TimeSide Documentation

Release 1.0

Guillaume Pellerin, Paul Brossier, Thomas Fillon, Antoine Grandry

Jul 21, 2022

CONTENTS

Contents:

CHAPTER

ONE

TIMESIDE : SCALABLE AUDIO PROCESSING FRAMEWORK AND SERVER WRITTEN IN PYTHON

TimeSide is a python framework enabling low and high level audio analysis, imaging, transcoding, streaming and labelling. Its high-level API is designed to enable complex processing on very large datasets of any audio or video assets with a plug-in architecture, a secure scalable backend and an extensible dynamic web frontend.

1.1 Use cases

- Scaled audio computing (filtering, machine learning, etc)
- Web audio visualization
- Audio process prototyping
- Realtime and on-demand transcoding and streaming over the web
- Automatic segmentation and labelling synchronized with audio events

1.2 Goals

- **Do** asynchronous and fast audio processing with Python,
- **Decode** audio frames from **any** audio or video media format into numpy arrays,
- **Analyze** audio content with some state-of-the-art audio feature extraction libraries like Aubio, Yaafe and VAMP as well as some pure python processors
- **Visualize** sounds with various fancy waveforms, spectrograms and other cool graphers,
- **Transcode** audio data in various media formats and stream them through web apps,
- **Serialize** feature analysis data through various portable formats,
- **Provide** audio sources from plateform like YouTube or Deezer
- **Deliver** analysis and transcode on provided or uploaded tracks over the web through a REST API
- **Playback** and **interact on demand** through a smart high-level HTML5 extensible player,
- **Index**, **tag** and **annotate** audio archives with semantic metadata (see Telemeta which embed TimeSide).
- **Deploy** and **scale** your own audio processing engine through any infrastructure

1.3 Funding and support

To fund the project and continue our fast development process, we need your explicit support. So if you use TimeSide in production or even in a development or experimental setup, please let us know by:

- staring or forking the project on GitHub
- tweeting something to @parisson_studio or @telemeta
- drop us an email on <support@parisson.com> or <pow@ircam.fr>

Thanks for your help!

CHAPTER

TWO

NEWS

2.1 1.0

- Server refactoring:
  - audio process run on items (REST API track's model)
  - several tools, views, models and serializers
  - REST API's schema on OpenAPI 3 specification and automatic Redoc generation
- Move core and server from Python 2.7 to 3.7
- Upgrade Django to 2.2, Django REST Framework to 3.11, Celery to 4.4
- Add an Aubio based decoder
- Add core and server processors' versioning and server process' run time
- Regroup all dependencies on pip requirements removing conda use
- Add *Provider package* as a core API component and as a REST API model
- Add provider plugins *DeezerPreview*, *DeezerComplete* and *YouTube*
- Improve server unit testing
- Add JWT authentication on REST API
- Various bug fixes
- Add core, server and workers logging

2.2 0.9

- Upgrade all python dependencies
- Add Vamp, Essentia, Yaafe, librosa, PyTorch, Tensorflow libs and wrappers
- Add a few analyzing plugins (Essentia Dissonance, Vamp Constant Q, Vamp Tempo, Vamp general wrapper, Yaafe general wrapper)
- Add processor parameter management
- Add processor inheritance
- Improve HTML5 player with clever data streaming
- Improve REST API and various serialzers

- Improve unit testing
- Various bug fixes

2.3 0.8

- Add *Docker* support for instant installation. This allows to run TimeSide now on *any* OS platform!
- Add Jupyter Notebook support for easy prototyping, experimenting and sharing (see the examples in the doc).
- Add an experimental web server and REST API based on Django REST Framework, Redis and Celery. This now provides a real web audio processing server with high scaling capabilities thanks to Docker (clustering) and Celery (multiprocessing).
- Start the development of a new player interface thanks to Angular and WavesJS.
- Huge cleanup of JS files. Please now use bower to get all JS dependencies as listed in settings.
- Add metadata export to Elan annotation files.
- Fix and improve some data structures in analyzer result containers.
- Many various bugfixes.

