This is a brief and informal document targeted to those who want to deal with the MPEG format. If you are one of them, you probably already know what is MPEG audio. If not, jump to http://www.mp3.com/ or http://www.layer3.org/ where you will find more details and also more links.

NOTE: You cannot just search the Internet and find the MPEG audio specs. It is copyrighted and you will have to pay quite a bit to get the Paper. That's why I made this. Informations I got are gathered from the internet, and mostly originate from sources I found available. Despite my custom to always specify the sources, I am not able to do it this time. Sorry, I did not maintain the list. :(

**This is not decoding specs, it just informs you how to read the MPEG headers and the MPEG TAG. MPEG Version 1, 2 and 2.5 and Layer I, II and III are supported, the MP3 TAG (MP3v1 and MP3v1.1) also.**. Those of you who use Delphi may find my MPGTools Delphi unit useful, it is where I implemented this stuff.

# MPEG Audio Frame Header

An MPEG audio file is separated in smaller parts called frames. Each frame is independent. It has its own header and audio informations. There is no file header. Therefore, you can cut any part of MPEG file and play it correctly.

When you want to read info about an MPEG file, it is usually enough to find the first frame, read its header and assume that the other frames are the same (which may not be always the case).

The frame header is constituated by the very first four bytes (32bits) in a frame. The first eleven bits of a frame header are always set and they are called "frame sync". Therefore, you can search through the file for the first occurence of eleven bits set (meaning that you have to find a byte with a value of 255, and followed by a byte with its three most significant bits set). Then you read the whole header and check if the values are correct. You will see in next table the exact meaning of each bits in the header, and which values may be checked for validity. Each value that is specified as reserved, invalid, bad, or not allowed should indicate an invalid header.

Frames may have a CRC check, but it's pretty rare. The CRC is 16 bits long and, if it exists, it follows the frame header. After the CRC comes the audio data. You may calculate the length of the frame and use it if you need to read other headers too or just want to calculate the CRC of the frame, to compare it with the one you read from the file. This is actually a very good method to check the MPEG header validity.

Here is "graphical" presentation of the header content. The letters are used to indicate the different fields. In the table, you can see the details about the content of each field.

**AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM**

| Sign | Length (bits) | Position (bits) | Description |
|---|---|---|---|
| A | 11 | (31-21) | Frame sync (all bits set) |
| B | 2 | (20,19) | MPEG Audio version<br>00 - MPEG Version 2.5<br>01 - reserved<br>10 - MPEG Version 2<br>11 - MPEG Version 1 |
| C | 2 | (18,17) | Layer description<br>00 - reserved<br>01 - Layer III<br>10 - Layer II<br>11 - Layer I |
| D | 1 | (16) | Protection bit<br>0 - Protected by CRC (16bit crc follows header)<br>1 - Not protected |
| E | 4 | (15,12) | Bitrate index |

| bits | V1,L1 | V1,L2 | V1,L3 | V2,L1 | V2,L2 | V2,L3 |
|---|---|---|---|---|---|---|
| 0000 | free | free | free | free | free | free |
| 0001 | 32 | 32 | 32 | 32 | 32 | 8 (8) |
| 0010 | 64 | 48 | 40 | 64 | 48 | 16 (16) |
| 0011 | 96 | 56 | 48 | 96 | 56 | 24 (24) |
| 0100 | 128 | 64 | 56 | 128 | 64 | 32 (32) |
| 0101 | 160 | 80 | 64 | 160 | 80 | 64 (40) |
| 0110 | 192 | 96 | 80 | 192 | 96 | 80 (48) |
| 0111 | 224 | 112 | 96 | 224 | 112 | 56 (56) |
| 1000 | 256 | 128 | 112 | 256 | 128 | 64 (64) |
| 1001 | 288 | 160 | 128 | 288 | 160 | 128 (80) |
| 1010 | 320 | 192 | 160 | 320 | 192 | 160 (96) |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1011 | 352 | 224 | 192 | 352 | 224 | 112 (112) |
| 1100 | 384 | 256 | 224 | 384 | 256 | 128 (128) |
| 1101 | 416 | 320 | 256 | 416 | 320 | 256 (144) |
| 1110 | 448 | 384 | 320 | 448 | 384 | 320 (160) |
| 1111 | bad | bad | bad | bad | bad | bad |

NOTES: All values are in kbps
V1 - MPEG Version 1
V2 - MPEG Version 2 and Version 2.5
L1 - Layer I
L2 - Layer II
L3 - Layer III
"free" means variable bitrate.
"bad" means that this is not an allowed value

The values in parentheses are from different sources which claim that those values are valid for V2,L2 and V2,L3. If anyone can confirm please let me know.

