

## Python Audio Tools Manual

*Brian Langenberger*  
*tuffy@users.sourceforge.net*

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/us/> or send a letter to: Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.



29 June 2009

## Table of Contents

Installation . . . . .	1
Prerequisites . . . . .	1
Installation Procedure . . . . .	2
Fixing Installation Problems . . . . .	2
The Configuration File . . . . .	3
Options precedence . . . . .	3
The Filename Format . . . . .	4
Using the Tools . . . . .	5
cd2xmcd . . . . .	6
cd2track . . . . .	7
track2track . . . . .	8
track2xmcd . . . . .	9
editxmcd . . . . .	10
tracktag . . . . .	11
trackrename . . . . .	12
track2cd . . . . .	13
trackinfo . . . . .	14
tracklength . . . . .	15
trackplay . . . . .	16
record2track . . . . .	17
trackcmp . . . . .	18
trackcat . . . . .	19
tracksplit . . . . .	20
coverview . . . . .	21
coverdump . . . . .	22
Python Reference . . . . .	23
Module Attributes . . . . .	23
Module Functions . . . . .	24
AudioFile . . . . .	25
Metadata . . . . .	28
Cuesheets . . . . .	32
PCMReader . . . . .	33
PCMConverter . . . . .	33
CDDA . . . . .	34
XMCD . . . . .	36

## 1. Installation

### 1.1. Prerequisites

- Python 2.5.0 <http://www.python.org>
- Construct (included) <http://construct.wikispaces.com>
- libcdio <http://www.gnu.org/software/libcdio>
- FLAC <http://flac.sourceforge.net>
- LAME <http://lame.sourceforge.net>
- TwoLAME <http://www.twolame.org>
- mpg123 <http://www.mpg123.de/>
- vorbis-tools <http://www.xiph.org>
- Speex <http://www.speex.org>
- Musepack <http://www.musepack.net>
- FAAC/FAAD2 <http://www.audiocoding.com>
- Nero AAC Codec <http://www.nero.com/enu/downloads-nerodigital-nero-aac-codec.php>
- WavPack <http://www.wavpack.com>
- CDRecord <http://cdrecord.berlios.de/old/private/cdrecord.html>
- cdrdao <http://cdrdao.sourceforge.net/>

Python 2.5.0 or higher, libcdio and Construct 2.00 or higher are required. The rest are optional but recommended in order to get the most out of these tools.

Python Audio Tools Prerequisites, by Format						
Format	Suffix	Decoding	Encoding	Metadata Reading	Metadata Writing	ReplayGain
RIFF WAVE	.wav	Python	Python	N/A	N/A	N/A
AIFF	.aiff	Python	Python	N/A	N/A	N/A
Sun AU	.au	Python	Python	N/A	N/A	N/A
FLAC	.flac	flac	flac	Python	Python	metaflac
Ogg FLAC	.oga	flac	flac	Python	Python	N/A
WavPack	.wv	wvunpack	wavpack	Python	Python	wvgain
MP3	.mp3	lame / mpg123	lame	Python	Python	N/A
MP2	.mp2	lame / mpg123	twolame	Python	Python	N/A
Ogg Vorbis	.ogg	oggdec	oggenc	Python	vorbiscomment	vorbisgain
Ogg Speex	.spx	speexdec	speexenc	Python	Python	N/A
M4A	.m4a	faad / neroAacDec	faac / neroAacEnc	Python	Python	N/A
Musepack	.mpc	mpcdec	mpcenc	Python	Python	N/A

## 1.2. Installation Procedure

To install Python Audio Tools, you simply need to run:

```
make install
```

as root from the audiotools source directory. This will use the Python interpreter to install the audiotools Python module and the executable scripts. It will then install the man pages from the `doc/` subdirectory. If you do not have the Construct Python module installed, you can run:

```
make construct_install
```

as root from the audiotools source directory to install it.

To verify your Python Audio Tools installation, run:

```
audiotools-config
```

as a normal user. This will load the `audiotools` Python module, if possible, and deliver a listing of available audio formats and current system settings.

### 1.2.1. Fixing Installation Problems

- **The audiotools.cdio module doesn't build correctly**

Check that you have the CDIO library installed, commonly known as `libcdio`. If `libcdio` is installed and the module still doesn't build, ensure that you've also installed any accompanying `libcdio-devel` package.

- **The Construct module isn't found**

Run the `make construct_install` command from the audiotools source directory, as root, to install this module.

- **The installer complains about missing 'python/config/Makefile' of some sort**

Your distribution probably has separate "python" and "python-devel" packages and "python-devel" is not yet installed. Use your package manager to install "python-devel" and try installing Python Audio Tools again.

- **audiotools-config lists formats as unavailable**

Certain audio formats require external programs. For instance, to use FLAC files, Python Audio Tools requires the `flac` and `metaflac` programs. If these cannot be found in the regular executable search path or from the config file (which will be explained later), you will not be able to use that format.

Check your system's package manager for programs which may be available but not yet installed. If that fails to find an appropriate precompiled binary for your system, use the web sites listed in the 'Prerequisites' section for download and installation instructions.

- **The system configuration listed by audiotools-config is not correct**

The FreeDB server, CD-ROM device and file name format are adjustable from the `audiotools.cfg` configuration file, which will be covered in the next section.

- **PulseAudio playback isn't working**

Python Audio Tools requires the `pacat` and `pactl` executables for playback via the `track-play` executable. If you have the PulseAudio server installed but these aren't present, ensure you've also installed an accompanying "pulseaudio-utils" package.

If `pacat` and `pactl` are installed and PulseAudio playback still isn't working, make sure the PulseAudio server is up and running with the following command:

```
% pactl stat
```

- **My Python interpreter isn't found, or I wish to use a different one**

The first line of `Makefile` is which Python interpreter is being used for installation of both the Python Audio Tools and Construct module. For instance, to use a Python interpreter located at `/opt/python/bin/python`, you should change that line to read:

```
export PYTHON = /opt/python/bin/python
```

Running `make` will then invoke the new interpreter for installation of the audiotools module and scripts.

### 1.3. The Configuration File

Because not all systems are alike, and hard-coding values into the application is not pleasant for non-programmers, Python Audio Tools comes with a small config file named `audiotools.cfg` which contains user-editable values. These values include the default FreeDB server to use, the default filename format for new tracks, the default CD-ROM device to use for audio extraction/burning and, optionally, alternative paths to required executables.

