inside MP3 inside

MP3 file structure Constant bitrate Variable bitrate TAG v1 TAG v2

MP3 File Structure

can be expressed in this scheme (TAGs are optional):

[TAG v2] Frame1 Frame2 Frame3... [TAG v1]

MP3 file is divided into a small blocks - frames. Each frame has constant time length 0.026 sec. But size of one frame (in Bytes) varies according to bitrate. Eg. for 128kbps it is (normally) 417 Bytes and for 192kbps 626 Bytes.

The first 4 Bytes of each frame is **frame header** and the rest is audio data.

Frame header consists of information about frame (bitrate, stereo mode...) and because of that frames are independent items. Each of them can have its own characteristic. It is used eg. in Variable Bitate files, where each frame can have different bitrate.

Frame header has this structure (each letter is one bit):

AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM

A **Frame synchronizer**
All bits are set to 1. It is used for finding the beginning of frame. But these values can occur many times in binary file so you should test next values from header for validity (eg. bitrate bits arent 1111, sampling rate frequency isnt 11 etc.). But you can never be 100% sure if you find a header. Next method is to find the first header and then go through all frames - almost exact, but time consuming.
Be careful with the first frame! It doesnt have to start at the first Byte in file. Either TAG v2 can be included or file can contains of some crap at the beginning.
Anyway - to find a header is a little problem.

B **MPEG version ID**

00	MPEG Version 2.5 (not an official standard)
01	reserved
10	MPEG Version 2
11	MPEG Version 1

In most MP3 files these value should be 11.

C **Layer**

00	reserved
01	Layer III

10 Layer II
11 Layer I

In most MP3 files these value should be 01 (because MP3 = MPEG 1 Layer 3).

D **CRC Protection**

0 Protected by CRC
1 Not protected

Frames may have some form of check sum - CRC check. CRC is 16 bit long and, if exists, it follows frame header. And then comes audio data.
But almost all MP3 files doesnt contain CRC.

E **Bitrate index**

0000	free
0001	32
0010	40
0011	48
0100	56
0101	64
0110	80
0111	96
1000	112
1001	128
1010	160
1011	192
1100	224
1101	256
1110	320
1111	bad

All values are in kbps.

F **Samplig rate frequency index**

00 44100
01 48000
10 32000
11 reserved

All values are in Hz. In most MP3 files these value should be 00.
Note: Sample frequency 44100 means that one second of audio information is hacked to 44100 pieces. And each 1/44100 sec. is audio value taken and encoded into digital form.

G **Padding**

0 Frame is not padded
1 Frame is padded

Padding is used to fit the bitrates exactly. If you use frames with length 417 Bytes (for 128kbps) it doesnt give exact data flow 128kbps. So you can set Padding and add one extra Byte at the end of some frames to create exact 128kbps.
Fuck it.

H **Private bit**

It can be freely used for specific needs of an application, eg. it can execute some application specific events.
No special meaning, forget it.

I **Channel**

00	Stereo	Similar to Dual mono, 2 channels, but bitrate can be different for each one and is coded dynamically. Eg. if channel 1 is silent, the second one will get higher bitrate.

01	Joint Stereo	Mostly used in MP3. One channel is common (mid) and is used mainly for common and lower tones. The second is (side) channel for encoding differences between normal channels. Note: Stereo effect is listenable properly only for higher tones because for lower ones is length of sound wave so long that you are not able to distinguish phase move.
10	Dual	Also known as Dual mono; 2 separate channels.
11	Mono (single channel	Normal mono.

J **Mode extension** (only if Joint Stereo is set)

	Intensity Stereo	MS Stereo
00	off	off
01	on	off
10	off	on
11	on	on

Tells which mode for JointStereo is used.

K **Copyright**

0 Audio is not copyrighted

1 Audio is copyrighted

No special use.

L **Original**

0 Copy of original media

1 Original media

No special use.

M **Emphasis**

00 None

01 50/15

10 reserved

11 CCIT J.17

Tells if there are emphasised frequencies above cca. 3.2 kHz.

Formula for counting frame length in Bytes:

FrameLen = int((144 * BitRate / SampleRate) + Padding);

Eg. for Bitrate = 192kbps, SampleRate = 44.1kHz a Padding = Yes
FrameLen = int((144 * 192000 / 44100) + 1) = 627 Bytes
int() means round to bottom. FrameLen includes frame header.

