TimeSide : Web Audio Components

Guillaume Pellerin

December 8, 2011 - v0.1

Why TimeSide ?

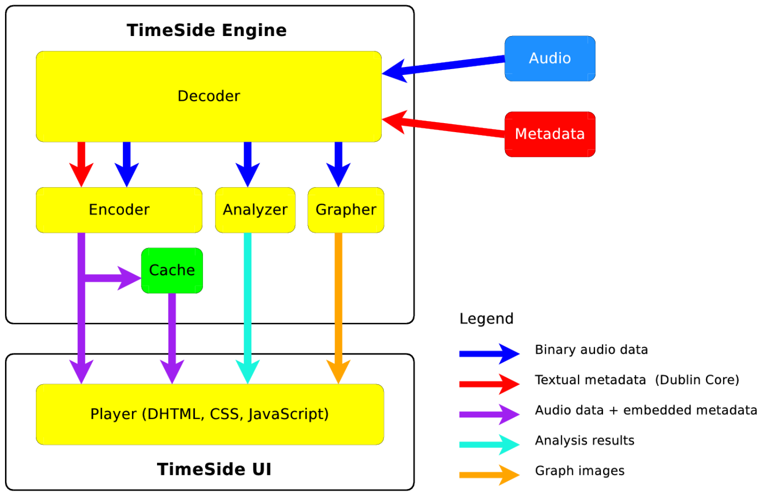
We do need an audio framework for :

- on the fly audio format transcoding
- metadata management and embedding
- time, frequency and typical analysis
- sound visualization : waveforms, spectrogram, etc..

TimeSide is a set of client and server side components for audio-enabling web sites and applications. It includes a powerful DHTML-based interactive player, with support for time-marking.

The server side components provide generic APIs for easy transcoding, metadata embedding, sound visualization and audio analysis.

Architecture diagram



Processor API 1/2 (see also API)

```
class IProcessor(Interface):
 """Common processor interface"""

 @staticmethod
 def id():
     """Short alphanumeric, lower-case string which uniquely identify this
     processor, suitable for use as an HTTP/GET argument value, in filenames,
     etc..."""

     # implementation: only letters and digits are allowed. An exception will
     # be raised by MetaProcessor if the id is malformed or not unique amongst
     # registered processors.

 def setup(self, channels=None, samplerate=None, nframes=None):
     """Allocate internal resources and reset state, so that this processor is
     ready for a new run.

     The channels, samplerate and/or nframes arguments may be required by
     processors which accept input. An error will occur if any of
     these arguments is passed to an output-only processor such as a decoder.
     """

     # implementations should always call the parent method

 def channels(self):
     """Number of channels in the data returned by process(). May be different from
     the number of channels passed to setup()"""
```

Processor API 2/2 (see also API)

```
def samplerate(self):
    """Samplerate of the data returned by process(). May be different from
    the samplerate passed to setup()"""

def nframes():
    """The total number of frames that this processor can output, or None if
    the duration is unknown."""

def process(self, frames=None, eod=False):
    """Process input frames and return a (output_frames, eod) tuple.
    Both input and output frames are 2D numpy arrays, where columns are
    channels, and containing an undetermined number of frames.  eod=True
    means that the end-of-data has been reached.

    Output-only processors (such as decoders) will raise an exception if the
    frames argument is not None. All processors (even encoders) return data,
    even if that means returning the input unchanged.

    Warning: it is required to call setup() before this method."""

def release(self):
    """Release resources owned by this processor. The processor cannot
    be used anymore after calling this method."""

    # implementations should always call the parent method
```

Usage : piping (see other examples here)

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

```
from timeside import Decoder, OggEncoder

decoder = Decoder('myfile.wav')
encoder = OggEncoder("myfile.ogg")
pipe    = decoder | encoder
pipe.run()
```

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

```
from timeside import Decoder, Spectrogram

decoder     = Decoder('myfile.wav')
spectrogram = Spectrogram(width=400, height=150)

(decoder | spectrogram).run()

spectrogram.render().save('graph.png')
```

Usage : piping (see other examples here)

