Package: av (via r-universe)

October 27, 2026

Type Package

Title Working with Audio and Video in R

Version 0.9.6

Description Bindings to 'FFmpeg' <http://www.ffmpeg.org/> AV library for working with audio and video in R. Generates high quality video from images or R graphics with custom audio. Also offers high performance tools for reading raw audio, creating 'spectrograms', and converting between countless audio / video formats. This package interfaces directly to the C API and does not require any command line utilities.

License MIT + file LICENSE

URL https://ropensci.r-universe.dev/av, https://docs.ropensci.org/av/

BugReports https://github.com/ropensci/av/issues

Encoding UTF-8

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

SystemRequirements FFmpeg (>= 3.2); with at least libx264 and lame (mp3) drivers. MacOS Homebrew: ffmpeg. Debian/Ubuntu: libavfilter-dev. Fedora/RHEL: either ffmpeg-devel from https://rpmfusion.org (preferred), or libavfilter-free-devel which is a more limited version available on Fedora/EPEL9.

Depends R (>= 3.5)

Imports graphics

Config/Needs/website seewave, ggplot2, phonTools, signal, tuneR

Suggests curl, testthat, ps, ggplot2, gapminder

Language en-US

Config/pak/sysreqs libavfilter-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2026-10-27 18:13:30 UTC

RemoteUrl https://github.com/ropensci/av

RemoteRef master

RemoteSha a8dbf77ac586f5fa8ec6040a9a3bf4e73432ad00

Contents

av_video_images . . . 2
capturing . . . 3
demo . . . 4
encoding . . . 5
formats . . . 7
info . . . 8
logging . . . 8
read_audio_fft . . . 9
window functions . . . 10

Index **12**

`av_video_images` *Convert video to images*

Description

Splits a video file in a set of image files. Default image format is jpeg which has good speed and compression. Use `format = "png"` for losless images.

Usage

```
av_video_images(
  video,
  destdir = tempfile(),
  format = "jpg",
  fps = NULL,
  trim = NULL
)
```

Arguments

`video`	an input video
`destdir`	directory where to save the png files
`format`	image format such as `png` or `jpeg`, must be available from `av_encoders()`
`fps`	sample rate of images. Use `NULL` to get all images.
`trim`	string value for ffmpeg trim filter for example `"10:15"` for seconds or `"start_frame=100:end_frame=11` for frames.

Details

For large input videos you can set fps to sample only a limited number of images per second. This also works with fractions, for example `fps = 0.2` will output one image for every 5 sec of video.

Examples

```
## Not run:
curl::curl_download('https://jeroen.github.io/images/blackbear.mp4', 'blackbear.mp4')
av_video_images('blackbear.mp4', fps = 1, trim = "10:20")

## End(Not run)
```

`capturing` *Record Video from Graphics Device*

Description

Runs the expression and captures all plots into a video. The av_spectrogram_video function is a wrapper that plots data from read_audio_fft with a moving bar and background audio.

Usage

```
av_capture_graphics(
  expr,
  output = "output.mp4",
  width = 720,
  height = 480,
  framerate = 1,
  vfilter = "null",
  audio = NULL,
  verbose = TRUE,
  ...
)

av_spectrogram_video(
  audio,
  output = "output.mp4",
  framerate = 25,
  verbose = interactive(),
  ...
)
```

Arguments

`expr`	an R expression that generates the graphics to capture
`output`	name of the output file. File extension must correspond to a known container format such as `mp4`, `mkv`, `mov`, or `flv`.

width	width in pixels of the graphics device
height	height in pixels of the graphics device
framerate	video framerate in frames per seconds. This is the input fps, the output fps may be different if you specify a filter that modifies speed or interpolates frames.
vfilter	a string defining an ffmpeg filter graph. This is the same parameter as the `-vf` argument in the `ffmpeg` command line utility.
audio	path to media file with audio stream
verbose	emit some output and a progress meter counting processed images. Must be `TRUE` or `FALSE` or an integer with a valid av_log_level.
...	extra graphics parameters passed to `png()`