2.4 0.7.1

- fix django version to 1.6.10 (sync with Telemeta 1.5)

2.5 0.7

- Code refactoring:
  - Create a new module *timeside.plugins* and move processors therein: timeside.plugins.decoder,analyzer, timeside.plugins.encoder, timeside.plugins.fx
  - WARNING: to properly manage the namespace packages structure, the TimeSide main module is now *timeside.core* and code should now be initialized with *import timeside.core*
  - *timeside.plugins* is now a namespace package enabling external plugins to be **automatically** plugged into TimeSide (see for example timeside-diadems). This now makes TimeSide a **real** plugin host, yeah!
  - A dummy timeside plugin will soon be provided for easy development start.
- Move all analyzers developped by the partners of the Diadems project to a new repository: timeside-diadems
- Many fixes for a better processing by Travis-CI
- Add a dox file to test the docker building continously on various distributions

For older news, please visit: https://github.com/Parisson/TimeSide/blob/master/NEWS.rst

CHAPTER

THREE

ARCHITECTURE

The streaming architecture of TimeSide relies on 2 main parts: a processing engine including various plugin processors written in pure Python and a user interface providing some web based visualization and playback tools in pure HTML5.

CHAPTER

FOUR

DIVE IN

Let's produce a really simple audio analysis of an audio file. First, list all available plugins:

```
>>> import timeside.core
>>> timeside.core.list_processors()
IProcessor
==========
...
```

Define some processors:

```
>>> from timeside.core import get_processor
>>> from timeside.core.tools.test_samples import samples
>>> wavfile = samples['sweep.wav']
>>> decoder  =  get_processor('file_decoder')(wavfile)
>>> grapher  =  get_processor('waveform_simple')()
>>> analyzer =  get_processor('level')()
>>> encoder  =  get_processor('vorbis_encoder')('sweep.ogg')
```

Then run the *magic* pipeline:

```
>>> (decoder | grapher | analyzer | encoder).run()
```

Render the grapher results:

```
>>> grapher.render(output='waveform.png')
```

Show the analyzer results:

```
>>> print 'Level:', analyzer.results
Level: {'level.max': AnalyzerResult(...), 'level.rms': AnalyzerResult(...)}
```

So, in only one pass, the audio file has been decoded, analyzed, graphed and transcoded.

For more extensive examples, please see the full documentation.

CHAPTER

FIVE

INSTALL

Thanks to Docker, Timeside is now fully available as a docker composition ready to work. The docker based composition bundles some powerfull applications and modern frameworks out-of-the-box like: Python, Conda, Numpy, Jupyter, Gstreamer, Django, Celery, Haystack, ElasticSearch, MySQL, Redis, uWSGI, Nginx and many more.

First, install Docker and docker-compose

Then clone TimeSide:

```
git clone --recursive https://github.com/Parisson/TimeSide.git
cd TimeSide
docker-compose pull
```

That's it! Now please go to the documentation to see how to use it.

Note: If you need to user TimeSide outside a docker image please refer to the rules of the Dockerfile which is based on a Debian stable system. But we do not provide any kind of free support in this usercase anymore (the dependency list is now huge). To get commercial support in more various usecases, please reach the Parisson dev team.

CHAPTER

SIX

USER INTERFACES

6.1 Ipython

To run the ipython shell, just do it through the docker composition:

```
docker-compose run app ipython
```

6.2 Notebook

You can also run your code in the wonderful Jupyter Notebook which gives you a web interface to run your own code and share the results with your collaborators:

```
docker-compose -f docker-compose.yml -f env/notebook.yml up
```

and then browse http://localhost:8888 to access the Jupyter notebook interface. Use the token given in the docker logs of the *notebook* container to login.

Warning: Running a Jupyter notebook server with this setup in a non-secured network is not safe. See Running a notebook server for a documented solution to this security problem.

