PySoundFile Documentation

Release 0.6.0

Bastian Bechtold, Matthias Geier

May 02, 2020

Contents

PySoundFile is an audio library based on libsndfile, CFFI and Numpy. Full documentation is available on pysoundfile.readthedocs.org.

PySoundFile can read and write sound files. File reading/writing is supported through libsndfile, which is a free, cross-platform, open-source (LGPL) library for reading and writing many different sampled sound file formats that runs on many platforms including Windows, OS X, and Unix. It is accessed through CFFI, which is a foreign function interface for Python calling C code. CFFI is supported for CPython 2.6+, 3.x and PyPy 2.0+. PySoundFile represents audio data as NumPy arrays.

1 Breaking Changes

The latest release of PySoundFile cleans up many small inconsistencies, particularly in the the ordering and naming of function arguments. Therefore, old code will probably not work any more.

It also adds a number of great new features, such as global `read` and `write` functions that do not require you to open a `SoundFile`, or a `blocks` function that can read a sound file one block at a time. It has also grown a lot more flexible and powerful at opening things like streams, buffers, or file descriptors.

With all these improvements, we feel that the indexing interface is not needed any more. It is now officially marked as deprecated and might be removed in the future.

2 Installation

PySoundFile depends on the Python packages CFFI and Numpy, and the system library libsndfile.

To install the Python dependencies, I recommend using the Anaconda Distribution of Python. Anaconda provides the `conda` package manager, which will install all dependencies using `conda install cffi numpy` (conda is also independently available on pip).

You will also need to install libsndfile. On Windows, libsndfile is included in the binary installers (see below). On OS X, homebrew can install libsndfile using `brew install libsndfile`. On Linux, use your distribution's package manager, for example `sudo apt-get install libsndfile`.

With CFFI, Numpy, and libsndfile installed, you can use pip to install PySoundFile with `pip install pysoundfile` or `pip install pysoundfile --user` if you don't have administrator privileges. If you are running Windows you should download the Windows installers for PySoundFile instead (which also include libsndfile):

PySoundFile-0.6.0.win-amd64-py2.7
PySoundFile-0.6.0.win-amd64-py3.4
PySoundFile-0.6.0.win32-py2.7
PySoundFile-0.6.0.win32-py3.4

3 Read/Write Functions

Data can be written to the file using `write()`, or read from the file using `read()`. PySoundFile can open all file formats that libsndfile supports, for example WAV, FLAC, OGG and MAT files.

Here is an example for a program that reads a wave file and copies it into an ogg-vorbis file:

```
import pysoundfile as sf

data, samplerate = sf.read('existing_file.wav')
sf.write(data, 'new_file.ogg', samplerate=samplerate)
```

4 Block Processing

Sound files can also be read in short, optionally overlapping blocks. For example, this calculates the signal level for each block of a long file:

```
import numpy as np
import pysoundfile as sf

rms = [np.sqrt(np.mean(block**2)) for block in
       sf.blocks('myfile.wav', blocksize=1024, overlap=512)]
```

5 SoundFile Objects

Sound files can also be opened as SoundFile objects. Every SoundFile has a specific sample rate, data format and a set number of channels.

If a file is opened, it is kept open for as long as the SoundFile object exists. The file closes when the object is garbage collected, but you should use the `close()` method or the context manager to close the file explicitly:

```
import pysoundfile as sf

with sf.SoundFile('myfile.wav', 'rw') as f:
    while f.tell() < len(f):
        pos = f.tell()
        data = f.read(1024)
        f.seek(pos)
        f.write(data*2)
```

All data access uses frames as index. A frame is one discrete time-step in the sound file. Every frame contains as many samples as there are channels in the file.

6 RAW Files

Pysoundfile can usually auto-detect the file type of sound files. This is not possible for RAW files, though. This is a useful idiom for opening RAW files without having to provide all the format for every file:

```
import pysoundfile as sf

format = {'format':'RAW', 'subtype':'FLOAT', 'endian':'FILE'}
data = sf.read('myfile.raw', dtype='float32', **format)
sf.write(data, 'otherfile.raw', **format)
```

7 Virtual IO

If you have an open file-like object, Pysoundfile can open it just like regular files:

```
import pysoundfile as sf
with open('filename.flac', 'rb') as f:
    data, samplerate = sf.read(f)
```

Here is an example using an HTTP request:

```
from io import BytesIO
import pysoundfile as sf
import requests

f = BytesIO()
response = requests.get('http://www.example.com/my.flac', stream=True)
for data in response.iter_content(4096):
    if data:
        f.write(data)
f.seek(0)
data, samplerate = sf.read(f)
```

8 Accessing File Metadata

In addition to audio data, there are a number of text fields in some sound files. In particular, you can set a title, a copyright notice, a software description, the artist name, a comment, a date, the album name, a license, a track number and a genre. Note however, that not all of these fields are supported for every file format.