The format of this file is straightforward. It consists of one or more sections. Each section contains one or more `key: value` pairs.

audiotools.cfg options		
Section	Key	Value
[FreeDB]	server port	the default FreeDB server hostname the default FreeDB server port
[System]	cdrom cdrom_read_offset maximum_jobs fs_encoding io_encoding	the default CD-ROM device to use for CD reading/writing the sample offset to apply when reading tracks the maximum amount of processes to run simultaneously when converting tracks the filesystem's text encoding, for reading/writing filenames the terminal's text encoding, when generating screen output
[Filenames]	format	the default filename format for new tracks
[Binaries]	flac lame ...	the flac executable to use for encoding/decoding FLAC files the lame executable to use for encoding/decoding MP3 files which executable to use other than the default
[Thumbnail]	format size	the image format to use for thumbnails, such as 'jpeg' or 'png' the maximum size of each thumbnail in pixels (e.g. a value of 100 results in a maximum size of 100×100 pixel thumbnails)

#### 1.3.1. Options precedence

When Python Audio Tools needs an option, such as a FreeDB server name, it looks in the following order:

1. The command-line. For example, specifying the `--freedb-server` command-line option when executing a program overrides any values in config files.
2. The `$(HOME)/.audiotools.cfg` config file. This file has the same syntax as the system-wide `.audiotools.cfg` file. Since it need only contain the options that one wishes to override, it may be smaller than the system-wide config file.
3. The system-wide config file found in `/etc/audiotools.cfg`.

If no command-line option is given and no config files are found Python Audio Tools will use some hard-coded default value, which may not be what you want.

### 1.3.2. The Filename Format

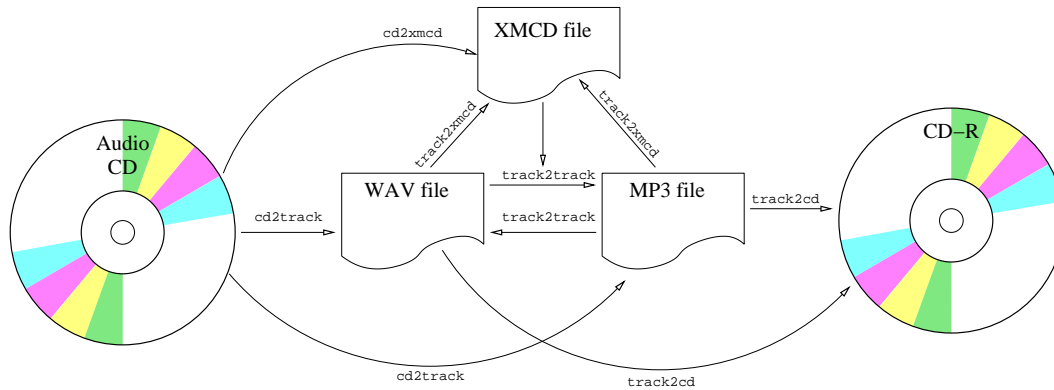
The default filename format is a template string which contains Python-style `%(key)s` wildcards which are replaced by actual values when tracks are created or renamed. Actual values are pulled from the file's own metadata or an external XMCD file.

Key	Value
<code>%(track_number)2.2d</code>	the track's number on the CD, from 01-99
<code>%(track_total)d</code>	the total number of tracks on the CD, from 01-99
<code>%(album_number)d</code>	the CD's album number, for multi-disc compilations
<code>%(album_total)d</code>	the total number of CDs in the set
<code>%(album_track_number)s</code>	a combination of album and track number as one field (e.g. an album number of 2 and track number of 13 has the value 213 but an empty album number and track number of 5 has the value 05)
<code>%(track_name)s</code>	the track's name
<code>%(album_name)s</code>	the album's name
<code>%(artist_name)s</code>	the track's artist name
<code>%(performer_name)s</code>	the track's performer name
<code>%(composer_name)s</code>	the track's composer name
<code>%(conductor_name)s</code>	the track's conductor name
<code>%(media)s</code>	the track's source media
<code>%(ISRC)s</code>	the track's ISRC
<code>%(catalog)s</code>	the track's catalog number
<code>%(copyright)s</code>	the track's copyright information
<code>%(publisher)s</code>	the track's publisher
<code>%(year)s</code>	the track's publication year
<code>%(date)s</code>	the track's original recording date
<code>%(suffix)s</code>	the track's suffix, based on its file type (e.g. MP3 files will have the suffix mp3)

All other text in the format string is left as-is.

## 2. Using the Tools

These Python Audio Tools are command-line programs which allow the user to extract audio from CDs, convert audio from one format to another, rename audio tracks and burn audio back to CDs. They are designed to provide a complete toolchain of audio handling utilities, but also to interoperate well with other tools.



Since Python Audio Tools is comprised of many small utilities, this example diagram illustrates how one would go from physical media to encoded audio files and back again. It looks more complicated than it is. Let's go through the process of ripping a CD as an example. From a command-line prompt (indicated by '%'), run:

```
% cd2track
```

This will access your CD and extract all the audio files on it into the default audio format, resulting in files named 'track01.cdda.wav', 'track02.cdda.wav' and so forth. But if you wanted MP3 files, there's more work to be done.

```
% track2track -t mp3 *.wav
```

Will convert those .wav files to MP3 files (indicated by the `-t mp3` flag which tells the program what file type we want). Unfortunately, that gives us raw, untagged MP3s without names. That's where the XMCD data comes in. XMCD files, retrieved from FreeDB, contain the information we need.

```
% cd2xmcd -x album.xmcd
```

```
% track2track -x album.xmcd -t mp3 *.wav
```

Fetches the data to a file called 'album.xmcd' which we then feed to `track2track` so it knows what to call our new MP3 files. Combining `cd2track`, `cd2xmcd` and `track2track` is a nice way to rip audio in batches. For instance, since ripping and converting happen at different speeds, you could run each process in its own window to keep your computer as busy as possible.

However, there is an easier way. `cd2track` doesn't have to rip CDs to .wav files; it can go directly to MP3s if you ask it to. It still requires an XMCD file to know what to name those tracks, but our ripping process can be reduced to:

```
% cd2xmcd -x album.xmcd
```

```
% cd2track -x album.xmcd -t mp3
```

If you don't like the names in `album.xmcd`, feel free to edit it either with the `editxmcd` program or a text editor.

Lossless audio is just as easy to work with. Creating FLAC files from a CD is almost identical:

```
% cd2xmcd -x album.xmcd
```

```
% cd2track -x album.xmcd -t flac
```

But since FLAC files are pretty big, you might want to convert them for portable use:

```
% track2track -t mp3 *.flac
```

Unlike going from .wav to .mp3, these FLAC files are already tagged with information and so an XMCD file isn't required.