6.3 Use you own data

The *var/media* directory is mounted in */srv/media* inside the container so you can use it to exchange data between the host and the app container.

6.4 Web Server

TimeSide now includes an experimental web service with a REST API:

```
git clone https://github.com/Parisson/TimeSide.git
cd TimeSide
docker-compose up db
```

This will pull all needed images for running the server and then initialize the database. Leave the session with CTRL+C and then finally do:

```
docker-compose up
```

This will initialize everything and create a bunch a test sample boilerplate. You can browse the TimeSide API at:

http://localhost:8000/timeside/api/

and the admin interface (login: admin, password: admin) at:

http://localhost:8000/timeside/admin

Note: A documentation about using the objects and processors from the webserver will be written soon. We need help on this!

All (raw, still experimental) results are accessible at :

http://localhost:8000/timeside/

Tip: On MacOS or Windows, replace "localhost" by the virtual machine IP given by *docker-machine ip timeside*

To process some data by hand in the web environment context, just start a django shell session:

```
docker-compose run app manage.py shell
```

To run the webserver in background as a daemon, just add the *-d* option:

```
docker-compose up -d
```

6.5 Batch

A shell script is provided to enable preset based and recursive processing through your command line interface:

```
timeside-launch -h
Usage: bin/timeside-launch [options] -c file.conf file1.wav [file2.wav ...]
 help: bin/timeside-launch -h

Options:
 -h, --help            show this help message and exit
 -v, --verbose         be verbose
 -q, --quiet           be quiet
 -C <config_file>, --conf=<config_file>
                       configuration file
 -s <samplerate>, --samplerate=<samplerate>
                       samplerate at which to run the pipeline
 -c <channels>, --channels=<channels>
                       number of channels to run the pipeline with
 -b <blocksize>, --blocksize=<blocksize>
                       blocksize at which to run the pipeline
 -a <analyzers>, --analyzers=<analyzers>
                       analyzers in the pipeline
 -g <graphers>, --graphers=<graphers>
                       graphers in the pipeline
 -e <encoders>, --encoders=<encoders>
                       encoders in the pipeline
```

(continues on next page)

(continued from previous page)

```
-R <formats>, --results-formats=<formats>
                        list of results output formats for the analyzers
                        results
-I <formats>, --images-formats=<formats>
                        list of graph output formats for the analyzers results
-o <outputdir>, --ouput-directory=<outputdir>
                        output directory
```

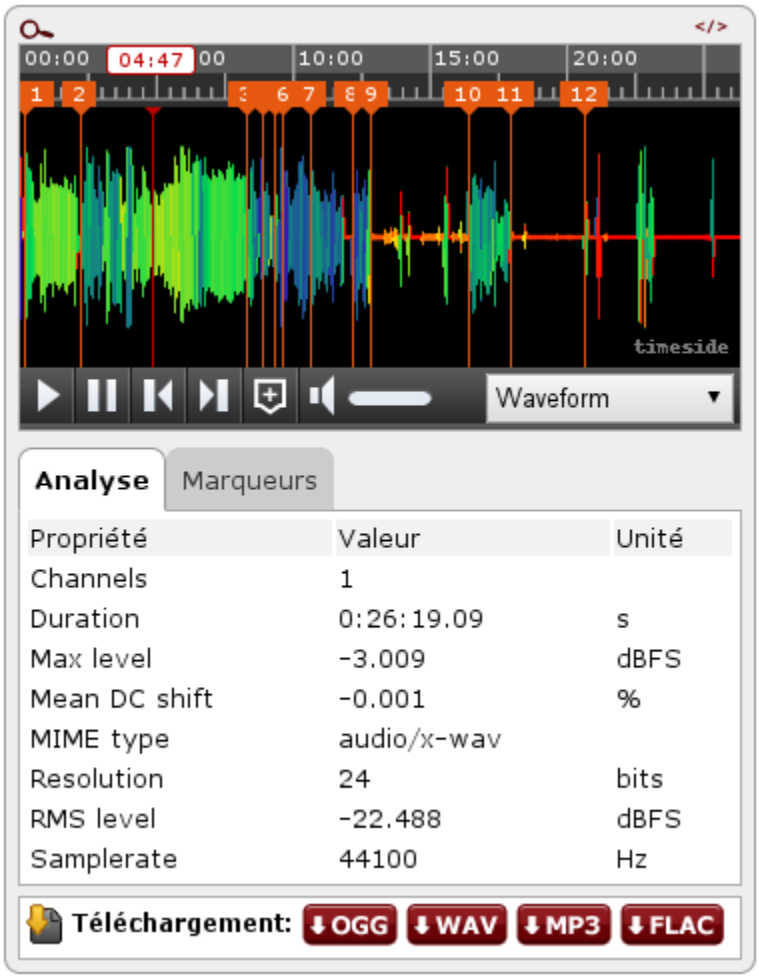
Find some preset examples in examples/presets/

