Audio Formats Reference

Brian Langenberger

April 19, 2010

Contents

1 Introduction

This book is intended as a reference for anyone who's ever looked at their collection of audio files and wondered how they worked. Though still a work-in-progress, my goal is to create documentation on the full decoding/encoding process of as many audio formats as possible.

Though to be honest, the audience for this is myself. I enjoy figuring out the little details of how these formats operate. And as I figure them out and implement them in Python Audio Tools, I then add some documentation here on what I've just discovered. That way, when I have to come back to something six months from now, I can return to some written documentation instead of having to go directly to my source code.

Therefore, I try to make my documentation as clear and concise as possible. Otherwise, what's the advantage over simply diving back into the source? Yet this process often turns into a challenge of its own; I'll discover that a topic I thought I'd understood wasn't so easy to grasp once I had to simplify and explain it to some hypothetical future self. Thus, I'll have to learn it better in order to explain it better.

That said, there's still much work left to do. Because it's a repository of my knowledge, it also illustrates the limits of my knowledge. Many formats are little more than "stubs", containing just enough information to extract such metadata as sample rate or bits-per-sample. These are formats in which my Python Audio Tools passes the encoding/decoding task to a binary "black-box" executable since I haven't yet taken the time to learn how to perform that work myself. But my hope is that as I learn more, this work will become more fleshed-out and widely useful.

In the meantime, by including it with Python Audio Tools, my hope is that someone else with some passing interest might also get some use out of what I've learned. And though I strive for accuracy (for my own sake, at least) I cannot guarantee it. When in doubt, consult the references on page 109 for links to external sources which may have additional information.

2 the Basics

2.1 Hexadecimal

In order to understand hexadecimal, it's important to re-familiarize oneself with decimal, which everyone reading this should be familiar with. In ordinary decimal numbers, there are a total of ten characters per digit: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Because there are ten, we'll call it base-10. So the number 675 is made up of the digits 6, 7 and 5 and can be calculated in the following way:

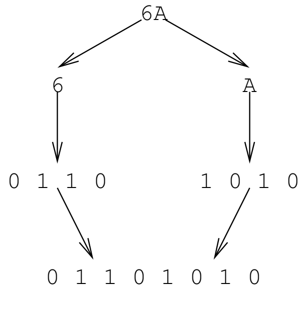
$$(6 \times 10^2) + (7 \times 10^1) + (5 \times 10^0) = 675 \tag{2.1}$$

In hexadecimal, there are sixteen characters per digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. A, B, C, D, E and F correspond to the decimal numbers 10, 11, 12, 13, 14 and 15, respectively. Because there are sixteen, we'll call it base-16. So the number 2A3 is made up of the digits 2, A and 3 and can be calculated in the following way:

$$(2 \times 16^2) + (10 \times 16^1) + (3 \times 16^0) = 675 \tag{2.2}$$

Why use hexadecimal? The reason brings us back to binary file formats, which are made up of bytes. Each byte is made up of 8 bits and can have a value from 0 to 255, in decimal. Representing a binary file in hexadecimal means a byte requires exactly two digits with values from 0 to FF. That saves us a lot of space versus trying to represent bytes in decimal.

Hexadecimal has another important property when dealing with binary data. Because each digit has 16 possible values, each hexadecimal digit represents exactly 4 bits ($16 = 2^4$). This makes it very easy to go back and forth between hexadecimal and binary. For instance, let's take the byte 6A:



Hex	Binary	Decimal	Hex	Binary	Decimal
0	0 0 0 0	0	8	1 0 0 0	8
1	0 0 0 1	1	9	1 0 0 1	9
2	0 0 1 0	2	A	1 0 1 0	10
3	0 0 1 1	3	B	1 0 1 1	11
4	0 1 0 0	4	C	1 1 0 0	12
5	0 1 0 1	5	D	1 1 0 1	13
6	0 1 1 0	6	E	1 1 1 0	14
7	0 1 1 1	7	F	1 1 1 1	15

Going from binary
to hexadecimal is a simple matter of reversing the process.

2.2 Signed integers

Signed integers are typically stored as “2’s-complement” values. To decode them, one needs to know the integer’s size in bits, its topmost (most-signficant) bit value and the value of its remaining bits.

$$\text{signed value} = \begin{cases} \text{remaining bits} & \text{if topmost bit} = 0 \\ \text{remaining bits} - (2^{\text{integer size}-1}) & \text{if topmost bit} = 1 \end{cases} \tag{2.3}$$

For example, take an 8-bit integer whose bit values are `00000101`. Since the topmost bit is `0`, its value is simply `0000101`, which is 5 in base-10 ($2^2 + 2^0 = 5$).

Next, let’s take an integer whose bit values are `11111011`. Its topmost bit is `1` and its remaining bits are `1111011`, which is 123 in base-10 ($2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = 123$). Therefore:

$$\begin{aligned} \text{signed value} &= 123 - 2^{8-1} && (2.4) \\ &= 123 - 128 && (2.5) \\ &= -5 && (2.6) \end{aligned}$$

Transforming a signed integer into its unsigned 2’s-complement value is a simple matter of reversing the process.

$$\text{unsigned value} = \begin{cases} \text{signed value} & \text{if signed value} \geq 0 \\ 2^{\text{integer size}} - (-\text{signed value}) & \text{if signed value} < 0 \end{cases} \tag{2.7}$$

For example, let’s convert the value -20 to a signed, 8-bit integer:

$$\begin{aligned} \text{unsigned value} &= 2^8 - (- - 20) && (2.8) \\ &= 256 - 20 && (2.9) \\ &= 236 && (2.10) \end{aligned}$$

which is `11101100` in binary ($2^7 + 2^6 + 2^5 + 2^3 + 2^2 = 236$).

2.3 Endianness

You will need to know about endianness anytime a single value spans multiple bytes. As an example, let's take the first 16 bytes of a small RIFF WAVE file:

```
52 49 46 46 54 9b 12 00  57 41 56 45 66 6d 74 20
```

The first four bytes are the ASCII string 'RIFF' (`0x52 0x49 0x46 0x46`). The next four bytes are a 32-bit unsigned integer which is a size value. Reading from left to right, that value would be `0x549B1200`. That's almost 1.5 gigabytes. Since this file is nowhere near that large, we're clearly not reading those bytes correctly.

The key is that RIFF WAVE files are 'little endian'. In plain English, that means we have to read in those bytes from right to left. Thus, the value is actually `0x00129B54`. That's a little over 1 megabyte, which is closer to our expectations.

Remember that little endian reverses the bytes, not the hexadecimal digits. Simply reversing the string to `0x0021B945` is not correct.

When converting a signed, little-endian value to an integer, the 2's-complement decoding comes *after* one performs the endianness reversing. For example, given a signed 16-bit little-endian value of `0xFBFF`, one firsts reorders the bytes to `0xFFFB` before decoding it to a signed value ($32763 - 2^{15} = -5$).

Conversely, when converting a signed integer to a little-endian value, the endian reversing comes *after* one performs the 2's-complement encoding.

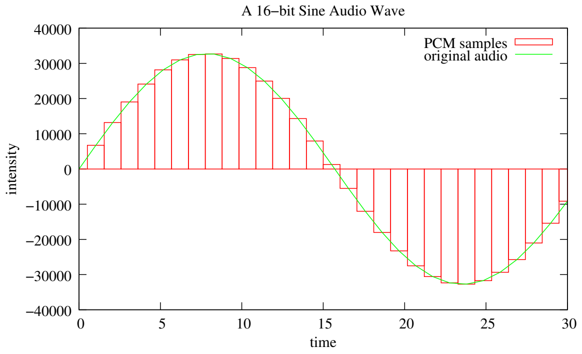
2.4 Character Encodings

Many audio formats store metadata, which contains information about the song's name, artist, album and so forth. This information is stored as text, but it's important to know what sort of text in order to read it and display it properly.

As an example, take the simple character `é`. In latin-1 encoding, it is stored as a single byte `0xE9`. In UTF-8 encoding, it is stored as the bytes `0xC3A9`. In UTF-16BE encoding, it is stored as the bytes `0x00E9`.

Although decoding and encoding text is a complex subject beyond the scope of this document, you must always be aware that metadata may not be 7-bit ASCII text and should handle it properly in whatever encoding is supported by the metadata formats. Look to your programming tools for libraries to assist in Unicode text handling.

2.5 PCM



Pulse-Code Modulation is a method for transforming an analog audio signal into a digital representation. It takes that signal, 'samples' its intensity at discrete intervals and yields a stream of signed integer values. By replaying those values to a speaker at the same speed and intensity, a close approximation of the original signal is produced.

Let's take some example bytes from a CD-quality PCM stream:

```
1B 00 43 FF  1D 00 45 FF  1C 00 4E FF  1E 00 59 FF
```

CD-quality is 16-bit, 2 channel, 44100Hz. 16-bit means those bytes are split into 16-bit signed, little-endian samples. Therefore, our bytes are actually the integer samples:

```
27 -189 29 -187 28 -178 30 -167
```

The number of channels indicates how many speakers the signal supports. 2 channels means the samples are sent to 2 different speakers. PCM interleaves its samples, sending one sample to each channel simultaneously before moving on to the next set. In the case of 2 channels, the first sample is sent to the left speaker and the second is sent to the right speaker. So, our stream of data now looks like:

left speaker	right speaker
27	-189
29	-187
28	-178
30	-167

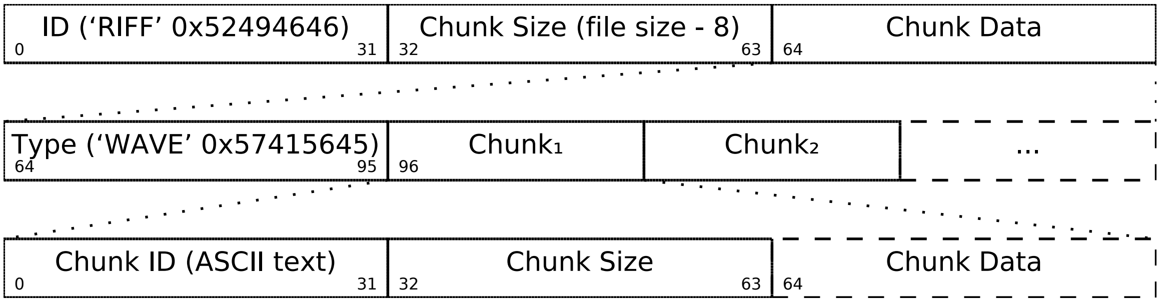
44100Hz means those pairs of samples are sent at the rate of 44100 per second. Thus, our set of 4 samples takes precisely 1/11025th of a second when replayed.

A channel-independent block of samples is commonly referred to as a 'frame'. In this example, we have a total of 4 PCM frames. However, the term 'frame' appears a lot in digital audio. It is important not to confuse a PCM frame with a CD frame (a block of audio 1/75th of a second long), an MP3 frame, a FLAC frame or any other sort of frame.

3 Waveform Audio File Format

The Waveform Audio File Format is the most common form of PCM container. What that means is that the file is mostly PCM data with a small amount of header data to tell applications what format the PCM data is in. Since RIFF WAVE originated on Intel processors, everything in it is little-endian.

3.1 the RIFF WAVE Stream



‘Chunk Size’ is the total size of the chunk, minus 8 bytes for the chunk header.

3.2 the Classic ‘fmt’ Chunk

Wave files with 2 channels or less, and 16 bits-per-sample or less, use a classic ‘fmt’ chunk to indicate its PCM data format. This chunk is required to appear before the ‘data’ chunk.

Chunk ID (‘fmt ’ 0x666D7420) 0 31	Chunk Size (16) 32 63
Compression Code (0x0001) 64 79	Channel Count 80 95
Sample Rate 96 127	
Average Bytes per Second 128 159	
Block Align 160 175	Bits per Sample 176 191

$$\text{Average Bytes per Second} = \frac{\text{Sample Rate} \times \text{Channel Count} \times \text{Bits per Sample}}{8} \tag{3.1}$$

$$\text{Block Align} = \frac{\text{Channel Count} \times \text{Bits per Sample}}{8} \tag{3.2}$$

3.3 the WAVEFORMATEXTENSIBLE ‘fmt’ Chunk

Wave files with more than 2 channels or more than 16 bits-per-sample should use a WAVEFORMATEXTENSIBLE ‘fmt’ chunk which contains additional fields for channel assignment.

<table>
<tr><td colspan="2">Chunk ID (‘fmt ’ 0x666D7420)
0 31</td><td colspan="2">Chunk Size (40)
32 63</td></tr>
<tr><td colspan="2">Compression Code (0xFFFE)
64 79</td><td colspan="2">Channel Count
80 95</td></tr>
<tr><td colspan="4">Sample Rate
96 127</td></tr>
<tr><td colspan="4">Average Bytes per Second
128 159</td></tr>
<tr><td colspan="2">Block Align
160 175</td><td colspan="2">Bits per Sample
176 191</td></tr>
<tr><td colspan="2">CB Size (22)
192 207</td><td colspan="2">Valid Bits per Sample
208 223</td></tr>
<tr><td>Front Right of Center
224</td><td>Front Left of Center
225</td><td>Rear Right
226</td><td>Rear Left
227</td></tr>
<tr><td>LFE
228</td><td>Front Center
229</td><td>Front Right
230</td><td>Front Left
231</td></tr>
<tr><td>Top Back Left
232</td><td>Top Front Right
233</td><td>Top Front Center
234</td><td>Top Front Left
235</td></tr>
<tr><td>Top Center
236</td><td>Side Right
237</td><td>Side Left
238</td><td>Back Center
239</td></tr>
<tr><td>Undefined
240 245</td><td>Top Back Right
246</td><td>Top Back Center
247</td><td>Undefined
248 255</td></tr>
<tr><td colspan="4">Sub Format (0x0100000000001000800000aa00389b71)
256 383</td></tr>
</table>

Note that the ‘Average Bytes per Second’ and ‘Block Align’ fields are calculated the same as a classic fmt chunk.

3.4 the ‘data’ Chunk

<table>
<tr><td>Chunk ID (‘data’ 0x64617461)
0 31</td><td>Chunk Size
32 63</td></tr>
<tr><td colspan="2">PCM Data
64</td></tr>
</table>

‘PCM Data’ is a stream of PCM samples stored in little-endian format.

3.5 Channel assignment

Channels whose bits are set in the WAVEFORMATEXTENSIBLE ‘fmt’ chunk appear in the following order:

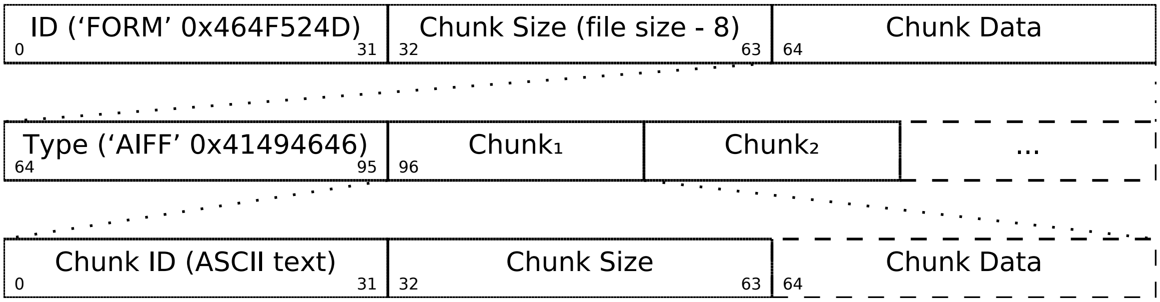
Index	Channel	Mask Bit
1	Front Left	`0x1`
2	Front Right	`0x2`
3	Front Center	`0x4`
4	LFE	`0x8`
5	Back Left	`0x10`
6	Back Right	`0x20`
7	Front Left of Center	`0x40`
8	Front Right of Center	`0x80`
9	Back Center	`0x100`
10	Side Left	`0x200`
11	Side Right	`0x400`
12	Top Center	`0x800`
13	Top Front Left	`0x1000`
14	Top Front Center	`0x2000`
15	Top Front Right	`0x4000`
16	Top Back Left	`0x8000`
17	Top Back Center	`0x10000`
18	Top Back Right	`0x20000`

For example, if the file’s channel mask is set to `0x33`, it contains the channels ‘Front Left’, ‘Front Right’, ‘Back Left’ and ‘Back Right’, in that order.

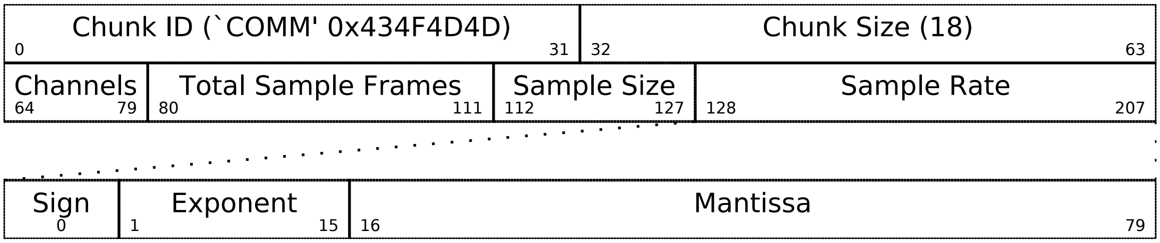
4 Audio Interchange File Format

AIFF is the Audio Interchange File Format. It is popular on Apple computers and is a precursor to the more widespread WAVE format. All values in AIFF are stored as big-endian.

4.1 the AIFF file stream



4.2 the COMM chunk



The Sample Rate field is an 80-bit IEEE Standard 754 floating point value instead of the big-endian integers common to all the other fields.

$$\text{Value} = (-)\frac{\text{Mantissa}}{2^{63}} \times 2^{\text{Exponent}-16383} \tag{4.1}$$

For example, given a sign bit of 0, an exponent value of 0x400E and a mantissa value of 0xAC44000000000000:

$$\begin{aligned} \text{Value} &= \frac{12413046472939929600}{2^{63}} \times 2^{16398-16383} && (4.2) \\ &= 1.3458251953125 \times 2^{15} && (4.3) \\ &= 44100.0 && (4.4) \end{aligned}$$

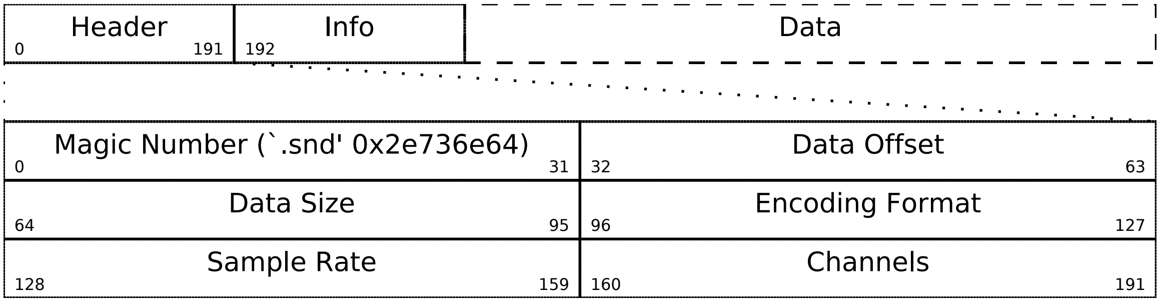
4.3 the SSND chunk

Chunk ID (`SSND' 0x53534E44) 0 31	Chunk Size 32 63
Offset 64 95	Block Size 96 127
PCM Data 128	

5 Sun AU

The AU file format was invented by Sun Microsystems and also used on NeXT systems. All values in AU are stored as big-endian. It supports a wide array of data formats, including µ-law logarithmic encoding, but can also be used as a PCM container.

5.1 the Sun AU file stream



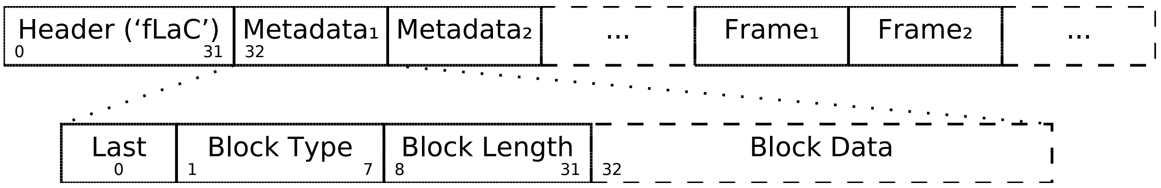
value	encoding format
1	8-bit G.711 µ-law
2	8-bit linear PCM
3	16-bit linear PCM
4	24-bit linear PCM
5	32-bit linear PCM
6	32-bit IEEE floating point
7	64-bit IEEE floating point
8	Fragmented sample data
9	DSP program
10	8-bit fixed point
11	16-bit fixed point
12	24-bit fixed point
13	32-bit fixed point
18	16-bit linear with emphasis
19	16-bit linear compressed
20	16-bit linear with emphasis and compression
21	Music kit DSP commands
23	4-bit ISDN µ-law compressed using the ITU-T G.721 ADPCM voice data encoding scheme
24	ITU-T G.722 ADPCM
25	ITU-T G.723 3-bit ADPCM
26	ITU-T G.723 5-bit ADPCM
27	8-bit G.711 A-law

6 Free Lossless Audio Codec

FLAC compresses PCM audio data losslessly using predictors and a residual. FLACs contain checksumming to verify their integrity, contain comment tags for metadata and are streamable.

Except for the contents of the VORBIS_COMMENT metadata block, everything in FLAC is big-endian.

6.1 the FLAC file Stream

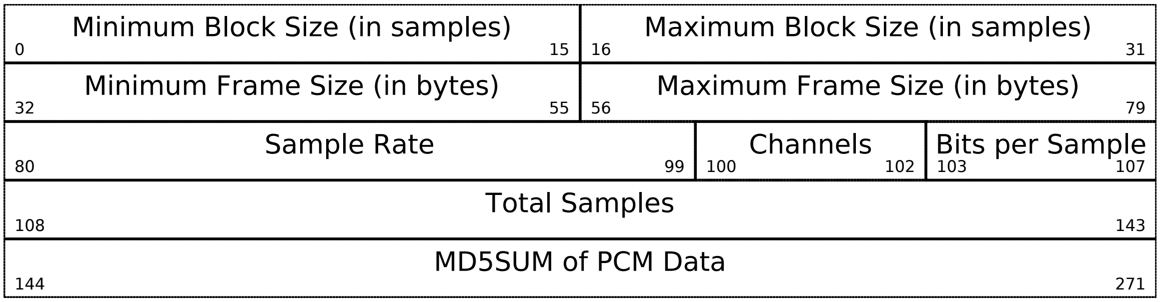


"Last" is 0 when there are no additional metadata blocks and 1 when it is the final block before the the audio frames. "Block Length" is the size of the metadata block data to follow, not including the header.

Block Type	Block
0	STREAMINFO
1	PADDING
2	APPLICATION
3	SEEKTABLE
4	VORBIS_COMMENT
5	CUESHEET
6	PICTURE
7-126	reserved
127	invalid

6.2 FLAC Metadata Blocks

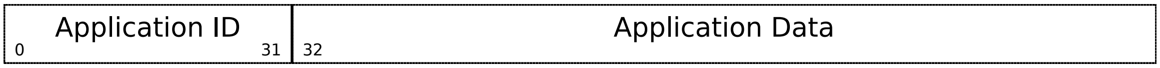
6.2.1 STREAMINFO


Minimum Block Size (in samples)
0
15
Maximum Block Size (in samples)
16
31
Minimum Frame Size (in bytes)
32
55
Maximum Frame Size (in bytes)
56
79
Sample Rate
80
99
Channels
100
102
Bits per Sample
103
107
Total Samples
108
143
MD5SUM of PCM Data
144
271

6.2.2 PADDING

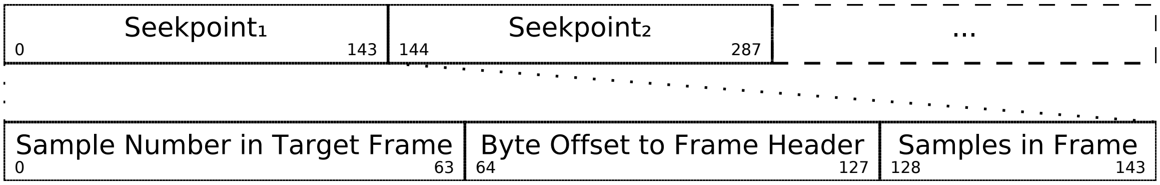
PADDING is simply a block full of NULL (`0x00`) bytes. Its purpose is to provide extra metadata space within the FLAC file. By having a padding block, other metadata blocks can be grown or shrunk without having to rewrite the entire FLAC file by removing or adding space to the padding.

6.2.3 APPLICATION

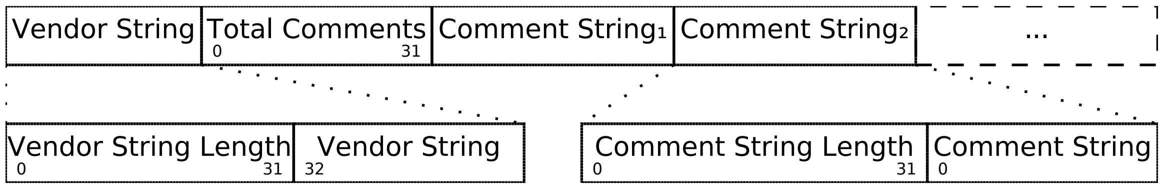


APPLICATION is a general-purpose metadata block used by a variety of different programs. Its contents are defined by the ASCII Application ID value.

6.2.4 SEEKTABLE



6.2.5 VORBIS_COMMENT



The length fields are all little-endian. The Vendor String and Comment Strings are all UTF-8 encoded. Keys are not case-sensitive and may occur multiple times, indicating multiple values for the same field. For instance, a track with multiple artists may have more than one `ARTIST`.

ALBUM album name

ARTIST artist name, band name, composer, author, etc.

CATALOGNUMBER* CD spine number

COMPOSER* the work's author

CONDUCTOR* performing ensemble's leader

COPYRIGHT copyright attribution

DATE recording date

DESCRIPTION a short description

DISCNUMBER* disc number for multi-volume work

ENGINEER* the recording masterer

ENSEMBLE* performing group

GENRE a short music genre label

GUEST ARTIST* collaborating artist

ISRC ISRC number for the track

LICENSE license information

LOCATION recording location

OPUS* number of the work

ORGANIZATION record label

PART* track's movement title

PERFORMER performer name, orchestra, actor, etc.

