welcome: please sign in
location: id3guide

Applies to: ID3v2.3 Table of Contents

Contents

Introduction

There are many people who enjoy .MP3 compressed music. The MP3 specification only defined the storage of musical data and did not provide the storage of metadata related to the musical composition; e.g., title, composer and artist, publisher, etc. The ID3 tag standard was created to remedy this need...

Rationale: Specifications are like grammar: they provide the rules of formatting data but few clues as to how to speak concisely and efficiently. The goal of this document is to answer the "Should I do..." and "Would doing X make it easier/faster/smaller..." type questions.

Audience: This document is geared toward programmers dealing directly with ID3v2.3 tags. There is a heavy slant towards writing tags correctly since decoding is relatively straightforward. You should familiarize yourself with the ID3v2 and related standards since this document refers to and uses the terminology from those documents.

Coverage: Although ID3v2 tags were created for use with MPEG Layer-3 audio streams, flexibility was a goal from the start. Even though much of this document refers to .MP3 files, the principles are generally applicable to other formats.

Use of this document

These are merely Guidelines and as such they are not part of the ID3v2 standard.

1. If anything in this document contradicts the published ID3v2 standard, then the standard shall prevail. The ID3v2 standard always has the final word.
2. These Guidelines are not binding on anyone (whereas the standard is binding, for obvious reasons.) This means you, as a programmer, should never assume anyone else will abide by

these guidelines. A protocol designer once said:

> "*Be flexible in what you accept, but strict in what you write.*"

3. Use your judgement and common sense.

What's new in ID3v2.3?

For starters, the naming convention changed. "ID3v2" now refers to the family of frame-based tagging methods which utilize the 10-byte header beginning with "ID3" followed by version information. "ID3v2.3.0" refers to the informal standard dated September 1998. The informal standard formerly known as "ID3v2" has been renamed to "ID3v2.2"

This document concentrates on ID3v2.3. The previous standard, ID3v2.2, is now obsolete.

A summary of the differences between ID3v2.3.0 and v2.2 is listed below. Section numbers match the ID3v2.3.0 Informal Standard.

Structural changes:

- The Frame Header has been expanded. The Frame ID and Frame Size fields are now four bytes long. Two bytes of Flags have been added. See §3.3.
- An optional Extended Header can follow the ID3v2 tag header §3.2. Among other things, the extended header can store a CRC of the entire tag.

Clarifications:

- Pre-defined Frame IDs have been renamed to four characters long.
- Unicode strings are now required to start with the Byte Order Mark (BOM) character. See §3.3.
- Compression and encryption are done on a frame-by-frame basis. The compression algorithm are those offered in zlib. §3.3
- Attached Picture [APIC] frame: the Image Format field has been replaced MIME Type §4.15.

New frames:

- Position Synchronization frame [POSS] §4.22
- Terms of Use frame [USER] §4.23
- Ownership frame [OWNE] §4.24
- Commercial frame [COMR] §4.25
- Private frame [PRIV] §4.28
- Encryption Method Registration [ENCR] §4.25
- Group ID Registration [GRID] §4.26.Used in conjunction with new frame features described in §3.3
- Text frames: Internet Radio Station Name [TRSN] and Internet Radio Station Owner [TRSO]
- URL link frames: Official internet radio station homepage [WORS] and Payment [WPAY]

Deleted frames:

- Encrypted meta-frame.

Programming Considerations

Padding

The use of padding and how much should be used has been a matter of debate ever since the first draft of ID3v2. (Take a look in the mailing list archives for arguments.)

Before making recommendations let's quantify the issue:

- A two minute song compressed at 44Khz and 128Kb/sec takes about 1990K (slightly less than 2MB) of disk space. The average size of .MP3 files in my collection is 3.5MB
- ID3v1 and Lyrics3 tags are only 128 bytes and 3K, respectively - and they served most songs just fine.
- An ID3v2 tag with basic information (title, artist/composer, copyright) is typically less than 1K in size. Toss in publisher information and lyrics and it comes out to 3K or so.
- Adding a 3K tag to a 2MB song will increase its size by 0.15%
- Adding a 10K tag to a 2MB song increases its size by 0.5%
- Suppose a CD-ROM holds 160 songs. Tagging all of them with 3K tags requires less than a megabyte.

So in reality tags are tiny compared to the amount of audio data accompanying it. A few KB of padding will not increase the file size noticeably. Remember also the padding is required to be 0's, which means on a slow modem link the padding is compressed significantly.

There is but one reason to use padding: since ID3v2 tags are stored at the beginning of the file, it would be a major pain to rewrite the entire multi-megabyte file to add a 50-byte frame. Padding reserves space in advance.

The issue then is how much padding should be used.