6.6 Web player

TimeSide comes with a smart and pure **HTML5** audio player.

Features:

- embed it in any audio web application
- stream, playback and download various audio formats on the fly
- synchronize sound with text, bitmap and vectorial events
- seek through various semantic, analytic and time synced data
- fully skinnable with CSS style



Examples of the player embeded in the Telemeta open web audio CMS:

- http://parisson.telemeta.org/archives/items/PRS_07_01_03/
- http://archives.crem-cnrs.fr/items/CNRSMH_I_1956_002_001_01/

Development documentation:

- https://github.com/Parisson/TimeSide/wiki/Ui-Guide

CHAPTER

SEVEN

DOCUMENTATION

- General documentation: https://timeside.readthedocs.io/en/latest/index.html
- Tutorials: https://timeside.readthedocs.io/en/latest/tutorials/index.html
- RESTful API: https://sandbox.wasabi.telemeta.org/timeside/api/docs/
- Publications: https://github.com/Parisson/Telemeta-doc
- Slides: https://ircam-web.github.io/timeside-slides/#1
- Some (old) notebooks: http://mybinder.org/repo/thomasfillon/Timeside-demos
- Player UI wiki (v1): https://github.com/Parisson/TimeSide/wiki/Ui-Guide
- A player example (v1): http://archives.crem-cnrs.fr/archives/items/CNRSMH_E_2004_017_001_01/

CHAPTER

EIGHT

TIMESIDE : TUTORIALS

Contents:

8.1 Quick start

A most basic operation, transcoding, is easily performed with two processors:

```
>>> import timeside
>>> from timeside.core.tools.test_samples import samples
>>> from timeside.core import get_processor
>>> decoder = get_processor('file_decoder')(samples["sweep.wav"])
>>> encoder = get_processor('vorbis_encoder')("sweep.ogg")
>>> pipe = decoder | encoder
>>> pipe.run()
```

As one can see in the above example, creating a processing pipe is performed with the binary OR operator.

Audio data visualisation can be performed using graphers, such as Waveform and Spectrogram. All graphers return an image:

```
>>> import timeside
>>> from timeside.core.tools.test_samples import samples
>>> from timeside.core import get_processor
>>> decoder =  get_processor('file_decoder')(samples["sweep.wav"])
>>> spectrogram = get_processor('spectrogram_lin')(width=400, height=150)
>>> (decoder | spectrogram).run()
>>> spectrogram.render('graph.png')
```

It is possible to create longer pipes, as well as subpipes, here for both analysis and encoding:

```
>>> import timeside
>>> from timeside.core.tools.test_samples import samples
>>> from timeside.core import get_processor
>>> decoder = get_processor('file_decoder')(samples["sweep.wav"])
>>> levels = get_processor('level')()
>>> encoders = get_processor('mp3_encoder')('sweep.mp3') | get_processor('flac_encoder
↪')('sweep.flac')
>>> (decoder | levels | encoders).run()
>>> print levels.results
```

8.2 Data management

TimeSide offers various ways to access to audio data or metadata. *AnalyzerResult* is the python data structure where TimeSide embeds all the data resulting from a given analyzer processors after a run. It is thus the base object to access the analysis results and all the corresponding metadata. Bellow are some examples of use of the AnalyzerResult object and some of its methods.