Constant Bitrate (CBR)

Here is no problema. All frames should be the same (except of audio data).
But only SHOULD. God damn if they vary in Padding or Emphasis. But I met CBR files where all frame headers were the same and frame length should be 626 Bytes. But some frames were 625, 627, 628...
So be careful, in MP3 files you cant rely on nothing. But in most cases frame length is correct.

Variable Bitrate (VBR)

This system was created to minimize file lengths and to preserve sound quality.
Higher frequencies generally needs more space for encoding (thats why many codecs cut all frequencies above cca 16kHz) and lower tones requires less. So if some part of song doesnt consist of higher tones then using eg. 192kbps is wasting of space. It should be enough to use only eg. 96kbps.
And it is the principle of VBR. Codec looks over frame and then choose bitrate suitable for its sound quality.

It sounds perfect but it brings some problems:
If you want to jump over 2 minutes in song, it is not a problem with CBR because you are able simply count amount of Bytes which is necessary to skip. But it is impossible with VBR. Frame lengths should be arbitrary so you have to either go frame by frame and counts (time consuming and very unpractical) or use another mechanism for approximate count.
If you want to cut 5 minutes from the middle of VBR file (all we know CDs where last song takes 10 minutes but 5 minutes is a pure silence, HELL!) problems are the same.
Result? VBR files are more difficult for controlling and adjusting. And I dont like feeling that sound quality changes in every moment. And AFAIK many codecs have problems with creation VBR in good quality.
Personally I cant see any reason why to use VBR - I dont give a fuck if size of one CD in MP3 is 55 MB with CBR or 51 MB with VBR. But everybody has a different taste... some people prefer VBR.

VBR File Structure

is the same as for CBR. But the first frame doesnt contain audio data and it is used for special information about VBR file.

Structure of the first frame:

Byte	Content
0-3	Standard audio frame header (as descripted above). Mostly it contains values FF FB 30 4C, from which you can count FrameLen = 156 Bytes. And thats exactly enough space for storing VBR info. This header contains some important information valid for the whole file: - MPEG (MPEG1 or MPEG2) - SAMPLING rate frequency index - CHANNEL (JointStereo etc.)
4-x	Not used till string "Xing" (58 69 6E 67). This string is used as a main VBR file identifier. If it is not found, file is supposed to be CBR. This string can be placed at different locations according to values of MPEG and CHANNEL (ya, these from a few lines upwards):
36-39	**"Xing" for MPEG1 and CHANNEL != mono (mostly used)**
21-24	"Xing" for MPEG1 and CHANNEL == mono
21-24	"Xing" for MPEG2 and CHANNEL != mono
13-16	"Xing" for MPEG2 and CHANNEL == mono

After "Xing" string there are placed flags, number of frames in file and a size of file in Bytes. Each of these items has 4 Bytes and it is stored as 'int' number in memory. The first is the most significant Byte and the last is the least.

Following schema is for MPEG1 and CHANNEL != mono:

40-43 **Flags**

Value	Name	Description
00 00 00 01	Frames Flag	set if value for number of frames in file is stored
00 00 00 02	Bytes Flag	set if value for filesize in Bytes is stored
00 00 00 04	TOC Flag	set if values for TOC (see below) are stored
00 00 00 08	VBR Scale Flag	set if values for VBR scale are stored

All these values can be stored simultaneously.

44-47 **Frames**
Number of frames in file (including the first info one)

48-51 **Bytes**
File length in Bytes

52-151 **TOC (Table of Contents)**
Contains of 100 indexes (one Byte length) for easier lookup in file. Approximately solves problem with moving inside file.
Each Byte has a value according this formula:
(TOC[i] / 256) * fileLenInBytes
So if song lasts eg. 240 sec. and you want to jump to 60. sec. (and file is 5 000 000 Bytes length) you can use:
TOC[(60/240)*100] = TOC[25]
and corresponding Byte in file is then **approximately** at:
(TOC[25]/256) * 5000000

If you want to trim VBR file you should also reconstruct Frames, Bytes and TOC properly.

152-155 **VBR Scale**
I dont know exactly system of storing of this values but this item probably doesnt have deeper meaning.

Complicated? Ya, that is.