F 2 (11,10) Sampling rate frequency index (values are in Hz)

| bits | MPEG1 | MPEG2 | MPEG2.5 |
|---|---|---|---|
| 00 | 44100 | 22050 | 11025 |
| 01 | 48000 | 24000 | 12000 |
| 10 | 32000 | 16000 | 8000 |
| 11 | reserv. | reserv. | reserv. |

G 1 (9) Padding bit
0 - frame is not padded
1 - frame is padded with one extra bit

H 1 (8) Private bit (unknown purpose)

I 2 (7,6) Channel Mode
00 - Stereo
01 - Joint stereo (Stereo)
10 - Dual channel (Stereo)
11 - Single channel (Mono)

J 2 (5,4) Mode extension (Only if Joint stereo)

| value | Intensity stereo | MS stereo |
|---|---|---|
| 00 | off | off |
| 01 | on | off |
| 10 | off | on |
| 11 | on | on |

K 1 (3) Copyright
0 - Audio is not copyrighted
1 - Audio is copyrighted

L 1 (2) Original
0 - Copy of original media
1 - Original media

M 2 (1,0) Emphasis
00 - none
01 - 50/15 ms
10 - reserved
11 - CCIT J.17

**How to calculate frame size**

Read the BitRate, SampleRate and Padding (as value of one or zero) of the frame header and use the formula:

FrameSize = 144 * BitRate / SampleRate + Padding

Example: BitRate = 128000, SampleRate=441000, Padding=0 ==> FrameSize=417 bytes

# MPEG Audio Tag MP3v1

The TAG is used to describe the MPEG Audio file. It contains information about artist, title, album, publishing year and genre. There is some extra space

for comments. It is exactly 128 bytes long and is located at very end of the audio data. You can get it by reading the last 128 bytes of the MPEG audio file.

```
AAABBBBB BBBBBBBB BBBBBBBB BBBBBBBB
BCCCCCCC CCCCCCCC CCCCCCCC CCCCCCCD
DDDDDDDD DDDDDDDD DDDDDDDD DDDDEEE
EFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFG
```

| Sign | Length (bytes) | Position (bytes) | Description |
|---|---|---|---|
| A | 3 | (0-2) | Tag identification. Must contain 'TAG' if tag exists and is correct. |
| B | 30 | (3-32) | Title |
| C | 30 | (33-62) | Artist |
| D | 30 | (63-92) | Album |
| E | 4 | (93-96) | Year |
| F | 30 | (97-126) | Comment |
| G | 1 | (127) | Genre |

The specification asks for all fields to be padded with null character (ASCII 0). However, not all applications respect this (an example is WinAmp which pads fields with <space>, ASCII 32).

There is a small change proposed in **MP3v1.1** structure. The last byte of the Comment field may be used to specify the track number of a song in an album. It should contain a null character (ASCII 0) if the information is unknown.

Genre is a numeric field which may have one of the following values:

| | | | |
|---|---|---|---|
| 0 'Blues' | 20 'Alternative' | 40 'AlternRock' | 60 'Top 40' |
| 1 'Classic Rock' | 21 'Ska' | 41 'Bass' | 61 'Christian Rap' |
| 2 'Country' | 22 'Death Metal' | 42 'Soul' | 62 'Pop/Funk' |
| 3 'Dance' | 23 'Pranks' | 43 'Punk' | 63 'Jungle' |
| 4 'Disco' | 24 'Soundtrack' | 44 'Space' | 64 'Native American' |
| 5 'Funk' | 25 'Euro-Techno' | 45 'Meditative' | 65 'Cabaret' |
| 6 'Grunge' | 26 'Ambient' | 46 'Instrumental Pop' | 66 'New Wave' |
| 7 'Hip-Hop' | 27 'Trip-Hop' | 47 'Instrumental Rock' | 67 'Psychadelic' |
| 8 'Jazz' | 28 'Vocal' | 48 'Ethnic' | 68 'Rave' |
| 9 'Metal' | 29 'Jazz+Funk' | 49 'Gothic' | 69 'Showtunes' |
| 10 'New Age' | 30 'Fusion' | 50 'Darkwave' | 70 'Trailer' |
| 11 'Oldies' | 31 'Trance' | 51 'Techno-Industrial' | 71 'Lo-Fi' |
| 12 'Other' | 32 'Classical' | 52 'Electronic' | 72 'Tribal' |
| 13 'Pop' | 33 'Instrumental' | 53 'Pop-Folk' | 73 'Acid Punk' |
| 14 'R&B' | 34 'Acid' | 54 'Eurodance' | 74 'Acid Jazz' |
| 15 'Rap' | 35 'House' | 55 'Dream' | 75 'Polka' |
| 16 'Reggae' | 36 'Game' | 56 'Southern Rock' | 76 'Retro' |
| 17 'Rock' | 37 'Sound Clip' | 57 'Comedy' | 77 'Musical' |
| 18 'Techno' | 38 'Gospel' | 58 'Cult' | 78 'Rock & Roll' |
| 19 'Industrial' | 39 'Noise' | 59 'Gangsta' | 79 'Hard Rock' |

Any other value should be considered as 'Unknown'

# MPEG Audio Tag MP3v2

This is new proposed TAG format which is different than MP3v1 and MP3v1.1. Complete tech specs for it may be found at http://www.id3.com/.

---

Created on September 1998. by Predrag Supurovic.
Thanks to Jean Nicolle for debugging and polishing of this document.

# Mpeg Audio Datafile Format Specification

MPEG audio datafile (*.m3d) is database of informations gathered from mpeg audio files. You do not need to have mpeg audio file itself to get it's info if you previously stored that info in database. This is very useful for cataloguing data when mpeg audio files are on removable media.

Main purpose of this file format is mpeg audio info distribution among applications. It may be used as personal database, or as catalogue.

It is currently supported by MPGScript MPEG audio cataloguing application and MPGTools Delphi unit for accessing informations from MPEG files.

Last updated version of this document may be found at http://www.dv.co.yu/mpgscript/m3dspecs.htm

## File format specification

Generally, M3D is divided in four sections: file signature, application identification, header info, and mpeg data. In file structure they are sorted out like this:

### <file signature>

This section signs file as MPEG Audio datafile and determines file version. Version numbers are defined by author and may not be changed by third parties. You are entitled to create only files according to current version definition. Current version is 1.2 and this document describes it. If you want to read older versions, look support site of MPGTools Delphi Unit for details. Check this site for future structure updates.

| | | |
|---|---|---|
| <file_id> | 8 characters | Always contains #9'MP3DATA' characters. This part may be used by third party applications to recognize file as MPEG Audio Datafile. |
| <version> | 1 byte | Contains file version number (currently 1) |
| <subversion> | 1 byte | Contains file subversion number (currently 2) |

### <application id>

This section describes application used to create file. It also may contain other, application specific data. Third party applications may or may read data from this section or just skip them.

| | | |
|---|---|---|
| <block_length> | 1 byte | Contains length of <application id> section excluding <block_length> byte |
| < app_name_length> | 1 byte | Contains number of bytes used for application name (maximum value is 15). |
| <app_name> | up to 15 characters | Containing application name. Should be used to determine which application created m3d file. Application name must be registered with author to avoid same ID-s for different applications. |
| <custom data> | | Space used by application. Author of that application is responsible for publishing data structure of this block if he wants third parties to use his data. He does not have to do that. This block length may be calculated as <block_length>-<app_name_length>-1. It may contain any additional data application needs |

### <header>

Contains header info about owner of file, catalogue info, order info or else. It may use different structure, therefore it's divided into three blocks:

| | | |
|---|---|---|
| <header_type> | 1 byte | Describes type of header:<br><br>0 - custom header used by third party application (use <app id> to see which one) that created this header. Other applications should just skip reading header informations. Third party applications may publish their custom format. In that case, it is advisable to contact author of m3d to register its own type.<br><br>1 - catalogue header. Should be used for distributed MPEG catalogues.<br><br>Pascal (Delphi) structure definition:<br><br>`TMPEGDataCatalogue = packed record`<br>`  Title : string[30];  { Catalogue title }`<br>`  Publisher : string[30];  { Catalogue publisher name }`<br>`  City : String[30];  { Publisher's contact info }`<br>`  ZIP : String[10];` |

```
    Country : String[20];
    Address : String[30];
    Phone: String[15];
    Fax: string[15];
    Email: string[30];
    WWWURL: string[30];
  end;
```

2 - order header. Should be used for MPEG orders generated from catalogues.

