NAME

MP3::Tag::ID3v2 − Read / Write ID3v2.x.y tags from mp3 audio files

SYNOPSIS

MP3::Tag::ID3v2 supports

* Reading of ID3v2.2.0 and ID3v2.3.0 tags (some ID3v2.4.0 frames too)
* Writing of ID3v2.3.0 tags

MP3::Tag::ID3v2 is designed to be called from the MP3::Tag module. If you want to make calls from user code, please consider using highest-level wrapper code in MP3::Tag, such as *update_tags()* and *select_id3v2_frame_by_descr()*.

Low-level creation code:

```
use MP3::Tag;
$mp3 = MP3::Tag->new($filename);

# read an existing tag
$mp3->get_tags();
$id3v2 = $mp3->{ID3v2} if exists $mp3->{ID3v2};

# or create a new tag
$id3v2 = $mp3->new_tag("ID3v2");
```

See MP3::Tag for information on the above used functions.

* Reading a tag, very low-level:

```
$frameIDs_hash = $id3v2->get_frame_ids('truename');

foreach my $frame (keys %$frameIDs_hash) {
    my ($name, @info) = $id3v2->get_frames($frame);
    for my $info (@info) {
        if (ref $info) {
            print "$name ($frame):\n";
            while(my ($key,$val)=each %$info) {
                print " * $key => $val\n";
           }
        } else {
            print "$name: $info\n";
        }
    }
}
```

* Adding / Changing / Removing a frame in memory (higher-level)

```
$t = $id3v2->frame_select("TIT2", undef, undef); # Very flexible

$c = $id3v2->frame_select_by_descr("COMM(fre,fra,eng,#0)[]");
$t = $id3v2->frame_select_by_descr("TIT2");
     $id3v2->frame_select_by_descr("TIT2", "MyT"); # Set/Change
     $id3v2->frame_select_by_descr("RBUF", $n1, $n2, $n3); # Set/Change
     $id3v2->frame_select_by_descr("RBUF", "$n1;$n2;$n3"); # Set/Change
     $id3v2->frame_select_by_descr("TIT2", undef); # Remove
```

* Adding / Changing / Removing a frame in memory (low-level)

```
$id3v2->add_frame("TIT2", "Title of the audio");
$id3v2->change_frame("TALB","Greatest Album");
$id3v2->remove_frame("TLAN");
```

* Output the modified-in-memory version of the tag:

```
$id3v2->write_tag();
```

* Removing the whole tag from the file

```
$id3v2->remove_tag();
```

* Get information about supported frames

```
%tags = $id3v2->supported_frames();
while (($fname, $longname) = each %tags) {
    print "$fname $longname: ",
          join(", ", @{$id3v2->what_data($fname)}), "\n";
}
```

AUTHOR

Thomas Geffert, thg AT users DOT sourceforge DOT net Ilya Zakharevich, ilyaz AT cpan DOT org

DESCRIPTION

get_frame_ids()

```
$frameIDs = $tag->get_frame_ids;
$frameIDs = $tag->get_frame_ids('truename');

[old name: getFrameIDs() . The old name is still available, but you should u
```

get_frame_ids loops through all frames, which exist in the tag. It returns a hash reference with a list of all available Frame IDs. The keys of the returned hash are 4−character−codes (short names), the internal names of the frames, the according value is the english (long) name of the frame.

You can use this list to iterate over all frames to get their data, or to check if a specific frame is included in the tag.

If there are multiple occurences of a frame in one tag, the first frame is returned with its normal short name, following frames of this type get a ’01’, ’02’, ’03’, ... appended to this name. These names can then used with `get_frame` to get the information of these frames. These fake frames are not returned if `'truename'` argument is set; one can still use `get_frames()` to extract the info for all of the frames with the given short name.

get_frame()

```
($info, $name, @rest) = $tag->get_frame($ID);
($info, $name, @rest) = $tag->get_frame($ID, 'raw');

[old name: getFrame() . The old name is still available, but you should use
```

get_frame gets the contents of a specific frame, which must be specified by the 4−character−ID (aka short name). You can use `get_frame_ids` to get the IDs of the tag, or use IDs which you hope to find in the tag. If the ID is not found, `get_frame` returns empty list, so `$info` and `$name` become undefined.