TAGs

TAG is name for data space in MP3 file where some text informations (song name, artist, album...) can be stored.

TAG ver.1

is old and simple. It takes **always** 128 Bytes at the very end of file (after the last audio frame).

Structure:

Bytes	Length	Content
0-2	3	Tag identifier. Must contain "TAG" string if Tag is valid.
3-32	30	Song Name
33-62	30	Artist
63-92	30	Album
93-96	4	Year
97-126	30	Comment
127	1	Genre

126. Byte can also be used as the number of song. Items should be padded with NULL (ASCII 0) or with a space (ASCII 32).

Values for Genre are predefined, you can find them eg. in WinAmp. Only the most important ones:

22	Death Metal
138	Black Metal
144	Thrash Metal
9	Metal
43	Punk
129	Hardcore

Problems with TAG v1 are obvious: you can store only a few items (you have a bad luck for eg.

country of an artist, used codec, author of lyric...) and there is only a limited space for items (you also have a bad luck if song name has more than 30 chars).
But despite that is TAG v1 commonly used by most of "MP3 release" groups.

TAG ver.2

is newer, bigger and uncut. And... much more complicated.
Is placed at the very beginning of file (before all audio frames).

Structure:

TAG v2 contains of header and frames (dont mismatch it with audio frames and their headers!!!).

Header (10 Bytes)

Bytes	Content
0-2	TAG identifier. It contains of string "ID3"
3-4	TAG version. Can be eg. 03 00
5	Flags
6-9	Size of TAG

Flags Byte has this structure (in bits):
abc00000
where the first 3 bits indicate Unsynchronization, Extended header and Experimental indicator.
Flags normally dont have special meaning, can be set to 00.

Size of TAG is encoded into 4 Bytes. But not to be so easy, the most significant bit in each Byte is set to 0 and ignored. Only remaining 7 bits are used. The reason is to avoid mismatch with audio frame header which has the first synchro Byte FF).
Eg. TAG len 257 is encoded as 00 00 02 01.
Size of TAG doesnt contain header itself so total lenght of previous TAG is 257 + 10 Bytes.

Frames

Each frame is used for storing one information - eg. Artist or Album.
Frame consists of header and body.

Header(10 Bytes)

Bytes	Content
0-3	Frame identifier
4-7	Size
8-9	Flags

Frame identifier consist of four characters. There are many predefined values you can use. But eg. current version of WinAmp displays only these ones:

Iden.	Description
TRCK	Track number
TENC	Encoded By
WXXX	URL
TCOP	Frame identifier
TOPE	Original Artist
TCOM	Composer
TCON	Genre
COMM	Comments
TYER	Year

TALB	Album
TPE1	Artist
TIT2	Song name

You can freely define you own frame, eg. with identifier "CUNT" but you cant expect thet some player will be able to diaplay that. Anyway you can store any information into MP3 file without limits.

Size is stored from the most significant Byte to the least. It doesnt include frame header, so total length of frame is Size + 10. Warning: After the header always one Byte with value 00 follows and then begins frame body. Size has to include this Byte.

Flags in most cases are set to 00 00. But they have this structure (in bits):

abc00000 ijk00000 //why they are not stored in one Byte???

Flag	Description
a	TAG alter preservation
b	File alter preservation
c	Read only
i	Compression
j	Encryption
k	Grouping identity

Example of frame:

54	50	45	31	00	00	00	07	00	00	00	53	53	6C	61	79	65	72
T	P	E	1				7				S	l	a	y	e	r	

To be even more complicated, some frames can have special structure. Eg. COMM (comments) aslo contains language version which is not normally displayed:

43	4F	4D	4D	00	00	00	0C	00	00	00	65	6E	67	00	63	6F	6D	6D	65	6E	74
C	O	M	M				0C				e	n	g		c	o	m	m	e	n	t

For TCON (genre or style) you can use predefined values as in TAG v1. Then frame body looks like "(144)Thrash Metal" and the number corresponds with TAG v1. Or you can simply write your own style and then body is "Brutal Black" and is stored like normal frame body.

Complete description of TAG v2 you can find here.

Home

* Kurzy * Last minute Zájezdy * Letenky * Kurzy akcie a burza * Reality a bydlení * Akcie * Rejsřík * Prague hotels *