See Also

Other av: `demo()`, `encoding`, `formats`, `info`, `logging`, `read_audio_fft()`

Examples

```
library(gapminder)
library(ggplot2)
makeplot <- function(){
  datalist <- split(gapminder, gapminder$year)
  lapply(datalist, function(data){
    p <- ggplot(data, aes(gdpPercap, lifeExp, size = pop, color = continent)) +
    scale_size("population", limits = range(gapminder$pop)) + geom_point() + ylim(20, 90) +
    scale_x_log10(limits = range(gapminder$gdpPercap)) + ggtitle(data$year) + theme_classic()
    print(p)
  })
}

# Play 1 plot per sec, and use an interpolation filter to convert into 10 fps
video_file <- file.path(tempdir(), 'output.mp4')
av_capture_graphics(makeplot(), video_file, 1280, 720, res = 144, vfilter = 'framerate=fps=10')
av::av_media_info(video_file)
# utils::browseURL(video_file)
```

---

demo *Demo Video*

Description

Generates random video for testing purposes.

Usage

```
av_demo(
  output = "demo.mp4",
  width = 960,
  height = 720,
  framerate = 5,
  verbose = TRUE,
  ...
)
```

Arguments

output	name of the output file. File extension must correspond to a known container format such as mp4, mkv, mov, or flv.
width	width in pixels of the graphics device
height	height in pixels of the graphics device
framerate	video framerate in frames per seconds. This is the input fps, the output fps may be different if you specify a filter that modifies speed or interpolates frames.
verbose	emit some output and a progress meter counting processed images. Must be TRUE or FALSE or an integer with a valid av_log_level.
...	other parameters passed to av_capture_graphics.

See Also

Other av: capturing, encoding, formats, info, logging, read_audio_fft()

---

encoding *Encode or Convert Audio / Video*

Description

Encodes a set of images into a video, using custom container format, codec, fps, video filters, and audio track. If input contains video files, this effectively combines and converts them to the specified output format.

Usage

```
av_encode_video(
  input,
  output = "output.mp4",
  framerate = 24,
  vfilter = "null",
  codec = NULL,
  audio = NULL,
  verbose = TRUE
```

```
)

av_video_convert(video, output = "output.mp4", verbose = TRUE)

av_audio_convert(
  audio,
  output = "output.mp3",
  format = NULL,
  channels = NULL,
  sample_rate = NULL,
  bit_rate = NULL,
  start_time = NULL,
  total_time = NULL,
  verbose = interactive()
)
```

Arguments

input	a vector with image or video files. A video input file is treated as a series of images. All input files should have the same width and height.
output	name of the output file. File extension must correspond to a known container format such as mp4, mkv, mov, or flv.
framerate	video framerate in frames per seconds. This is the input fps, the output fps may be different if you specify a filter that modifies speed or interpolates frames.
vfilter	a string defining an ffmpeg filter graph. This is the same parameter as the -vf argument in the ffmpeg command line utility.
codec	name of the video codec as listed in av_encoders. The default is libx264 for most formats, which usually the best choice.
audio	audio or video input file with sound for the output video
verbose	emit some output and a progress meter counting processed images. Must be TRUE or FALSE or an integer with a valid av_log_level.
video	input video file with optionally also an audio track
format	a valid output format name from the list of av_muxers(). Default NULL infers format from the file extension.
channels	number of output channels. Default NULL will match input.
sample_rate	output sampling rate. Default NULL will match input.
bit_rate	output bitrate (quality). A common value is 192000. Default NULL will match input, except for lossless input (such as wav or flac): its bitrate is far above what a lossy encoder accepts, so the encoder's own default is used instead.
start_time	number greater than 0, seeks in the input file to position.
total_time	approximate number of seconds at which to limit the duration of the output file.

Details

The target container format and audio/video codes are automatically determined from the file extension of the output file, for example `mp4`, `mkv`, `mov`, or `flv`. For video output, most systems also support `gif` output, but the compression~quality for gif is really bad. The gifski package is better suited for generating animated gif files. Still using a proper video format is results in much better quality.

