JEVisualizer documentation

Hannes Ihalainen

October 13, 2018

Contents

1 Introduction

JEVisualizer is a command line program that creates a visualization video for music. The development started, when we wanted to add visualizations for our music on YouTube. One important goal is to develop JEVisualizer in such a way that it can be configured to create very different visualizations - only imagination should be the limit.

JEVisualizer is designed to be used from command line and configured in configuration files. Config files and command lines are nice, much more nice than GUIs. GUIs are slow, they often crash for no reason, and creating a GUI would take overs 9000% time compared to a creating a simple program without a GUIs.

That's why we don't need any GUIs.

No GUI is a wise choice.

2 How does it work

The basic idea is that JEVisualizer takes as an input any number of *tracks*, and outputs a video that visualizes those *tracks*.

A track contains a number of *notes*, which get different intensities. Currently the program can extract tracks from the channels of wav-files (using an algorithm that detects frequencies) or from the tracks of mmp-files (Linux multimedia studio -files).

For the output-video, any number of *layers* can be configured. *A layer* can contain a picture, any number of *drawers* and any number of *filters*.

A drawer visualizes some notes from some track(s).

Filters are used to create cool effects on layers or output-video.

More about all these in section "How to config".

3 Parameters that can be given to JEVisualizer

Following parameters can be given to JEVisualizer.

–config file Sets a file from which main configuration is loaded. Default is main.config

–verbose0, -v0 Sets verbose level to 0. Only errors are printed on stdout.

–verbose1, -v1 Sets verbose level to 1. Errors and warnings are printed on stdout.

–verbose2, -v2 Sets verbose level to 2. Errors, warnings and some other information are printed on stdout.

–verbose3, -v3 Sets verbose level to 2. Errors, warnings and some more other information are printed on stdout.

–no-run Do not run create the output-video, only load tracks. -# is the delimeter between parameter name and key

4 How to config

JEVisualizer contains a number of different config-files.

First is **main.config**. It contains information about tracks and the name of visualizer-config file.

Second is **visualizer.config**. It defines basic information about the output-video, like width, height, fps, filename etc. It also contains the layers as filenames.

Thirdly there can be any number of **layer-config-files**. Each layer-config-file contains information about that layer. It can contain a background-image, information about when the layer is visible, a number of drawers and a number of different filters.

From configuration-files it is also possible to include other configuration files, so there could also be a different config-file for some drawers or filter etc.

4.1 JEVisualizer configuration file format

JEVisualizer uses its own syntax in configuration files.

Here are the basic rules of JEVisualizer configuration syntax: Configuration files are of a simple attribute-value format. Each attribute and value -pair must be on a single line (except when value is inside "", " or , when it can contain linebreaks). Basic format is: **attribute=value**.

spaces and tabs	spaces and tabs don't matter (except in values inside "", " or {})
linebreaks	linebreaks do matter, they separate different attribute-value pairs.
commenting	// comments the end of line, multiline comments can be written between /* and */.
"", " and	Values containing spaces and linebreaks can be set inside "", " or {}. "", " and {} do note have any functional differences. Inside of "", " or {} characters can be escaped using character. Note: p={c={...}} wouldn't work as expected, something like p={c=\{...\}} should be used.

including other files — Special command: #include="filename", includes config file "filename".

#-character — #-character in attribute is the delimeter between name and key. (for some attributes there can be different keys...)

UPPER and lower case — Parameter names will be converted to smallcase when reading file. A=b is same as a=b.

Example:

```
PARAMETER1=VALUE
PARAMETER2="VALUE WITH SPACES"
PARAMETER3={
  P=2
  P=2
  P3="SOME VALUE WITH SPACES"
}
#include="file2.config"
PARAMETER4#KEY1=2.0
PARAMETER4#6=9.2
```

For further details see implementation in *config.h* and *config.cpp*.

4.2 main.config

Following paramaters can be used in main.config

TRACK#id — The information about track *id*, where *id* must be an integer $0 \leq id$. Value should be multiline value with following parameters:

- TYPE — Type of the track: value should be WAV (1) or MMP (2).
- FILE — Filename where the data is. WAV-file should be of type **16-bit**, **32-bit float** is currently **not** suppored.

Parameters of track of type WAV :

- CHANNEL — The chanel from which the track is extracted. Usually 0, but for example stereo wav-files it would be possible to create a track for left channel and a track for right channel (0 and 1)
- F0 — Lowest frequency (in Hz) to extract from the file
- F1 — Upper bound for frequencies to extract from file (in Hz)
- FK — A multiplier for frequencies: frequencies $F0 * FK^i$ are extracted. One example of good values is 1.0594630943592953.

CHLEN — A parameter for the frequency-extracting algorithm: a sample of length at most CHLEN is used to determine how loud certain frequency is at certain time. (however, this is a kind of legacy parameter, since the algorithm has changed a bit...) A good value is like 1.

THR — A parameter for the frequency-extracting algorithm that: Removes noise from the outcome, if the loudness of a note is less than THR, the loudness is set to zero. A good value is 0.01, since the loudness is estimated with a value on range $[0, 1]$.

See file wav.cpp for further details about the parameters above.

Parameters of track of type MMP:

TRACK — The index of track in MMP. If unsure about the index, a good idea is to set it to zero, run JEVIsualizer -v2 –no-run, and the track indices with track names are printed on stdout.

VISUALIZER — Filename of the visualizer config file. Usually "visualizer.config".