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

```
from timeside import Decoder, Waveform, MaxLevel, MeanLevel, Mp3Encoder, FlacEncoder

decoder  = Decoder('myfile.wav')
max      = MaxLevel()
mean     = MeanLevel()
encoders = Mp3Encoder('myfile.mp3') | FlacEncoder('myfile.flac')

(decoder | max | mean | encoders).run()
print "Max level: %s, Mean level: %s" % (max, mean)
```

A common operation : normalization

```
from timeside import Decoder, MaxLevel, Gain, WavEncoder

decoder  = Decoder('source.wav')
max      = MaxLevel()

(decoder | max).run()

gain = max.result() > 0 and (1 / max.result()) or 1

(decoder | Gain(gain) | WavEncoder('normalized.wav')).run()
```

Grapher API (see also API)

```
class IGrapher(IProcessor):
  """Media item visualizer driver interface"""

  # implementation: graphers which need to know the total number of frames
  # should raise an exception in setup() if the nframes argument is None

  def __init__(self, width, height):
      """Create a new grapher. width and height are generally
      in pixels but could be something else for eg. svg rendering, etc.. """

      # implementation: additional optionnal arguments are allowed

  @staticmethod
  def name():
      """Return the graph name, such as "Waveform", "Spectral view",
      etc..  """

  def set_colors(self, background=None, scheme=None):
      """Set the colors used for image generation. background is a RGB tuple,
      and scheme a a predefined color theme name"""

  def render(self):
      """Return a PIL Image object visually representing all of the data passed
      by repeatedly calling process()"""
```

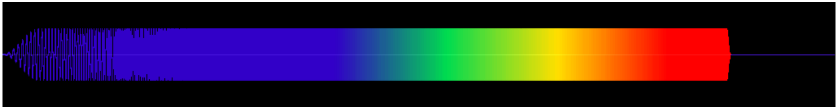
Grapher usage example (see other examples here)

```
from timeside.tests.api import examples
from timeside.core import *
from timeside.api import *
from timeside.tests.api.gstreamer import FileDecoder
import os

sample_dir = '../samples'
img_dir = '../results/img'
if not os.path.exists(img_dir):
    os.mkdir(img_dir)

test_dict = {'sweep.wav': 'waveform_wav.png',
            'sweep.flac': 'waveform_flac.png',
            'sweep.ogg': 'waveform_ogg.png',
            'sweep.mp3': 'waveform_mp3.png',
            }

for source, image in test_dict.iteritems():
    audio = os.path.join(os.path.dirname(__file__), sample_dir + os.sep + source)
    image = img_dir + os.sep + image
    print 'Test : decoder(%s) | waveform (%s)' % (source, image)
    decoder  = FileDecoder(audio)
    waveform = examples.Waveform(width=1024, height=256, output=image, bg_color=(0,0,0), color_scheme='default'
    (decoder | waveform).run()
    print 'frames per pixel = ', waveform.graph.samples_per_pixel
    print "render waveform to: %s" %  image
    waveform.render()
```

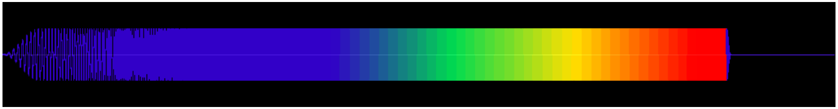
Waveform principle with Spectral Centroid