Otherwise it extracts the contents of the frame. Frames in ID3v2 tags can be very small, or complex and huge. That is the reason, that `get_frame` returns the frame data in two ways, depending on the tag.

If it is a simple tag, with only one piece of data, these data is returned directly as ($info, `$name`), where `$info` is the text string, and `$name` is the long (english) name of the frame.

If the frame consist of different pieces of data, `$info` is a hash reference, `$name` is again the long name of the frame.

The hash, to which `$info` points, contains key/value pairs, where the key is always the name of the data, and the value is the data itself.

If the name starts with a underscore (as eg ’_code’), the data is probably binary data and not printable. If the name starts without an underscore, it should be a text string and printable.

If the second parameter is given as `'raw'`, the whole frame data is returned, but not the frame header. If the second parameter is `'intact'`, no mangling of embedded `"\0"` and trailing spaces is performed. If the second parameter is `'hash'`, then, additionally, the result is always in the hash format; likewise, if it is `'array'`, the result is an array reference (with `key => value` pairs same as with `'hash'`, but ordered as in the frame). If it is `'array_nokey'`, only the ‘‘value’’ parts are returned (in particular, the result is suitable to give to *add_frame()*, *change_frame()*); in addition, if it is `'array_nodecode'`, then keys are not returned, and the

setting of `decode_encoding_v2` is ignored. (The ‘‘return array’’ flavors don’t massage the fields for better consumption by humans, so the fields should be in format suitable for *frame_add()*.)

If the data was stored compressed, it is uncompressed before it is returned (even in raw mode). Then `$info` contains a string with all data (which might be binary), and `$name` the long frame name.

See also MP3::Tag::ID3v2−Data for a list of all supported frames, and some other explanations of the returned data structure.

If more than one frame with name `$ID` is present, `@rest` contains `$info` fields for all consequent frames with the same name. Note that after removal of frames there may be holes in the list of frame names (as in `FRAM FRAM01 FRAM02`) in the case when multiple frames of the given type were present; the removed frames are returned as `undef`.

! Encrypted frames are not supported yet !

! Some frames are not supported yet, but the most common ones are supported !

get_frame_descr()

```
$long_name = $self->get_frame_descr($fname);
```

returns a ‘‘long name’’ for the frame (such as `COMM(eng)[lyricist birthdate]`), appropriate for interpolation, or for *frame_select_by_descr()*.

get_frame_descriptors()

```
@long_names = $self->get_frame_descriptors();
```

return ‘‘long names’’ for the frames in the tag (see `get_frame_descr`).

get_frame_option()

```
$options = get_frame_option($ID);

Option is a hash reference, the hash contains all possible options.
The value for each option is 0 or 1.

groupid    -- not supported yet
encryption -- not supported yet
compression -- Compresses frame before writing tag;
               compression/uncompression is done automatically
read_only   -- Ignored by this library, should be obeyed by application
file_preserv -- Ignored by this library, should be obeyed by application
tag_preserv -- Ignored by this library, should be obeyed by application
```

set_frame_option()

```
$options = set_frame_option($ID, $option, $value);

Set $option to $value (0 or 1). If successfull the new set of
options is returned, undef otherwise.

groupid    -- not supported yet
encryption -- not supported yet
compression -- Compresses frame before writing tag;
               compression/uncompression is done automatically
read_only   -- Ignored by this library, should be obeyed by application
file_preserv -- Ignored by this library, should be obeyed by application
tag_preserv -- Ignored by this library, should be obeyed by application
```

get_frames()

```
($name, @info) = get_frames($ID);
($name, @info) = get_frames($ID, 'raw');
```

Same as *get_frame()* with different order of the returned values. `$name` and elements of the array `@info` have the same semantic as for *get_frame()*; each frame with id `$ID` produces one elements of array `@info`.

as_bin()

```
$tag2 = $id3v2->as_bin($ignore_error, $update_file, $raw_ok);
```

Returns the the current content of the ID3v2 tag as a string good to write to a file; it contains all the necessary footers and headers.

