mutagen

Release 1.23.-1

July 16, 2014

Contents

There are two different ways to load files in Mutagen, but both provide similar interfaces. The first is the `Metadata` API, which deals only in metadata tags. The second is the `FileType` API, which is a superset of the `mutagen` API, and contains information about the audio data itself.

Both Metadata and FileType objects present a dict-like interface to edit tags. FileType objects also have an 'info' attribute that gives information about the song length, as well as per-format information. In addition, both support the load(filename), save(filename), and delete(filename) instance methods; if no filename is given to save or delete, the last loaded filename is used.

This tutorial is only an outline of Mutagen's API. For the full details, you should read the docstrings (pydoc mutagen) or source code.

CHAPTER 1

Easy Examples

The following code loads a file, sets its title, prints all tag data, then saves the file, first on a FLAC file, then on a Musepack file. The code is almost identical.

```
from mutagen.flac import FLAC
audio = FLAC("example.flac")
audio["title"] = "An example"
audio.pprint()
audio.save()

from mutagen.apev2 import APEv2
audio = APEv2("example.mpc")
audio["title"] = "An example"
audio.pprint()
audio.save()
```

The following example gets the length and bitrate of an MP3 file:

```
from mutagen.mp3 import MP3
audio = MP3("example.mp3")
print audio.info.length, audio.info.bitrate
```

The following deletes an ID3 tag from an MP3 file:

```
from mutagen.id3 import ID3
audio = ID3("example.mp3")
audio.delete()
```

CHAPTER 2

Hard Examples: ID3

Unlike Vorbis, FLAC, and APEv2 comments, ID3 data is highly structured. Because of this, the interface for ID3 tags is very different from the APEv2 or Vorbis/FLAC interface. For example, to set the title of an ID3 tag, you need to do the following:

```
from mutagen.id3 import ID3, TIT2
audio = ID3("example.mp3")
audio.add(TIT2(encoding=3, text=u"An example"))
audio.save()
```

If you use the ID3 module, you should familiarize yourself with how ID3v2 tags are stored, by reading the the details of the ID3v2 standard at http://www.id3.org/develop.html.

CHAPTER 3

Easy ID3

Since reading standards is hard, Mutagen also provides a simpler ID3 interface.

```
from mutagen.easyid3 import EasyID3
audio = EasyID3("example.mp3")
audio["title"] = u"An example"
audio.save()
```

Because of the simpler interface, only a few keys can be edited by EasyID3; to see them, use:

```
from mutagen.easyid3 import EasyID3
print EasyID3.valid_keys.keys()
```

By default, mutagen.mp3.MP3 uses the real ID3 class. You can make it use EasyID3 as follows:

```
from mutagen.easyid3 import EasyID3
from mutagen.mp3 import MP3
audio = MP3("example.mp3", ID3=EasyID3)
audio.pprint()
```

CHAPTER 4

Unicode

Mutagen has full Unicode support for all formats. When you assign text strings, we strongly recommend using Python unicode objects rather than str objects. If you use str objects, Mutagen will assume they are in UTF-8.

(This does not apply to strings that must be interpreted as bytes, for example filenames. Those should be passed as str objectss, and will remain str objects within Mutagen.)

CHAPTER 5

Multiple Values

Most tag formats support multiple values for each key, so when you access then (e.g. `audio["title"]`) you will get a list of strings rather than a single one (`[u"An example"]` rather than `u"An example"`). Similarly, you can assign a list of strings rather than a single one.

CHAPTER 6

Changelog

```
1.23 - 2014.05.14
 * tools: Don't crash in misconfigured envs, fall back to utf-8.
 * mp3: Return correct mimetype for MP2 files. (#163)
 * id3: deterministic sorting of frames. (#166)
 * AIFF support (#146, Evan Purkhiser)

1.22 - 2013.09.08
 * Minimum required Python version is now 2.6
 * Online API reference at https://mutagen.readthedocs.org/
 * EasyID3:
   * Fix crash with empty TXXX values. (#135)
 * ID3:
   * id3v2.3 writing support (#85)
   * Add iTunes podcast frames (TGID, TDES, WFED) (#141)
   * Updated id3v1 genre list
 * MP4:
   * add_tags() will not replace existing tags. (#101)
   * Don't ignore tags if parsing unknown atoms fails.
   * Raise on invalid 64bit atom size (#132, Sidnei da Silva)
 * APEv2:
   * Handle invalid tag item count. (#145, Dawid Zamirski)
 * Ogg:
   * Faster parsing of files with large packets.
 * VComment:
   * Preserve text case for field names added through the dict interface (#152)
 * mid3v2:
   * New -e,--escape switch to enable interpretation of escape sequences and
     makes escaping of the colon separator possible. (#159)
 * mid3iconv:
   * Convert COMM frames (#128)

1.21 - 2013.01.30
 * Fix Python 2.3 compatibility (broken in 1.19).
 * Fix many warnings triggered by -3. (#27)
 * mid3v2:
   * Add --TXXX support. (#62, Tim Phipps)
   * Add --POPM support. (#71)
   * Allow setting multiple COMM or TXXX frames with one command line.
 * FLAC:
   * Try to handle corrupt Vorbis comment block sizes. (#52)
   * Try to handle corrupt Picture block sizes (#106, Christoph Reiter)
   * Don't leak file handle with PyPy (#111, Marien Zwart)
 * ID3:
```

```
    * MakeID3v1: Do not generate bad tags when given short dates. (#69)
    * ParseID3v1: Parse short (< 128 byte) tags generated by old Mutagen
      implementations of MakeID3v1, and tags with garbage on the front.
    * pprint: Sort frames by name.
    * Upgrade unknown 2.3 frames (#97, Christoph Reiter)
    * Fix handling of invalid SYLT frames (#105, Christoph Reiter)
  * MP3:
    * Fix error when loading extremely small MP3s. (#72)
    * Fix rounding error in CBR length calculation (#93, Christoph Reiter)
  * Use 'open' rather than 'file' everywhere. (#74, Dan Callahan)
  * mid3iconv:
    * Accurately copy QL-style frame encoding behavior. (#75)
    * Skip unopenable files. (#79)
  * ID3FileType:
    * Remember which tag type load() was called with even if the file
      doesn't yet have any ID3 tags. (#89)
  * VComment:
    * Prevent MemoryError when parsing invalid header (#112, Jyrki Pulliainen)
  * ASF:
    * Don't corrupt files on the second save() call (#81, Christoph Reiter)
    * Always store GUID objects in the MetadataLibraryBlock (#81)
  * OggTheora: Fix length/bitrate calculation. (#99, Christoph Reiter)
  * MP4:
    * Less strict MP4 covr atom parsing. (#86, Lukáš Lalinský)
    * Support atoms that extend to the end of the file. (#109, Sidnei da Silva)
    * Preserve freeform format flags (#103, Christoph Reiter)
  * OggOpus support. (#115, Christoph Reiter)
  * Musepack:
    * Fix SV7 bitrate calculation (#7, Christoph Reiter)
    * Support SV8 (#7, Christoph Reiter)

1.20 - 2010.08.04
  * ASF: Don't store blocks over 64K in the MetadataObject block;
    use the MetadataLibraryBlock instead. (#60, Lukáš Lalinský)
  * ID3: Faster parsing of files with lots of padding. (#65, Christoph Reiter)
  * FLAC: Correct check for audio data start. (#67)

1.19 - 2010.02.18
  * ID3:
    * POPM: 'count' is optional; the attribute may not exist. (#33)
    * TimeStampTextFrame: Fix a TypeError in unicode comparisons. (#43)
    * MakeID3v1: Translate TYER into ID3v1 year if TDRC is not present. (#42)
  * mid3v2:
    * Allow --delete followed by --frame, and --genre 1 --genre 2. (#37)
    * Add --quiet and --verbose flags. (#40)
  * moggsplit: --m3u option to write an M3U playlist of the new files. (#39)
  * mid3iconv: Fix crash when processing TCML or TIPL frames. (#41)
  * VCommentDict: Correctly normalize key names for .keys() iterator. (#45)
  * MP3: Correct length calculation for MPEG-2 files. (#46)
  * oggflac: Fix typo in docstring. (#53)
  * EasyID3: Force UTF-8 encoding. (#54)
  * EasyMP4: Fix 'genre' translation. (#56)

1.18 - 2009.10.22
  * ASF:
    * Distinguish between empty and absent tag values in
      ContentDescriptionObjects. (#29)
  * mid3iconv:
```

```
   * Fix a crash when processing empty (invalid) text frames.
 * MAJOR API INCOMPATIBILITY!!!!
   * EasyID3FileType is now in mutagen.easyid3, not mutagen.id3. This
     change was necessary to restore API compatibility with 1.16, as
     1.17 accidentally contained a circular import preventing
     mutagen.easyid3 from importing by itself. (#32)

1.17 - 2009.10.07
 * ID3:
   * Support for the iTunes non-standard TSO2 and TSOC frames.
   * Attempt to recover from bad SYLT frames. (#2)
   * Attempt to recover from faulty extended header flags. (#4, #21)
   * Fix a bug in ID3v2.4 footer flag detection, (#5)
 * MP4:
   * Don't fail or double-encode UTF-8 strings when given a str.
   * Don't corrupt 64 bit atom sizes when resizing atoms. (#17)
 * EasyID3:
   * Extension API for defining new "easy" tags at runtime.
   * Support for many, many more tags.
 * OggVorbis, OggSpeex: Handle bitrates below 0 as per the spec. (#30)
 * EasyMP4: Like EasyID3, but for iTunes MPEG-4 files.
 * mutagen.File: New 'easy=True' argument to create new EasyMP3, EasyMP4,
   EasyTrueAudio, and EasyID3FileType instances.

1.16 - 2009.06.15
 * Website / code repository move.
 * Bug Fixes:
   * EasyID3: Invalid keys now raise KeyError (and ValueError).
   * mutagen.File: .flac files with an ID3 tag will be opened as FLAC.
 * MAJOR API INCOMPATIBILITY!!!!
   * Python 2.6 has required us to rename the .format attribute of M4A/MP4
     cover atoms, because it conflicts with the new str.format method.
     It has been renamed .imageformat.

1.15 - 2008.12.01
 * Bug Fixes:
   * mutagen.File: Import order no longer affects what type is returned.
   * mutagen.id3: Compression of frames is now disabled.
   * mutagen.flac.StreamInfo: Fix channel mask (support channels > 2). [35]
   * mutagen.mp3: Ignore Xing headers if they are obviously wrong.

1.14 - 2008.05.31
 * Bug Fixes:
   * MP4/M4A: Fixed saving of atoms with 64-bit size on 64-bit platforms.
   * MP4: Conversion of 'gnre' atoms to '\xa9gen' text atoms now correctly
     produces a list of string values, not just a single value.
   * ID3: Broken RVA2 frames are now discarded. (Vladislav Naumov)
   * ID3: Use long integers when appropriate.
  * VCommentDict: Raise UnicodeEncodeErrors when trying to use a Unicode
    key that is not valid ASCII; keys are also normalized to ASCII
    str objects. (Forest Bond)
 * Tests:
   * FLAC: Use 2**64 instead of 2**32 to test overflow behavior.

1.13 - 2007.12.03
 * Bug Fixes:
   * FLAC: Raise IOError, instead of UnboundLocalError, when trying
     to open a non-existant file. (Lukáš Lalinský, Debian #448734)
```

```
    * Throw out invalid frames when upgrading from 2.3 to 2.4.
    * Fixed reading of Unicode strings from ASF files on big-endian
      platforms.
  * TCP/TCMP support. (Debian #452231)
  * Faster implementation of file-writing when mmap fails, and
    exclusive advisory locking when available.
  * Test cases to ensure Mutagen is not vulnerable to CVE-2007-4619.
    It is not now, nor was it ever.
  * Use VBRI header to calculate length of VBR MP3 files if the Xing
    header is not found.

1.12 - 2007.08.04
  * Write important ID3v2 frames near the start. (Lukáš Lalinský)
  * Clean up distutils functions.

1.11 - 2007.04.26
  * New Features:
    * mid3v2 can now set URL frames. (Vladislav Naumov)
    * Musepack: Skip ID3v2 tags. (Lukáš Lalinský)
  * Bug Fixes:
    * mid3iconv: Skip all timestamp frames. (Lukáš Lalinský)
    * WavPack: More accurate length calculation. ('ak')
    * PairedTextFrame: Fix typo in documentation. (Lukáš Lalinský)
    * ID3: Fixed incorrect TDAT conversion. The format is DDMM, not
      MMDD. (Lukáš Lalinský)
  * API:
    * Metadata no longer inherits from dict.
    * Relatedly, the MRO has changed on several types.
    * More documentation for MP4 atoms. (Lukáš Lalinský)
    * Prefer MP3 for files with unknown extensions and ID3 tags.

1.10.1 - 2007.01.23
  * Bug Fixes:
    * Documentation mentions ASF support.
    * APEv2 flags and valid keys are fixed.
    * Tests pass on Python 2.3 again.

1.10 - 2007.01.21
  * New Features:
    * FLAC: Skip ID3 tags. Added option to delete them on save.
    * EncodedTextSpec: Make private members more private.
    * Corrupted Oggs generated by GStreamer (e.g. Sound Juicer) can be read.
    * FileTypes have a .mime attribute which is a list of likely MIME types
      for the file.
    * ASF (WMA/WMV) support.
  * Bug Fixes:
    * ID3: Fixed reading of v2.3 tags with unsynchronized data.
    * ID3: The data length indicator for compressed tags is written
      as a synch-safe integer.

1.9 - 2006.12.09
  * New Features:
    * OptimFROG support.
    * New mutagen.mp4 module with support for multiple data fields per atom
      and more compatible tag saving implementation.
    * Support for embedded pictures in FLAC files (new in FLAC 1.1.3).
  * mutagen.m4a is deprecated in favor of mutagen.mp4.
```

```
1.8 - 2006.10.02
 * New Features:
   * MonkeysAudio support. (#851, Lukáš Lalinský)
   * APEv2 support on Python 2.5; see API-NOTES. (#852)

1.7.1 - 2006.09.24
 * Bug Fixes:
   * Expose full ID3 tag size as .size. (#848)

 * New Features:
   * Musepack Replay Gain data is available in SV7 files.

1.7 - 2006.09.15
 * Bug Fixes:
   * Trying to save an empty tag deletes it. (#813)
   * The semi-public API removal mentioned in 1.6's API-NOTES happened.
   * Stricter frame ID validation. (#830, Lukáš Lalinský)
   * Use os.path.devnull on Win32/Mac OS X. (#831, Lukáš Lalinský)

 * New Features:
   * FLAC cuesheet and seektable support. (#791, Nuutti Kotivuori)
   * Kwargs can be passed to ID3 constructors. (#824, Lukáš Lalinský)
   * mutagen.musepack: Read/tag Musepack files. (#825, Lukáš Lalinský)

 * Tools:
   * mutagen-inspect responds immediately to keyboard interrupts.

1.6 - 2006.08.09
 * Bug Fixes:
   * IOError rather than NameError is raised when File succeeds in
     typefinding but fails in stream parsing.
   * errors= kwarg is correctly interpreted for FLAC tags now.
   * Handle struct.pack API change in Python 2.5b2. (SF #1530559)
   * Metadata 'load' methods always reset in-memory tags.
   * Metadata 'delete' methods always clear in-memory tags.

 * New Features:
   * Vorbis comment vendor strings include the Mutagen version.
   * mutagen.id3: Read ASPI, ETCO, SYTC, MLLT, EQU2, and LINK frames.
   * mutagen.m4a: Read/tag MPEG-4 AAC audio files with iTunes tags. (#681)
   * mutagen.oggspeex: Read/tag Ogg Speex files.
   * mutagen.trueaudio: Read/tag True Audio files.
   * mutagen.wavpack: Read/tag WavPack files.

 * Tools:
   * mid3v2: --delete-frames. (#635)

1.5.1 - 2006.06.26
 * Bug Fixes:
   * Handle ENODEV from mmap (e.g. on fuse+sshfs).
   * Reduce test rerun time.

1.5 - 2006.06.20
 * Bug Fixes:
   * APEv2
      * Invalid Lyrics3v2 tags are ignored/overwritten.
      * Binary values are autodetected as documented.
   * OggVorbis, OggFLAC:
```

```
      * Write when the setup packet spans multiple pages.
      * Zero granule position for header packets.

 * New Features:
   * mutagen.oggtheora: Read/tag Ogg Theora files.
   * Test Ogg formats with ogginfo, if present.

1.4 - 2006.06.03
 * Bug Fixes:
   * EasyID3: Fix tag["key"] = "string" handler. (#693)
   * APEv2:
     * Skip Lyrics3v2 tags. (Miguel Angel Alvarez)
     * Avoid infinite loop on malformed tags at the start of the file.
   * Proper ANSI semantics for file positioning. (#707)

 * New Features:
   * VComment: Handle malformed Vorbis comments when errors='ignore' or
     errors='replace' is passed to VComment#load.
     (Bastian Kleineidam, #696)
   * Test running is now controlled through setup.py (./setup.py test).
   * Test coverage data can be generated (./setup.py coverage).
   * Considerably more test coverage.

1.3 - 2006.05.29
 * New Features:
   * mutagen.File: Automatic file type detection.
   * mutagen.ogg: Generic Ogg stream parsing. (#612)
   * mutagen.oggflac: Read/tag Ogg FLAC files.
   * mutagen.oggvorbis no longer depends on pyvorbis.
   * ID3: SYLT support. (#672)

1.2 - 2006.04.23
 * Bug Fixes:
   * MP3: Load files with zeroed Xing headers. (#626)
   * ID3: Upgrade ID3v2.2 PIC tags to ID3v2.4 APIC tags properly.
   * Tests exit with non-zero status if any have failed.
   * Full dict protocol support for VCommentDict, FileType, and APEv2 objects.

 * New features:
   * mutagen.oggvorbis gives pyvorbis a Mutagen-like API.
   * mutagen.easyid3 makes simple ID3 tag changes easier.
   * A brief TUTORIAL was added.

 * Tools:
   * mid3iconv, a clone of id3iconv, was added by Emfox Zhou. (#605)

1.1 - 2006.04.04
 * ID3:
  * Frame and Spec objects are not hashable.
  * COMM, USER: Accept non-ASCII (completely invalid) language codes.
  * Enable redundant data length bit for compressed frames.

1.0 - 2006.03.13
 * mutagen.FileType, an abstract container for tags and stream information.
 * MP3: A new FileType subclass for MPEG audio files.
 * FLAC:
  * Add FLAC#delete.
  * Raise correct exception when saving to a non-FLAC file.
```

```
  * FLAC#vc is deprecated in favor of FLAC#tags.
 * VComment (used by FLAC):
  * VComment#clear to clear all tags.
  * VComment#as_dict to return a dict of the tags.
 * ID3:
  * Fix typos in PRIV#_pprint, OWNE#_pprint, UFID#_pprint.
 * mutagen-pony: Try finding lengths as well as tags.
 * mutagen-inspect: Output stream information with tags.

0.9 - 2006.02.21
 * Initial release.
```

