OsSom

Release 0.1.0a0

Jun 30, 2020

Contents

Audio IO with easy access to memory buffers for real-time analysis and processing of data.

Author: João Vitor Gutkoski Paes

CHAPTER 1

Installation

Simply *pip install* it:

```
~ $ pip install ossom
```

or

```
~ $ pip install git+https://github.com/chum4k3r/ossom.git
```

CHAPTER 2

Getting Started

First you should try:

```
>>> import ossom
>>> configs = ossom.Configurations()
>>> print(configs)
```

See *Examples* for more instructions

CHAPTER 3

API documentation

3.1 Audio data and buffer abstraction

These classes are designed to provide objects that will handle reading and writing of audio data as numpy arrays.

They are two: *Audio*, which is a read-only object, and *AudioBuffer*, that is a subclass of *Audio* and *multiprocessing.shared_memory.SharedMemory*, thus providing a cross-processes data read and write functionality.

Created on Fri May 29 16:07:04 2020.

@author: João Vitor Gutkoski Paes

class ossom.**Audio**(*data: numpy.ndarray*, *samplerate: int*, *blocksize: int = 512*)

Audio data interface.

__init__(*data: numpy.ndarray*, *samplerate: int*, *blocksize: int = 512*) → None

Audio objects are a representation of a waveform and a sample rate.

They hold the basic information to play sound.

Parameters

- **data** (*np.ndarray*) – An array containing audio data, or a buffer to be filled with audio.
- **samplerate** (*int*) – Audio sample rate, or how many data represent one second of data.
- **blocksize** (*int, optional*) – Amount of samples to read on each call to *next*. The default is config.blocksize.

Returns

Return type None

__getitem__(*key*)

Route Audio getitem to numpy.ndarray getitem.

__iter__()
Iterate method.

__next__() → numpy.ndarray
Return data from *ridx* to *ridx* + *blocksize*.

read_next(*blocksize: int*) → numpy.ndarray
Read data from *ridx* to *ridx* + *blocksize*.

Parameters **blocksize** (*int*) – The amount of data to read.

Raises StopIteration – Unavailable to read more data than *nsamples*.

Returns **data** – A numpy array with *blocksize* rows and *nchannels* columns.

Return type np.ndarray

ridx
Read data index.

blocksize
Amount of samples read on each call to *next()*.

samplerate
Sample rate of the audio.

data
Audio data as a numpy.ndarray.

nsamples
Total number of data.

nchannels
Total number of channels.

duration
Total time duration.

samplesize
Size of one sample of audio.

dtype
Type of the data.

bytesize
Size, in bytes, of whole array. Same as *samplesize * nsamples * nchannels*.

class ossom.**AudioBuffer**(*name: str*, *samplerate: int*, *buffersize: int*, *nchannels: int*, *blocksize: int*, *dtype: numpy.dtype = dtype('float32')*)
Audio data in a shared memory buffer.

__init__(*name: str*, *samplerate: int*, *buffersize: int*, *nchannels: int*, *blocksize: int*, *dtype: numpy.dtype = dtype('float32')*) → None
Buffer object intended to read and write audio samples.

Parameters

- **name** (*str*) – The SharedMemory name, can be None to automatically generate one.
- **samplerate** (*int*) – Audio sampling rate.
- **buffersize** (*int*) – Total number of samples.
- **nchannels** (*int*) – Total number of channels.
- **blocksize** (*int*) – Amount of samples to read on each call to *next*

- **dtype** (*np.dtype*) – Sample data type. The default is config.dtype.

Returns

Return type None.

__del__()
Guarantee that SharedMemory calls close and unlink.

widx
Write data index.

ready2read
How many samples are ready to read.

If the write index is smaller than read index returns None

Returns Amount of samples available to read.

Return type int

is_full
Check if ringbuffer is full or not.

clear()
Set all data to zero.

get_audio(*blocksize: int = None*, *copy: bool = False*) → ossom.audio.Audio
Audio object that points to shared memory buffer, or a copy of it.

Parameters

- **blocksize** (*int, optional*) – The read block size. The default is None.
- **copy** (*bool, optional*) – If set to *True*, the *Audio* is just a copy of the buffer. The default is False.

