# audiolab Documentation

***Release 0.10.1***

**David Cournapeau**

December 16, 2008

# CONTENTS

CHAPTER

# ONE

# Introduction

For people doing audio processing, it is useful to be able to import data from audio files, and export them back, as well as listening to the results of some processing; matlab have functions such as wavread, wavwrite, soundsc, etc... for that purposes. The goal of audiolab is to give those capabilities to the scipy environment by wrapping the excellent library libsndfile from Erik Castro de Lopo. Audiolab supports all format supported by libsndfile, including wav, aiff, ircam files, flac (an open source lossless compressed format) and ogg vorbist (an open source compressed format - simlar to MP3 without the license issues); see here for a complete list.

**The main features of audiolab are:**

- reading all formats supported by sndfile directly into numpy arrays.
- writing all formats supported by sndfile directly from numpy arrays.
- A matlab-like API (ala wavread) for some file formats. Wav, aiff, flac, au, and ogg formats are supported.
- A play function to output data from numpy array into the sound device of your computer (Only ALSA for linux and CoreAudio for Mac OS X is implemented ATM).

**Note**: The library is still in flux: the API can still change before the 1.0 version. That baing said, the version 0.10 was a major change, and I don't expect any more major changes.

**Note**: The online version of this document is not always up to date. The pdf included in the package is the reference, and always in sync with the package. If something does not work, please refer first to the pdf included in the package.

CHAPTER

# TWO

# Download and installation

## 2.1 Supported platforms

Audiolab has been run succesfully on the following platforms:

- linux ubuntu (32 and 64 bits)
- windows XP (32 bits)
- Mac OS X (10.5)

I would be interested to hear anyone who succeesfully used it on other platforms.

## 2.2 Download

audiolab is part of scikits: its source can be downloaded directly from the scikits svn repository:

```
svn co http://svn.scipy.org/svn/scikits/trunk/audiolab
```

## 2.3 Requirements

audiolab requires the following softwares:

- a python interpreter.
- libsndfile
- numpy (any version >= 1.0 should work).
- setuptools

On Ubuntu, you can install the dependencies as follow:

```
sudo apt-get install python-dev python-numpy python-setuptools libsndfile-dev
```

## 2.4 Optional

Audiolab can optionally install audio backends. For now, only alsa is supported, for linux audio support. You need alsa headers for this to work; on Ubuntu, you can install them with the following command:

```
sudo apt-get install libasound2-dev
```

## 2.5 Installation

For unix users, if libsndfile is installed in standart location (eg /usr/lib, /usr/local/lib), the installer should be able to find them automatically, and you only need to do a "python setup.py install". In other cases, you need to create a file site.cfg to set the location of libsndfile and its header (there are site.cfg examples which should give you an idea how to use them on your platform).

## 2.6 License

audiolab is released under the LGPL, which forces you to release back the modifications you may make in the version of audiolab you are distributing, but you can still use it in closed softwares, as long as you don't use a modified version of it. The soundio module to output data onto sound devices is under the BSD, though.

CHAPTER

# THREE

# Overview

For simple usage, the matlab-like API is the simplest to use. For example, if you want to read a wav file, you can do it in one function call

```
from audiolab import wavread

data, fs, enc = wavread('test.wav')
```

This read the file test.wav, and returns the data, sampling rate as well as the encoding as a string. Similar function exists for writing, and for other formats: wav, aiff, au, ircam, ogg and flac formats are supported through this simple API.

## 3.1 Sndfile class

For more control (for example writing with a non default encoding, controling output array dtype), the Sndfile class should be used. Internally, the simple functions are just wrappers around this class. Let's see a simple example on how to use the Sndfile class for reading:

```
import numpy as np
from scikits.audiolab import Sndfile

f = Sndfile('test.wav', 'r')

# Sndfile instances can be queried for the audio file meta-data
fs = f.samplerate
nc = f.channels
enc = f.encoding

# Reading is straightfoward
data = f.read_frames(1000)

# This reads the next 1000 frames, e.g. from 1000 to 2000, but as single precision
data_float = f.read_frames(1000, dtype=np.float32)
```

As you can see the usage for reading is straightfoward. A Sndfile instance first created, and the instance is used for reading, as well as for quering meta-data about the file, like the sampling rate or the number of channels.