CHAPTER 7

API Notes

This file documents deprecated parts of the Mutagen API. New code should not use these parts, and several months after being added here, they may be removed. Note that we do not intend to ever deprecate or remove large portions of the API. All of these are corner cases that arose from when Mutagen was still part of Quod Libet, and should never be encountered in normal use.

7.1 General

FileType constructors require a filename. However, the 'delete' and 'save' methods should not be called with one.

No modules, types, functions, or attributes beginning with '_' are considered public API. These can and do change drastically between Mutagen versions. This is the standard Python way of marking a function protected or private.

Mutagen's goal is to adhere as closely as possible to published specifications. If you try to abuse Mutagen to make it write things in a non-standard fashion, Joe will update Mutagen to break your program. If you want to do nonstandard things, write your own broken library.

7.2 FLAC

The 'vc' attribute predates the FileType API and has been deprecated since Mutagen 0.9; this also applies to the 'add_vc' method. The standard 'tags' attribute and 'add_tags' method should be used instead.

7.3 ID3

None of the Spec objects are considered part of the public API.

7.4 APEv2

Python 2.5 forced an API change in the APEv2 reading code. Some things which were case-insensitive are now case-sensitive. For example, given:

```
tag = APEv2()
tag["Foo"] = "Bar"
print "foo" in tag.keys()
```

Mutagen 1.7.1 and earlier would print "True", as the keys were a str subclass that compared case-insensitively. However, Mutagen 1.8 and above print "False", as the keys are normal strings.

```
print "foo" in tag
```

Still prints "True", however, as __getitem__, __delitem__, and __setitem__ (and so any operations on the dict itself) remain case-insensitive.

As of 1.10.1, Mutagen no longer allows non-ASCII keys in APEv2 tags. This is in accordance with the APEv2 standard. A KeyError is raised if you try.

7.5 M4A

mutagen.m4a is deprecated. You should use mutagen.mp4 instead.

7.6 MP4

There is no MPEG-4 iTunes metadata standard. Mutagen's features are known to lead to problems in other implementations. For example, FAAD will crash when reading a file with multiple "tmpo" atoms. iTunes itself is our main compatibility target.

Python 2.6 forced an API change in the MP4 (and M4A) code, by introducing the str.format instance method. Previously the cover image format was available via the .format attribute; it is now available via the .imageformat attribute. On versions of Python prior to 2.6, it is also still available as .format.

CHAPTER 8

Compatibility / Bugs

Mutagen writes ID3v2.4 tags which id3lib cannot read. If you enable ID3v1 tag saving (pass v1=2 to ID3.save), id3lib will read those.

iTunes has a bug in its handling of very large ID3 tags (such as tags that contain an attached picture). Mutagen can read tags from iTunes, but iTunes may not be able to read tags written by Quod Libet.

Mutagen has had several bugs in correct sync-safe parsing and writing of data length flags in ID3 tags. This will only affect files with very large or compressed ID3 frames (e.g. APIC). As of 1.10 we believe them all to be fixed.

Prior to 1.10.1, Mutagen wrote an incorrect flag for APEv2 tags that claimed they did not have footers. This has been fixed, however it means that all APEv2 tags written before 1.10.1 are corrupt.

Prior to 1.16, the MP4 cover atom used a .format attribute to indicate the image format (JPEG/PNG). Python 2.6 added a str.format method which conflicts with this. 1.17 provides .imageformat when running on any version, and still provides .format when running on a version before 2.6.