9 News

2013-08-27 V0.1.0 Bastian Bechtold: Initial prototype. A simple wrapper for libsndfile in Python

2013-08-30 V0.2.0 Bastian Bechtold: Bugfixes and more consistency with PySoundCard

2013-08-30 V0.2.1 Bastian Bechtold: Bugfixes

2013-09-27 V0.3.0 Bastian Bechtold: Added binary installer for Windows, and context manager

2013-11-06 V0.3.1 Bastian Bechtold: Switched from distutils to setuptools for easier installation

2013-11-29 V0.4.0 Bastian Bechtold: Thanks to David Blewett, now with Virtual IO!

2013-12-08 V0.4.1 Bastian Bechtold: Thanks to Xidorn Quan, FLAC files are not float32 any more.

2014-02-26 V0.5.0 Bastian Bechtold: Thanks to Matthias Geier, improved seeking and a flush() method.

2015-01-19 V0.6.0 Bastian Bechtold: A big, big thank you to Matthias Geier, who did most of the work!

- Switched to `float64` as default data type.
- Function arguments changed for consistency.
- Added unit tests.
- Added global `read()`, `write()`, `blocks()` convenience functions.
- Documentation overhaul and hosting on readthedocs.
- Added `'x'` open mode.
- Added `tell()` method.
- Added `__repr__()` method.

10 API Documentation

PySoundFile is an audio library based on libsndfile, CFFI and NumPy.

Sound files can be read or written directly using the functions *read()* and *write()*. To read a sound file in a block-wise fashion, use *blocks()*. Alternatively, sound files can be opened as *SoundFile* objects.

For further information, see http://pysoundfile.readthedocs.org/.

pysoundfile.**read**(*file*, *frames=-1*, *start=0*, *stop=None*, *dtype='float64'*, *always_2d=True*, *fill_value=None*, *out=None*, *samplerate=None*, *channels=None*, *format=None*, *subtype=None*, *endian=None*, *closefd=True*)

Provide audio data from a sound file as NumPy array.

By default, the whole file is read from the beginning, but the position to start reading can be specified with *start* and the number of frames to read can be specified with *frames*. Alternatively, a range can be specified with *start* and *stop*.

If there is less data left in the file than requested, the rest of the frames are filled with *fill_value*. If no *fill_value* is specified, a smaller array is returned.

Parameters

- **file** (*str or int or file-like object*) – The file to read from. See *SoundFile* for details.
- **frames** (*int, optional*) – The number of frames to read. If *frames* is negative, the whole rest of the file is read. Not allowed if *stop* is given.
- **start** (*int, optional*) – Where to start reading. A negative value counts from the end.
- **stop** (*int, optional*) – The index after the last frame to be read. A negative value counts from the end. Not allowed if *frames* is given.
- **dtype** (*{'float64', 'float32', 'int32', 'int16'}, optional*) – Data type of the returned array, by default `'float64'`. Floating point audio data is typically in the range from `-1.0` to `1.0`. Integer data is in the range from `-2**15` to `2**15-1` for `'int16'` and from `-2**31` to `2**31-1` for `'int32'`.

Returns

- **audiodata** (*numpy.ndarray or type(out)*) – A two-dimensional NumPy array is returned, where the channels are stored along the first dimension, i.e. as columns. A two-dimensional array is returned even if the sound file has only one channel. Use `always_2d=False` to return a one-dimensional array in this case.

  If *out* was specified, it is returned. If *out* has more frames than available in the file (or if *frames* is smaller than the length of *out*) and no *fill_value* is given, then only a part of *out* is overwritten and a view containing all valid frames is returned.
- **samplerate** (*int*) – The sample rate of the audio file.