Returns The buffer as a read-only Audio object.

Return type *Audio*

write_next(*data: numpy.ndarray*) → int
Write data to buffer.

If *widx* gets equal to *nsamples* the buffer is set to full. Checking can be made using the *is_full* property.

Parameters **data** (*np.ndarray*) – Samples to write on buffer.

Returns Amount of written samples.

Return type int

3.2 Streaming audio data

Lacks documentation!

Created on Sun Jun 28 21:12:12 2020

@author: João Vitor Gutkoski Paes

class ossom.**Recorder**(*id: int = None, samplerate: int = 48000, blocksize: int = 512, channels: List[int] = [0, 1], buffersize: int = 480000, dtype: numpy.dtype = dtype('float32'), loopback: bool = False*)
Recorder class.

__init__(*id: int = None, samplerate: int = 48000, blocksize: int = 512, channels: List[int] = [0, 1], buffersize: int = 480000, dtype: numpy.dtype = dtype('float32'), loopback: bool = False*)
Record audio from input device directly into shared memory.

Parameters

- **id** (*int or str, optional*) – DESCRIPTION. The default is None.
- **samplerate** (*int, optional*) – DESCRIPTION. The default is config.samplerate.
- **blocksize** (*int, optional*) – DESCRIPTION. The default is config.blocksize.
- **channels** (*List[int], optional*) – DESCRIPTION. The default is config.channels.
- **buffersize** (*int, optional*) – DESCRIPTION. The default is config.buffersize.
- **dtype** (*_np.dtype, optional*) – DESCRIPTION. The default is config.dtype.
- **loopback** (*bool, optional*) – DESCRIPTION. The default is False.

Returns

Return type None.

channels
The device channels to record from. Zero indexed.

class ossom.**Player**(*id: int = None, samplerate: int = 48000, blocksize: int = 512, channels: List[int] = [0, 1], buffersize: int = 480000, dtype: numpy.dtype = dtype('float32')*)

channels
The device channels to output data to. Zero indexed.

3.3 Monitoring audio streams

Lacks documentation!

Created on Sat Jun 27 20:25:33 2020

@author: João Vitor Gutkoski Paes

class ossom.**Monitor**(*target: callable = <function Monitor.<lambda>>, samplerate: int = 48000, waittime: float = 1.0, args: tuple = (0,)*)
Monitor class.

__init__(*target: callable = <function Monitor.<lambda>>, samplerate: int = 48000, waittime: float = 1.0, args: tuple = (0,)*)
Control a multiprocessing.Process to visualize data from recording or playing.

Parameters

- **target** (*callable, optional*) – DESCRIPTION. The default is None.
- **args** (*tuple, optional*) – DESCRIPTION. The default is None.

Returns

Return type None.

__call__(*strm: Union[ossom.streamer.Recorder, ossom.streamer.Player] = None, blocksize: int = None*)
Configure the monitor and starts de process.

Parameters

- **strm** (*Union[Recorder, Player], optional*) – DESCRIPTION. The default is None.
- **buffersize** (*int, optional*) – DESCRIPTION. The default is None.

Returns

Return type None.

setup()
Any setup step needed to the end monitoring object. Must be overriden on subclasses.

tear_down()
Any destroying step needed to finish the end monitoring object. Must be overriden on subclasses.

start()
Start the parallel process.

wait()
Finish the parallel process.

3.4 Basic package property configurations

This module provide a singleton object to hold configurations such as *samplerate*, *blocksize* and any relevant parameter that is necessary to the main *Recorder* and *Player* classes.

These values are used as default case and are available to any OsSom class that might need them.

Created on Sun Jun 28 21:40:13 2020

@author: João Vitor Gutkoski Paes

class ossom.**Configurations**
Global OsSom configurations.