Pacal (Delphi) structure defition:

```
  TMPEGDataOrder1v1 = packed record
    CustomerID : string[15]; { customer unique ID
                               used by catalogue publisher }
    Name : string[30];       { customer name and           }
    City : String[30];       { other contact data          }
    ZIP : String[10];
    Country : String[20];
    Address : String[30];
    Phone: String[15];
    Fax: string[15];
    Email: string[30];
  end;
```

Other - we may define other publicly available header types, that may be used by all applications. It is recommended to let us know about specific headers you use for your application, so we may add it to public header types if they are of general interest.

| | | |
|---|---|---|
| <header_length> | 2 bytes | Contains length of header data. Maximum value is 65535. It may be used to simplify reading header info and to skip unsupported header types. |
| <header_data> | up to 65535 bytes | Contains additional info about owner and data in file. It may have predefined or custom structure, which is described by <header_type> |

**<mpeg_data_records>**

This section contains unlimited number of MPEG data records.

This is Pascal (Delphi) record structure (note that string3, string30, string4, string255 and string20 are actually string[3], string[30[, string[4], string[255] and string[20]):

```
TMPEGData1v2 = packed record
  Header : String3;        { Should contain "TAG" if header is correct }
  Title : String30;        { Song title  }
  Artist : String30;       { Artist name }
  Album : String30;        { Album  }
  Year : String4;          { Year }
  Comment : String30;      { Comment }
  Genre : Byte;            { Genre code }
  Track : byte;            { Track number on Album }
  Duration : word;         { Song duration }
  FileLength : LongInt;    { File length }
  Version : byte;          { MPEG audio version index (1 - Version 1,
                             2 - Version 2,  3 - Version 2.5,
                             0 - unknown }
  Layer : byte;            { Layer (1, 2, 3, 0 - unknown) }
  SampleRate : LongInt;    { Sampling rate in Hz}
  BitRate : LongInt;       { Bit Rate }
  BPM : word;              { bits per minute - for future use }
  Mode : byte;             { Number of channels (0 - Stereo,
                             1 - Joint-Stereo, 2 - Dual-channel,
                             3 - Single-Channel) }
  Copyright : Boolean;     { Copyrighted? }
  Original : Boolean;      { Original? }
  ErrorProtection : boolean; { Error protected? }
  Padding : Boolean;       { If frame is padded }
  FrameLength : Word;      { total frame size including CRC }
  CRC : word;              { 16 bit File CRC (without TAG).
                             Not implemented yet. }
  FileName : String255;    { MPEG audio file name }
  FileDateTime : LongInt;  { File last modification date and time in
                             DOS internal format }
```

```
  FileAttr : Word;         { File attributes }
  VolumeLabel : string20;  { Disk label }
  Selected : word;         { If this field's value is greater than
                             zero then file is selected. Value
                             determines order of selection. }
  Reserved : array[1..45] of byte; { for future use }
end;
```

---

# EXERCISE IMAGE FILE FORMATS FOR THE WEB

To complete this exercise, submit a jpg and a gif file format image. Make sure correct resolution before save for web. Make sure you choose the right format for the right image type.

**Balance compression and quality**

**goal is the smallest possible file size w/out loosing too much quality**

**Png - portable network graphic format (Native file format for Fireworks**

new format
compression based on deflation which is zip technology
lossless 24 bit color compression yields 16 million colors
**.png-24** instead of **.jpg**
**.png-8** instead of **.gif**
supports interlacing
256 levels of transparency (gif has 1)
saves gamma curve w/image - looks the same on both platforms
.png files bigger than jpg & gif
doesn't support animation
not supported universally by older browsers
expect it to gain popularity as it gains more browser support