Other Parameters

- **always_2d** (*bool, optional*) – By default, audio data is always returned as a two-dimensional array, even if the audio file has only one channel. With `always_2d=False`, reading a mono sound file will return a one-dimensional array.
- **fill_value** (*float, optional*) – If more frames are requested than available in the file, the rest of the output is be filled with *fill_value*. If *fill_value* is not specified, a smaller array is returned.

- **out** (*numpy.ndarray or subclass, optional*) – If *out* is specified, the data is written into the given array instead of creating a new array. In this case, the arguments *dtype* and *always_2d* are silently ignored! If *frames* is not given, it is obtained from the length of *out*.
- **samplerate, channels, format, subtype, endian, closefd** – See *SoundFile*.

Examples

```
>>> import pysoundfile as sf
>>> data, samplerate = sf.read('stereo_file.wav')
>>> data
array([[ 0.71329652,  0.06294799],
       [-0.26450912, -0.38874483],
       ...
       [ 0.67398441, -0.11516333]])
>>> samplerate
44100
```

pysoundfile.**write**(*data*, *file*, *samplerate*, *subtype=None*, *endian=None*, *format=None*, *closefd=True*, *exclusive_creation=True*)

Write data to a sound file.

Parameters

- **data** (*array_like*) – The data to write. Usually two-dimensional (channels x frames), but one-dimensional *data* can be used for mono files. Only the data types `'float64'`, `'float32'`, `'int32'` and `'int16'` are supported.

  Note: The data type of *data* does **not** select the data type of the written file. Audio data will be converted to the given *subtype*.

- **file** (*str or int or file-like object*) – The file to write to. See *SoundFile* for details.
- **samplerate** (*int*) – The sample rate of the audio data.
- **subtype** (*str, optional*) – See *default_subtype()* for the default value and *available_subtypes()* for all possible values.

Other Parameters

- **exclusive_creation** (*bool*) – If `True` (the default), the file is opened with `mode='x'`. Otherwise, it is opened with `mode='w'`.
- **format, endian, closefd** – See *SoundFile*.

Examples

Write 10 frames of random data to a file:

```
>>> import numpy as np
>>> import pysoundfile as sf
>>> sf.write(np.random.randn(10, 2), 'stereo_file.wav', 44100, 'PCM_24')
```

pysoundfile.**blocks**(*file*, *blocksize=None*, *overlap=0*, *frames=-1*, *start=0*, *stop=None*, *dtype='float64'*, *always_2d=True*, *fill_value=None*, *out=None*, *samplerate=None*, *channels=None*, *format=None*, *subtype=None*, *endian=None*, *closefd=True*)

Return a generator for block-wise reading.

By default, iteration starts at the beginning and stops at the end of the file. Use *start* to start at a later position and *frames* or *stop* to stop earlier.

If you stop iterating over the generator before it's exhausted, the sound file is not closed. This is normally not a problem because the file is opened in read-only mode. To close the file properly, the generator's `close()` method can be called.

Parameters

- **file** (*str or int or file-like object*) – The file to read from. See *SoundFile* for details.
- **blocksize** (*int*) – The number of frames to read per block. Either this or *out* must be given.
- **overlap** (*int, optional*) – The number of frames to rewind between each block.

Yields *numpy.ndarray or type(out)* – Blocks of audio data. If *out* was given, and the requested frames are not an integer multiple of the length of *out*, and no *fill_value* was given, the last block will be a smaller view into *out*.

Other Parameters

- **frames, start, stop** – See *read()*.
- **dtype** (*{'float64', 'float32', 'int32', 'int16'}, optional*) – See *read()*.
- **always_2d, fill_value, out** – See *read()*.
- **samplerate, channels, format, subtype, endian, closefd** – See *SoundFile*.

Examples

```
>>> import pysoundfile as sf
>>> for block in sf.blocks('stereo_file.wav', blocksize=1024):
>>>     pass  # do something with 'block'
```

pysoundfile.**available_formats**()

Return a dictionary of available major formats.

Examples

```
>>> import pysoundfile as sf
>>> sf.available_formats()
{'FLAC': 'FLAC (FLAC Lossless Audio Codec)',
 'OGG': 'OGG (OGG Container format)',
 'WAV': 'WAV (Microsoft)',
 'AIFF': 'AIFF (Apple/SGI)',
 ...
 'WAVEX': 'WAVEX (Microsoft)',
 'RAW': 'RAW (header-less)',
 'MAT5': 'MAT5 (GNU Octave 2.1 / Matlab 5.0)'}
```

pysoundfile.**available_subtypes**(*format=None*)

Return a dictionary of available subtypes.