## 2.1. cd2xmcd

cd2xmcd(1)

cd2xmcd(1)

### NAME

cd2xmcd – retrieves metadata from a compact disc

### SYNOPSIS

cd2xmcd [OPTIONS]

### DESCRIPTION

cd2xmcd takes a compact disc device and retrieves an XMCD file containing its metadata such as track name, album name and artist. If no data can be found, an empty XMCD file is written which can be edited by the user.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-x filename, --xmcd=filename**

The file in which the XMCD data will be placed. **cd2xmcd(1)** will refuse to overwrite an existing file if present. If no filename is given, XMCD data will be written to standard output.

**-c cdrom, --cdrom=cdrom**

The cdrom device to read CD information from

**-s server, --freedb-server=server**

The FreeDB server name to query for XMCD data

**-p port, --freedb-port=port**

The FreeDB port number to query for XMCD data

**-i, --id** Calculate and display the disc ID rather than retrieve XMCD data

**-D, --default**

If multiple matching discs are found, select the first choice automatically

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### EXAMPLE

Obtain an XMCD file called *album.xmcd* from the contents of */dev/cdrom*.

**cd2xmcd -x album.xmcd -c /dev/cdrom**

## 2.2. cd2track

cd2track(1)

cd2track(1)

### NAME

cd2track – extract audio files

### SYNOPSIS

cd2track [OPTIONS] [track 1] [track 2] ...

### DESCRIPTION

cd2track extracts audio files from a compact disc and encodes them to tracks. If track numbers are given, extracts only those tracks. Otherwise, extracts the entire disc.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-c cdrom, --cdrom=cdrom**

The cdrom device to extract tracks from

**-s speed, --speed=speed**

The speed in which to read CD data at

**-t type, --type=type**

The audio format to extract the given tracks to. For a list of available audio formats, try: **-t help**

**-q quality, --quality=quality**

The desired quality of the extracted tracks. For a list of available quality modes for a given format, try: **-q help**

**-x filename, --xmcd=filename**

The XMCD file to use for metadata for the extracted tracks. XMCD files can be obtained from the **track2xmcd(1)**, **cd2xmcd(1)** and **editxmcd(1)** programs.

**-d directory, --dir=directory**

The target directory for the converted tracks. If none is given, the current working directory is used (see **getcwd(3)**). If the target directory does not exist, it will be created automatically.

**--format=format string**

The format string to use for new filenames. Template fields are replaced with metadata values when new tracks are created. All other text is left as-is. If this option is omitted, a default format string is used.

**--album-number=number**

The album number of this CD, if it is one of a series of albums.

**--replay-gain**

Add ReplayGain metadata to newly created tracks

**--no-replay-gain**

Do not add ReplayGain metadata to newly created tracks

Depending on the audio format used, ReplayGain is added automatically by default when it is implemented by inserting a metadata tag. When ReplayGain is implemented by modifying audio data, it is not added automatically and must be explicitly enabled.

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### EXAMPLE

Extract all of the tracks from */dev/cdrom* as FLAC files at the default quality using metadata from *album.xmcd*

## 2.3. track2track

track2track(1)

track2track(1)

### NAME

track2track – convert audio tracks

### SYNOPSIS

track2track [OPTIONS] <track 1> [track 2] ...

### DESCRIPTION

track2track converts audio tracks from one format to another.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-t type, --type=type**

The audio format to convert the given tracks to. For a list of available audio formats, try: **-t help**

**-q quality, --quality=quality**

The desired quality of the converted tracks. For a list of available quality modes for a given format, try: **-q help**

**-x filename, --xmcd=filename**

The XMCD file to use for metadata for the converted tracks. If no XMCD file is given, metadata will be taken from the source tracks, if present. XMCD files can be obtained from the **track2xmcd(1)**, **cd2xmcd(1)** and **editxmcd(1)** programs.

**-d directory, --dir=directory**

The target directory for the converted tracks. If none is given, the current working directory is used (see **getcwd(3)**). If the target directory does not exist, it will be created automatically. This option is not compatible with **-o**

**--format=format string**

The format string to use for new filenames. Template fields are replaced with metadata values when new tracks are created. All other text is left as-is. If this option is omitted, a default format string is used.

**-o filename, --output=filename**

An explicit output filename for the converted track. With this option, only a single input track is allowed. It overrides the **-d** option and the default output filename.

**-j processes, --joint=processes**

The maximum number of tracks to convert at one time. If one has multiple CPUs or CPU cores, allowing **track2track(1)** to use all of them simultaneously can greatly increase encoding speed.

**-T, --thumbnail**

Convert embedded images to smaller thumbnails during conversion.

**--replay-gain**

Add ReplayGain metadata to newly created tracks

**--no-replay-gain**

Do not add ReplayGain metadata to newly created tracks

Depending on the audio format used, ReplayGain is added automatically by default when it is implemented by inserting a metadata tag. When ReplayGain is implemented by modifying audio data, it is not added automatically and must be explicitly enabled.

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

## 2.4. track2xmcd

track2xmcd(1)

track2xmcd(1)

### NAME

track2xmcd – retrieves metadata from audio tracks

### SYNOPSIS

track2xmcd [OPTIONS] <track 1> [track 2] ...

### DESCRIPTION

track2xmcd takes a list of audio tracks and retrieves an XMCD file containing their metadata such as track name, album name and artist. If no data can be found, an empty XMCD file is written which can be edited by the user.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-x filename, --xmcd=filename**

The file in which the XMCD data will be placed. **track2xmcd(1)** will refuse to overwrite an existing file if present. If no filename is given, XMCD data will be written to standard output.

**-m, --metadata**

Obtain XMCD data from the given tracks rather than obtaining it from a FreeDB server

**-s server, --freedb-server=server**

The FreeDB server name to query for XMCD data

**-p port, --freedb-port=port**

The FreeDB port number to query for XMCD data

**-i, --id** Calculate and display the disc ID rather than retrieve XMCD data

**-D, --default**

If multiple matching discs are found, select the first choice automatically

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### EXAMPLES

Obtain an XMCD file called *album.xmcd* from all the WAVE files in the current directory:

```
track2xmcd -x album.xmcd *.wav
```

Build an XMCD file from FLAC metadata and write it to standard output:

```
track2xmcd -m *.flac
```

## 2.5. editxmcd

editxmcd(1)

editxmcd(1)

### NAME

editxmcd – edits new or existing XMCD metadata files

### SYNOPSIS

editxmcd [OPTIONS] [track 1] [track 2] ...