Mutagen 1.18 moved EasyID3FileType to mutagen.easyid3, rather than mutagen.id3, which was used in 1.17. Keeping in mutagen.id3 caused circular import problems. To import EasyID3FileType correctly in 1.17 and 1.18 or later:

```
import mutagen.id3
try:
    from mutagen.easyid3 import EasyID3FileType
except ImportError:
    # Mutagen 1.17.
    from mutagen.id3 import EasyID3FileType
```

Mutagen 1.19 made it possible for POPM to have no 'count' attribute. Previously, files that generated POPM frames of this type would fail to load at all.

When given date frames less than four characters long (which are already outside the ID3v2 specification), Mutagen 1.20 and earlier would write invalid ID3v1 tags that were too short. Mutagen 1.21 will parse these and fix them if it finds them while saving.

CHAPTER 9

API

9.1 Main Module

Mutagen aims to be an all purpose multimedia tagging library.

```
import mutagen.[format]
metadata = mutagen.[format].Open(filename)
```

metadata acts like a dictionary of tags in the file. Tags are generally a list of string-like values, but may have additional methods available depending on tag or format. They may also be entirely different objects for certain keys, again depending on format.

mutagen.**File** (*filename*, *options=None*, *easy=False*)
Guess the type of the file and try to open it.

The file type is decided by several things, such as the first 128 bytes (which usually contains a file type identifier), the filename extension, and the presence of existing tags.

If no appropriate type could be found, None is returned.

Parameters

- **options** – Sequence of `FileType` implementations, defaults to all included ones.
- **easy** – If the easy wrappers should be returnd if available. For example `EasyMP3` instead of `MP3`.

mutagen.**version = (1, 23, -1)**
Version tuple.

mutagen.**version_string = '1.23.-1'**
Version string.

9.1.1 Base Classes

class mutagen.**FileType** (*filename*)
Bases: `mutagen._util.DictMixin`

An abstract object wrapping tags and audio stream information.

Attributes:

•info – stream information (length, bitrate, sample rate)

•tags – metadata tags, if any

Each file format has different potential tags and stream information.

FileTypes implement an interface very similar to Metadata; the dict interface, save, load, and delete calls on a FileType call the appropriate methods on its tag data.

delete()
Remove tags from a file.

save()
Save metadata tags.

add_tags()
Adds new tags to the file.

Raises if tags already exist.

mime
A list of mime types

pprint()
Print stream information and comment key=value pairs.

class mutagen.**Metadata**(**args*, ***kwargs*)
An abstract dict-like object.

Metadata is the base class for many of the tag objects in Mutagen.

delete()
Remove tags from a file.

save()
Save changes to a file.

9.1.2 Internal Classes

Utility classes for Mutagen.

You should not rely on the interfaces here being stable. They are intended for internal use in Mutagen only.

class mutagen._util.**DictMixin**
Implement the dict API using keys() and __*item__ methods.

Similar to UserDict.DictMixin, this takes a class that defines __getitem__, __setitem__, __delitem__, and keys(), and turns it into a full dict-like object.

UserDict.DictMixin is not suitable for this purpose because it's an old-style class.

This class is not optimized for very large dictionaries; many functions have linear memory requirements. I recommend you override some of these functions if speed is required.

class mutagen._util.**DictProxy**(**args*, ***kwargs*)
Bases: mutagen._util.DictMixin

9.2 ID3v2

ID3v2 reading and writing.

This is based off of the following references:

- http://id3.org/id3v2.4.0-structure

- http://id3.org/id3v2.4.0-frames
- http://id3.org/id3v2.3.0
- http://id3.org/id3v2-00
- http://id3.org/ID3v1

Its largest deviation from the above (versions 2.3 and 2.2) is that it will not interpret the / characters as a separator, and will almost always accept null separators to generate multi-valued text frames.

Because ID3 frame structure differs between frame types, each frame is implemented as a different class (e.g. TIT2 as mutagen.id3.TIT2). Each frame's documentation contains a list of its attributes.

Since this file's documentation is a little unwieldy, you are probably interested in the `ID3` class to start with.

9.2.1 ID3 Frames

Frame Base Classes

class `mutagen.id3.`**`Frame`**
Bases: `object`

Fundamental unit of ID3 data.

ID3 tags are split into frames. Each frame has a potentially different structure, and so this base class is not very featureful.

`FrameID`
ID3v2 three or four character frame ID

`HashKey`
An internal key used to ensure frame uniqueness in a tag

classmethod `fromData`(*id3*, *tflags*, *data*)
Construct this ID3 frame from raw string data.

`pprint`()
Return a human-readable representation of the frame.

class `mutagen.id3.`**`BinaryFrame`**(*data='None'*)
Bases: `mutagen._id3frames.Frame`

Binary data

The 'data' attribute contains the raw byte string.

class `mutagen.id3.`**`FrameOpt`**
Bases: `mutagen._id3frames.Frame`

A frame with optional parts.

Some ID3 frames have optional data; this class extends Frame to provide support for those parts.

class `mutagen.id3.`**`PairedTextFrame`**(*encoding=None*, *people=*[ ])
Bases: `mutagen._id3frames.Frame`

Paired text strings.

Some ID3 frames pair text strings, to associate names with a more specific involvement in the song. The 'people' attribute of these frames contains a list of pairs:

```
[['trumpet', 'Miles Davis'], ['bass', 'Paul Chambers']]
```

Like text frames, these frames also have an encoding attribute.

class mutagen.id3.**TextFrame**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.Frame

Text strings.

Text frames support casts to unicode or str objects, as well as list-like indexing, extend, and append.

Iterating over a TextFrame iterates over its strings, not its characters.

Text frames have a 'text' attribute which is the list of strings, and an 'encoding' attribute; 0 for ISO-8859 1, 1 UTF-16, 2 for UTF-16BE, and 3 for UTF-8. If you don't want to worry about encodings, just set it to 3.

append(*value*)
Append a string.

extend(*value*)
Extend the list by appending all strings from the given list.

class mutagen.id3.**UrlFrame**(*url=u'None'*)
Bases: mutagen._id3frames.Frame

A frame containing a URL string.

The ID3 specification is silent about IRIs and normalized URL forms. Mutagen assumes all URLs in files are encoded as Latin 1, but string conversion of this frame returns a UTF-8 representation for compatibility with other string conversions.

The only sane way to handle URLs in MP3s is to restrict them to ASCII.

class mutagen.id3.**NumericPartTextFrame**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Multivalue numerical text strings.

These strings indicate 'part (e.g. track) X of Y', and unary plus returns the first value:

```
frame = TRCK('4/15')
track = +frame # track == 4
```

class mutagen.id3.**NumericTextFrame**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Numerical text strings.

The numeric value of these frames can be gotten with unary plus, e.g.:

```
frame = TLEN('12345')
length = +frame
```

class mutagen.id3.**TimeStampTextFrame**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

A list of time stamps.

The 'text' attribute in this frame is a list of ID3TimeStamp objects, not a list of strings.

class mutagen.id3.**UrlFrameU**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

ID3v2.3/4 Frames

class mutagen.id3.**AENC**(*owner=u'None'*, *preview_start=None*, *preview_length=None*)
Bases: mutagen._id3frames.FrameOpt

Audio encryption.

Attributes:

- owner – key identifying this encryption type
- preview_start – unencrypted data block offset
- preview_length – number of unencrypted blocks
- data – data required for decryption (optional)

Mutagen cannot decrypt files.

class mutagen.id3.**APIC**(*encoding=None*, *mime=u'None'*, *type=None*, *desc=u'None'*, *data='None'*)
Bases: mutagen._id3frames.Frame

Attached (or linked) Picture.

Attributes:

- encoding – text encoding for the description
- mime – a MIME type (e.g. image/jpeg) or '–>' if the data is a URI
- type – the source of the image (3 is the album front cover)
- desc – a text description of the image
- data – raw image data, as a byte string

Mutagen will automatically compress large images when saving tags.

class mutagen.id3.**ASPI**(*S=None*, *L=None*, *N=None*, *b=None*, *Fi=None*)
Bases: mutagen._id3frames.Frame

Audio seek point index.

Attributes: S, L, N, b, and Fi. For the meaning of these, see the ID3v2.4 specification. Fi is a list of integers.

class mutagen.id3.**COMM**(*encoding=None*, *lang=None*, *desc=u'None'*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

User comment.

User comment frames have a descrption, like TXXX, and also a three letter ISO language code in the 'lang' attribute.

class mutagen.id3.**COMR**(*encoding=None*, *price=u'None'*, *valid_until=None*, *contact=u'None'*, *format=None*, *seller=u'None'*, *desc=u'None'*)
Bases: mutagen._id3frames.FrameOpt

Commercial frame.

class mutagen.id3.**ENCR**(*owner=u'None'*, *method=None*, *data='None'*)
Bases: mutagen._id3frames.Frame

Encryption method registration.

The standard does not allow multiple ENCR frames with the same owner or the same method. Mutagen only verifies that the owner is unique.

class mutagen.id3.**EQU2**(*method=None*, *desc=u'None'*, *adjustments=None*)
Bases: mutagen._id3frames.Frame

Equalisation (2).

Attributes: method – interpolation method (0 = band, 1 = linear) desc – identifying description adjustments – list of (frequency, vol_adjustment) pairs

class mutagen.id3.**ETCO**(*format=None*, *events=None*)
Bases: mutagen._id3frames.Frame

Event timing codes.

class mutagen.id3.**GEOB**(*encoding=None*, *mime=u'None'*, *filename=u'None'*, *desc=u'None'*, *data='None'*)
Bases: mutagen._id3frames.Frame

General Encapsulated Object.

A blob of binary data, that is not a picture (those go in APIC).

Attributes:

- encoding – encoding of the description
- mime – MIME type of the data or '–>' if the data is a URI
- filename – suggested filename if extracted
- desc – text description of the data
- data – raw data, as a byte string

class mutagen.id3.**GRID**(*owner=u'None'*, *group=None*)
Bases: mutagen._id3frames.FrameOpt

Group identification registration.

class mutagen.id3.**IPLS**(*encoding=None*, *people=[]*)
Bases: mutagen._id3frames.TIPL

Involved People List

class mutagen.id3.**LINK**(*frameid=None*, *url=u'None'*)
Bases: mutagen._id3frames.FrameOpt

Linked information.

Attributes:

- frameid – the ID of the linked frame
- url – the location of the linked frame
- data – further ID information for the frame

class mutagen.id3.**MCDI**(*data='None'*)
Bases: mutagen._id3frames.BinaryFrame

Binary dump of CD's TOC

class mutagen.id3.**MLLT**(*frames=None*, *bytes=None*, *milliseconds=None*, *bits_for_bytes=None*, *bits_for_milliseconds=None*, *data='None'*)
Bases: mutagen._id3frames.Frame

MPEG location lookup table.

This frame's attributes may be changed in the future based on feedback from real-world use.

class mutagen.id3.**OWNE**(*encoding=None*, *price=u'None'*, *date=None*, *seller=u'None'*)
Bases: mutagen._id3frames.Frame

Ownership frame.

class mutagen.id3.**PCNT**(*count=None*)
Bases: mutagen._id3frames.Frame

Play counter.

The 'count' attribute contains the (recorded) number of times this file has been played.

This frame is basically obsoleted by POPM.

class mutagen.id3.**POPM**(*email=u'None'*, *rating=None*)
Bases: mutagen._id3frames.FrameOpt

Popularimeter.

This frame keys a rating (out of 255) and a play count to an email address.