The read_frames method can optionally take a dtype argument like many numpy functions, to select the dtype of the output array. The exact semantics are more complicated than with numpy though, because of audio encoding specificities (see encoding section).

Writing audio file from data in numpy arrays is a bit more complicated, because you need to tell the Sndfile class about the file type, encoding and endianness, as well as the sampling rate and number of channels. For simplicity, the file format, encoding and endianness is controled from an helper class, Format:

```
import numpy as np
from scikits.audiolab import Format, Sndfile

filename = 'foo.wav'

# Create some data to save as audio data: one second of stereo white noise
data = np.random.randn(48000, 2)

# Create a Sndfile instance for writing wav files @ 48000 Hz
format = Format('wav')
f = Sndfile(filename, 'w', format, 2, 48000)

# Write the first 500 frames of the signal. Note that the write_frames method
# uses tmp's numpy dtype to determine how to write to the file; sndfile also
# converts the data on the fly if necessary
f.write_frames(data[:500])

f.close()
```

The Format class can be used to control more precisely the encoding or the endianness of the written file:

```
from scikits.audiolab import Format, Sndfile

# Use 24 bits encoding, big endian
format = Format('wav', 'pcm24', 'big')
f = Sndfile('foo.wav', 'w', format, 2, 48000)
```

Not all file formats and encodings combinations are possible. Also, the exact number of file formats and encodings available depend on your version of libsndfile. Both can be queried at runtime with the functions available_file_formats and available_encodings. The following example print all available file formats and encodings (the output depends on your installed libsndfile):

```
from scikits.audiolab import available_file_formats, available_encodings

for format in available_file_formats():
    print "File format %s is supported; available encodings are:" % format
    for enc in available_encodings(format):
        print "\t%s" % enc
    print ""
```

## 3.2 Sound output

audiolab now has some facilities to output sound from numpy arrays:

```
import numpy as np
from scikits.audiolab import play
```

```
# output one second of stereo gaussian white noise at 48000 hz
play(0.05 * np.random.randn(2, 48000), rate=48000)
```

CHAPTER

# FOUR

Full API

## 4.1 Audio file IO

### 4.1.1 The `Format` class

The Format class is used to control meta-data specific to one type of audio file (file format, encoding and endianness). It is mainly useful when writing files or reading raw (header-less) audio files. A Format instance can be queried for its related meta-data:

```
>>> from scikits.audiolab import Format
>>> a = Format() # By default, 16 bits PCM wav file
>>> print a # Will print a detail description of the format
```

**class `Format`**()

This class represents an audio file format. It knows about audio file format (wav, aiff, etc...), encoding (pcm, etc...) and endianness.

**Parameters** **type** : str

the major file format (wav, etc...).

**encoding** : str

the encoding (pcm16, etc..).

**endianness** : str

the endianess.

**See Also:**

`Sndfile`

#### Notes

The possible values for type, and encoding depend on your installed libsndfile. You can query the possible values with the functions available_file_formats and available_encodings.

**`file_format`**

Returns the file format.

**file_format_description**
Returns the full description of the file format.

**encoding**
Returns the encoding.

**encoding_description**
Returns the full description of the encoding.

**endianness**
Returns the endianness.

The following two functions can be used to query the available formats and encodings. The exact list of formats depend on the libsndfile audiolab was built against.

```
>>> from scikits.audiolab import available_encodings
>>> # List encodings supported for the wav format
>>> print available_encodings('wav')
```

**available_file_formats**()
available_file_formats() Return lists of available file formats supported by audiolab.

**available_encodings**()
available_encodings(major) Return lists of available encoding for the given major format.

### 4.1.2 The Sndfile class

Sndfile is the main class for audio file IO.

**class Sndfile**()
Sndfile(filename, mode=r, Format format=None, int channels=0, int samplerate=0) Sndfile is the core class to read/write audio files. Once an instance is created, it can be used to read and/or writes data from numpy arrays, query the audio file meta-data, etc...