PRODUCER* person responsible for the project

PRODUCTNUMBER* UPC, EAN, or JAN code

PUBLISHER* album's publisher

RELEASE DATE* date the album was published

REMIXER* person who created the remix

SOURCE ARTIST* artist of the work being performed

SOURCE MEDIUM* CD, radio, cassette, vinyl LP, etc.

SOURCE WORK* a soundtrack's original work

SPARS* DDD, ADD, AAD, etc.

SUBTITLE* for multiple track names in a single file

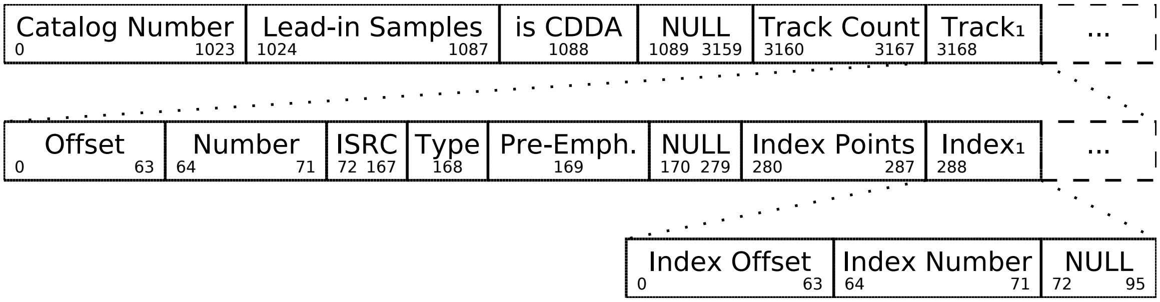
TITLE track name

TRACKNUMBER track number

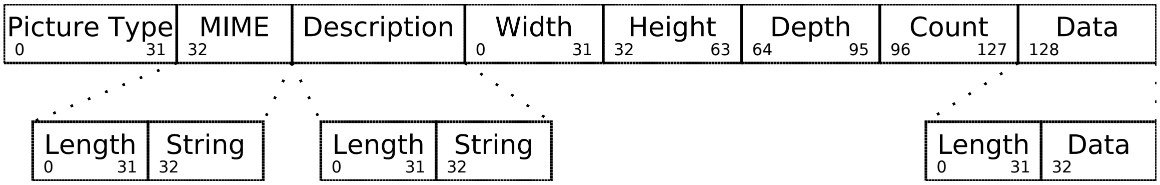
VERSION track version

Fields marked with * are proposed extension fields and not part of the official Vorbis comment specification.

6.2.6 CUESHEET



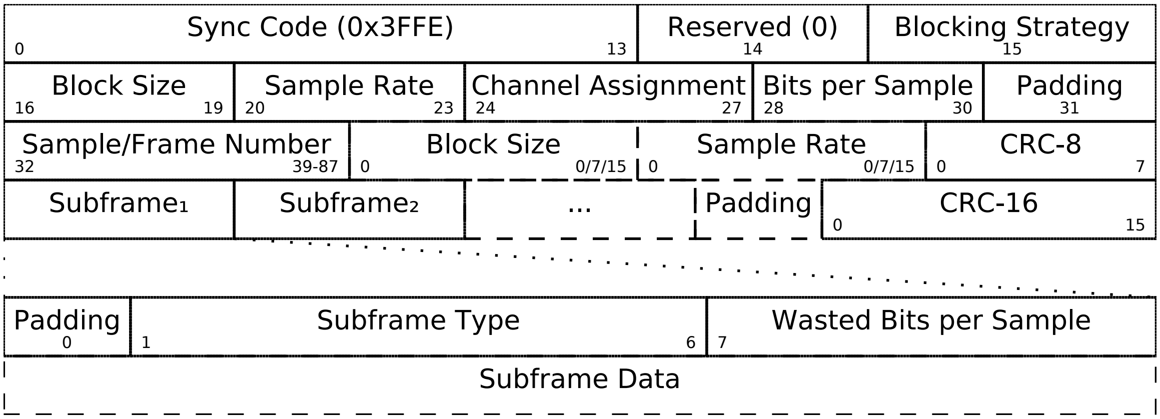
6.2.7 PICTURE



Picture Type	Type
0	Other
1	32x32 pixels ‘file icon’ (PNG only)
2	Other file icon
3	Cover (front)
4	Cover (back)
5	Leaflet page
6	Media (e.g. label side of CD)
7	Lead artist / Lead performer / Soloist
8	Artist / Performer
9	Conductor
10	Band / Orchestra
11	Composer
12	Lyricist / Text writer
13	Recording location
14	During recording
15	During performance
16	Movie / Video screen capture
17	A bright coloured fish
18	Illustration
19	Band / Artist logotype
20	Publisher / Studio logotype

6.3 FLAC Decoding

A FLAC stream is made up of individual FLAC frames, as follows:



Value	Block Size	Sample Rate	Channels	Assignment	Bits per Sample	Value
0000	STREAMINFO	STREAMINFO	1	front center	STREAMINFO	0000
0001	192	88200	2	front left, front right	8	0001
0010	576	176400	3	f. left, f. right, f. center	12	0010
0011	1152	192000	4	f. left, f. right, back left, back right	reserved	0011
0100	2304	8000	5	f. L, f. R, f. C, b. L, b. R	16	0100
0101	4608	16000	6	f. L, f. R, f. C, LFE, b. L, b. R	20	0101
0110	8 bits (+1)	22050	7	undefined	24	0110
0111	16 bits (+1)	24000	8	undefined	reserved	0111
1000	256	32000	2	0 left, 1 difference		1000
1001	512	44100	2	0 difference, 1 right		1001
1010	1024	48000	2	0 average, 1 difference		1010
1011	2048	96000		reserved		1011
1100	4096	8 bits (in kHz)		reserved		1100
1101	8192	16 bits (in Hz)		reserved		1101
1110	16384	16 bits (in 10s of Hz)		reserved		1110
1111	32768	invalid		reserved		1111

Sample/Frame Number is a UTF-8 coded value. If the blocking strategy is 0, it decodes to a 32-bit frame number. If the blocking strategy is 1, it decodes to a 36-bit sample number.

There is one Subframe per channel.

'Wasted Bits Per Sample' is typically a single bit set to 0, indicating no wasted bits per sample. If set to 1, a unary-encoded value follows which indicates how many bits are wasted per sample.

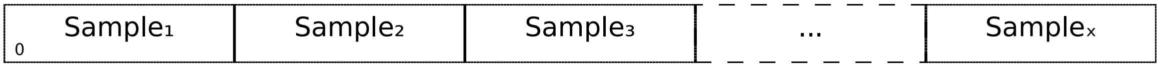
Padding is added as needed between the final subframe and CRC-16 in order to byte-align frames.

Value	Subframe Type
000000	SUBFRAME_CONSTANT
000001	SUBFRAME_VERBATIM
00001x	reserved
0001xx	reserved
001xxx	SUBFRAME_FIXED
	xxx = predictor order
01xxxx	reserved
1xxxxx	SUBFRAME_LPC
	xxxxx = predictor order - 1

6.3.1 CONSTANT subframe

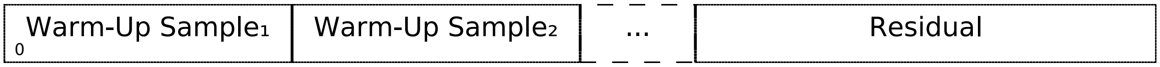
This is the simplest possible subframe. It consists of a single value whose size is equal to the subframe's 'Bits per Sample'. For instance, a 16-bit subframe would have CONSTANT subframes 16 bits in length. The value of the subframe is the value of all samples the subframe contains. An obvious use of this subframe is to store an entire subframe's worth of digital silence (samples with a value of 0) very efficiently.

6.3.2 VERBATIM subframe



This subframe's length is equal to the subframe's 'Bits per Sample' multiplied by the frame's 'Block Size'. Since it does no compression whatsoever and simply stores audio samples as-is, this subframe is only suitable for especially noisy portions of a track where no suitable predictor can be found.

6.3.3 FIXED subframe



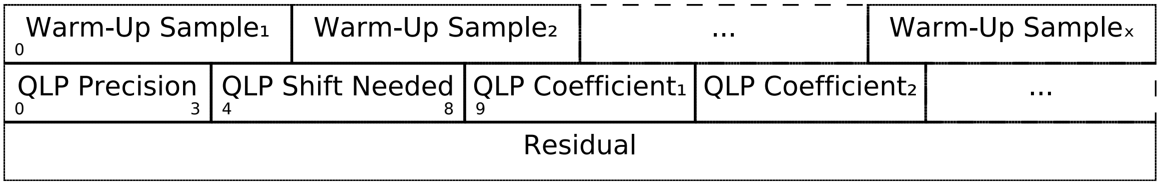
The number of warm-up samples equals the 'Predictor Order' (which is encoded in the 'Subframe Type'). Each warm-up sample is the same size as the subframe's 'Bits per Sample'. These samples are sent out as-is; they are the subframe's 'starting point' upon which further samples build when decompressing the stream. Determining the value of the current sample is then a matter of looking backwards at previously decoded samples (or warm-up samples), applying a simple formula on their values (which depends on the Predictor Order) and adding the residual.

Order	Calculation
0	$Sample_i = Residual_i$
1	$Sample_i = Sample_{i-1} + Residual_i$
2	$Sample_i = (2 \times Sample_{i-1}) - Sample_{i-2} + Residual_i$
3	$Sample_i = (3 \times Sample_{i-1}) - (3 \times Sample_{i-2}) + Sample_{i-3} + Residual_i$
4	$Sample_i = (4 \times Sample_{i-1}) - (6 \times Sample_{i-2}) + (4 \times Sample_{i-3}) - Sample_{i-4} + Residual_i$

Let's run through a simple example in which the Predictor Order is 1. Note that residual does not apply to warm-up samples. How to extract the encoded residual will be covered in a later section.

Index	Residual	Sample
0		(warm-up) **10**
1	1	$10 + 1 = \mathbf{11}$
2	2	$11 + 2 = \mathbf{13}$
3	-2	$13 - 2 = \mathbf{11}$
4	1	$11 + 1 = \mathbf{12}$
5	-1	$12 - 1 = \mathbf{11}$

6.3.4 LPC Subframe



Warm-Up Sample₁ 0 | Warm-Up Sample₂ | ... | Warm-Up Sampleₓ

QLP Precision 0 3 | QLP Shift Needed 4 8 | QLP Coefficient₁ 9 | QLP Coefficient₂ | ...

Residual

The number of warm-up samples equals the ‘LPC Order’ (which is encoded in the ‘Subframe Type’). The size of each QLP Coefficient is equal to ‘QLP Precision’ number of bits, plus 1. ‘QLP Shift Needed’ and the value of each Coefficient are signed two’s-complement integers. The number of Coefficients equals the ‘LPC Order’.

$$Sample_i = \left\lfloor \frac{\sum_{j=0}^{Order-1} \text{QLP Coeffcient}_j \times \text{Sample}_{i-j-1}}{2^{\text{QLP Shift Needed}}} \right\rfloor + Residual_i \tag{6.1}$$

This simply means we’re taking the sum of the calculated values from 0 to Order - 1, bit-shifting that sum down and added the residual when determining the current sample. Much like the FIXED subframe, LPC subframes also contain warm-up samples which serve as our calculation’s starting point.

In this example, the LPC Order is 5, the QLP Shift Needed is 9 and the encoded Coefficients are as follows:

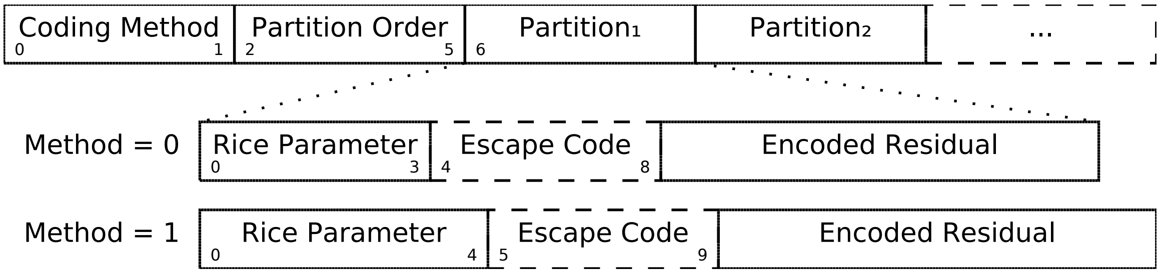
QLP Coefficient$_0$	1241
QLP Coefficient$_1$	-944
QLP Coefficient$_2$	14
QLP Coefficient$_3$	342
QLP Coefficient$_4$	-147

Index	Residual	Sample
0		(warm-up) **1053**
1		(warm-up) **1116**
2		(warm-up) **1257**
3		(warm-up) **1423**
4		(warm-up) **1529**
5	11	$(1241 \times 1529) + (-944 \times 1423) + (14 \times 1257) + (342 \times 1116) + (-147 \times 1053) = 798656$ $\lfloor 798656 \div 2^9 \rfloor = 1559 + 11 = \mathbf{1570}$
6	79	$(1241 \times 1570) + (-944 \times 1529) + (14 \times 1423) + (342 \times 1257) + (-147 \times 1116) = 790758$ $\lfloor 790758 \div 2^9 \rfloor = 1544 + 79 = \mathbf{1623}$
7	24	$(1241 \times 1623) + (-944 \times 1570) + (14 \times 1529) + (342 \times 1423) + (-147 \times 1257) = 855356$ $\lfloor 855356 \div 2^9 \rfloor = 1670 + 24 = \mathbf{1694}$
8	-81	$(1241 \times 1694) + (-944 \times 1623) + (14 \times 1570) + (342 \times 1529) + (-147 \times 1423) = 905859$ $\lfloor 905859 \div 2^9 \rfloor = 1769 - 81 = \mathbf{1688}$
9	-72	$(1241 \times 1688) + (-944 \times 1694) + (14 \times 1623) + (342 \times 1570) + (-147 \times 1529) = 830571$ $\lfloor 830571 \div 2^9 \rfloor = 1622 - 72 = \mathbf{1550}$

In this instance, division should always round down and *not* towards zero.

6.3.5 the Residual

Though the FLAC format allows for different forms of residual coding, two forms of partitioned Rice are the only ones currently supported. The difference between the two is that when ‘Coding Method’ is 0, the Rice Parameter in each partition is 4 bits. When the ‘Coding Method’ is 1, that parameter is 5 bits.



There are $2^{\text{Partition Order}}$ number of Partitions. The number of decoded samples in a Partition depends on the its position in the subframe. The first partition in the subframe contains:

$$\text{Total Samples} = \frac{\text{Frame's Block Size}}{2^{\text{Partition Order}}} - \text{Predictor Order} \tag{6.2}$$

Subsequent partitions contain:

$$\text{Total Samples} = \frac{\text{Frame's Block Size}}{2^{\text{Partition Order}}} \tag{6.3}$$

Unless the Partition Order is 0. In that case:

$$\text{Total Samples} = \text{Frame's Block Size} - \text{Predictor Order} \tag{6.4}$$

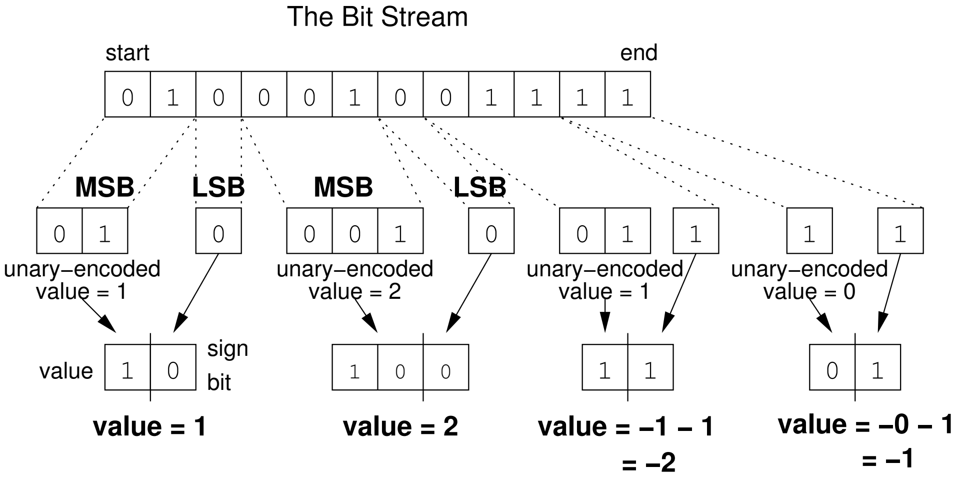
since there is only one partition which takes up the entire block.

If all of the bits in ‘Rice Parameter’ are set, the partition is unencoded binary using ‘Escape Code’ number of bits per sample.

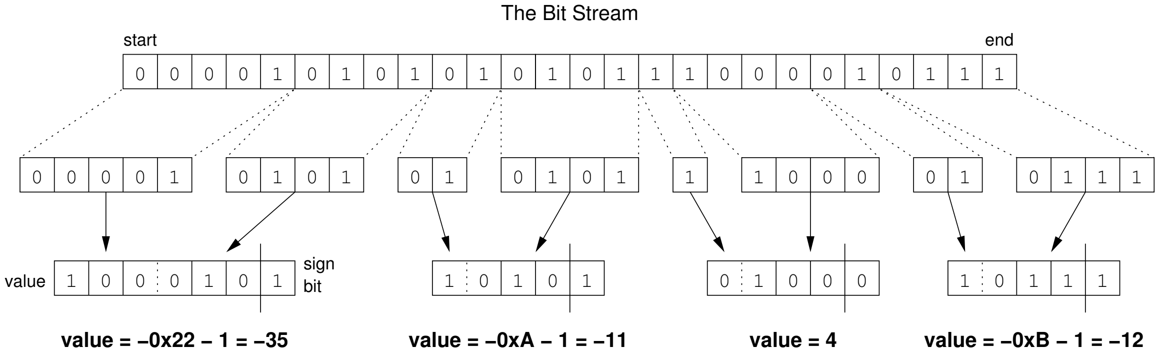
Rice Encoding

The residual uses Rice coding to compress lots of mostly small values in a very small amount of space. To decode it, one first needs the Rice parameter. Take a unary-encoded value[1] from the bit stream, which are our most significant bits (MSB). Then take 'parameter' number of additional bits, which are our least significant bits (LSB). Combine the two sets into our new value, making the MSB set as the high bits and the LSB set as the low bits. Bit 0 of this new value is the sign bit. If it is 0, the actual value is equal to the rest of the bits. If it is 1, the actual value is equal to the rest of the bits, multiplied by -1 and minus 1.

This is less complicated than it sounds, so let's run through an example in which the Rice parameter is 1:



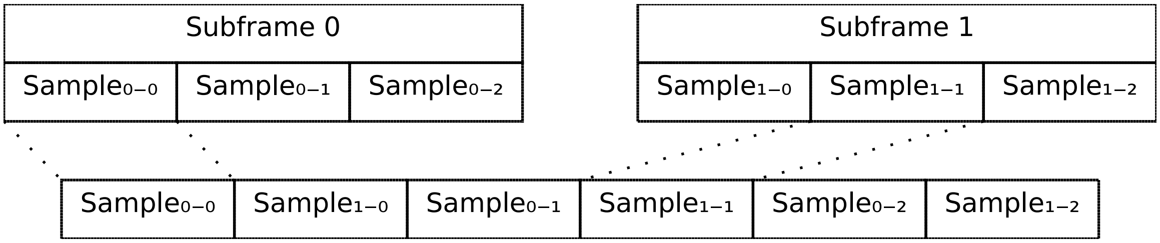
Now, let's run through another example in which the Rice parameter is 4:



[1] In this instance, unary-encoding is a simple matter of counting the number of 0 bits before the next 1 bit. The resulting sum is the value.

6.3.6 Channels

Since most audio has more than one channel, it is important to understand how FLAC handles putting it back together. When channels are stored independently, one simply interleaves them together in the proper order. Let's take an example of 2 channel, 16-bit audio stored this way:



This is the simplest case. However, in the case of difference channels, one subframe will contain actual channel data and the other channel will contain signed difference data which is applied to that actual data in order to reconstruct both channels. It's very important to remember that the difference channel has 1 additional bit per sample which will be consumed during reconstruction. Why 1 additional bit? Let's take an example where the left sample's value is -30000 and the right sample's value is +30000. Storing this pair as left + difference means the left sample remains -30000 and the difference is -60000 $(-30000 - -60000 = +30000)$. -60000 won't fit into a 16-bit signed integer. Adding that 1 additional bit doubles our range of values and that's just enough to cover any possible difference between two samples.

Assignment	Channel 0	Channel 1	Left Channel	Right Channel
`1000`	left	difference	left	left − difference
`1001`	difference	right	right + difference	right
`1010`	mid	side	$(((\text{mid} \ll 1)\vert(\text{side}\&1)) + \text{side}) \gg 1$	$(((\text{mid} \ll 1)\vert(\text{side}\&1)) - \text{side}) \gg 1$

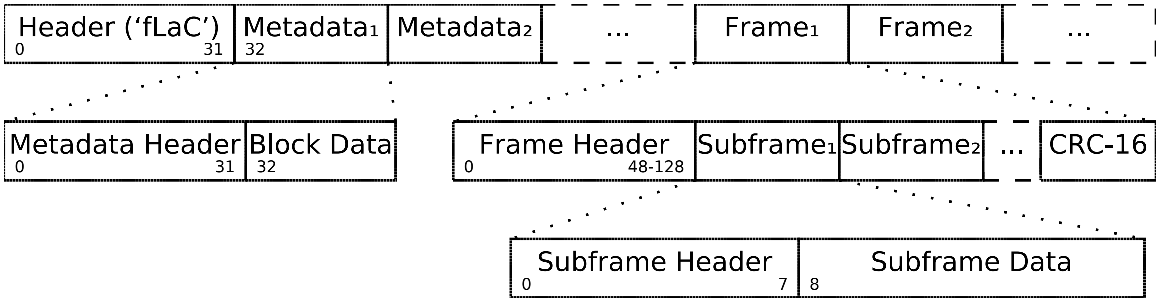
The mid channel case is another unusual exception. We're prepending the mid channel with bit 0 from the side channel, performing the addition/subtraction and then discarding that bit before assigning the results to the left and right channels.

6.3.7 Wasted Bits per Sample

Though rare in practice, FLAC subframes support 'wasted bits per sample'. Put simply, these wasted bits are removed during subframe calculation and restored to the subframe's least significant bits as zero value bits when it is returned. For instance, a subframe with 1 wasted bit per sample in a 16-bit FLAC stream is treated as having only 15 bits per sample when reading warm-up samples and then all through the rest of the subframe calculation. That wasted zero bit is then prepended to each sample prior to returning the subframe.

6.4 FLAC Encoding

For the purposes of discussing FLAC encoding, we'll assume one has a stream of input PCM values along with the stream's sample rate, number of channels and bits per sample. Creating a valid FLAC file is then a matter of writing the proper file header, metadata blocks and FLAC frames.



bits	value
1	0 if additional metadata blocks follow, 1 if not
7	0 for STREAMINFO, 1 for PADDING, 4 for VORBIS_COMMENT, etc.
24	the length of the block data in bytes, not including the header

Figure 6.1: Metadata Header

6.4.1 the STREAMINFO metadata block

bits	value
16	the minimum FLAC frame size, in PCM frames
16	the maximum FLAC frame size, in PCM frames
24	the minimum FLAC frame size, in bytes
24	the maximum FLAC frame size, in bytes
20	the stream's sample rate, in Hz
3	the stream's channel count, minus one
5	the stream's bit-per-sample, minus one
36	the stream's total number of PCM frames
128	an MD5 sum of the PCM stream's bytes

This metadata block must come first and is the only required block in a FLAC file.

When encoding a FLAC file, many of these fields cannot be known in advance. Instead, one must keep track of those values during encoding and then rewrite the STREAMINFO block when finished.

6.4.2 Frame header

bits	value
14	`0x3FFE` sync code
1	`0` reserved
1	`0` if the header encodes the frame number, `1` if it encodes the sample number
4	this frame's block size, as encoded PCM frames[a]
4	this frame's encoded sample rate[a]
4	this frame's encoded channel assignment[a]
3	this frame's encoded bits per sample[a]
1	`0` padding
8-56	the frame number, or sample number, UTF-8 encoded and starting from 0
0/8/16	the number of PCM frames (minus one) in this FLAC frame
	if block size is `0x6` (8 bits) or `0x7` (16 bits)
0/8/16	the sample rate of this FLAC frame
	if sample rate is `0xC` (8 bits), `0xD` (16 bits) or `0xE` (16 bits)
8	the CRC-8 of all data from the beginning of the frame header

[a] See table on page 25

6.4.3 Channel assignment

If the input stream has a number of channels other than 2, one has no choice but to store them independently. If the number of channels equals 2, one can try all four possible assignments (left-difference, difference-right, mid-side and independent) and use the one which takes the least amount of space.

6.4.4 Subframe header

bits	value
1	`0` padding
`000000`	`SUBFRAME_CONSTANT`
`000001`	`SUBFRAME_VERBATIM`
`001xxx`	`SUBFRAME_FIXED` (xxx = Predictor Order)
`1xxxxx`	`SUBFRAME_LPC` (xxxxx = Predictor Order - 1)
1	`0` if no wasted bits per sample, `1` if a unary-encoded number follows
0+	the number of wasted bits per sample (minus one) encoded as unary

6.4.5 the CONSTANT subframe

If all the samples in a subframe are identical, one can encode them using a `CONSTANT` subframe, which is essentially a single sample value that gets duplicated ‘block size’ number of times when decoded.

6.4.6 the VERBATIM subframe

This subframe simply stores all the samples as-is, with no compression whatsoever. It is a fallback encoding method for when no other subframe makes one’s data any smaller.

6.4.7 the FIXED subframe

This subframe consists of ‘predictor order’ number of unencoded warm-up samples followed by a residual. Determining which predictor order to use on a given set of input samples depends on their minimum delta sum. This process is best explained by example:

Note that the numbers in italics play a part in the delta calculation to their right, but do **not** figure into the delta’s absolute value sum, below.