It is recommended to use let ffmpeg choose the suitable codec for a given container format. Most video formats default to the `libx264` video codec which has excellent compression and works on all modern browsers, operating systems, and digital TVs.

To convert from/to raw PCM audio, use file extensions `".ub"` or `".sb"` for 8bit unsigned or signed respectively, or `".uw"` or `".sw"` for 16-bit, see `extensions` in `av_muxers()`. Alternatively can also convert to other raw audio PCM by setting for example `format = "u16le"` (i.e. unsigned 16-bit little-endian) or another option from the `name` column in `av_muxers()`.

It is safe to interrupt the encoding process by pressing CTRL+C, or via setTimeLimit. When the encoding is interrupted, the output stream is properly finalized and all open files and resources are properly closed.

See Also

Other av: `capturing`, `demo()`, `formats`, `info`, `logging`, `read_audio_fft()`

`formats` *AV Formats*

Description

List supported filters, codecs and container formats.

Usage

```
av_encoders()

av_decoders()

av_filters()

av_muxers()

av_demuxers()
```

Details

Encoders and decoders convert between raw video/audio frames and compressed stream data for storage or transfer. However such a compressed data stream by itself does not constitute a valid video format yet. Muxers are needed to interleave one or more audio/video/subtitle streams, along with timestamps, metadata, etc, into a proper file format, such as mp4 or mkv.

Conversely, demuxers are needed to read a file format into the separate data streams for subsequent decoding into raw audio/video frames. Most operating systems natively support demuxing and decoding common formats and codecs, needed to play those videos. However for encoding and muxing such videos, ffmpeg must have been configured with specific external libraries for a given codec or format.

See Also

Other av: capturing, demo(), encoding, info, logging, read_audio_fft()

---

info *Video Info*

Description

Get video info such as width, height, format, duration and framerate. This may also be used for audio input files.

Usage

```
av_media_info(file)
```

Arguments

file	path to an existing file

See Also

Other av: capturing, demo(), encoding, formats, logging, read_audio_fft()

---

logging *Logging*

Description

Get or set the log level.

Usage

```
av_log_level(set = NULL)
```

Arguments

set	new log level value

See Also

Other av: capturing, demo(), encoding, formats, info, read_audio_fft()

read_audio_fft *Read audio binary and frequency data*

Description

Reads raw audio data from any common audio or video format. Use read_audio_bin to get raw PCM audio samples, or read_audio_fft to stream-convert directly into frequency domain (spectrum) data using FFmpeg built-in FFT.

Usage

```
read_audio_fft(
  audio,
  window = hanning(1024),
  overlap = 0.75,
  sample_rate = NULL,
  start_time = NULL,
  end_time = NULL
)

read_audio_bin(
  audio,
  channels = NULL,
  sample_rate = NULL,
  start_time = NULL,
  end_time = NULL
)

write_audio_bin(
  pcm_data,
  pcm_channels = 1L,
  pcm_format = "s32le",
  output = "output.mp3",
  ...
)
```

Arguments

audio	path to the input sound or video file containing the audio stream
window	vector with weights defining the moving fft window function. The length of this vector is the size of the window and hence determines the output frequency range.
overlap	value between 0 and 1 of overlap proportion between moving fft windows
sample_rate	downsample audio to reduce FFT output size. Default keeps sample rate from the input file.

start_time, end_time
: position (in seconds) to cut input stream to be processed.

channels
: number of output channels, set to 1 to convert to mono sound

pcm_data
: integer vector as returned by read_audio_bin

pcm_channels
: number of channels in the data. Use the same value as you entered in read_audio_bin.

pcm_format
: this is always s32le (signed 32-bit integer) for now

output
: passed to av_audio_convert

...
: other paramters for av_audio_convert

Details

Currently read_audio_fft automatically converts input audio to mono channel such that we get a single matrix. Use the plot() method on data returned by read_audio_fft to show the spectrogram. The av_spectrogram_video generates a video that plays the audio while showing an animated spectrogram with moving status bar, which is very cool.