**Parameters** **filename** : string or int
name of the file to open (string), or file descriptor (integer)
**mode** : string
'r' for read, 'w' for write, or 'rw' for read and write.
**format** : Format
Required when opening a new file for writing, or to read raw audio files (without header).
**channels** : int
number of channels.
**samplerate** : int
sampling rate.
**Returns** **sndfile: as Sndfile instance.** :

#### Notes

format, channels and samplerate need to be given only in the write modes and for raw files.

### Read methods

**read_frames**()
Sndfile.read_frames(self, scikits.audiolab.pysndfile.sndfile.sf_count_t nframes, dtype=<???>) Read the given number of frames and put the data into a numpy array of the requested dtype.

**Parameters** **nframes** : int
number of frames to read.
**dtype** : numpy dtype
dtype of the returned array containing read data (see note).

## Notes

if float are requested when the file contains integer data, you will get normalized data (that is the max possible integer will be 1.0, and the minimal possible value -1.0).

if integers are requested when the file contains floating point data, it may give wrong results because there is an ambiguity: if the floating data are normalized, you can get a file with only 0 ! Getting integer data from files encoded in normalized floating point is not supported (this is an audiolab limitation: sndfile supports it).

### Write methods

**write_frames**()
Sndfile.write_frames(self, ndarray input) write given number frames into file.

**Parameters** **input** : ndarray
array containing data to write.

## Notes

One column per channel.

updates the write pointer.

if the input type is float, and the file encoding is an integer type, you should make sure the input data are normalized normalized data (that is in the range [-1..1] - which will corresponds to the maximum range allowed by the integer bitwidth).

**sync**()
Sndfile.sync(self) call the operating system's function to force the writing of all file cache buffers to disk the file.

No effect if file is open as read

### Meta-data

**samplerate**
Returns the sampling rate of the file

**channels**
Returns the number of channels

**frames**
Returns the number of frames in the file

**format**
Returns the Format instance attached to the file.

### Encoding and array dtype

The most common encoding for common audio files like wav of aiff is signed 16 bits integers. Sndfile and hence audiolab enables many more encodings like unsigned 8 bits, floating point. Generally, when using the data for processing, the encoding of choice is floating point; the exact type is controled through the array dtype. When the array dtype and the file encoding don't match, there has to be some conversion.

When converting between integer PCM formats of differing size (ie both file encoding and input/output array dtype is an integer type), the Sndfile class obeys one simple rule:

- Whenever integer data is moved from one sized container to another sized container, the most significant bit in the source container will become the most significant bit in the destination container.

When either the encoding is an integer but the numpy array dtype is a float type, different rules apply. The default behaviour when reading floating point data (array dtype is float) from a file with integer data is normalisation. Regardless of whether data in the file is 8, 16, 24 or 32 bit wide, the data will be read as floating point data in the range [-1.0, 1.0]. Similarly, data in the range [-1.0, 1.0] will be written to an integer PCM file so that a data value of 1.0 will be the largest allowable integer for the given bit width.

## 4.2 Sound output

audiolab now has some facilities to output sound from numpy arrays: the function play is a wrapper around a platform-specific audio backend. For now, only ALSA backend (Linux) and Core Audio backend (Mac OS X) are implemented. Other backends (for windows, OSS for Solaris/BSD) may be added later, although it is not a priority for me. Patchs are welcomed, particularly for windows.

**play**(*input, fs=44100*)

Play the signal in vector input to the default output device.

Only floating point input are supported for now: input is assumed to be in the -1..1 range. Any values outside this range will be clipped by the device.

**Parameters** **input: array** :

input signal of rank 2. Each row is assumed to be one channel.

**fs: int** sampling rate (in Hz)

CHAPTER

# FIVE

# Obsolete API

This section documents the old, deprecated API.

> **NOTE** The old sndfile and formatinfo has been obsoleted in 0.10. Those classes were based on ctypes code, the new code is based on cython, and is more reliable, as well as more conformant to python conventions. In 0.10, the sndfile and formatinfo classes are thin wrappers around the Sndfile and Format classes, and you are advised to use those instead.

