ffmpeg Documentation

Table of Contents

1. Synopsis

The generic syntax is:

```
ffmpeg [global options] [[infile options][‘-i’ infile]]... {[outfile options] outfile}...
```

2. Description

ffmpeg is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

ffmpeg reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output filename. Anything found on the command line which cannot be interpreted as an option is considered to be an output filename.

Each input or output file can in principle contain any number of streams of different types (video/audio/subtitle/attachment/data). Allowed number and/or types of streams can be limited by the container format. Selecting, which streams from which inputs go into output, is done either automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1` etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. See also the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

- To set the video bitrate of the output file to 64kbit/s:

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

- To force the frame rate of the output file to 24 fps:

```
ffmpeg -i input.avi -r 24 output.avi
```

- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

```
ffmpeg -r 1 -i input.m2v -r 24 output.avi
```

The format option may be needed for raw input files.

3. Detailed description

The transcoding process in `ffmpeg` for each output can be described by the following diagram:

```
 _______              ______________               _________              ______________            ________
|       |            |              |             |         |            |              |          |        |
| input |  demuxer   | encoded data |   decoder   | decoded |  encoder   | encoded data |  muxer   | output |
| file  | ---------> | packets      |  ---------> | frames  | ---------> | packets      | -------> | file   |
|_______|            |______________|             |_________|            |______________|          |________|
```

`ffmpeg` calls the libavformat library (containing demuxers) to read input files and get packets containing encoded data from them. When there are multiple input files, `ffmpeg` tries to keep them synchronized by tracking lowest timestamp on any active input stream.

Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering the frames are passed to the encoder, which encodes them and outputs encoded packets again. Finally those are passed to the muxer, which writes the encoded packets to the output file.

3.1 Filtering

Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs - simple and complex.

3.1.1 Simple filtergraphs

Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:

```
 _________                        __________              ______________
|         |                      |          |            |              |
| decoded |  simple filtergraph  | filtered |  encoder   | encoded data |
| frames  | -------------------> | frames   | ---------> | packets      |
|_________|                      |__________|            |______________|
```

Simple filtergraphs are configured with the per-stream '`-filter`' option (with '`-vf`' and '`-af`' aliases for video and audio respectively). A simple filtergraph for video can look for example like this:

```
 _______        _____________        _______        _____        ________
|       |      |             |      |       |      |     |      |        |
| input | ---> | deinterlace | ---> | scale | ---> | fps | ---> | output |
|_______|      |_____________|      |_______|      |_____|      |________|
```

Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

3.1.2 Complex filtergraphs

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case e.g. when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:

```
 _________
|         |
| input 0 |\                    __________
|_________| \                  |          |
             \   _________    /| output 0 |
              \ |         |  / |__________|
 _________     \| complex | /
|         |     |         |/
| input 1 |---->| filter  |\
|_________|     |         | \   __________
               /| graph   |  \ |          |
              / |         |   \| output 1 |
 _________   /  |_________|    |__________|
|         | /
| input 2 |/
|_________|
```

Complex filtergraphs are configured with the '`-filter_complex`' option. Note that this option is global, since a complex filtergraph by its nature cannot be unambiguously associated with a single stream or file.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

3.2 Stream copy

Stream copy is a mode selected by supplying the `copy` parameter to the ‘`-codec`’ option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will in this case simplify to this:

```
 _______              ______________            ________
|       |            |              |          |        |
| input |  demuxer   | encoded data |  muxer   | output |
| file  | ---------> | packets      | -------> | file   |
|_______|            |______________|          |________|
```

Since there is no decoding or encoding, it is very fast and there is no quality loss. However it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

4. Stream selection

By default ffmpeg includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria; for video it is the stream with the highest resolution, for audio the stream with the most channels, for subtitle it’s the first subtitle stream. In the case where several streams of the same type rate equally, the lowest numbered stream is chosen.

You can disable some of those defaults by using `-vn/-an/-sn` options. For full manual control, use the `-map` option, which disables the defaults just described.

5. Options

All the numerical options, if not specified otherwise, accept in input a string representing a number, which may contain one of the SI unit prefixes, for example ’K’, ’M’, ’G’. If ’i’ is appended after the prefix, binary prefixes are used, which are based on powers of 1024 instead of powers of 1000. The ’B’ postfix multiplies the value by 8, and can be appended after a unit prefix or used alone. This allows using for example ’KB’, ’MiB’, ’G’ and ’B’ as number postfix.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing with "no" the option name, for example using "-nofoo" in the command line will set to false the boolean option with name "foo".

5.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) does a given option belong to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. `-codec:a:1 ac3` option contains `a:1` stream specifier, which matches the second audio stream. Therefore it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, the option is then applied to all of them. E.g. the stream specifier in `-b:a 128k` matches all audio streams.

An empty stream specifier matches all streams, for example `-codec copy` or `-codec: copy` would copy all the streams without reencoding.

Possible forms of stream specifiers are:

‘*stream_index*’

> Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

‘*stream_type*[:*stream_index*]’

> *stream_type* is one of: ’v’ for video, ’a’ for audio, ’s’ for subtitle, ’d’ for data and ’t’ for attachments. If *stream_index* is given, then matches stream number *stream_index* of this type. Otherwise matches all streams of this type.

‘p:*program_id*[:*stream_index*]’

> If *stream_index* is given, then matches stream number *stream_index* in program with id *program_id*. Otherwise matches all streams in this program.

‘#*stream_id*’

> Matches the stream by format-specific ID.

5.2 Generic options

These options are shared amongst the av* tools.

‘`-L`’

> Show license.

‘`-h, -?, -help, --help [`*arg*`]`’

Show help. An optional parameter may be specified to print help about a specific item.

Possible values of *arg* are:

‘`decoder=`*`decoder_name`*’

Print detailed information about the decoder named *decoder_name*. Use the ‘`-decoders`’ option to get a list of all decoders.

‘`encoder=`*`encoder_name`*’

Print detailed information about the encoder named *encoder_name*. Use the ‘`-encoders`’ option to get a list of all encoders.

‘`demuxer=`*`demuxer_name`*’

Print detailed information about the demuxer named *demuxer_name*. Use the ‘`-formats`’ option to get a list of all demuxers and muxers.

‘`muxer=`*`muxer_name`*’

Print detailed information about the muxer named *muxer_name*. Use the ‘`-formats`’ option to get a list of all muxers and demuxers.

‘`-version`’

Show version.

‘`-formats`’

Show available formats.

The fields preceding the format names have the following meanings:

‘`D`’

Decoding available

‘`E`’

Encoding available

‘`-codecs`’

Show all codecs known to libavcodec.

Note that the term ’codec’ is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

‘`-decoders`’

Show available decoders.

‘`-encoders`’

Show all available encoders.

‘`-bsfs`’

Show available bitstream filters.

‘`-protocols`’

Show available protocols.

‘`-filters`’

Show available libavfilter filters.

‘`-pix_fmts`’

Show available pixel formats.

‘`-sample_fmts`’

Show available sample formats.

‘`-layouts`’

Show channel names and standard channel layouts.

‘`-loglevel` *`loglevel`* `| -v` *`loglevel`*’

Set the logging level used by the library. *loglevel* is a number or a string containing one of the following values:

‘`quiet`’
‘`panic`’
‘`fatal`’
‘`error`’
‘`warning`’
‘`info`’
‘`verbose`’
‘`debug`’

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable `AV_LOG_FORCE_NOCOLOR` or `NO_COLOR`, or can be forced setting the environment variable `AV_LOG_FORCE_COLOR`. The use of the environment variable `NO_COLOR` is deprecated and will be dropped in a following FFmpeg version.

‘`-report`’

Dump full command line and console output to a file named `*program*-*YYYYMMDD*-*HHMMSS*.log` in the current directory. This file can be useful for bug reports. It also implies `-loglevel verbose`.

Setting the environment variable `FFREPORT` to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ’:’. The following option is recognized:

‘`file`’

set the file name to use for the report; `%p` is expanded to the name of the program, `%t` is expanded to a timestamp, `%%` is expanded to a plain `%`

Errors in parsing the environment variable are not fatal, and will not appear in the report.

‘`-cpuflags flags (*global*)`’

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you’re doing.

```
ffmpeg -cpuflags -sse+mmx ...
ffmpeg -cpuflags mmx ...
ffmpeg -cpuflags 0 ...
```

5.3 AVOptions

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the ‘`-help`’ option. They are separated into two categories:

‘`generic`’

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

‘`private`’

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the ‘`id3v2_version`’ private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are obviously per-stream, so the chapter on stream specifiers applies to them

Note ‘`-nooption`’ syntax cannot be used for boolean AVOptions, use ‘`-option 0`’/‘`-option 1`’.

Note2 old undocumented way of specifying per-stream AVOptions by prepending v/a/s to the options name is now obsolete and will be removed soon.

5.4 Codec AVOptions

‘`-b[:stream_specifier]` *integer* (*output,audio,video*)’

set bitrate (in bits/s)

‘`-ab[:stream_specifier]` *integer* (*output,audio*)’

set bitrate (in bits/s)

‘`-bt[:stream_specifier]` *integer* (*output,video*)’

Set video bitrate tolerance (in bits/s). In 1-pass mode, bitrate tolerance specifies how far ratecontrol is willing to deviate from the target average bitrate value. This is not related to min/max bitrate. Lowering tolerance too much has an adverse effect on quality.

‘`-flags[:stream_specifier]` *flags* (*input/output,audio,video,subtitles*)’

Possible values:

‘`mv4`’

use four motion vector by macroblock (mpeg4)

‘`qpel`’

use 1/4 pel motion compensation

‘`loop`’

use loop filter

‘`qscale`’

use fixed qscale

‘`gmc`’

use gmc

‘`mv0`’

always try a mb with mv=<0,0>

‘`input_preserved`’
‘`pass1`’

use internal 2pass ratecontrol in first pass mode

‘`pass2`’

use internal 2pass ratecontrol in second pass mode

‘`gray`’

only decode/encode grayscale

‘`emu_edge`’

don’t draw edges

‘`psnr`’

error[?] variables will be set during encoding

‘`truncated`’
‘`naq`’

normalize adaptive quantization

‘`ildct`’

use interlaced dct

‘`low_delay`’

force low delay

‘`global_header`’

place global headers in extradata instead of every keyframe

‘`bitexact`’

use only bitexact stuff (except (i)dct)

‘`aic`’

h263 advanced intra coding / mpeg4 ac prediction ‘`cbp`’

Deprecated, use mpegvideo private options instead

‘`qprd`’

Deprecated, use mpegvideo private options instead

‘`ilme`’

interlaced motion estimation

‘`cgop`’

closed gop

‘`-sub_id[:stream_specifier] *integer* ()`’
‘`-me_method[:stream_specifier] *integer* (*output,video*)`’

set motion estimation method

Possible values:

‘`zero`’

zero motion estimation (fastest)

‘`full`’

full motion estimation (slowest)

‘`epzs`’

EPZS motion estimation (default)

‘`esa`’

esa motion estimation (alias for full)

‘`tesa`’

tesa motion estimation

‘`dia`’

dia motion estimation (alias for epzs)

‘`log`’

log motion estimation

‘`phods`’

phods motion estimation

‘`x1`’

X1 motion estimation

‘`hex`’

hex motion estimation

‘`umh`’

umh motion estimation

‘`iter`’

iter motion estimation

‘`-extradata_size[:stream_specifier]` *integer* `()`’
‘`-time_base[:stream_specifier]` *rational number* `()`’
‘`-g[:stream_specifier]` *integer* `(`*output,video*`)`’

set the group of picture size

‘`-ar[:stream_specifier]` *integer* `(`*input/output,audio*`)`’

set audio sampling rate (in Hz)