### DESCRIPTION

editxmcd can edit existing XMCD files, or take a list of audio tracks and allows one to edit a new XMCD file from those tracks' metadata. It does not perform FreeDB lookups.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-x filename, --xmcd=filename**

The XMCD file to edit. If the XMCD file exists, that file will be edited and any audio tracks will be ignored. If the XMCD file does not exist, audio metadata will be retrieved from the given audio tracks and a new XMCD file will be created.

### EXAMPLES

Edit the data in an existing *album.xmcd*:

```
editxmcd -x album.xmcd
```

Build a new file called *album.xmcd* from all of the WAVE files in the current directory:

```
editxmcd -x album.xmcd *.wav
```

## 2.6. tracktag

tracktag(1)

tracktag(1)

### NAME

tracktag – updates audio file metadata

### SYNOPSIS

tracktag [OPTIONS] <track 1> [track 2] ...

### DESCRIPTION

tracktag takes new metadata values and a list of audio files and updates those files with the new values.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-r, --replace**

Completely replace all metadata with new values. By default, **tracktag(1)** will only update metadata values which have been specified and will leave all others as-is. This option will erase *all* existing metadata and replace it with any specified values.

**--name=TRACK NAME**

The file's track name

**--artist=ARTIST NAME**

The file's artist name

**--performer=PERFORMER NAME**

The file's performer name

**--composer=COMPOSER NAME**

The file's composer name

**--conductor=CONDUCTOR NAME**

The file's conductor name

**--album=ALBUM NAME**

The file's album name

**--catalog=CATALOG NUMBER**

The file's catalog number

**--number=TRACK NUMBER**

The file's track number

**--track-total=TRACK TOTAL**

The total number of tracks in the album

**--album-number=ALBUM NUMBER**

The file's album number, if its album is part of a set of albums

**--album-total=ALBUM TOTAL**

The total number of albums in a set of albums

**--ISRC=CODE**

The file's ISRC

**--publisher=PUBLISHER NAME**

The file's publisher

**--media-type=TYPE**

The file's media type, such as *CD*

**--year=YEAR**

The file's release year

## 2.7. trackrename

trackrename(1)

trackrename(1)

### NAME

trackrename – renames files based on metadata

### SYNOPSIS

trackrename [OPTIONS] <track 1> [track 2] ...

### DESCRIPTION

trackrename takes a list of audio files and renames them based on external or internal metadata.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-x filename, --xmcd=filename**

The filename to retrieve metadata from. If present, the tracks will not only be renamed based on the XMCD file's contents but their own metadata will be set from those contents. If omitted, the tracks will be renamed based on their own metadata contents but will not otherwise be modified.

**--format=format string**

The format string to use for new filenames. Template fields are replaced with metadata values during renaming. All other text is left as-is. If this option is omitted, a default format string is used.

tab(:);|c s||c|c||1|1|. \_ Template Fields Key:Value = %(track\_number)2.2d:the track's number on the CD %(track\_total)d:the total number of tracks on the CD %(album\_number)d:the CD's album number %(album\_total)d:the total number of CDs in the set %(album\_track\_number)s:combination of album and track number %(track\_name)s:the track's name %(album\_name)s:the album's name %(artist\_name)s:the track's artist name %(performer\_name)s:the track's performer name %(composer\_name)s:the track's composer name %(conductor\_name)s:the track's conductor name %(media)s:the track's source media %(ISRC)s:the track's ISRC %(catalog)s:the track's catalog number %(copyright)s:the track's copyright information %(publisher)s:the track's publisher %(year)s:the track's publication year %(date)s:the track's original recording date %(suffix)s:the track's suffix \_

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### EXAMPLES

Rename all of the FLAC files in the current directory based on their Vorbis Comment metadata:

```
track2rename *.flac
```

Rename all of the MP3s in *targetdir* based on the contents of *album.xmcd*.

```
trackrename -x album.xmcd targetdir/*.mp3
```

Rename *file.flac* with a custom template.

```
trackrename "--format=%(track_number)2.2d - %(album_name)s - %(track_name)s.%(suffix)s" file.flac
```

## 2.8. track2cd

track2cd(1)

track2cd(1)

### NAME

track2cd – records audio compact disc from tracks

### SYNOPSIS

track2cd [OPTIONS] <track 1> [track 2] ...

### DESCRIPTION

track2cd takes a list of audio tracks and writes their contents to an audio compact disc.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-c cdrom, --cdrom=cdrom**

The cdrom device to write the CD to

**-s speed, --speed=speed**

The speed in which to write the CD at

**--cue=filename**

A cuesheet or **cdrdao(1)** TOC file to use when writing. If given, the created CD will have the same track layout and metadata as the cuesheet. This makes the burned CD identical to the original.

**-j processes, --joint=processes**

The maximum number of tracks to convert to temporary files at one time. If one has multiple CPUs or CPU cores, allowing **track2cd(1)** to use all of them simultaneously can greatly increase transcoding speed.

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### EXAMPLE

Write all of the WAVEs in the current directory to */dev/cdrom*:

```
track2cd -c /dev/cdrom *.wav
```

## 2.9. trackinfo

trackinfo(1)

trackinfo(1)

### NAME

trackinfo – prints information and metadata from tracks

### SYNOPSIS

trackinfo [OPTIONS] <track 1> [track 2] ...

### DESCRIPTION

trackinfo takes a list of audio tracks and writes information about them to standard output. This includes track length, number of channels, sample rate, bits-per-sample, filename and additional metadata, if any.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-n, --no-metadata**

Do not print track metadata such as track name, album name and so forth

**-b, --bitrate**

Prints the average bitrate of the track in kilobytes-per-second instead of the usual track information

**-%, --percentage**

Prints the percentage compression of the track instead of the usual information

## 2.10. tracklength

tracklength(1)

tracklength(1)

### NAME

tracklength – prints the total length of all tracks

### SYNOPSIS

tracklength <track 1> [track 2] ...

### DESCRIPTION

tracklength takes a list of audio tracks and writes their total length to standard output in hours:minutes:seconds format. If directories are given, tracklength searches them recursively and adds the length of any supported audio files found to the total.

### OPTIONS

**-h, --help**

Show a list of options and exit

## 2.11. trackplay

trackplay(1)

trackplay(1)

### NAME

trackplay – plays audio tracks to an OSS or PulseAudio device

### SYNOPSIS

trackplay <track 1> [track 2] ...

