Co-funded by the
Creative Europe Programme
of the European Union

This project has been funded with support from the European Commission. This document content reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

B-AIR airoscope

& AIR platform phase-two context

SYNOPSIS

airloader :: perl script that loads [gpp-airoscope] analysis yaml data into perl data-structures.

gpp-airoscope :: AIR platform sound analysis tools

This manual is formally a part of airloader but serves as a wider explanation of whole web audio/control framework concieved to support the B-AIR project's AIR platform. B-AIR Project, Creative Europe 2020-23.

[**gpp-airoscope**] PUREDATA PATCH

A swiss-knife audio-analysis tool modules that can be switched on or off depending on which parts of the analysis we want to use. The patch is intended to be used as a background process that takes care of performing various types of audio analysis or musical tissue, symultaneously and in realtime.

It works by performing analyses on a mono music stream and writes the results of these analyses to files (one per analysis) in yaml format, suitable for fast reading and conversion into data structures in a wide variety of programming languages.

TWO MODES OF EXECUTION

[**gpp-airoscope**] can be (a) executed from within another Puredata patch, as a patch element, or (b) called as execution parameter from command line or any kind of external application.

In case of (a) the [gpp-airoscope] patch will provide one control inlet and two control outlets.

Control inlet accepts a single stream of commands, generated as pd messages:

```
infile /path/of/some/input/audiofile.wav - input file to analyse
outfile /path/of/some/output/audiofile_root_without_extension - root of
output file, several files with different extensions will be created
using this root
drop <tool> | all - do not use that tool ( use 'all' to drop all tools )
use <tool> | all - use that tool ( or use 'all' )
start bang; - start the process
<tool> <specific_command> - send a tool-specific command to a tool. See
specific tools' descriptions
```

In case of b) the invoking script or application should call Puredata in the following way:

```
pd -nogui -send "<COMMAND> <param>; <COMMAND> <param>; <COMMAND>
<param>;" gpp-airoscope.pd
```

where <COMMAND> <param> can be any of the following:

```
INFILE /path/of/some/input/audiofile.wav
OUTFILE /path/of/some/output/audiofile_root_without_extension
PARAM <pass_params_to_separate_tools> - see separate tools' description
DO <something>, where <something> can be whether 'drop <tool|all>' or
'use <tool|all>' command.
START bang; - start the analysis process.
```

AIROSCOPE TOOLS

temper

Records universal tempo analysis based on the publicly available aubio tool. It looks for areas of stable and unstable tempo, marks where they start in time, records individual rhythmic accents within them and adds additional analysis of average values for the entire recording or individual regions.

There are two temper subsystems that enable consistent following of tempo changes: [gpp-sepachain~] and [gpp-fifostat~]. [gpp-sepachain~] module is a separation chain. It analyses a stream of numbers and performs statistical operatins using [gpp-fifostat~], dividing the main output according to the statistical analysis performed. In other words, when [gpp-sepachain~] thinks that statistically the prevailing value of a stream has changed, it will output additional data describing the change and will remap the stream to another output.

temper input parameters:

- `fiforange` <from> <to> - the span of values processed.
- `fifosize` <number_of_values> - actual size of fifo, affecting decisions when to acknowledge that the tempo has changed
- `transition_threshold` <factor> - number ranging from 0.5-1 that determines what share of fifo should be populated with certain value in order to acknowledge that the value 'is prevailing'.
- `aubio_threshold` <factor> - parameter for [aubiotempo~] patch, describing the sensitivity of the tempo recognition tool. See documentation on [aubiotempo~] and libaubio.