Attributes:

- email – email this POPM frame is for
- rating – rating from 0 to 255
- count – number of times the files has been played (optional)

class mutagen.id3.**POSS**(*format=None*, *position=None*)
Bases: mutagen._id3frames.Frame

Position synchronisation frame

Attribute:

- format – format of the position attribute (frames or milliseconds)
- position – current position of the file

class mutagen.id3.**PRIV**(*owner=u'None'*, *data='None'*)
Bases: mutagen._id3frames.Frame

Private frame.

class mutagen.id3.**RBUF**(*size=None*)
Bases: mutagen._id3frames.FrameOpt

Recommended buffer size.

Attributes:

- size – recommended buffer size in bytes
- info – if ID3 tags may be elsewhere in the file (optional)
- offset – the location of the next ID3 tag, if any

Mutagen will not find the next tag itself.

class mutagen.id3.**RVA2**(*desc=u'None'*, *channel=None*, *gain=None*, *peak=None*)
Bases: mutagen._id3frames.Frame

Relative volume adjustment (2).

This frame is used to implemented volume scaling, and in particular, normalization using ReplayGain.

Attributes:

- desc – description or context of this adjustment
- channel – audio channel to adjust (master is 1)
- gain – a + or - dB gain relative to some reference level
- peak – peak of the audio as a floating point number, [0, 1]

When storing ReplayGain tags, use descriptions of 'album' and 'track' on channel 1.

class mutagen.id3.**RVRB**(*left=None*, *right=None*, *bounce_left=None*, *bounce_right=None*, *feedback_ltl=None*, *feedback_ltr=None*, *feedback_rtr=None*, *feedback_rtl=None*, *premix_ltr=None*, *premix_rtl=None*)
Bases: mutagen._id3frames.Frame

Reverb.

class mutagen.id3.**SEEK**(*offset=None*)
Bases: mutagen._id3frames.Frame

Seek frame.

Mutagen does not find tags at seek offsets.

class mutagen.id3.**SIGN**(*group=None*, *sig='None'*)
Bases: mutagen._id3frames.Frame

Signature frame.

class mutagen.id3.**SYLT**(*encoding=None*, *lang=None*, *format=None*, *type=None*, *desc=u'None'*, *text=None*)
Bases: mutagen._id3frames.Frame

Synchronised lyrics/text.

class mutagen.id3.**SYTC**(*format=None*, *data='None'*)
Bases: mutagen._id3frames.Frame

Synchronised tempo codes.

This frame's attributes may be changed in the future based on feedback from real-world use.

class mutagen.id3.**TALB**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Album

class mutagen.id3.**TBPM**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericTextFrame

Beats per minute

class mutagen.id3.**TCMP**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericTextFrame

iTunes Compilation Flag

class mutagen.id3.**TCOM**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Composer

class mutagen.id3.**TCON**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Content type (Genre)

ID3 has several ways genres can be represented; for convenience, use the 'genres' property rather than the 'text' attribute.

genres
A list of genres parsed from the raw text data.

class mutagen.id3.**TCOP**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Copyright (c)

class mutagen.id3.**TDAT**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Date of recording (DDMM)

class mutagen.id3.**TDEN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TimeStampTextFrame

Encoding Time

class mutagen.id3.**TDLY**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericTextFrame

Audio Delay (ms)

class mutagen.id3.**TDOR**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TimeStampTextFrame

Original Release Time

class mutagen.id3.**TDRC**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TimeStampTextFrame

Recording Time

class mutagen.id3.**TDRL**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TimeStampTextFrame

Release Time

class mutagen.id3.**TDTG**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TimeStampTextFrame

Tagging Time

class mutagen.id3.**TENC**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Encoder

class mutagen.id3.**TEXT**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Lyricist

class mutagen.id3.**TFLT**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

File type

class mutagen.id3.**TIME**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Time of recording (HHMM)

class mutagen.id3.**TIPL**(*encoding=None*, *people=[]*)
Bases: mutagen._id3frames.PairedTextFrame

Involved People List

class mutagen.id3.**TIT1**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Content group description

class mutagen.id3.**TIT2**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Title

class mutagen.id3.**TIT3**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Subtitle/Description refinement

class mutagen.id3.**TKEY**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Starting Key

class mutagen.id3.**TLAN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Audio Languages

class mutagen.id3.**TLEN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericTextFrame

Audio Length (ms)

class mutagen.id3.**TMCL**(*encoding=None*, *people=[]*)
Bases: mutagen._id3frames.PairedTextFrame

Musicians Credits List

class mutagen.id3.**TMED**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Source Media Type

class mutagen.id3.**TMOO**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Mood

class mutagen.id3.**TOAL**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Original Album

class mutagen.id3.**TOFN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Original Filename

class mutagen.id3.**TOLY**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Original Lyricist

class mutagen.id3.**TOPE**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Original Artist/Performer

class mutagen.id3.**TORY**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericTextFrame

Original Release Year

class mutagen.id3.**TOWN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Owner/Licensee

class mutagen.id3.**TPE1**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Lead Artist/Performer/Soloist/Group

class mutagen.id3.**TPE2**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Band/Orchestra/Accompaniment

class mutagen.id3.**TPE3**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Conductor

class mutagen.id3.**TPE4**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Interpreter/Remixer/Modifier

class mutagen.id3.**TPOS**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericPartTextFrame

Part of set

class mutagen.id3.**TPRO**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Produced (P)

class mutagen.id3.**TPUB**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Publisher

class mutagen.id3.**TRCK**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.NumericPartTextFrame

Track Number

class mutagen.id3.**TRDA**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Recording Dates

class mutagen.id3.**TRSN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TextFrame

Internet Radio Station Name

class mutagen.id3.**TRSO**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

Internet Radio Station Owner

class mutagen.id3.**TSIZ**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.NumericTextFrame

Size of audio data (bytes)

class mutagen.id3.**TSO2**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

iTunes Album Artist Sort

class mutagen.id3.**TSOA**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

Album Sort Order key

class mutagen.id3.**TSOC**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

iTunes Composer Sort

class mutagen.id3.**TSOP**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

Perfomer Sort Order key

class mutagen.id3.**TSOT**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

Title Sort Order key

class mutagen.id3.**TSRC**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

International Standard Recording Code (ISRC)

class mutagen.id3.**TSSE**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

Encoder settings

class mutagen.id3.**TSST**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

Set Subtitle

class mutagen.id3.**TXXX**(*encoding=None*, *desc=u'None'*, *text=[ ]*)
Bases: mutagen._id3frames.TextFrame

User-defined text data.

TXXX frames have a 'desc' attribute which is set to any Unicode value (though the encoding of the text and the description must be the same). Many taggers use this frame to store freeform keys.

class mutagen.id3.**TYER**(*encoding=None*, *text=[ ]*)
Bases: mutagen._id3frames.NumericTextFrame

Year of recording

class mutagen.id3.**UFID**(*owner=u'None'*, *data='None'*)
Bases: mutagen._id3frames.Frame

Unique file identifier.

Attributes:

- owner – format/type of identifier
- data – identifier

class mutagen.id3.**USER**(*encoding=None*, *lang=None*, *text=u'None'*)
Bases: mutagen._id3frames.Frame

Terms of use.

Attributes:

- encoding – text encoding
- lang – ISO three letter language code
- text – licensing terms for the audio

class mutagen.id3.**USLT**(*encoding=None*, *lang=None*, *desc=u'None'*, *text=u'None'*)
Bases: mutagen._id3frames.Frame

Unsynchronised lyrics/text transcription.

Lyrics have a three letter ISO language code ('lang'), a description ('desc'), and a block of plain text ('text').

class mutagen.id3.**WCOM**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrameU

Commercial Information

class mutagen.id3.**WCOP**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

Copyright Information

class mutagen.id3.**WOAF**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

Official File Information

class mutagen.id3.**WOAR**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrameU

Official Artist/Performer Information

class mutagen.id3.**WOAS**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

Official Source Information

class mutagen.id3.**WORS**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

Official Internet Radio Information

class mutagen.id3.**WPAY**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

Payment Information

class mutagen.id3.**WPUB**(*url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

Official Publisher Information

class mutagen.id3.**WXXX**(*encoding=None*, *desc=u'None'*, *url=u'None'*)
Bases: mutagen._id3frames.UrlFrame

User-defined URL data.

Like TXXX, this has a freeform description associated with it.

ID3v2.2 Frames

class mutagen.id3.**BUF**(*size=None*)
Bases: mutagen._id3frames.RBUF

Recommended buffer size

class mutagen.id3.**CNT**(*count=None*)
Bases: mutagen._id3frames.PCNT

Play counter

class mutagen.id3.**COM**(*encoding=None*, *lang=None*, *desc=u'None'*, *text=[]*)
Bases: mutagen._id3frames.COMM

Comment

class mutagen.id3.**CRA**(*owner=u'None'*, *preview_start=None*, *preview_length=None*)
Bases: mutagen._id3frames.AENC

Audio encryption

class mutagen.id3.**CRM**(*owner=u'None'*, *desc=u'None'*, *data='None'*)
Bases: mutagen._id3frames.Frame

Encrypted meta frame

class mutagen.id3.**ETC**(*format=None*, *events=None*)
Bases: mutagen._id3frames.ETCO

Event timing codes

class mutagen.id3.**GEO**(*encoding=None*, *mime=u'None'*, *filename=u'None'*, *desc=u'None'*, *data='None'*)
Bases: mutagen._id3frames.GEOB

General Encapsulated Object

class mutagen.id3.**IPL**(*encoding=None*, *people=[]*)
Bases: mutagen._id3frames.IPLS

Involved people list

class mutagen.id3.**LNK**(*frameid=None*, *url=u'None'*)
Bases: mutagen._id3frames.LINK

Linked information

class mutagen.id3.**MCI**(*data='None'*)
Bases: mutagen._id3frames.MCDI

Binary dump of CD's TOC

class mutagen.id3.**MLL**(*frames=None*, *bytes=None*, *milliseconds=None*, *bits_for_bytes=None*, *bits_for_milliseconds=None*, *data='None'*)
Bases: mutagen._id3frames.MLLT

MPEG location lookup table

class mutagen.id3.**PIC**(*encoding=None*, *mime=None*, *type=None*, *desc=u'None'*, *data='None'*)
Bases: mutagen._id3frames.APIC

Attached Picture.