If `$ignore_error` is TRUE, the frames the module does not know how to write are skipped; otherwise it is an error to have such a frame. Returns undef on error.

If the optional argument `$update_file` is TRUE, an additional action is performed: if the audio file does not contain an ID3v2 tag, or the tag in the file is smaller than the built ID3v2 tag, the necessary 0-padding is inserted before the audio content of the file so that it is able to accommodate the build tag (and the `tagsize` field of `$id3v2` is updated correspondingly); in any case the header length of `$tag2` is set to reflect the space in the beginning of the audio file.

Unless `$update_file` has `'padding'` as a substring, the actual length of the string `$tag2` is not modified, so if it is smaller than the reserved space in the file, one needs to add some 0 padding at the end. Note that if the size of reserved space can shrink (as with `id3v2_shrink` configuration option), then without this option it would be hard to calculate necessary padding by hand.

If `$raw_ok` option is given, but not `$update_file`, the original contents is returned for unmodified tags.

as_bin_raw()

```
$tag2 = $id3v2->as_bin_raw($ignore_error, $update_file);
```

same as *as_bin()* with `$raw_ok` flag.

write_tag()

```
$id3v2->write_tag($ignore_error);
```

Saves all frames to the file. It tries to update the file in place, when the space of the old tag is big enough for the new tag. Otherwise it creates a temp file with a new tag (i.e. copies the whole mp3 file) and renames/moves it to the original file name.

An extended header with CRC checksum is not supported yet.

Encryption of frames and group ids are not supported. If `$ignore_error` is set, these options are ignored and the frames are saved without these options. If `$ignore_error` is not set and a tag with an unsupported option should be save, the tag is not written and a 0 is returned.

If a tag with an encrypted frame is read, and the frame is not changed it can be saved encrypted again.

ID3v2.2 tags are converted automatically to ID3v2.3 tags during writing. If a frame cannot be converted automatically (PIC; CMR), writing aborts and returns a 0. If `$ignore_error` is true, only not convertable frames are ignored and not written, but the rest of the tag is saved as ID3v2.3.

At the moment the tag is automatically unsynchronized.

If the tag is written successfully, 1 is returned.

remove_tag()

```
$id3v2->remove_tag();
```

Removes the whole tag from the file by copying the whole mp3-file to a temp-file and renaming/moving that to the original filename.

Do not use *remove_tag()* if you only want to change a header, as otherwise the file is copied unnecessarily. Use *write_tag()* directly, which will override an old tag.

add_frame()

```
$fn = $id3v2->add_frame($fname, @data);
```

Add a new frame, identified by the short name `$fname`. The number of elements of array `@data` should be as described in the ID3v2.3 standard. (See also MP3::Tag::ID3v2-Data.) There are two exceptions: if `@data` is empty, it is filled with necessary number of `""`); if one of required elements is `encoding`, it may be omitted or be `undef`, meaning the arguments are in

''Plain Perl (=ISOLatin−1 or Unicode) encoding''.

It returns the the short name `$fn` (which can differ from `$fname`, when an `$fname` frame already exists). If no other frame of this kind is allowed, an empty string is returned. Otherwise the name of the newly created frame is returned (which can have a 01 or 02 or ... appended).

You have to call *write_tag()* to save the changes to the file.