__init__()
This class is a singleton, which means that any "new" instance is actually the same, pointed to by a new name.

```
>>> new_config = ossom.Configurations()
>>> new_config is ossom.config
True
```

Basically it provides a one time setup interface to convinently create ‘Recorder‘s and ‘Player‘s with only default parameters.

```
>>> rec1 = Recorder(samplerate=44100)
>>> config.samplerate = 44100
>>> rec2 = Recorder()
>>> rec1.samplerate == rec2.samplerate
True
```

Returns

Return type None.

samplerate
Sample rate, or sampling frequency.

blocksize
Size of audio chunk read on calls to *next*.

buffersize
Total size of the audio buffer held by *Recorder* or *Player* objects.

dtype
The audio data type.

channels
Dictionary containing two lists that represents the device channels to be used. Zero indexed.

inChannels
List of requested device input channels. Zero indexed.

outChannels
List of requested device output channels. Zero indexed.

reset()
Set all configuration values back to module default.

3.5 Utilities

Utilities module.

Created on Tue May 5 00:31:42 2020

@author: João Vitor Gutkoski Paes

3.5.1 Mathematical utilities

Provide mathematical functions set up to process data from Audio objects.

Created on Tue May 5 00:34:36 2020

@author: João Vitor Gutkoski Paes

ossom.utils.maths.**apply_channelwise**
Apply func on each column of an audio data array.

Parameters

- **auddata** (*np.ndarray*) – Audio data as numpy array.
- **func** (*callable*) – The function to apply on each channel. Must receive data as a 1d array.

Returns **chwise** – Output of func for each channel as an array.

Return type np.ndarray

ossom.utils.maths.**max_abs**
Maximum of the absolute values of the input array, for each channel (column).

See *apply_channelwise* for more info

Parameters **auddata** (*np.ndarray*) – Input array of audio data.

Returns **ma** – Columnwise *max* of *abs* of *arr*.

Return type np.ndarray

ossom.utils.maths.**rms**
Root of the mean of the squared values of input array, for each channel.

See *apply_channelwise* for more info.

Parameters **auddata** (*np.ndarray*) – Input array of audio data.

Returns **rms** – Columnwise *sqrt* of *mean* of *arr**2*.

Return type np.ndarray

ossom.utils.maths.**dB**
Decibel level of input array, for each channel.

See *apply_channelwise* and *rms* for more info.

Parameters

- **auddata** (*np.ndarray*) – Input array of audio data.
- **power** (*bool, optional*) – If True, assumes a power signal, which need not to be squared. The default is False.
- **ref** (*float, optional*) – The decibel reference. The default is 1.0.

Returns Channelwise audio levels, in decibel.

Return type _np.ndarray

ossom.utils.maths.**noise**
Generate random noise for audio data.

Parameters

- **level** (*float*) – Level of audio.
- **samplerate** (*int*) – Sample rate.
- **tlen** (*float*) – Total time length.
- **nchannels** (*int*) – Total number of channels.

Returns The complete noise data.

Return type _np.ndarray

3.5.2 Colore Textos

Simple module to color texts on command line output.

@author: João Vitor Gutkoski Paes

class ossom.utils.**ColorStr**(*font: str = 'clear'*, *back: str = 'clear'*)
Color string

__init__(*font: str = 'clear'*, *back: str = 'clear'*) → None
Pintor de linhas.

Example use:

```
>>> black_on_white_str = ColorStr("black", "white")
>>> red_on_green_str = ColorStr("red", "green")
>>> print(black_on_white_str("Estou usando colore.ColorStr"))
>>> print(red_on_green_str("I'm using colore.ColorStr"))
```

Parameters

- **font** (*str, optional*) – Cor da fonte | Font color. Defaults to "clear".
- **back** (*str, optional*) – Cor do fundo | Background color. Defaults to "clear".

Returns None.

__call__(*text: str = None*) → str
Paint the text with its font and backgroud colors.

Parameters **text** (*str, optional*) – The text to be painted. Defaults to None.

Returns Colored text and background.

Return type str

fntclr
Font color.

font
Alias for fntclr.

bgrclr
Background color.

background
Alias for bgrclr.

back
Alias for bgrclr.

3.5.3 Frequency utilities

Provides frequency and fractional octave frequency bands functionalities.

@author: João Vitor Gutkoski Paes

ossom.utils.freq.**freq_to_band**(*freq: float*, *nthOct: int*, *ref: float*, *base: int*) → int
Band number from frequency value.