Sweep test waveforms - [20 ; 22050] Hz - 0:07

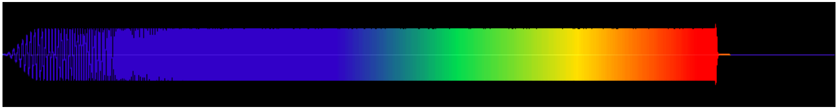
- sweep.wav



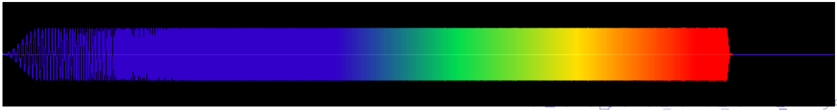
- sweep.flac



- sweep.mp3

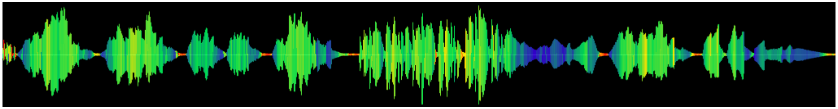


- sweep.ogg

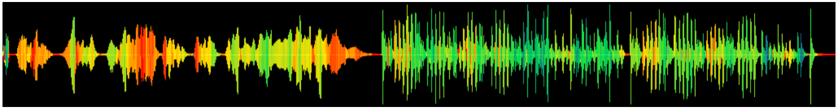


Solo music waveforms

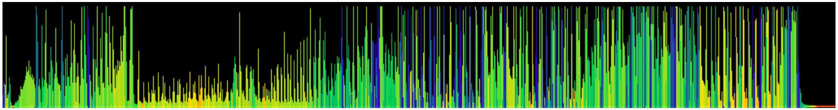
- Saxophone - 1:14



- Violon - 1:19

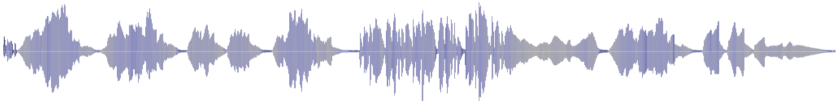


- Drums - 2:20

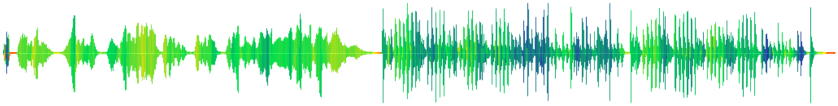


Solo music waveforms - other color schemes

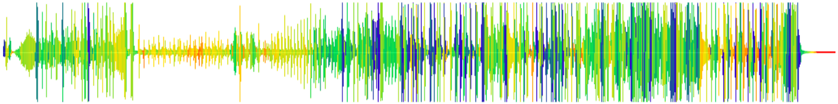
- Saxophone - 1:14 - purple



- Violon - 1:19 - default

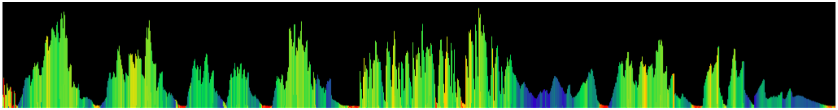


- Drums - 2:20 - ISO

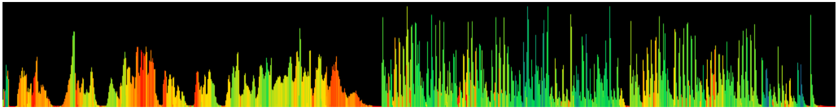


Why display redundant information ?

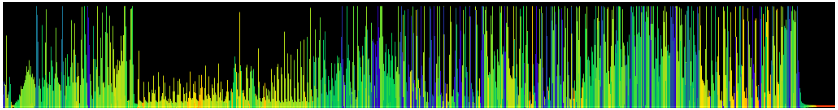
- Saxophone - 1:14



- Violon - 1:19

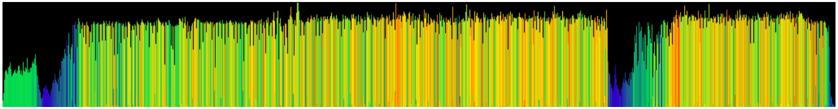


- Drums - 2:20

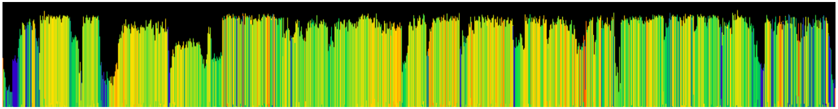


Mixed music waveforms (white bg)

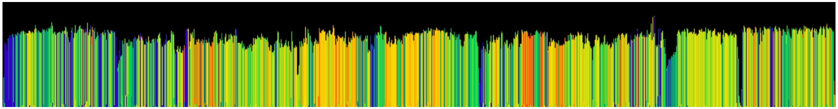
- Pop dance song - 4:40



- Electronica Mix - 1:00:34

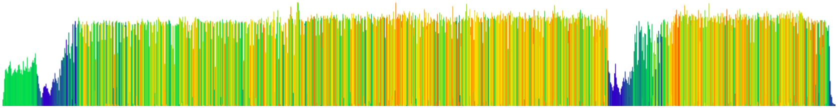


- Techno Mix - 1:12:40

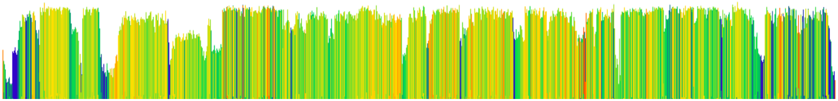


Mixed music waveforms (black bg)

