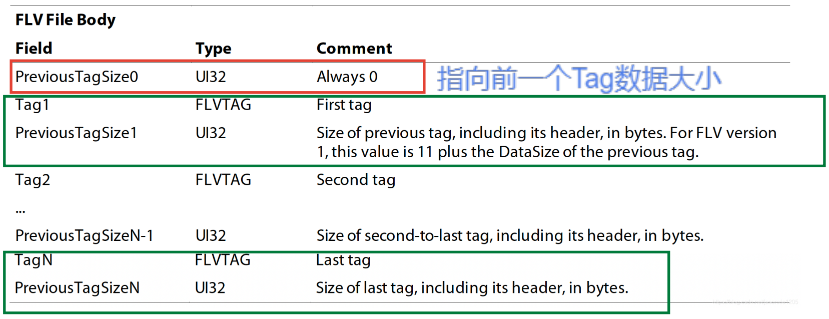
FLV header

Field	Type	Comment
Signature	UI8	Signature byte always 'F' (0x46)
Signature	UI8	Signature byte always 'L' (0x4C)
Signature	UI8	Signature byte always 'V' (0x56)
Version	UI8	File version (for example, 0x01 for FLV version 1)
TypeFlagsReserved	UB [5]	Shall be 0
TypeFlagsAudio	UB [1]	1 = Audio tags are present
TypeFlagsReserved	UB [1]	Shall be 0
TypeFlagsVideo	UB [1]	1 = Video tags are present
DataOffset	UI32	The length of this header in bytes

TypeFlagsAudio和TypeFlagsVideo用来标志Body中是否有音频或视频流，FLV文件中用Tag这个术语来称呼文件中的媒体数据单元。

DataOffset是为了以后协议扩展，如果header部分长度发生变化，旧版本的应用可以根据DataOffset跳过新的扩展头，直接读取body中的Tags。

Header之后就是Body部分，Body由多个Tag组成，每个Tag前有一个Previous Tag Size字段，这个字段标识了前一个Tag的长度，方便向前索引Tag，第一个Previous Tag Size为0，最后一个Previous Tag Size后面没有Tag数据。