Examples (with `$id3v2->` omitted):

```
$f = add_frame('TIT2', 0, 'Abba');   # $f='TIT2'
$f = add_frame('TIT2', 'Abba');      # $f='TIT201', encoding=0 implicit

$f = add_frame('COMM', 'ENG', 'Short text', 'This is a comment');

$f = add_frame('COMM');              # creates an empty frame

$f = add_frame('COMM', 'ENG');       # ! wrong ! $f=undef, becaues number
                                     # of arguments is wrong

$f = add_frame('RBUF', $n1, $n2, $n3);
$f = add_frame('RBUF', $n1, $n2);      # last field of RBUF is optional
```

If a frame has optional fields *and* `encoding` (only `COMR` frame as of ID3v2.4), there may be an ambiguity which fields are omitted. It is resolved this way: the `encoding` field can be omitted only if all other optional frames are omitted too (set it to `undef` instead).

add_frame_split()

The same as *add_frame()*, but if the number of arguments is unsufficient, would *split()* the last argument on `;` to obtain the needed number of arguments. Should be avoided unless it is known that the fields do not contain `;` (except for `POPM RBUF RVRB SYTC`, where splitting may be done non-ambiguously).

```
# No ambiguity, since numbers do not contain ";":
$f = add_frame_split('RBUF', "$n1;$n2;$n3");
```

For `COMR` frame, in case when the fields are `join()`ed by `';'`, `encoding` field may be present only if all the other fields are present.

change_frame()

```
$id3v2->change_frame($fname, @data);
```

Change an existing frame, which is identified by its short name `$fname` eg as returned by *get_frame_ids()*. `@data` must be same as in *add_frame()*.

If the frame `$fname` does not exist, undef is returned.

You have to call *write_tag()* to save the changes to the file.

remove_frame()

```
$id3v2->remove_frame($fname);
```

Remove an existing frame. `$fname` is the short name of a frame, eg as returned by *get_frame_ids()*.

You have to call *write_tag()* to save the changes to the file.

copy_frames($from, `$to`, `$overwrite`, [$keep_flags, `$f_ids`])

Copies specified frames between `MP3::Tag::ID3v2` objects `$from`, `$to`. Unless `$keep_flags`, the copied frames have their flags cleared. If the array reference `$f_ids` is not specified, all the frames (but `GRID` and `TLEN`) are considered (subject to `$overwrite`), otherwise `$f_ids` should contain short frame ids to consider. Group ID flag is always cleared.

If `$overwrite` is `'delete'`, frames with the same descriptors (as returned by *get_frame_descr()*) in `$to` are deleted first, then all the specified frames are copied. If `$overwrite` is FALSE, only frames with descriptors not present in `$to` are copied. (If one of these two conditions is not met, the result may be not conformant to standards.)

Returns count of copied frames.

is_modified()

```
$id3v2->is_modified;
```

Returns true if the tag was modified after it was created.

supported_frames()

```
$frames = $id3v2->supported_frames();
```

Returns a hash reference with all supported frames. The keys of the hash are the short names of the supported frames, the according values are the long (english) names of the frames.

what_data()

```
($data, $res_inp, $data_map) = $id3v2->what_data($fname);
```

Returns an array reference with the needed data fields for a given frame. At this moment only the internal field names are returned, without any additional information about the data format of this field. Names beginning with an underscore (normally '_data') can contain binary data. (The `_encoding` field is skipped in this list, since it is usually auto-deduced by this module.)

`$resp_inp` is a reference to a hash (keyed by the field name) describing restrictions for the content of the data field. If the entry is undef, no restriction exists. Otherwise it is a hash. The keys of the hash are the allowed input, the correspodending value is the value which is actually stored in this field. If the value is undef then the key itself is valid for saving. If the hash contains an entry with ''_FREE'', the hash contains only suggestions for the input, but other input is also allowed.

`$data_map` contains values of `$resp_inp` in the order of fields of a frame (including `_encoding`).