Parameters **format** (*str*) – If given, only compatible subtypes are returned.

Examples

```
>>> import pysoundfile as sf
>>> sf.available_subtypes('FLAC')
{'PCM_24': 'Signed 24 bit PCM',
 'PCM_16': 'Signed 16 bit PCM',
 'PCM_S8': 'Signed 8 bit PCM'}
```

pysoundfile.**check_format**(*format*, *subtype=None*, *endian=None*)

Check if the combination of format/subtype/endian is valid.

Examples

```
>>> import pysoundfile as sf
>>> sf.check_format('WAV', 'PCM_24')
True
>>> sf.check_format('FLAC', 'VORBIS')
False
```

pysoundfile.**default_subtype**(*format*)

Return the default subtype for a given format.

Examples

```
>>> import pysoundfile as sf
>>> sf.default_subtype('WAV')
'PCM_16'
>>> sf.default_subtype('MAT5')
'DOUBLE'
```

class pysoundfile.**SoundFile**(*file*, *mode='r'*, *samplerate=None*, *channels=None*, *subtype=None*, *endian=None*, *format=None*, *closefd=True*)

Open a sound file.

If a file is opened with *mode* `'r'` (the default) or `'r+'`, no sample rate, channels or file format need to be given because the information is obtained from the file. An exception is the `'RAW'` data format, which always requires these data points.

File formats consist of three case-insensitive strings:

- a *major format* which is by default obtained from the extension of the file name (if known) and which can be forced with the format argument (e.g. `format='WAVEX'`).
- a *subtype*, e.g. `'PCM_24'`. Most major formats have a default subtype which is used if no subtype is specified.
- an *endian-ness*, which doesn't have to be specified at all in most cases.

A *SoundFile* object is a *context manager*, which means if used in a "with" statement, *close()* is automatically called when reaching the end of the code block inside the "with" statement.

Parameters

- **file** (*str or int or file-like object*) – The file to open. This can be a file name, a file descriptor or a Python file object (or a similar object with the methods `read()`/`readinto()`, `write()`, `seek()` and `tell()`).
- **mode** (*{'r', 'r+', 'w', 'w+', 'x', 'x+'}, optional*) – Open mode. Has to begin with one of these three characters: `'r'` for reading, `'w'` for writing (truncates *file*) or `'x'` for writing (raises an error if *file* already exists). Additionally, it may contain `'+'` to open *file* for both reading and writing. The character `'b'` for *binary mode* is implied because all sound files have to be opened in this mode. If *file* is a file descriptor or a file-like object, `'w'` doesn't truncate and `'x'` doesn't raise an error.
- **samplerate** (*int*) – The sample rate of the file. If *mode* contains `'r'`, this is obtained from the file (except for `'RAW'` files).
- **channels** (*int*) – The number of channels of the file. If *mode* contains `'r'`, this is obtained from the file (except for `'RAW'` files).
- **subtype** (*str, sometimes optional*) – The subtype of the sound file. If *mode* contains `'r'`, this is obtained from the file (except for `'RAW'` files), if not, the default value depends on the selected *format* (see *default_subtype()*). See *available_subtypes()* for all possible subtypes for a given *format*.
- **endian** (*{'FILE', 'LITTLE', 'BIG', 'CPU'}, sometimes optional*) – The endian-ness of the sound file. If *mode* contains `'r'`, this is obtained from the file (except for `'RAW'` files), if not, the default value is `'FILE'`, which is correct in most cases.
- **format** (*str, sometimes optional*) – The major format of the sound file. If *mode* contains `'r'`, this is obtained from the file (except for `'RAW'` files), if not, the default value is determined from the file extension. See *available_formats()* for all possible values.
- **closefd** (*bool, optional*) – Whether to close the file descriptor on *close()*. Only applicable if the *file* argument is a file descriptor.