Usage : AnalyzerResult(data_mode=None, time_mode=None)

Four different *time_mode* can be specified :

- 'framewise' : data are returned on a frame basis (i.e. with specified blocksize, stepsize and framerate)
- 'global' : a global data value is return for the entire audio item
- 'segment' : data are returned on a segment basis (i.e. with specified start time and duration)
- 'event' : data are returned on a instantaneous event basis (i.e. with specified start time)

Two different *data_mode* can be specified :

- 'value' : data are returned as numpy Array of arbitrary type
- 'label' : data are returned as label indexes (specified by the label_metadata key)

Default values are *time_mode* = *'framewise'* and *data_mode* = *'value'*

See : `timeside.core.analyzer.AnalyzerResult()`, `timeside.core.analyzer.AnalyzerResult`

8.2.1 Default

Create a new analyzer result without default arguments

```
>>> from timeside.core.analyzer import AnalyzerResult
>>> res = AnalyzerResult()
```

```
>>> res.keys()
['id_metadata', 'data_object', 'audio_metadata', 'parameters']
```

```
>>> for key,value in res.items():
...     print '%s : %s' % (key, value)
...
id_metadata : {'description': '', 'author': '', 'version': '', 'date': '', 'proc_uuid
↪': '', 'id': '', 'unit': '', 'name': ''}
data_object : {'y_value': array([], dtype=float64), 'value': array([], dtype=float64),
↪ 'frame_metadata': {'blocksize': None, 'samplerate': None, 'stepsize': None}}
audio_metadata : {'sha1': '', 'is_segment': None, 'uri': '', 'channels': None, 'start
↪': 0, 'channelsManagement': '', 'duration': None}
parameters : {}
```

8.2.2 Specification of time_mode

Four different time_mode can be specified :

- 'framewise' : data are returned on a frame basis (i.e. with specified blocksize, stepsize and framerate)
- 'global' : a global data value is return for the entire audio item
- 'segment' : data are returned on a segment basis (i.e. with specified start time and duration)
- 'event' : data are returned on a segment basis (i.e. with specified start time)

Framewise

```
>>> res = AnalyzerResult(time_mode='framewise')
>>> res.data_object.keys()
['value', 'y_value', 'frame_metadata']
```

Global

No frame metadata information is needed for these modes. The 'frame_metadata' key/attribute is deleted.

```
>>> res = AnalyzerResult(time_mode='global')
>>> res.data_object.keys()
['value', 'y_value']
```

```
>>> res.data_object
GlobalValueObject(value=array([], dtype=float64), y_value=array([], dtype=float64))
```

Segment

```
>>> res = AnalyzerResult(time_mode='segment')
>>> res.keys()
['id_metadata', 'data_object', 'audio_metadata', 'parameters']
>>> res.data_object
SegmentValueObject(value=array([], dtype=float64), y_value=array([], dtype=float64),␣
↪time=array([], dtype=float64), duration=array([], dtype=float64))
```

Event

```
>>> res = AnalyzerResult(time_mode='event')
>>> res.keys()
['id_metadata', 'data_object', 'audio_metadata', 'parameters']
>>> res.data_object
EventValueObject(value=array([], dtype=float64), y_value=array([], dtype=float64),␣
↪time=array([], dtype=float64))
```

8.2.3 Specification of data_mode

Two different data_mode can be specified :

- 'value' : data are returned as numpy Array of arbitrary type
- 'label' : data are returned as label indexes (specified by the label_metadata key)

Value

```
>>> res = AnalyzerResult(data_mode='value')
>>> res.data_object.keys()
['value', 'y_value', 'frame_metadata']
```