In this example, Δ^1’s value of 26 is the smallest. Therefore, when compressing this set of samples in a FIXED subframe, it’s best to use a predictor order of 1.

The predictor order indicates how many warm-up samples to take from the PCM stream. Determining the residual values can then be done automatically based on the current Sample_i and previously encoded samples, or warm-up samples.

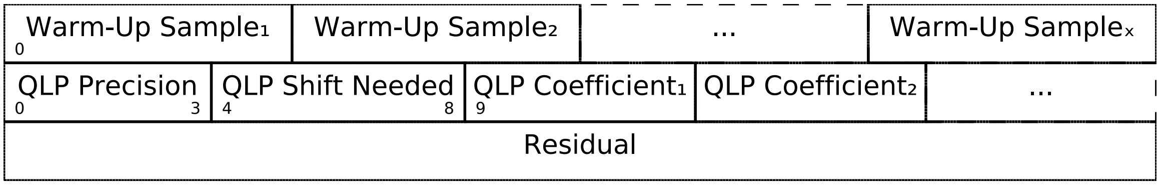
In this example, warm-up sample is -40 and the residual values are: -1 1 1 1 0 3 0 -4 -1 0 1 1 1 4 -3 1 4 -1 -1

index	sample	Δ^0	Δ^1	Δ^2	Δ^3	Δ^4
0	-40					
1	-41	*-41*				
2	-40	*-40*	*-1*			
3	-39	*-39*	*-1*	*0*		
4	-38	-38	*-1*	*0*	*0*	
5	-38	-38	0	-1	1	-1
6	-35	-35	-3	3	-4	5
7	-35	-35	0	-3	6	-10
8	-39	-39	4	-4	1	5
9	-40	-40	1	3	-7	8
10	-40	-40	0	1	2	-9
11	-39	-39	-1	1	0	2
12	-38	-38	-1	0	1	-1
13	-37	-37	-1	0	0	1
14	-33	-33	-4	3	-3	3
15	-36	-36	3	-7	10	-13
16	-35	-35	-1	4	-11	21
17	-31	-31	-4	3	1	-12
18	-32	-32	1	-5	8	-7
19	-33	-33	1	0	-5	13
$\|sum\|$		579	26	38	60	111

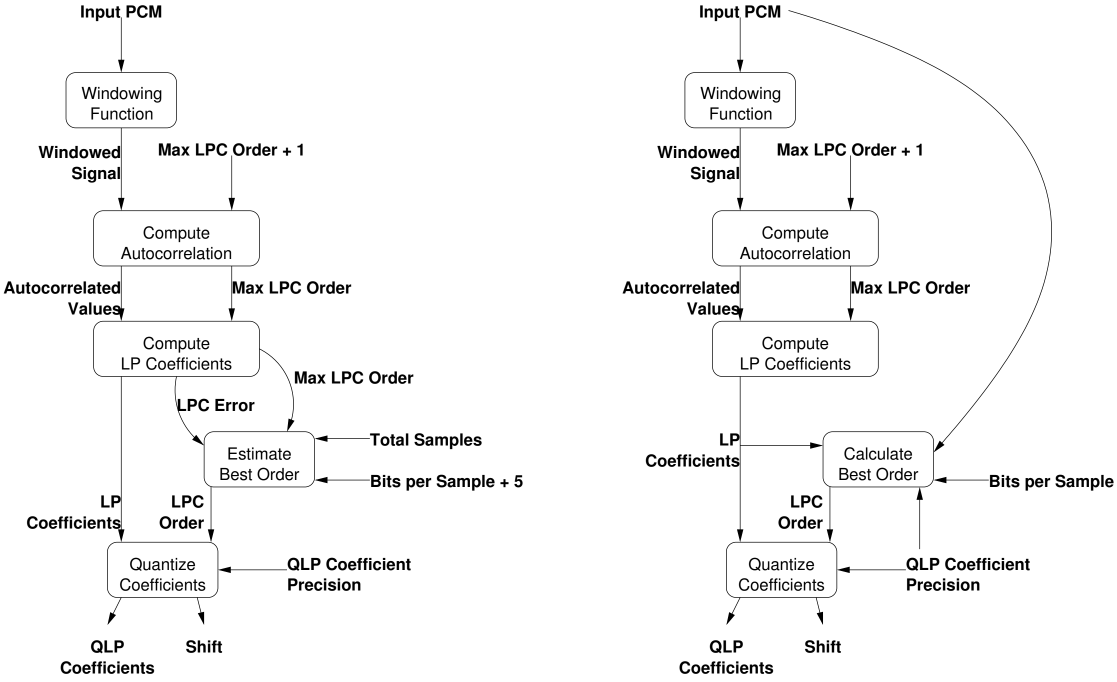
Order	Calculation
0	$\text{Residual}_i = \text{Sample}_i$
1	$\text{Residual}_i = \text{Sample}_i - \text{Sample}_{i-1}$
2	$\text{Residual}_i = \text{Sample}_i - ((2 \times \text{Sample}_{i-1}) - \text{Sample}_{i-2})$
3	$\text{Residual}_i = \text{Sample}_i - ((3 \times \text{Sample}_{i-1}) - (3 \times \text{Sample}_{i-2}) + \text{Sample}_{i-3})$
4	$\text{Residual}_i = \text{Sample}_i - ((4 \times \text{Sample}_{i-1}) - (6 \times \text{Sample}_{i-2}) + (4 \times \text{Sample}_{i-3}) - \text{Sample}_{i-4})$

6.4.8 the LPC subframe

Unlike the FIXED subframe which required only input samples and a predictor order, LPC subframes also require a list of QLP coefficients, a QLP precision value of those coefficients, and a QLP shift needed value.



Determining these values for a given input PCM signal is a somewhat complicated process which depends on whether one is performing an exhaustive LP coefficient order search or not:



(a) non-exhaustive search

(b) exhaustive search

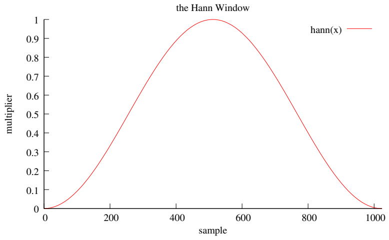
Windowing

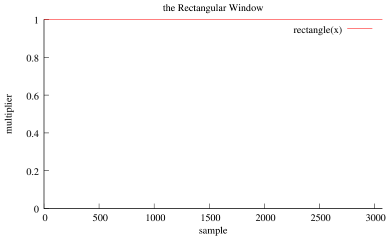
The first step in LPC subframe encoding is 'windowing' the input signal. Put simply, this is a process of multiplying each input sample by an equivalent value from the window, which are floats from 0.0 to 1.0. In this case, the default is a Tukey window with a ratio of 0.5. A Tukey window is a combination of the Hann and Rectangular windows. The ratio of 0.5 means there's 0.5 samples in the Hann window per sample in the Rectangular window.

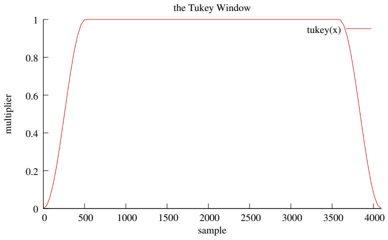
$$\text{hann}(n) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{\text{sample count} - 1}\right)\right) \tag{6.5}$$

$$\text{rectangle}(n) = 1.0 \tag{6.6}$$

The Tukey window is defined by taking a Hann window, splitting it at the halfway point, and inserting a Rectangular window between the two.





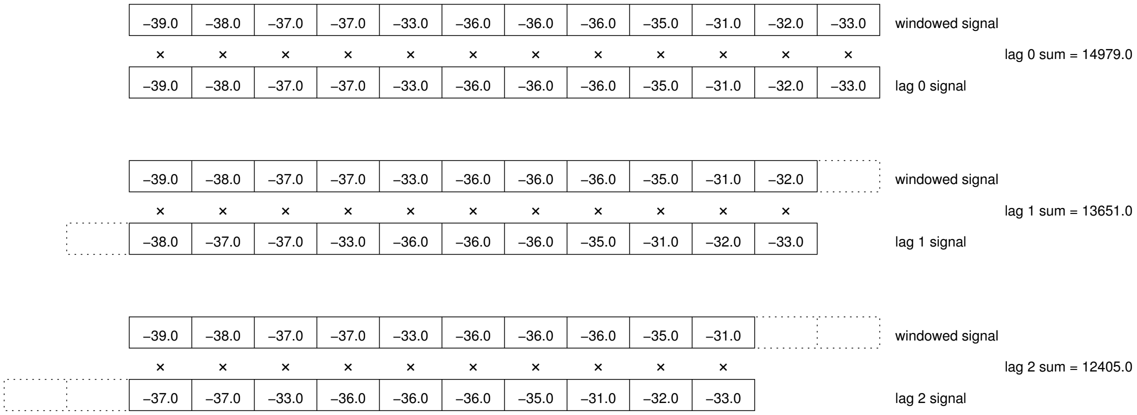


Let's run through a short example with 20 samples:

	input		Tukey		windowed
index	sample		window		signal
0	-40	×	0.0000	=	0.00
1	-41	×	0.1464	=	-6.00
2	-40	×	0.5000	=	-20.00
3	-39	×	0.8536	=	-33.29
4	-38	×	1.0000	=	-38.00
5	-38	×	1.0000	=	-38.00
6	-35	×	1.0000	=	-35.00
7	-35	×	1.0000	=	-35.00
8	-39	×	1.0000	=	-39.00
9	-40	×	1.0000	=	-40.00
10	-40	×	1.0000	=	-40.00
11	-39	×	1.0000	=	-39.00
12	-38	×	1.0000	=	-38.00
13	-37	×	1.0000	=	-37.00
14	-33	×	1.0000	=	-33.00
15	-36	×	1.0000	=	-36.00
16	-35	×	0.8536	=	-29.88
17	-31	×	0.5000	=	-15.50
18	-32	×	0.1464	=	-4.68
19	-33	×	0.0000	=	0.00

Computing autocorrelation

Once our input samples have been converted to a windowed signal, we then compute the autocorrelation values from that signal. Each autocorrelation value is determined by multiplying the signal's samples by the samples of a lagged version of that same signal, and then taking the sum. The lagged signal is simply the original signal with 'lag' number of samples removed from the beginning.



–39.0 –38.0 –37.0 –37.0 –33.0 –36.0 –36.0 –36.0 –35.0 –31.0 –32.0 –33.0 windowed signal
× × × × × × × × × × × × lag 0 sum = 14979.0
–39.0 –38.0 –37.0 –37.0 –33.0 –36.0 –36.0 –36.0 –35.0 –31.0 –32.0 –33.0 lag 0 signal

–39.0 –38.0 –37.0 –37.0 –33.0 –36.0 –36.0 –36.0 –35.0 –31.0 –32.0 windowed signal
× × × × × × × × × × × lag 1 sum = 13651.0
–38.0 –37.0 –37.0 –33.0 –36.0 –36.0 –36.0 –35.0 –31.0 –32.0 –33.0 lag 1 signal

–39.0 –38.0 –37.0 –37.0 –33.0 –36.0 –36.0 –36.0 –35.0 –31.0 windowed signal
× × × × × × × × × × lag 2 sum = 12405.0
–37.0 –37.0 –33.0 –36.0 –36.0 –36.0 –35.0 –31.0 –32.0 –33.0 lag 2 signal

The lagged sums from 0 to the maximum LPC order are our autocorrelation values. In this example, they are 14979.0, 13651.0 and 12405.0.

LP coefficient calculation

Calculating the LP coefficients uses the Levinson-Durbin recursive method.[2] Our inputs are M, the maximum LPC order minus 1, and r autocorrelation values, from $r(0)$ to $r(M-1)$. Our outputs are a, a list of LP coefficient lists from a_{11} to $a_{(M-1)(M-1)}$, and E, a list of error values from E_0 to $E_{(M-1)}$. q_m and κ_m are temporary values.

Initial values:

$$E_0 = r(0) \tag{6.7}$$

$$a_{11} = \kappa_1 = \frac{r(1)}{E_0} \tag{6.8}$$

$$E_1 = E_0(1 - {\kappa_1}^2) \tag{6.9}$$

[2] This algorithm is taken from `http://www.engineer.tamuk.edu/SPark/chap7.pdf`

With $m \geq 2$, the following recurive algorithm is performed:

Step 1. $$q_m = r(m) - \sum_{i=1}^{m-1} a_{i(m-1)} r(m-i) \tag{6.10}$$

Step 2. $$\kappa_m = \frac{q_m}{E_{(m-1)}} \tag{6.11}$$

Step 3. $$a_{mm} = \kappa_m \tag{6.12}$$

Step 4. $$a_{im} = a_{i(m-1)} - \kappa_m a_{(m-i)(m-1)} \text{ for } i = 1,\ i = 2,...,i = m-1 \tag{6.13}$$

Step 5. $$E_m = E_{m-1}(1 - {\kappa_m}^2) \tag{6.14}$$

Step 6. If $m < M$ then $m \leftarrow m + 1$ and goto step 1. If $m = M$ then stop. (6.15)

Let's run through an example in which $M = 4$, $r(0) = 11018$, $r(1) = 9690$, $r(2) = 8443$ and $r(3) = 7280$:

$$E_0 = r(0) = 11018 \tag{6.16}$$

$$a_{11} = \kappa_1 = \frac{r(1)}{E_0} = \frac{9690}{11018} = 0.8795 \tag{6.17}$$

$$E_1 = E_o(1 - {\kappa_1}^2) = 11018(1 - 0.8795^2) = 2495 \tag{6.18}$$

$$q_2 = r(2) - \sum_{i=1}^{1} a_{i1} r(2-i) = 8443 - (0.8795)(9690) = -79.35 \tag{6.19}$$

$$\kappa_2 = \frac{q_2}{E_1} = \frac{-79.35}{2495} = -0.0318 \tag{6.20}$$

$$a_{22} = \kappa_2 = -0.0318 \tag{6.21}$$

$$a_{12} = a_{11} - \kappa_2 a_{11} = 0.8795 - (-0.0318)(0.8795) = 0.9074 \tag{6.22}$$

$$E_2 = E_1(1 - {\kappa_2}^2) = 2495(1 - -0.0318^2) = 2492 \tag{6.23}$$

$$q_3 = r(3) - \sum_{i=1}^{2} a_{i2} r(3-i) = 7280 - ((0.9074)(8443) + (-0.0318)(9690)) = -73.04 \tag{6.24}$$

$$\kappa_3 = \frac{q_3}{E_2} = \frac{-73.04}{2492} = -0.0293 \tag{6.25}$$

$$a_{33} = \kappa_3 = -0.0293 \tag{6.26}$$

$$a_{13} = a_{12} - \kappa_3 a_{22} = 0.9074 - (-0.0293)(-0.0318) = 0.9065 \tag{6.27}$$

$$a_{23} = a_{22} - \kappa_3 a_{12} = -0.0318 - (-0.0293)(0.9074) = -0.0052 \tag{6.28}$$

$$E_3 = E_2(1 - {\kappa_3}^2) = 2492(1 - -0.0293^2) = 2490 \tag{6.29}$$

Our final values are:

$a_{11} = 0.8795$			(6.30)
$a_{12} = 0.9074$	$a_{22} = -0.0318$		(6.31)
$a_{13} = 0.9065$	$a_{23} = -0.0052$	$a_{33} = -0.0293$	(6.32)
$E_1 = 2495$	$E_2 = 2492$	$E_3 = 2490$	(6.33)

These values have been rounded to the nearest significant digit and will not be an exact match to those generated by a computer.

Best order estimation

At this point, we have an array of prospective LP coefficient lists, a list of error values and must decide which LPC order to use. There are two ways to accomplish this: we can either estimate the total bits from the error values or perform an exhaustive search. Making the estimation requires the total number of samples in the subframe, the number of overhead bits per order (by default, this is the number of bits per sample in the subframe, plus 5), and an error scale constant in addition to the LPC error values:

$$\text{Error Scale} = \frac{\ln(2)^2}{2 \times \text{Total Samples}} \tag{6.34}$$

Once the error scale has been calculated, one can generate a 'Bits per Residual' estimation function which, given an LPC Error value, returns what its name implies:

$$\text{Bits per Residual(LPC Error)} = \frac{\ln(\text{Error Scale} \times \text{LPC Error})}{2 \times \ln(2)} \tag{6.35}$$

With this function, we can estimate how many bits the entire LPC subframe will take for each LPC Error value and its associated Order:

$$\text{Total Bits(LPC Error, Order)} = \text{Bits per Residual(LPC Error)} \times (\text{Total Samples} - \text{Order}) + (\text{Order} \times \text{Overhead bits}) \tag{6.36}$$

Continuing with our example, we have 20 samples and now have the error values of 2495, 2492 and 2490. This gives us an error scale of: $\frac{\ln(2)^2}{2\times 20} = \frac{.6931^2}{40} = .01201$

At LPC order 1, our bits per residual are:

$$\frac{\ln(.01201 \times 2495)}{2 \times ln(2)} = \frac{\ln(29.96)}{1.386} = 2.453 \tag{6.37}$$

And our total bits are:

$$(2.453 \times (20 - 1)) + (1 \times (16 + 5)) = 46.61 + 21 = 67.61 \tag{6.38}$$

At LPC order 2, our bits per residual are:

$$\frac{\ln(.01201 \times 2492)}{2 \times \ln(2)} = \frac{\ln(29.92)}{1.386} = 2.452 \tag{6.39}$$

And our total bits are:

$$(2.452 \times (20 - 2)) + (2 \times (16 + 5)) = 44.14 + 42 = 86.14 \tag{6.40}$$

At LPC order 3, our bits per residual are:

$$\frac{\ln(.01201 \times 2490)}{2 \times \ln(2)} = \frac{\ln(29.90)}{1.386} = 2.451 \tag{6.41}$$

And our total bits are:

$$(2.451 \times (20 - 3)) + (3 \times (16 + 5)) = 41.67 + 63 = 104.7 \tag{6.42}$$

Therefore, since the total bits for order 1 are the smallest, the best order for this group of samples is 1.

Though as you'll notice, the bits per residual for order 3 were the smallest. So if this group of samples was very large, it's likely that order 3 would prevail since the residuals multiplied by a smaller bits per residual would counteract the relatively fixed overhead bits per order value.

Best order exhaustive search

In a curious bit of recursion, finding the best order for an LPC subframe via an exhaustive search requires taking each list of LP Coefficients calculated previously, quantizing them into a list of QLP Coefficients and a QLP Shift Needed value,[3] determining the total amount of bits each hypothetical LPC subframe uses and using the LPC order which uses the fewest.

Remember that building an LPC subframe requires the following values: LPC Order, QLP Precision, QLP Shift Needed and QLP Coefficients along with the subframe's samples and bits-per-sample. For each possible LPC Order, the QLP Shift Needed and the QLP Coefficient list values can be calculated by quantizing the LP Coefficients. QLP Precision is the size of each QLP Coefficient list value in the subframe header. Simply choose the field with the largest number of bits in the QLP Coefficient list for the QLP Precision value.

Finally, instead of writing these hypothetical LPC subframes directly to disk, one only has to capture how many bits they *would* use. The hypothetical LPC subframe that uses the fewest number of bits is the one we should actually write to disk.

[3] Quantizing coefficients will be covered in the next section.

Quantizing coefficients

Quantizing coefficients is a process of taking a list of LP Coefficients along with a QLP Coefficients Precision value and returning a list of QLP Coefficients and a QLP Shift Needed value. The first step is determining the upper and lower limits of the QLP Coefficients:

$$\text{QLP coefficient maximum} = 2^{precision-1} - 1 \tag{6.43}$$

$$\text{QLP coefficient mininum} = -2^{precision-1} \tag{6.44}$$

The QLP Coefficients Precision value is typically based on the encoder's block size:

Block Size	Precision	Block Size	Precision
Size $\leq$ 192	7	Size $\leq$ 384	8
Size $\leq$ 576	9	Size $\leq$ 1152	10
Size $\leq$ 2304	11	Size $\leq$ 4608	12
Size $>$ 4608	13		

So in our example of a block of 20 samples,

$$\text{QLP Coefficient maximum} = 2^{7-1} - 1 = 64 - 1 = 63 \tag{6.45}$$

$$\text{QLP Coefficient minimum} = -2^{7-1} = -64 \tag{6.46}$$

Now we determine the initial QLP Shift Needed value:

$$\text{shift} = \text{precision} - \left\lceil \frac{\log(\max(|\text{LP Coefficients}|))}{\log(2)} \right\rceil - 1 \tag{6.47}$$

where 'shift' is adjusted if necessary such that: $0 \leq \text{shift} \leq$ `0xF` , since it must fit into a 5-bit signed field and negative shifts are no-ops in the FLAC decoder.

Continuing our ongoing example, let's assume we're quantizing the LP coefficients 0.9065, -0.0052 and -0.0293. So our shift should be:

$$\text{shift} = 7 - \left\lceil \frac{\log(0.9065)}{\log(2)} \right\rceil - 1 = 7 - \left\lceil \frac{-0.0981}{0.6931} \right\rceil - 1 = 7 - 0 - 1 = 6 \tag{6.48}$$

Finally, we determine the QLP Coefficient values themselves via a small recursive routine:

$$X(i) = E(i-1) + (\text{LP Coefficient}_i \times 2^{shift}) \tag{6.49}$$

$$\text{QLP Coefficient}_i = \text{round}(X(i)) \tag{6.50}$$

$$E(i) = X(i) - \text{QLP Coefficient}_i \tag{6.51}$$

where $E(0) = 0$ and each QLP Coefficient is adjusted prior to calculating the next $E(i)$ value such that: QLP coefficient minimum $\leq$ QLP Coefficient$_i$ $\leq$ QLP coefficient maximum

So to finish our LPC example:

$$\begin{aligned}
X(1) &= E(0) + (0.9065 \times 2^6) = 0 + 58.016 = \mathbf{58.016} && (6.52)\\
\text{QLP Coefficient}_1 &= \text{round}(58.016) = \mathbf{58} && (6.53)\\
E(1) &= X(1) - \text{QLP Coefficient}_1 = 58.016 - 58 = \mathbf{0.016} && (6.54)\\
X(2) &= E(1) + (-0.0052 \times 2^6) = 0.016 + -0.3328 = \mathbf{0.3168} && (6.55)\\
\text{QLP Coefficient}_2 &= \text{round}(0.3168) = \mathbf{0} && (6.56)\\
E(2) &= X(2) - \text{QLP Coefficient}_2 = 0.3168 - 0 = \mathbf{0.3168} && (6.57)\\
X(3) &= E(2) + (-0.0293 \times 2^6) = 0.3168 + -1.875 = \mathbf{-1.558} && (6.58)\\
\text{QLP Coefficient}_3 &= \text{round}(-1.558) = \mathbf{-2} && (6.59)\\
E(3) &= X(3) - \text{QLP Coefficient}_3 = -1.558 - -2 = \mathbf{0.4420} && (6.60)
\end{aligned}$$

Therefore, the LPC order is 3. The QLP Coefficients are 58, 0 and -2. The QLP Shift Needed value is 6. And, the QLP precision value can be calculated from the bits required for the largest absolute QLP Coefficient value. In this case, 6 bits are required to hold the value 58 so QLP precision can be 6.

Calculating LPC residual

A number of warm-up samples equal to LPC Order are taken from the input PCM and the subframe's residuals are calculated according to the following formula:

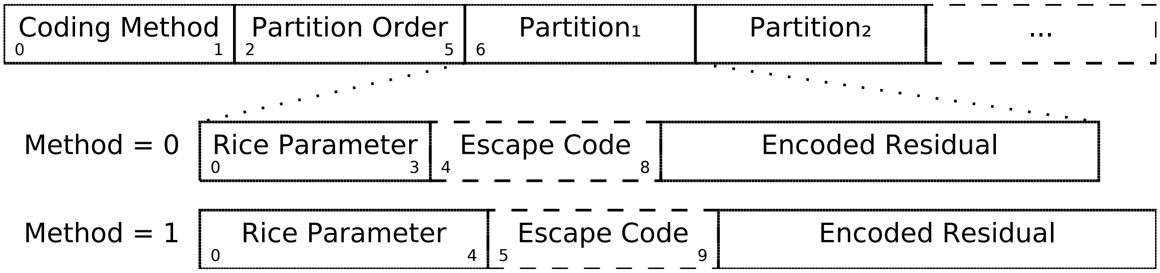
$$\text{Residual}_i = \text{Sample}_i - \left\lfloor \frac{\sum_{j=0}^{Order-1} \text{QLP Coeffcient}_j \times \text{Sample}_{i-j-1}}{2^{\text{QLP Shift Needed}}} \right\rfloor \qquad (6.61)$$

For example, given the samples 1053, 1116, 1257, 1423, 1529, 1570, 1623, 1694, 1688, 1550, the coefficients: 1241, -944, 14, 342, -147 and a QLP Shift Needed value of 9, our residuals are as follows:

Index	Sample	Residual
0	(warm-up) 1053	
1	(warm-up) 1116	
2	(warm-up) 1257	
3	(warm-up) 1423	
4	(warm-up) 1529	
5	1570	$1570 - \left\lfloor \frac{(1241\times1529)+(-944\times1423)+(14\times1257)+(342\times1116)+(-147\times1053)}{2^9} \right\rfloor = 1570 - \left\lfloor \frac{798656}{512} \right\rfloor = \mathbf{11}$
6	1623	$1623 - \left\lfloor \frac{(1241\times1570)+(-944\times1529)+(14\times1423)+(342\times1257)+(-147\times1116)}{2^9} \right\rfloor = 1623 - \left\lfloor \frac{790758}{512} \right\rfloor = \mathbf{79}$
7	1694	$1694 - \left\lfloor \frac{(1241\times1623)+(-944\times1570)+(14\times1529)+(342\times1423)+(-147\times1257)}{2^9} \right\rfloor = 1694 - \left\lfloor \frac{855356}{512} \right\rfloor = \mathbf{24}$
8	1688	$1688 - \left\lfloor \frac{(1241\times1694)+(-944\times1623)+(14\times1570)+(342\times1529)+(-147\times1423)}{2^9} \right\rfloor = 1688 - \left\lfloor \frac{905859}{512} \right\rfloor = \mathbf{-81}$
9	1550	$1550 - \left\lfloor \frac{(1241\times1688)+(-944\times1694)+(14\times1623)+(342\times1570)+(-147\times1529)}{2^9} \right\rfloor = 1550 - \left\lfloor \frac{830571}{512} \right\rfloor = \mathbf{-72}$

6.4.9 the Residual

Given a stream of residual values, one must place them in one or more partitions, each with its own Rice parameter, and prepended with a small header:



The residual's coding method is typically 0, unless one is encoding audio with more than 16 bits-per-sample and one of the partitions requests a Rice parameter higher than 2^4. The residual's partition order is chosen exhaustively, which means trying all of them within a certain range (e.g. 0 to 5) such that the residuals can be divided evenly between them and then the partition order which uses the smallest estimated amount of space is chosen.

