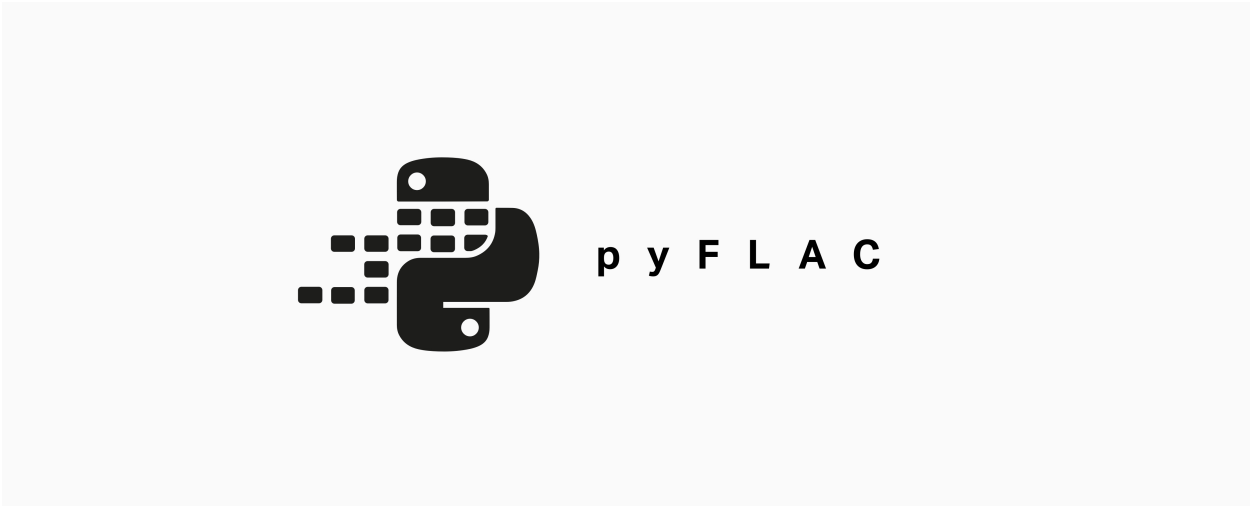
pyFLAC

Joe Todd

Apr 16, 2024

CONTENTS



A simple Pythonic interface for libFLAC.

> FLAC stands for Free Lossless Audio Codec, an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality. This is similar to how Zip works, except with FLAC you will get much better compression because it is designed specifically for audio.

pyFLAC allows you to encode and decode raw audio data directly to/from a file, or in real-time using callbacks.

You can use pip to download and install the latest release with a single command.

```
pip3 install pyflac
```

Note: pyFLAC depends on `libsndfile`, which requires an extra install step on Linux distributions. See the SoundFile documentation for more information.

CHAPTER

ONE

SUPPORTED PLATFORMS

- **macOS** (Intel/Apple Silicon)
- **Linux** (x86_64/arm64)
- **RPi** Zero/2/3/4
- **Windows** 8/10/11

CHAPTER

TWO

CLI

pyFLAC comes bundled with a command line tool to quickly convert between WAV and FLAC files. For more information, print the help info.

```
pyflac --help
```

Note: If you didn't install pyFLAC globally then the command line tool will not be installed on your PATH. However you should still be able to access the tool with `python3 -m pyflac`.

CHAPTER

THREE

EXAMPLES

`passthrough.py`

Read a WAV file and pass the audio through the encoder/decoder for the purposes of illustration.

```
python3 passthrough.py
```

This example asserts that the uncompressed data is exactly equal to the original signal.

`stream.py`

Stream audio from the microphone input and pass through the encoder printing the effectiveness of the compression to the terminal.

```
python3 stream.py
```

Note: This example requires `sounddevice`, which can be installed with pip. See the sounddevice documentation for more information.

CHAPTER

FOUR

LIMITATIONS

- pyFLAC only supports 16-bit and 32-bit audio.
- FLAC metadata handling is not implemented.
- The built in libraries do not include OGG support.