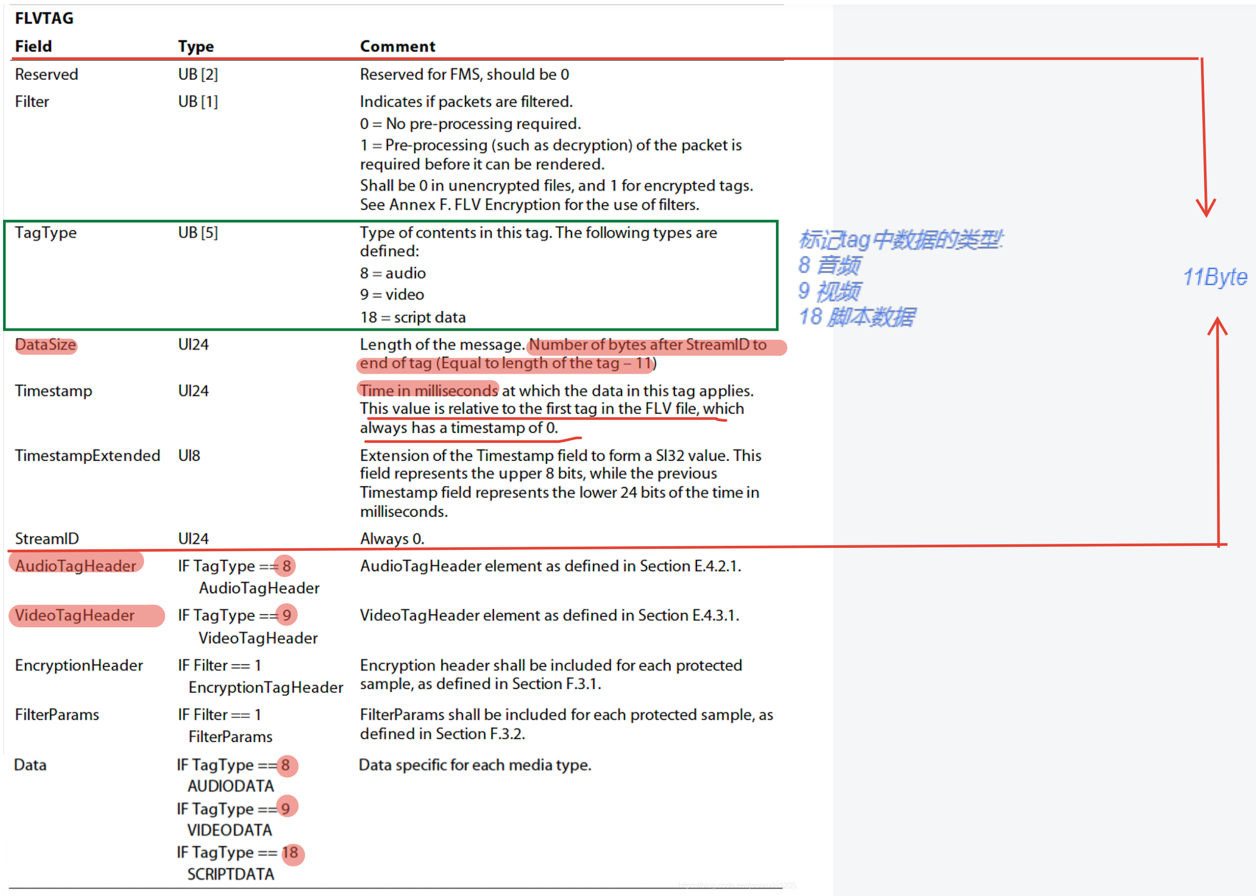
FLV File Body

Field	Type	Comment
PreviousTagSize0	UI32	Always 0
Tag1	FLVTAG	First tag
PreviousTagSize1	UI32	Size of previous tag, including its header, in bytes. For FLV version 1, this value is 11 plus the DataSize of the previous tag.
Tag2	FLVTAG	Second tag
...		
PreviousTagSizeN-1	UI32	Size of second-to-last tag, including its header, in bytes.
TagN	FLVTAG	Last tag
PreviousTagSizeN	UI32	Size of last tag, including its header, in bytes.

FLV Body

每个Tag中包含载荷数据和对应的元数据，如果开启了加密还包括加密信息。播放时，每个tag中的媒体时间戳由tag中的timestamp字段决定，如果载荷数据中也包含时间相关信息，应该被忽略。