Choosing the best Rice parameter is a matter of selecting the smallest value of 'x' such that:

$$\text{sample count} \times 2^x > \sum_{i=0}^{\text{residual count}-1} |\text{residual}_i| \tag{6.62}$$

Again, this is easier to understand with a block of example residuals, 19 in total:

index	residual_i	$\lvert\text{residual}_i\rvert$
0	-1	1
1	1	1
2	1	1
3	1	1
4	0	0
5	3	3
6	0	0
7	-4	4
8	-1	1
9	0	0
10	1	1
11	1	1
12	1	1
13	4	4
14	-3	3
15	1	1
16	4	4
17	-1	1
18	-1	1
|sum|		**29**

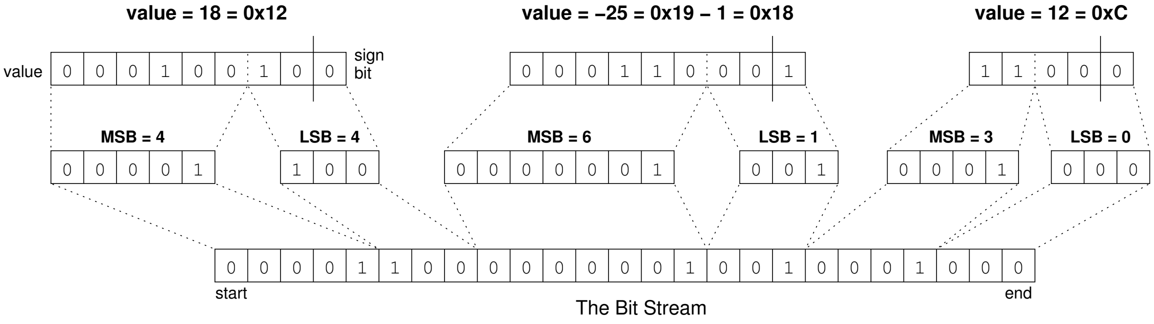
19×2^0 is not larger than 29. 19×2^1 is larger than 29, so the best Rice parameter for this block of residuals is 1. Remember that the Rice parameter's maximum value is limited to

2^4 using coding method 0, or 2^5 using coding method 1.

Residual values

Encoding individual residual values to Rice coding requires only the Rice parameter and the values themselves. First, one must convert any negative values to positive by multiplying it by -1, subtracting 1 and prepending a 1 bit. If the value is already positive, prepend a 0 bit instead. Next, we split out new value into most significant bits (MSB) and least significant bits (LSB) where the length of the LSB is equal to the Rice parameter and MSB contains the remaining bits. The MSB value is written unary encoded, whereas the LSB is written as-is.

As with residual decoding, this process is not as difficult as it sounds and is best explained by an example, in this case using a parameter of 3 and encoding the residual values 18, -25 and 12:



6.4.10 Checksums

Calculating the frame header's CRC-8 and frame footer's CRC-16 is necessary both for FLAC encoders and decoders, but the process is the same for each.

CRC-8

Given a byte of input and the previous CRC-8 checksum, or 0 as an initial value, the current checksum can be calculated as follows:

$$\text{checksum}_i = \text{CRC8}(\textit{byte}\ \textbf{xor}\ \text{checksum}_{i-1}) \tag{6.63}$$

	0x?0	0x?1	0x?2	0x?3	0x?4	0x?5	0x?6	0x?7	0x?8	0x?9	0x?A	0x?B	0x?C	0x?D	0x?E	0x?F
0x0?	0x00	0x07	0x0E	0x09	0x1C	0x1B	0x12	0x15	0x38	0x3F	0x36	0x31	0x24	0x23	0x2A	0x2D
0x1?	0x70	0x77	0x7E	0x79	0x6C	0x6B	0x62	0x65	0x48	0x4F	0x46	0x41	0x54	0x53	0x5A	0x5D
0x2?	0xE0	0xE7	0xEE	0xE9	0xFC	0xFB	0xF2	0xF5	0xD8	0xDF	0xD6	0xD1	0xC4	0xC3	0xCA	0xCD
0x3?	0x90	0x97	0x9E	0x99	0x8C	0x8B	0x82	0x85	0xA8	0xAF	0xA6	0xA1	0xB4	0xB3	0xBA	0xBD
0x4?	0xC7	0xC0	0xC9	0xCE	0xDB	0xDC	0xD5	0xD2	0xFF	0xF8	0xF1	0xF6	0xE3	0xE4	0xED	0xEA
0x5?	0xB7	0xB0	0xB9	0xBE	0xAB	0xAC	0xA5	0xA2	0x8F	0x88	0x81	0x86	0x93	0x94	0x9D	0x9A
0x6?	0x27	0x20	0x29	0x2E	0x3B	0x3C	0x35	0x32	0x1F	0x18	0x11	0x16	0x03	0x04	0x0D	0x0A
0x7?	0x57	0x50	0x59	0x5E	0x4B	0x4C	0x45	0x42	0x6F	0x68	0x61	0x66	0x73	0x74	0x7D	0x7A
0x8?	0x89	0x8E	0x87	0x80	0x95	0x92	0x9B	0x9C	0xB1	0xB6	0xBF	0xB8	0xAD	0xAA	0xA3	0xA4
0x9?	0xF9	0xFE	0xF7	0xF0	0xE5	0xE2	0xEB	0xEC	0xC1	0xC6	0xCF	0xC8	0xDD	0xDA	0xD3	0xD4
0xA?	0x69	0x6E	0x67	0x60	0x75	0x72	0x7B	0x7C	0x51	0x56	0x5F	0x58	0x4D	0x4A	0x43	0x44
0xB?	0x19	0x1E	0x17	0x10	0x05	0x02	0x0B	0x0C	0x21	0x26	0x2F	0x28	0x3D	0x3A	0x33	0x34
0xC?	0x4E	0x49	0x40	0x47	0x52	0x55	0x5C	0x5B	0x76	0x71	0x78	0x7F	0x6A	0x6D	0x64	0x63
0xD?	0x3E	0x39	0x30	0x37	0x22	0x25	0x2C	0x2B	0x06	0x01	0x08	0x0F	0x1A	0x1D	0x14	0x13
0xE?	0xAE	0xA9	0xA0	0xA7	0xB2	0xB5	0xBC	0xBB	0x96	0x91	0x98	0x9F	0x8A	0x8D	0x84	0x83
0xF?	0xDE	0xD9	0xD0	0xD7	0xC2	0xC5	0xCC	0xCB	0xE6	0xE1	0xE8	0xEF	0xFA	0xFD	0xF4	0xF3

For example, given the header bytes: 0xFF, 0xF8, 0xCC, 0x1C, 0x00 and 0xC0:

$$\text{checksum}_0 = \text{CRC8}(\texttt{0xFF}\ \textbf{xor}\ \texttt{0x00}) = \text{CRC8}(\texttt{0xFF}) = \texttt{0xF3} \tag{6.64}$$
$$\text{checksum}_1 = \text{CRC8}(\texttt{0xF8}\ \textbf{xor}\ \texttt{0xF3}) = \text{CRC8}(\texttt{0x0B}) = \texttt{0x31} \tag{6.65}$$
$$\text{checksum}_2 = \text{CRC8}(\texttt{0xCC}\ \textbf{xor}\ \texttt{0x31}) = \text{CRC8}(\texttt{0xFD}) = \texttt{0xFD} \tag{6.66}$$
$$\text{checksum}_3 = \text{CRC8}(\texttt{0x1C}\ \textbf{xor}\ \texttt{0xFD}) = \text{CRC8}(\texttt{0xE1}) = \texttt{0xA9} \tag{6.67}$$
$$\text{checksum}_4 = \text{CRC8}(\texttt{0x00}\ \textbf{xor}\ \texttt{0xA9}) = \text{CRC8}(\texttt{0xA9}) = \texttt{0x56} \tag{6.68}$$
$$\text{checksum}_5 = \text{CRC8}(\texttt{0xC0}\ \textbf{xor}\ \texttt{0x56}) = \text{CRC8}(\texttt{0x96}) = \texttt{0xEB} \tag{6.69}$$

Thus, the next byte after the header should be `0xEB`. Furthermore, when the checksum byte itself is run through the checksumming procedure:

$$\text{checksum}_6 = \text{CRC8}(\texttt{0xEB}\ \textbf{xor}\ \texttt{0xEB}) = \text{CRC8}(\texttt{0x00}) = \texttt{0x00} \tag{6.70}$$

the result will always be 0. This is a handy way to verify a frame header's checksum since the checksum of the header's bytes along with the header's checksum itself will always result in 0.

CRC-16

CRC-16 is used to checksum the entire FLAC frame, including the header and any padding bits after the final subframe. Given a byte of input and the previous CRC-16 checksum, or 0 as an initial value, the current checksum can be calculated as follows:

$$\text{checksum}_i = \text{CRC16}(\mathit{byte}\ \mathbf{xor}\ (\text{checksum}_{i-1} \gg 8))\ \mathbf{xor}\ (\text{checksum}_{i-1} \ll 8) \tag{6.71}$$

and the checksum is always truncated to 16-bits.

	0x?0	0x?1	0x?2	0x?3	0x?4	0x?5	0x?6	0x?7	0x?8	0x?9	0x?A	0x?B	0x?C	0x?D	0x?E	0x?F
0x0?	0000	8005	800f	000a	801b	001e	0014	8011	8033	0036	003c	8039	0028	802d	8027	0022
0x1?	8063	0066	006c	8069	0078	807d	8077	0072	0050	8055	805f	005a	804b	004e	0044	8041
0x2?	80c3	00c6	00cc	80c9	00d8	80dd	80d7	00d2	00f0	80f5	80ff	00fa	80eb	00ee	00e4	80e1
0x3?	00a0	80a5	80af	00aa	80bb	00be	00b4	80b1	8093	0096	009c	8099	0088	808d	8087	0082
0x4?	8183	0186	018c	8189	0198	819d	8197	0192	01b0	81b5	81bf	01ba	81ab	01ae	01a4	81a1
0x5?	01e0	81e5	81ef	01ea	81fb	01fe	01f4	81f1	81d3	01d6	01dc	81d9	01c8	81cd	81c7	01c2
0x6?	0140	8145	814f	014a	815b	015e	0154	8151	8173	0176	017c	8179	0168	816d	8167	0162
0x7?	8123	0126	012c	8129	0138	813d	8137	0132	0110	8115	811f	011a	810b	010e	0104	8101
0x8?	8303	0306	030c	8309	0318	831d	8317	0312	0330	8335	833f	033a	832b	032e	0324	8321
0x9?	0360	8365	836f	036a	837b	037e	0374	8371	8353	0356	035c	8359	0348	834d	8347	0342
0xA?	03c0	83c5	83cf	03ca	83db	03de	03d4	83d1	83f3	03f6	03fc	83f9	03e8	83ed	83e7	03e2
0xB?	83a3	03a6	03ac	83a9	03b8	83bd	83b7	03b2	0390	8395	839f	039a	838b	038e	0384	8381
0xC?	0280	8285	828f	028a	829b	029e	0294	8291	82b3	02b6	02bc	82b9	02a8	82ad	82a7	02a2
0xD?	82e3	02e6	02ec	82e9	02f8	82fd	82f7	02f2	02d0	82d5	82df	02da	82cb	02ce	02c4	82c1
0xE?	8243	0246	024c	8249	0258	825d	8257	0252	0270	8275	827f	027a	826b	026e	0264	8261
0xF?	0220	8225	822f	022a	823b	023e	0234	8231	8213	0216	021c	8219	0208	820d	8207	0202

For example, given the frame bytes: 0xFF, 0xF8, 0xCC, 0x1C, 0x00, 0xC0, 0xEB, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 and 0x00, the frame's CRC-16 can be calculated as follows:

$$\text{checksum}_0 = \text{CRC16}(\texttt{0xFF}\ \mathbf{xor}\ (\texttt{0x0000} \gg 8))\ \mathbf{xor}\ (\texttt{0x0000} \ll 8) = \text{CRC16}(\texttt{0xFF})\ \mathbf{xor}\ \texttt{0x0000} = \texttt{0x0202} \tag{6.72}$$

$$\text{checksum}_1 = \text{CRC16}(\texttt{0xF8}\ \mathbf{xor}\ (\texttt{0x0202} \gg 8))\ \mathbf{xor}\ (\texttt{0x0202} \ll 8) = \text{CRC16}(\texttt{0xFA})\ \mathbf{xor}\ \texttt{0x0200} = \texttt{0x001C} \tag{6.73}$$

$$\text{checksum}_2 = \text{CRC16}(\texttt{0xCC}\ \mathbf{xor}\ (\texttt{0x001C} \gg 8))\ \mathbf{xor}\ (\texttt{0x001C} \ll 8) = \text{CRC16}(\texttt{0xCC})\ \mathbf{xor}\ \texttt{0x1C00} = \texttt{0x1EA8} \tag{6.74}$$

$$\text{checksum}_3 = \text{CRC16}(\texttt{0x1C}\ \mathbf{xor}\ (\texttt{0x1EA8} \gg 8))\ \mathbf{xor}\ (\texttt{0x1EA8} \ll 8) = \text{CRC16}(\texttt{0x02})\ \mathbf{xor}\ \texttt{0xA800} = \texttt{0x280F} \tag{6.75}$$

$$\text{checksum}_4 = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x280F} \gg 8))\ \mathbf{xor}\ (\texttt{0x280F} \ll 8) = \text{CRC16}(\texttt{0x28})\ \mathbf{xor}\ \texttt{0x0F00} = \texttt{0x0FF0} \tag{6.76}$$

$$\text{checksum}_5 = \text{CRC16}(\texttt{0xC0}\ \mathbf{xor}\ (\texttt{0x0FF0} \gg 8))\ \mathbf{xor}\ (\texttt{0x0FF0} \ll 8) = \text{CRC16}(\texttt{0xCF})\ \mathbf{xor}\ \texttt{0xF000} = \texttt{0xF2A2} \tag{6.77}$$

$$\text{checksum}_6 = \text{CRC16}(\texttt{0xEB}\ \mathbf{xor}\ (\texttt{0xF2A2} \gg 8))\ \mathbf{xor}\ (\texttt{0xF2A2} \ll 8) = \text{CRC16}(\texttt{0x19})\ \mathbf{xor}\ \texttt{0xA200} = \texttt{0x2255} \tag{6.78}$$

$$\text{checksum}_7 = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x2255} \gg 8))\ \mathbf{xor}\ (\texttt{0x2255} \ll 8) = \text{CRC16}(\texttt{0x22})\ \mathbf{xor}\ \texttt{0x5500} = \texttt{0x55CC} \tag{6.79}$$

$$\text{checksum}_8 = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x55CC} \gg 8))\ \mathbf{xor}\ (\texttt{0x55CC} \ll 8) = \text{CRC16}(\texttt{0x55})\ \mathbf{xor}\ \texttt{0xCC00} = \texttt{0xCDFE} \tag{6.80}$$

$$\text{checksum}_9 = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0xCDFE} \gg 8))\ \mathbf{xor}\ (\texttt{0xCDFE} \ll 8) = \text{CRC16}(\texttt{0xCD})\ \mathbf{xor}\ \texttt{0xFE00} = \texttt{0x7CAD} \tag{6.81}$$

$$\text{checksum}_{10} = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x7CAD} \gg 8))\ \mathbf{xor}\ (\texttt{0x7CAD} \ll 8) = \text{CRC16}(\texttt{0x7C})\ \mathbf{xor}\ \texttt{0xAD00} = \texttt{0x2C0B} \tag{6.82}$$

$$\text{checksum}_{11} = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x2C0B} \gg 8))\ \mathbf{xor}\ (\texttt{0x2C0B} \ll 8) = \text{CRC16}(\texttt{0x2C})\ \mathbf{xor}\ \texttt{0x0B00} = \texttt{0x8BEB} \tag{6.83}$$

$$\text{checksum}_{12} = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x8BEB} \gg 8))\ \mathbf{xor}\ (\texttt{0x8BEB} \ll 8) = \text{CRC16}(\texttt{0x8B})\ \mathbf{xor}\ \texttt{0xEB00} = \texttt{0xE83A} \tag{6.84}$$

$$\text{checksum}_{13} = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0xE83A} \gg 8))\ \mathbf{xor}\ (\texttt{0xE83A} \ll 8) = \text{CRC16}(\texttt{0xE8})\ \mathbf{xor}\ \texttt{0x3A00} = \texttt{0x3870} \tag{6.85}$$

$$\text{checksum}_{14} = \text{CRC16}(\texttt{0x00}\ \mathbf{xor}\ (\texttt{0x3870} \gg 8))\ \mathbf{xor}\ (\texttt{0x3870} \ll 8) = \text{CRC16}(\texttt{0x38})\ \mathbf{xor}\ \texttt{0x7000} = \texttt{0xF093} \tag{6.86}$$

Thus, the next two bytes after the final subframe should be `0xF0` and `0x93`. Again, when the checksum bytes are run through the checksumming procedure:

$$\text{checksum}_{15} = \text{CRC16}(\texttt{0xF0}\ \mathbf{xor}\ (\texttt{0xF093} \gg 8))\ \mathbf{xor}\ (\texttt{0xF093} \ll 8) = \text{CRC16}(\texttt{0x00})\ \mathbf{xor}\ \texttt{0x9300} = \texttt{0x9300} \tag{6.87}$$

$$\text{checksum}_{16} = \text{CRC16}(\texttt{0x93}\ \mathbf{xor}\ (\texttt{0x9300} \gg 8))\ \mathbf{xor}\ (\texttt{0x9300} \ll 8) = \text{CRC16}(\texttt{0x00})\ \mathbf{xor}\ \texttt{0x0000} = \texttt{0x0000} \tag{6.88}$$

the result will also always be 0, just as in the CRC-8.

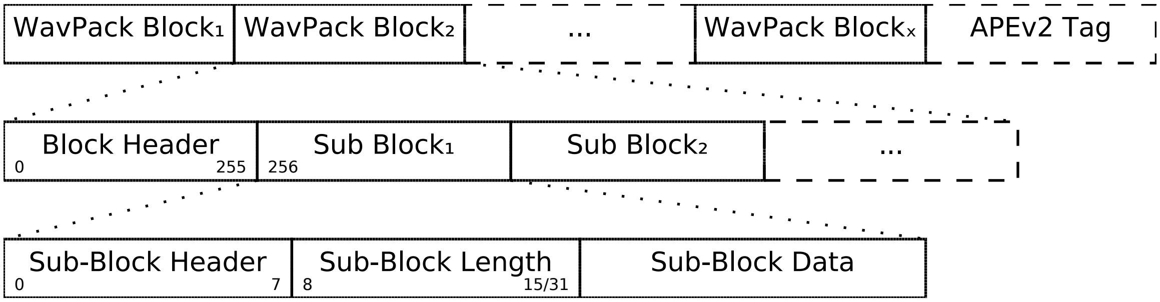
7 WavPack

WavPack is a format for compressing Wave files, typically in lossless mode. Notably, it also has a lossy mode and even a hybrid mode which allows the 'correction' file to be separated from a lossy core.

Metadata is stored as an APEv2 tag, which is described on page 52.

All of its fields are little-endian.

7.1 the WavPack file stream



7.2 the WavPack block header

Block ID `wvpk' (0x7776706B) 0 31	Block Size 32 63		
Version 64 79	Track Number 80 87	Index Number 88 95	Total Samples 96 127
Block Index 128 159	Block Samples 160 191		
Floating Point Data 192	Hybrid Noise Shaping 193	Channel Decorrelation 194	Joint Stereo 195
Hybrid Mode 196	Mono Output 197	Bits-per-Sample 198 199	
Left Shift Data (low) 200 202	Final Block 203		
Initial Block 204	Hbd. Noise Balanced 205	Hbd. Controls Bitrate 206	Extended Size Integers 207
Sampling Rate (low) 208	Maximum Magnitude 209 211		
Maximum Magnitude (cont.) 212 213	Left Shift Data (high) 214 215		
Reserved 216	False Stereo 217	Use IIR 218	Reserved 219
Reserved 220	Sampling Rate (high) 221 223		
CRC 224 255			

The 'flags' field is stored as a little-endian 32-bit integer. Since some fields cross byte boundaries, their high and low bits will be far apart when written in this format where the bits are ordered the way they appear in the file.

'Block Size' is the length of everything past everything past the block header, minus 24 bytes.

'Bits per Sample' is one of 4 values:

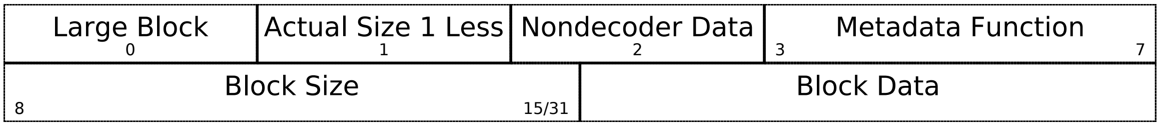
00 = 8 bps, 01 = 16 bps, 10 = 24 bps, 11 = 32 bps .

'Mono Output' bit indicates the channel count. If 1, this block has 1 channel. If 0, this block has 2 channels. For an audio stream with more than 2 channels, check the 'Initial Block' and 'Final Block' bits to indicate the start and end of the channels. As an example:

Initial Block	Final Block	Mono Output	Channels
1	0	0	2
0	0	1	1
0	0	1	1
0	1	0	2
		Total	6

value	sample rate
0000	6000
0001	8000
0010	9600
0011	11025
0100	12000
0101	16000
0110	22050
0111	24000
1000	32000
1001	44100
1010	48000
1011	64000
1100	88200
1101	96000
1110	192000
1111	reserved

7.2.1 WavPack sub-block header

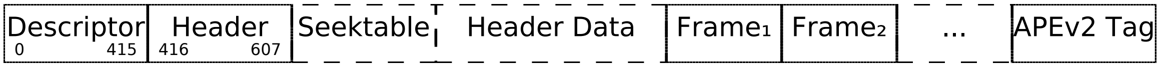


If the 'Large Block' field is 0, the 'Block Size' field is 8 bits long. If it is 1, the 'Block Size' field is 24 bits long. The 'Block Size' field is the length of 'Block Data', in 16-bit words rather than bytes. If 'Actual Size 1 Less' is set, that means 'Block Data' doesn't contain an even number of bytes; it is padded with a single null byte at the end in order to fit. If 'Nondecoder Data' is set, that means the decoder does not have to understand the contents of this particular sub-block in order to decode the audio.

8 Monkey's Audio

Monkey's Audio is a lossless RIFF WAVE compressor. Unlike FLAC, which is a PCM compressor, Monkey's Audio also stores IFF chunks and reproduces the original WAVE file in its entirety rather than storing only the data it contains. All of its fields are little-endian.

8.1 the Monkey's Audio file stream



8.2 the Monkey's Audio descriptor

ID (`MAC' 0x4D414320)	0	31
Version	32	63
Descriptor Bytes	64	95
Header Bytes	96	127
Seektable Bytes	128	159
Header Data Bytes	160	191
Frame Data Bytes	192	223
Frame Data Bytes (High)	224	255
Terminating Data Bytes	256	287
MD5 Sum	288	415

Version is the encoding software's version times 1000. i.e. Monkey's Audio 3.99 = 3990

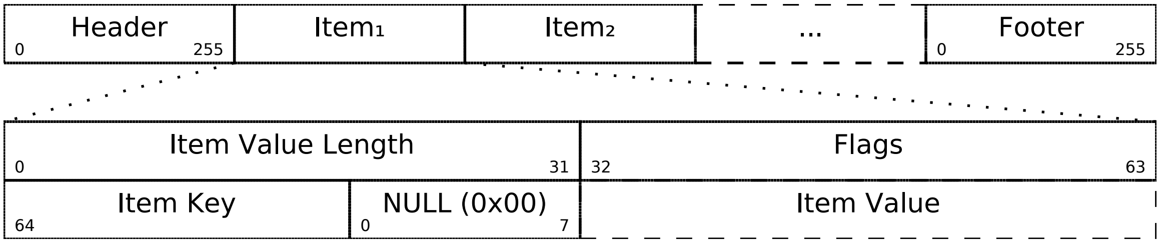
8.3 the Monkey's Audio header

Compression Level	0	15
Format Flags	16	31
Blocks Per Frame	32	63
Final Frame Blocks	64	95
Total Frames	96	127
Bits Per Sample	128	143
Channels	144	159
Sample Rate	160	191

$$\text{Length in Seconds} = \frac{((\text{Total Frames} - 1) \times \text{Blocks Per Frame}) + \text{Final Frame Blocks}}{\text{Sample Rate}} \tag{8.1}$$

8.4 the APEv2 tag

The APEv2 tag is a little-endian metadata tag appended to Monkey's Audio files, among others.



Item Key is an ASCII string from the range 0x20 to 0x7E. Item Value is typically a UTF-8 encoded string, but may also be binary depending on the Flags.