Example for picture types of the APIC frame:

```
{"Other"                                => "\x00",
 "32x32 pixels 'file icon' (PNG only)" => "\x01",
 "Other file icon"                      => "\x02",
 ...}
```

title([@new_title])

Returns the title composed of the tags configured via `MP3::Tag->config('v2title')` call (with default 'Title/Songname/Content description' (TIT2)) from the tag. (For backward compatibility may be called by deprecated name *song()* as well.)

Sets TIT2 frame if given the optional arguments `@new_title`. If this is an empty string, the frame is removed.

_comment([$language])

Returns the file comment (COMM with an empty 'Description') from the tag, or ''Subtitle/Description refinement'' (TIT3) frame (unless it is considered a part of the title).

comment()

```
$val = $id3v2->comment();
$newframe = $id3v2->comment('Just a comment for freddy', 'personal', 'eng')
```

Returns the file comment (COMM frame with the 'Description' field in `default_descr_c` configuration variable, defalting to `''`) from the tag, or ''Subtitle/Description refinement'' (TIT3) frame (unless it is considered a part of the title).

If optional arguments ($comment, `$short`, `$language`) are present, sets the comment frame. If `$language` is omited, uses the `default_language` configuration variable (default is `XXX`). If not `XXX`, this should be lowercase 3-letter abbreviation according to ISO-639-2).

If `$short` is not defined, uses the `default_descr_c` configuration variable. If `$comment` is an empty string, the frame is removed.

frame_select($fname, `$descrs`, `$languages` [, `$newval1`, ...])

Used to get/set/delete frames which may be not necessarily unique in a tag.

```
# Select short-description='', prefere language 'eng', then 'rus', then
# the third COMM frame, then any (in this case, the first or the second)
# COMM frame
$val = $id3v2->frame_select('COMM', '', ['eng', 'rus', '#2', '']); # Read
$new = $id3v2->frame_select('COMM', '', ['eng', 'rus', '#2'],      # Write
                            'Comment with empty "Description" and "eng"');
$new = $id3v2->frame_select('COMM', '', ['eng', 'rus', '#2'],      # Delete
                            undef);
```

Returns the contents of the first frame named `$fname` with a 'Description' field in the specified array reference `$descrs` and the language in the list of specified languages `$languages`; empty return otherwise. If the frame is a "simple frame", the frame is returned as a string, otherwise as a hash reference; a "simple frame" should consist of one of Text/URL/_Data fields, with possible addition of Language and Description fields (if the corresponding arguments were defined).

The lists `$descrs` and `$languages` of one element can be flattened to become this element (as with `''` above). If the lists are not defined, no restriction is applied; to get the same effect with defined arguments, use `$languages` of `''`, and/or `$descrs` a hash reference. Language of the form `'#NUMBER'` selects the NUMBER's (0-based) frame with frame name `$fname`.

If optional arguments `$newval1...` are given, **ALL** the found frames are removed; if only one such argument `undef` is given, this is the only action. Otherwise, a new frame is created afterwards (the first elements of `$descrs` and `$languages` are used as the short description and the language, defaulting to `''` and the `default_language` configuration variable (which, in turn, defaults to XXX; if not XXX, this should be lowercase 3-letter abbreviation according to ISO-639-2). If new frame is created, the frame's name is returned; otherwise the count of removed frames is returned.