### DESCRIPTION

trackinfo takes a list of audio tracks and plays them to a PulseAudio server or Open Sound System-compatible device, such as */dev/dsp*.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-T, --track-replaygain**

apply track ReplayGain during playback, if present

**-A, --album-replaygain**

apply album ReplayGain during playback, if present

**-o OUTPUT, --output=OUTPUT**

The output system to use. Open Sound System is always available. If PulseAudio is available and running, it will be the default.

### ENVIRONMENT

#### AUDIODEV

If **AUDIODEV** is set, **trackplay(1)** uses it as the device to play the given audio tracks when using OSS for output. Otherwise, the default */dev/dsp* is used.

## 2.12. record2track

record2track(1)

record2track(1)

### NAME

record2track – record audio to track

### SYNOPSIS

record2track [OPTIONS] ...

### DESCRIPTION

record2track records audio from an external source and saves it to a track.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-t type, --type=type**

The audio format to convert the recorded audio to. For a list of available audio formats, try: **-t help**

**-q quality, --quality=quality**

The desired quality of the recorded audio. For a list of available quality modes for a given format, try: **-q help**

**-o filename, --output=filename**

The output filename for the recorded track.

**--sample-rate=rate**

The sample rate to record audio at. Defaults to 44100Hz.

**--bits-per-sample=bps**

The bits-per-sample to record audio at. Defaults to 16.

**--channels=channels**

The number of channels to record audio at. Defaults to 2.

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

## 2.13. trackcmp

trackcmp(1)

trackcmp(1)

### NAME

trackcmp – compares one or more audio tracks for equivalence

### SYNOPSIS

trackcmp [OPTIONS] <track 1> <track 2>

### DESCRIPTION

trackcmp takes two audio tracks and compares their PCM data. If they are exactly the same, it prints no output and returns 0. If not, a message is displayed and 1 is returned.

trackcmp may also take two directories as arguments. In that case, any audio files in both directories are compared as per a single file. Track numbers are used to determine which file should be compared to which.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-i, --inexact**

Ignore binary silence at the start and end of files when determining whether they match

## 2.14. trackcat

trackcat(1)

trackcat(1)

### NAME

trackcat – concatenate two or more audio tracks

### SYNOPSIS

trackcat [OPTIONS] <track 1> <track 2> [track 3] ...

### DESCRIPTION

trackcat combines the audio data from two or more audio tracks into a single output track.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-t type, --type=type**

The audio format of the output track. For a list of available audio formats, try: **-t help**

**-q quality, --quality=quality**

The desired quality of the output track. For a list of available quality modes for a given format, try: **-q help**

**-o filename, --output=filename**

The output filename of the concatenated track.

**--cue=filename**

A cuesheet or **cdriao**(1) TOC file to embed in the concatenated track. At present, this feature is only supported when outputting FLAC or WavPack formatted files.

### EXAMPLE

Convert all of the WAVE files in *sourcedir* to a single FLAC file, *album.flac* at the highest possible compression (quality 8):

```
trackcat -t flac -q 8 -o album.flac sourcedir/*.wav
```

## 2.15. tracksplit

tracksplit(1)

tracksplit(1)

### NAME

tracksplit – split audio track

### SYNOPSIS

tracksplit [OPTIONS] <track>

### DESCRIPTION

tracksplit splits an audio track into several smaller tracks.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**--cue=filename**

A cuesheet or **cdrdao**(1) TOC file to use for splitting. This file contains a series of track length indices which indicate the length of each sub-track. FLAC or WavPack files with embedded cuesheets may omit this argument.

**-t type, --type=type**

The audio format to convert the split tracks to. For a list of available audio formats, try: **-t help**

**-q quality, --quality=quality**

The desired quality of the split tracks. For a list of available quality modes for a given format, try: **-q help**

**-x filename, --xmcd=filename**

The XMCD file to use for metadata for the split tracks. XMCD files can be obtained from the **track2xmcd**(1), **cd2xmcd**(1) and **editxmcd**(1) programs.

**-d directory, --dir=directory**

The target directory for the split tracks. If none is given, the current working directory is used (see **getcwd**(3)). If the target directory does not exist, it will be created automatically.

**--format=format string**

The format string to use for new filenames. Template fields are replaced with metadata values when new tracks are created. All other text is left as-is. If this option is omitted, a default format string is used.

**-j processes, --joint=processes**

The maximum number of tracks to extract at one time. If one has multiple CPUs or CPU cores, allowing **tracksplit**(1) to use all of them simultaneously can greatly increase splitting speed.

**--replay-gain**

Add ReplayGain metadata to newly created tracks

**--no-replay-gain**

Do not add ReplayGain metadata to newly split tracks

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### EXAMPLES

Split *CDImage.wav* into several WAVE files using the cuesheet *CDImage.cue*:

```
tracksplit --cue=CDImage.cue CDImage.wav
```

Split *album.flac* into several FLAC files using *album.toc* and metadata from *album.xmcd*:

```
tracksplit --cue=album.toc -x album.xmcd album.flac
```

Obtain *CDImage.cue* using the **cdrdao**(1) utilities:

## 2.16. coverview

coverview(1)

coverview(1)

### NAME

coverview – display cover images

### SYNOPSIS

coverview <track 1> [track 2] ...

### DESCRIPTION

coverview displays cover images which are embedded in audio tracks.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

## 2.17. coverdump

coverdump(1)

coverdump(1)

### NAME

coverdump – extracts cover images to files

### SYNOPSIS

coverdump [-d directory] <track>

### DESCRIPTION

coverdump takes an audio track and extracts all of its embedded cover images to individual files.

### OPTIONS

**-h, --help**

Show a list of options and exit

**--version**

Show the program's version and exit

**-d directory, --dir=directory**

The target directory for the extracted image files. If none is given, the current working directory is used.

**-V VERBOSITY, --verbose=VERBOSITY**

The level of output to display. Choose between 'normal', 'quiet' and 'debug'.

### 3. Python Reference

#### 3.1. Module Attributes

##### **AVAILABLE\_TYPES**

A tuple of AudioFile-compatible classes defined by this module.

##### **BIN**

A system binaries lookup class. This acts as a dict, taking the string of an executable and returns a string containing the path to use to execute that executable. Since the system `audiotools.cfg` config file and a user's config file may specify executables in nonstandard locations, this class will handle the task of checking for them appropriately.

##### **BUFFER\_SIZE**

An integer representing the amount of data to transfer between PCM handlers at a time.

##### **DEFAULT\_CDROM**

A string containing the default CD-ROM device to use for ripping and burning audio.

##### **FILENAME\_FORMAT**

A string formatted with Python "%" -based substitution parameters for use when creating new audio tracks.