4.1 API Reference

CHAPTER

FIVE

ENCODER

To encode raw audio data with pyFLAC you can either write encoded data directly to a file or process in real-time.

class pyflac.**FileEncoder**(*input_file: Path*, *output_file: Path = None*, *compression_level: int = 5*, *blocksize: int = 0*, *streamable_subset: bool = True*, *verify: bool = False*)

The pyFLAC file encoder reads the raw audio data from the WAV file and writes the encoded audio data to a FLAC file.

Note that the input WAV file must be either PCM_16 or PCM_32.

Parameters

- **input_file** (*pathlib.Path*) – Path to the input WAV file
- **output_file** (*pathlib.Path*) – Path to the output FLAC file, a temporary file will be created if unspecified.
- **compression_level** (*int*) – The compression level parameter that varies from 0 (fastest) to 8 (slowest). The default setting is 5, see https://en.wikipedia.org/wiki/FLAC for more details.
- **blocksize** (*int*) – The size of the block to be returned in the callback. The default is 0 which allows libFLAC to determine the best block size.
- **streamable_subset** (*bool*) – Whether to use the streamable subset for encoding. If true the encoder will check settings for compatibility. If false, the settings may take advantage of the full range that the format allows.
- **verify** (*bool*) – If `True`, the encoder will verify it's own encoded output by feeding it through an internal decoder and comparing the original signal against the decoded signal. If a mismatch occurs, the *process* method will raise a *EncoderProcessException*. Note that this will slow the encoding process by the extra time required for decoding and comparison.

Raises

ValueError – If any invalid values are passed in to the constructor.

process() → bytes

Process the audio data from the WAV file.

Returns

(bytes) – The FLAC encoded bytes.

Raises

EncoderProcessException – if an error occurs when processing the samples

class pyflac.**StreamEncoder**(*sample_rate: int*, *write_callback: Callable[[bytes, int, int, int], None]*, *seek_callback: Callable[[int], None] = None*, *tell_callback: Callable[[], int] = None*, *metadata_callback: Callable[[int], None] = None*, *compression_level: int = 5*, *blocksize: int = 0*, *streamable_subset: bool = True*, *verify: bool = False*, *limit_min_bitrate: bool = False*)

The pyFLAC stream encoder is used for real-time compression of raw audio data.

Raw audio data is passed in via the *process* method, and chunks of compressed data is passed back to the user via the `write_callback`.

Parameters

- **sample_rate** (*int*) – The raw audio sample rate (Hz)
- **write_callback** (*fn*) – Function to call when there is compressed data ready, see the example below for more information.
- **seek_callback** (*fn*) – Optional function to call when the encoder wants to seek within the output file.
- **tell_callback** (*fn*) – Optional function to call when the encoder wants to find the current position within the output file.
- **compression_level** (*int*) – The compression level parameter that varies from 0 (fastest) to 8 (slowest). The default setting is 5, see https://en.wikipedia.org/wiki/FLAC for more details.
- **blocksize** (*int*) – The size of the block to be returned in the callback. The default is 0 which allows libFLAC to determine the best block size.
- **streamable_subset** (*bool*) – Whether to use the streamable subset for encoding. If true the encoder will check settings for compatibility. If false, the settings may take advantage of the full range that the format allows.
- **verify** (*bool*) – If `True`, the encoder will verify its own encoded output by feeding it through an internal decoder and comparing the original signal against the decoded signal. If a mismatch occurs, the *process* method will raise a *EncoderProcessException*. Note that this will slow the encoding process by the extra time required for decoding and comparison.
- **limit_min_bitrate** (*bool*) – If `True`, the encoder will not output frames which contain only constant subframes, which can be beneficial for streaming applications.

Examples

An example write callback which adds the encoded data to a queue for later processing.

```
def write_callback(self,
                   buffer: bytes,
                   num_bytes: int,
                   num_samples: int,
                   current_frame: int):
    if num_samples == 0:
        # If there are no samples in the encoded data, this is
        # a FLAC header. The header data will arrive in several
        # different callbacks. Otherwise `num_samples` will be
        # the block size value.
        pass

    self.queue.append(buffer)
    self.total_bytes += num_bytes
```

Raises

ValueError – If any invalid values are passed in to the constructor.

finish() → bool

Finish the encoding process. This flushes the encoding buffer, releases resources, resets the encoder settings to their defaults, and returns the encoder state to `EncoderState.UNINITIALIZED`.