As a generalization, APIC frames are handled too, using `Picture Type` instead of `Language`, and auto-calculating `MIME type` for (currently) TIFF/JPEG/GIF/PNG/BMP and octet-stream. Only frames with `MIME type` coinciding with the auto-calculated value are considered as "simple frames". One can use both the 1-byte format for `Picture Type`, and the long names used in the ID3v2 documentation; the default value is `'Cover (front)'`.

```
# Choose APIC with empty description, picture_type='Leaflet page'
my $data = $id3v2->frame_select('APIC', '', 'Leaflet page')
    or die "no expected APIC frame found";
my $format = ( ref $data ? $data->{'MIME type'}
                         : $id3v2->_Data_to_MIME($data) );
# I know what to do with application/pdf only (sp?) and 'image/gif'
die "Do not know what to do with this APIC format: `$format'"
    unless $format eq 'application/pdf' or $format eq 'image/gif';
$data = $data->{_Data} if ref $data;         # handle non-simple frame

# Set APIC frame with empty description (front cover if no other present)
# from content of file.gif
my $data = do { open my $f, '<', 'file.gif' and binmode $f or die;
                undef $/; <$f>};
my $new_frame = $id3v2->frame_select('APIC', '', undef, $data);
```

Frames with multiple "content" fields may be set by providing multiple values to set. Alternatively, one can also `join()` the values with `';'` if the splitting is not ambiguous, e.g., for `POPM RBUF RVRB SYTC`. (For frames `GEOD` and `COMR`, which have a `Description` field, it should be specified among these values.)

```
$id3v2->frame_select("RBUF", undef, undef, $n1, $n2, $n3);
$id3v2->frame_select("RBUF", undef, undef, "$n1;$n2;$n3");
```

(By the way: consider using the method *select_id3v2_frame()* on the "parent" MP3::Tag object instead [see "select_id3v2_frame" in MP3::Tag], or *frame_select_by_descr()*.)

_Data_to_MIME
Internal method to extract MIME type from a string the image file content. Returns `application/octet-stream` for unrecognized data (unless extra TRUE argument is given).

```
$format = $id3v2->_Data_to_MIME($data);
```

Currently, only the first 4 bytes of the string are inspected.

frame_list()
Same as *frame_select()*, but returns the list of found frames, each an array reference `[$N, $f]` with `$N` the 0-based ordinal (among frames with the given short name), and `$f` the contents of a frame.

frame_have()
Same as *frame_select()*, but returns the count of found frames.

frame_select_by_descr()
frame_have_by_descr()
frame_list_by_descr()

```
$c = $id3v2->frame_select_by_descr("COMM(fre,fra,eng,#0)[]");
$t = $id3v2->frame_select_by_descr("TIT2");
     $id3v2->frame_select_by_descr("TIT2", "MyT"); # Set/Change
     $id3v2->frame_select_by_descr("RBUF", $n1, $n2, $n3); # Set/Change
     $id3v2->frame_select_by_descr("RBUF", "$n1;$n2;$n3"); # Set/Change
     $id3v2->frame_select_by_descr("TIT2", undef); # Remove
```

Same as *frame_select()*, *frame_have()*, *frame_list()*, but take one string argument instead of `$fname`, `$descrs`, `$languages`. The argument should be of the form

```
NAME(langs)[descr]
```

Both `(langs)` and `[descr]` parts may be omitted; *langs* should contain comma-separated list of needed languages; no protection by backslashes is needed in *descr*. *frame_select_by_descr()* will return a hash if `(lang)` is omited, but the frame has a language field; likewise for `[descr]`; see below for alternatives.

Remember that when *frame_select_by_descr()* is used for modification, **ALL** found frames are deleted before a new one is added.

(By the way: consider using the method *select_id3v2_frame_by_descr()* on the ''parent'' MP3::Tag object instead; see ''select_id3v2_frame_by_descr'' in MP3::Tag.)

frame_select_by_descr_simple()
Same as *frame_select_by_descr()*, but if no language is given, will not consider the frame as ''complicated'' frame even if it contains a language field.

frame_select_by_descr_simpler()
Same as *frame_select_by_descr_simple()*, but if no `Description` is given, will not consider the frame as ''complicated'' frame even if it contains a `Description` field.

year([@new_year])
Returns the year (TYER/TDRC) from the tag.

Sets TYER and TDRC frames if given the optional arguments `@new_year`. If this is an empty string, the frame is removed.