James's recommendations:

- What do most people care about? Probably title, artist, composer, band, publisher, lyrics and copyright. 4K will usually be enough for all that plus some miscellaneous information such as media type, file size, playcounter, time codes, etc.
- If you are editing an existing tag and run out of room (meaning you must rewrite the entire file) then double the amount of space reserved for the tag. For example, a file has a 4K tag filled with 3K of data. User wants to add 2K of data, so you rewrite the file reserving 8K at the beginning; then save the 5K of data and leaving 3K of padding.
    Obviously there should be an upper bound on doubling; a reasonable number would be 16K or 32K. For example, a file has a 25K tag but only 4K of padding left and the user wants to add 7K of new data. Rewrite the file and reserve 41K: save 28K of tag data and leave 13K of padding.
- Pictures are large and vary in size. If there are pictures then you should add more padding; use your judgement here.
- Do not forgo padding in files destined for read-only media or streaming applications. The user may want to save the file to another location and edit the tag.
- Include an option to let the user disable padding if (s)he really wants to.

Martin has a good idea: Add enough padding to round out the file to a full cluster. Obviously a minimum amount of padding has to be added to be useful, but beyond the minimum, everything up to the next cluster size will not occupy additional disk space. (Each operating system has its own terminology; Microsoft uses cluster, Apple uses allocation block. How files are stored on disk is beyond the scope of this document; consult an OS book and API reference for details.)

Here are some common cluster sizes:

Operating system (file system type)	**Disk size**						
	< 256MB	up to 512MB	up to 1GB	up to 2GB	up to 8GB	up to 16GB	> 16GB

DOS & Windows 95 (FAT16)	4K	8K	16K	32K	N/A (FAT16 can't handle drives > 4GB)	
Win95 OSR2 & Win98 (FAT32)	N/A	4K			8K	16K or 32K
Windows NT (NTFS)	0.5K to 64K (0.5K to 4K clusters are most common)					
MacOS pre-8.1 (HFS)	0.5K - 4K	4.5K - 8K	8.5K - 16K	16.5K - 32K	N/A (HFS can't handle disks > 2GB)	
MacOS 8.1 and later (HFS+)	0.5K	1K	2K	Default is 4K; user-selectable up to 4K		
CD-ROM, 650MB (ISO 9660)	always 2K					
DVD-ROM (UDF)	always 2K					

Advanced operating systems employ tricks to optimize disk space. For example, Novell NetWare 4 uses 64K disk blocks but can split a block into 0.5K pieces, thus creating the illusion of 512-byte blocks. UNIX file systems have their own methods to utilize disk space efficiently.

James's shortcut on choosing an appropriate cluster size: 2K (Nice even number, not too big, not too small)

Read-only media

Never assume a .MP3 file is writable unless the user is specifically editing the tag. MP3 files can be stored on read-only media such as a CD-ROM or a network share. If the user is editing the tag, it is an error if the file is write-protected because the user's changes cannot be saved. If a player is merely updating the playcounter or popularity-meter, it should not pop up a message complaining the file cannot be written to.

James: Include a visual cue to indicate whether the file is write-protected. For example, a small icon similar in size and color to the stereo indicator in WinAMP. A quick glance will determine whether certain settings will be saved.

"Insignificant" frames

"Insignificant" frames are small frames that don't necessarily have meaning when a ID3v2 tag is created. The issue is whether the tag editor should add these frames when they do not already exist.

For example, if Joe is adding tags to a fresh batch of .MP3 files, should the tag editor include a playcounter [PCNT] frame in the tag? The tag editor has no idea how many times the particular file has been played; unless Joe tells it otherwise, the editor has to create a [PCNT] frame with a count of 0.

There are two ways to look at this. On the one hand, the frame is so small it takes almost no effort to include it. Since there will usually be a few hundred bytes of padding, the ten bytes used by the counter frame is in a sense "free." A player will probably add it later anyway when the file is used.

On the other hand, if the frame is not holding useful information, why bother adding it? Padding, if used, effectively reserves space for these frames. Consider also the absence of a frame can be meaningful: the lack of a playcounter frame may indicate "I do not know how many times this piece has been played" as opposed to a count of 0, which indicates "This song has never been played."

These can be considered "insignificant" frames:

- Playcounter [PCNT]

  (**James**: an integrated ripper-encoder-tagger can include a playcount of 0 in newly created files, since by definition they have never been used..)
- Others, anyone?

Martin: It is considered good manner to allow the user to disable writing of frames (s)he doesn't want. At least in an advanced menu in a hidden place. Perhaps the user doesn't want the TSI, TLE, TMT and MLL frame to be added automatically, even though it might be the default setting.

Preferred image formats

A question Martin gets often is why the ID3v2 document says "... PNG and JPEG picture format should be used..." Interoperability means it is almost guaranteed another ID3v2 tag/picture decoder can handle the PNG and JPEG formats. The chances a Macintosh application can display BMP files are slim; likewise, the complexities of EPS and TIFF are best left out.

Consider these other arguments favoring PNG and JPEG:

- PNG compresses better than BMP, GIF and most other lossless formats.
- JPEG has superior compression to most formats, especially for photographic pictures. (JPEG is not suitable for line-art and computer-generated graphics - use PNG for these.)
- Both formats are patent- and royalty-free (which is not the case with GIF, whose LZW compression algorithm is patented by Unisys.)