Abstract Abstract

Album album name

Artist performing artist

Bibliography Bibliography/Discography

Catalog catalog number

Comment user comment

Composer original composer

Conductor conductor

Copyright copyright holder

Debut album debut album name

Dummy place holder

EAN/UPC EAN-13/UPC-A bar code identifier

File file location

Genre genre

Index indexes for quick access

Introplay characteric part of piece for intro playing

ISBN ISBN number with check digit

ISRC International Standard Recording Number

Language used Language(s) for music/spoken words

LC Label Code

Media source media

Publicationright publication right holder

Publisher record label or publisher

Record Date record date

Record Location record location

Related location of related information

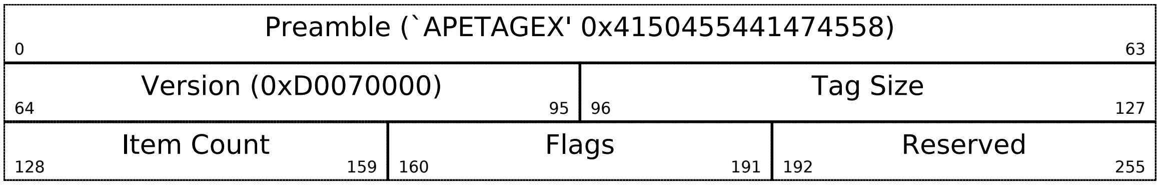
Subtitle track subtitle

Title track title

Track track number

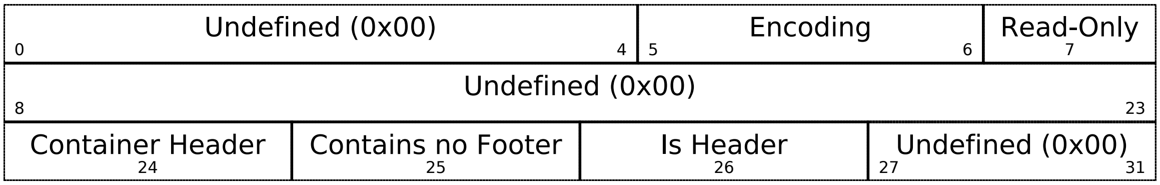
Year release date

8.4.1 the APEv2 tag header/footer



The format of the APEv2 header and footer are identical except for the 'Is Header' tag. 'Version' is typically 2000 (stored little-endian). 'Tag Size' is the size of the entire APEv2 tag, including the footer but excluding the header. 'Item Count' is the number of individual tag items.

8.4.2 the APEv2 flags



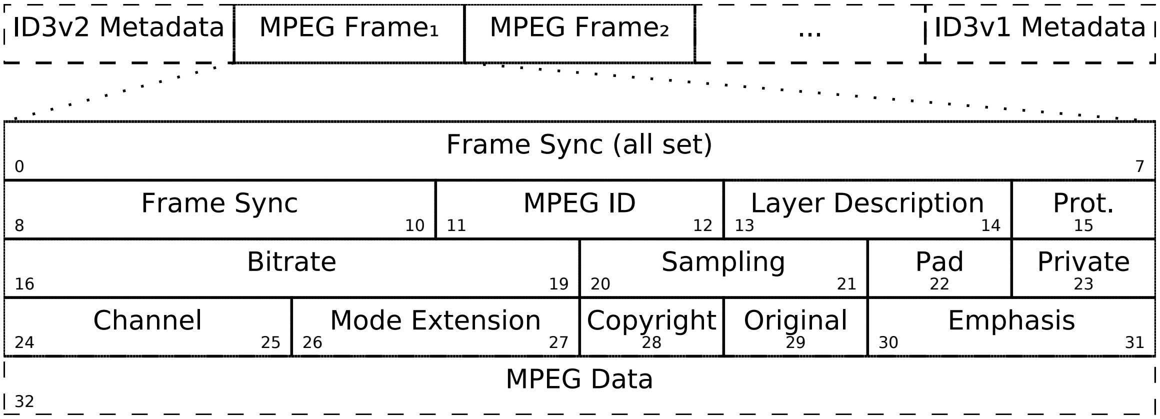
This flags field is used by both the APEv2 header/footer and the individual tag items. The 'Encoding' field indicates the encoding of its value:
`00` = UTF-8, `01` = Binary, `10` = External Link, `11` = Reserved .

9 MP3

MP3 is the de-facto standard for lossy audio. It is little more than a series of MPEG frames with an optional ID3v2 metadata header and optional ID3v1 metadata footer.

MP3 decoders are assumed to be very tolerant of anything in the stream that doesn't look like an MPEG frame, ignoring such junk until the next frame is found. Since MP3 files have no standard container format in which non-MPEG data can be placed, metadata such as ID3 tags are often made 'sync-safe' by formatting them in a way that decoders won't confuse tags for MPEG frames.

9.1 the MP3 file Stream



			Sample Rate			
bits	MPEG ID	Description	MPEG-1	MPEG-2	MPEG-2.5	Channels
00	MPEG-2.5	reserved	44100	22050	11025	Stereo
01	reserved	Layer III	48000	24000	12000	Joint stereo
10	MPEG-2	Layer II	32000	16000	8000	Dual channel stereo
11	MPEG-1	Layer I	reserved	reserved	reserved	Mono

Layer I frames always contain 384 samples. Layer II and Layer III frames always contain 1152 samples. If the 'Protection' bit is set, the frame header is followed by a 16 bit CRC.

	MPEG-1	MPEG-1	MPEG-1	MPEG-2	MPEG-2
bits	Layer-1	Layer-2	Layer-3	Layer-1	Layer-2/3
0000	free	free	free	free	free
0001	32	32	32	32	8
0010	64	48	40	48	16
0011	96	56	48	56	24
0100	128	64	56	64	32
0101	160	80	64	80	40
0110	192	96	80	96	48
0111	224	112	96	112	56
1000	256	128	112	128	64
1001	288	160	128	144	80
1010	320	192	160	160	96
1011	352	224	192	176	112
1100	384	256	224	192	128
1101	416	320	256	224	144
1110	448	384	320	256	160
1111	bad	bad	bad	bad	bad

Table 9.1: Bitrate in 1000 bits per second

To find the total size of an MPEG frame, use one of the following formulas:

Layer I:

$$\text{Byte Length} = \left(\frac{12 \times \text{Bitrate}}{\text{Sample Rate}} + \text{Pad} \right) \times 4 \tag{9.1}$$

Layer II/III:

$$\text{Byte Length} = \frac{144 \times \text{Bitrate}}{\text{Sample Rate}} + \text{Pad} \tag{9.2}$$

For example, an MPEG-1 Layer III frame with a sampling rate of 44100, a bitrate of 128kbps and a set pad bit is 418 bytes long, including the header.

$$\frac{144 \times 128000}{44100} + 1 = 418 \tag{9.3}$$

9.1.1 the Xing header

An MP3 frame header contains the track's sampling rate, bits-per-sample and number of channels. However, because MP3 files are little more than concatenated MPEG frames, there is no obvious place to store the track's total length. Since the length of each frame is a constant number of samples, one can calculate the track length by counting the number of frames. This method is the most accurate but is also quite slow.

For MP3 files in which all frames have the same bitrate - also known as constant bitrate, or CBR files - one can divide the total size of file (minus any ID3 headers/footers), by the bitrate to determine its length. If an MP3 file has no Xing header in its first frame, one can assume it is CBR.

An MP3 file that does contain a Xing header in its first frame can be assumed to be variable bitrate, or VBR. In that case, the rate of the first frame cannot be used as a basis to calculate the length of the entire file. Instead, one must use the information from the Xing header which contains that length.

All of the fields within a Xing header are big-endian.

Header `Xing' (0x58696E67)	0	31
Flags	32	63
Number of Frames	64	95
Bytes	96	127
TOC Entry$_1$	128	135
TOC Entry$_2$	136	143
...	144	919
TOC Entry$_{100}$	920	927
Quality	928	959

9.2 ID3v1 tags

ID3v1 tags are very simple metadata tags appended to an MP3 file. All of the fields are fixed length and the text encoding is undefined. There are two versions of ID3v1 tags. ID3v1.1 has a track number field as a 1 byte value at the end of the comment field. If the byte just before the end is not null (0x00), assume we're dealing with a classic ID3v1 tag without a track number.

9.2.1 ID3v1

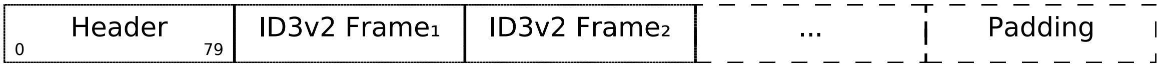
Header (`TAG' 0x544147)	0	23
Track Title	24	263
Artist Name	264	503
Album Name	504	743
Year	744	775
Comment	776	1015
Genre	1016	1023

9.2.2 ID3v1.1

Header (`TAG' 0x544147)	0	23
Track Title	24	263
Artist Name	264	503
Album Name	504	743
Year	744	775
Comment	776	999
NULL	1000	1007
Track Number	1008	1015
Genre	1016	1023

9.3 ID3v2 tags

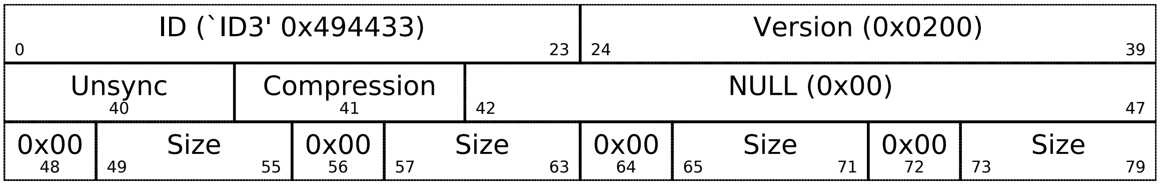
The ID3v2 tag was invented to address the deficiencies in the original ID3v1 tag. ID3v2 comes in three similar but not entirely compatible variants: ID3v2.2, ID3v2.3 and ID3v2.4. All of its fields are big-endian.



Header 0 79	ID3v2 Frame$_1$	ID3v2 Frame$_2$	...	Padding

9.3.1 ID3v2.2

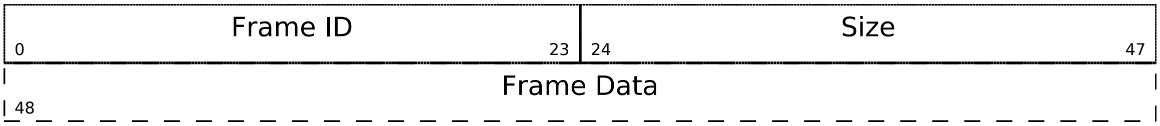
ID3v2.2 header



<table>
<tr><td colspan="4">ID (`ID3' 0x494433)
0 23</td><td colspan="4">Version (0x0200)
24 39</td></tr>
<tr><td colspan="2">Unsync
40</td><td colspan="2">Compression
41</td><td colspan="4">NULL (0x00)
42 47</td></tr>
<tr><td>0x00
48</td><td>Size
49 55</td><td>0x00
56</td><td>Size
57 63</td><td>0x00
64</td><td>Size
65 71</td><td>0x00
72</td><td>Size
73 79</td></tr>
</table>

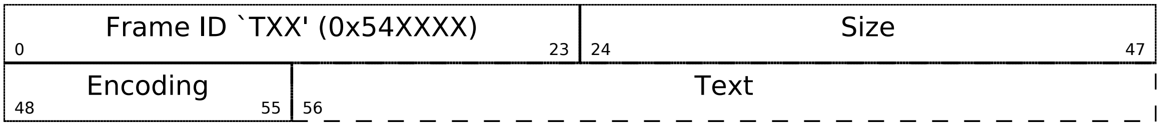
The single Size field is split by NULL (0x00) bytes in order to make it ‘sync-safe’. That is, no possible size value will result in a false MP3 frame sync (11 bits set in a row).

ID3v2.2 frame



<table>
<tr><td>Frame ID
0 23</td><td>Size
24 47</td></tr>
<tr><td colspan="2">Frame Data
48</td></tr>
</table>

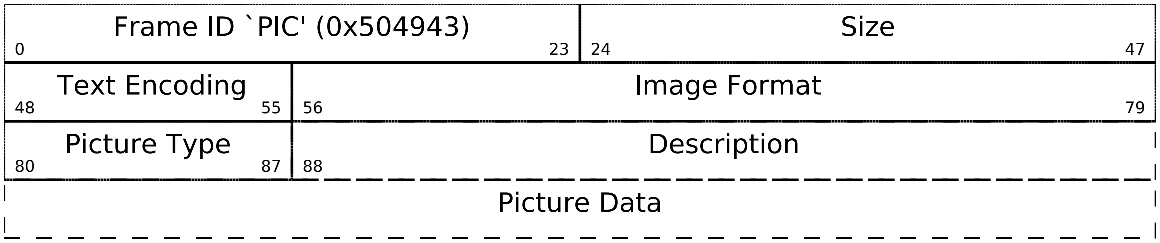
Frame ID’s that begin with the letter ‘T’ (0x54) are text frames. These have an additional text encoding byte before the actual text data. All text strings may be terminated by a null character (0x00 or 0x0000, depending on the encoding).



Encoding Byte	Text Encoding
0x00	ISO-8859-1
0x01	UCS-16

ID3v2.2 PIC frame

‘PIC’ frames are attached pictures. This allows an ID3v2.2 tag to contain a JPEG or PNG image, typically of album artwork which can be displayed to the user when the track is played.



Text Encoding is the encoding of the Description field. Its value is either ISO-8859-1 or UCS-16 - the same as in text frames. Image Format is a 3 byte string indicating the format of the image, typically ‘JPG’ for JPEG images or ’PNG’ for PNG images. Description is a NULL-terminated C-string which contains a text description of the image.

value	type	value	type
0	Other	1	32x32 pixels ‘file icon’ (PNG only)
2	Other file icon	3	Cover (front)
4	Cover (back)	5	Leaflet page
6	Media (e.g. label side of CD)	7	Lead artist / Lead performer / Soloist
8	Artist / Performer	9	Conductor
10	Band / Orchestra	11	Composer
12	Lyricist / Text writer	13	Recording location
14	During recording	15	During performance
16	Movie / Video screen capture	17	A bright coloured fish
18	Illustration	19	Band / Artist logotype
20	Publisher / Studio logotype		

Table 9.2: PIC image types

ID3v2.2 frame IDs

`BUF`	Recommended buffer size
`CNT`	Play counter
`COM`	Comments
`CRA`	Audio encryption
`CRM`	Encrypted meta frame
`ETC`	Event timing codes
`EQU`	Equalization
`GEO`	General encapsulated object
`IPL`	Involved people list
`LNK`	Linked information
`MCI`	Music CD Identifier
`MLL`	MPEG location lookup table
`PIC`	Attached picture
`POP`	Popularimeter
`REV`	Reverb
`RVA`	Relative volume adjustment
`SLT`	Synchronized lyric/text
`STC`	Synced tempo codes
`TAL`	Album/Movie/Show title
`TBP`	BPM (Beats Per Minute)
`TCM`	Composer
`TCO`	Content type
`TCR`	Copyright message
`TDA`	Date
`TDY`	Playlist delay
`TEN`	Encoded by
`TFT`	File type
`TIM`	Time
`TKE`	Initial key
`TLA`	Language(s)
`TLE`	Length
`TMT`	Media type
`TOA`	Original artist(s) / performer(s)
`TOF`	Original filename
`TOL`	Original Lyricist(s) / text writer(s)
`TOR`	Original release year
`TOT`	Original album / Movie / Show title
`TP1`	Lead artist(s) / performer(s) / Soloist(s) / Performing group
`TP2`	Band / Orchestra / Accompaniment
`TP3`	Conductor / Performer refinement
`TP4`	Interpreted, remixed, or otherwise modified by
`TPA`	Part of a set
`TPB`	Publisher
`TRC`	ISRC (International Standard Recording Code)
`TRD`	Recording dates
`TRK`	Track number / Position in set
`TSI`	Size
`TSS`	Software / hardware and settings used for encoding
`TT1`	Content group description
`TT2`	Title / Songname / Content description
`TT3`	Subtitle / Description refinement
`TXT`	Lyricist / text writer
`TXX`	User defined text information frame
`TYE`	Year
`UFI`	Unique file identifier
`ULT`	Unsychronized lyric / text transcription
`WAF`	Official audio file webpage
`WAR`	Official artist / performer webpage
`WAS`	Official audio source webpage
`WCM`	Commercial information
`WCP`	Copyright / Legal information
`WPB`	Publishers official webpage
`WXX`	User defined URL link frame

9.3.2 ID3v2.3

ID3v2.3 header

ID (`ID3' 0x494433) 0 23	Version (0x0300) 24 39						
Unsync 40	Extended 41	Experimental 42	Footer 43	NULL (0x00) 44 47			
0x00 48	Size 49 55	0x00 56	Size 57 63	0x00 64	Size 65 71	0x00 72	Size 73 79

The single Size field is split by NULL (0x00) bytes in order to make it 'sync-safe'.

ID3v2.3 frame

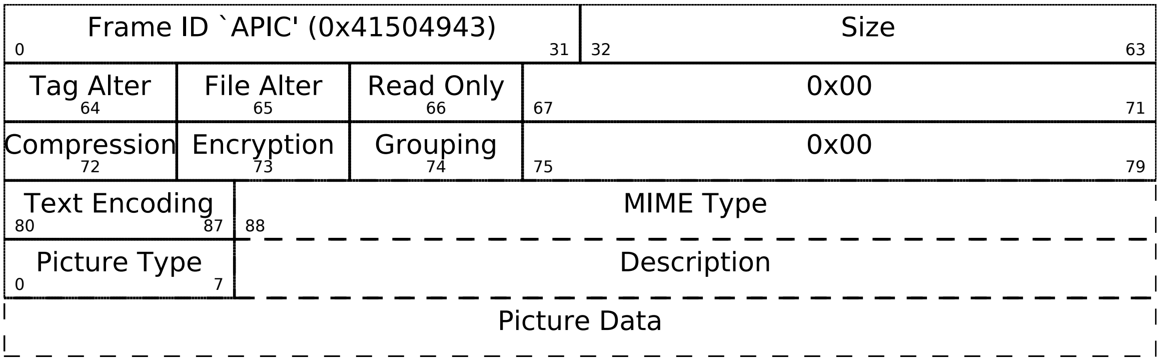
Frame ID 0 31	Size 32 63		
Tag Alter 64	File Alter 65	Read Only 66	NULL (0x00) 67 71
Compression 72	Encryption 73	Grouping 74	NULL (0x00) 75 79
Frame Data 80			

Frame ID's that begin with the letter 'T' (0x54) are text frames. These have an additional text encoding byte before the actual text data. All text strings may be terminated by a null character (0x00 or 0x0000, depending on the encoding).

Frame ID 'TXXX' (0x54XXXXXX) 0 31	Size 32 63		
Tag Alter 64	File Alter 65	Read Only 66	NULL (0x00) 67 71
Compression 72	Encryption 73	Grouping 74	NULL (0x00) 75 79
Encoding 80 87	Text 80		

Encoding Byte	Text Encoding
0x00	ISO-8859-1
0x01	UCS-16

ID3v2.3 APIC frame



Text Encoding is the encoding of the Description field. Its value is either ISO-8859-1 or UCS-16 - the same as in text frames. MIME Type is a NULL-terminated, ASCII C-string which contains the image's MIME type, such as 'image/jpeg' or 'image/png'. Description is a NULL-terminated C-string which contains a text description of the image.

value	type	value	type
0	Other	1	32x32 pixels 'file icon' (PNG only)
2	Other file icon	3	Cover (front)
4	Cover (back)	5	Leaflet page
6	Media (e.g. label side of CD)	7	Lead artist / Lead performer / Soloist
8	Artist / Performer	9	Conductor
10	Band / Orchestra	11	Composer
12	Lyricist / Text writer	13	Recording location
14	During recording	15	During performance
16	Movie / Video screen capture	17	A bright coloured fish
18	Illustration	19	Band / Artist logotype
20	Publisher / Studio logotype		

Table 9.3: APIC image types

ID3v2.3 frame IDs

AENC	Audio encryption
APIC	Attached picture
COMM	Comments
COMR	Commercial frame
ENCR	Encryption method registration
EQUA	Equalization
ETCO	Event timing codes
GEOB	General encapsulated object
GRID	Group identification registration
IPLS	Involved people list
LINK	Linked information
MCDI	Music CD identifier
MLLT	MPEG location lookup table
OWNE	Ownership frame
PRIV	Private frame
PCNT	Play counter
POPM	Popularimeter
POSS	Position synchronisation frame
RBUF	Recommended buffer size
RVAD	Relative volume adjustment
RVRB	Reverb
SYLT	Synchronized lyric / text
SYTC	Synchronized tempo codes
TALB	Album /Movie / Show title
TBPM	BPM (beats per minute)
TCOM	Composer
TCON	Content type
TCOP	Copyright message
TDAT	Date
TDLY	Playlist delay
TENC	Encoded by
TEXT	Lyricist / Text writer
TFLT	File type
TIME	Time
TIT1	Content group description
TIT2	Title / songname / content description
TIT3	Subtitle / Description refinement
TKEY	Initial key
TLAN	Language(s)
TLEN	Length
TMED	Media type
TOAL	Original album/movie/show title
TOFN	Original filename
TOLY	Original lyricist(s) / text writer(s)
TOPE	Original artist(s) / performer(s)
TORY	Original release year
TOWN	File owner / licensee
TPE1	Lead performer(s) / Soloist(s)
TPE2	Band / orchestra / accompaniment
TPE3	Conductor / performer refinement
TPE4	Interpreted, remixed, or otherwise modified by
TPOS	Part of a set
TPUB	Publisher
TRCK	Track number / Position in set
TRDA	Recording dates
TRSN	Internet radio station name
TRSO	Internet radio station owner
TSIZ	Size
TSRC	ISRC (international standard recording code)
TSSE	Software/Hardware and encoding settings
TYER	Year
TXXX	User defined text information frame
UFID	Unique file identifier
USER	Terms of use
USLT	Unsychronized lyric / text transcription
WCOM	Commercial information
WCOP	Copyright / Legal information
WOAF	Official audio file webpage
WOAR	Official artist/performer webpage
WOAS	Official audio source webpage
WORS	Official internet radio station homepage
WPAY	Payment
WPUB	Publishers official webpage
WXXX	User defined URL link frame

9.3.3 ID3v2.4

ID3v2.4 header

ID (`ID3' 0x494433) 0 23	Version (0x0400) 24 39						
Unsync 40	Extended 41	Experimental 42	Footer 43	NULL (0x00) 44 47			
0x00 48	Size 49 55	0x00 56	Size 57 63	0x00 64	Size 65 71	0x00 72	Size 73 79

ID3v2.4 frame

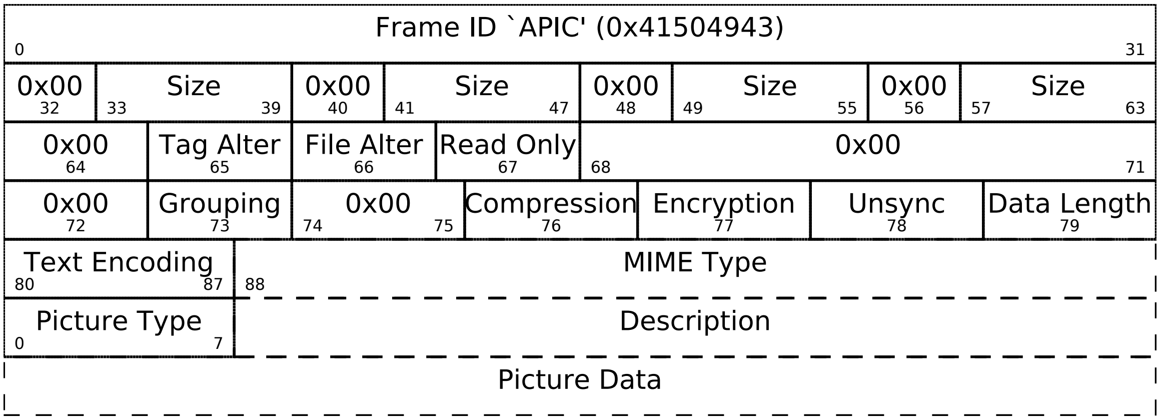
Frame ID 0 31							
0x00 32	Size 33 39	0x00 40	Size 41 47	0x00 48	Size 49 55	0x00 56	Size 57 63
0x00 64	Tag Alter 65	File Alter 66	Read Only 67	0x00 68 71			
0x00 72	Grouping 73	0x00 74 75	Compression 76	Encryption 77	Unsync 78	Data Length 79	
Frame Data 80							

Frame ID's that begin with the letter 'T' (0x54) are text frames. These have an additional text encoding byte before the actual text data. All text strings may be terminated by a null character (0x00 or 0x0000, depending on the encoding).