The 'mime' attribute of an ID3v2.2 attached picture must be either 'PNG' or 'JPG'.

class mutagen.id3.**POP**(*email=u'None'*, *rating=None*)
Bases: mutagen._id3frames.POPM

Popularimeter

class mutagen.id3.**REV**(*left=None*, *right=None*, *bounce_left=None*, *bounce_right=None*, *feedback_ltl=None*, *feedback_ltr=None*, *feedback_rtr=None*, *feedback_rtl=None*, *premix_ltr=None*, *premix_rtl=None*)
Bases: mutagen._id3frames.RVRB

Reverb

class mutagen.id3.**SLT**(*encoding=None*, *lang=None*, *format=None*, *type=None*, *desc=u'None'*, *text=None*)
Bases: mutagen._id3frames.SYLT

Synchronised lyrics/text

class mutagen.id3.**STC**(*format=None*, *data='None'*)
Bases: mutagen._id3frames.SYTC

Synced tempo codes

class mutagen.id3.**TAL**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TALB

Album

class mutagen.id3.**TBP**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TBPM

Beats per minute

class mutagen.id3.**TCM**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TCOM

Composer

class mutagen.id3.**TCO**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TCON

Content Type (Genre)

class mutagen.id3.**TCP**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TCMP

iTunes Compilation Flag

class mutagen.id3.**TCR**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TCOP

Copyright (C)

class mutagen.id3.**TDA**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TDAT

Date of recording (DDMM)

class mutagen.id3.**TDY**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TDLY

Audio Delay (ms)

class mutagen.id3.**TEN**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TENC

Encoder

class mutagen.id3.**TFT**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TFLT

File Type

class mutagen.id3.**TIM**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TIME

Time of recording (HHMM)

class mutagen.id3.**TKE**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TKEY

Starting Key

class mutagen.id3.**TLA**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TLAN

Audio Language(s)

class mutagen.id3.**TLE**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TLEN

Audio Length (ms)

class mutagen.id3.**TMT**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TMED

Source Media Type

class mutagen.id3.**TOA**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TOPE

Original Artist/Perfomer

class mutagen.id3.**TOF**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TOFN

Original Filename

class mutagen.id3.**TOL**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TOLY

Original Lyricist

class mutagen.id3.**TOR**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TORY

Original Release Year

class mutagen.id3.**TOT**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TOAL

Original Album

class mutagen.id3.**TP1**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TPE1

Lead Artist/Performer/Soloist/Group

class mutagen.id3.**TP2**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TPE2

Band/Orchestra/Accompaniment

class mutagen.id3.**TP3**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TPE3

Conductor

class mutagen.id3.**TP4**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TPE4

Interpreter/Remixer/Modifier

class mutagen.id3.**TPA**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TPOS

Part of set

class mutagen.id3.**TPB**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TPUB

Publisher

class mutagen.id3.**TRC**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TSRC

International Standard Recording Code (ISRC)

class mutagen.id3.**TRD**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TRDA

Recording Dates

class mutagen.id3.**TRK**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TRCK

Track Number

class mutagen.id3.**TSI**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TSIZ

Audio Data size (bytes)

class mutagen.id3.**TSS**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TSSE

Encoder settings

class mutagen.id3.**TT1**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TIT1

Content group description

class mutagen.id3.**TT2**(*encoding=None*, *text=[]*)
Bases: mutagen._id3frames.TIT2

Title

class mutagen.id3.**TT3**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TIT3

Subtitle/Description refinement

class mutagen.id3.**TXT**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TEXT

Lyricist

class mutagen.id3.**TXX**(*encoding=None*, *desc=u'None'*, *text=*[ ])
Bases: mutagen._id3frames.TXXX

User-defined Text

class mutagen.id3.**TYE**(*encoding=None*, *text=*[ ])
Bases: mutagen._id3frames.TYER

Year of recording

class mutagen.id3.**UFI**(*owner=u'None'*, *data='None'*)
Bases: mutagen._id3frames.UFID

Unique File Identifier

class mutagen.id3.**ULT**(*encoding=None*, *lang=None*, *desc=u'None'*, *text=u'None'*)
Bases: mutagen._id3frames.USLT

Unsychronised lyrics/text transcription

class mutagen.id3.**WAF**(*url=u'None'*)
Bases: mutagen._id3frames.WOAF

Official File Information

class mutagen.id3.**WAR**(*url=u'None'*)
Bases: mutagen._id3frames.WOAR

Official Artist/Performer Information

class mutagen.id3.**WAS**(*url=u'None'*)
Bases: mutagen._id3frames.WOAS

Official Source Information

class mutagen.id3.**WCM**(*url=u'None'*)
Bases: mutagen._id3frames.WCOM

Commercial Information

class mutagen.id3.**WCP**(*url=u'None'*)
Bases: mutagen._id3frames.WCOP

Copyright Information

class mutagen.id3.**WPB**(*url=u'None'*)
Bases: mutagen._id3frames.WPUB

Official Publisher Information

class mutagen.id3.**WXX**(*encoding=None*, *desc=u'None'*, *url=u'None'*)
Bases: mutagen._id3frames.WXXX

User-defined URL

9.2.2 ID3

class mutagen.id3.**ID3**
Bases: mutagen._util.DictProxy, mutagen.Metadata

A file with an ID3v2 tag.

Attributes:

- version – ID3 tag version as a tuple
- unknown_frames – raw frame data of any unknown frames found
- size – the total size of the ID3 tag, including the header

add (*frame*)
Add a frame to the tag.

delall (*key*)
Delete all tags of a given kind; see getall.

delete (*filename=None*, *delete_v1=True*, *delete_v2=True*)
Remove tags from a file.

If no filename is given, the one most recently loaded is used.

Keyword arguments:

- delete_v1 – delete any ID3v1 tag
- delete_v2 – delete any ID3v2 tag

getall (*key*)
Return all frames with a given name (the list may be empty).

This is best explained by examples:

```
id3.getall('TIT2') == [id3['TIT2']]
id3.getall('TTTT') == []
id3.getall('TXXX') == [TXXX(desc='woo', text='bar'),
                       TXXX(desc='baz', text='quuuux'), ...]
```

Since this is based on the frame's HashKey, which is colon-separated, you can use it to do things like `getall('COMM:MusicMatch')` or `getall('TXXX:QuodLibet:')`.

load (*filename*, *known_frames=None*, *translate=True*, *v2_version=4*)
Load tags from a filename.

Keyword arguments:

- filename – filename to load tag data from
- known_frames – dict mapping frame IDs to Frame objects
- **translate – Update all tags to ID3v2.3/4 internally. If you** intend to save, this must be true or you have to call update_to_v23() / update_to_v24() manually.
- v2_version – if update_to_v23 or update_to_v24 get called (3 or 4)

Example of loading a custom frame:

```
my_frames = dict(mutagen.id3.Frames)
class XMYF(Frame): ...
my_frames["XMYF"] = XMYF
mutagen.id3.ID3(filename, known_frames=my_frames)
```

pprint()
Return tags in a human-readable format.

"Human-readable" is used loosely here. The format is intended to mirror that used for Vorbis or APEv2 output, e.g.

```
TIT2=My Title
```

However, ID3 frames can have multiple keys:

```
POPM=user@example.org=3 128/255
```

save(*filename=None*, *v1=1*, *v2_version=4*, *v23_sep='/'*)
Save changes to a file.

If no filename is given, the one most recently loaded is used.

Keyword arguments: v1 – if 0, ID3v1 tags will be removed

if 1, ID3v1 tags will be updated but not added if 2, ID3v1 tags will be created and/or updated

v2 – version of ID3v2 tags (3 or 4).

By default Mutagen saves ID3v2.4 tags. If you want to save ID3v2.3 tags, you must call method update_to_v23 before saving the file.

v23_sep – the separator used to join multiple text values if v2_version == 3. Defaults to '/' but if it's None will be the ID3v2v2.4 null separator.

The lack of a way to update only an ID3v1 tag is intentional.

setall(*key*, *values*)
Delete frames of the given type and add frames in 'values'.

update_to_v23()
Convert older (and newer) tags into an ID3v2.3 tag.

This updates incompatible ID3v2 frames to ID3v2.3 ones. If you intend to save tags as ID3v2.3, you must call this function at some point.

If you want to to go off spec and include some v2.4 frames in v2.3, remove them before calling this and add them back afterwards.

update_to_v24()
Convert older tags into an ID3v2.4 tag.

This updates old ID3v2 frames to ID3v2.4 ones (e.g. TYER to TDRC). If you intend to save tags, you must call this function at some point; it is called by default when loading the tag.

class mutagen.id3.**ID3FileType**(*filename*, *ID3=None*)
An unknown type of file with ID3 tags.

add_tags(*ID3=None*)
Add an empty ID3 tag to the file.

A custom tag reader may be used in instead of the default mutagen.id3.ID3 object, e.g. an EasyID3 reader.

load(*filename*, *ID3=None*, ***kwargs*)
Load stream and tag information from a file.

A custom tag reader may be used in instead of the default mutagen.id3.ID3 object, e.g. an EasyID3 reader.

9.2.3 EasyID3

Easier access to ID3 tags.

EasyID3 is a wrapper around mutagen.id3.ID3 to make ID3 tags appear more like Vorbis or APEv2 tags.

class mutagen.easyid3.**EasyID3**(*filename=None*)

Bases: mutagen._util.DictMixin, mutagen.Metadata

A file with an ID3 tag.

Like Vorbis comments, EasyID3 keys are case-insensitive ASCII strings. Only a subset of ID3 frames are supported by default. Use EasyID3.RegisterKey and its wrappers to support more.

You can also set the GetFallback, SetFallback, and DeleteFallback to generic key getter/setter/deleter functions, which are called if no specific handler is registered for a key. Additionally, ListFallback can be used to supply an arbitrary list of extra keys. These can be set on EasyID3 or on individual instances after creation.

To use an EasyID3 class with mutagen.mp3.MP3:

```
from mutagen.mp3 import EasyMP3 as MP3
MP3(filename)
```

Because many of the attributes are constructed on the fly, things like the following will not work:

```
ezid3["performer"].append("Joe")
```

Instead, you must do:

```
values = ezid3["performer"]
values.append("Joe")
ezid3["performer"] = values
```

classmethod RegisterKey(*key*, *getter=None*, *setter=None*, *deleter=None*, *lister=None*)

Register a new key mapping.

A key mapping is four functions, a getter, setter, deleter, and lister. The key may be either a string or a glob pattern.

The getter, deleted, and lister receive an ID3 instance and the requested key name. The setter also receives the desired value, which will be a list of strings.

The getter, setter, and deleter are used to implement __getitem__, __setitem__, and __delitem__.

The lister is used to implement keys(). It should return a list of keys that are actually in the ID3 instance, provided by its associated getter.

classmethod RegisterTXXXKey(*key*, *desc*)

Register a user-defined text frame key.

Some ID3 tags are stored in TXXX frames, which allow a freeform 'description' which acts as a subkey, e.g. TXXX:BARCODE.:

```
EasyID3.RegisterTXXXKey('barcode', 'BARCODE').
```

classmethod RegisterTextKey(*key*, *frameid*)

Register a text key.

If the key you need to register is a simple one-to-one mapping of ID3 frame name to EasyID3 key, then you can use this function:

```
EasyID3.RegisterTextKey("title", "TIT2")
```

pprint()
Print tag key=value pairs.

class mutagen.easyid3.**EasyID3FileType**(*filename=None*, **args*, ***kwargs*)
Bases: mutagen.id3.ID3FileType

Like ID3FileType, but uses EasyID3 for tags.

9.2.4 MP3

MPEG audio stream information and tags.

class mutagen.mp3.**MP3**(*filename*, *ID3=None*)
Bases: mutagen.id3.ID3FileType

An MPEG audio (usually MPEG-1 Layer 3) file.

Variables

- **info** – MPEGInfo
- **tags** – ID3

class mutagen.mp3.**MPEGInfo**
MPEG audio stream information

Parse information about an MPEG audio file. This also reads the Xing VBR header format.

This code was implemented based on the format documentation at http://mpgedit.org/mpgedit/mpeg_format/mpeghdr.htm.

Useful attributes:

•length – audio length, in seconds

•bitrate – audio bitrate, in bits per second

•sketchy – if true, the file may not be valid MPEG audio

Useless attributes:

•version – MPEG version (1, 2, 2.5)

•layer – 1, 2, or 3

•mode – One of STEREO, JOINTSTEREO, DUALCHANNEL, or MONO (0-3)

•protected – whether or not the file is "protected"

•padding – whether or not audio frames are padded

•sample_rate – audio sample rate, in Hz

class mutagen.mp3.**EasyMP3**(*filename*, *ID3=None*)
Bases: mutagen.mp3.MP3

Like MP3, but uses EasyID3 for tags.