The format is similar to timestamps of IDv2.4.0, but ranges can be separated by `-` or `--`, and non-contiguous dates are separated by `,` (comma). If periods need to be specified via duration, then one needs to use the ISO 8601 `/`-notation (e.g., see

```
http://www.mcs.vuw.ac.nz/technical/software/SGML/doc/iso8601/ISO8601.html
```

); the `duration/end_timestamp` is not supported.

On output, ranges of timestamps are converted to `-` or `--` separated format depending on whether the timestamps are years, or have additional fields.

If configuration variable `year_is_timestamp` is false, the return value is always the year only (of the first timestamp of a composite timestamp).

Recall that ID3v2.4.0 timestamp has format yyyy−MM−ddTHH:mm:ss (year, ‘‘−’’, month, ‘‘−’’, day, ‘‘T’’, hour (out of 24), ‘‘:’’, minutes, ‘‘:’’, seconds), but the precision may be reduced by removing as many time indicators as wanted. Hence valid timestamps are yyyy, yyyy-MM, yyyy-MM-dd, yyyy-MM-ddTHH, yyyy−MM−ddTHH:mm and yyyy−MM−ddTHH:mm:ss. All time stamps are UTC. For durations, use the slash character as described in 8601, and for multiple noncontiguous dates, use multiple strings, if allowed by the frame definition.

track([$new_track])

Returns the track number (TRCK) from the tag.

Sets TRCK frame if given the optional arguments `@new_track`. If this is an empty string or 0, the frame is removed.

artist([`$new_artist`])

Returns the artist name; it is the first existing frame from the list of

```
TPE1      Lead artist/Lead performer/Soloist/Performing group
TPE2      Band/Orchestra/Accompaniment
TCOM      Composer
TPE3      Conductor
TEXT      Lyricist/Text writer
```

Sets TPE1 frame if given the optional arguments `@new_artist`. If this is an empty string, the frame is removed.

album([`$new_album`])

Returns the album name (TALB) from the tag. If none is found, returns the ‘‘Content group description’’ (TIT1) frame (unless it is considered a part of the title).

Sets TALB frame if given the optional arguments `@new_album`. If this is an empty string, the frame is removed.

genre([`$new_genre`])

Returns the genre string from TCON frame of the tag.

Sets TCON frame if given the optional arguments `@new_genre`. If this is an empty string, the frame is removed.

version()

```
$version = $id3v2->version();
($major, $revision) = $id3v2->version();
```

Returns the version of the ID3v2 tag. It returns a formatted string like ‘‘3.0’’ or an array containing the major part (eg. 3) and revision part (eg. 0) of the version number.

new()

```
$tag = new($mp3fileobj);
```

`new()` needs as parameter a mp3fileobj, as created by `MP3::Tag::File`. `new` tries to find a ID3v2 tag in the mp3fileobj. If it does not find a tag it returns undef. Otherwise it reads the tag header, as well as an extended header, if available. It reads the rest of the tag in a buffer, does unsynchronizing if necessary, and returns a ID3v2−object. At this moment only ID3v2.3 is supported. Any extended header with CRC data is ignored, so no CRC check is done at the moment. The ID3v2−object can be used to extract information from the tag.

Please use

```
$mp3 = MP3::Tag->new($filename);
$mp3->get_tags();                ## to find an existing tag, or
$id3v2 = $mp3->new_tag("ID3v2");  ## to create a new tag
```

instead of using this function directly

SEE ALSO

MP3::Tag, MP3::Tag::ID3v1, MP3::Tag::ID3v2−Data

ID3v2 standard − http://www.id3.org <http://www.id3.org/id3v2−00>, <http://www.id3.org/d3v2.3.0>, <http://www.id3.org/id3v2.4.0−structure>, <http://www.id3.org/id3v2.4.0−frames>,

<http://id3lib.sourceforge.net/id3/id3v2.4.0−changes.txt>.

COPYRIGHT