temper output structure:

header => (overall recording data)

```
# regions: <number of regions found>

# beats: <number of beats found>

# fifostat_size: <fifostat size used>

# fifostat_range_from: <fifostat range used from>

# fifostat_range_to: <fifostat range used to>

# transition_threshold: <transition threshold used>

# aubio_threshold: <aubio threshold used>

# average_tempo: <overall average tempo of the recording>
```

region => <list of regions> where each <region> contains:

```
 # id: <region_id>

 # is_elusive: <boolean> - whether the prevailing tempo can be
determined, or not

 # average_tempo: <tempo>

 # prevalent tempo: <tempo> - only if <is_elusive> = 0

 # at_time: <timing>

 # at_ms: <miliseconds>

 # beat: <list_of_beats> where a <beat> contains:

 # tempo: <beat's tempo regarding to the previous beat>

 # id: <beat id>

 # at_ms: <beat timing in miliseconds>

 # at_time: <beat timing>
```

dynoz

Records volume change analysis. It records volume changes, dividing the entire range into ten basic values, from absolute silence (tacet) to maximum volume (fortissimo possibile). It records the time of individual dynamic changes, the RMS value of the envelope, the associated musical symbols and various types of averaging.

The output dynamic range is divided into 10 bands which can be described numerically (degree) or symbolically (symbol)

```
0 = tacet (absolute silence)
1 = pppp
2 = ppp
3 = pp
4 = p
5 = mp
6 = mf
7 = f
8 = ff
9 = fff
10 = ffff
```

These are absolute dynamic, not 'psychological' values.

dynoz output structure:

header => (overall recording data)

```
 # number_of_dynamics: <number of dynamic changes found>

 # average_dynamics_occurences_degree: <average dynamics degree
statistically detected>

 # average_dynamics_occurences_symbol: <average dynamics symbol
statistically detected>

 # average_dynamics_duration_degree: <average dynamics degree expressed
as the share of the duration share>

 # average_dynamics_duration_symbol: <average dynamics symbol expressed
as the share of the duration share>

 # min_dynamics_degree: <minimal dynamics degree>
```

max_dynamics_degree: <maximal dynamics degree found>

min_dynamics_symbol: <minimal dynamics symbol found>

max_dynamics_symbol: <maximal dynamics symbol found>

min_env_detected: <minimal envelope ([env~] RMS) value detected>

max_env_detected: <maximal envelope ([env~] RMS) value detected>

pitcher

Records the analysis of the pitch base change. This, from a musical point of view, is usually the pitch that occurs in the treble (highest voice) and therefore specifically characterises the current sound situation. At each pitch change, it records the time, frequency, standard musical note, its name and octave, as well as the maximums and averages of the events or time content of pitches.

Each pitch is described as follows:

frequency = exact frequency (Hz)
note_midivalue = numeric midi note value
note_name = musical note name (using sharps):

c,c#,d,d#,e,f,f#,g,g#,a,a#,b

note_octave = in which octave musical note name appears:

-4 subsubcontra octave
-3 subcontra octave
-2 contraoctave
-1 major octave
0 minor octave
1 one-line octave
2 two-line octave
3 three-line octave
(...)

note_detune_cents = detune value in cents (the difference in cents from the note_name frequency @ A1 = 440Hz tuning)

pitcher output structure:

header=> (overall recording data)

number_of_pitches: <number of pitches found>

average_frequency: <average frequency in Hz>

average_note_name: <average note name>

average_note_octave: <average note octave>

average_note_detune_cents: <average note detune_cents>

pitch=> list of pitches where each <pitch> contains:

frequency: <frequency in Hz>

note_name: <note name as explained above>

note_octave: <note octave as explained above>

note_detune_cents: <note detune value as explained above>

```
# env: <pitch envelope>

# id: <pitch id>

# note_midivalue: <note_midivalue as explained>

# at_ms: <pitch timing in miliseconds>

# at_time: <pitch timing>
```

beater and bonker

Beater and bonker have properties and structure similar to temper, just the tempo-analysis method differs.

EXAMPLES

An example of puredata **-send** block for command-line call of [gpp-airoscope]:

```
INFILE /some/file;
OUTFILE /some/file/root;
DO drop all;
DO use dynoz;
DO use temper;
DO use pitcher;
PARAM temper fiforange 0 500;
PARAM temper fifosize 8;
PARAM temper transition_threshold 0.8;
PARAM temper aubio_threshold 0.3;
START bang;
```

AUTHOR

Gregor Pirs <gregor.pirs@guest.arnes.si>

VERSION

[20-11-2023]