‘`-ac[:stream_specifier]` *integer* `(`*input/output,audio*`)`’

set number of audio channels

‘`-cutoff[:stream_specifier]` *integer* `(`*output,audio*`)`’

set cutoff bandwidth

‘`-frame_size[:stream_specifier]` *integer* `(`*output,audio*`)`’
‘`-frame_number[:stream_specifier]` *integer* `()`’
‘`-delay[:stream_specifier]` *integer* `()`’
‘`-qcomp[:stream_specifier]` *float* `(`*output,video*`)`’

video quantizer scale compression (VBR). Constant of ratecontrol equation. Recommended range for default rc_eq: 0.0-1.0

‘-qblur[:stream_specifier] *float* (*output,video*)’

video quantizer scale blur (VBR)

‘-qmin[:stream_specifier] *integer* (*output,video*)’

min video quantizer scale (VBR)

‘-qmax[:stream_specifier] *integer* (*output,video*)’

max video quantizer scale (VBR)

‘-qdiff[:stream_specifier] *integer* (*output,video*)’

max difference between the quantizer scale (VBR)

‘-bf[:stream_specifier] *integer* (*output,video*)’

use ’frames’ B frames

‘-b_qfactor[:stream_specifier] *float* (*output,video*)’

qp factor between p and b frames

‘-rc_strategy[:stream_specifier] *integer* (*output,video*)’

ratecontrol method

‘-b_strategy[:stream_specifier] *integer* (*output,video*)’

strategy to choose between I/P/B-frames

‘-ps[:stream_specifier] *integer* (*output,video*)’

rtp payload size in bytes

‘-mv_bits[:stream_specifier] *integer* ()’
‘-header_bits[:stream_specifier] *integer* ()’
‘-i_tex_bits[:stream_specifier] *integer* ()’
‘-p_tex_bits[:stream_specifier] *integer* ()’
‘-i_count[:stream_specifier] *integer* ()’
‘-p_count[:stream_specifier] *integer* ()’
‘-skip_count[:stream_specifier] *integer* ()’
‘-misc_bits[:stream_specifier] *integer* ()’
‘-frame_bits[:stream_specifier] *integer* ()’
‘-codec_tag[:stream_specifier] *integer* ()’
‘-bug[:stream_specifier] *flags* (*input,video*)’

workaround not auto detected encoder bugs

Possible values:

‘`autodetect`’
‘`old_msmpeg4`’

some old lavc generated msmpeg4v3 files (no autodetection)

‘`xvid_ilace`’

Xvid interlacing bug (autodetected if fourcc==XVIX)

‘`ump4`’

(autodetected if fourcc==UMP4)

‘`no_padding`’

padding bug (autodetected)

‘`amv`’
‘`ac_vlc`’

illegal vlc bug (autodetected per fourcc)

‘`qpel_chroma`’
‘`std_qpel`’

old standard qpel (autodetected per fourcc/version)

‘`qpel_chroma2`’
‘`direct_blocksize`’

direct-qpel-blocksize bug (autodetected per fourcc/version)

‘`edge`’

edge padding bug (autodetected per fourcc/version)

‘`hpel_chroma`’
‘`dc_clip`’
‘`ms`’

workaround various bugs in microsofts broken decoders

‘`trunc`’

trancated frames

‘`-lelim[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

single coefficient elimination threshold for luminance (negative values also consider dc coefficient)

‘`-celim[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

single coefficient elimination threshold for chrominance (negative values also consider dc coefficient)

‘`-strict[:stream_specifier] `*`integer`*` (`*`input/output,audio,video`*`)`’

how strictly to follow the standards

Possible values:

‘`very`’

strictly conform to a older more strict version of the spec or reference software

‘`strict`’

strictly conform to all the things in the spec no matter what consequences

‘`normal`’
‘`unofficial`’

allow unofficial extensions

‘`experimental`’

allow non standardized experimental things

‘`-b_qoffset[:stream_specifier] `*`float`*` (`*`output,video`*`)`’

qp offset between P and B frames

‘`-err_detect[:stream_specifier] `*`flags`*` (`*`input,audio,video`*`)`’

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

> detect bitstream specification deviations

‘`buffer`’

> detect improper bitstream length

‘`explode`’

> abort decoding on minor error detection

‘`careful`’

> consider things that violate the spec and have not been seen in the wild as errors

‘`compliant`’

> consider all spec non compliancies as errors

‘`aggressive`’

> consider things that a sane encoder should not do as an error

‘`-has_b_frames[:stream_specifier]` *`integer`* `()`’
‘`-block_align[:stream_specifier]` *`integer`* `()`’
‘`-mpeg_quant[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

use MPEG quantizers instead of H.263

‘`-qsquish[:stream_specifier]` *`float`* `(`*`output,video`*`)`’

how to keep quantizer between qmin and qmax (0 = clip, 1 = use differentiable function)

‘`-rc_qmod_amp[:stream_specifier]` *`float`* `(`*`output,video`*`)`’

experimental quantizer modulation

‘`-rc_qmod_freq[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

experimental quantizer modulation

‘`-rc_override_count[:stream_specifier]` *`integer`* `()`’
‘`-rc_eq[:stream_specifier]` *`string`* `(`*`output,video`*`)`’

Set rate control equation. When computing the expression, besides the standard functions defined in the section ’Expression Evaluation’, the following functions are available: bits2qp(bits), qp2bits(qp). Also the following constants are available: iTex pTex tex mv fCode iCount mcVar var isI isP isB avgQP qComp avgIITex avgPITex avgPPTex avgBPTex avgTex.

‘`-maxrate[:stream_specifier] ` *`integer`* `(`*`output,audio,video`*`)`’

Set max bitrate tolerance (in bits/s). Requires bufsize to be set.

‘`-minrate[:stream_specifier] ` *`integer`* `(`*`output,audio,video`*`)`’

Set min bitrate tolerance (in bits/s). Most useful in setting up a CBR encode. It is of little use elsewise.

‘`-bufsize[:stream_specifier] ` *`integer`* `(`*`output,audio,video`*`)`’

set ratecontrol buffer size (in bits)

‘`-rc_buf_aggressivity[:stream_specifier] ` *`float`* `(`*`output,video`*`)`’

currently useless

‘`-i_qfactor[:stream_specifier] ` *`float`* `(`*`output,video`*`)`’

qp factor between P and I frames

‘`-i_qoffset[:stream_specifier] ` *`float`* `(`*`output,video`*`)`’

qp offset between P and I frames

‘`-rc_init_cplx[:stream_specifier] ` *`float`* `(`*`output,video`*`)`’

initial complexity for 1-pass encoding

‘`-dct[:stream_specifier] ` *`integer`* `(`*`output,video`*`)`’

DCT algorithm

Possible values:

‘`auto`’

autoselect a good one (default)

‘`fastint`’

fast integer

‘`int`’

accurate integer

‘`mmx`’

‘`altivec`’
‘`faan`’

floating point AAN DCT

‘`-lumi_mask[:stream_specifier] *float* (*output,video*)`’

compresses bright areas stronger than medium ones

‘`-tcplx_mask[:stream_specifier] *float* (*output,video*)`’

temporal complexity masking

‘`-scplx_mask[:stream_specifier] *float* (*output,video*)`’

spatial complexity masking

‘`-p_mask[:stream_specifier] *float* (*output,video*)`’

inter masking

‘`-dark_mask[:stream_specifier] *float* (*output,video*)`’

compresses dark areas stronger than medium ones

‘`-idct[:stream_specifier] *integer* (*input/output,video*)`’

select IDCT implementation

Possible values:

‘`auto`’
‘`int`’
‘`simple`’
‘`simplemmx`’
‘`libmpeg2mmx`’
‘`mmi`’
‘`arm`’
‘`altivec`’
‘`sh4`’
‘`simplearm`’
‘`simplearmv5te`’
‘`simplearmv6`’
‘`simpleneon`’
‘`simplealpha`’
‘`h264`’
‘`vp3`’

‘`ipp`’
‘`xvidmmx`’
‘`faani`’

floating point AAN IDCT

‘`-slice_count[:stream_specifier]` *integer* `()`’
‘`-ec[:stream_specifier]` *flags* (*input,video*)’

set error concealment strategy

Possible values:

‘`guess_mvs`’

iterative motion vector (MV) search (slow)

‘`deblock`’

use strong deblock filter for damaged MBs

‘`-bits_per_coded_sample[:stream_specifier]` *integer* `()`’
‘`-pred[:stream_specifier]` *integer* (*output,video*)’

prediction method

Possible values:

‘`left`’
‘`plane`’
‘`median`’
‘`-aspect[:stream_specifier]` *rational number* (*output,video*)’

sample aspect ratio

‘`-debug[:stream_specifier]` *flags* (*input/output,audio,video,subtitles*)’

print specific debug info

Possible values:

‘`pict`’

picture info

‘`rc`’

rate control

‘`bitstream`’
‘`mb_type`’

macroblock (MB) type

‘`qp`’

per-block quantization parameter (QP)

‘`mv`’

motion vector

‘`dct_coeff`’
‘`skip`’
‘`startcode`’
‘`pts`’
‘`er`’

error recognition

‘`mmco`’

memory management control operations (H.264)

‘`bugs`’
‘`vis_qp`’

visualize quantization parameter (QP), lower QP are tinted greener

‘`vis_mb_type`’

visualize block types

‘`buffers`’

picture buffer allocations

‘`thread_ops`’

threading operations

‘`-vismv[:stream_specifier]` *`integer`* `(`*`input,video`*`)`’

visualize motion vectors (MVs)

Possible values:

‘`pf`’

forward predicted MVs of P-frames

‘`bf`’

forward predicted MVs of B-frames

‘`bb`’

backward predicted MVs of B-frames

‘`-cmp[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

full pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

‘`-subcmp[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

sub pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’
‘`-mbcmp[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

macroblock compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

‘`-ildctcmp[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

interlaced dct compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’
‘`-dia_size[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

diamond type & size for motion estimation

‘`-last_pred[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

amount of motion predictors from the previous frame

‘`-preme[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

pre motion estimation

‘`-precmp[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

pre motion estimation compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’
‘`-pre_dia_size[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

diamond type & size for motion estimation pre-pass

‘`-subq[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

sub pel motion estimation quality

‘`-dtg_active_format[:stream_specifier]` *`integer`* `()`’
‘`-me_range[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

limit motion vectors range (1023 for DivX player)

‘`-ibias[:stream_specifier]` *integer* (*output,video*)’

intra quant bias

‘`-pbias[:stream_specifier]` *integer* (*output,video*)’

inter quant bias

‘`-color_table_id[:stream_specifier]` *integer* ()’
‘`-global_quality[:stream_specifier]` *integer* (*output,audio,video*)’
‘`-coder[:stream_specifier]` *integer* (*output,video*)’

Possible values:

‘`vlc`’

variable length coder / huffman coder

‘`ac`’

arithmetic coder

‘`raw`’

raw (no encoding)

‘`rle`’

run-length coder

‘`deflate`’

deflate-based coder

‘`-context[:stream_specifier]` *integer* (*output,video*)’

context model

‘`-slice_flags[:stream_specifier]` *integer* ()’
‘`-xvmc_acceleration[:stream_specifier]` *integer* ()’
‘`-mbd[:stream_specifier]` *integer* (*output,video*)’

macroblock decision algorithm (high quality mode)

Possible values:

‘`simple`’

use mbcmp (default)

‘`bits`’

use fewest bits

‘`rd`’

use best rate distortion

‘`-stream_codec_tag[:stream_specifier]` *integer* `()`’
‘`-sc_threshold[:stream_specifier]` *integer* (*output,video*)’

scene change threshold

‘`-lmin[:stream_specifier]` *integer* (*output,video*)’

min lagrange factor (VBR)

‘`-lmax[:stream_specifier]` *integer* (*output,video*)’

max lagrange factor (VBR)

‘`-nr[:stream_specifier]` *integer* (*output,video*)’

noise reduction

‘`-rc_init_occupancy[:stream_specifier]` *integer* (*output,video*)’

number of bits which should be loaded into the rc buffer before decoding starts

‘`-inter_threshold[:stream_specifier]` *integer* (*output,video*)’
‘`-flags2[:stream_specifier]` *flags* (*input/output,audio,video*)’