Frame ID 'TXXX' (0x54XXXXXX) 0 31							
0x00 32	Size 33 39	0x00 40	Size 41 47	0x00 48	Size 49 55	0x00 56	Size 57 63
0x00 64	Tag Alter 65	File Alter 66	Read Only 67	0x00 68 71			
0x00 72	Grouping 73	0x00 74 75	Compression 76	Encryption 77	Unsync 78	Data Length 79	
Encoding 80 87	Text 80						

Encoding Byte	Text Encoding
0x00	ISO-8859-1
0x01	UTF-16
0x02	UTF-16BE
0x03	UTF-8

ID3v2.4 APIC frame



Text Encoding is the encoding of the Description field. Its value is either ISO-8859-1, UTF-16 or UTF-8 - the same as in text frames. MIME Type is a NULL-terminated, ASCII C-string which contains the image's MIME type, such as 'image/jpeg' or 'image/png'. Description is a NULL-terminated C-string which contains a text description of the image.

value	type	value	type
0	Other	1	32x32 pixels 'file icon' (PNG only)
2	Other file icon	3	Cover (front)
4	Cover (back)	5	Leaflet page
6	Media (e.g. label side of CD)	7	Lead artist / Lead performer / Soloist
8	Artist / Performer	9	Conductor
10	Band / Orchestra	11	Composer
12	Lyricist / Text writer	13	Recording location
14	During recording	15	During performance
16	Movie / Video screen capture	17	A bright coloured fish
18	Illustration	19	Band / Artist logotype
20	Publisher / Studio logotype		

Table 9.4: APIC image types

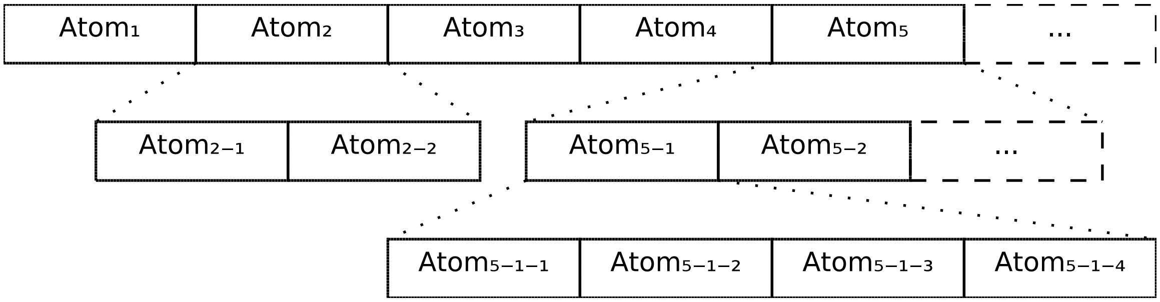
ID3v2.4 frame IDs

AENC	Audio encryption
APIC	Attached picture
ASPI	Audio seek point index
COMM	Comments
COMR	Commercial frame
ENCR	Encryption method registration
EQU2	Equalisation (2)
ETCO	Event timing codes
GEOB	General encapsulated object
GRID	Group identification registration
LINK	Linked information
MCDI	Music CD identifier
MLLT	MPEG location lookup table
OWNE	Ownership frame
PRIV	Private frame
PCNT	Play counter
POPM	Popularimeter
POSS	Position synchronisation frame
RBUF	Recommended buffer size
RVA2	Relative volume adjustment (2)
RVRB	Reverb
SEEK	Seek frame
SIGN	Signature frame
SYLT	Synchronised lyric/text
SYTC	Synchronised tempo codes
TALB	Album/Movie/Show title
TBPM	BPM (beats per minute)
TCOM	Composer
TCON	Content type
TCOP	Copyright message
TDEN	Encoding time
TDLY	Playlist delay
TDOR	Original release time
TDRC	Recording time
TDRL	Release time
TDTG	Tagging time
TENC	Encoded by
TEXT	Lyricist/Text writer
TFLT	File type
TIPL	Involved people list
TIT1	Content group description
TIT2	Title/songname/content description
TIT3	Subtitle/Description refinement
TKEY	Initial key
TLAN	Language(s)
TLEN	Length
TMCL	Musician credits list
TMED	Media type
TMOO	Mood
TOAL	Original album/movie/show title
TOFN	Original filename
TOLY	Original lyricist(s)/text writer(s)
TOPE	Original artist(s)/performer(s)
TOWN	File owner/licensee
TPE1	Lead performer(s)/Soloist(s)
TPE2	Band/orchestra/accompaniment
TPE3	Conductor/performer refinement
TPE4	Interpreted, remixed, or otherwise modified by
TPOS	Part of a set
TPRO	Produced notice
TPUB	Publisher
TRCK	Track number/Position in set
TRSN	Internet radio station name
TRSO	Internet radio station owner
TSOA	Album sort order
TSOP	Performer sort order
TSOT	Title sort order
TSRC	ISRC (international standard recording code)
TSSE	Software/Hardware and settings used for encoding
TSST	Set subtitle
TXXX	User defined text information frame
UFID	Unique file identifier
USER	Terms of use
USLT	Unsynchronised lyric/text transcription
WCOM	Commercial information
WCOP	Copyright/Legal information
WOAF	Official audio file webpage
WOAR	Official artist/performer webpage
WOAS	Official audio source webpage
WORS	Official Internet radio station homepage
WPAY	Payment
WPUB	Publishers official webpage
WXXX	User defined URL link frame

10 M4A

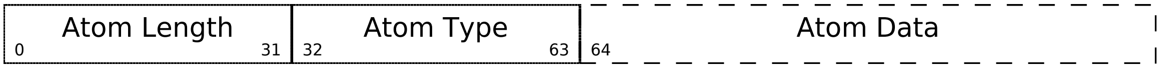
M4A is typically AAC audio in a QuickTime container stream, though it may also contain other formats such as MPEG-1 audio.

10.1 the QuickTime file stream

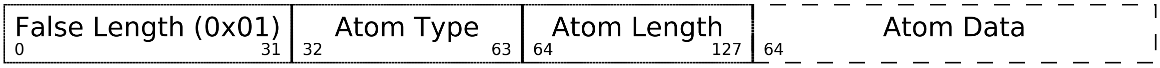


Unlike other chunked formats such as RIFF WAVE, QuickTime's atom chunks may be containers for other atoms. All of its fields are big-endian.

10.1.1 a QuickTime atom



Atom Type is an ASCII string. Atom Length is the length of the entire atom, including the header. If Atom Length is 0, the atom continues until the end of the file. If Atom Length is 1, the atom has an extended size. This means there is a 64-bit length field immediately after the header which is the atom's actual size.

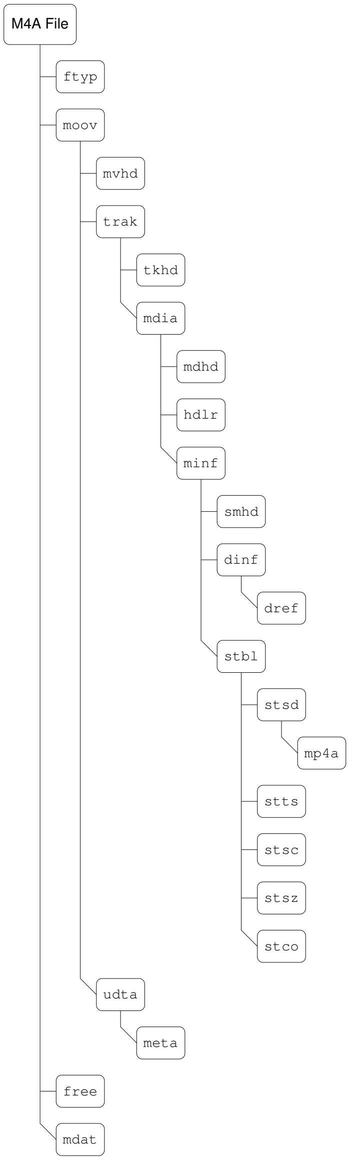


10.1.2 Container atoms

There is no flag or field to tell a QuickTime parser which of its atoms are containers and which ones are not. If an atom is known to be a container, one can treat its Atom Data as a QuickTime stream and parse it in a recursive fashion.

10.2 M4A atoms

A typical M4A begins with an ‘ftyp’ atom indicating its file type, followed by a ‘moov’ atom containing a copious amount of file metadata, an optional ‘free’ atom with nothing but empty space (so that metadata can be resized, if necessary) and an ‘mdat’ atom containing the song data itself.



10.2.1 the ftyp atom

ftyp Length 0 31	`ftyp' (0x66747970) 32 63
Major Brand 64 95	Major Brand Version 96 127
Compatible Brand$_1$ 128 159	... 160

The ‘Major Brand’ and ‘Compatible Brand’ fields are ASCII strings. ‘Major Brand Version’ is an integer.

10.2.2 the mvhd atom

mvhd Length 0 31		`mvhd' (0x6D766864) 32 63	
Version 64 71	Flags (0x000000) 72 95		
Created Mac UTC Date 96 127/159		Modified Mac UTC Date 128/160 159/223	
Time Scale 160/224 191/255		Duration 192/256 223/319	
Playback Speed 224/320 255/351		User Volume 256/352 271/367	Reserved (0) 272/368 351/447
WGM A 352/448 383/479	WGM B 384/480 415/511	WGM U 416/512 447/543	WGM C 448/544 479/575
WGM D 480/576 511/607	WGM V 512/608 543/639	WGM X 544/640 575/671	WGM Y 576/672 607/703
WGM W 608/704 639/735	QuickTime Preview 640/736 703/799		
QuickTime Still Poster 704/800 735/831		QuickTime Selection Time 736/832 799/895	
QuickTime Current Time 800/896 831/927		next/new track ID 832/928 863/959	

If ‘Version’ is 0, ‘Created Mac UTC Date’, ‘Modified Mac UTC Date’ and ‘Duration’ are 32-bit fields. If it is 1, they are 64-bit fields.

10.2.3 the tkhd atom

<table>
<tr><td colspan="12">tkhd Length
0 31</td><td colspan="12">`tkhd' (0x746B6864)
32 63</td></tr>
<tr><td colspan="3">Version
64 71</td><td colspan="4">Reserved (0)
72 91</td><td colspan="4">Track in Poster
92</td><td colspan="5">Track in Preview
93</td><td colspan="4">Track in Movie
94</td><td colspan="4">Track Enabled
95</td></tr>
<tr><td colspan="12">Created Mac UTC Date
96 127/159</td><td colspan="12">Modified Mac UTC Date
128/160 159/223</td></tr>
<tr><td colspan="6">Track ID
160/224 191/255</td><td colspan="18">Reserved (0)
192/256 255/319</td></tr>
<tr><td colspan="12">Duration
256/320 287/383</td><td colspan="12">Reserved (0)
288/384 319/415</td></tr>
<tr><td colspan="6">Video Layer
320/415 335/431</td><td colspan="6">QuickTime Alt
336/432 351/447</td><td colspan="6">Audio Volume
352/448 367/463</td><td colspan="6">Reserved (0)
368/464 383/479</td></tr>
<tr><td colspan="6">VGM value A
384/480 415/511</td><td colspan="6">VGM value B
416/512 447/543</td><td colspan="6">VGM value U
448/544 479/575</td><td colspan="6">VGM value C
480/576 511/607</td></tr>
<tr><td colspan="6">VGM value D
512/608 543/639</td><td colspan="6">VGM value V
544/640 575/671</td><td colspan="6">VGM value X
576/672 607/703</td><td colspan="6">VGM value Y
608/704 639/735</td></tr>
<tr><td colspan="6">VGM value W
640/736 671/767</td><td colspan="18">Video Frame Size
672/768 735/831</td></tr>
</table>

As with ‘mvhd’, if ‘Version’ is 0, ‘Created Mac UTC Date’, ‘Modified Mac UTC Date’ and ‘Duration’ are 32-bit fields. If it is 1, they are 64-bit fields.

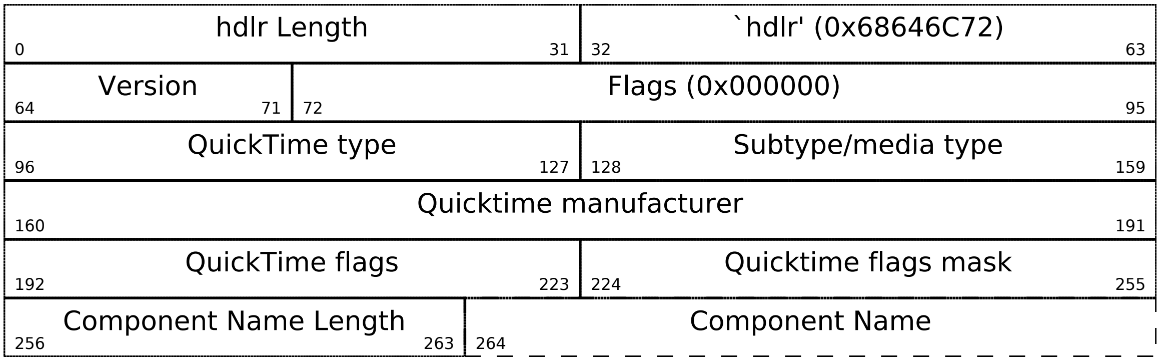
10.2.4 the mdhd atom

The `mdhd` atom contains track information such as samples-per-second, track length and creation/modification times.

<table>
<tr><td colspan="12">mdhd Length
0 31</td><td colspan="12">`mdhd' (0x6D646864)
32 63</td></tr>
<tr><td colspan="6">Version
64 71</td><td colspan="18">Flags (0x000000)
72 95</td></tr>
<tr><td colspan="12">Created Mac UTC Date
96 127/159</td><td colspan="12">Modified Mac UTC Date
128/60 159/223</td></tr>
<tr><td colspan="12">Sample Rate
160/224 191/255</td><td colspan="12">Track Length
192/256 223/319</td></tr>
<tr><td colspan="5">Pad
224/320</td><td colspan="7">Language
225/321 239/335</td><td colspan="12">Quality
240/336 255/351</td></tr>
</table>

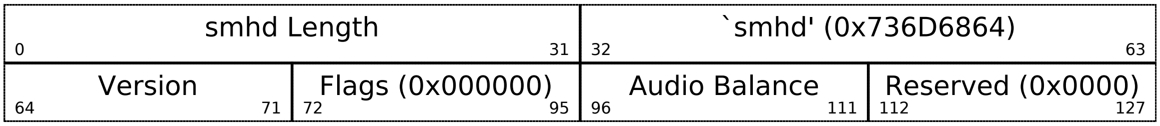
As with ‘mvhd’, if ‘Version’ is 0, ‘Created Mac UTC Date’, ‘Modified Mac UTC Date’ and ‘Track Length’ are 32-bit fields. If it is 1, they are 64-bit fields.

10.2.5 the hdlr atom

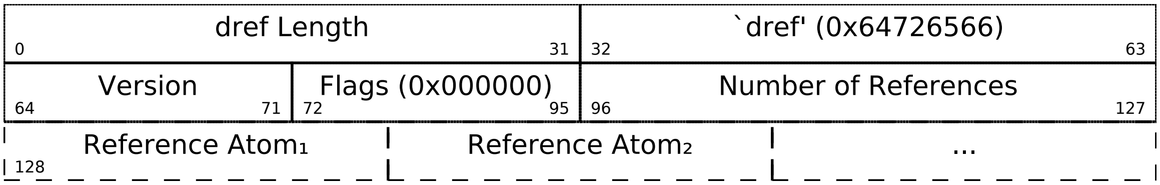


'QuickTime flags', 'QuickTime flags mask' and 'Component Name Length' are integers. The rest are ASCII strings.

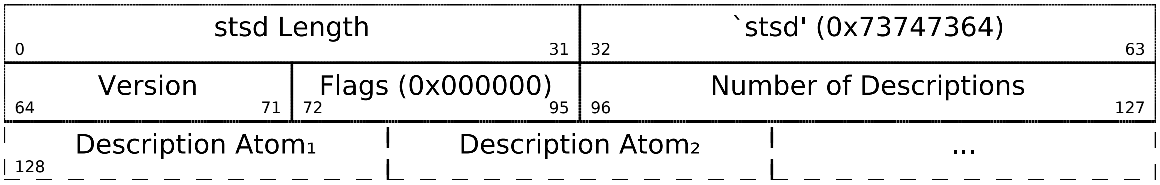
10.2.6 the smhd atom



10.2.7 the dref atom

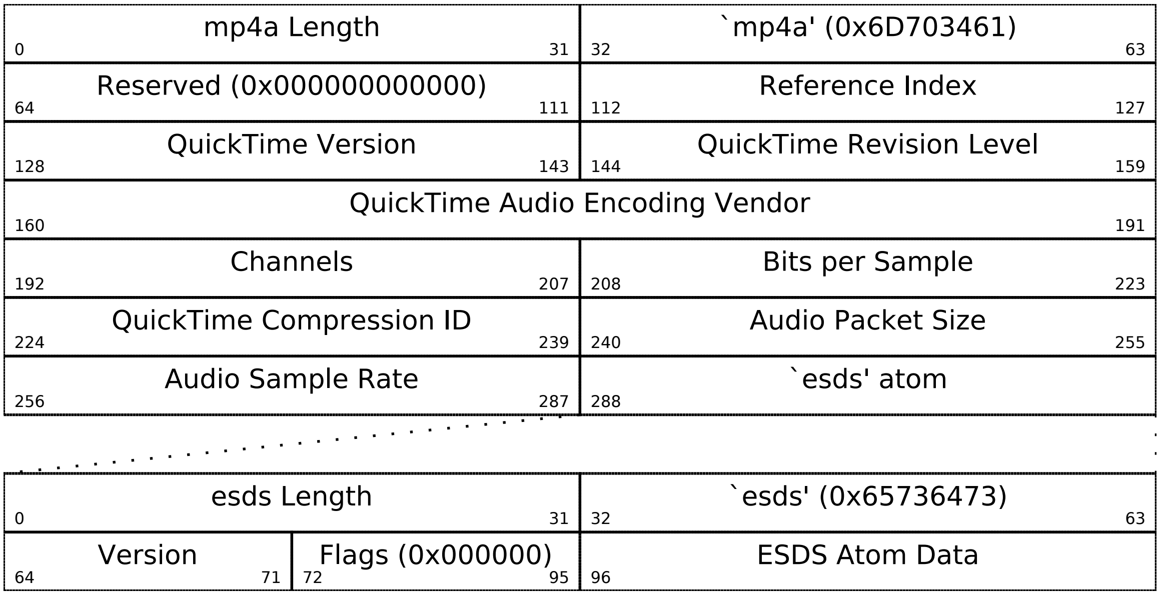


10.2.8 the stsd atom



10.2.9 the mp4a atom

The mp4a atom contains information such as the number of channels and bits-per-sample. It can be found in the stsd atom.



10.2.10 the stts atom

<table>
<tr><td colspan="2">stts Length
0 31</td><td colspan="2">`stts' (0x73747473)
32 63</td></tr>
<tr><td>Version
64 71</td><td>Flags (0x000000)
72 95</td><td colspan="2">Number of Times
96 127</td></tr>
<tr><td colspan="2">Frame Count$_1$
128 159</td><td colspan="2">Duration$_1$
160 191</td></tr>
<tr><td colspan="2">Frame Count$_2$
192 223</td><td colspan="2">Duration$_2$
224 255</td></tr>
<tr><td colspan="4">...
256</td></tr>
</table>

10.2.11 the stsc atom

<table>
<tr><td colspan="6">stsc Length
0 31</td><td colspan="6">`stsc' (0x73747363)
32 63</td></tr>
<tr><td colspan="3">Version
64 71</td><td colspan="3">Flags (0x000000)
72 95</td><td colspan="6">Number of Blocks
96 127</td></tr>
<tr><td colspan="4">First Chunk$_1$
128 159</td><td colspan="4">Samples per Chunk$_1$
160 191</td><td colspan="4">Sample Duration Index$_1$
192 223</td></tr>
<tr><td colspan="4">First Chunk$_2$
224 255</td><td colspan="4">Samples per Chunk$_2$
256 287</td><td colspan="4">Sample Duration Index$_2$
288 319</td></tr>
<tr><td colspan="12">...
320</td></tr>
</table>

10.2.12 the stsz atom

<table>
<tr><td colspan="6">stsz Length
0 31</td><td colspan="6">`stsz' (0x7374737A)
32 63</td></tr>
<tr><td colspan="3">Version
64 71</td><td colspan="3">Flags (0x000000)
72 95</td><td colspan="6">Number of Block Sizes
96 127</td></tr>
<tr><td colspan="4">Block Size$_1$
128 159</td><td colspan="4">Block Size$_2$
160 191</td><td colspan="4">...
192</td></tr>
</table>

10.2.13 the stco atom

stsz Length 0 31	`stsz' (0x7374737A) 32 63

Version 64 71	Flags (0x000000) 72 95	Number of Offsets 96 127

Offset$_1$ 128 159	Offset$_2$ 160 191	... 192

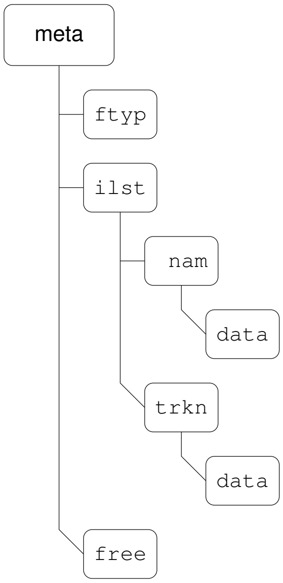
Offsets point to an absolute position in the M4A file of AAC data in the `mdat` atom. Therefore, if the `moov` atom size changes (which can happen by writing new metadata in its `meta` child atom) the `mdat` atom may move and these obsolute offsets will change. In that instance, they **must** be re-adjusted in the `stco` atom or the file may become unplayable.

10.2.14 the meta atom

meta Length 0 31	`meta' (0x6D657461) 32 63

Version 64 71	Flags (0x000000) 72 95

`ftyp' atom 96	`ilst' atom	`free' atom	...



The atoms within the `ilst` container are all containers themselves, each with a `data` atom of its own. Notice that many of `ilst`'s sub-atoms begin with the non-ASCII 0xA9 byte.

data Length 0 31	`data' (0x64617461) 32 63
Type 64 95	Reserved (0x00000000) 96 127

Data 128

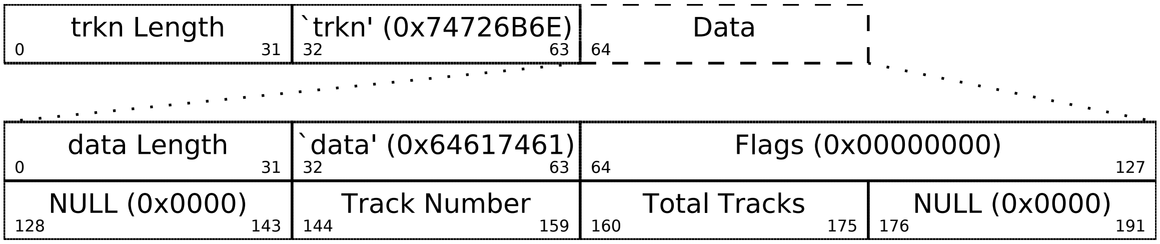
Text data atoms have a type of 1. Binary data atoms typically have a type of 0.

Atom	Description	Atom	Description	Atom	Description
`alb`	Album Nam	`ART`	Track Artist	`cmt`	Comments
`covr`	Cover Image	`cpil`	Compilation	`cprt`	Copyright
`day`	Year	`disk`	Disc Number	`gnre`	Genre
`grp`	Grouping	`----`	iTunes-specific	`nam`	Track Name
`rtng`	Rating	`tmpo`	BMP	`too`	Encoder
`trkn`	Track Number	`wrt`	Composer		

Table 10.1: Known `ilst` sub-atoms

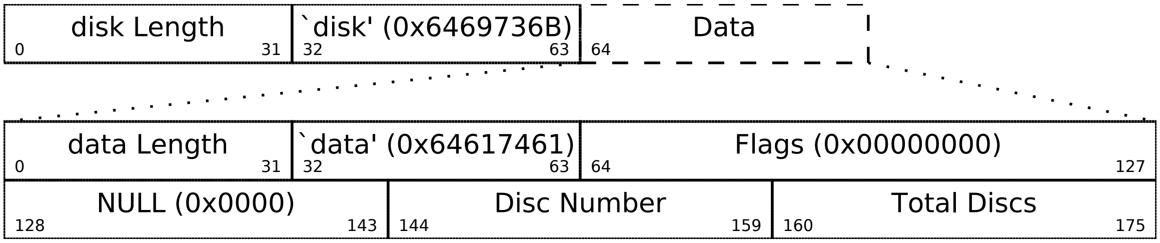
the trkn sub-atom

`trkn` is a binary sub-atom of `meta` which contains the track number.



the disk sub-atom

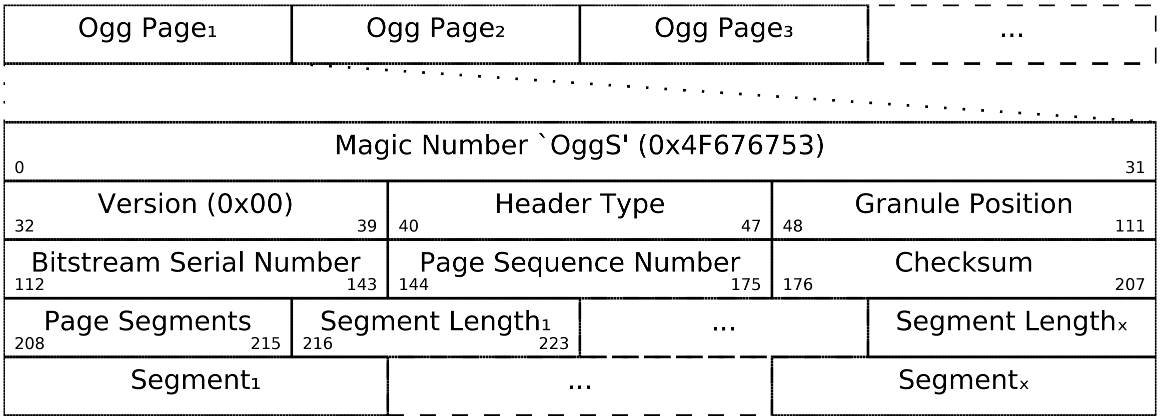
`disk` is a binary sub-atom of `meta` which contains the disc number. For example, if the track belongs to the first disc in a set of two discs, the sub-atom will contain that information.



11 Ogg Vorbis

Ogg Vorbis is Vorbis audio in an Ogg container. Ogg containers are a series of Ogg pages, each containing one or more segments of data. All of the fields within Ogg Vorbis are little-endian.

11.1 Ogg file stream



bits	Header Type
001	Continuation
010	Beginning of Stream
100	End of Stream

'Granule position' is a time marker. In the case of Ogg Vorbis, it is the sample count.

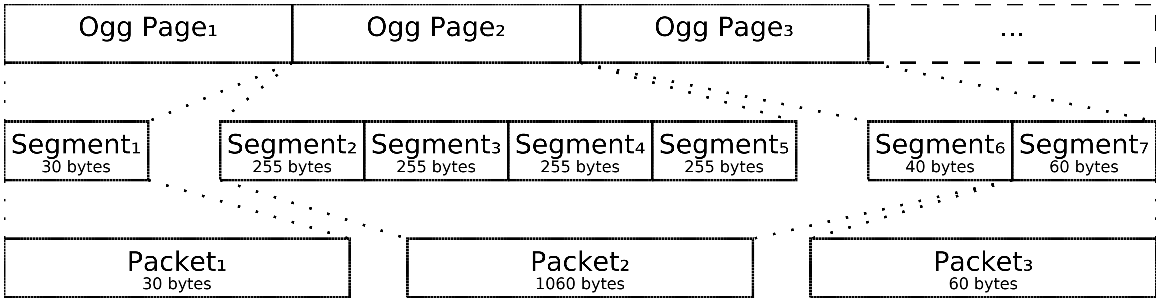
'Bitstream Serial Number' is an identifier for the given bitstream which is unique within the Ogg file. For instance, an Ogg file might contain both video and audio pages, interleaved. The Ogg pages for the audio will have a different serial number from those of the video so that the decoder knows where to send the data of each.