Variables

- **info** – MPEGInfo
- **tags** – EasyID3

9.2.5 TrueAudio

True Audio audio stream information and tags.

True Audio is a lossless format designed for real-time encoding and decoding. This module is based on the documentation at http://www.true-audio.com/TTA_Lossless_Audio_Codec_-_Format_Description

True Audio files use ID3 tags.

class `mutagen.trueaudio.`**`TrueAudio`**(*filename*, *ID3=None*)
Bases: `mutagen.id3.ID3FileType`

A True Audio file.

Variables

- **info** – `TrueAudioInfo`
- **tags** – `ID3`

class `mutagen.trueaudio.`**`TrueAudioInfo`**
True Audio stream information.

Attributes:

•length - audio length, in seconds

•sample_rate - audio sample rate, in Hz

class `mutagen.trueaudio.`**`EasyTrueAudio`**(*filename*, *ID3=None*)
Bases: `mutagen.trueaudio.TrueAudio`

Like MP3, but uses EasyID3 for tags.

Variables

- **info** – `TrueAudioInfo`
- **tags** – `EasyID3`

9.2.6 AIFF

AIFF audio stream information and tags.

class `mutagen.aiff.`**`AIFF`**(*filename*)
Bases: `mutagen.FileType`

An AIFF audio file.

Variables

- **info** – `AIFFInfo`
- **tags** – `ID3`

`add_tags`()
Add an empty ID3 tag to the file.

`load`(*filename*, ***kwargs*)
Load stream and tag information from a file.

class `mutagen.aiff.`**`AIFFInfo`**
AIFF audio stream information.

Information is parsed from the COMM chunk of the AIFF file

Useful attributes:

- length – audio length, in seconds
- bitrate – audio bitrate, in bits per second
- channels – The number of audio channels
- sample_rate – audio sample rate, in Hz
- sample_size – The audio sample size

9.3 FLAC

Read and write FLAC Vorbis comments and stream information.

Read more about FLAC at http://flac.sourceforge.net.

FLAC supports arbitrary metadata blocks. The two most interesting ones are the FLAC stream information block, and the Vorbis comment block; these are also the only ones Mutagen can currently read.

This module does not handle Ogg FLAC files.

Based off documentation available at http://flac.sourceforge.net/format.html

class `mutagen.flac.`**`FLAC`**(*filename*)

Bases: `mutagen.FileType`

A FLAC audio file.

Attributes:

- info – stream information (length, bitrate, sample rate)
- tags – metadata tags, if any
- cuesheet – CueSheet object, if any
- seektable – SeekTable object, if any
- pictures – list of embedded pictures

`add_picture`(*picture*)

Add a new picture to the file.

`add_tags`()

Add a Vorbis comment block to the file.

`add_vorbiscomment`()

Add a Vorbis comment block to the file.

`clear_pictures`()

Delete all pictures from the file.

`delete`(*filename=None*)

Remove Vorbis comments from a file.

If no filename is given, the one most recently loaded is used.

`load`(*filename*)

Load file information from a filename.

`pictures`

List of embedded pictures

save (*filename=None*, *deleteid3=False*)
Save metadata blocks to a file.

If no filename is given, the one most recently loaded is used.

class mutagen.flac.**StreamInfo** (*data*)
FLAC stream information.

This contains information about the audio data in the FLAC file. Unlike most stream information objects in Mutagen, changes to this one will rewritten to the file when it is saved. Unless you are actually changing the audio stream itself, don't change any attributes of this block.

Attributes:

•min_blocksize – minimum audio block size

•max_blocksize – maximum audio block size

•sample_rate – audio sample rate in Hz

•channels – audio channels (1 for mono, 2 for stereo)

•bits_per_sample – bits per sample

•total_samples – total samples in file

•length – audio length in seconds

class mutagen.flac.**Picture** (*data=None*)
Read and write FLAC embed pictures.

Attributes:

•type – picture type (same as types for ID3 APIC frames)

•mime – MIME type of the picture

•desc – picture's description

•width – width in pixels

•height – height in pixels

•depth – color depth in bits-per-pixel

•colors – number of colors for indexed palettes (like GIF), 0 for non-indexed

•data – picture data

class mutagen.flac.**SeekTable** (*data*)
Read and write FLAC seek tables.

Attributes:

•seekpoints – list of SeekPoint objects

class mutagen.flac.**CueSheet** (*data*)
Read and write FLAC embedded cue sheets.

Number of tracks should be from 1 to 100. There should always be exactly one lead-out track and that track must be the last track in the cue sheet.

Attributes:

•media_catalog_number – media catalog number in ASCII

•lead_in_samples – number of lead-in samples

•compact_disc – true if the cuesheet corresponds to a compact disc

- tracks – list of CueSheetTrack objects
- lead_out – lead-out as CueSheetTrack or None if lead-out was not found

class mutagen.flac.**CueSheetTrack**(*track_number*, *start_offset*, *isrc=''*, *type_=0*, *pre_emphasis=False*)

A track in a cuesheet.

For CD-DA, track_numbers must be 1-99, or 170 for the lead-out. Track_numbers must be unique within a cue sheet. There must be atleast one index in every track except the lead-out track which must have none.

Attributes:

- track_number – track number
- start_offset – track offset in samples from start of FLAC stream
- isrc – ISRC code
- type – 0 for audio, 1 for digital data
- pre_emphasis – true if the track is recorded with pre-emphasis
- indexes – list of CueSheetTrackIndex objects

class mutagen.flac.**CueSheetTrackIndex**

Index for a track in a cuesheet.

For CD-DA, an index_number of 0 corresponds to the track pre-gap. The first index in a track must have a number of 0 or 1, and subsequently, index_numbers must increase by 1. Index_numbers must be unique within a track. And index_offset must be evenly divisible by 588 samples.

Attributes:

- index_number – index point number
- index_offset – offset in samples from track start

9.4 OGG

9.4.1 Ogg bitstreams and pages

Read and write Ogg bitstreams and pages.

This module reads and writes a subset of the Ogg bitstream format version 0. It does *not* read or write Ogg Vorbis files! For that, you should use mutagen.oggvorbis.

This implementation is based on the RFC 3533 standard found at http://www.xiph.org/ogg/doc/rfc3533.txt.

exception mutagen.ogg.**error**

Ogg stream parsing errors.

class mutagen.ogg.**OggFileType**(*filename*)

Bases: mutagen.FileType

An generic Ogg file.

class mutagen.ogg.**OggPage**(*fileobj=None*)

A single Ogg page (not necessarily a single encoded packet).

A page is a header of 26 bytes, followed by the length of the data, followed by the data.

The constructor is givin a file-like object pointing to the start of an Ogg page. After the constructor is finished it is pointing to the start of the next page.

Attributes:

- version – stream structure version (currently always 0)
- position – absolute stream position (default -1)
- serial – logical stream serial number (default 0)
- sequence – page sequence number within logical stream (default 0)
- offset – offset this page was read from (default None)
- complete – if the last packet on this page is complete (default True)
- packets – list of raw packet data (default [])

Note that if 'complete' is false, the next page's 'continued' property must be true (so set both when constructing pages).

If a file-like object is supplied to the constructor, the above attributes will be filled in based on it.

`continued`
The first packet is continued from the previous page.

classmethod `find_last` (*klass*, *fileobj*, *serial*)
Find the last page of the stream 'serial'.

If the file is not multiplexed this function is fast. If it is, it must read the whole the stream.

This finds the last page in the actual file object, or the last page in the stream (with eos set), whichever comes first.

`first`
This is the first page of a logical bitstream.

classmethod `from_packets` (*klass*, *packets*, *sequence=0*, *default_size=4096*, *wiggle_room=2048*)
Construct a list of Ogg pages from a list of packet data.

The algorithm will generate pages of approximately default_size in size (rounded down to the nearest multiple of 255). However, it will also allow pages to increase to approximately default_size + wiggle_room if allowing the wiggle room would finish a packet (only one packet will be finished in this way per page; if the next packet would fit into the wiggle room, it still starts on a new page).

This method reduces packet fragmentation when packet sizes are slightly larger than the default page size, while still ensuring most pages are of the average size.

Pages are numbered started at 'sequence'; other information is uninitialized.

`last`
This is the last page of a logical bitstream.

classmethod `renumber` (*klass*, *fileobj*, *serial*, *start*)
Renumber pages belonging to a specified logical stream.

fileobj must be opened with mode r+b or w+b.

Starting at page number 'start', renumber all pages belonging to logical stream 'serial'. Other pages will be ignored.

fileobj must point to the start of a valid Ogg page; any occuring after it and part of the specified logical stream will be numbered. No adjustment will be made to the data in the pages nor the granule position; only the page number, and so also the CRC.

If an error occurs (e.g. non-Ogg data is found), fileobj will be left pointing to the place in the stream the error occured, but the invalid data will be left intact (since this function does not change the total file size).

classmethod **replace**(*klass*, *fileobj*, *old_pages*, *new_pages*)
Replace old_pages with new_pages within fileobj.

old_pages must have come from reading fileobj originally. new_pages are assumed to have the 'same' data as old_pages, and so the serial and sequence numbers will be copied, as will the flags for the first and last pages.

fileobj will be resized and pages renumbered as necessary. As such, it must be opened r+b or w+b.

size
Total frame size.

classmethod **to_packets**(*klass*, *pages*, *strict=False*)
Construct a list of packet data from a list of Ogg pages.

If strict is true, the first page must start a new packet, and the last page must end the last packet.

write()
Return a string encoding of the page header and data.

A ValueError is raised if the data is too big to fit in a single page.

9.4.2 Ogg Vorbis

Read and write Ogg Vorbis comments.

This module handles Vorbis files wrapped in an Ogg bitstream. The first Vorbis stream found is used.

Read more about Ogg Vorbis at http://vorbis.com/. This module is based on the specification at http://www.xiph.org/vorbis/doc/Vorbis_I_spec.html.

exception mutagen.oggvorbis.**error**
Bases: mutagen.ogg.error

exception mutagen.oggvorbis.**OggVorbisHeaderError**
Bases: mutagen.oggvorbis.error

class mutagen.oggvorbis.**OggVorbis**(*filename*)
Bases: mutagen.ogg.OggFileType

An Ogg Vorbis file.

class mutagen.oggvorbis.**OggVorbisInfo**(*fileobj*)
Ogg Vorbis stream information.

Attributes:

•length - file length in seconds, as a float

•bitrate - nominal ('average') bitrate in bits per second, as an int

9.4.3 Ogg Opus

Read and write Ogg Opus comments.

This module handles Opus files wrapped in an Ogg bitstream. The first Opus stream found is used.

Based on http://tools.ietf.org/html/draft-terriberry-oggopus-01

exception mutagen.oggopus.**error**
Bases: mutagen.ogg.error

exception mutagen.oggopus.**OggOpusHeaderError**
Bases: mutagen.oggopus.error

class mutagen.oggopus.**OggOpus**(*filename*)
Bases: mutagen.ogg.OggFileType

An Ogg Opus file.

class mutagen.oggopus.**OggOpusInfo**(*fileobj*)
Ogg Opus stream information.

Attributes:

•length - file length in seconds, as a float

•channels - number of channels

9.4.4 Ogg Speex

Read and write Ogg Speex comments.

This module handles Speex files wrapped in an Ogg bitstream. The first Speex stream found is used.