Possible values:

‘`fast`’

allow non spec compliant speedup tricks

‘`sgop`’

Deprecated, use mpegvideo private options instead

‘`noout`’

skip bitstream encoding

‘`local_header`’

place global headers at every keyframe instead of in extradata

‘`chunks`’

Frame data might be split into multiple chunks

‘`showall`’

Show all frames before the first keyframe

‘`skiprd`’

Deprecated, use mpegvideo private options instead

‘`-error[:stream_specifier]` *integer* (*output,video*)’
‘`-qns[:stream_specifier]` *integer* (*output,video*)’

deprecated, use mpegvideo private options instead

‘`-threads[:stream_specifier]` *integer* (*input/output,video*)’

Possible values:

‘`auto`’

detect a good number of threads

‘`-me_threshold[:stream_specifier]` *integer* (*output,video*)’

motion estimaton threshold

‘`-mb_threshold[:stream_specifier]` *integer* (*output,video*)’

macroblock threshold

‘`-dc[:stream_specifier]` *integer* (*output,video*)’

intra_dc_precision

‘`-nssew[:stream_specifier]` *integer* (*output,video*)’

nsse weight

‘`-skip_top[:stream_specifier]` *integer* (*input,video*)’

number of macroblock rows at the top which are skipped

‘`-skip_bottom[:stream_specifier]` *integer* (*input,video*)’

number of macroblock rows at the bottom which are skipped

‘`-profile[:stream_specifier]` *integer* (*output,audio,video*)’

Possible values:

‘`unknown`’
‘`aac_main`’
‘`aac_low`’
‘`aac_ssr`’
‘`aac_ltp`’
‘`aac_he`’
‘`aac_he_v2`’
‘`aac_ld`’
‘`aac_eld`’
‘`dts`’
‘`dts_es`’
‘`dts_96_24`’
‘`dts_hd_hra`’
‘`dts_hd_ma`’

‘`-level[:stream_specifier]` *integer* (*output,audio,video*)’

Possible values:

‘`unknown`’

‘`-lowres[:stream_specifier]` *integer* (*input,audio,video*)’

decode at 1= 1/2, 2=1/4, 3=1/8 resolutions

‘`-skip_threshold[:stream_specifier]` *integer* (*output,video*)’

frame skip threshold

‘`-skip_factor[:stream_specifier]` *integer* (*output,video*)’

frame skip factor

‘`-skip_exp[:stream_specifier]` *integer* (*output,video*)’

frame skip exponent

‘`-skipcmp[:stream_specifier]` *integer* (*output,video*)’

frame skip compare function

Possible values:

‘`sad`’

: sum of absolute differences, fast (default)

‘`sse`’

: sum of squared errors

‘`satd`’

: sum of absolute Hadamard transformed differences

‘`dct`’

: sum of absolute DCT transformed differences

‘`psnr`’

: sum of squared quantization errors (avoid, low quality)

‘`bit`’

: number of bits needed for the block

‘`rd`’

: rate distortion optimal, slow

‘`zero`’

: 0

‘`vsad`’

: sum of absolute vertical differences

‘`vsse`’

: sum of squared vertical differences

‘`nsse`’

: noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

‘`-border_mask[:stream_specifier] `*`float`*` (`*`output,video`*`)`’

increases the quantizer for macroblocks close to borders

‘`-mblmin[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

min macroblock lagrange factor (VBR)

‘`-mblmax[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

max macroblock lagrange factor (VBR)

‘`-mepc[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

motion estimation bitrate penalty compensation (1.0 = 256)

‘`-skip_loop_filter[:stream_specifier] `*`integer`*` (`*`input,video`*`)`’

Possible values:

‘`none`’
‘`default`’
‘`noref`’
‘`bidir`’
‘`nokey`’
‘`all`’

‘`-skip_idct[:stream_specifier] `*`integer`*` (`*`input,video`*`)`’

Possible values:

‘`none`’
‘`default`’
‘`noref`’
‘`bidir`’
‘`nokey`’
‘`all`’

‘`-skip_frame[:stream_specifier] `*`integer`*` (`*`input,video`*`)`’

Possible values:

‘none’
‘default’
‘noref’
‘bidir’
‘nokey’
‘all’

‘-bidir_refine[:stream_specifier] *integer* (*output,video*)’

refine the two motion vectors used in bidirectional macroblocks

‘-brd_scale[:stream_specifier] *integer* (*output,video*)’

downscales frames for dynamic B-frame decision

‘-keyint_min[:stream_specifier] *integer* (*output,video*)’

minimum interval between IDR-frames

‘-refs[:stream_specifier] *integer* (*output,video*)’

reference frames to consider for motion compensation

‘-chromaoffset[:stream_specifier] *integer* (*output,video*)’

chroma qp offset from luma

‘-trellis[:stream_specifier] *integer* (*output,audio,video*)’

rate-distortion optimal quantization

‘-sc_factor[:stream_specifier] *integer* (*output,video*)’

multiplied by qscale for each frame and added to scene_change_score

‘-mv0_threshold[:stream_specifier] *integer* (*output,video*)’
‘-b_sensitivity[:stream_specifier] *integer* (*output,video*)’

adjusts sensitivity of b_frame_strategy 1

‘-compression_level[:stream_specifier] *integer* (*output,audio,video*)’
‘-min_prediction_order[:stream_specifier] *integer* (*output,audio*)’
‘-max_prediction_order[:stream_specifier] *integer* (*output,audio*)’
‘-timecode_frame_start[:stream_specifier] *integer* (*output,video*)’

GOP timecode frame start number, in non drop frame format

‘-request_channels[:stream_specifier] *integer* (*input,audio*)’ set desired number of audio channels

‘`-bits_per_raw_sample[:stream_specifier]` *integer* `()`’
‘`-channel_layout[:stream_specifier]` *integer* `(`*input/output,audio*`)`’

Possible values:

‘`-request_channel_layout[:stream_specifier]` *integer* `(`*input,audio*`)`’

Possible values:

‘`-rc_max_vbv_use[:stream_specifier]` *float* `(`*output,video*`)`’
‘`-rc_min_vbv_use[:stream_specifier]` *float* `(`*output,video*`)`’
‘`-ticks_per_frame[:stream_specifier]` *integer* `(`*input/output,audio,video*`)`’
‘`-color_primaries[:stream_specifier]` *integer* `(`*input/output,video*`)`’
‘`-color_trc[:stream_specifier]` *integer* `(`*input/output,video*`)`’
‘`-colorspace[:stream_specifier]` *integer* `(`*input/output,video*`)`’
‘`-color_range[:stream_specifier]` *integer* `(`*input/output,video*`)`’
‘`-chroma_sample_location[:stream_specifier]` *integer* `(`*input/output,video*`)`’
‘`-log_level_offset[:stream_specifier]` *integer* `()`’

set the log level offset

‘`-slices[:stream_specifier]` *integer* `(`*output,video*`)`’

number of slices, used in parallelized encoding

‘`-thread_type[:stream_specifier]` *flags* `(`*input/output,video*`)`’

select multithreading type

Possible values:

‘`slice`’
‘`frame`’
‘`-audio_service_type[:stream_specifier]` *integer* `(`*output,audio*`)`’

audio service type

Possible values:

‘`ma`’

Main Audio Service

‘`ef`’

Effects

‘`vi`’

Visually Impaired

‘`hi`’

Hearing Impaired

‘`di`’

Dialogue

‘`co`’

Commentary

‘`em`’

Emergency

‘`vo`’

Voice Over

‘`ka`’

Karaoke

‘`-request_sample_fmt[:stream_specifier]` *`value`* `(`*`input,audio`*`)`’

sample format audio decoders should prefer

Possible values:

‘`-pkt_timebase[:stream_specifier]` *`rational number`* `()`’

5.5 Format AVOptions

‘`-avioflags` *`flags`* `(`*`input/output`*`)`’

Possible values:

‘`direct`’

reduce buffering

‘`-probesize` *`integer`* `(`*`input`*`)`’

set probing size

‘`-packetsize` *`integer`* `(`*`output`*`)`’

set packet size

‘`-fflags` *`flags`* `(`*`input/output`*`)`’

Possible values:

‘`ignidx`’

ignore index

‘`genpts`’

generate pts

‘`nofillin`’

do not fill in missing values that can be exactly calculated

‘`noparse`’

disable AVParsers, this needs nofillin too

‘`igndts`’

ignore dts

‘`discardcorrupt`’

discard corrupted frames

‘`sortdts`’

try to interleave outputted packets by dts

‘`keepside`’

dont merge side data

‘`latm`’

enable RTP MP4A-LATM payload

‘`nobuffer`’

reduce the latency introduced by optional buffering

‘`-analyzeduration` *`integer`* `(`*`input`*`)`’

how many microseconds are analyzed to estimate duration

‘`-cryptokey` *`hexadecimal string`* `(`*`input`*`)`’

decryption key

‘`-indexmem` *`integer`* `(`*`input`*`)`’

max memory used for timestamp index (per stream)

‘`-rtbufsize` *`integer`* `(`*`input`*`)`’

max memory used for buffering real-time frames

‘`-fdebug` *`flags`* `(`*`input/output`*`)`’

print specific debug info

Possible values:

‘`ts`’

‘`-max_delay` *`integer`* `(`*`input/output`*`)`’

maximum muxing or demuxing delay in microseconds

‘`-fpsprobesize` *`integer`* `(`*`input`*`)`’

number of frames used to probe fps

‘`-audio_preload` *`integer`* `(`*`output`*`)`’

microseconds by which audio packets should be interleaved earlier

‘`-chunk_duration` *`integer`* `(`*`output`*`)`’

microseconds for each chunk

‘`-chunk_size` *`integer`* `(`*`output`*`)`’

size in bytes for each chunk

‘`-f_err_detect` *`flags`* `(`*`input`*`)`’

set error detection flags (deprecated; use err_detect, save via avconv)

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

consider things that violate the spec and have not been seen in the wild as errors

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

‘`-err_detect` *`flags`* `(`*`input`*`)`’

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

consider things that violate the spec and have not been seen in the wild as errors

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

‘`-use_wallclock_as_timestamps` *`integer`* `(`*`input`*`)`’

use wallclock as timestamps

‘`-avoid_negative_ts` *`integer`* `(`*`output`*`)`’

avoid negative timestamps

‘`-skip_initial_bytes` *`integer`* `(`*`input`*`)`’

skip initial bytes

5.6 Main options

‘`-f` *`fmt`* `(`*`input/output`*`)`’

Force input or output file format. The format is normally auto detected for input files and guessed from file extension for output files, so this option is not needed in most cases.

‘`-i` *`filename`* `(`*`input`*`)`’

input file name

‘`-y (`*`global`*`)`’

Overwrite output files without asking.

‘`-n (`*`global`*`)`’

Do not overwrite output files but exit if file exists.

‘`-c[:`*`stream_specifier`*`]` *`codec`* `(`*`input/output,per-stream`*`)`’
‘`-codec[:`*`stream_specifier`*`]` *`codec`* `(`*`input/output,per-stream`*`)`’

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

'`-t` *duration* (*output*)'

Stop writing the output after its duration reaches *duration*. *duration* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

'`-fs` *limit_size* (*output*)'

Set the file size limit, expressed in bytes.

'`-ss` *position* (*input/output*)'

When used as an input option (before `-i`), seeks in this input file to *position*. When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*. This is slower, but more accurate.

position may be either in seconds or in `hh:mm:ss[.xxx]` form.

'`-itsoffset` *offset* (*input*)'

Set the input time offset in seconds. `[-]hh:mm:ss[.xxx]` syntax is also supported. The offset is added to the timestamps of the input files. Specifying a positive offset means that the corresponding streams are delayed by *offset* seconds.