ON-END — Command that is executed after the video is ready. (Often something like a ffmpeg -command that adds sound to the video)

A-FREQUENCY — The frequency of how often the track-values do change. Good value is about the FPS. Value should be a floating-point number (or an integer)

Example of main.config:

```
TRACK#0={
  TYPE=WAV
  F0=16.35
  F1=10000
  FK=1.0594630943592953
  CHLEN=1
  THR=0.01
  FILE='media/soundfile.wav'
}

TRACK#1={
  TYPE=MMP
  FILE='media/llms-file.mmp'
  TRACK=23
}

A-FREQUENCY=40
```

```
VISUALIZER="visualizer.config"

ON-END="ffmpeg -i output/video.ogv -i media/soundfile.ogg -shortest -vcodec copy output/video_with_
-y"
```

Implementation of the reading of the parameters of main.config can be found in *controller.cpp*.

4.3 visualizer.config

Following paramaters can be used in visualizer.config (or whatever filename is used for the configuration of visualizer)

FPS	The fps-rate of the output-video. Value shuold be a floating-point-number (or an integer)
W or WIDTH	The width of the output-video in pixels.
H or HEIGHT	The height of the output-video in pixels.
OUTPUT-FILE	The filename of the output-video. Should be .ogg/.ogv since ogg-format is used.
FIRST-FRAME	The index of the first frame that is drawn.
LAST-FRAME	The index of the last frame that is drawn.
FPV#id	Set a parameter for formulas, see section formulas below for further details. Possible values: frame, fps, w, h, time and null.
LAYER#id	Set a filename, where *id*th layer is defined. Layers are drawn in order 0, 1, 2, ...

Example of visualizer.config:

```
W=640
H=480

FPS=40
OUTPUT-FILE="output/test.ogv"

FIRST-FRAME=0
LAST-FRAME=600

FPV#0=frame
LAYER#0="layer0.config"
LAYER#1="layer1.config"
```

Implementation of the reading of the parameters of visualizer.config can be found in visualizer.cpp.

4.4 Formulas

Most of the values that are used are given as datatype Formula (defined in *formula.h* and *formula.cpp*). The value of a formula is calculated again every time a new frame is drawn. Formula can be either a simple constant or it can depend on many variables.
Basic format of formula is either:
1. CONSTANT , where CONSTANT is simple floating-point number
2. (CONSTANT, ARRAY_OF_VARIABLE_COEFFICIENTS] MIN_VALUE, MAX_VALUE, SIN_COEFFICIENT, IN_SIN)
Value of this type of formula is:
$min(max($
$CONSTANT + \sum_i ARRAY_i * variable_i + SIN\_COEFFICIENT * sin(IN\_SIN),$
$MIN\_VALUE), MAX\_VALUE)$
Each parameter is another formula
Formula can have all parameters or n first of them, like for example
$(CONSTANT,\ ARRAY\_OF\_VARIABLE_COEFFICIENTS)$

Variables the formulas can use are also defined in config-files with:
$FPV\#i = name$
This sets that $variable_i$ is getting the value from *name* ,where *name* is some valid variable that can be used as a *formula parameter variable*. For example $frame$ can be used as a *formula parameter variable* in visualizer.config.

Examples of valid formulas:

5	Formula has constant value 5.
(5)	Formula contains a constant formula with value $5 - >$ does same as formula 5 above, but is a bit slower to calculated.
$(5,\ [1])$	Formula has value $5 + FPV\#0$
$(0, [],\ -1,\ 1,\ 1,\ (0,\ 1))$	Formula has value $sin(FPV\#0)$. (Min and max don't have effect since the value of *sin* is always between -1 and 1)

The bracket type used can be anywhere either round brackets () or box brackets []. However, it is recommended to use box brackets to indicate an array.

4.4.1 FPV:s

The variables formulas use are defined with FPV#i=name. Different variables can be used in different scopes. Variables that are defined in an outer scope, are inherited to the inner scope, even when the variable wouldn't otherwise be accessible on that scope.
The different scopes are:

visualizer	doesn't have a parent-scope
layer	each layer has a *visualizer* as a parent
drawer	each drawer has a *layer* as a parent
filter	each filter has a *layer* as a parent

So if in visualizer.config there is defined FPV#0=frame, all scopes inherit this variable and have the index of current frame as a $variabe_0$. (unless FPV#0 is rewritten or removed in some scope, in which case all the children of that scope inherit the rewritten or removed $variable_0$)
Here is a list of different variable names that for all scopes. (this list is not necessarily up to date, but at least we have a list)
In visualizer:

frame	the index of the frame
fps	the fps of output-video
w	the width of output-video in pixels
h	the height of output-video in pixels
null	unset parameter

In layers:

layer-width	the width of the layer.
layer-height	the height of the layer.
track-valuesNOTES	notes whose values are summed up and this sum is the variable: NOTES are given as follows: TRACKID:FROMNOTE-TONOTE,TRACKID2:FROMNOTE2-TONOTE2 for example: track-values0:0-13,1:0-13 is a sum of notes 0-13 on tracks 0 and 1.

null	unset a parameter

In drawers:

track-valuesNOTES	notes whose values are summed up and this sum is the variable: NOTES are given as follows: TRACKID:FROMNOTE-TONOTE,TRACKID2:FROMNOTE2-TONOTE2 for example: track-values0:0-13,1:0-13 is a sum of notes 0-13 on tracks 0 and 1.
null	unset a parameter

In filters there is currently no possibility to edit FPV:s.