'Page Sequence Number' is an integer counter which starts from 0 and increments 1 for each Ogg page. Multiple bitstreams will have separate sequence numbers.

'Checksum' is a 32-bit checksum of the entire Ogg page.

The 'Page Segments' value indicates how many segments are in this Ogg page. Each segment will have an 8-bit length. If that length is 255, it indicates the next segment is part of the current one and should be concatenated with it when creating packets from the segments. In this way, packets larger than 255 bytes can be stored in an Ogg page. If the final segment in the Ogg page has a length of 255 bytes, the packet it is a part of continues into the next Ogg page.

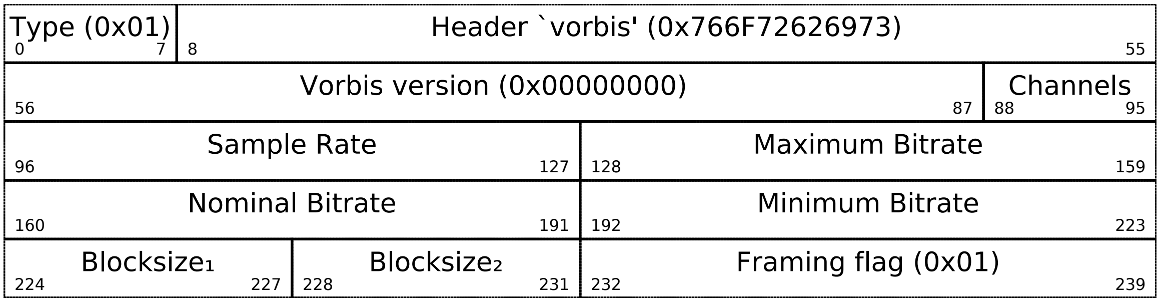
11.1.1 Ogg packets



This is an example Ogg stream to illustrate a few key points about the format. Note that Ogg pages may have one or more segments, and packets are composed of one of more segments, yet the boundaries between packets are segments that are less than 255 bytes long. Which segment belongs to which Ogg page is not important for building packets.

11.2 the Identification packet

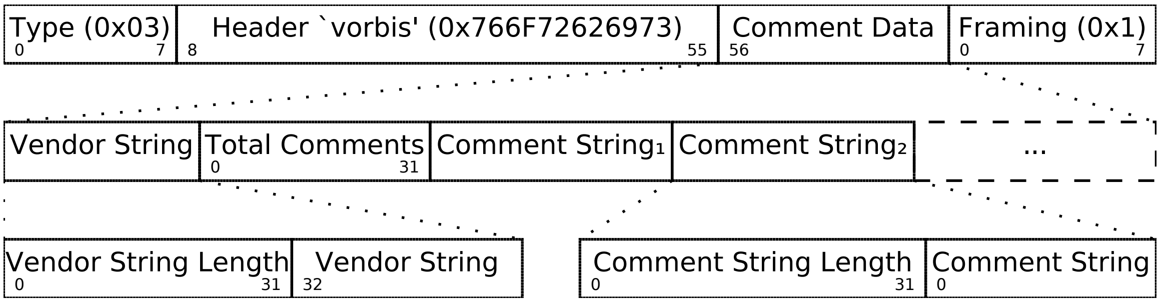
The first packet within a Vorbis stream is the Identification packet. This contains the sample rate and number of channels. Vorbis does not have a bits-per-sample field, as samples are stored internally as floating point values and are converted into a certain number of bits in the decoding process. To find the total samples, use the 'Granule Position' value in the stream's final Ogg page.



Type (0x01)	0	7
Header `vorbis' (0x766F72626973)	8	55
Vorbis version (0x00000000)	56	87
Channels	88	95
Sample Rate	96	127
Maximum Bitrate	128	159
Nominal Bitrate	160	191
Minimum Bitrate	192	223
Blocksize1	224	227
Blocksize2	228	231
Framing flag (0x01)	232	239

11.3 the Comment packet

The second packet within a Vorbis stream is the Comment packet.



The length fields are all little-endian. The Vendor String and Comment Strings are all UTF-8 encoded. Keys are not case-sensitive and may occur multiple times, indicating multiple values for the same field. For instance, a track with multiple artists may have more than one `ARTIST`.

ALBUM album name

ARTIST artist name, band name, composer, author, etc.

CATALOGNUMBER* CD spine number

COMPOSER* the work's author

CONDUCTOR* performing ensemble's leader

COPYRIGHT copyright attribution

DATE recording date

DESCRIPTION a short description

DISCNUMBER* disc number for multi-volume work

ENGINEER* the recording masterer

ENSEMBLE* performing group

GENRE a short music genre label

GUEST ARTIST* collaborating artist

ISRC ISRC number for the track

LICENSE license information

LOCATION recording location

OPUS* number of the work

ORGANIZATION record label

PART* track's movement title

PERFORMER performer name, orchestra, actor, etc.

PRODUCER* person responsible for the project

PRODUCTNUMBER* UPC, EAN, or JAN code

PUBLISHER* album's publisher

RELEASE DATE* date the album was published

REMIXER* person who created the remix

SOURCE ARTIST* artist of the work being performed

SOURCE MEDIUM* CD, radio, cassette, vinyl LP, etc.

SOURCE WORK* a soundtrack's original work

SPARS* DDD, ADD, AAD, etc.

SUBTITLE* for multiple track names in a single file

TITLE track name

TRACKNUMBER track number

VERSION track version

Fields marked with * are proposed extension fields and not part of the official Vorbis comment specification.

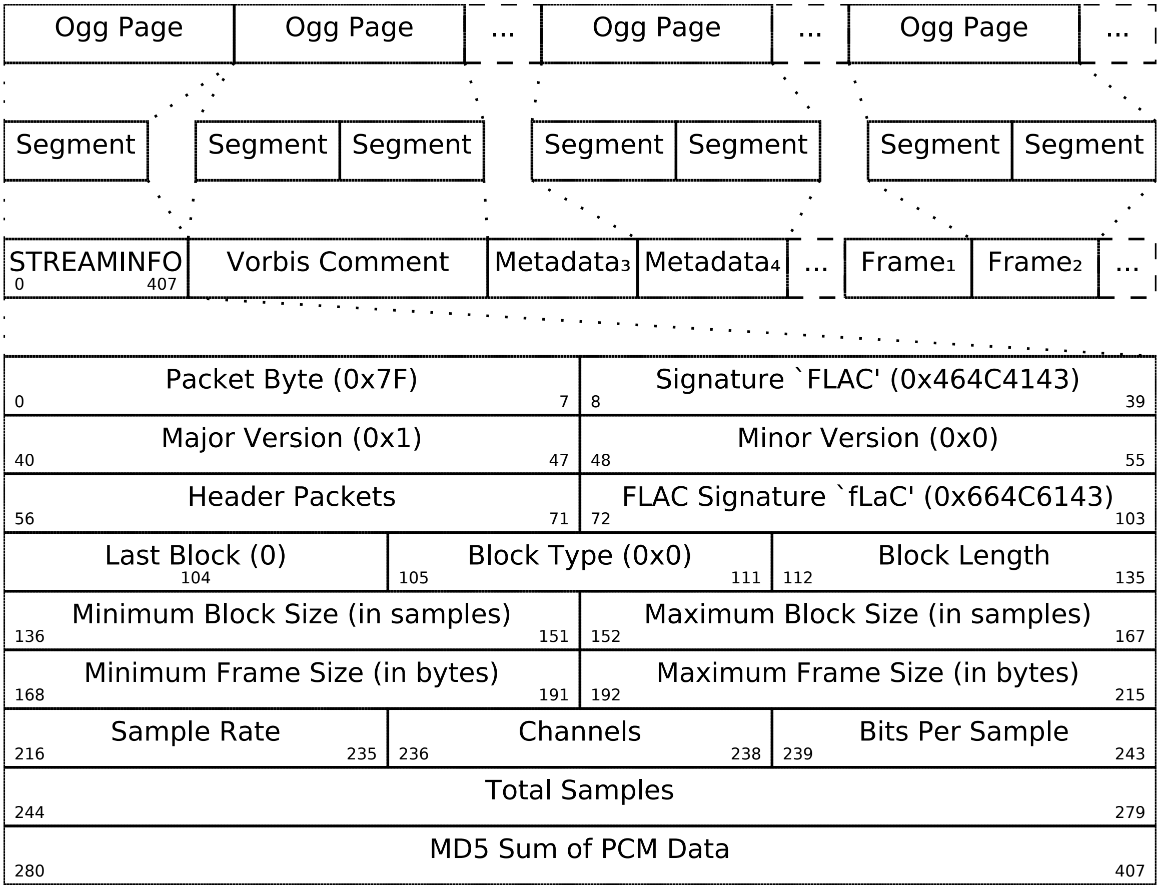
11.4 Channel assignment

channel count	channel 1	channel 2	channel 3	channel 4	channel 5	channel 6	channel 7	channel 8
1	front center							
2	front left	front right						
3	front left	front center	front right					
4	front left	front right	back left	back right				
5	front left	front center	front right	back left	back right			
6	front left	front center	front right	back left	back right	LFE		
7	front left	front center	front right	side left	side right	back center	LFE	
8	front left	front center	front right	side left	side right	back left	back right	LFE
8+	defined by application							

12 Ogg FLAC

Ogg FLAC is a FLAC audio stream in an Ogg container.

12.1 the Ogg FLAC file stream



Field	Start bit	End bit
Packet Byte (0x7F)	0	7
Signature `FLAC' (0x464C4143)	8	39
Major Version (0x1)	40	47
Minor Version (0x0)	48	55
Header Packets	56	71
FLAC Signature `fLaC' (0x664C6143)	72	103
Last Block (0)	104	
Block Type (0x0)	105	111
Block Length	112	135
Minimum Block Size (in samples)	136	151
Maximum Block Size (in samples)	152	167
Minimum Frame Size (in bytes)	168	191
Maximum Frame Size (in bytes)	192	215
Sample Rate	216	235
Channels	236	238
Bits Per Sample	239	243
Total Samples	244	279
MD5 Sum of PCM Data	280	407

Subsequent FLAC metadata blocks are stored 1 per packet. Each contains the 32-bit FLAC metadata block header in addition to the metadata itself. The `VORBIS_COMMENT` metadata block is required to immediately follow the `STREAMINFO` block, but all others may appear in any order.

13 Ogg Speex

Ogg Speex is Speex audio in an Ogg container. Speex is a lossy audio codec optimized for speech. All of the fields within Ogg Speex are little-endian.

How Ogg containers break up data packets into segments and pages has already been explained in the Ogg Vorbis section on page 75. Therefore, I shall move directly to the Ogg Speex packets themselves.

13.1 the header packet

The first packet within a Speex stream is the Header packet. It contains the number of channels and sampling rate. Like Vorbis, the number of bits per sample is generated during decoding and the total number of samples is pulled from the 'Granule Position' field in the Ogg stream.

<table>
<tr><td colspan="6">Speex String `Speex   ' (0x5370656578202020)
0 63</td></tr>
<tr><td colspan="3">Speex Version
64 223</td><td colspan="3">Speex Version ID
224 255</td></tr>
<tr><td colspan="2">Header Size
256 287</td><td colspan="2">Sampling Rate
288 319</td><td colspan="2">Mode
320 351</td></tr>
<tr><td colspan="2">Mode Bitstream Version
352 383</td><td colspan="2">Number of Channels
384 415</td><td colspan="2">Bitrate
416 447</td></tr>
<tr><td colspan="2">Frame Size
448 479</td><td colspan="2">VBR
480 511</td><td colspan="2">Frames Per Packet
512 543</td></tr>
<tr><td colspan="2">Extra Headers
544 575</td><td colspan="2">Reserved₁
576 607</td><td colspan="2">Reserved₂
608 639</td></tr>
</table>

13.2 the comment packet

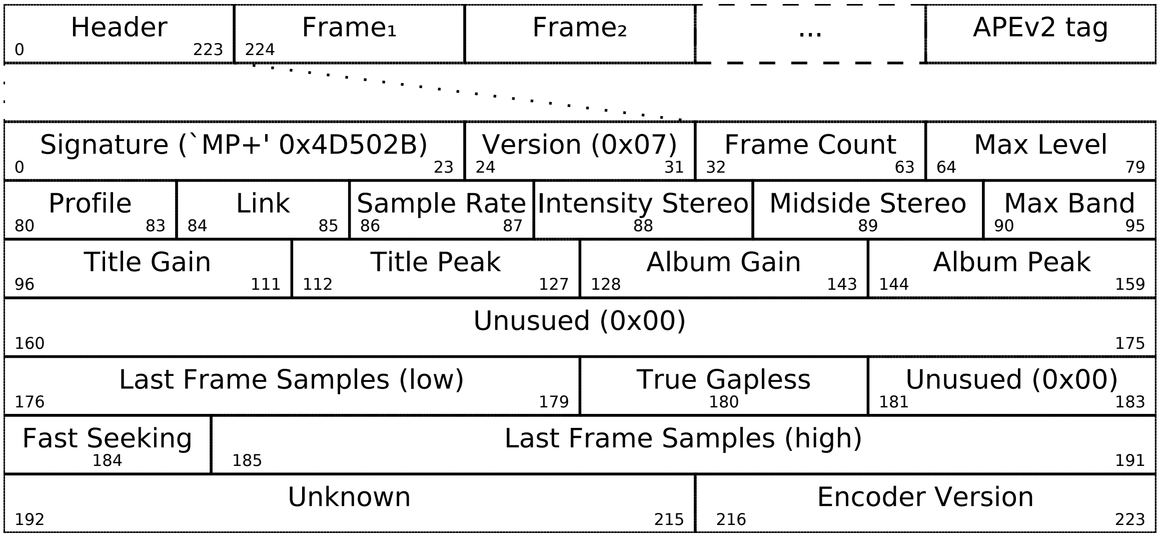
The second packet within a Speex stream is the Comment packet. This is identical to the comments used by Ogg Vorbis which is detailed on page 77.

14 Musepack

Musepack is a lossy audio format based on MP2 and designed for transparency. It comes in two varieties: SV7 and SV8 where 'SV' stands for Stream Version. These container versions differ so heavily that they must be considered separately from one another.

14.1 the SV7 file stream

This is the earliest version of Musepack with wide support. All of its fields are little-endian. Each frame contains 1152 samples per channel. Therefore:



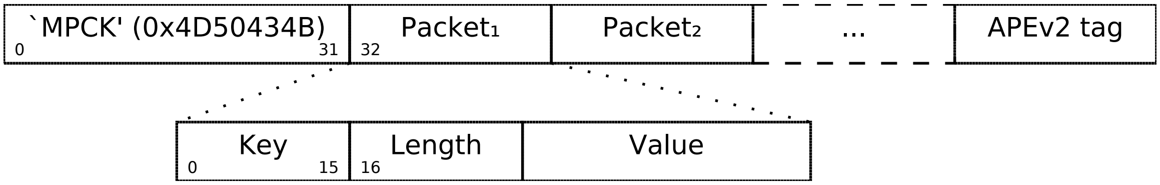
$$\text{Total Samples} = ((\text{Frame Count} - 1) \times 1152) + \text{Last Frame Samples} \tag{14.1}$$

Musepack files always have exactly 2 channels and its lossy samples are stored as floating point. Its sampling rate is one of four values:

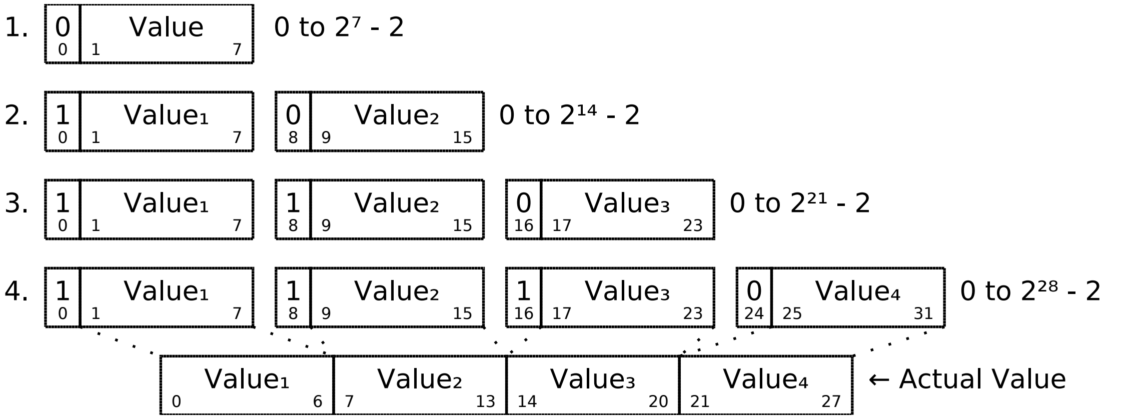
$\texttt{00} =$ 44100Hz, $\texttt{01} =$ 48000Hz, $\texttt{10} =$ 37800Hz, $\texttt{11} =$ 32000Hz .

14.2 the SV8 file stream

This is the latest version of the Musepack stream. All of its fields are big-endian.

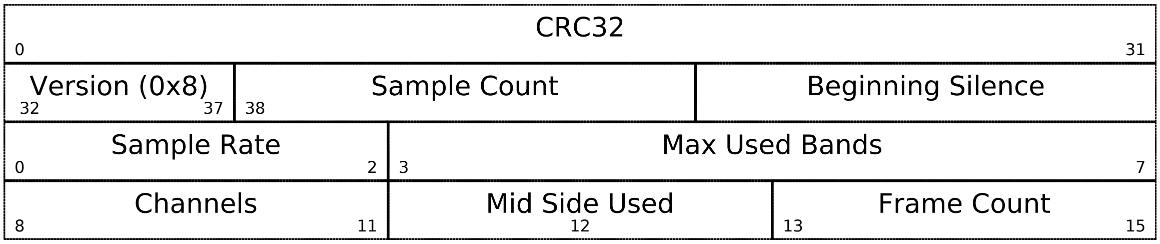


'Key' is a two character uppercase ASCII string (i.e. each digit must be between the characters 0x41 and 0x5A, inclusive). 'Length' is a variable length field indicating the size of the entire packet, including the header. This is a Nut-encoded field whose total size depends on whether the eighth bit of each byte is 0 or 1. The remaining seven bits of each byte combine to form the field's value, which is big-endian.



14.2.1 the SH packet

This is the Stream Header, which must be found before the first audio packet in the file.



'CRC32' is a checksum of everything in the header, not including the checksum itself. 'Sample Count' is the total number of samples, as a Nut-encoded value. 'Beginning Silence' is the number of silence samples at the start of the stream, also as a Nut-encoded value. ‘Channels’ is the total number of channels in the stream, minus 1. ‘Mid Side Used’ indicates the channels are stored using mid-side stereo. ‘Frame Count’ is used to calculate the total number of frames per audio packet:

$$\text{Number of Frames} = 4^{\text{Frame Count}} \tag{14.2}$$

‘Sample Rate’ is one of four values:
`000` = 44100Hz, `001` = 48000Hz, `010` = 37800Hz, `011` = 32000Hz .

14.2.2 the SE packet

This is an empty packet that denotes the end of the Musepack stream. A decoder should ignore everything after this packet, which allows for metadata tags such as APEv2 to be placed at the end of the file.

14.2.3 the RG packet

This is ReplayGain information about the file.

Version (0x1) 0 7	
Title Gain 8 23	Title Peak 24 39
Album Gain 40 55	Album Peak 56 71

14.2.4 the EI packet

This is information about the Musepack encoder.

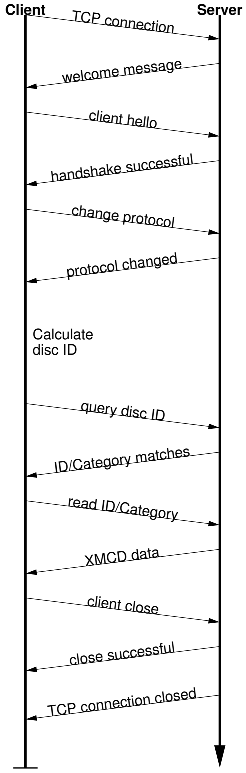
Profile 0 6	PNS 7	Major Version 8 15
Minor Version 16 23		Build 24 31

15 FreeDB

Because compact discs do not usually contain metadata about track names, album names and so forth, that information must be retrieved from an external source. FreeDB is a service which allows users to submit CD metadata and to retrieve the metadata submitted by others. Both actions require a category and a 32-bit disc ID number, which combine to form a unique identifier for a particular CD.

15.1 Native Protocol

FreeDB's native protocol runs as a service on TCP port 8880.



- After connecting, the client and server exchange a handshake using the `hello` command. The server will not do anything without this handshake.

- Next the client changes to protocol level 6 with the `proto` command. This is necessary because only the highest protocol supports UTF-8 text encoding. Without this, any characters not in the latin-1 set will not be sent properly.

- Once that is accomplished, the client should calculate the 32-bit disc ID from the track information.

- One then sends the 32-bit disc ID and additional disc information to the server with the `query` command to retrieve a list of matching disc IDs, genres and titles. If there are multiple matches, the user must be prompted to choose one of the matches.

- When our match is known, the client uses the `read` command to retrieve the actual XMCD data.

- Finally, the `close` command is used to sever the connection and complete the transaction.

15.1.1 the disc ID

FreeDB uses a big-endian 32-bit disc ID to differentiate on disc from another.

Offset Seconds Digit Sum	Total Length in Seconds	Track Count
0 … 7	8 … 23	24 … 31

'Track Count' is self-explanatory. 'Total Length' is the total length of all the tracks, not counting the initial 2 second lead-in. 'Offset Seconds Digit Sum' is the sum of the digits of all the disc's track offsets, in seconds, and truncated to 8 bits. Remember to count the initial 2 second/150 frame lead-in when calculating offsets.

Track	Length			Offset		
Number	in M:SS	in seconds	in frames	in M:SS	in seconds	in frames
1	3:37	217	16340	0:02	2	150
2	3:23	203	15294	3:39	219	16490
3	3:37	217	16340	7:03	423	31784
4	3:20	200	15045	10:41	641	48124

In this example, 'Track Count' is **4**. 'Total Length' is $\frac{16340+15294+16340+15045}{75} = \mathbf{840}$

There are 75 frames per second, and one must remember to count fractions of seconds when calculating the total disc length.

The 'Offset Seconds Digit Sum' is calculated by looking at the 'Offset in Seconds' column. Those values are 2, 219, 423 and 641. One must take all of those digits and add them, which works out to $2 + 2 + 1 + 9 + 4 + 2 + 3 + 6 + 4 + 1 = \mathbf{34}$

This means our three values are 34, 840 and 4. In hexadecimal, they are 0x22, 0x0348 and 0x04. Combining them into a single value yields 0x22034804. Thus, our FreeDB disc ID is 22034804

15.1.2 Initial greeting

From Server

```
<code>␣<host>␣CDDBP␣server␣<version>␣ready␣at␣<datetime>
```

	200	OK, reading/writing allowed
	201	OK, read-only
`<code>`	432	No connections allowed: permission denied
	433	No connections allowed: X users allowed, Y currently active
	434	No connections allowed: system load too high
`<hostname>`		the server's host name
`<version>`		the server's version
`<datetime>`		the current date and time

15.1.3 Client-server handshake

To Server

```
cddb␣hello␣<username>␣<hostname>␣<clientname>␣<version>
```

`<username>`	login name of user
`<hostname>`	host name of client
`<clientname>`	name of client program
`<version>`	version of client program

From Server

```
<code>␣hello␣and␣welcome␣<username>@<hostname>␣running␣<client>␣<version>
```

	200	handshake successful
`<code>`	402	already shook hands
	431	handshake unsuccessful, closing connection
`<username>`		login name of user
`<hostname>`		host name of client
`<clientname>`		name of client program
`<version>`		version of client program

15.1.4 Set protocol level

To Server

```
proto␣[level]
```

`[level]`	protocol level as integer (optional)

From Server

```
<code>␣CDDB␣protocol␣level:␣<current>,␣supported␣<supported>

OR

<code>␣OK,␣protocol␣version␣now:␣␣<current>
```

`<code>`	200	displaying current protocol level
	201	protocol level set
	501	illegal protocol level
	502	protocol level already at `<current>`
`<current>`		the current protocol level of this connection
`<supported>`		the maximum supported protocol level

15.1.5 Query database

To Server

```
cddb␣query␣<disc_id>␣<track_count>␣<offset_1>␣<...>␣<offset_n>␣<seconds>
```

`<disc_id>`	32-bit disc ID
`<track_count>`	number of tracks in CD
`<offset>`	frame offset of each track
`<seconds>`	total length of CD in seconds

From Server

```
<code>␣<category>␣<disc_id>␣<disc_title>

OR

<code>␣close␣matches␣found
<category>␣<disc_id>␣<disc_title>
<category>␣<disc_id>␣<disc_title>
<...>
.

OR

<code>␣exact␣matches␣found
<category>␣<disc_id>␣<disc_title>
<category>␣<disc_id>␣<disc_title>
<...>
.
```

`<code>`	200	Found exact match
	211	Found inexact matches, list follows
	202	No match found
	210	Found exact matches, list follows
	403	Database entry corrupt
	409	no handshake
`<category>`		category string
`<disc_id>`		32-bit disc ID
`<disc_title>`		disc title string

15.1.6 Read XMCD data

To Server

```
cddb␣read␣<category>␣<disc_id>
```

`<category>`	category string
`<disc_id>`	32-bit disc ID

From Server

```
<code>␣<category>␣<disc_id>
<XMCD_file_data>
<...>
.
```

	210	XMCD data follows
	401	XMCD data not found
`<code>`	402	server error
	403	database entry corrupt
	409	no handshake
`<category>`		category string
`<disc_id>`		32-bit disc ID

15.1.7 Close connection

To Server

```
quit
```

From Server

```
<code>␣<hostname>␣<message>
```

`<code>`	230	Closing connection. Goodbye.
	530	error, closing connection.
`<message>`		exit message
`<hostname>`		server's host name

15.2 Web protocol

FreeDB’s web protocol runs as a service on HTTP port 80. A web client POSTs data to a location, typically: `cddb/cddb.cgi` and retrieves results. This method is similar to the native protocol and the returned data is identical. However, since HTTP POST requests are stateless, there are no separate `hello`, `proto` and `quit` commands; these are issued along with the primary server command or are implied.

key	value
`hello`	`<username> <hostname> <clientname> <version>`
`proto`	`<protocol>`
`cmd`	`<command>`

Table 15.1: POST arguments

For example, to execute the `read` command on disc ID `AABBCCDD` in the `soundtrack` category, one can POST the following string:

```
cmd=read+soundtrack+aabbccdd&hello=username+hostname+audiotools+1.0&proto=6
```

15.3 XMCD

XMCD files are text files encoded either in UTF-8, ISO-8859-1 or US-ASCII. All begin with the string ‘`# XMCD`’. Lines are delimited by either the 0x0A character or the 0x0D 0x0A character pair. All lines must be less than 256 characters long, including delimiters. Blank lines are prohibited. Lines that begin with the ‘`#`’ character are comments. Curiously, the comments themselves are expected by FreeDB to contain important information such as track offsets and disc length. Fortunately, FreeDB clients can safely ignore such information unless submitting a new disc entry.