'`-timestamp` *time* (*output*)'

Set the recording timestamp in the container. The syntax for *time* is:

```
now|([(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...])|(HHMMSS[.m...]))[Z|z])
```

If the value is "now" it takes the current time. Time is local time unless 'Z' or 'z' is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

'`-metadata[:metadata_specifier]` *key*=*value* (*output,per-metadata*)'

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams or chapters. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

```
ffmpeg -i in.avi -metadata title="my title" out.flv
```

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:1 language=eng OUTPUT
```

'`-target` *type* (*output*)'

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

```
ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
```

Nevertheless you can specify additional options as long as you know they do not conflict with the standard, as in:

```
ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
```

'`-dframes` *number* (*output*)'

Set the number of data frames to record. This is an alias for `-frames:d`.

'`-frames[:`*stream_specifier*`]` *framecount* (*output,per-stream*)'

Stop writing to the stream after *framecount* frames.

'`-q[:`*stream_specifier*`]` *q* (*output,per-stream*)'

‘`-qscale[:`*`stream_specifier`*`]` *`q`* `(`*`output,per-stream`*`)`’

Use fixed quality scale (VBR). The meaning of *q* is codec-dependent.

‘`-filter[:`*`stream_specifier`*`]` *`filter_graph`* `(`*`output,per-stream`*`)`’

filter_graph is a description of the filter graph to apply to the stream. Use `-filters` to show all the available filters (including also sources and sinks).

See also the ‘`-filter_complex`’ option if you want to create filter graphs with multiple inputs and/or outputs.

‘`-pre[:`*`stream_specifier`*`]` *`preset_name`* `(`*`output,per-stream`*`)`’

Specify the preset for matching stream(s).

‘`-stats (`*`global`*`)`’

Print encoding progress/statistics. On by default.

‘`-progress` *`url`* `(`*`global`*`)`’

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key*=*value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

‘`-stdin`’

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

‘`-debug_ts (`*`global`*`)`’

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option `-fdebug ts`.

‘`-attach` *`filename`* `(`*`output`*`)`’

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

‘`-dump_attachment[:`*`stream_specifier`*`]` *`filename`* `(`*`input,per-stream`*`)`’

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the `filename` metadata tag will be used.

E.g. to extract the first attachment to a file named ’out.ttf’:

```
ffmpeg -dump_attachment:t:0 out.ttf INPUT
```

To extract all attachments to files determined by the `filename` tag:

```
ffmpeg -dump_attachment:t "" INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

5.7 Video Options

‘`-vframes` *`number`* `(`*`output`*`)`’

Set the number of video frames to record. This is an alias for `-frames:v`.

‘`-r[:`*`stream_specifier`*`]` *`fps`* `(`*`input/output,per-stream`*`)`’

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps* (note that this actually causes the `fps` filter to be inserted to the end of the corresponding filtergraph).

‘`-s[:`*`stream_specifier`*`]` *`size`* `(`*`input/output,per-stream`*`)`’

Set frame size.

As an input option, this is a shortcut for the '`video_size`' private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the `scale` video filter to the *end* of the corresponding filtergraph. Please use the `scale` filter directly to insert it at the beginning or some other place.

The format is '`wxh`' (default - same as source).

'`-aspect[:`*`stream_specifier`*`]` *`aspect`* `(`*`output,per-stream`*`)`'

Set the video display aspect ratio specified by *aspect*.

aspect can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

'`-croptop` *`size`*'
'`-cropbottom` *`size`*'
'`-cropleft` *`size`*'
'`-cropright` *`size`*'

All the crop options have been removed. Use -vf crop=width:height:x:y instead.

'`-padtop` *`size`*'
'`-padbottom` *`size`*'
'`-padleft` *`size`*'
'`-padright` *`size`*'
'`-padcolor` *`hex_color`*'

All the pad options have been removed. Use -vf pad=width:height:x:y:color instead.

'`-vn (`*`output`*`)`'

Disable video recording.

'`-vcodec` *`codec`* `(`*`output`*`)`'

Set the video codec. This is an alias for `-codec:v`.

'`-pass[:`*`stream_specifier`*`]` *`n`* `(`*`output,per-stream`*`)`'

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

```
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
```

‘`-passlogfile[:`*`stream_specifier`*`]` *`prefix`* `(`*`output,per-stream`*`)`’

Set two-pass log file name prefix to *prefix*, the default file name prefix is “ffmpeg2pass”. The complete file name will be ‘`PREFIX-N.log`’, where N is a number specific to the output stream

‘`-vlang` *`code`*’

Set the ISO 639 language code (3 letters) of the current video stream.

‘`-vf` *`filter_graph`* `(`*`output`*`)`’

filter_graph is a description of the filter graph to apply to the input video. Use the option "-filters" to show all the available filters (including also sources and sinks). This is an alias for `-filter:v`.

5.8 Advanced Video Options

‘`-pix_fmt[:`*`stream_specifier`*`]` *`format`* `(`*`input/output,per-stream`*`)`’

Set pixel format. Use `-pix_fmts` to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a `+`, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions inside filter graphs are disabled. If *pix_fmt* is a single `+`, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

‘`-sws_flags` *`flags`* `(`*`input/output`*`)`’

Set SwScaler flags.

‘`-vdt` *`n`*’

Discard threshold.

‘`-rc_override[:`*`stream_specifier`*`]` *`override`* `(`*`output,per-stream`*`)`’

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

‘`-deinterlace`’

Deinterlace pictures. This option is deprecated since the deinterlacing is very low quality. Use the yadif filter with `-filter:v yadif`.

‘`-ilme`’

Force interlacing support in encoder (MPEG-2 and MPEG-4 only). Use this option if your input file is interlaced and you want to keep the interlaced format for minimum losses. The alternative is to deinterlace the input stream with ‘`-deinterlace`’, but deinterlacing introduces losses.

‘`-psnr`’

Calculate PSNR of compressed frames.

‘`-vstats`’

Dump video coding statistics to ‘`vstats_HHMMSS.log`’.

‘`-vstats_file` *file*’

Dump video coding statistics to *file*.

‘`-top[:`*stream_specifier*`]` *n* `(`*output,per-stream*`)`’

top=1/bottom=0/auto=-1 field first

‘`-dc` *precision*’

Intra_dc_precision.

‘`-vtag` *fourcc/tag* `(`*output*`)`’

Force video tag/fourcc. This is an alias for `-tag:v`.

‘`-qphist (`*global*`)`’

Show QP histogram

‘`-vbsf` *bitstream_filter*’

Deprecated see -bsf

‘`-force_key_frames[:`*stream_specifier*`]` *time*`[,`*time*`...]` `(`*output,per-stream*`)`’

Force key frames at the specified timestamps, more precisely at the first frames after each specified time. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file. The timestamps must be specified in ascending order.

‘`-copyinkf[:`*stream_specifier*`]` `(`*output,per-stream*`)`’

When doing stream copy, copy also non-key frames found at the beginning.

5.9 Audio Options

‘`-aframes` *`number`* `(`*`output`*`)`’

> Set the number of audio frames to record. This is an alias for `-frames:a`.

‘`-ar[:`*`stream_specifier`*`]` *`freq`* `(`*`input/output,per-stream`*`)`’

> Set the audio sampling frequency. For output streams it is set by default to the frequency of the corresponding input stream. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

‘`-aq` *`q`* `(`*`output`*`)`’

> Set the audio quality (codec-specific, VBR). This is an alias for -q:a.

‘`-ac[:`*`stream_specifier`*`]` *`channels`* `(`*`input/output,per-stream`*`)`’

> Set the number of audio channels. For output streams it is set by default to the number of input audio channels. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

‘`-an (`*`output`*`)`’

> Disable audio recording.

‘`-acodec` *`codec`* `(`*`input/output`*`)`’

> Set the audio codec. This is an alias for `-codec:a`.

‘`-sample_fmt[:`*`stream_specifier`*`]` *`sample_fmt`* `(`*`output,per-stream`*`)`’

> Set the audio sample format. Use `-sample_fmts` to get a list of supported sample formats.

‘`-af` *`filter_graph`* `(`*`output`*`)`’

> *filter_graph* is a description of the filter graph to apply to the input audio. Use the option "-filters" to show all the available filters (including also sources and sinks). This is an alias for `-filter:a`.

5.10 Advanced Audio options:

‘`-atag` *`fourcc/tag`* `(`*`output`*`)`’

> Force audio tag/fourcc. This is an alias for `-tag:a`.

‘`-absf` *`bitstream_filter`*’

Deprecated, see -bsf

5.11 Subtitle options:

‘`-slang` *`code`*’

Set the ISO 639 language code (3 letters) of the current subtitle stream.

‘`-scodec` *`codec`* `(`*`input/output`*`)`’

Set the subtitle codec. This is an alias for `-codec:s`.

‘`-sn (`*`output`*`)`’

Disable subtitle recording.

‘`-sbsf` *`bitstream_filter`*’

Deprecated, see -bsf

5.12 Advanced Subtitle options:

‘`-fix_sub_duration`’

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

5.13 Advanced options

‘`-map [-]`*`input_file_id`*`[:`*`stream_specifier`*`][,`*`sync_file_id`*`[:`*`stream_specifier`*`]] |` *`[linklabel]`* `(`*`output`*`)`’

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A `-` character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the '`-filter_complex`' option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in '`INPUT`' identified by "0:1" to the (single) output stream in '`out.wav`'.

For example, to select the stream with index 2 from input file '`a.mov`' (specified by the identifier "0:2"), and stream with index 6 from input '`b.mov`' (specified by the identifier "1:6"), and copy them to the output file '`out.mov`':

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

```
ffmpeg -i INPUT -map 0:v -map 0:a:2 OUTPUT
```

To map all the streams except the second audio, use negative mappings

```
ffmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT
```

Note that using this option disables the default mappings for this output file.

'`-map_channel [`*`input_file_id.stream_specifier.channel_id`*`|-1][:`*`output_file_id.stream_specifier`*`]`'

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don’t match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can’t for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here ‘`input.mkv`’) with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

‘`-map_metadata[:`*`metadata_spec_out`*`]` *`infile`*`[:`*`metadata_spec_in`*`] (`*`output,per-metadata`*`)`’

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

‘*`g`*’

global metadata, i.e. metadata that applies to the whole file

‘`s[:`*`stream_spec`*`]`’

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

‘`c:`*`chapter_index`*’

per-chapter metadata. *chapter_index* is the zero-based chapter index.

‘`p:`*`program_index`*’

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

```
ffmpeg -i in.ogg -map_metadata 0:s:0 out.mp3
```

To do the reverse, i.e. copy global metadata to all audio streams:

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

‘`-map_chapters `*`input_file_index`*` (`*`output`*`)`’

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

‘`-benchmark (`*`global`*`)`’

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

‘`-benchmark_all (`*`global`*`)`’

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

‘`-timelimit `*`duration`*` (`*`global`*`)`’

Exit after ffmpeg has been running for *duration* seconds.

‘`-dump (`*`global`*`)`’

Dump each input packet to stderr.

‘`-hex (`*`global`*`)`’

When dumping packets, also dump the payload.

‘`-re (`*`input`*`)`’

Read input at native frame rate. Mainly used to simulate a grab device. By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming). If your input(s) is coming from some other live streaming source (through HTTP or UDP for example) the server might already be in real-time, thus the option will likely not be required. On the other hand, this is meaningful if your input(s) is a file you are trying to push in real-time.

‘`-loop_input`’

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

‘`-loop_output `*`number_of_times`*’

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

‘`-vsync `*`parameter`*’

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

‘`0, passthrough`’

Each frame is passed with its timestamp from the demuxer to the muxer.

‘`1, cfr`’

Frames will be duplicated and dropped to achieve exactly the requested constant framerate.

‘`2, vfr`’

Frames are passed through with their timestamp or dropped so as to prevent 2 frames from having the same timestamp.