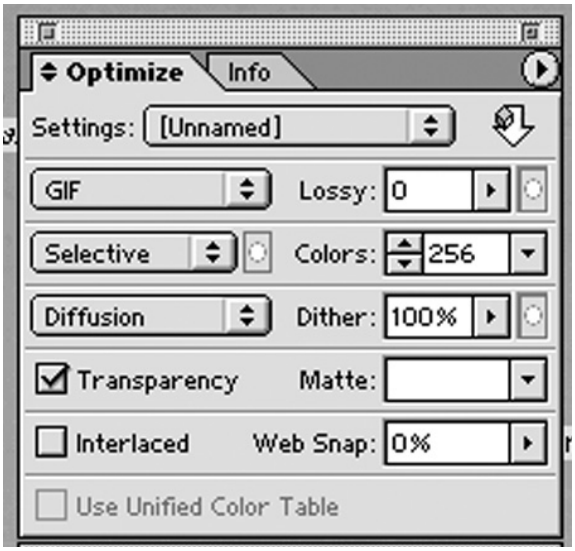
**Jpg or Jpeg - (joint photographic experts group) PHOTOGRAPHS WITH MANY COLORS**

equivalent to mpeg in movies but still image
single most important format for images w/more than 256 colors
saving splits brightness & color and compresses each individually
doesn't do well with high contrast areas
doesn't do well for grayscale images
not good for text
helps compression to blur image slightly
**no transparency**
**not lossless -** every time you save image shifts colors and degrades image
keep original tif and resave
minimize loss if you save with the same settings
progressive -simular to interlacing gif
appears in browser while downloading 3-5 passes
high quality compression 10:1-20:1 minimal loss of quality
medium quality compression 30:1-50:1 some visible color shifts
low quality 100:1 serious loss in quality
browsers view differently because of different decoders

**Gif - (Graphical Interchange Format) FLAT AREAS OF COLOR IN IMAGE**

most flexible format
2 compression techniques
LZW - from developers Lempel, Ziv & Welch (also in tif format)
based on pattern recognition
looks for pixels in a row (saves 5 x red instead of red 5 times)
CLUT - Color Look Up Table
few photos use full spectrum of colors
CLUT eliminates the colors not used cutting it down to 256 = 2/3 size reduction
this is called indexing and also available under mode - more options in save for web
Adjust color depth of CLUT relative to LZW compression
fewer colors in CLUT = better LZW
too much quality loss requires dithering which hurts LZW

**Options in optimization palette**




**Lossy** - uses patterns found by LZW algorithm and stores in compression table use slider to specify how much

**Dithering** - takes 2 colors from Color Table & makes missing color
improves visual quality of image
- increases file size because they compress less efficiently
- reduce the # of colors until drop in color quality > then adjust dithering to improve display keeping eye on file size

noise - best for the job
diffusion - simular to noise in effect, set amount in the slider
pattern - **don't use** mixes adjacent color in a regular pattern that you can see

**Interlaced** - important to use if you have a lot of images on pg
rearranges pixel rows so image displays gradually
viewers think its quicker but it actually takes slightly longer
.gif uses 4 passe to display - slight increase in size
.jpg lets you choose 3-5 passes - slightly smaller

**Transparency & Matte** - lets background color of web page show through transparent pixels halos-matte color select background of web page (anti-aliasing)

***Can't use interlacing & transparency in same image unwanted pixels appear in transparent areas***

**Color Reducing Algorithms** - decides which colors to include in CLUT
each image is unique & has different requirements
- adaptive - most often used with good results
  picks most frequently used colors in graphic not necessarily web safe colors so some dithering and color shifts
  web snap lets you shift colors to closets web safe
  ◆ marks web safe in color table
- selective & perceptual - simular
  selective is recommended by Adobe (default)
  favors broad areas of color and preservation of web colors
  perceptive - priority to color which human eye has greater sensitivity
- web - no dithering, no color changes, the same on both platforms except for gamma
  use only on graphics that have web safe colors
  worst mode for everything else/not used often

**Change web snap slider to 25% and change between adaptive/selective/perceptive**
**VIEW>BROWSER DITHER VIEW>PREVIEW** to see what it looks like in 256 colors
Other algorithms are exact, mac os/windows systems, previous, uniform
- Custom - make a common CLUT palette for all of your images
  use when many images with the same shades of color on the pg (colors don't shift)
  if lots of different images have their own adaptive color palette pc runs out of memory