What we are interested in are the `KEY=value` pairs in the rest of the file.

key	value
`DISCID`	a comma-separated list of 32-bit disc IDs
`DTITLE`	an artist name and album name, separated by ‘ / ’
`DYEAR`	a 4 digit disc release year
`DGENRE`	the disc’s FreeDB category string
`TITLE`**X**	the track title, or the track artist name and track title, separated by ‘ / ’ **X** is an integer starting from 0
`EXTD`	extended data about the disc
`EXTT`**X**	extended data about the track **X** is an integer starting from 0
`PLAYORDER`	a comma-separated list of track numbers

Multiple identical keys should have their values concatenated (minus the newline delimiter), which allows a single key to have a value longer than the 256 characters line length.

16 MusicBrainz

MusicBrainz is another CD metadata retrieval service similar to FreeDB, but designed to eliminate many of FreeDB's limitations. For example, MusicBrainz has a more robust disc ID calculation mechanism, it has an easier way to disambiguate database entries in case of collision, and its XML metadata format is less prone to errors (track names with '/' characters are a particular problem for FreeDB).

However, because it is a newer service, it's common to find disc entries that are on FreeDB but do not yet have a MusicBrainz entry - whereas the converse is much more rare. Therefore, a metadata looking program would be wise to check both services if possible.

16.1 Searching releases

This is analagous to FreeDB's search routine in which one calculates a CD's disc ID, submits it to MusicBrainz via an HTTP get query and receives information such as album name, artist name, track names and so forth as an XML file.

16.1.1 the disc ID

Calculating a MusicBrainz disc ID requires knowing a CD's first track number, last track number, track offsets (in CD frames) and lead out track offset (also in CD frames). For example, given the following CD:

Track	Length			Offset		
Number	in M:SS	in seconds	in frames	in M:SS	in seconds	in frames
1	3:37	217	16340	0:02	2	150
2	3:23	203	15294	3:39	219	16490
3	3:37	217	16340	7:03	423	31784
4	3:20	200	15045	10:41	641	48124

The first track number is 1, the last track number is 4, the track offsets are 150, 16490, 31784 and 48124, and the lead out track offset is 63169 (track 4's offset 48124 plus its length of 15045).

These numbers are then converted to 0-padded, big-endian hexadecimal strings with the track numbers using 2 digits and the offsets using 8 digits. In this example, the first track number becomes 01, the last track number becomes 04, the track offsets become 00000096, 0000406A, 00007C28 and 0000BBFC, and the lead out track offset becomes 0000F6C1.

These individual strings are then combined into a single 804 byte string:

First Track Number	Last Track Number	Lead Out Offset	Offset$_1$	Offset$_2$	...	Offset$_{99}$
0 15	16 31	32 95	96 127	128 191		6368 6431

Excess track offsets are treated as having an offset value of 0, or a string value of 00000000. Our string starts with 01040000F6C1000000960000406A00007C280000BBFC and is padded with an additional 760 '0' characters which I'll omit for brevity.

That string is then passed through the SHA-1 hashing algorithm[1] which results in a 20 byte hash value. Remember to use the binary hash value, not its 40 byte ASCII hexadecimal one.

In our example, this yields the hash: 0xDA3D930462773DD57BBE43B535AD6A457138F079

The resulting hash value is then encoded to a 28 byte Base64[2] string. However, unlike standard Base64, MusicBrainz's disc ID replaces the characters '=', '+' and '/' with '-', '.' and '_' respectively to make the value better suited to HTTP requests. So to complete our example, the hash value becomes a disc ID of 2j2TBGJ3PdV7vkO1Na1qRXE48Hk-

[1] This is described in RFC3174

[2] This is described in RFC3548 and RFC4648

16.1.2 Server query

MusicBrainz runs as a service on HTTP port 80. To retrieve Release information, one can make a GET request to `/ws/1/release` using the following fields:

key	value
`type`	`xml`
`discid`	`<disc ID string>`

For example, to retrieve the Release data for disc ID `2jmj7l5rSw0yVb_vlWAYkK_YBwk-` one sends the GET query:

```
type=xml&discid=2jmj7l5rSw0yVb_vlWAYkK_YBwk-
```

Whether the Release is found in the MusicBrainz database or not, an XML file will always be generated.

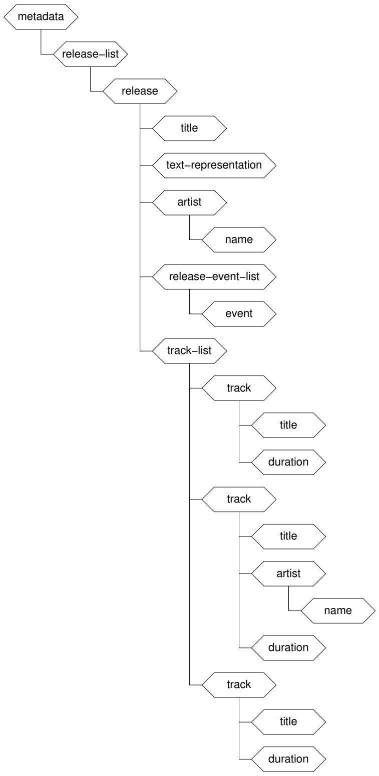
16.1.3 Release XML

All XML files returned by a MusicBrainz query consist of a `<metadata>` tag container. When making a Release query, it contains a `<release-list>` which is itself a container for zero or more `<release>` tags, depending on how many Release entries match the submitted disc ID.

The `<release>` tag typically contains a `<title>` which is the album's name, an `<artist>` tag which is the album's primary artist, a `<release-event-list>` tag containing information such as the album's release date and catalog number, and finally a `<track-list>` which contains all the track data.

The `<track>` tags are always listed in order of their appearance in the album. Each contains a `<title>` which is the track's name, a `<duration>` which is the track's length in milliseconds, and optionally an `<artist>` tag which is information about a track-specific artist, for instances where the track's artist differs from the album's artist.

In addition, the `<release>` , `<artist>` , and `<track>` tags all contain an 'id' attribute with 32 hex digits in the format '12345678-9abc-def1-2345-6789abcdef1234'. These uniquely identify the Release, Artist and Track information in the MusicBrainz database and can be used for direct lookups.



16.2 MusicBrainz XML

The following is the complete specification for MusicBrainz XML output in RELAX NG Compact syntax from http://bugs.musicbrainz.org/browser/mmd-schema/trunk/schema and converted to compact syntax for better readability.

```
Schema Start

default namespace id3034801 = "http://musicbrainz.org/ns/mmd-1.0#"

namespace local = ""

namespace inh = inherit

start = def_metadata-element

def_metadata-element =
   element metadata
   {
      attribute generator { xsd:anyURI }?,
      attribute created { xsd:dateTime }?,
      def_artist-element?,
      def_release-element?,
      def_release-group-element?,
      def_track-element?,
      def_label-element?,
      def_artist-list?,
      def_release-list?,
      def_release-group-list?,
      def_track-list?,
      def_label-list?,
      def_metadata-element_extension
   }

def_artist-element =
   element artist
   {
      attribute id { xsd:anyURI }?,
      attribute type { xsd:anyURI }?,
      def_artist-attribute_extension,
      element name { text }?,
      element sort-name { text }?,
      element disambiguation { text }?,
      element life-span
      {
         attribute begin { def_incomplete-date }?,
         attribute end { def_incomplete-date }?
      }?,
      def_alias-list?,
      def_release-list?,
      def_release-group-list?,
      def_relation-list*,
      def_tag-list?,
      def_user-tag-list?,
      def_rating?,
      def_user-rating?,
      def_artist-element_extension
   }
```

```
def_release-element =
   element release
   {
      attribute id { xsd:anyURI }?,
      attribute type { def_URI-list }?,
      def_release-attribute_extension,
      element title { text }?,
      element text-representation
      {
         attribute language { def_iso-639 }?,
         attribute script { def_iso-15924 }?
      }?,
      element asin { xsd:string { pattern = "[A-Z0-9]{10}" } }?,
      def_artist-element?,
      def_release-group-element?,
      def_release-event-list?,
      def_disc-list?,
      def_puid-list?,
      def_track-list?,
      def_relation-list*,
      def_tag-list?,
      def_user-tag-list?,
      def_rating?,
      def_user-rating?,
      def_release-element_extension
   }

def_release-group-element =
   element release-group
   {
      attribute id { xsd:anyURI }?,
      attribute type { def_URI-list }?,
      def_release-group-attribute_extension,
      element title { text }?,
      def_artist-element?,
      def_release-list?,
      def_release-group-element_extension
   }

def_track-element =
   element track
   {
      attribute id { xsd:anyURI }?,
      def_track-attribute_extension,
      element title { text }?,
      element duration { xsd:nonNegativeInteger }?,
      element isrc-list { element isrc { attribute id { def_isrc } }* }?,
      def_artist-element?,
      def_release-list?,
      def_puid-list?,
      def_relation-list*,
      def_tag-list?,
      def_user-tag-list?,
      def_rating?,
      def_user-rating?,
      def_track-element_extension
   }
```

```
def_label-element =
   element label
   {
      attribute id { xsd:anyURI }?,
      attribute type { xsd:anyURI }?,
      def_label-attribute_extension,
      element name { text }?,
      element sort-name { text }?,
      element label-code { xsd:nonNegativeInteger }?,
      element disambiguation { text }?,
      element country { def_iso-3166 }?,
      element life-span
      {
         attribute begin { def_incomplete-date }?,
         attribute end { def_incomplete-date }?
      }?,
      def_alias-list?,
      def_release-list?,
      def_release-group-list?,
      def_relation-list*,
      def_tag-list?,
      def_user-tag-list?,
      def_rating?,
      def_user-rating?,
      def_label-element_extension
   }

def_relation-element =
   element relation
   {
      attribute type { xsd:anyURI },
      attribute target { xsd:anyURI },
      attribute direction { def_direction }?,
      attribute attributes { def_URI-list }?,
      attribute begin { def_incomplete-date }?,
      attribute end { def_incomplete-date }?,
      (
         def_artist-element
       | def_release-element
       | def_track-element
       | def_relation-element_extension
      )?
   }

def_alias =
   element alias
   {
      attribute type { xsd:anyURI }?,
      attribute script { def_iso-15924 }?,
      text
   }

def_tag = element tag { attribute count { xsd:nonNegativeInteger }?, text }

def_user-tag = element user-tag { text }

def_rating =
   element rating
   {
      attribute votes-count { xsd:nonNegativeInteger }?,
      xsd:float
   }

def_user-rating = element user-rating { xsd:nonNegativeInteger }

def_metadata-element_extension = def_extension_element?

def_artist-element_extension = def_extension_element*

def_release-element_extension = def_extension_element*

def_release-group-element_extension = def_extension_element*

def_track-element_extension = def_extension_element*

def_label-element_extension = def_extension_element*

def_relation-element_extension = def_extension_element

def_artist-attribute_extension = def_extension_attribute*

def_release-attribute_extension = def_extension_attribute*

def_release-group-attribute_extension = def_extension_attribute*

def_track-attribute_extension = def_extension_attribute*

def_label-attribute_extension = def_extension_attribute*

def_extension_element =
   element * - (id3034801:* | local:*)
   {
      ( attribute * { text } | text | def_anything )*
   }

def_extension_attribute = attribute * - (id3034801:* | local:*) { text }

def_anything =
   element * - local:* { ( attribute * { text } | text | def_anything )* }

def_artist-list =
   element artist-list { def_list-attributes, def_artist-element* }

def_release-list =
   element release-list { def_list-attributes, def_release-element* }

def_release-group-list =
   element release-group-list
   {
      def_list-attributes,
      def_release-group-element*
   }

def_alias-list = element alias-list { def_list-attributes, def_alias* }

def_track-list = element track-list { def_list-attributes, def_track-element* }

def_label-list = element label-list { def_list-attributes, def_label-element* }
```

```
def_release-event-list =
   element release-event-list
   {
      def_list-attributes,
      element event
      {
         attribute date { def_incomplete-date },
         attribute country { def_iso-3166 }?,
         attribute catalog-number { text }?,
         attribute barcode { text }?,
         attribute format { xsd:anyURI }?,
         def_label-element?
      }*
   }

def_disc-list =
   element disc-list
   {
      def_list-attributes,
      element disc
      {
         attribute id { xsd:string { pattern = "[a-zA-Z0-9._]{27}-" } },
         attribute sectors { xsd:nonNegativeInteger }?
      }*
   }

def_puid-list =
   element puid-list
   {
      def_list-attributes,
      element puid { attribute id { def_uuid } }*
   }

def_relation-list =
   element relation-list
   {
      attribute target-type { xsd:anyURI },
      def_list-attributes,
      def_relation-element*
   }

def_tag-list = element tag-list { def_list-attributes, def_tag* }

def_user-tag-list =
   element user-tag-list { def_list-attributes, def_user-tag* }

def_list-attributes =
   attribute count { xsd:nonNegativeInteger }?,
   attribute offset { xsd:nonNegativeInteger }?

def_URI-list = list { xsd:anyURI+ }

def_incomplete-date =
   xsd:string { pattern = "[0-9]{4}(-[0-9]{2})?(-[0-9]{2})?" }

def_iso-3166 = xsd:string { pattern = "[A-Z]{2}" }

def_iso-639 = xsd:string { pattern = "[A-Z]{3}" }
```

```
def_iso-15924 = xsd:string { pattern = "[A-Z][a-z]{3}" }

def_isrc = xsd:string { pattern = "[A-Z]{2}[A-Z0-9]{3}[0-9]{2}[0-9]{5}" }

def_uuid = xsd:string { pattern = "[0-9a-f]{8}(-[0-9a-f]{4}){3}-[0-9a-f]{12}" }

def_direction = "both" | "forward" | "backward"
```

Schema End

17 ReplayGain

The ReplayGain standard is designed to address the problem of highly variable music loudness. For example, let's assume we have two audio tracks, A and B, and that track B is much louder than A. If played in sequence, the listener will have to scramble for the volume control once B starts in order to have a comfortable experience. ReplayGain solves this problem by calculating the overall loudness of a track as a delta (some positive or negative number of decibels, in relation to a reference loudness value). This delta is then applied during playback, which has the same effect as turning the volume up or down so that the user doesn't have to.

ReplayGain requires four floating-point values which are typically stored as metadata in each audio track: 'track gain', a positive or negative number of decibels representing the loudness delta of this particular track, 'track peak', the highest sample value of this particular track from a range of 0.0 to 1.0, 'album gain', a positive or negative number of decibels representing the loudness delta of the track's entire album and 'album peak', the highest sample value of the track's entire album from a range of 0.0 to 1.0.

17.1 Applying ReplayGain

The user will be expected to choose whether to apply 'album gain' or 'track gain' during playback. When listening to audio on an album-by-album basis, album gain keeps quiet tracks quiet and loud tracks loud within the context of that album. When listening to audio on a track-by-track basis, perhaps as a randomly shuffled set, track gain keeps them all to roughly the same loudness. So from an implementation perspective, a program only needs to apply the given gain and peak value to the stream being played back. Applying the gain value to each input PCM sample is quite simple:

$$\text{Output}_i = \text{Input}_i \times 10^{\frac{\text{gain}}{20}} \tag{17.1}$$

For example, if the gain is -2.19, each sample should be multiplied by $10^{\frac{-2.19}{20}}$ or about 0.777.

If the gain is negative, the PCM stream gets quieter than it was originally. If the gain is positive, the PCM stream gets louder. However, increasing the value of each sample may cause a problem if doing so sends any samples beyond the maximum value the stream can hold. For example, if the gain indicates we should be multiplying each sample by 1.28 and we encounter a 16-bit input sample with a value of 32000, the resulting output sample of 34560 is outside of the stream's 16-bit signed range (-32678 to 32767). That will result in 'clipping' the audio peaks, which doesn't sound good.

Preventing this is what ReplayGain's peak value is for; it's the highest PCM value in the stream and no multiplier should push that value beyond 1.0. Thus, if the peak value of a stream is 0.9765625, no ReplayGain value should generate a multiplier higher than 1.024 ($0.9765625 \times 1.024 = 1.0$).

17.2 Calculating ReplayGain

As explained earlier, ReplayGain requires a peak and gain value which are split into 'track' and 'album' varieties for a total of four. The 'track' values require the PCM data for the particular track we're generating data for. The 'album' values require the PCM data for the entire album, concatenated together into a single stream.

Determining the peak value is very straightforward. We simply convert each sample's value to the range of 0.0 to 1.0 and find the highest value which occurs in the stream. For signed samples, the conversion process is also simple:

$$\text{Output}_i = \frac{|\text{Input}_i|}{2^{\text{bits per sample}-1}} \tag{17.2}$$

Determining the gain value is a more complicated process. It involves running the input stream through an equal loudness filter, breaking that stream into 50 millisecond long blocks, and then determining a final value based on the value of those blocks.

17.2.1 the equal loudness filter

Because people don't perceive all frequencies of sounds as having equal loudness, ReplayGain runs audio through a filter which emphasizes ones we hear as loud and deemphasizes ones we hear as quiet. This equal loudness filtering is actually comprised of two separate filters: Yule and Butter (these are Infinite Impulse Response filters named after their creators). Each works on a similar principle.

The basic premise is that each output sample is derived from multiplying 'order' number of previous input samples by certain values (which depend on the filter) *and* 'order' number of previous output samples by a different set of values (also depending on the filter) and then combining the results. This filter is applied independently to each channel. In purely mathematical terms, it looks like this:

$$\text{Output}_i = \left(\sum_{j=i-order}^{i} \text{Input}_j \times \text{Input Filter}_j\right) - \left(\sum_{k=i-order}^{i-1} \text{Output}_k \times \text{Output Filter}_k\right) \tag{17.3}$$

'Input Filter' and 'Output Filter' are lists of predefined values. 'Order' refers to the size of those lists. When filtering at the start of the stream, treat any samples before the beginning as 0.

a filtering example

Let's assume we have a 44100Hz stream and our previous input and output samples are as follows:

sample	Input$_i$	Yule$_i$	Butter$_i$
89	-33	-14.90	
90	-32	-14.93	
91	-35	-14.65	
92	-32	-14.46	
93	-30	-14.15	
94	-32	-13.58	
95	-33	-13.18	
96	-30	-13.16	
97	-30	-13.12	0.41
98	-30	-12.89	0.61
99	-32	-12.81	0.66

If the value of sample 100 from the input stream is -30, here's how we calculate output sample 100:

sample	Input$_i$		Yule Input Filter$_i$		result	Yule$_i$		Yule Output Filter$_i$		result
90	-32	×	-0.00187763777362	=	0.06	-14.93	×	0.13149317958807999	=	-1.96
91	-35	×	0.0067461368224699999	=	-0.24	-14.65	×	-0.75104302451432003	=	11.00
92	-32	×	-0.0024087905158400001	=	0.08	-14.46	×	2.1961168489077401	=	-31.76
93	-30	×	0.016248649629749999	=	-0.49	-14.15	×	-4.3947099607955904	=	62.19
94	-32	×	-0.025963385129149998	=	0.83	-13.58	×	6.8540154093699801	=	-93.08
95	-33	×	0.022452932533390001	=	-0.74	-13.18	×	-8.8149868137015499	=	116.18
96	-30	×	-0.0083499090493599996	=	0.25	-13.16	×	9.4769360780128	=	-124.72
97	-30	×	-0.0085116564546900003	=	0.26	-13.12	×	-8.5475152747187408	=	112.14
98	-30	×	-0.0084870937985100006	=	0.25	-12.89	×	6.3631777756614802	=	-82.02
99	-32	×	-0.029110078089480001	=	0.93	-12.81	×	-3.4784594855007098	=	44.56
100	**-30**	×	0.054186564064300002	=	-1.63					
			input values sum	=	-0.44			output values sum	=	12.53

Therefore, $\text{Yule}_{100} = -0.44 - 12.53 = -12.97$

We're not quite done yet. Remember, ReplayGain's equal loudness filter requires both a Yule *and* Butter filter, in that order. Notice how Butter's input samples are Yule's output samples. Thus, our next input sample to the Butter filter is -12.97. Calculating sample 100 is now a similar process:

sample	Yule$_i$		Butter Input Filter$_i$		result	Butter$_i$		Butter Output Filter$_i$		result
98	-12.89	×	0.98500175787241995	=	-12.70	0.61	×	0.97022847566350001	=	0.59
99	-12.81	×	-1.9700035157448399	=	25.24	0.66	×	-1.96977855582618	=	-1.30
100	**-12.97**	×	0.98500175787241995	=	-12.78					
			input values sum	=	-0.24			output values sum	=	-0.71

Therefore, $\text{Butter}_{100} = -0.24 - -0.71 = 0.47$, which is the next sample from the equal loudness filter.

17.2.2 RMS energy blocks

The next step is to take our stream of filtered samples and convert them to a list of blocks, each 1/20th of a second long. For example, a 44100Hz stream is sliced into blocks containing 2205 PCM frames each.

We then figure out the total energy value of each block by taking the Root Mean Square of the block's samples and converting to decibels, hence the name RMS.

$$\text{Block dB}_i = 10 \times \log_{10}\left(\frac{\left(\frac{\sum\limits_{x=0}^{\text{Block Length}-1} \text{Left Sample}_x{}^2}{\text{Block Length}}\right) + \left(\frac{\sum\limits_{y=0}^{\text{Block Length}-1} \text{Right Sample}_y{}^2}{\text{Block Length}}\right)}{2} + 10^{-10}\right) \tag{17.4}$$

For mono streams, use the same value for both the left and right samples (this will cause the addition and dividing by 2 to cancel each other out). As a partial example involving 2205 PCM frames:

Sample	Left Value	Left Value^2	Right Value	Right Value^2
998	115	13225	-43	1849
999	111	12321	-38	1444
1000	107	11449	-36	1296
...	...		...	
	Left Value^2 sum = 7106715		Right Value^2 sum = 11642400	

$$\frac{(\frac{7106715}{2205}) + (\frac{11642400}{2205})}{2} = 4251 \tag{17.5}$$

$$10 \times \log_{10}(4251 + 10^{-10}) = 36.28 \tag{17.6}$$

Thus, the decibel value of this block is 36.28.

17.2.3 Statistical processing and calibration

At this point, we've converted our stream of input samples into a list of RMS energy blocks. We now pick the 95th percentile value as the audio stream's representative value. That means we first sort them from lowest to highest, then pick the one at the 95% position. For example, if we have a total of 2400 decibel blocks (from a 2 minute song), the value of block 2280 is our representative.

Finally, we take the difference between a reference value of pink noise and our representative value for the final gain value. The reference pink noise value is typically 64.82 dB. Therefore, if our representative value is 67.01 dB, the resulting gain value is -2.19 dB $(64.82 - 67.01 = -2.19)$.

Appendices

A References

- Wave File Format Specifications
  `http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html`

- Audio File Format Specifications
  `http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/AIFF/AIFF.html`

- AU Audio File Format
  `http://www.opengroup.org/public/pubs/external/auformat.html`

- FLAC Format Specification
  `http://flac.sourceforge.net/format.html`

- APEv2 Specification
  `http://wiki.hydrogenaudio.org/index.php?title=APEv2_specification`

- WavPack 4.0 File / Block Format
  `http://www.wavpack.com/file_format.txt`

- MPEG Audio Compression Basics
  `http://www.datavoyage.com/mpgscript/mpeghdr.htm`

- What is ID3v1
  `http://www.id3.org/ID3v1`

- The ID3v2 Documents
  `http://www.id3.org/Developer_Information`

- The Ogg File Format
  `http://en.wikipedia.org/wiki/Ogg#File_format`

- Vorbis I Specification
  `http://xiph.org/vorbis/doc/Vorbis_I_spec.html`

- Proposals for Extending Ogg Vorbis Comments
  `http://www.reallylongword.org/articles/vorbiscomment/`

- Speex Documentation
  `http://www.speex.org/docs/`

- Musepack Stream Version 7 Format Specification
  http://trac.musepack.net/trac/wiki/SV7Specification
- Parsing and Writing QuickTime Files in Java
  http://www.onjava.com/pub/a/onjava/2003/02/19/qt_file_format.html
- ISO 14496-1 Media Format
  http://xhelmboyx.tripod.com/formats/mp4-layout.txt
- FreeDB Information
  http://www.freedb.org/en/download__miscellaneous.11.html
- ReplayGain
  http://replaygain.hydrogenaudio.org

B License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE (”CCPL” OR ”LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

B.1 Definitions

1. **”Adaptation”** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image (”synching”) will be considered an Adaptation for the purpose of this License.

2. **”Collection”** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which

together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

3. **"Creative Commons Compatible License"** means a license that is listed at `http://creativecommons.org/compatiblelicenses` that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.

4. **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

5. **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

6. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

7. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

8. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent

it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

9. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

10. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

11. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

B.2 Fair Dealing Rights.

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

B.3 License Grant.

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

1. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

2. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

3. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

4. to Distribute and Publicly Perform Adaptations.

5. For the avoidance of doubt:

   a) **Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
   b) **Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
   c) **Voluntary License Schemes**. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

B.4 Restrictions.

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License.

If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.

2. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

3. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation

(e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

4. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

B.5 Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

B.6 Limitation on Liability.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B.7 Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

B.8 Miscellaneous

1. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

2. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

6. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.