Martin: The freedom to use whatever format you like in the picture frame is of course a freedom under responsibility. Do you want to make cross platform tags? Do you want to avoid legal trouble, at least for the principle? If so, use PNG and JPEG.

Dirk: Applications may want to convert the incoming picture type to PNG or JPEG first, or failing that, at least inform the user that they should be using PNG or JPEG, but allow the user to override this if they insist on using, say, a PCX.

Multiple tags

Decoders should be prepared to handle multiple ID3v2 tags per stream. This is especially important for players/decoders wanting to handle netradio-type applications since the notion of distinct files may not apply.

Dirk: An easy way to achieve this would be to use a central 'frame dispatch' routine, kind of like a demultiplexer.

Unsynchronization

The ID3v2 standard states "The only purpose of the 'unsychronisation scheme' is to make the ID3v2 tag as compatible as possible with existing software *[at the time the ID3v2.2 standard was drafted]*" What are the implications?

1. MP3 decoders, regardless of age, will not be affected by extraneous data (i.e., tags) that does not contain a MPEG sync sequence.
2. There is minimal impact if a decoder does not recognize ID3v2 tags but encounters something with a sync sequence. At worst a click or pop will be heard at the beginning of the piece, as if some data corruption has occurred.
3. New software is expected to take advantage of ID3v2. Unsynchronization is not necessary with ID3v2 compliant software.

(Software which does not behave according to items 1 and 2 above are categorically deemed "broken." Microsoft's Media Player is an example of such software.)

Martin: I believe that unsynchronization should be done as seldom as possible since it increases the size of the tag as well as the parsing time. In other words, I think unsynchronization should be turned off as default.

However, it is important to be able to undo unsynchronization when reading tags; otherwise unsynchronized tags will not be read correctly.

Pitfalls & General Advice

RTFM (details are important!)

When the ID3v2 document refers to another standard it is assumed that the implementor is familiar with that standard as well. When it says URL it does not mean 'www.buymusic.com', it means ' http://www.buymusic.com/' A "URL containing an e-mail address" must include the ' mailto:' qualifier. Those who at least take a quick look at the related standards will not make these kinds of mistakes.

If these details are so important, why aren't they spelled out in the ID3v2 standard? According to Martin, "There has been people who said to me, 'You must write these kind of things in the document, or else people will make these mistakes.' The latest ID3 v2.01 draft is *only* 74,657 bytes of pure text because I did not. Guess why there are references to the standards in the document!"

Compression before encryption

Encryption works by scrambling data to produce what amounts to random bits for an attacker. Data compression works by replacing or removing redundant information. It follows that the output of any decent crypto algorithm will not be compressible; therefore if you wish to use both compression and encryption, you must compress the data before encrypting it.

Compressing before encryption also affords better security, since predictable headers and such will be hidden by compression.

Remember: *compression* comes before *encryption* in the dictionary (at least in English) and that is how it should be in your application.

Validating user input

Never trust the input from the user to be sensible or formatted correctly. Is the ISRC a valid one? Does the month 13 exist? Could this piece of music be from disc 5 out of a 3 disc set?

Dirk: Personally, I think having input validation occur when the user signifies that the tag is "finished" is a good way of going about it, if only because I know I'll get half way through something and want to stop and do something else before I forget.

James: Extremely strict validation becomes restrictions on versatility and usability. An option to turn off or accept potentially invalid data can be included somewhere. For example, by default Microsoft Visual Basic checks for syntax errors as code is entered; however, users can disable that feature if they wish to delay syntax checking until compile time.

Things to watch out for (this is by no means an all-inclusive list):

- Properly formatted URLs (see RTFM, above.) But should your application verify whether the URL actually exists? That may be too complicated; Martin and James's opinion is no

verification.
- Proper MIME types.
- Are characters valid? Most text fields forbid control characters, even newlines. NULs are not allowed except in Unicode strings or to indicate end-of-string for terminated strings.
- Numerical strings must consist solely of the characters '0' through '9'
- Are lists handled properly? The list separator is a slash: / What happens if the user enters a / in an item?
- Are characters such as (escaped where necessary?
- Do dates and times make sense? Years must be four digits long and be in the right century. (No Y2K problems here!)
- Is an image really the format the user claims it is? Some people believe renaming a .BMP file to .PNG will change its format accordingly.

Your application should never crash because the user does something stupid. Likewise, your application should not crash because of erroneous or malformed data in a tag. It is particularly important your application can recognize Unicode text frames and ignore them if the operating system or your program cannot handle Unicode.

Credits & Contributors

- Dirk Mahoney, creator of the ID3Lib C++ tag processing library.
- James Lin, author of these Guidelines.
- Martin Nilsson, author of the ID3v2 standard.

References

- www.id3.org: Home of the ID3v2 Informal Standard and lots of other useful material.

Copyright & Legal Notice

id3guide (last edited 2013-11-24 23:12:11 by DanONeill)