学不可以已 关注



FLVTAG

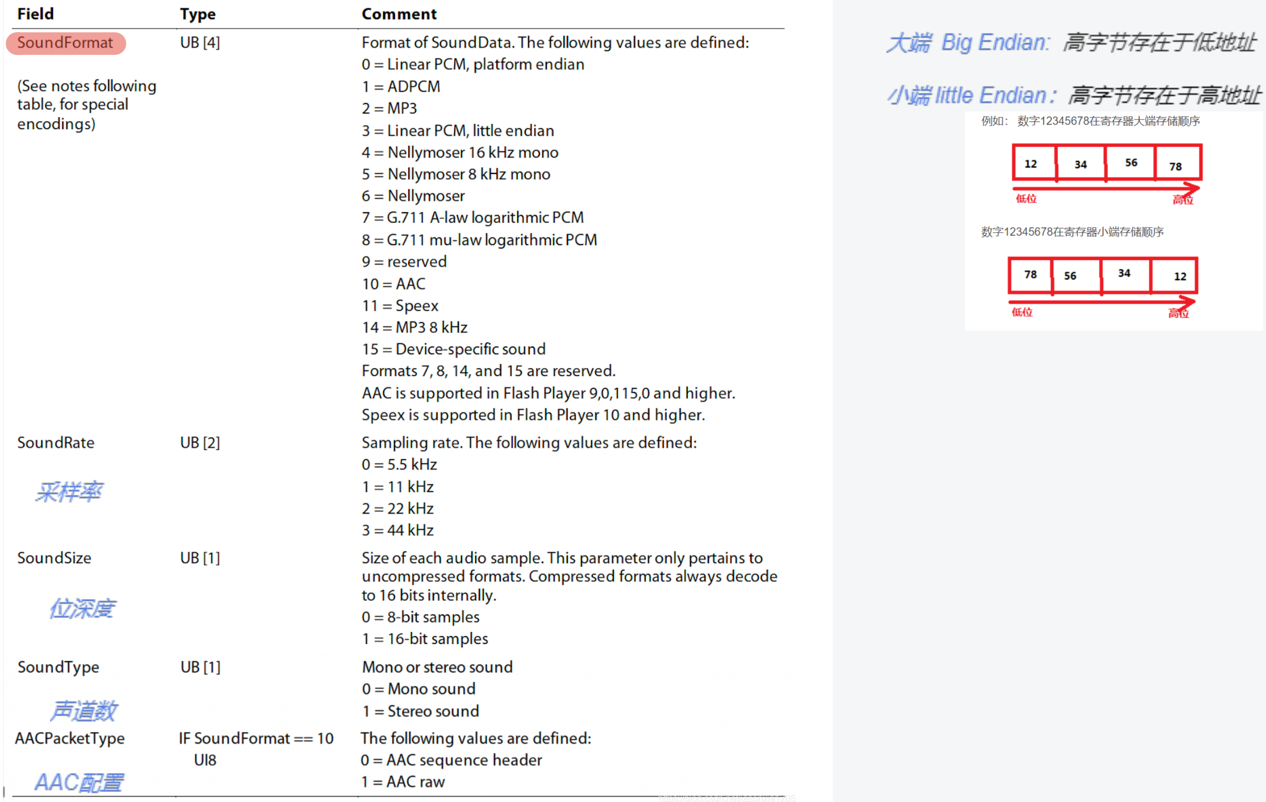
Field	Type	Comment
Reserved	UB [2]	Reserved for FMS, should be 0
Filter	UB [1]	Indicates if packets are filtered. 0 = No pre-processing required. 1 = Pre-processing (such as decryption) of the packet is required before it can be rendered. Shall be 0 in unencrypted files, and 1 for encrypted tags. See Annex F. FLV Encryption for the use of filters.
TagType	UB [5]	Type of contents in this tag. The following types are defined: 8 = audio 9 = video 18 = script data
DataSize	UI24	Length of the message. Number of bytes after StreamID to end of tag (Equal to length of the tag – 11)
Timestamp	UI24	Time in milliseconds at which the data in this tag applies. This value is relative to the first tag in the FLV file, which always has a timestamp of 0.
TimestampExtended	UI8	Extension of the Timestamp field to form a SI32 value. This field represents the upper 8 bits, while the previous Timestamp field represents the lower 24 bits of the time in milliseconds.
StreamID	UI24	Always 0.
AudioTagHeader	IF TagType == 8 AudioTagHeader	AudioTagHeader element as defined in Section E.4.2.1.
VideoTagHeader	IF TagType == 9 VideoTagHeader	VideoTagHeader element as defined in Section E.4.3.1.
EncryptionHeader	IF Filter == 1 EncryptionTagHeader	Encryption header shall be included for each protected sample, as defined in Section F.3.1.
FilterParams	IF Filter == 1 FilterParams	FilterParams shall be included for each protected sample, as defined in Section F.3.2.

Data	IF TagType == 8
	AUDIODATA
	IF TagType == 9
	VIDEODATA
	IF TagType == 18
	SCRIPTDATA
		Data specific for each media type.

DataSize是每个Tag的长度减去StreamID之前（包含StreamID）字段的11个字节。这11个字节也可以叫Tag Header。

Tag Header之后就是Payload的元数据，也就是对应的AudioTagHeader/VideoTagHeader/EncryptionHeader+FilterParams。

对于音频Tag（TagType == 8）AudioTagHeader包含了音频相关的元数据，包括音频数据类型，采样率，采样位数，声道数。

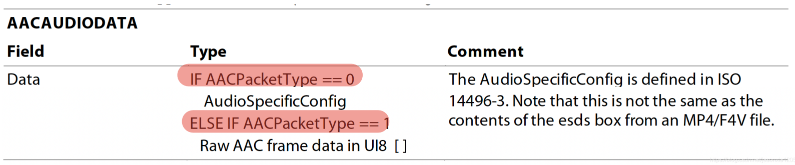


Field	Type	Comment
SoundFormat (See notes following table, for special encodings)	UB [4]	Format of SoundData. The following values are defined: 0 = Linear PCM, platform endian 1 = ADPCM 2 = MP3 3 = Linear PCM, little endian 4 = Nellymoser 16 kHz mono 5 = Nellymoser 8 kHz mono 6 = Nellymoser 7 = G.711 A-law logarithmic PCM 8 = G.711 mu-law logarithmic PCM 9 = reserved 10 = AAC 11 = Speex 14 = MP3 8 kHz 15 = Device-specific sound Formats 7, 8, 14, and 15 are reserved. AAC is supported in Flash Player 9,0,115,0 and higher. Speex is supported in Flash Player 10 and higher.
SoundRate	UB [2]	Sampling rate. The following values are defined: 0 = 5.5 kHz 1 = 11 kHz 2 = 22 kHz 3 = 44 kHz
SoundSize	UB [1]	Size of each audio sample. This parameter only pertains to uncompressed formats. Compressed formats always decode to 16 bits internally. 0 = 8-bit samples 1 = 16-bit samples
SoundType	UB [1]	Mono or stereo sound 0 = Mono sound 1 = Stereo sound
AACPacketType	IF SoundFormat == 10 UI8	The following values are defined: 0 = AAC sequence header 1 = AAC raw