In the dataObject key, the 'value' key is kept and the 'label' key is deleted.

```
>>> res.data_object
FrameValueObject(value=array([], dtype=float64), y_value=array([], dtype=float64),␣
↪frame_metadata=FrameMetadata(samplerate=None, blocksize=None, stepsize=None))
```

Label

A *label_metadata* key is added.

```
>>> res = AnalyzerResult(data_mode='label')
>>> res.data_object.keys()
['label', 'label_metadata', 'frame_metadata']
```

```
>>> res.data_object
FrameLabelObject(label=array([], dtype=int64), label_
↪metadata=LabelMetadata(label=None, description=None, label_type='mono'), frame_
↪metadata=FrameMetadata(samplerate=None, blocksize=None, stepsize=None))
```

8.3 Using the 'stack' (previously decoded frames)

This is an example of using the *stack* argument in `timeside.plugins.decoder.file.FileDecoder` to run a pipe with previously decoded frames stacked in memory on a second pass.

First, let's import everything and define the audio file source :

```
>>> import timeside.core
>>> from timeside.core import get_processor
>>> from timeside.core.tools.test_samples import samples
>>> import numpy as np
>>> audio_file = samples['sweep.mp3']
```

Then let's setup a `FileDecoder` with argument *stack=True* (default argument is *stack=False*) :

```
>>> decoder = timeside.plugins.decoder.file.FileDecoder(audio_file, stack=True)
```

Setup an arbitrary analyzer to check that decoding process from file and from stack are equivalent:

```
>>> level = get_processor('level')()
>>> pipe = (decoder | level)
>>> print pipe.processors
[file_decoder-{}, level-{}]
```

Run the pipe:

```
>>> pipe.run()
```

The processed frames are stored in the pipe attribute *frames_stack* as a list of frames :

```
>>> print type(pipe.frames_stack)
<type 'list'>
```

First frame :

```
>>> print pipe.frames_stack[0]
(array([[...]], dtype=float32), False)
```

Last frame :

```
>>> print pipe.frames_stack[-1]
(array([[...]], dtype=float32), True)
```

If the pipe is used for a second run, the processed frames stored in the stack are passed to the other processors without decoding the audio source again.

8.4 Streaming out encoded audio

Instead of calling a pipe.run(), the chunks of an encoding processor can also be retrieved and streamed outside the pipe during the process.

```
>>> import timeside
>>> from timeside.core import get_processor
>>> from timeside.core.tools.test_samples import samples
>>> import numpy as np
>>> audio_file = samples['sweep.wav']
>>> decoder = get_processor('file_decoder')(audio_file, duration=1)
>>> output = '/tmp/test.mp3'
>>> encoder = get_processor('mp3_encoder')(output, streaming=True, overwrite=True)
>>> pipe = decoder | encoder
```

Create a process callback method so that you can retrieve end send the chunks:

```
>>> def streaming_callback():
...     for chunk in pipe.stream():
...         # Do something with chunk
...         print chunk.timestamp
```

Now you can use the callback to stream the audio data outside TimeSide!

```
>>> streaming_callback()
```

CHAPTER

NINE

TIMESIDE CORE API

9.1 List of available processors

9.1.1 Encoder

- **flac_aubio_encoder 1.0**: FLAC encoder based on aubio
- **vorbis_aubio_encoder 1.0**: OGG Vorbis encoder based on aubio
- **wav_aubio_encoder 1.0**: Wav encoder based on aubio
- **live_encoder 1.0**: Gstreamer-based Audio Sink
- **flac_encoder 1.0**: FLAC encoder based on Gstreamer
- **aac_encoder 1.0**: AAC encoder based on Gstreamer
- **mp3_encoder 1.0**: MP3 encoder based on Gstreamer
- **vorbis_encoder 1.0**: OGG Vorbis encoder based on Gstreamer
- **opus_encoder 1.0**: Opus encoder based on Gstreamer
- **wav_encoder 1.0**: WAV encoder based on Gstreamer
- **webm_encoder 1.0**: WebM encoder based on Gstreamer