Parameters

- **freq** (*float*) – The frequency value.
- **nthOct** (*int*) – How many bands per octave.
- **ref** (*float*) – Frequency of reference, or band number 0.
- **base** (*int*) – Either 10 or 2.

Raises ValueError – If base is not 10 nor 2 raises value error.

Returns The band number from center.

Return type int

ossom.utils.freq.**fractional_octave_frequencies**(*nthOct: int = 3*, *freqRange: Tuple[float] = (20.0, 20000.0)*, *refFreq: float = 1000.0*, *base: int = 10*) → numpy.ndarray
Lower, center and upper frequency values of all bands within range.

Parameters

- **nthOct** (*int, optional*) – bands of octave/nthOct. The default is 3.
- **freqRange** (*Tuple[float], optional*) – frequency range. These frequencies are inside the lower and higher band, respectively. The default is (20., 20000.).
- **refFreq** (*float, optional*) – Center frequency of center band. The default is 1000..
- **base** (*int, optional*) – Either 10 or 2. The default is 10.

Returns **freqs** – Array with shape (N, 3).

Return type numpy.ndarray

ossom.utils.freq.**normalize_frequencies**(*freqs: numpy.ndarray*, *samplingRate: int = 44100*) → numpy.ndarray

Normalize frequencies for any sampling rate.

Parameters

- **freqs** (*np.ndarray*) – DESCRIPTION.
- **samplingRate** (*int, optional*) – DESCRIPTION. The default is 44100.

Returns DESCRIPTION.

Return type TYPE

ossom.utils.freq.**freqs_to_center_and_edges**(*freqs: numpy.ndarray*) → Tuple[numpy.ndarray]

Separate the array returned from *fractional_octave_frequencies*.

The returned arrays corresponde to the center and edge frequencies of the fractional octave bands

Parameters **freqs** (*np.ndarray*) – Array returned from *fractional_octave_frequencies*.

Returns

- **center** (*np.ndarray*) – Center frequencies of the bands.
- **edges** (*np.ndarray*) – Edge frequencies (lower and upper) of the bands.

3.5.4 File logging

This module provide two classes: *Now* and *Logger*.

The first is just a wrap around time.localtime() with human readable interface of the exact time instant the object is created. The latter is a simple file logger that uses *Now* instances to record information.

Created on Sat Jun 6 00:39:46 2020

@author: joaovitor

class ossom.utils.**Now**

Human readable data about date and time.

hour

Wall clock hours.

min

Wall clock minutes.

sec

Wall clock seconds.

year

Year.

month
Month.

day
Month day.

wday
Week day.

GMT
Time zone.

class ossom.utils.**Logger**(*name: str = 'common'*, *ext: str = 'log'*, *title: str = 'title'*, *logend: str = 'end.'*)
Simple logger.

__init__(*name: str = 'common'*, *ext: str = 'log'*, *title: str = 'title'*, *logend: str = 'end.'*) → None
File based logger.

Parameters

- **name** (*str, optional*) – File name. The default is 'common'.
- **ext** (*str, optional*) – File extension. The default is 'log'.
- **title** (*str, optional*) – A title or description for the log. The default is 'title'.
- **logend** (*str, optional*) – A tag that represents the end of a logging session. The default is 'end.'.

Returns

Return type None

__del__()
Close underlying file on del statement.

name
Log file name.

ext
File extension.

title
Log title.

time
Time at log creation.

header
File header.

logend
End of logging.

file
File pointer.

fopen()
Open the log file, enabling *read* and *write*.

start_log()
Write header to log file.

log(*message: str = None*) → int
Write *message* to the log file along with the current time.

Parameters **message** (*str, optional*) – The content of the logging. If no input is given, randomizes an item from *LogFile._randlog*. The default is None.

Returns **b** – The amount of bytes written to log file.

Return type int

end_log()
Write the end of logging tag.

fclose()
Close the log file.

print_log()
Print the log file content. File must be opened.

CHAPTER 4

Examples

The examples can be used directly from command line:

```
>>> python3 play_rec_log.py
```

Still under development.

