Informal standard
Document: id3v2.3.0.txt

M. Nilsson
3rd February 1999

ID3 tag version 2.3.0

Status of this document

This document is an informal standard and replaces the ID3v2.2.0 standard [ID3v2]. The informal standard is released so that implementors could have a set standard before a formal standard is set. The formal standard will use another version or revision number if not identical to what is described in this document. The contents in this document may change for clarifications but never for added or altered functionallity.

Distribution of this document is unlimited.

Abstract

This document describes the ID3v2.3.0, which is a more developed version of the ID3v2 informal standard [ID3v2] (version 2.2.0), evolved from the ID3 tagging system. The ID3v2 offers a flexible way of storing information about an audio file within itself to determine its origin and contents. The information may be technical information, such as equalisation curves, as well as related meta information, such as title, performer, copyright etc.

1. Table of contents

2. Conventions in this document

In the examples, text within "" is a text string exactly as it appears in a file. Numbers preceded with $ are hexadecimal and numbers preceded with % are binary. $xx is used to indicate a byte with unknown content. %x is used to indicate a bit with unknown content. The most significant bit (MSB) of a byte is called 'bit 7' and the least significant bit (LSB) is called 'bit 0'.

A tag is the whole tag described in this document. A frame is a block of information in the tag. The tag consists of a header, frames and optional padding. A field is a piece of information; one value, a string etc. A numeric string is a string that consists of the characters 0-9 only.

3. ID3v2 overview

The two biggest design goals were to be able to implement ID3v2 without disturbing old software too much and that ID3v2 should be as flexible and expandable as possible.

The first criterion is met by the simple fact that the MPEG [MPEG] decoding software uses a syncsignal, embedded in the audiostream, to 'lock on to' the audio. Since the ID3v2 tag doesn't contain a valid syncsignal, no software will attempt to play the tag. If, for any reason, coincidence make a syncsignal appear within the tag it will be taken care of by the 'unsynchronisation scheme' described in section 5.

The second criterion has made a more noticeable impact on the design of the ID3v2 tag. It is constructed as a container for several information blocks, called frames, whose format need not be known to the software that encounters them. At the start of every frame there is an identifier that explains the frames' format and content, and a size descriptor that allows software to skip unknown frames.

If a total revision of the ID3v2 tag should be needed, there is a version number and a size descriptor in the ID3v2 header.

The ID3 tag described in this document is mainly targeted at files encoded with MPEG-1/2 layer I, MPEG-1/2 layer II, MPEG-1/2 layer III and MPEG-2.5, but may work with other types of encoded audio.

The bitorder in ID3v2 is most significant bit first (MSB). The byteorder in multibyte numbers is most significant byte first (e.g. $12345678 would be encoded $12 34 56 78).

It is permitted to include padding after all the final frame (at the end of the ID3 tag), making the size of all the frames together smaller than the size given in the head of the tag. A possible purpose of this padding is to allow for adding a few additional frames or enlarge existing frames within the tag without having to rewrite the entire file. The value of the padding bytes must be $00.

3.1. ID3v2 header

The ID3v2 tag header, which should be the first information in the file, is 10 bytes as follows:

```
ID3v2/file identifier      "ID3"
ID3v2 version              $03 00
ID3v2 flags                %abc00000
ID3v2 size             4 * %0xxxxxxx
```

The first three bytes of the tag are always "ID3" to indicate that this is an ID3v2 tag, directly followed by the two version bytes. The first byte of ID3v2 version is it's major version, while the second byte is its revision number. In this case this is ID3v2.3.0. All revisions are backwards compatible while major versions are not. If software with ID3v2.2.0 and below support should encounter version three or higher it should simply ignore the whole tag. Version and revision will never be $FF.

The version is followed by one the ID3v2 flags field, of which currently only three flags are used.

a - Unsynchronisation

Bit 7 in the 'ID3v2 flags' indicates whether or not unsynchronisation is used (see section 5 for details); a set bit indicates usage.

b - Extended header

The second bit (bit 6) indicates whether or not the header is followed by an extended header. The extended header is described in section 3.2.

c - Experimental indicator

The third bit (bit 5) should be used as an 'experimental indicator'. This flag should always be set when the tag is in an experimental stage.