9.1.2 Decoder

- **array_decoder 1.0**: Decoder taking Numpy array as input
- **aubio_decoder 1.0**: File decoder based on aubio
- **file_decoder 1.0**: File Decoder based on Gstreamer

9.1.3 Grapher

- **grapher_aubio_pitch 1.0**: Image representing Pitch
- **grapher_aubio_silence 1.0**: Image representing Aubio Silence
- **grapher_dissonance 1.0**: Image representing Dissonance
- **grapher_vamp_cqt 1.0**: Image representing Constant Q Transform
- **grapher_loudness_itu 1.0**: Image representing Loudness ITU
- **spectrogram 1.0**: Image representing Linear Spectrogram

- **grapher_onset_detection_function 1.0**: Image representing Onset detection
- **grapher_waveform 1.0**: Image representing Waveform from Analyzer
- **spectrogram_log 1.0**: Logarithmic scaled spectrogram (level vs. frequency vs. time).
- **spectrogram_lin 1.0**: Linear scaled spectrogram (level vs. frequency vs. time).
- **waveform_simple 1.0**: Simple monochrome waveform image.
- **waveform_centroid 1.0**: Waveform where peaks are colored relatively to the spectral centroids of each frame buffer.
- **waveform_contour_black 1.0**: Black amplitude contour waveform.
- **waveform_contour_white 1.0**: an white amplitude contour wavform.
- **waveform_transparent 1.0**: Transparent waveform.

9.1.4 Analyzer

- **aubio_melenergy 0.4.6**: Aubio Mel Energy analyzer
- **aubio_mfcc 0.4.6**: Aubio MFCC analyzer
- **aubio_pitch 0.4.6**: Aubio Pitch estimation analyzer
- **aubio_silence 0.4.6**: Aubio Silence detection analyzer
- **aubio_specdesc 0.4.6**: Aubio Spectral Descriptors collection analyzer
- **aubio_temporal 0.4.6**: Aubio Temporal analyzer
- **essentia_dissonance 2.1b5.dev416**: Dissonance from Essentia
- **vamp_constantq 1.1.0**: Constant Q transform from QMUL vamp plugins
- **vamp_simple_host 1.1.0**: Vamp plugins library interface analyzer
- **loudness_itu 1.0**: Measure of audio loudness using standard ITU-R BS.1770-3
- **spectrogram_analyzer 1.0**: Spectrogram image builder with an extensible buffer based on tables
- **onset_detection_function 1.0**: Onset Detection Function analyzer
- **spectrogram_analyzer_buffer 1.0**: Spectrogram image builder with an extensible buffer based on tables
- **waveform_analyzer 1.0**: Waveform analyzer

9.1.5 ValueAnalyzer

- **mean_dc_shift 1.0**: Mean DC shift analyzer
- **essentia_dissonance_value 2.1b5.dev416**: Mean Dissonance Value from Essentia
- **vamp_tempo 1.1.0**: Tempo from QMUL vamp plugins
- **vamp_tuning 1.1.0**: Tuning from NNLS Chroma vamp plugins
- **level 1.0**: Audio level analyzer

9.1.6 Effect

- **fx_gain 1.0**: Gain effect processor

9.2 Decoder package

9.2.1 File Decoder

9.2.2 Array Decoder

9.2.3 Live Decoder

9.3 Analyzer package

9.3.1 Core

AnalyzerResult

AnalyzerResultContainer

9.3.2 Analyzers

Timeside Core Analyzers

Global analyzers

Mean DC Shift

Level

Value Analyzers

Spectrogram

Analyzer from External librairies

Aubio

aubio is a tool designed for the extraction of annotations from audio signals. Its features include segmenting a sound file before each of its attacks, performing pitch detection, tapping the beat and producing midi streams from live audio. See http://aubio.org/