采样率
位深度
声道数
AAC配置
大端 Big Endian: 高字节存在于低地址
小端 little Endian：高字节存在于高地址
例如：数字12345678在寄存器大端存储顺序
12
34
56
78
低位
高位
数字12345678在寄存器小端存储顺序
78
56
34
12
低位
高位

学不可以已 关注

这里重点看一下AAC格式。对于AAC格式，SoundType字段应该置为1，SoundRate应该置为3，但是这里并不表示AAC音频流必须是44.1K立体声，播放器在AAC格式下，会忽略这两个字段的含义，而是从AAC媒体流中获取相应的信息。对于AAC格式，在AudioTagHeader中用AACPacketType字段，表示载荷数据是是AAC Sequence Header还是Raw Data。不考虑加密，在AudioTagHeader后，按照AACPacketType会有两种格式的载荷。



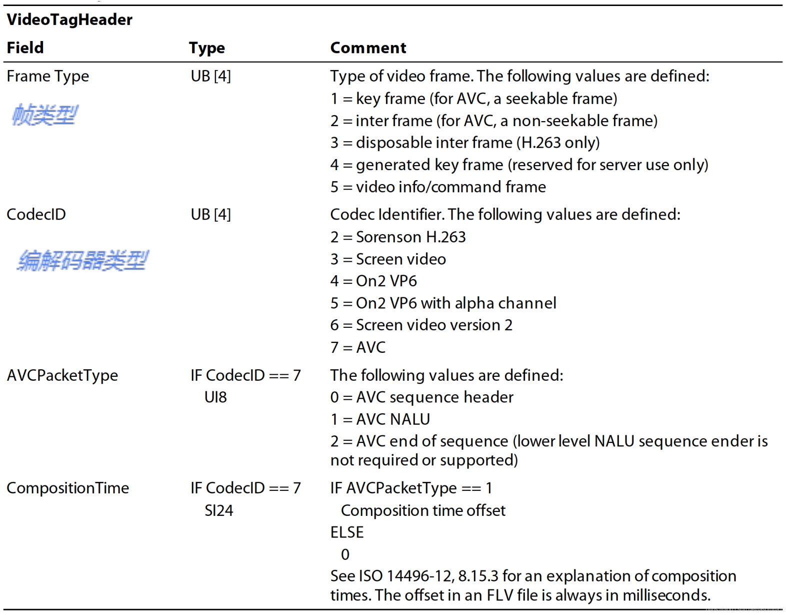
AACAUDIODATA

Field	Type	Comment

Data	IF AACPacketType == 0
	AudioSpecificConfig
	ELSE IF AACPacketType == 1
	Raw AAC frame data in UI8 [ ]
		The AudioSpecificConfig is defined in ISO 14496-3. Note that this is not the same as the contents of the esds box from an MP4/F4V file.

如果是AAC序列头，数据部分包含的是ISO 14496-3中定义的AudioSpecificConfig数据。ISO 14496就是常说的MPEG专家组定义的MPEG-4规范，其中的第三部分是音频部分，AudioSpecificConfig详细定义在Table1.15。具体AudioSpecificConfig的解析就不再这里详细描述了，简单说就是会根据AudioObjectType定义AudioObject的一些属性。常见的AudioObjectType有1:AAC Main，2:AAC LC(low complexity)，5:AAC HE(high efficiency/scale band replication)。f4v格式支持上面提到的3种类型，flv支持哪些类型规范中没有明确说明。

对于视频Tag（TagType=9）类似SWF文件格式中定义的VideoFrame tag。详细的SWF文件格式规范可以从这里找到https://www.adobe.com/devnet/swf.html。VideoTagHeader中包含了视频相关的元数据。