4.1 Play random noise while recording and logging the RMS and dB level of audio

play_rec_log.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jun  1 19:27:43 2020.

@author: joaovitor
"""


import numpy as _np
import numba as _nb
from ossom import Recorder, Player, Audio, Monitor, Configurations
from ossom.utils import max_abs, rms, dB, Logger


config = Configurations()


class LogMonitor(Logger, Monitor):
    """File logger based monitor."""
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, name: str = 'example', ext: str = 'log',
                 samplerate: int = config.samplerate, waittime: float = 0.125,
                 title: str = 'Example logging.', logend: str = 'D End.') -> None:
        Logger.__init__(self, name, ext, title, logend)
        self.fclose()  # On windows it fails if the file is open on process start.
        Monitor.__init__(self, self.do_logging, samplerate, waittime, tuple())
        return

    def setup(self):
        """Open log file."""
        self.fopen()
        self.start_log()
        return

    def do_logging(self, data):
        """Process and log."""
        RMS = rms(data)
        db = dB(RMS)
        self.log(f'Data shape={data.shape}\tRMS={RMS}\tdB={db}')
        return

    def tear_down(self):
        """End the log and close file."""
        self.end_log()
        self.fclose()
        return


@_nb.njit
def _noise(gain: float, samplerate: int, tlen: float, nchannels: int) -> _np.ndarray:
    shape = (int(samplerate*tlen), nchannels)
    noise = _np.zeros(shape, dtype=_np.float32)
    noise[:] = _np.random.randn(*shape)
    noise[:, ] /= max_abs(noise)
    grms = 10**(gain/20)
    return grms*noise


def noise(gain: float = -6,
          samplerate: int = 48000,
          blocksize: int = 64,
          tlen: float = 5.0,
          nchannels: int = 2) -> Audio:
    """
    Create an AudioGenerator of random noise.

    Parameters
    ----------
    gain : float, optional
        DESCRIPTION. The default is 1/(2**0.5).
    samplerate : int, optional
        DESCRIPTION. The default is 44100.
    buffersize : int, optional
        DESCRIPTION. The default is 64.
    tlen : float, optional
        DESCRIPTION. The default is 5.0.
    nchannels : int, optional
```

(continues on next page)

(continued from previous page)

```
        DESCRIPTION. The default is 1.

    Returns
    -------
    AudioGenerator
        DESCRIPTION.

    """
    data = _noise(gain, samplerate, tlen, nchannels)
    return Audio(data, samplerate, blocksize)


def retrieve_device_ids():
    """Select audio IO devices."""
    default = config.device
    print(config.list_devices())
    print("\nEscolha os dispositivos separados por vírgula")
    try:
        return [int(dev.strip()) for dev in input("No formato IN, OUT: ").split(",")]
    except ValueError:
        return default


if __name__ == "__main__":
    # Generate an Audio object of a random white noise
    ng = noise()

    # Prepare a monitoring object that stores in a file logger.
    lgr = LogMonitor()

    # Select default devices.
    # config.device = retrieve_device_ids()

    # Create a recorder object to capture audio data.
    r = Recorder()

    # Create a player object to playback audio data.
    p = Player()

    # Tells logger which streamer object (player or recorder) to watch and how
    # many samples to read at each iteration.
    lgr(r, r.samplerate//8)

    # Start the monitor before the audio streamer
    lgr.start()
    # Plays the white noise
    p(ng)
    # While capturing audio
    r(ng.duration)

    # And asks logger to block until recorder has finished
    lgr.wait()

    # # Now set the logger to watch the player object
    lgr(p, p.samplerate//8)

    # # Retrieves a copy of the recorded audio.
```

(continues on next page)

(continued from previous page)

```
    a = r.get_record()

    # # Start the monitor before the audio streamer
    lgr.start()
    # # Playback the audio.
    p(a)

    # # Finishes logger services.
    lgr.wait()

    # Delete objects for memory cleanup. Explicit deleting is necessary.
    # del r, p, lgr
```

Python Module Index

o

Index

Symbols

A

B

C

D

E

F

G

H

I

L