Aubio Melenergy

Aubio MFCC

Aubio Pitch

Aubio Spectral Descriptors collection

Aubio Temporal

Yaafe

9.3.3 Preprocessors

downmix_to_mono

frames_adapter

9.4 Encoder package

9.4.1 Core module

9.4.2 Encoders

Flac encoder

Aac encoder

Mp3 encoder

Vorbis encoder

Wav encoder

WebM encoder

AudioSink encoder

9.5 Grapher package

9.5.1 Core module

9.5.2 Graphers

Waveform

WaveformCentroid

WaveformTransparent

WaveformContour

SpectrogramLog

SpectrogramLin

9.6 Provider package

9.6.1 Core module

9.6.2 Providers

YouTube

DeezerPreview

DeezerComplete

CHAPTER

TEN

DEVELOPMENT

10.1 Developing within TimeSide

If the TimeSide library gives you everything you need to develop you own plugin, it is advised to start with one existing. For example, starting from the DC analyzer:

```
git clone https://github.com/Parisson/TimeSide.git
cd TimeSide
git checkout dev
cp timeside/plugins/analyzer/dc.py timeside/plugins/analyzer/my_analyzer.py
```

Before coding, start docker with mounting the local directory as a volume:

```
docker run -it -v .:/srv/lib/timeside parisson/timeside:latest ipython
```

or use the development composition to start a notebook or the webserver:

```
docker-compose -f docker-compose.yml -f conf/dev.yml up
```

10.2 Developing your own external plugins

If the (already huge) python module bundle provided by TimeSide is to short for you, it is possible to make your own plugin bundle outside the core module thanks to the TimeSide namespace. An extensive example of what you can do is available in the DIADEMS project repository. You can also start with the dummy plugin:

```
git clone https://github.com/Parisson/TimeSide-Dummy.git
cd TimeSide-Dummy
docker run -it -v ./timeside/plugins/:/srv/lib/timeside/timeside/plugins parisson/
↪timeside:latest ipython
```

or:

```
docker-compose -f docker-compose.yml -f conf/dummy.yml up
```

CHAPTER

ELEVEN

PRODUCTION

11.1 Deploying

and bleeding edge frameworks like: Nginx, PostgreSQL, Redis, Celery, Django, Django REST Framework and Python. It thus provides a safe and continuous way to deploy your project from an early development stage to a massive production environment. Our docker composition already bundles some powerful containers

> **Warning:** Before any serious production usecase, you *must* modify all the passwords and secret keys in the configuration files of the sandbox.

Thanks to Celery, each TimeSide worker of the server will process each task asynchronously over independant threads so that you can load all the cores of your CPU.

11.2 Scaling

To scale it up through your cluster, Docker finally provides some nice tools for orchestrating it very easily: Machine and Swarm.

CHAPTER

TWELVE

SPONSORS AND PARTNERS

- IRCAM (Paris, France)
- Parisson (Paris, France)
- CNRS: National Center of Science Research (France)
- Huma-Num: big data equipment for digital humanities (CNRS, France)
- CREM: French National Center of Ethomusicology Research (France)
- Université Pierre et Marie Curie (UPMC Paris, France)
- ANR: Agence Nationale de la Recherche (France)
- MNHN : Museum National d'Histoire Naturelle (Paris, France)
- C4DM : Center for Digital Music, Queen Mary University (London, United Kingdom)
- NYU Steinhardt : Music and Performing Arts Professions, New York University (New York, USA)

CHAPTER

THIRTEEN

RELATED PROJECTS

- Telemeta : Open web audio platform
- Sound archives of the CNRS, CREM and the “Musée de l’Homme” in Paris, France
- DIADEMS sponsored by the ANR.
- DaCaRyh, Data science for the study of calypso-rhythm through history
- KAMoulox Online unmixing of large historical archives
- NYU+CREM+Parisson : arabic music analysis from the full CREM database
- WASABI: Web Audio Semantic Aggregated in the Browser for Indexation, sponsored by the ANR

CHAPTER

FOURTEEN

COPYRIGHTS

CHAPTER

FIFTEEN

LICENSE

TimeSide is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

TimeSide is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

Read the LICENSE.txt file for more details.

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

T