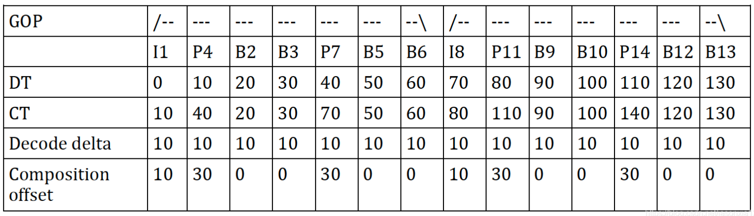


VideoTagHeader

Field	Type	Comment
Frame Type 帧类型	UB [4]	Type of video frame. The following values are defined: 1 = key frame (for AVC, a seekable frame) 2 = inter frame (for AVC, a non-seekable frame) 3 = disposable inter frame (H.263 only) 4 = generated key frame (reserved for server use only) 5 = video info/command frame
CodecID 编解码器类型	UB [4]	Codec Identifier. The following values are defined: 2 = Sorenson H.263 3 = Screen video 4 = On2 VP6 5 = On2 VP6 with alpha channel 6 = Screen video version 2 7 = AVC
AVCPacketType	IF CodecID == 7 UI8	The following values are defined: 0 = AVC sequence header 1 = AVC NALU 2 = AVC end of sequence (lower level NALU sequence ender is not required or supported)
CompositionTime	IF CodecID == 7 SI24	IF AVCPacketType == 1 Composition time offset ELSE 0 See ISO 14496-12, 8.15.3 for an explanation of composition times. The offset in an FLV file is always in milliseconds.

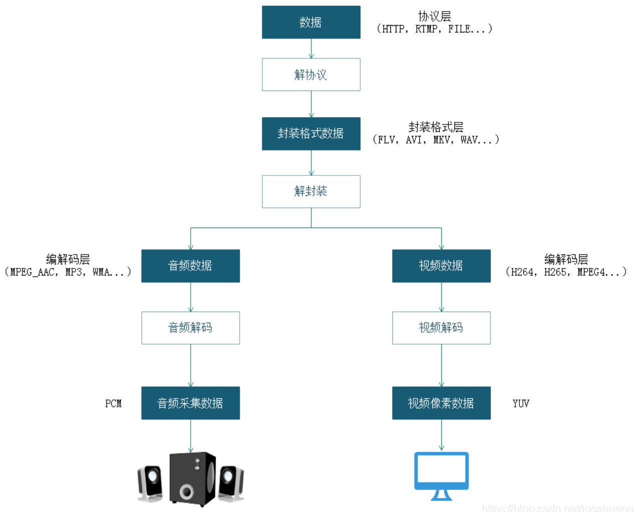
这里重点看一下AVC格式。AVC格式就指视频流是H264 AVC码流，对于AVC码流，VideoTagHeader中还包含了两个字段：AVCPacketType用来表示数据部分是序列头还是NALU（还有一种类型是end of sequence，不太清楚使用的场景）；CompositionTime用来当AVCPacketType是NALU时表示NALU的composition time offset，具体的定义在ISO 14496-12 (ISO Base Media File Format) 8.15.3，这里不太确定原协议引用的是ISO 14496-12的哪个版本，2015版本在8.15.3没有找到相应内容，在8.6.1Time to Sample Boxes里有提到相关定义，一个相关图表如下：

Table 2 — Closed GOP Example



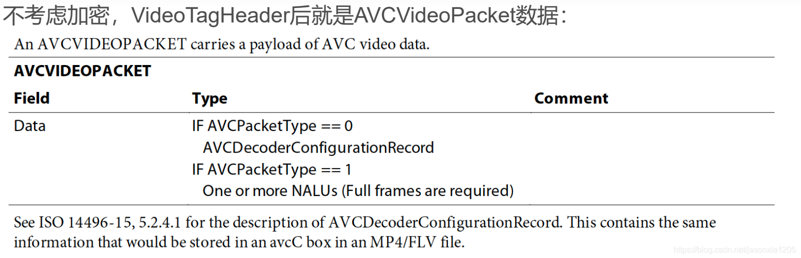
GOP	/--	---	---	---	---	---	--\	/--	---	---	---	---	---	--\
	I1	P4	B2	B3	P7	B5	B6	I8	P11	B9	B10	P14	B12	B13
DT	0	10	20	30	40	50	60	70	80	90	100	110	120	130
CT	10	40	20	30	70	50	60	80	110	90	100	140	120	130
Decode delta	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Composition offset	10	30	0	0	30	0	0	10	30	0	0	30	0	0

我理解Composition Time是指一帧组装到视频流中的时间，Composition time offset是指Composition Time和Decoding Time之间的offset，这个offset在flv中用毫秒表示。



学不可以已 关注

不考虑加密，VideoTagHeader后就是AVCVideoPacket数据：



An AVCVIDEOPACKET carries a payload of AVC video data.