Read more about Ogg Speex at http://www.speex.org/. This module is based on the specification at http://www.speex.org/manual2/node7.html and clarifications after personal communication with Jean-Marc, http://lists.xiph.org/pipermail/speex-dev/2006-July/004676.html.

exception mutagen.oggspeex.**error**
Bases: mutagen.ogg.error

exception mutagen.oggspeex.**OggSpeexHeaderError**
Bases: mutagen.oggspeex.error

class mutagen.oggspeex.**OggSpeex**(*filename*)
Bases: mutagen.ogg.OggFileType

An Ogg Speex file.

class mutagen.oggspeex.**OggSpeexInfo**(*fileobj*)
Ogg Speex stream information.

Attributes:

•bitrate - nominal bitrate in bits per second

•channels - number of channels

•length - file length in seconds, as a float

The reference encoder does not set the bitrate; in this case, the bitrate will be 0.

9.4.5 Ogg Theora

Read and write Ogg Theora comments.

This module handles Theora files wrapped in an Ogg bitstream. The first Theora stream found is used.

Based on the specification at http://theora.org/doc/Theora_I_spec.pdf.

exception mutagen.oggtheora.**error**
Bases: mutagen.ogg.error

exception mutagen.oggtheora.**OggTheoraHeaderError**
Bases: mutagen.oggtheora.error

class mutagen.oggtheora.**OggTheora**(*filename*)
Bases: mutagen.ogg.OggFileType

An Ogg Theora file.

class mutagen.oggtheora.**OggTheoraInfo**(*fileobj*)
Ogg Theora stream information.

Attributes:

- length - file length in seconds, as a float
- fps - video frames per second, as a float

9.4.6 Ogg FLAC

Read and write Ogg FLAC comments.

This module handles FLAC files wrapped in an Ogg bitstream. The first FLAC stream found is used. For 'naked' FLACs, see mutagen.flac.

This module is based off the specification at http://flac.sourceforge.net/ogg_mapping.html.

exception mutagen.oggflac.**error**
Bases: mutagen.ogg.error

exception mutagen.oggflac.**OggFLACHeaderError**
Bases: mutagen.oggflac.error

class mutagen.oggflac.**OggFLAC**(*filename*)
Bases: mutagen.ogg.OggFileType

An Ogg FLAC file.

class mutagen.oggflac.**OggFLACStreamInfo**(*data*)
Ogg FLAC general header and stream info.

This encompasses the Ogg wrapper for the FLAC STREAMINFO metadata block, as well as the Ogg codec setup that precedes it.

Attributes (in addition to StreamInfo's):

- packets – number of metadata packets
- serial – Ogg logical stream serial number

9.5 APEv2

APEv2 reading and writing.

The APEv2 format is most commonly used with Musepack files, but is also the format of choice for WavPack and other formats. Some MP3s also have APEv2 tags, but this can cause problems with many MP3 decoders and taggers.

APEv2 tags, like Vorbis comments, are freeform key=value pairs. APEv2 keys can be any ASCII string with characters from 0x20 to 0x7E, between 2 and 255 characters long. Keys are case-sensitive, but readers are recommended to be case insensitive, and it is forbidden to multiple keys which differ only in case. Keys are usually stored title-cased (e.g. 'Artist' rather than 'artist').

APEv2 values are slightly more structured than Vorbis comments; values are flagged as one of text, binary, or an external reference (usually a URI).

Based off the format specification found at http://wiki.hydrogenaudio.org/index.php?title=APEv2_specification.

9.5.1 APEv2

exception mutagen.apev2.**error**

exception mutagen.apev2.**APENoHeaderError**

exception mutagen.apev2.**APEUnsupportedVersionError**

exception mutagen.apev2.**APEBadItemError**

class mutagen.apev2.**APEv2File**(*filename*)
Bases: mutagen.FileType

add_tags()

load(*filename*)

static score(*filename*, *fileobj*, *header*)

class mutagen.apev2.**APEv2**(**args*, ***kwargs*)
Bases: mutagen.apev2._CIDictProxy, mutagen.Metadata

A file with an APEv2 tag.

ID3v1 tags are silently ignored and overwritten.

delete(*filename=None*)
Remove tags from a file.

load(*filename*)
Load tags from a filename.

pprint()
Return tag key=value pairs in a human-readable format.

save(*filename=None*)
Save changes to a file.

If no filename is given, the one most recently loaded is used.

Tags are always written at the end of the file, and include a header and a footer.

9.5.2 Musepack

Musepack audio streams with APEv2 tags.

Musepack is an audio format originally based on the MPEG-1 Layer-2 algorithms. Stream versions 4 through 7 are supported.

For more information, see http://www.musepack.net/.

class mutagen.musepack.**Musepack**(*filename=None*, **args*, ***kwargs*)
Bases: mutagen.apev2.APEv2File

class mutagen.musepack.**MusepackInfo**(*fileobj*)
Musepack stream information.

Attributes:

- channels – number of audio channels
- length – file length in seconds, as a float
- sample_rate – audio sampling rate in Hz
- bitrate – audio bitrate, in bits per second
- version – Musepack stream version

Optional Attributes:

- title_gain, title_peak – Replay Gain and peak data for this song
- album_gain, album_peak – Replay Gain and peak data for this album

These attributes are only available in stream version 7/8. The gains are a float, +/- some dB. The peaks are a percentage [0..1] of the maximum amplitude. This means to get a number comparable to VorbisGain, you must multiply the peak by 2.

9.5.3 WavPack

WavPack reading and writing.

WavPack is a lossless format that uses APEv2 tags. Read http://www.wavpack.com/ for more information.

class mutagen.wavpack.**WavPack**(*filename=None*, **args*, ***kwargs*)
Bases: mutagen.apev2.APEv2File

class mutagen.wavpack.**WavPackInfo**(*fileobj*)
WavPack stream information.

Attributes:

- channels - number of audio channels (1 or 2)
- length - file length in seconds, as a float
- sample_rate - audio sampling rate in Hz
- version - WavPack stream version

9.5.4 Monkey's Audio

Monkey's Audio streams with APEv2 tags.

Monkey's Audio is a very efficient lossless audio compressor developed by Matt Ashland.

For more information, see http://www.monkeysaudio.com/.

class mutagen.monkeysaudio.**MonkeysAudio**(*filename=None*, **args*, ***kwargs*)
Bases: mutagen.apev2.APEv2File

class mutagen.monkeysaudio.**MonkeysAudioInfo**(*fileobj*)
Monkey's Audio stream information.

Attributes:

- channels – number of audio channels
- length – file length in seconds, as a float
- sample_rate – audio sampling rate in Hz
- bits_per_sample – bits per sample

•version – Monkey's Audio stream version, as a float (eg: 3.99)

9.5.5 OptimFROG

OptimFROG audio streams with APEv2 tags.

OptimFROG is a lossless audio compression program. Its main goal is to reduce at maximum the size of audio files, while permitting bit identical restoration for all input. It is similar with the ZIP compression, but it is highly specialized to compress audio data.

Only versions 4.5 and higher are supported.

For more information, see http://www.losslessaudio.org/

class mutagen.optimfrog.**OptimFROG**(*filename=None*, **args*, ***kwargs*)
Bases: mutagen.apev2.APEv2File

class mutagen.optimfrog.**OptimFROGInfo**(*fileobj*)
OptimFROG stream information.

Attributes:

•channels - number of audio channels

•length - file length in seconds, as a float

•sample_rate - audio sampling rate in Hz

9.6 MP4

Read and write MPEG-4 audio files with iTunes metadata.

This module will read MPEG-4 audio information and metadata, as found in Apple's MP4 (aka M4A, M4B, M4P) files.

There is no official specification for this format. The source code for TagLib, FAAD, and various MPEG specifications at

- http://developer.apple.com/documentation/QuickTime/QTFF/
- http://www.geocities.com/xhelmboyx/quicktime/formats/mp4-layout.txt
- http://standards.iso.org/ittf/PubliclyAvailableStandards/c041828_ISO_IEC_14496-12_2005(E).zip
- http://wiki.multimedia.cx/index.php?title=Apple_QuickTime

were all consulted.

9.6.1 MP4

class mutagen.mp4.**MP4**(*filename*)
Bases: mutagen.FileType

An MPEG-4 audio file, probably containing AAC.

If more than one track is present in the file, the first is used. Only audio ('soun') tracks will be read.

Variables

- **info** – MP4Info

- **tags** – MP4Tags

class mutagen.mp4.**MP4Tags**

Bases: mutagen._util.DictProxy, mutagen.Metadata

Dictionary containing Apple iTunes metadata list key/values.

Keys are four byte identifiers, except for freeform ('----') keys. Values are usually unicode strings, but some atoms have a special structure:

Text values (multiple values per key are supported):

- '\xa9nam' – track title
- '\xa9alb' – album
- '\xa9ART' – artist
- 'aART' – album artist
- '\xa9wrt' – composer
- '\xa9day' – year
- '\xa9cmt' – comment
- 'desc' – description (usually used in podcasts)
- 'purd' – purchase date
- '\xa9grp' – grouping
- '\xa9gen' – genre
- '\xa9lyr' – lyrics
- 'purl' – podcast URL
- 'egid' – podcast episode GUID
- 'catg' – podcast category
- 'keyw' – podcast keywords
- '\xa9too' – encoded by
- 'cprt' – copyright
- 'soal' – album sort order
- 'soaa' – album artist sort order
- 'soar' – artist sort order
- 'sonm' – title sort order
- 'soco' – composer sort order
- 'sosn' – show sort order
- 'tvsh' – show name

Boolean values:

- 'cpil' – part of a compilation
- 'pgap' – part of a gapless album
- 'pcst' – podcast (iTunes reads this only on import)

Tuples of ints (multiple values per key are supported):

- 'trkn' – track number, total tracks
- 'disk' – disc number, total discs

Others:

- 'tmpo' – tempo/BPM, 16 bit int
- 'covr' – cover artwork, list of MP4Cover objects (which are tagged strs)
- 'gnre' – ID3v1 genre. Not supported, use '\xa9gen' instead.

The freeform '----' frames use a key in the format '----:mean:name' where 'mean' is usually 'com.apple.iTunes' and 'name' is a unique identifier for this frame. The value is a str, but is probably text that can be decoded as UTF-8. Multiple values per key are supported.

MP4 tag data cannot exist outside of the structure of an MP4 file, so this class should not be manually instantiated.

Unknown non-text tags are removed.

delete (*filename*)
Remove the metadata from the given filename.

save (*filename*)
Save the metadata to the given filename.

class mutagen.mp4.**MP4Info**
MPEG-4 stream information.

Attributes:

- bitrate – bitrate in bits per second, as an int
- length – file length in seconds, as a float
- channels – number of audio channels
- sample_rate – audio sampling rate in Hz
- bits_per_sample – bits per sample

class mutagen.mp4.**MP4Cover** (*data*, *imageformat=13*)
A cover artwork.

Attributes:

- imageformat – format of the image (either FORMAT_JPEG or FORMAT_PNG)

class mutagen.mp4.**MP4FreeForm** (*data*, *dataformat=1*)
A freeform value.

Attributes:

- dataformat – format of the data (either FORMAT_TEXT or FORMAT_DATA)

mutagen.mp4.**Open** (*filename*)

mutagen.mp4.**delete** (*filename*)
Remove tags from a file.

9.6.2 EasyMP4

class mutagen.easymp4.**EasyMP4** (*filename*)
Bases: mutagen.mp4.MP4

Like MP4, but uses EasyMP4Tags for tags.