Examples

```
>>> from pysoundfile import SoundFile
```

Open an existing file for reading:

```
>>> myfile = SoundFile('existing_file.wav')
>>> # do something with myfile
>>> myfile.close()
```

Create a new sound file for reading and writing using a with statement:

```
>>> with SoundFile('new_file.wav', 'x+', 44100, 2) as myfile:
>>>     # do something with myfile
>>>     # ...
>>>     assert not myfile.closed
>>>     # myfile.close() is called automatically at the end
>>> assert myfile.closed
```

name

The file name of the sound file.

mode
The open mode the sound file was opened with.

samplerate
The sample rate of the sound file.

channels
The number of channels in the sound file.

format
The major format of the sound file.

subtype
The subtype of data in the the sound file.

endian
The endian-ness of the data in the sound file.

format_info
A description of the major format of the sound file.

subtype_info
A description of the subtype of the sound file.

sections
The number of sections of the sound file.

closed
Whether the sound file is closed or not.

seekable()
Return True if the file supports seeking.

seek(*frames*, *whence=0*)
Set the read/write position.

Parameters

- **frames** (*int*) – The frame index or offset to seek.
- **whence** (*{SEEK_SET, SEEK_CUR, SEEK_END}, optional*) – By default (`whence=SEEK_SET`), *frames* are counted from the beginning of the file. `whence=SEEK_CUR` seeks from the current position (positive and negative values are allowed for *frames*). `whence=SEEK_END` seeks from the end (use negative value for *frames*).

Returns *int* – The new absolute read/write position in frames.

Examples

```
>>> from pysoundfile import SoundFile, SEEK_END
>>> myfile = SoundFile('stereo_file.wav')
```

Seek to the beginning of the file:

```
>>> myfile.seek(0)
0
```

Seek to the end of the file:

```
>>> myfile.seek(0, SEEK_END)
44100  # this is the file length
```

tell()
Return the current read/write position.

read(*frames=-1*, *dtype='float64'*, *always_2d=True*, *fill_value=None*, *out=None*)
Read from the file and return data as NumPy array.

Reads the given number of frames in the given data format starting at the current read/write position. This advances the read/write position by the same number of frames. By default, all frames from the current read/write position to the end of the file are returned. Use *seek()* to move the current read/write position.

Parameters

- **frames** (*int, optional*) – The number of frames to read. If `frames < 0`, the whole rest of the file is read.
- **dtype** (*{'float64', 'float32', 'int32', 'int16'}, optional*) – See *read()*.

Returns *numpy.ndarray or type(out)* – The read data; either a new array or *out* or a view into *out*. See *read()* for details.

Other Parameters **always_2d, fill_value, out** – See *read()*.

Examples

```
>>> from pysoundfile import SoundFile
>>> myfile = SoundFile('stereo_file.wav')
```

Reading 3 frames from a stereo file:

```
>>> myfile.read(3)
array([[ 0.71329652,  0.06294799],
       [-0.26450912, -0.38874483],
       [ 0.67398441, -0.11516333]])
>>> myfile.close()
```

write(*data*)
Write audio data to the file.

Writes a number of frames at the read/write position to the file. This also advances the read/write position by the same number of frames and enlarges the file if necessary.

Parameters **data** (*array_like*) – See *write()*.

blocks(*blocksize=None*, *overlap=0*, *frames=-1*, *dtype='float64'*, *always_2d=True*, *fill_value=None*, *out=None*)
Return a generator for block-wise reading.

By default, the generator yields blocks of the given *blocksize* (using a given *overlap*) until the end of the file is reached; *frames* can be used to stop earlier.

Parameters

- **blocksize** (*int*) – The number of frames to read per block. Either this or *out* must be given.
- **overlap** (*int, optional*) – The number of frames to rewind between each block.

- **frames** (*int, optional*) – The number of frames to read. If `frames < 1`, the file is read until the end.
- **dtype** (*{'float64', 'float32', 'int32', 'int16'}, optional*) – See *read()*.

Yields *numpy.ndarray or type(out)* – Blocks of audio data. See *blocks()* for details.

Other Parameters **always_2d, fill_value, out** – See *read()*.

flush()
Write unwritten data to the file system.

Data written with *write()* is not immediately written to the file system but buffered in memory to be written at a later time. Calling *flush()* makes sure that all changes are actually written to the file system.

This has no effect on files opened in read-only mode.

close()
Close the file. Can be called multiple times.

11 Index

- genindex

Python Module Index

p

Index