A well behaved program should always call this at the end.

Returns
(bool) – `True` if successful, `False` otherwise.

process(*samples: numpy.ndarray*)

Process some samples.

This method ensures the samples are contiguous in memory and then passes a pointer to the numpy array to the FLAC encoder to process.

On processing the first buffer of samples, the encoder is set up for the given amount of channels and data type. This is automatically determined from the numpy array.

Raises

- **TypeError** – if a numpy array of samples is not provided
- ***EncoderProcessException*** – if an error occurs when processing the samples

property state: ***EncoderState***

Property to return the encoder state

Type
EncoderState

CHAPTER

SIX

DECODER

To decode compressed data with pyFLAC you can either read the compressed data directly from a file or process in real-time.

class pyflac.**FileDecoder**(*input_file: Path*, *output_file: Path = None*)

The pyFLAC file decoder reads the encoded audio data directly from a FLAC file and writes to a WAV file.

Parameters

- **input_file** (*pathlib.Path*) – Path to the input FLAC file
- **output_file** (*pathlib.Path*) – Path to the output WAV file, a temporary file will be created if unspecified.

Raises

DecoderInitException – If initialisation of the decoder fails

process() → Tuple[numpy.ndarray, int]

Process the audio data from the FLAC file.

Returns

(tuple) – A tuple of the decoded numpy audio array, and the sample rate of the audio data.

Raises

DecoderProcessException – if any fatal read, write, or memory allocation error occurred (meaning decoding must stop)

class pyflac.**StreamDecoder**(*write_callback: Callable[[numpy.ndarray, int, int, int], None]*)

A pyFLAC stream decoder converts a stream of FLAC encoded bytes back to raw audio data.

The compressed data is passed in via the *process* method, and blocks of raw uncompressed audio is passed back to the user via the `callback`.

The *finish* method must be called at the end of the decoding process, otherwise the processing thread will be left running.

Parameters

write_callback (*fn*) – Function to call when there is uncompressed audio data ready, see the example below for more information.

Examples

An example callback which writes the audio data to file using SoundFile.

```
import soundfile as sf

def callback(self,
             audio: np.ndarray,
             sample_rate: int,
             num_channels: int,
             num_samples: int):

    # ------------------------------------------------------
    # Note: num_samples is the number of samples per channel
    # ------------------------------------------------------
    if self.output is None:
        self.output = sf.SoundFile(
            'output.wav', mode='w', channels=num_channels,
            samplerate=sample_rate
        )
    self.output.write(audio)
```

Raises
DecoderInitException – If initialisation of the decoder fails

finish()

Finish the decoding process.

This must be called at the end of the decoding process.

Flushes the decoding buffer, closes the processing thread, releases resources, resets the decoder settings to their defaults, and returns the decoder state to `DecoderState.UNINITIALIZED`.

Raises
DecoderProcessException – if any fatal read, write, or memory allocation error occurred.

process(*data: bytes*)

Instruct the decoder to process some data.

Note: This is a non-blocking function, data is processed in a background thread.

Parameters
data (*bytes*) – Bytes of FLAC data

property state: *DecoderState*

Property to return the decoder state

Type
DecoderState

class pyflac.**OneShotDecoder**(*write_callback: Callable[[numpy.ndarray, int, int, int], None]*, *buffer: bytes*)

A pyFLAC one-shot decoder converts a buffer of FLAC encoded bytes back to raw audio data. Unlike the *StreamDecoder* class, the one-shot decoder operates on a single block of data, and runs in a blocking manner, as opposed to in a background thread.

The compressed data is passed in via the constructor, and blocks of raw uncompressed audio is passed back to the user via the `callback`.