## 5.1 Overview

The following code shows you how to open a file for read, reading the first 1000 frames, and closing it:

```
import scikits.audiolab as  audiolab

a       = audiolab.sndfile('test.wav', 'read')
data    = a.read_frames(1000)
a.close()
```

## 5.2 Opening a file and getting its parameters

Once imported, audiolab gives you access the sndfile class, which is the class of audiolab use to open audio files. You create a sndfile instance when you want to open a file for reading or writing (the file test.flac is included in the audiolab package, in the test_data directory):

```
import scikits.audiolab as audiolab

filename    = 'test.wav'
a           = audiolab.sndfile(filename, 'read')

print a
```

Prints you the informations related to the file, like its sampling rate, the number of frames, etc... You can of course get each parameter individually by using the corresponding sndfile.get* accessors.

## 5.3 Importing audio data

Now that we've opened a file, we would like to read its audio content, right ? For now, you can only import the data as floating point data, float (32 bits) or double (64 bits). The function sndfile.read_frames read n frames, where a frame contains a sample of each channel (one in mono, 2 in stereo, etc...):

```
import numpy as N

import scikits.audiolab as audiolab

filename    = 'test.wav'
a           = audiolab.sndfile(filename, 'read')

tmp         = a.read_frames(1e4)
float_tmp   = a.read_frames(1e4, dtype = N.float32)

import pylab as P
P.plot(tmp[:])
```

The above code import 10000 frames, and plot the first channel using matplotlib (see below). A frame holds one sample from each channel: 1000 frames of a stereo file is 2000 samples. Each channel is one column of the numpy array. The read functions follow numpy conventions, that is by default, the data are read as double, but you can give a dtype argument to the function.

## 5.4 The format class

When opening a file for writing, you need to give various parameters related to the format such as the file format, the encoding. The format class is used to create valid formats from those parameters By default, the format class creates a format object with file type wav, and 16 bits pcm encoding:

```
from scikits.audiolab import formatinfo as format

f = format('aiff', 'ulaw')
print f

f = format('ircam', 'float32')
print f
```

prints back "Major Format: AIFF (Apple/SGI), Encoding Format: U-Law" and "Major Format: SF (Berkeley/IRCAM/CARL), Encoding Format: 32 bit float".

## 5.5 Writing data to a file

Opening a file for writing is a bit more complicated than reading; you need to say which format you are requesting, the number of channels and the sampling rate (in Hz) you are requesting; all thoses information are mandatory ! The class format is used to build a format understable by libsndfile from 'user-friendly' values. Let's see how it works.

```
from tempfile import mkstemp
from os import remove

import numpy as N
from scikits.audiolab import formatinfo as format
```

```
import scikits.audiolab as audiolab

# Create a temp file in the system temporary dir, and always remove
# it at the end
cd, filename    = mkstemp('tmptest.wav')
try:
    fmt         = format('wav', 'pcm24')
    nchannels   = 2
    fs          = 44100

    afile =  audiolab.sndfile(filename, 'write', fmt, nchannels, fs)

    # Create a stereo white noise, with Gaussian distribution
    tmp = 0.1 * N.random.randn(1000, nchannels)

    # Write the first 500 frames of the signal
    # Note that the write_frames method uses tmp's numpy dtype to determine how
    # to write to the file; sndfile also converts the data on the fly if necessary
    afile.write_frames(tmp, 500)

    afile.close()

    # Let's check that the written file has the expected meta data
    afile = audiolab.sndfile(filename, 'read')
    assert(afile.get_samplerate() == fs)
    assert(afile.get_channels() == nchannels)
    assert(afile.get_nframes() == 500)
finally:
    remove(filename)
```

CHAPTER

# SIX

# Future

## 6.1 Known bugs

- the function supported_* are broken (they never worked correctly). This will be fixed for audiolab 0.10
- there seems to be a problem when using libsndfile fseek facilities with flac files (which are necessary for the functions flacread/flacwrite). The problem seems to be with libFLAC; for this reason, seek in flac files is not enabled by default for now. See FLAC_SUPPORT.txt for more informations.

## 6.2 TODO

Before a 1.0 release, I would like to implement the follwings:

- support (at least some) meta-data embedded in some audio files format.
- support the libsndfile's error system.
- player on all major plateforms (windows is missing).