All the other flags should be cleared. If one of these undefined flags are set that might mean that the tag is not readable for a parser that does not know the flags function.

The ID3v2 tag size is encoded with four bytes where the most significant bit (bit 7) is set to zero in every byte, making a total of 28 bits. The zeroed bits are ignored, so a 257 bytes long tag is represented as $00 00 02 01.

The ID3v2 tag size is the size of the complete tag after unsychronisation, including padding, excluding the header but not excluding the extended header (total tag size - 10). Only 28 bits (representing up to 256MB) are used in the size description to avoid the introducuction of 'false syncsignals'.

An ID3v2 tag can be detected with the following pattern:
  $49 44 33 yy yy xx zz zz zz zz
Where yy is less than $FF, xx is the 'flags' byte and zz is less than $80.

3.2. ID3v2 extended header

The extended header contains information that is not vital to the correct parsing of the tag information, hence the extended header is optional.

Extended header size	$xx xx xx xx
Extended Flags	$xx xx
Size of padding	$xx xx xx xx

Where the 'Extended header size', currently 6 or 10 bytes, excludes itself. The 'Size of padding' is simply the total tag size excluding the frames and the headers, in other words the padding. The extended header is considered separate from the header proper, and as such is subject to unsynchronisation.

The extended flags are a secondary flag set which describes further attributes of the tag. These attributes are currently defined as follows

  %x0000000 00000000

x - CRC data present

  If this flag is set four bytes of CRC-32 data is appended to the extended header. The CRC should be calculated before unsynchronisation on the data between the extended header and the padding, i.e. the frames and only the frames.

Total frame CRC	$xx xx xx xx

3.3. ID3v2 frame overview

As the tag consists of a tag header and a tag body with one or more frames, all the frames consists of a frame header followed by one or more fields containing the actual information. The layout of the frame header:

```
  Frame ID   $xx xx xx xx  (four characters)
  Size       $xx xx xx xx
  Flags      $xx xx
```

The frame ID made out of the characters capital A-Z and 0-9. Identifiers beginning with "X", "Y" and "Z" are for experimental use and free for everyone to use, without the need to set the experimental bit in the tag header. Have in mind that someone else might have used the same identifier as you. All other identifiers are either used or reserved for future use.

The frame ID is followed by a size descriptor, making a total header size of ten bytes in every frame. The size is calculated as frame size excluding frame header (frame size - 10).

In the frame header the size descriptor is followed by two flags bytes. These flags are described in section 3.3.1.

There is no fixed order of the frames' appearance in the tag, although it is desired that the frames are arranged in order of significance concerning the recognition of the file. An example of such order: UFID, TIT2, MCDI, TRCK ...

A tag must contain at least one frame. A frame must be at least 1 byte big, excluding the header.

If nothing else is said a string is represented as ISO-8859-1 [ISO-8859-1] characters in the range $20 - $FF. Such strings are represented as <text string>, or <full text string> if newlines are allowed, in the frame descriptions. All Unicode strings [UNICODE] use 16-bit unicode 2.0 (ISO/IEC 10646-1:1993, UCS-2). Unicode strings must begin with the Unicode BOM ($FF FE or $FE FF) to identify the byte order.

All numeric strings and URLs [URL] are always encoded as ISO-8859-1. Terminated strings are terminated with $00 if encoded with ISO-8859-1 and $00 00 if encoded as unicode. If nothing else is said newline character is forbidden. In ISO-8859-1 a new line is represented, when allowed, with $0A only. Frames that allow different types of text encoding have a text encoding description byte directly after the frame size. If ISO-8859-1 is used this byte should be $00, if Unicode is used it should be $01. Strings dependent on encoding is represented as <text string according to encoding>, or <full text string according to encoding> if newlines are allowed. Any empty Unicode strings which are NULL-terminated may have the Unicode BOM followed by a Unicode NULL ($FF FE 00 00 or $FE FF 00 00).

The three byte language field is used to describe the language of the frame's content, according to ISO-639-2 [ISO-639-2].

All URLs [URL] may be relative, e.g. "picture.png", "../doc.txt".

If a frame is longer than it should be, e.g. having more fields than specified in this document, that indicates that additions to the frame have been made in a later version of the ID3v2 standard. This is reflected by the revision number in the header of the tag.

3.3.1. Frame header flags

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.