##### **FREEDB\_PORT**

An integer of the default FreeDB port number to use.

##### **FREEDB\_SERVER**

A string of the default FreeDB server name to use.

##### **FS\_ENCODING**

The encoding to use for newly created track names written on the filesystem.

##### **IO\_ENCODING**

The encoding to use when printing information to the screen.

##### **MAX\_JOBS**

The default maximum number of processes to run at once.

##### **TYPE\_MAP**

A dict of suffix strings->AudioFile-compatible classes containing only those classes whose required executables are present on the system.

##### **VERSION**

A string identifying the module's version.

### 3.2. Module Functions

#### **open** (filename)

Takes a filename and returns an AudioFile-compatible object corresponding to the type of audio file it is. If the file is not something supported, open raises an UnsupportedFile exception.

#### **open\_files** (filename\_list, [sorted])

Takes a list of filenames and returns a list of AudioFile-compatible objects. Anything not supported is filtered out. The default behavior is to return the list sorted by track number. If sorted is False, the list is returned as-is.

#### **parse\_xmcd\_file** (filename)

Takes a filename, opens it, parses the XMCD data and returns an AlbumMetaData object.

#### **build\_xmcd\_file** (audio\_files)

Takes a list of AudioFile-compatible objects and returns XMCD file data as a string.

#### **get\_xmcd** (disc\_id, output, freedb\_server, freedb\_server\_port)

Runs the entire XMCD-fetching sequence for the given disc\_id on the given FreeDB server and port (using the web-based interface) and delivering the XMCD file to the given output file handle. If there are no matches, a blank XMCD file template will be sent. If there is one match, that XMCD file will be sent. If there are multiple matches, this function will prompt for which of them to select via stderr and stdin.

This function is obviously not suited for querying in a GUI or anything other than a command-line. However, it is a small, well-commented routine that one can use to build other FreeDB querying functions from.

#### **pcm\_cmp** (pcmreader1, pcmreader2)

Takes two PCMReader-compatible objects and compares their output. Returns True if the PCM data is identical. Returns False if not.

#### **stripped\_pcm\_cmp** (pcmreader1, pcmreader2)

Takes two PCMReader-compatible objects and compares their output, stripping any NULL samples from the beginning and end of each. Returns True if that PCM data is identical. Returns False if not.

#### **transfer\_data** (from\_function, to\_function)

from\_function is expected to take an integer number of bytes and return a string that large, or an empty string when finished. to\_function is expected to take a string. These correspond to read() and write() methods, respectively. This function transfers strings from from\_function to to\_function until empty.

### 3.3. AudioFile

#### **class AudioFile (filename)**

filename should be the path to a valid audio file on the system of the same type as this class.

#### **bits\_per\_sample ()**

Returns the number of bits per PCM sample. 16 for CD audio.

#### **channels ()**

Returns the number of distinct audio channels. 2 for CD audio.

#### **sample\_rate ()**

Returns the number of samples per second. 44100 for CD audio.

#### **total\_frames ()**

Returns the total number of audio frames in the file. Each frame is a channel-independent block of individual samples. For example, 1 second of 44100Hz audio has 44100 frames, regardless of the number of channels it has.

#### **cd\_frames ()**

Returns the length of the file in CD frames. Each CD frame is 1/75th of a second. Most formats do not need to define this method as it can be derived from the sample\_rate and total\_frames methods.

#### **lossless ()**

Returns True if this audio file is stored losslessly. Returns False if not.

#### **get\_metadata ()**

Returns a Metadata-compatible object containing the track's metadata, such as track name, track number, artist name and so forth. This may return None if the track has no metadata or its format does not support metadata.

#### **set\_metadata (metadata)**

Takes a Metadata-compatible object and sets this track's metadata to those values. If metadata is None, this method does nothing. If the audio format does not support metadata, this method also does nothing.

#### **track\_number ()**

Returns this track's position in the album, starting from 1. The metadata's track number is checked first. If none is present or the metadata has no track number, try to guess the track number from the audio file's name. If we can't make any guess, returns 0.

#### **to\_pcm ()**

Returns a PCMReader-compatible object containing a stream of this audio file's PCM data.

@classmethod

#### **from\_pcm (filename, pcmreader, [compression])**

Builds and returns an AudioFile-compatible object with the given filename from the given pcmreader and, optionally, at the given compression setting.

@classmethod

**track\_name** (track\_number, track\_metadata, album\_number = 0, format = None)

Given a track number, a MetaData-compatible object and, optionally, an album number and format string, returns a default filename for this track.

@classmethod

**can\_add\_replay\_gain** ()

Returns True if all the necessary binaries are present to add ReplayGain metadata, and the format supports such metadata.

@classmethod

**lossless\_replay\_gain** ()

Returns True if adding ReplayGain metadata is a lossless process. For example, adding ReplayGain to FLAC files simply adds a tag and doesn't modify the files themselves. Returns False if adding ReplayGain metadata modifies the original files. For example, adding ReplayGain to MP3s via the `mp3gain` program modifies the tracks' MPEG frames themselves. This 'modify tag' versus 'modify contents' behavior affects whether ReplayGain metadata is added automatically or not.

@classmethod

**add\_replay\_gain** (filenames)

Given a list of filenames of AudioFiles with of the same class as this AudioFile class, calculate and add ReplayGain metadata to those files.

**replay\_gain** ()

Returns a ReplayGain object of the track's ReplayGain values, if present. Returns None if no values or present or ReplayGain is not supported.

@classmethod

**supports\_foreign\_riff\_chunks** ()

Returns True if this AudioFile class implementation supports foreign RIFF chunks. These are arbitrary chunks of data in RIFF WAVE files which sometimes contain useful audio metadata.

**has\_foreign\_riff\_chunks** ()

Returns True if this AudioFile instance contains nonstandard RIFF chunks. These methods are useful when transcoding. For example, if the source AudioFile contains foreign RIFF chunks and the target AudioFile class supports foreign RIFF chunks, a transcoder can make special effort to preserve those chunks.

**set\_cuesheet** (cuesheet)

Takes a cuesheet-compatible object with 'catalog', 'ISRCs', 'indexes' and 'pcm\_lengths' methods. Sets this AudioFile's embedded cuesheet, if possible.

**get\_cuesheet** ()

Returns this AudioFile's embedded cuesheet as a cuesheet-compatible object with 'catalog', 'ISRCs', 'indexes' and 'pcm\_lengths' methods. If no cuesheet is embedded, returns None.