Parameters

- **write_callback** (*fn*) – Function to call when there is uncompressed audio data ready, see the example below for more information.
- **buffer** (*bytes*) – The FLAC encoded audio data

Examples

An example callback which writes the audio data to file using SoundFile.

```
import soundfile as sf

def callback(self,
             audio: np.ndarray,
             sample_rate: int,
             num_channels: int,
             num_samples: int):

    # ----------------------------------------------------
    # Note: num_samples is the number of samples per channel
    # ----------------------------------------------------
    if self.output is None:
        self.output = sf.SoundFile(
            'output.wav', mode='w', channels=num_channels,
            samplerate=sample_rate
        )
    self.output.write(audio)
```

Raises

DecoderInitException – If initialisation of the decoder fails

CHAPTER

SEVEN

STATE

class pyflac.**EncoderState**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

The encoder state as a Python enumeration

class pyflac.**DecoderState**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

The decoder state as a Python enumeration

CHAPTER

EIGHT

EXCEPTIONS

class pyflac.**EncoderInitException**(*code*)

An exception raised if initialisation fails for a *StreamEncoder* or a *FileEncoder*.

class pyflac.**EncoderProcessException**

An exception raised if an error occurs during the processing of audio data.

class pyflac.**DecoderInitException**(*code*)

An exception raised if initialisation fails for a *StreamDecoder* or a *FileDecoder*.

class pyflac.**DecoderProcessException**

An exception raised if an error occurs during the processing of audio data.

8.1 Development

CHAPTER

NINE

CONTRIBUTING

If you find any bugs or other things that need improvement, or would like to add additional features, please create an issue or a pull request at https://github.com/sonos/pyFLAC.

You get started, grab the latest version of the code from GitHub:

```
git clone https://github.com/sonos/pyFLAC.git
cd pyflac
```

To build the package:

```
python3 pyflac/builder/encoder.py

python3 pyflac/builder/decoder.py
```

you can also install your local copy with pip:

```
pip3 install .
```

Before submitting a pull request, make sure all tests are passing and the test coverage has not decreased.

CHAPTER

TEN

TESTING

To run the test suite:

```
tox -r
```

CHAPTER

ELEVEN

DOCUMENTATION

If you make changes to the documentation, you can locally re-create the HTML pages using Sphinx. You can install it and the read the docs theme with:

```
pip3 install -r docs/requirements.txt
```

To create the HTML pages, use:

```
cd docs
make html
```

The generated files will be available in the directory `docs/_build/html`.

CHAPTER

TWELVE

PYFLAC CHANGELOG

v3.0.0

- Fixed bug in the shutdown behaviour of the `StreamDecoder` (see #22 and #23).
- Automatically detect bit depth of input data in the `FileEncoder`, and raise an error if not 16-bit or 32-bit PCM (see #24).
- Added a new `OneShotDecoder` to decode a buffer of FLAC data in a single blocking operation, without the use of threads. Courtesy of @GOAE.

v2.2.0

- **Updated FLAC library to v1.4.3.**
  See FLAC Changelog.
- Added support for `int32` data
- Added `limit_min_bitrate` property.
- Removed support for Python 3.7

v2.1.0

- Added support for Linux `arm64` architectures
- Added support for Darwin `arm64` architectures (macOS Apple Silicon)
- Fixed Raspberry Pi Zero library (see #13)
- Updated FLAC library to v1.3.4

v2.0.0

- Added `seek` and `tell` callbacks to `StreamEncoder`
- Renamed the write callbacks from `callback` to `write_callback` for `StreamEncoder` and `StreamDecoder`

v1.0.0

- Added a `StreamEncoder` to compress raw audio data on-the-fly into a FLAC byte stream
- Added a `StreamDecoder` to decompress a FLAC byte stream back to raw audio data
- Added a `FileEncoder` to convert a WAV file to FLAC encoded data, optionally saving to a FLAC file
- Added a `FileDecoder` to convert a FLAC file to raw audio data, optionally saving to a WAV file
- Bundled with libFLAC version 1.3.3

12.1 License

pyFLAC is distributed under an Apache 2.0 license allowing users to use the software for any purpose, to distribute it and to modify it.

pyFLAC includes prebuilt binaries of libFLAC for different architectures, these binaries are distributed under the following libFLAC license.

```
Copyright (C) 2000-2009  Josh Coalson
Copyright (C) 2020-2016  Xiph.Org Foundation

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

- Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

- Neither the name of the Xiph.org Foundation nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE FOUNDATION OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

12.2 Index

- genindex

INDEX