Variables

- **info** – MP4Info
- **tags** – EasyMP4Tags

classmethod RegisterKey(*key*, *getter=None*, *setter=None*, *deleter=None*, *lister=None*)
Register a new key mapping.

A key mapping is four functions, a getter, setter, deleter, and lister. The key may be either a string or a glob pattern.

The getter, deleted, and lister receive an MP4Tags instance and the requested key name. The setter also receives the desired value, which will be a list of strings.

The getter, setter, and deleter are used to implement __getitem__, __setitem__, and __delitem__.

The lister is used to implement keys(). It should return a list of keys that are actually in the MP4 instance, provided by its associated getter.

classmethod RegisterTextKey(*key*, *atomid*)
Register a text key.

If the key you need to register is a simple one-to-one mapping of MP4 atom name to EasyMP4Tags key, then you can use this function:

```
EasyMP4Tags.RegisterTextKey("artist", "©ART")
```

class mutagen.easymp4.**EasyMP4Tags**
Bases: mutagen._util.DictMixin, mutagen.Metadata

A file with MPEG-4 iTunes metadata.

Like Vorbis comments, EasyMP4Tags keys are case-insensitive ASCII strings, and values are a list of Unicode strings (and these lists are always of length 0 or 1).

If you need access to the full MP4 metadata feature set, you should use MP4, not EasyMP4.

classmethod RegisterFreeformKey(*key*, *name*, *mean='com.apple.iTunes'*)
Register a text key.

If the key you need to register is a simple one-to-one mapping of MP4 freeform atom (—-) and name to EasyMP4Tags key, then you can use this function:

```
EasyMP4Tags.RegisterFreeformKey(
    "musicbrainz_artistid", "MusicBrainz Artist Id")
```

classmethod RegisterIntKey(*key*, *atomid*, *min_value=0*, *max_value=65535*)
Register a scalar integer key.

classmethod RegisterKey(*key*, *getter=None*, *setter=None*, *deleter=None*, *lister=None*)
Register a new key mapping.

A key mapping is four functions, a getter, setter, deleter, and lister. The key may be either a string or a glob pattern.

The getter, deleted, and lister receive an MP4Tags instance and the requested key name. The setter also receives the desired value, which will be a list of strings.

The getter, setter, and deleter are used to implement __getitem__, __setitem__, and __delitem__.

The lister is used to implement keys(). It should return a list of keys that are actually in the MP4 instance, provided by its associated getter.

classmethod **RegisterTextKey**(*key*, *atomid*)
Register a text key.

If the key you need to register is a simple one-to-one mapping of MP4 atom name to EasyMP4Tags key, then you can use this function:

```
EasyMP4Tags.RegisterTextKey("artist", "©ART")
```

pprint()
Print tag key=value pairs.

9.7 ASF

Read and write ASF (Window Media Audio) files.

class mutagen.asf.**ASF**(*filename=None*, **args*, ***kwargs*)
Bases: mutagen.FileType

An ASF file, probably containing WMA or WMV.

load(*filename*)

save()

static **score**(*filename*, *fileobj*, *header*)

class mutagen.asf.**ASFInfo**
ASF stream information.

pprint()

CHAPTER 10

Tools

10.1 mid3iconv

10.1.1 convert ID3 tag encodings

Manual section 1

Date April 10th, 2006

SYNOPSIS

mid3iconv [*options*] *filename ...*

DESCRIPTION

mid3iconv converts ID3 tags from legacy encodings to Unicode and stores them using the ID3v2 format.

OPTIONS

--debug, -d	Print updated tags
--dry-run, -p	Do not actually modify files
--encoding, -e	Convert from this encoding. By default, your locale's default encoding is used.
--force-v1	Use an ID3v1 tag even if an ID3v2 tag is present
--quiet, -q	Only output errors
--remove-v1	Remove any ID3v1 tag after processing the files

AUTHOR

Emfox Zhou.

Based on id3iconv (http://www.cs.berkeley.edu/~zf/id3iconv/) by Feng Zhou.

10.2 mid3v2

10.2.1 audio tag editor similar to 'id3v2'

Manual section 1

Date October 30th, 2010

SYNOPSIS

mid3v2 [*options*] *filename ...*

DESCRIPTION

mid3v2 is a Mutagen-based replacement for id3lib's **id3v2**. It supports ID3v2.4 and more frames; it also does not have the numerous bugs that plague **id3v2**.

This program exists mostly for compatibility with programs that want to tag files using **id3v2**. For a more usable interface, we recommend Ex Falso.

OPTIONS

-q, --quiet Be quiet: do not mention file operations that perform the user's request. Warnings will still be printed.

-v, --verbose Be verbose: state all operations performed. This is the opposite of –quiet. This is the default.

-e, --escape Enable interpretation of backslash escapes for tag values. Makes it possible to escape the colon-separator in TXXX, COMM values like '\:' and insert escape sequences like '\n', '\t' etc.

-f, --list-frames Display all supported ID3v2.3/2.4 frames and their meanings.

-L, --list-genres List all ID3v1 numeric genres. These can be used to set TCON frames, but it is not recommended.

-l, --list List all tags in the files. The output format is *not* the same as **id3v2**'s; instead, it is easily parsable and readable. Some tags may not have human-readable representations.

--list-raw List all tags in the files, in raw format. Although this format is nominally human-readable, it may be very long if the tag contains embedded binary data.

-d, --delete-v2 Delete ID3v2 tags.

-s, --delete-v1 Delete ID3v1 tags.

-D, --delete-all Delete all ID3 tags.

–delete-frames=FID1,FID2,... Delete specific ID3v2 frames (or groups of frames) from the files.

-C, --convert Convert ID3v1 tags to ID3v2 tags. This will also happen automatically during any editing.

-a, –artist=artist Set the artist information (TPE1).

-A, –album=album Set the album information (TALB).

-t, –song=title Set the title information (TIT2).

-c, –comment=DESCRIPTION:COMMENT:LANGUAGE Set a comment (COMM). The language and description may be omitted, in which case the language defaults to English, and the description to an empty string.

-g, –genre=genre Set the genre information (TCON).

-y, –year=, –date=YYYY-[MM-DD] Set the year/date information (TDRC).

-Tnum/num, –track=num/num Set the track number (TRCK).

Any text or URL frame (those beginning with T or W) can be modified or added by prefixing the name of the frame with "–". For example, **–TIT3 "Monkey!"** will set the TIT3 (subtitle) frame to **Monkey!**.

The TXXX frame requires a colon-separated description key; many TXXX frames may be set in the file as long as they have different keys. To set this key, just separate the text with a colon, e.g. **–TXXX "ALBUMARTISTSORT:Examples, The"**.

The special POPM frame can be set in a similar way: **–POPM "bob@example.com:128:2"** to set Bob's rating to 128/255 with 2 plays.

BUGS

No sanity checking is done on the editing operations you perform, so mid3v2 will happily accept –TSIZ when editing an ID3v2.4 frame. However, it will also automatically throw it out during the next edit operation.

AUTHOR

Joe Wreschnig is the author of mid3v2, but he doesn't like to admit it.

10.3 moggsplit

10.3.1 split Ogg logical streams

Manual section 1

Date Nov 14th, 2009

SYNOPSIS

moggsplit *filename ...*

DESCRIPTION

moggsplit splits a multiplexed Ogg stream into separate files. For example, it can separate an OGM into separate Ogg DivX and Ogg Vorbis streams, or a chained Ogg Vorbis file into two separate files.

OPTIONS

--extension Use the supplied extension when generating new files; the default is **ogg**.

--pattern Use the supplied pattern when generating new files. This is a Python keyword format string with three variables, *base* for the original file's base name, *stream* for the stream's serial number, and ext for the extension give by **–extension**.

The default is **%(base)s-%(stream)d.%(ext)s**.

--m3u Generate an m3u playlist along with the newly generated files. Useful for large chained Oggs.

AUTHOR

Joe Wreschnig

10.4 mutagen-inspect

10.4.1 view Mutagen-supported audio tags

Manual section 1

Date May 27th, 2006

SYNOPSIS

mutagen-inspect *filename ...*

DESCRIPTION

mutagen-inspect loads and prints information about an audio file and its tags.

It is primarily intended as a debugging tool for Mutagen, but can be useful for extracting tags from the command line.

AUTHOR

Joe Wreschnig

10.5 mutagen-pony

10.5.1 scan a collection of MP3 files

Manual section 1

Date February 20th, 2006

SYNOPSIS

mutagen-pony *directory ...*

DESCRIPTION

mutagen-pony scans any directories given and reports on the kinds of tags in the MP3s it finds in them. Ride the pony.

It is primarily intended as a debugging tool for Mutagen.

AUTHORS

Michael Urman and Joe Wreschnig

CHAPTER 11

Mutagen Documentation

Note: This documentation is still incomplete and it's recommended to read the source for the full details.

11.1 What is Mutagen?

Mutagen is a Python module to handle audio metadata. It supports ASF, FLAC, M4A, Monkey's Audio, MP3, Musepack, Ogg Opus, Ogg FLAC, Ogg Speex, Ogg Theora, Ogg Vorbis, True Audio, WavPack, OptimFROG, and AIFF audio files. All versions of ID3v2 are supported, and all standard ID3v2.4 frames are parsed. It can read Xing headers to accurately calculate the bitrate and length of MP3s. ID3 and APEv2 tags can be edited regardless of audio format. It can also manipulate Ogg streams on an individual packet/page level.

Mutagen works on Python 2.6+ / PyPy and has no dependencies outside the CPython standard library.

There is a *brief tutorial with several API examples.*

11.2 Where do I get it?

Mutagen is hosted on Bitbucket. The download page will have the latest version or check out the Mercurial repository:

```
$ hg clone https://bitbucket.org/lazka/mutagen
```

11.3 Why Mutagen?

Quod Libet has more strenuous requirements in a tagging library than most programs that deal with tags. Furthermore, most tagging libraries suck. Therefore we felt it was necessary to write our own.

- Mutagen has a simple API, that is roughly the same across all tag formats and versions and integrates into Python's builtin types and interfaces.
- New frame types and file formats are easily added, and the behavior of the current formats can be changed by extending them.
- Freeform keys, multiple values, Unicode, and other advanced features were considered from the start and are fully supported.
- All ID3v2 versions and all ID3v2.4 frames are covered, including rare ones like POPM or RVA2.

- We take automated testing very seriously. All bug fixes are commited with a test that prevents them from recurring, and new features are committed with a full test suite.

11.4 Real World Use

Mutagen can load nearly every MP3 we have thrown at it (when it hasn't, we make it do so). Scripts are included so you can run the same tests on your collection.

The following software projects are using Mutagen for tagging:

- Ex Falso and Quod Libet, a flexible tagger and player
- Beets, a music library manager and MusicBrainz tagger
- Picard, cross-platform MusicBrainz tagger
- Puddletag, an audio tag editor
- Listen, a music player for GNOME
- Exaile, a media player aiming to be similar to KDE's AmaroK, but for GTK+
- ZOMG, a command-line player for ZSH
- pytagsfs, virtual file system for organizing media files by metadata
- Debian's version of JACK, an audio CD ripper, uses Mutagen to tag FLACs
- Amarok's replaygain script

11.5 Contact

For historical and practical reasons, Mutagen shares a mailing list and IRC channel (#quodlibet on irc.oftc.net) with Quod Libet. If you need help using Mutagen or would like to discuss the library, please use the mailing list. Bugs and patches should go to the issue tracker.

m