@classmethod

**has\_binaries** (system\_binaries)

Takes a `__system_binaries__` class, typically this module's BIN attribute. Checks the system executable paths (and any paths overridden with `__system_binaries__`) for all of the executables necessary to use this class. Returns True if all the binaries are present. Returns False if one or more is missing.

@classmethod

**is\_type** (file)

Takes a seekable file pointer which is rewound to the start of the file. Returns True if that header data indicates the file is of this class. Returns False if not. This is a simple magic numbers implementation so that we do not have to rely on filename suffixes to identify audio types.

### 3.4. Metadata

This is a human-readable collection of information about a single track on an album. It is designed as a lowest-common-denominator for conversion between audio formats with vastly different ways of storing track metadata.

```
class Metadata (track_name=u", track_number=0, album_name=u",  
artist_name=u", performer_name=u", composer_name=u", conductor_name=u",  
media=u", ISRC=u", catalog=u", copyright=u", publisher=u", year=u", date=u",  
album_number=0, images=[])
```

track\_number and album\_number are stored as integers. The images field, if present, should be a list of Image-compatible objects. The rest are stored as Unicode strings.

#### **track\_name**

The name of this track.

#### **track\_number**

The number of this track on the album as an integer between 1 and 99.

#### **album\_name**

The name of the album this track belongs to.

#### **artist\_name**

The name of this track's original artist.

#### **performer\_name**

The name of this track's performer.

#### **composer\_name**

The name of this track's composer.

#### **conductor\_name**

The name of this track's conductor.

#### **media**

The type of media this track comes from, such as u" CD "

#### **ISRC**

The ISRC of this track.

#### **catalog**

The catalog number of this track's album.

#### **copyright**

The copyright information for this track.

#### **publisher**

The name of this track's publisher.

## **year**

The release year for this track's album.

## **date**

The recording date of this track.

## **album\_number**

If the track's album is part of a series, this is the album's number in that series.

@classmethod

## **converted (metadata)**

This method takes a MetaData-compatible object and returns a new object of the same class. For instance, the 'converted' method of the VorbisComment class will return VorbisComment instances, or None if 'metadata' is None. The purpose of this is to remove the burden of metadata conversion from the AudioFile classes. Continuing our example, without 'converted', VorbisAudio will have different ways of handling a 'set\_metadata' call depending on whether the MetaData we're setting is already an instance of VorbisComment or not (e.g. from FlacAudio). By converting MetaData with this method, the VorbisAudio class only needs to know how to handle VorbisComment instances.

@classmethod

## **supports\_images ()**

Returns True if this particular subclass of MetaData supports embedded images. Returns False if not. Subclasses which do not support images will return empty lists when querying for images and raise exceptions if one attempts to add or delete images from them.

## **images ()**

Returns a list of our stored Images.

## **front\_covers ()**

Returns a list of our Images which are front covers.

## **back\_covers ()**

Returns a list of our Images which are back covers.

## **leaflet\_pages ()**

Returns a list of our Images which are leaflet pages.

## **media ()**

Returns a list of our Images of the album media.

## **other\_images ()**

Returns a list of our Images which aren't in the aforementioned categories.

## **add\_image (image)**

Takes an Image-compatible object and adds it to our internal list.

## **delete\_image (image)**

Takes an object, originally from our internal list, and deletes it from that list.

**class Image** (data, mime\_type, width, height, color\_depth, color\_count, description, type)

A container for image data. This is often the base class for metadata-specific embedded images.

**data**

A string containing the binary data that comprises this image.

**mime\_type**

A Unicode string storing the MIME type of this image, such as 'image/jpeg' or 'image/png'.

**width**

How wide this image is as an integer number of pixels.

**height**

How tall this image is as an integer number of pixels.

**color\_depth**

The number of colors this image contains as an integer number of bits-per-pixel.

**color\_count**

For palette-based images (such as GIF), the total number of colors the image contains. For images with no palette, this value is 0.

**description**

A description of the image as a Unicode string.

**type**

What portion of an album this image is of, as an integer.

- 0 | Front Cover
- 1 | Back Cover
- 2 | Leaflet Page
- 3 | Media
- 4 | Other

**type\_string ()**

Returns what portion of an album this image is of, as a string.

**suffix ()**

Returns a string of this image's file suffix, based on its MIME type. 'image/jpeg' returns a suffix of 'jpg', for example.

@classmethod

**new** (image\_data, description, type)

Given a string of image data, a Unicode description string and integer type, returns an Image object with the remaining fields filled-in based upon that image data. This allows a user to add a JPEG to an audio track, for example, by having the rest of the image information retrieved automatically based on its contents rather than having to specify those metrics.

**thumbnail** (width, height, format)

Given maximum size width and height integers and a format string (such as 'jpeg'), returns a new Image object formatted to the given parameters.

**class ReplayGain** (track\_gain, track\_peak, album\_gain, album\_peak)

A simple container class for ReplayGain metadata. All values are floats.

**track\_gain**

This track's ReplayGain value, in dB.

**track\_peak**

This track's peak value. This is used to prevent clipping.

**album\_gain**

The entire album's ReplayGain value, in dB.

**album\_peak**

The entire album's peak value.

### 3.5. Cuesheets

Although there is currently no base Cuesheet class, those expected by an AudioFile's 'set\_cuesheet' method and returned by one's 'set\_cuesheet' method should implement the following methods:

#### **catalog ()**

Returns a CD's catalog number as a plain string, or None if no catalog number is present in the cuesheet.

#### **ISRCs ()**

Returns a CD's set of ISRCs as a dict whose keys are track number integers and whose values are ISRC plain strings.

#### **indexes ()**

Returns an iterator of index tuples corresponding to the track indexes.

#### **pcm\_lengths (total\_length)**

Given the total length of the entire CD in PCM frames, returns a list of PCM lengths for all audio tracks within the cuesheet in PCM frames.

### 3.6. PCMReader

**class PCMReader** (file, sample\_rate, channels, bits\_per\_sample, [process])

A PCM generator class. It is designed to function much like a file reader object, but with additional metadata about the PCM data being generated.

**read** (bytes)

Returns a number of "bytes" of data from the PCM stream, as a string. We're not guaranteed to return exactly that amount of bytes, depending on the implementation of a given PCMReader. The stream is finished when the number of bytes is 0.

**close** ()

Closes the PCMReader. If a process is associated with the PCMReader, that process is finished upon calling close.

**file**

A file-like object with read and close methods.

**sample\_rate**

The PCM stream's number of samples-per-second. 44100 for CD audio.

**channels**

The PCM stream's number of channels. 2 for CD audio.