See Also

Other av: capturing, demo(), encoding, formats, info, logging

Examples

```
# Use a 5 sec fragment
wonderland <- system.file('samples/Synapsis-Wonderland.mp3', package='av')

# Read initial 5 sec as as frequency spectrum
fft_data <- read_audio_fft(wonderland, end_time = 5.0)
dim(fft_data)

# Plot the spectrogram
plot(fft_data)

# Show other parameters
dim(read_audio_fft(wonderland, end_time = 5.0, hamming(2048)))
dim(read_audio_fft(wonderland, end_time = 5.0, hamming(4096)))
```

---

window functions *Window functions*

Description

Several common windows function generators. The functions return a vector of weights to use in read_audio_fft.

Usage

```
hanning(n)

hamming(n)

blackman(n)

bartlett(n)

welch(n)

flattop(n)

bharris(n)

bnuttall(n)

sine(n)

nuttall(n)

bhann(n)

lanczos(n)

gauss(n)

tukey(n)

dolph(n)

cauchy(n)

parzen(n)

bohman(n)
```

Arguments

n	size of the window (number of weights to generate)

Examples

```
# Window functions
plot(hanning(1024), type = 'l', xlab = 'window', ylab = 'weight')
lines(hamming(1024), type = 'l', col = 'red')
lines(bartlett(1024), type = 'l', col = 'blue')
lines(welch(1024), type = 'l', col = 'purple')
lines(flattop(1024), type = 'l', col = 'darkgreen')
```

Index

∗ **av**
    capturing, 3
    demo, 4
    encoding, 5
    formats, 7
    info, 8
    logging, 8
    read_audio_fft, 9

av *(*encoding*)*, 5
av_audio_convert, *10*
av_audio_convert *(*encoding*)*, 5
av_capture_graphics, *5*
av_capture_graphics *(*capturing*)*, 3
av_decoders *(*formats*)*, 7
av_demo *(*demo*)*, 4
av_demuxers *(*formats*)*, 7
av_encode_video *(*encoding*)*, 5
av_encoders, *6*
av_encoders *(*formats*)*, 7
av_filters *(*formats*)*, 7
av_log_level, *4–6*
av_log_level *(*logging*)*, 8
av_media_info *(*info*)*, 8
av_muxers *(*formats*)*, 7
av_muxers(), *7*
av_spectrogram_video, *3*, *10*
av_spectrogram_video *(*capturing*)*, 3
av_video_convert *(*encoding*)*, 5
av_video_images, 2
av_video_info *(*info*)*, 8

bartlett *(*window functions*)*, 10
bhann *(*window functions*)*, 10
bharris *(*window functions*)*, 10
blackman *(*window functions*)*, 10
bnuttall *(*window functions*)*, 10
bohman *(*window functions*)*, 10

capturing, 3, *5*, *7, 8*, *10*
cauchy *(*window functions*)*, 10

demo, 4, *4*, *7, 8*, *10*
dolph *(*window functions*)*, 10

encoding, *4*, 5, *5*, *8*, *10*

fft window function, *9*
flattop *(*window functions*)*, 10
formats, *4, 5*, 7, *7, 8*, *10*

gauss *(*window functions*)*, 10

hamming *(*window functions*)*, 10
hanning *(*window functions*)*, 10

info, *4, 5*, *7*, 8, *8*, *10*

lanczos *(*window functions*)*, 10
logging, *4, 5*, *7*, 8, *8*, *10*

nuttall *(*window functions*)*, 10

parzen *(*window functions*)*, 10
png(), *4*

read_audio_bin, *9, 10*
read_audio_bin *(*read_audio_fft*)*, 9
read_audio_fft, *3–5*, *7, 8*, 9, *9, 10*

setTimeLimit, *7*
sine *(*window functions*)*, 10

tukey *(*window functions*)*, 10

welch *(*window functions*)*, 10
window functions, 10
write_audio_bin *(*read_audio_fft*)*, 9