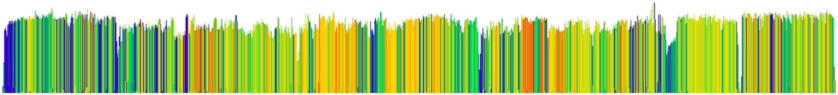
- Pop dance song - 4:40



- Electronica Mix - 1:00:34



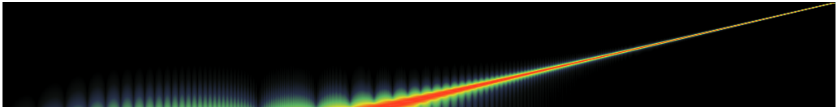
- Techno Mix - 1:12:40



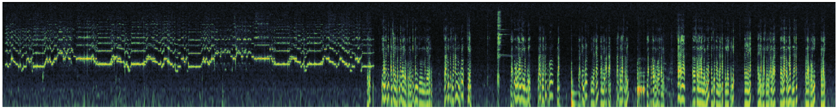
Spectrogram principle

Various spectrograms (black bg)

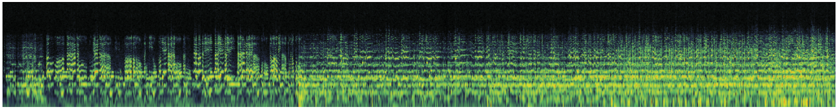
- Sweep test - 0:07 - telemeta link



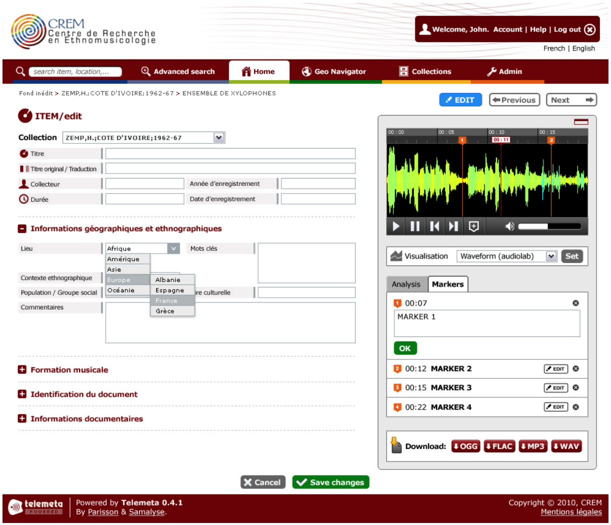
- Flute and Gong - 1:53 - telemeta link



- Voice and Gamelan - 3:50 - telemeta link



Telemeta 1.0 - edit page



CREM Centre de Recherche en Ethnomusicologie

Welcome, John. Account | Help | Log out

French | English

Fond inédit > ZEMP,H.;COTE D'IVOIRE;1962-67 > ENSEMBLE DE XYLOPHONES

ITEM/edit

Collection: ZEMP,H.;COTE D'IVOIRE;1962-67

Titre

Titre original / Traduction

Collecteur — Année d'enregistrement

Durée — Date d'enregistrement

Informations géographiques et ethnographiques

Lieu: Afrique (Afrique, Amérique, Asie, Europe, Océanie; Albanie, Espagne, France, Grèce) — Mots clés

Contexte ethnographique

Population / Groupe social — re culturelle

Commentaires

Formation musicale

Identification du document

Informations documentaires

Visualisation: Waveform (audiolab) Set

00:07 MARKER 1 OK

00:12 MARKER 2 EDIT

00:15 MARKER 3 EDIT

00:22 MARKER 4 EDIT

Download: OGG FLAC MP3 WAV

telemeta POWERED — Powered by Telemeta 0.4.1 By Parisson & Samalyse.

Copyright © 2010, CREM Mentions légales

Thank you !