**bits\_per\_sample**

The PCM stream's number of bits per sample. 16 for CD audio.

**process**

An optional subprocess-compatible object. If not None, its wait method will be called when the PCMReader is closed.

### 3.7. PCMConverter

**class PCMConverter** (pcmreader, sample\_rate, channels, bits\_per\_sample)

This is a PCMReader-compatible class which takes a PCMReader and converts its PCM stream to a new format with the given attributes using resampling, dithering and channel modification, if necessary. For example, it is not possible to create a CD from a PCMReader with a sample rate of 96000Hz, 6 channels and 24 bits-per-sample because CD audio must have a sample rate of 44100Hz, 2 channels and 16 bits-per-sample. But by wrapping PCMConverter around that PCMReader, like:

```
PCMConverter(pcmreader, 44100, 2, 16)
```

PCMConveter's read method will translate the stream to a CD-quality one.

### 3.8. CDDA

#### **class CDDA** (device\_name, [speed])

This class represents an audio CD at the given device and, optionally, at the given speed. device\_name is a string containing a CD-ROM device (e.g. "/dev/cdrom"). speed is an integer rate of speed at which to access that device.

#### **\_\_len\_\_** ()

Returns the number of tracks on the CD

#### **\_\_getitem\_\_** (track\_number)

Returns a CDTrackReader object for the given track number. Raises an IndexError if one tries to retrieve a track beyond the total number available, or a track number below 1.

#### **\_\_iter\_\_** ()

Generates an iterator over all of the tracks on this CD, as CDTrackReader objects.

#### **class CDTrackReader** (cdda, track\_number)

This is a PCMReader-compatible object containing the PCM data for the given track number on the given CDDA object. CDTrackReader should be retrieved from a CDDA object rather than instantiated directly.

#### **length** ()

Returns the length of this track in CD frames.

#### **offset** ()

Returns the total offset of this track from the beginning on the CD, in CD frames.

#### **read** (bytes)

Returns a sector-aligned number of bytes from the given track. This method returns as many sectors as possible to fulfill the requested number of bytes, or at least 1 sector if the number of bytes is too small. It returns an empty string when the sectors have been exhausted.

#### **close** ()

Closes this track for reading.

#### **rip\_log**

A CDTrackLog object containing the log data for this track.

## **class CDTrackLog ()**

This is a container for log data generated by CDTrackReader during sector extraction. It is created and set as a global cdio callback automatically by CDTrackReader. Because of this, only one CDTrackLog can be receiving log data from CD extraction at a time. This prohibits multiple CDTrackReaders from operating in the same Python interpreter at the same time (if one wishes to rip data from 2 CDs at once, for instance). Short of a major libcdio design change, there is little that can be done to eliminate this restriction.

### **\_\_str\_\_ ()**

Returns the log information as a string, formatted to be similar to that generated by the cdda2wav program.

### 3.9. XMCD

CD audio discs do not typically come with information about the album name, artist name, track names and so forth; they only contain audio data split into tracks. Therefore, that information must come from an external source. FreeDB is an open source for audio data with a wide coverage of albums and easy protocols to work with. It outputs XMCD files. Therefore, XMCD files are what Python Audio Tools use for external metadata not contained in the tracks themselves. If we cannot or do not wish to use FreeDB's XMCD files, ones can be created from AudioFiles.

#### **class XMCD** (values, offsets, length)

Our internal representation of an XMCD file. "values" is a dict of key->value pairs where key is an XMCD key string and value is its unicode value. For example, "TTITLE0":u"Track Name". "offsets" is a list of track offset integers, in CD frames. "length" is the total length of the album (including lead-in) in seconds.

This implements a dict interface with `__getitem__`, `__setitem__`, keys methods, etc. These calls are forwarded to its internal values dict.

#### **offsets**

Our offsets int list

#### **length**

Our length int

#### **build** ()

Returns an XMCD file string with values taken from our internal representation. This file will be in either ISO-8859-1 format, or UTF-8 format, depending on its contents.

#### **metadata** ()

Returns an AlbumMetaData object of our internal representation.

@classmethod

#### **read** (filename)

Takes a filename and returns an XMCD object. Raises IOError if the file can't be read. Raises XMCDException if a problem is encountered with the file's syntax.

@classmethod

#### **read\_data** (data)

Takes a unicode string of XMCD data and returns an XMCD object. Raises XMCDException if a problem is encountered with the data's syntax.

@classmethod

#### **from\_files** (audiofiles)

Takes a list of AudioFile-compatible objects from the same album, ordered by track\_number. Returns a corresponding XMCD object best matching that album from the tracks' lengths and metadata - if any.

## **class AlbumMetaData ()**

A track number->MetaData dictionary.

### **\_\_getitem\_\_ (track\_number)**

Returns a MetaData-compatible object for the given integer track\_number. Throws a KeyError if the track\_number is not in AlbumMetaData. track\_numbers start from 1.

## **class DiscID ([tracks])**

One must submit disc track information to FreeDB in order to receive an XMCD file. DiscID is a container object for that information. It can be created either from the CD audio disc itself, or from tracks ripped from that disc. If present, the "tracks" argument should be a list of track lengths, in CD frames.

### **add (track)**

Takes the length of a track in CD frames and adds it to this DiscID.

### **offsets ()**

Returns a list of CD track offsets based on the DiscID's track lengths.

### **idsuffix ()**

Returns a string containing the CD's number of tracks, track offsets and total length in seconds.

### **\_\_str\_\_ ()**

Returns a string containing the 32-bit FreeDB disc ID calculated from the track lengths.

### **toxmcid ()**

Returns an empty XMCD file as a string, suitable for populating by the user.

### **class FreeDB** (servername, port)

This is a connection to the FreeDB server at the given servername and port. It uses FreeDB's native protocol rather than the web-based one. Prints status information to stderr while it operates.

#### **connect ()**

Performs the socket connection to the FreeDB server.

#### **close ()**

Closes the socket connection to the FreeDB server.

#### **write (line)**

Writes a command line to the FreeDB server.

#### **read ()**

Reads a response line from the FreeDB server and returns the response code and response data as a tuple.

#### **query (disc\_id)**

Takes a DiscID object. Queries the FreeDB server for any or all matches and returns them as a list of category/id tuples.

#### **read\_data (category, id, output)**

Takes the category and id fields, as returned by query(), retrieves the XMCD data of that disc from the FreeDB server and sends it to output, which must have a write() method.

### **class FreeDBWeb** (servername, port)

This is a web-based connection to a FreeDB server. It is a subclass of FreeDB with all the same methods and can be used interchangeably.