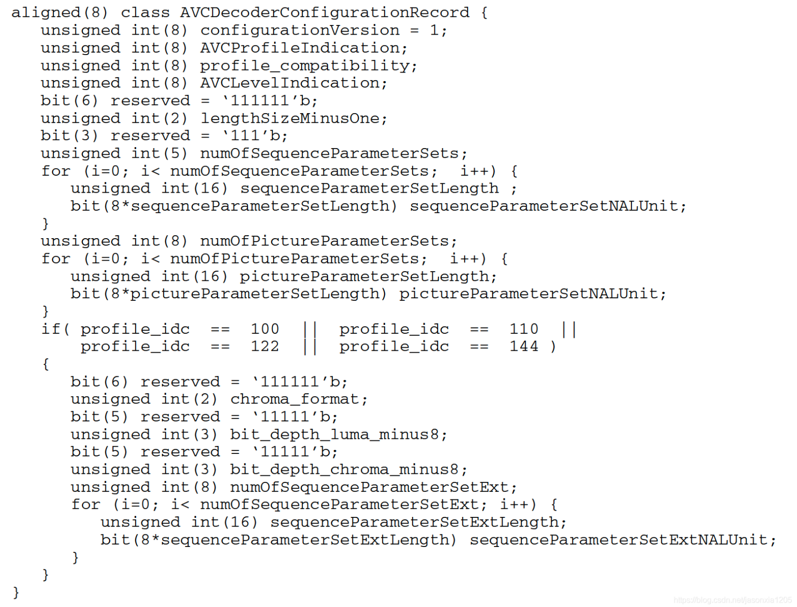
AVCVIDEOPACKET

Field	Type	Comment

Data	IF AVCPacketType == 0
	AVCDecoderConfigurationRecord
	IF AVCPacketType == 1
	One or more NALUs (Full frames are required)	

See ISO 14496-15, 5.2.4.1 for the description of AVCDecoderConfigurationRecord. This contains the same information that would be stored in an avcC box in an MP4/FLV file.

如果是AVC Sequence Header（AVCPacketType=0）则为AVCDecoderConfigurationRecord；如果是NALU类型（AVCPacketType=1）则为一个或多个NAL（必须是一个完整的帧，对于multi slice编码，需要保证将多个NAL打包到同一个Tag中）。AVCDecoderConfigurationRecord的定义在ISO 14496-15（NALU），语法如下，这里就不具体分析了。



```
aligned(8) class AVCDecoderConfigurationRecord {
   unsigned int(8) configurationVersion = 1;
   unsigned int(8) AVCProfileIndication;
   unsigned int(8) profile_compatibility;
   unsigned int(8) AVCLevelIndication;
   bit(6) reserved = '111111'b;
   unsigned int(2) lengthSizeMinusOne;
   bit(3) reserved = '111'b;
   unsigned int(5) numOfSequenceParameterSets;
   for (i=0; i< numOfSequenceParameterSets;  i++) {
      unsigned int(16) sequenceParameterSetLength ;
      bit(8*sequenceParameterSetLength) sequenceParameterSetNALUnit;
   }
   unsigned int(8) numOfPictureParameterSets;
   for (i=0; i< numOfPictureParameterSets;  i++) {
      unsigned int(16) pictureParameterSetLength;
      bit(8*pictureParameterSetLength) pictureParameterSetNALUnit;
   }
   if( profile_idc  ==  100  ||  profile_idc  ==  110  ||
       profile_idc  ==  122  ||  profile_idc  ==  144 )
   {
      bit(6) reserved = '111111'b;
      unsigned int(2) chroma_format;
      bit(5) reserved = '11111'b;
      unsigned int(3) bit_depth_luma_minus8;
      bit(5) reserved = '11111'b;
      unsigned int(3) bit_depth_chroma_minus8;
      unsigned int(8) numOfSequenceParameterSetExt;
      for (i=0; i< numOfSequenceParameterSetExt; i++) {
         unsigned int(16) sequenceParameterSetExtLength;
         bit(8*sequenceParameterSetExtLength) sequenceParameterSetExtNALUnit;
      }
   }
}
```

总体看FLV文件格式本身比较简单，但是从媒体的MetaData开始和媒体流格式密切相关，需要对媒体流编解码知识有基本的了解：例如VideoTagHeader中的CompsitionTime，和媒体容器文件相关性不大，是AVC解码相关概念。

显示推荐内容

学不可以已 关注