‘`drop`’

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

‘`-1, auto`’

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

‘`-async` *`samples_per_second`*’

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction. This option has been deprecated. Use the `asyncts` audio filter instead.

‘`-copyts`’

Copy timestamps from input to output.

‘`-copytb` *`mode`*’

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

‘`1`’

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid non monotonically increasing timestamps when copying video streams with variable frame rate.

‘`0`’

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

‘`-1`’

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

‘`-shortest (`*`output`*`)`’

Finish encoding when the shortest input stream ends.

‘`-dts_delta_threshold`’

Timestamp discontinuity delta threshold.

‘`-muxdelay` *`seconds`* `(`*`input`*`)`’

Set the maximum demux-decode delay.

‘`-muxpreload` *`seconds`* `(`*`input`*`)`’

Set the initial demux-decode delay.

‘`-streamid` *`output-stream-index`*`:`*`new-value`* `(`*`output`*`)`’

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

```
ffmpeg -i infile -streamid 0:33 -streamid 1:36 out.ts
```

‘`-bsf[:`*`stream_specifier`*`]` *`bitstream_filters`* `(`*`output,per-stream`*`)`’

Set bitstream filters for matching streams. *bistream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

```
ffmpeg -i h264.mp4 -c:v copy -bsf:v h264_mp4toannexb -an out.h264
```

```
ffmpeg -i file.mov -an -vn -bsf:s mov2textsub -c:s copy -f rawvideo sub.txt
```

‘`-tag[:`*`stream_specifier`*`]` *`codec_tag`* `(`*`per-stream`*`)`’

Force a tag/fourcc for matching streams.

‘-timecode *hh*:*mm*:*ss*SEP*ff*’

Specify Timecode for writing. *SEP* is ’:’ for non drop timecode and ’;’ (or ’.’) for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

‘-filter_complex *filtergraph* (*global*)’

Define a complex filter graph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the ‘-filter’ options. *filtergraph* is a description of the filter graph, as described in Filtergraph syntax.

Input link labels must refer to input streams using the `[file_index:stream_specifier]` syntax (i.e. the same as ‘-map’ uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with ‘-map’. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex ’[0:v][1:v]overlay[out]’ -map
’[out]’ out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex ’overlay[out]’ -map
’[out]’ out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

```
ffmpeg -i video.mkv -i image.png -filter_complex ’overlay’ out.mkv
```

To generate 5 seconds of pure red video using lavfi `color` source:

```
ffmpeg -filter_complex ’color=red’ -t 5 out.mkv
```

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720Ã576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

5.14 Preset files

A preset file contains a sequence of *option*=*value* pairs, one for each line, specifying a sequence of options which would be awkward to specify on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `vpre`, `apre`, `spre`, and `fpre` options. The `fpre` option takes the filename of the preset instead of a preset name as input and can be used for any kind of codec. For the `vpre`, `apre`, and `spre` options, the options specified in a preset file are applied to the currently selected codec of the same type as the preset option.

The argument passed to the `vpre`, `apre`, and `spre` preset options identifies the preset file to use according to the following rules:

First ffmpeg searches for a file named *arg*.ffpreset in the directories '`$FFMPEG_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the datadir defined at configuration time (usually '`PREFIX/share/ffmpeg`') or in a '`ffpresets`' folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file '`libvpx-1080p.ffpreset`'.

If no such file is found, then ffmpeg will search for a file named *codec_name*-*arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file '`libvpx-1080p.ffpreset`'.

6. Tips

- For streaming at very low bitrate application, use a low frame rate and a small GOP size. This is especially true for RealVideo where the Linux player does not seem to be very fast, so it can miss frames. An example is:

```
        ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
```

- The parameter ’q’ which is displayed while encoding is the current quantizer. The value 1 indicates that a very good quality could be achieved. The value 31 indicates the worst quality. If q=31 appears too often, it means that the encoder cannot compress enough to meet your bitrate. You must either increase the bitrate, decrease the frame rate or decrease the frame size.
- If your computer is not fast enough, you can speed up the compression at the expense of the compression ratio. You can use ’-me zero’ to speed up motion estimation, and ’-g 0’ to disable motion estimation completely (you have only I-frames, which means it is about as good as JPEG compression).
- To have very low audio bitrates, reduce the sampling frequency (down to 22050 Hz for MPEG audio, 22050 or 11025 for AC-3).
- To have a constant quality (but a variable bitrate), use the option ’-qscale n’ when ’n’ is between 1 (excellent quality) and 31 (worst quality).

7. Examples

7.1 Preset files

A preset file contains a sequence of *option*=*value* pairs, one for each line, specifying a sequence of options which can be specified also on the command line. Lines starting with the hash (’#’) character are ignored and are used to provide comments. Empty lines are also ignored. Check the ‘`presets`’ directory in the FFmpeg source tree for examples.

Preset files are specified with the `pre` option, this option takes a preset name as input. FFmpeg searches for a file named *preset_name*.avpreset in the directories ‘`$AVCONV_DATADIR`’ (if set), and ‘`$HOME/.ffmpeg`’, and in the data directory defined at configuration time (usually ‘`$PREFIX/share/ffmpeg`’) in that order. For example, if the argument is `libx264-max`, it will search for the file ‘`libx264-max.avpreset`’.

7.2 Video and Audio grabbing

If you specify the input format and device then ffmpeg can grab video and audio directly.

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing

Grab the X11 display with ffmpeg via

```
ffmpeg -f x11grab -s cif -r 25 -i :0.0 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable.

```
ffmpeg -f x11grab -s cif -r 25 -i :0.0+10,20 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable. 10 is the x-offset and 20 the y-offset for the grabbing.

7.4 Video and Audio file format conversion

Any supported file format and protocol can serve as input to ffmpeg:

Examples:

- You can use YUV files as input:

  ```
  ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
  ```

  It will use the files:

  ```
  /tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
  /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
  ```

  The Y files use twice the resolution of the U and V files. They are raw files, without header. They can be generated by all decent video decoders. You must specify the size of the image with the ‘`-s`’ option if ffmpeg cannot guess it.

- You can input from a raw YUV420P file:

  ```
  ffmpeg -i /tmp/test.yuv /tmp/out.avi
  ```

  test.yuv is a file containing raw YUV planar data. Each frame is composed of the Y plane followed by the U and V planes at half vertical and horizontal resolution.

- You can output to a raw YUV420P file:

  ```
  ffmpeg -i mydivx.avi hugefile.yuv
  ```

- You can set several input files and output files:

  ```
  ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
  ```

  Converts the audio file a.wav and the raw YUV video file a.yuv to MPEG file a.mpg.

- You can also do audio and video conversions at the same time:

  ```
  ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
  ```

  Converts a.wav to MPEG audio at 22050 Hz sample rate.

- You can encode to several formats at the same time and define a mapping from input stream to output streams:

  ```
  ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
  ```

  Converts a.wav to a.mp2 at 64 kbits and to b.mp2 at 128 kbits. '-map file:index' specifies which input stream is used for each output stream, in the order of the definition of output streams.

- You can transcode decrypted VOBs:

  ```
  ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
  ```

  This is a typical DVD ripping example; the input is a VOB file, the output an AVI file with MPEG-4 video and MP3 audio. Note that in this command we use B-frames so the MPEG-4 stream is DivX5 compatible, and GOP size is 300 which means one intra frame every 10 seconds for 29.97fps input video. Furthermore, the audio stream is MP3-encoded so you need to enable LAME support by passing `--enable-libmp3lame` to configure. The mapping is particularly useful for DVD transcoding to get the desired audio language.

  NOTE: To see the supported input formats, use `ffmpeg -formats`.

- You can extract images from a video, or create a video from many images:

  For extracting images from a video:

  ```
  ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
  ```

  This will extract one video frame per second from the video and will output them in files named '`foo-001.jpeg`', '`foo-002.jpeg`', etc. Images will be rescaled to fit the new WxH values.

  If you want to extract just a limited number of frames, you can use the above command in combination with the -vframes or -t option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

```
ffmpeg -f image2 -i foo-%03d.jpeg -r 12 -s WxH foo.avi
```

The syntax `foo-%03d.jpeg` specifies to use a decimal number composed of three digits padded with zeroes to express the sequence number. It is the same syntax supported by the C printf function, but only formats accepting a normal integer are suitable.

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -i 'foo-*.jpeg' -r 12 -s WxH foo.avi
```

- You can put many streams of the same type in the output:

  ```
  ffmpeg -i test1.avi -i test2.avi -map 0.3 -map 0.2 -map 0.1 -map 0.0 -c copy test12.nut
  ```

  The resulting output file ‘`test12.avi`’ will contain first four streams from the input file in reverse order.

- To force CBR video output:

  ```
  ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
  ```

- The four options lmin, lmax, mblmin and mblmax use ’lambda’ units, but you may use the QP2LAMBDA constant to easily convert from ’q’ units:

  ```
  ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
  ```

8. Syntax

This section documents the syntax and formats employed by the FFmpeg libraries and tools.

8.1 Quoting and escaping

FFmpeg adopts the following quoting and escaping mechanism, unless explicitly specified. The following rules are applied:

- `'` and `\` are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.

- A special character is escaped by prefixing it with a '\'.
- All characters enclosed between '' are included literally in the parsed string. The quote character `'` itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

Note that you may need to add a second level of escaping when using the command line or a script, which depends on the syntax of the adopted shell language.

The function `av_get_token` defined in '`libavutil/avstring.h`' can be used to parse a token quoted or escaped according to the rules defined above.

The tool '`tools/ffescape`' in the FFmpeg source tree can be used to automatically quote or escape a string in a script.

8.1.1 Examples

- Escape the string `Crime d'Amour` containing the `'` special character:

  ```
  Crime d\'Amour
  ```

- The string above contains a quote, so the `'` needs to be escaped when quoting it:

  ```
  'Crime d'\''Amour'
  ```

- Include leading or trailing whitespaces using quoting:

  ```
  '  this string starts and ends with whitespaces  '
  ```

- Escaping and quoting can be mixed together:

  ```
  ' The string '\'string\'' is a string '
  ```

- To include a literal `\` you can use either escaping or quoting:

  ```
  'c:\foo' can be written as c:\\foo
  ```

8.2 Date

The accepted syntax is:

```
[(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...]]])|(HHMMSS[.m...]]]))[Z]
now
```

If the value is "now" it takes the current time.

Time is local time unless Z is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

8.3 Time duration

The accepted syntax is:

```
[-]HH:MM:SS[.m...]
[-]S+[.m...]
```

HH expresses the number of hours, *MM* the number a of minutes and *SS* the number of seconds.

8.4 Video size

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation.

The following abbreviations are recognized:

‘`sqcif`’

128x96

‘`qcif`’

176x144

‘`cif`’

352x288

‘`4cif`’

704x576

‘`16cif`’

1408x1152

‘`qqvga`’

160x120

‘`qvga`’

320x240

‘`vga`’

640x480

‘`svga`’

800x600

‘`xga`’

1024x768

‘`uxga`’

1600x1200

‘`qxga`’

2048x1536

‘`sxga`’

1280x1024

‘`qsxga`’

2560x2048

‘`hsxga`’

5120x4096

‘`wvga`’

852x480

‘`wxga`’

1366x768

‘`wsxga`’

1600x1024

‘`wuxga`’

	1920x1200
‘`woxga`’	2560x1600
‘`wqsxga`’	3200x2048
‘`wquxga`’	3840x2400
‘`whsxga`’	6400x4096
‘`whuxga`’	7680x4800
‘`cga`’	320x200
‘`ega`’	640x350
‘`hd480`’	852x480
‘`hd720`’	1280x720
‘`hd1080`’	1920x1080

8.5 Video rate

Specify the frame rate of a video, expressed as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation.

The following abbreviations are recognized:

‘`ntsc`’

30000/1001

‘`pal`’

25/1

‘`qntsc`’

30000/1

‘`qpal`’

25/1

‘`sntsc`’

30000/1

‘`spal`’

25/1

‘`film`’

24/1

‘`ntsc-film`’

24000/1

8.6 Ratio

A ratio can be expressed as an expression, or in the form *numerator*:*denominator*.

Note that a ratio with infinite (1/0) or negative value is considered valid, so you should check on the returned value if you want to exclude those values.

The undefined value can be expressed using the "0:0" string.

8.7 Color

It can be the name of a color (case insensitive match) or a [0x|#]RRGGBB[AA] sequence, possibly followed by "@" and a string representing the alpha component.

The alpha component may be a string composed by "0x" followed by an hexadecimal number or a decimal number between 0.0 and 1.0, which represents the opacity value (0x00/0.0 means completely transparent, 0xff/1.0 completely opaque). If the alpha component is not specified then 0xff is assumed.

The string "random" will result in a random color.

9. Expression Evaluation

When evaluating an arithmetic expression, FFmpeg uses an internal formula evaluator, implemented through the ‘`libavutil/eval.h`’ interface.

An expression may contain unary, binary operators, constants, and functions.

Two expressions *expr1* and *expr2* can be combined to form another expression "*expr1*;*expr2*". *expr1* and *expr2* are evaluated in turn, and the new expression evaluates to the value of *expr2*.

The following binary operators are available: `+`, `-`, `*`, `/`, `^`.

The following unary operators are available: `+`, `-`.

The following functions are available:

‘`sinh(x)`’

Compute hyperbolic sine of *x*.

‘`cosh(x)`’

Compute hyperbolic cosine of *x*.

‘`tanh(x)`’

Compute hyperbolic tangent of *x*.

‘`sin(x)`’

Compute sine of *x*.

‘`cos(x)`’

Compute cosine of *x*.

‘`tan(x)`’

Compute tangent of *x*.

‘`atan(x)`’

Compute arctangent of *x*.

‘`asin(x)`’

Compute arcsine of *x*.

‘`acos(x)`’

Compute arccosine of *x*.

‘`exp(x)`’

Compute exponential of *x* (with base `e`, the Euler’s number).

‘`log(x)`’

Compute natural logarithm of *x*.

‘`abs(x)`’

Compute absolute value of *x*.

‘`squish(x)`’

Compute expression `1/(1 + exp(4*x))`.

‘`gauss(x)`’

Compute Gauss function of *x*, corresponding to `exp(-x*x/2) / sqrt(2*PI)`.

‘`isinf(x)`’

Return 1.0 if *x* is +/-INFINITY, 0.0 otherwise.

‘`isnan(x)`’

Return 1.0 if *x* is NAN, 0.0 otherwise.

‘`mod(x, y)`’

Compute the remainder of division of *x* by *y*.

‘`max(x, y)`’

Return the maximum between *x* and *y*.

‘`min(x, y)`’

Return the maximum between *x* and *y*.

‘`eq(x, y)`’

Return 1 if *x* and *y* are equivalent, 0 otherwise.

‘`gte(x, y)`’

Return 1 if *x* is greater than or equal to *y*, 0 otherwise.

‘`gt(x, y)`’

Return 1 if *x* is greater than *y*, 0 otherwise.

‘`lte(x, y)`’

Return 1 if *x* is lesser than or equal to *y*, 0 otherwise.

‘`lt(x, y)`’

Return 1 if *x* is lesser than *y*, 0 otherwise.

‘`st(var, expr)`’

Allow to store the value of the expression *expr* in an internal variable. *var* specifies the number of the variable where to store the value, and it is a value ranging from 0 to 9. The function returns the value stored in the internal variable. Note, Variables are currently not shared between expressions.

‘`ld(var)`’

Allow to load the value of the internal variable with number *var*, which was previously stored with st(*var*, *expr*). The function returns the loaded value.

‘`while(cond, expr)`’

Evaluate expression *expr* while the expression *cond* is non-zero, and returns the value of the last *expr* evaluation, or NAN if *cond* was always false.

‘`ceil(expr)`’

Round the value of expression *expr* upwards to the nearest integer. For example, "ceil(1.5)" is "2.0".

‘`floor(expr)`’

Round the value of expression *expr* downwards to the nearest integer. For example, "floor(-1.5)" is "-2.0".

‘`trunc(expr)`’

Round the value of expression *expr* towards zero to the nearest integer. For example, "trunc(-1.5)" is "-1.0".

‘`sqrt(expr)`’

Compute the square root of *expr*. This is equivalent to "(*expr*)^.5".

‘`not(expr)`’

Return 1.0 if *expr* is zero, 0.0 otherwise.

‘`pow(x, y)`’

Compute the power of *x* elevated *y*, it is equivalent to "(*x*)^(*y*)".

‘`random(x)`’

Return a pseudo random value between 0.0 and 1.0. *x* is the index of the internal variable which will be used to save the seed/state.

‘`hypot(x, y)`’

This function is similar to the C function with the same name; it returns "sqrt(*x***x* + *y***y*)", the length of the hypotenuse of a right triangle with sides of length *x* and *y*, or the distance of the point (*x*, *y*) from the origin.

‘`gcd(x, y)`’

Return the greatest common divisor of *x* and *y*. If both *x* and *y* are 0 or either or both are less than zero then behavior is undefined.

‘`if(x, y)`’

Evaluate *x*, and if the result is non-zero return the result of the evaluation of *y*, return 0 otherwise.

‘`ifnot(x, y)`’

Evaluate *x*, and if the result is zero return the result of the evaluation of *y*, return 0 otherwise.

‘`taylor(expr, x) taylor(expr, x, id)`’

Evaluate a taylor series at x. expr represents the LD(id)-th derivates of f(x) at 0. If id is not specified then 0 is assumed. note, when you have the derivatives at y instead of 0 taylor(expr, x-y) can be used When the series does not converge the results are undefined.

‘`root(expr, max)`’

Finds x where f(x)=0 in the interval 0..max. f() must be continuous or the result is undefined.

The following constants are available:

‘PI’

area of the unit disc, approximately 3.14

‘E’

exp(1) (Euler’s number), approximately 2.718

‘PHI’

golden ratio (1+sqrt(5))/2, approximately 1.618

Assuming that an expression is considered "true" if it has a non-zero value, note that:

`*` works like AND

`+` works like OR

and the construct:

```
if A then B else C
```

is equivalent to

```
if(A,B) + ifnot(A,C)
```

In your C code, you can extend the list of unary and binary functions, and define recognized constants, so that they are available for your expressions.

The evaluator also recognizes the International System number postfixes. If ’i’ is appended after the postfix, powers of 2 are used instead of powers of 10. The ’B’ postfix multiplies the value for 8, and can be appended after another postfix or used alone. This allows using for example ’KB’, ’MiB’, ’G’ and ’B’ as postfix.

Follows the list of available International System postfixes, with indication of the corresponding powers of 10 and of 2.

‘y’

-24 / -80

‘z’

-21 / -70

‘a’	-18 / -60
‘f’	-15 / -50
‘p’	-12 / -40
‘n’	-9 / -30
‘u’	-6 / -20
‘m’	-3 / -10
‘c’	-2
‘d’	-1
‘h’	2
‘k’	3 / 10
‘K’	3 / 10
‘M’	6 / 20
‘G’	

9 / 30

‘`T`’

12 / 40

‘`P`’

15 / 40

‘`E`’

18 / 50

‘`Z`’

21 / 60

‘`Y`’

24 / 70

10. Decoders

Decoders are configured elements in FFmpeg which allow the decoding of multimedia streams.

When you configure your FFmpeg build, all the supported native decoders are enabled by default. Decoders requiring an external library must be enabled manually via the corresponding `--enable-lib` option. You can list all available decoders using the configure option `--list-decoders`.

You can disable all the decoders with the configure option `--disable-decoders` and selectively enable / disable single decoders with the options `--enable-decoder=`*`DECODER`* / `--disable-decoder=`*`DECODER`*.

The option `-codecs` of the ff* tools will display the list of enabled decoders.

11. Video Decoders

A description of some of the currently available video decoders follows.

11.1 rawvideo

Raw video decoder.

This decoder decodes rawvideo streams.

11.1.1 Options

‘`top` *`top_field_first`*’

Specify the assumed field type of the input video.

‘`-1`’

the video is assumed to be progressive (default)

‘`0`’

bottom-field-first is assumed

‘`1`’

top-field-first is assumed

12. Audio Decoders

12.1 ffwavesynth

Internal wave synthetizer.

This decoder generates wave patterns according to predefined sequences. Its use is purely internal and the format of the data it accepts is not publicly documented.

13. Encoders

Encoders are configured elements in FFmpeg which allow the encoding of multimedia streams.

When you configure your FFmpeg build, all the supported native encoders are enabled by default. Encoders requiring an external library must be enabled manually via the corresponding `--enable-lib` option. You can list all available encoders using the configure option `--list-encoders`.

You can disable all the encoders with the configure option `--disable-encoders` and selectively enable / disable single encoders with the options `--enable-encoder=`*`ENCODER`* / `--disable-encoder=`*`ENCODER`*.

The option `-codecs` of the ff* tools will display the list of enabled encoders.

14. Audio Encoders

A description of some of the currently available audio encoders follows.

14.1 ac3 and ac3_fixed

AC-3 audio encoders.

These encoders implement part of ATSC A/52:2010 and ETSI TS 102 366, as well as the undocumented RealAudio 3 (a.k.a. dnet).

The *ac3* encoder uses floating-point math, while the *ac3_fixed* encoder only uses fixed-point integer math. This does not mean that one is always faster, just that one or the other may be better suited to a particular system. The floating-point encoder will generally produce better quality audio for a given bitrate. The *ac3_fixed* encoder is not the default codec for any of the output formats, so it must be specified explicitly using the option `-acodec ac3_fixed` in order to use it.

14.1.1 AC-3 Metadata

The AC-3 metadata options are used to set parameters that describe the audio, but in most cases do not affect the audio encoding itself. Some of the options do directly affect or influence the decoding and playback of the resulting bitstream, while others are just for informational purposes. A few of the options will add bits to the output stream that could otherwise be used for audio data, and will thus affect the quality of the output. Those will be indicated accordingly with a note in the option list below.

These parameters are described in detail in several publicly-available documents.

- A/52:2010 - Digital Audio Compression (AC-3) (E-AC-3) Standard
- A/54 - Guide to the Use of the ATSC Digital Television Standard
- Dolby Metadata Guide
- Dolby Digital Professional Encoding Guidelines

14.1.1.1 Metadata Control Options

‘`-per_frame_metadata` *`boolean`*’

Allow Per-Frame Metadata. Specifies if the encoder should check for changing metadata for each frame.

‘`0`’

The metadata values set at initialization will be used for every frame in the stream. (default)

‘`1`’

Metadata values can be changed before encoding each frame.

14.1.1.2 Downmix Levels

‘`-center_mixlev` *`level`*’

Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo. This field will only be written to the bitstream if a center channel is present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6dB gain

‘`-surround_mixlev` *`level`*’

Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo. This field will only be written to the bitstream if one or more surround channels are present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.500`’

Apply -6dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

14.1.1.3 Audio Production Information

Audio Production Information is optional information describing the mixing environment. Either none or both of the fields are written to the bitstream.

‘`-mixing_level` *`number`*’

Mixing Level. Specifies peak sound pressure level (SPL) in the production environment when the mix was mastered. Valid values are 80 to 111, or -1 for unknown or not indicated. The default value is -1, but that value cannot be used if the Audio Production Information is written to the bitstream. Therefore, if the `room_type` option is not the default value, the `mixing_level` option must not be -1.

‘`-room_type` *`type`*’

Room Type. Describes the equalization used during the final mixing session at the studio or on the dubbing stage. A large room is a dubbing stage with the industry standard X-curve equalization; a small room has flat equalization. This field will not be written to the bitstream if both the `mixing_level` option and the `room_type` option have the default values.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`large`’

Large Room

‘`2`’
‘`small`’

Small Room

14.1.1.4 Other Metadata Options

‘`-copyright` *`boolean`*’

Copyright Indicator. Specifies whether a copyright exists for this audio.

‘`0`’
‘`off`’

No Copyright Exists (default)

‘`1`’
‘`on`’

Copyright Exists

‘`-dialnorm` *`value`*’

Dialogue Normalization. Indicates how far the average dialogue level of the program is below digital 100% full scale (0 dBFS). This parameter determines a level shift during audio reproduction that sets the average volume of the dialogue to a preset level. The goal is to match volume level between program sources. A value of -31dB will result in no volume level change, relative to the source volume, during audio reproduction. Valid values are whole numbers in the range -31 to -1, with -31 being the default.

‘`-dsur_mode` *`mode`*’

Dolby Surround Mode. Specifies whether the stereo signal uses Dolby Surround (Pro Logic). This field will only be written to the bitstream if the audio stream is stereo. Using this option does **NOT** mean the encoder will actually apply Dolby Surround processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`off`’

Not Dolby Surround Encoded

‘`2`’
‘`on`’

Dolby Surround Encoded

‘`-original` *`boolean`*’

Original Bit Stream Indicator. Specifies whether this audio is from the original source and not a copy.

‘`0`’
‘`off`’

Not Original Source

‘`1`’
‘`on`’

Original Source (default)

14.1.2 Extended Bitstream Information

The extended bitstream options are part of the Alternate Bit Stream Syntax as specified in Annex D of the A/52:2010 standard. It is grouped into 2 parts. If any one parameter in a group is specified, all values in that group will be written to the bitstream. Default values are used for those that are written but have not been specified. If the mixing levels are written, the decoder will use these values instead of the ones specified in the `center_mixlev` and `surround_mixlev` options if it supports the Alternate Bit Stream Syntax.

14.1.2.1 Extended Bitstream Information - Part 1

‘`-dmix_mode` *`mode`*’

Preferred Stereo Downmix Mode. Allows the user to select either Lt/Rt (Dolby Surround) or Lo/Ro (normal stereo) as the preferred stereo downmix mode.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`ltrt`’

Lt/Rt Downmix Preferred

‘`2`’
‘`loro`’

Lo/Ro Downmix Preferred

‘`-ltrt_cmixlev` *`level`*’

Lt/Rt Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo in Lt/Rt mode.

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-ltrt_surmixlev` *`level`*’

Lt/Rt Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo in Lt/Rt mode.

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

‘`-loro_cmixlev` *`level`*’

Lo/Ro Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo in Lo/Ro mode.

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-loro_surmixlev` *`level`*’

Lo/Ro Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo in Lo/Ro mode.

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

14.1.2.2 Extended Bitstream Information - Part 2

‘`-dsurex_mode` *`mode`*’

> Dolby Surround EX Mode. Indicates whether the stream uses Dolby Surround EX (7.1 matrixed to 5.1). Using this option does **NOT** mean the encoder will actually apply Dolby Surround EX processing.
>
> ‘`0`’
> ‘`notindicated`’
>
> > Not Indicated (default)
>
> ‘`1`’
> ‘`on`’
>
> > Dolby Surround EX Off
>
> ‘`2`’
> ‘`off`’
>
> > Dolby Surround EX On

‘`-dheadphone_mode` *`mode`*’

> Dolby Headphone Mode. Indicates whether the stream uses Dolby Headphone encoding (multi-channel matrixed to 2.0 for use with headphones). Using this option does **NOT** mean the encoder will actually apply Dolby Headphone processing.
>
> ‘`0`’
> ‘`notindicated`’
>
> > Not Indicated (default)
>
> ‘`1`’
> ‘`on`’
>
> > Dolby Headphone Off
>
> ‘`2`’
> ‘`off`’
>
> > Dolby Headphone On

‘`-ad_conv_type` *`type`*’

> A/D Converter Type. Indicates whether the audio has passed through HDCD A/D conversion.

‘`0`’
‘`standard`’

Standard A/D Converter (default)

‘`1`’
‘`hdcd`’

HDCD A/D Converter

14.1.3 Other AC-3 Encoding Options

‘`-stereo_rematrixing` *`boolean`*’

Stereo Rematrixing. Enables/Disables use of rematrixing for stereo input. This is an optional AC-3 feature that increases quality by selectively encoding the left/right channels as mid/side. This option is enabled by default, and it is highly recommended that it be left as enabled except for testing purposes.

14.1.4 Floating-Point-Only AC-3 Encoding Options

These options are only valid for the floating-point encoder and do not exist for the fixed-point encoder due to the corresponding features not being implemented in fixed-point.

‘`-channel_coupling` *`boolean`*’

Enables/Disables use of channel coupling, which is an optional AC-3 feature that increases quality by combining high frequency information from multiple channels into a single channel. The per-channel high frequency information is sent with less accuracy in both the frequency and time domains. This allows more bits to be used for lower frequencies while preserving enough information to reconstruct the high frequencies. This option is enabled by default for the floating-point encoder and should generally be left as enabled except for testing purposes or to increase encoding speed.

‘`-1`’
‘`auto`’

Selected by Encoder (default)

‘`0`’
‘`off`’

Disable Channel Coupling

‘`1`’
‘`on`’

Enable Channel Coupling

‘`-cpl_start_band` *`number`*’

Coupling Start Band. Sets the channel coupling start band, from 1 to 15. If a value higher than the bandwidth is used, it will be reduced to 1 less than the coupling end band. If *auto* is used, the start band will be determined by the encoder based on the bit rate, sample rate, and channel layout. This option has no effect if channel coupling is disabled.

‘`-1`’
‘`auto`’

Selected by Encoder (default)

15. Video Encoders

A description of some of the currently available video encoders follows.

15.1 libtheora

Theora format supported through libtheora.

Requires the presence of the libtheora headers and library during configuration. You need to explicitly configure the build with `--enable-libtheora`.

15.1.1 Options

The following global options are mapped to internal libtheora options which affect the quality and the bitrate of the encoded stream.

‘`b`’

Set the video bitrate, only works if the `qscale` flag in ‘`flags`’ is not enabled.

‘`flags`’

Used to enable constant quality mode encoding through the ‘`qscale`’ flag, and to enable the `pass1` and `pass2` modes.

‘`g`’

Set the GOP size.

‘`global_quality`’

Set the global quality in lambda units, only works if the `qscale` flag in ‘`flags`’ is enabled. The value is clipped in the [0 - 10*`FF_QP2LAMBDA`] range, and then multiplied for 6.3 to get a value in the native libtheora range [0-63].

For example, to set maximum constant quality encoding with `ffmpeg`:

```
ffmpeg -i INPUT -flags:v qscale -global_quality:v "10*QP2LAMBDA" -codec:v libtheora OUTPUT.ogg
```

15.2 libvpx

VP8 format supported through libvpx.

Requires the presence of the libvpx headers and library during configuration. You need to explicitly configure the build with `--enable-libvpx`.

15.2.1 Options

Mapping from FFmpeg to libvpx options with conversion notes in parentheses.

‘`threads`’

g_threads

‘`profile`’

g_profile

‘`vb`’

rc_target_bitrate

‘`g`’

kf_max_dist

‘`keyint_min`’

kf_min_dist

‘`qmin`’

rc_min_quantizer

‘`qmax`’

rc_max_quantizer

‘`bufsize, vb`’

rc_buf_sz (`bufsize * 1000 / vb`)

rc_buf_optimal_sz (`bufsize * 1000 / vb * 5 / 6`)

‘`rc_init_occupancy, vb`’

rc_buf_initial_sz (`rc_init_occupancy * 1000 / vb`)

‘`rc_buffer_aggressivity`’

rc_undershoot_pct

‘`skip_threshold`’

rc_dropframe_thresh

‘`qcomp`’

rc_2pass_vbr_bias_pct

‘`maxrate, vb`’

rc_2pass_vbr_maxsection_pct (`maxrate * 100 / vb`)

‘`minrate, vb`’

rc_2pass_vbr_minsection_pct (`minrate * 100 / vb`)

‘`minrate, maxrate, vb`’

`VPX_CBR` (`minrate == maxrate == vb`)

‘`crf`’

`VPX_CQ`, `VP8E_SET_CQ_LEVEL`

‘`quality`’

‘*`best`*’

`VPX_DL_BEST_QUALITY`

‘*`good`*’

`VPX_DL_GOOD_QUALITY`

‘*`realtime`*’

VPX_DL_REALTIME

‘speed’

VP8E_SET_CPUUSED

‘nr’

VP8E_SET_NOISE_SENSITIVITY

‘mb_threshold’

VP8E_SET_STATIC_THRESHOLD

‘slices’

VP8E_SET_TOKEN_PARTITIONS

‘max-intra-rate’

VP8E_SET_MAX_INTRA_BITRATE_PCT

‘force_key_frames’

VPX_EFLAG_FORCE_KF

‘Alternate reference frame related’

‘vp8flags altref’

VP8E_SET_ENABLEAUTOALTREF

‘*arnr_max_frames*’

VP8E_SET_ARNR_MAXFRAMES

‘*arnr_type*’

VP8E_SET_ARNR_TYPE

‘*arnr_strength*’

VP8E_SET_ARNR_STRENGTH

‘*rc_lookahead*’

g_lag_in_frames

‘vp8flags error_resilient’

g_error_resilient

For more information about libvpx see: http://www.webmproject.org/

15.3 libx264

H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10 format supported through libx264.

Requires the presence of the libx264 headers and library during configuration. You need to explicitly configure the build with `--enable-libx264`.

15.3.1 Options

‘`preset` *`preset_name`*’

Set the encoding preset.

‘`tune` *`tune_name`*’

Tune the encoding params.

‘`fastfirstpass` *`bool`*’

Use fast settings when encoding first pass, default value is 1.

‘`profile` *`profile_name`*’

Set profile restrictions.

‘`level` *`level`*’

Specify level (as defined by Annex A). Deprecated in favor of *x264opts*.

‘`passlogfile` *`filename`*’

Specify filename for 2 pass stats. Deprecated in favor of *x264opts* (see *stats* libx264 option).

‘`wpredp` *`wpred_type`*’

Specify Weighted prediction for P-frames. Deprecated in favor of *x264opts* (see *weightp* libx264 option).

‘`x264opts` *`options`*’

Allow to set any x264 option, see x264 –fullhelp for a list.

options is a list of *key*=*value* couples separated by ":". In *filter* and *psy-rd* options that use ":" as a separator themselves, use "," instead. They accept it as well since long ago but this is kept undocumented for some reason.

For example to specify libx264 encoding options with `ffmpeg`:

```
ffmpeg -i foo.mpg -vcodec libx264 -x264opts keyint=123:min-keyint=20 -an out.mkv
```

For more information about libx264 and the supported options see:
http://www.videolan.org/developers/x264.html

16. Demuxers

Demuxers are configured elements in FFmpeg which allow to read the multimedia streams from a particular type of file.

When you configure your FFmpeg build, all the supported demuxers are enabled by default. You can list all available ones using the configure option "–list-demuxers".

You can disable all the demuxers using the configure option "–disable-demuxers", and selectively enable a single demuxer with the option "–enable-demuxer=*DEMUXER*", or disable it with the option "–disable-demuxer=*DEMUXER*".

The option "-formats" of the ff* tools will display the list of enabled demuxers.

The description of some of the currently available demuxers follows.

16.1 image2

Image file demuxer.

This demuxer reads from a list of image files specified by a pattern. The syntax and meaning of the pattern is specified by the option *pattern_type*.

The pattern may contain a suffix which is used to automatically determine the format of the images contained in the files.

The size, the pixel format, and the format of each image must be the same for all the files in the sequence.

This demuxer accepts the following options:

‘`framerate`’

Set the framerate for the video stream. It defaults to 25.

‘`loop`’

If set to 1, loop over the input. Default value is 0.

‘`pattern_type`’

Select the pattern type used to interpret the provided filename.

pattern_type accepts one of the following values.

‘`sequence`’

Select a sequence pattern type, used to specify a sequence of files indexed by sequential numbers.

A sequence pattern may contain the string "%d" or "%0*N*d", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0*N*d" is used, the string representing the number in each filename is 0-padded and *N* is the total number of 0-padded digits representing the number. The literal character ’%’ can be specified in the pattern with the string "%%".

If the sequence pattern contains "%d" or "%0*N*d", the first filename of the file list specified by the pattern must contain a number inclusively contained between *start_number* and *start_number*+*start_number_range*-1, and all the following numbers must be sequential.

For example the pattern "img-%03d.bmp" will match a sequence of filenames of the form ‘`img-001.bmp`’, ‘`img-002.bmp`’, ..., ‘`img-010.bmp`’, etc.; the pattern "i%%m%%g-%d.jpg" will match a sequence of filenames of the form ‘`i%m%g-1.jpg`’, ‘`i%m%g-2.jpg`’, ..., ‘`i%m%g-10.jpg`’, etc.

Note that the pattern must not necessarily contain "%d" or "%0*N*d", for example to convert a single image file ‘`img.jpeg`’ you can employ the command:

```
ffmpeg -i img.jpeg img.png
```

‘`glob`’

Select a glob wildcard pattern type.

The pattern is interpreted like a `glob()` pattern. This is only selectable if libavformat was compiled with globbing support.

‘`glob_sequence` *(deprecated, will be removed)*’

Select a mixed glob wildcard/sequence pattern.

If your version of libavformat was compiled with globbing support, and the provided pattern contains at least one glob meta character among `%*?[]{}` that is preceded by an unescaped "%", the pattern is interpreted like a `glob()` pattern, otherwise it is interpreted like a sequence pattern.

All glob special characters `%*?[]{}` must be prefixed with "%". To escape a literal "%" you shall use "%%".

For example the pattern `foo-%*.jpeg` will match all the filenames prefixed by "foo-" and terminating with ".jpeg", and `foo-%?%?%?.jpeg` will match all the filenames prefixed with "foo-", followed by a sequence of three characters, and terminating with ".jpeg".

This pattern type is deprecated in favor of *glob* and *sequence*.

Default value is *glob_sequence*.

‘`pixel_format`’

Set the pixel format of the images to read. If not specified the pixel format is guessed from the first image file in the sequence.

‘`start_number`’

Set the index of the file matched by the image file pattern to start to read from. Default value is 0.

‘`start_number_range`’

Set the index interval range to check when looking for the first image file in the sequence, starting from *start_number*. Default value is 5.

‘`video_size`’

Set the video size of the images to read. If not specified the video size is guessed from the first image file in the sequence.

16.1.1 Examples

- Use `ffmpeg` for creating a video from the images in the file sequence ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., assuming an input frame rate of 10 frames per second:

```
ffmpeg -i ’img-%03d.jpeg’ -r 10 out.mkv
```

- As above, but start by reading from a file with index 100 in the sequence:

```
ffmpeg -start_number 100 -i ’img-%03d.jpeg’ -r 10 out.mkv
```

- Read images matching the "*.png" glob pattern , that is all the files terminating with the ".png" suffix:

```
ffmpeg -pattern_type glob -i "*.png" -r 10 out.mkv
```

16.2 applehttp

Apple HTTP Live Streaming demuxer.

This demuxer presents all AVStreams from all variant streams. The id field is set to the bitrate variant index number. By setting the discard flags on AVStreams (by pressing 'a' or 'v' in ffplay), the caller can decide which variant streams to actually receive. The total bitrate of the variant that the stream belongs to is available in a metadata key named "variant_bitrate".

16.3 sbg

SBaGen script demuxer.

This demuxer reads the script language used by SBaGen http://uazu.net/sbagen/ to generate binaural beats sessions. A SBG script looks like that:

```
-SE
a: 300-2.5/3 440+4.5/0
b: 300-2.5/0 440+4.5/3
off: -
NOW      == a
+0:07:00 == b
+0:14:00 == a
+0:21:00 == b
+0:30:00    off
```

A SBG script can mix absolute and relative timestamps. If the script uses either only absolute timestamps (including the script start time) or only relative ones, then its layout is fixed, and the conversion is straightforward. On the other hand, if the script mixes both kind of timestamps, then the *NOW* reference for relative timestamps will be taken from the current time of day at the time the script is read, and the script layout will be frozen according to that reference. That means that if the script is directly played, the actual times will match the absolute timestamps up to the sound controller's clock accuracy, but if the user somehow pauses the playback or seeks, all times will be shifted accordingly.

17. Muxers

Muxers are configured elements in FFmpeg which allow writing multimedia streams to a particular type of file.

When you configure your FFmpeg build, all the supported muxers are enabled by default. You can list all available muxers using the configure option `--list-muxers`.

You can disable all the muxers with the configure option `--disable-muxers` and selectively enable / disable single muxers with the options `--enable-muxer=`*`MUXER`* / `--disable-muxer=`*`MUXER`*.

The option `-formats` of the ff* tools will display the list of enabled muxers.

A description of some of the currently available muxers follows.

17.1 crc

CRC (Cyclic Redundancy Check) testing format.

This muxer computes and prints the Adler-32 CRC of all the input audio and video frames. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the CRC.

The output of the muxer consists of a single line of the form: CRC=0x*CRC*, where *CRC* is a hexadecimal number 0-padded to 8 digits containing the CRC for all the decoded input frames.

For example to compute the CRC of the input, and store it in the file ‘`out.crc`’:

```
ffmpeg -i INPUT -f crc out.crc
```

You can print the CRC to stdout with the command:

```
ffmpeg -i INPUT -f crc -
```

You can select the output format of each frame with `ffmpeg` by specifying the audio and video codec and format. For example to compute the CRC of the input audio converted to PCM unsigned 8-bit and the input video converted to MPEG-2 video, use the command:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f crc -
```

See also the framecrc muxer.

17.2 framecrc

Per-packet CRC (Cyclic Redundancy Check) testing format.

This muxer computes and prints the Adler-32 CRC for each audio and video packet. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the CRC.

The output of the muxer consists of a line for each audio and video packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, 0xCRC
```

CRC is a hexadecimal number 0-padded to 8 digits containing the CRC of the packet.

For example to compute the CRC of the audio and video frames in '`INPUT`', converted to raw audio and video packets, and store it in the file '`out.crc`':

```
ffmpeg -i INPUT -f framecrc out.crc
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framecrc -
```

With `ffmpeg`, you can select the output format to which the audio and video frames are encoded before computing the CRC for each packet by specifying the audio and video codec. For example, to compute the CRC of each decoded input audio frame converted to PCM unsigned 8-bit and of each decoded input video frame converted to MPEG-2 video, use the command:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f framecrc -
```

See also the crc muxer.

17.3 framemd5

Per-packet MD5 testing format.

This muxer computes and prints the MD5 hash for each audio and video packet. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the hash.

The output of the muxer consists of a line for each audio and video packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, MD5
```

MD5 is a hexadecimal number representing the computed MD5 hash for the packet.

For example to compute the MD5 of the audio and video frames in '`INPUT`', converted to raw audio and video packets, and store it in the file '`out.md5`':

```
ffmpeg -i INPUT -f framemd5 out.md5
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framemd5 -
```

See also the md5 muxer.

17.4 hls

Apple HTTP Live Streaming muxer that segments MPEG-TS according to the HTTP Live Streaming specification.

It creates a playlist file and numbered segment files. The output filename specifies the playlist filename; the segment filenames receive the same basename as the playlist, a sequential number and a .ts extension.

```
ffmpeg -i in.nut out.m3u8
```

‘`-hls_time segment length in seconds`’
‘`-hls_list_size maximum number of playlist entries`’
‘`-hls_wrap number after which index wraps`’

17.5 ico

ICO file muxer.

Microsoft’s icon file format (ICO) has some strict limitations that should be noted:

- Size cannot exceed 256 pixels in any dimension
- Only BMP and PNG images can be stored
- If a BMP image is used, it must be one of the following pixel formats:

BMP Bit Depth	FFmpeg Pixel Format
1bit	pal8
4bit	pal8
8bit	pal8
16bit	rgb555le
24bit	bgr24
32bit	bgra

- If a BMP image is used, it must use the BITMAPINFOHEADER DIB header
- If a PNG image is used, it must use the rgba pixel format

17.6 image2

Image file muxer.

The image file muxer writes video frames to image files.

The output filenames are specified by a pattern, which can be used to produce sequentially numbered series of files. The pattern may contain the string "%d" or "%0*N*d", this string specifies the position of the characters representing a numbering in the filenames. If the form "%0*N*d" is used, the string representing the number in each filename is 0-padded to *N* digits. The literal character ’%’ can be specified in the pattern with the string "%%".

If the pattern contains "%d" or "%0*N*d", the first filename of the file list specified will contain the number 1, all the following numbers will be sequential.

The pattern may contain a suffix which is used to automatically determine the format of the image files to write.

For example the pattern "img-%03d.bmp" will specify a sequence of filenames of the form ‘`img-001.bmp`’, ‘`img-002.bmp`’, ..., ‘`img-010.bmp`’, etc. The pattern "img%%-%d.jpg" will specify a sequence of filenames of the form ‘`img%-1.jpg`’, ‘`img%-2.jpg`’, ..., ‘`img%-10.jpg`’, etc.

The following example shows how to use `ffmpeg` for creating a sequence of files ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., taking one image every second from the input video:

```
ffmpeg -i in.avi -vsync 1 -r 1 -f image2 'img-%03d.jpeg'
```

Note that with `ffmpeg`, if the format is not specified with the `-f` option and the output filename specifies an image file format, the image2 muxer is automatically selected, so the previous command can be written as:

```
ffmpeg -i in.avi -vsync 1 -r 1 'img-%03d.jpeg'
```

Note also that the pattern must not necessarily contain "%d" or "%0*N*d", for example to create a single image file ‘`img.jpeg`’ from the input video you can employ the command:

```
ffmpeg -i in.avi -f image2 -frames:v 1 img.jpeg
```

The image muxer supports the .Y.U.V image file format. This format is special in that that each image frame consists of three files, for each of the YUV420P components. To read or write this image file format, specify the name of the ’.Y’ file. The muxer will automatically open the ’.U’ and ’.V’ files as required.

17.7 md5

MD5 testing format.

This muxer computes and prints the MD5 hash of all the input audio and video frames. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the hash.

The output of the muxer consists of a single line of the form: MD5=*MD5*, where *MD5* is a hexadecimal number representing the computed MD5 hash.

For example to compute the MD5 hash of the input converted to raw audio and video, and store it in the file ‘`out.md5`’:

```
ffmpeg -i INPUT -f md5 out.md5
```

You can print the MD5 to stdout with the command:

```
ffmpeg -i INPUT -f md5 -
```

See also the framemd5 muxer.

17.8 MOV/MP4/ISMV

The mov/mp4/ismv muxer supports fragmentation. Normally, a MOV/MP4 file has all the metadata about all packets stored in one location (written at the end of the file, it can be moved to the start for better playback by adding *faststart* to the *movflags*, or using the `qt-faststart` tool). A fragmented file consists of a number of fragments, where packets and metadata about these packets are stored together. Writing a fragmented file has the advantage that the file is decodable even if the writing is interrupted (while a normal MOV/MP4 is undecodable if it is not properly finished), and it requires less memory when writing very long files (since writing normal MOV/MP4 files stores info about every single packet in memory until the file is closed). The downside is that it is less compatible with other applications.

Fragmentation is enabled by setting one of the AVOptions that define how to cut the file into fragments:

‘`-moov_size` *`bytes`*’

Reserves space for the moov atom at the beginning of the file instead of placing the moov atom at the end. If the space reserved is insufficient, muxing will fail.

‘`-movflags frag_keyframe`’

Start a new fragment at each video keyframe.

‘`-frag_duration` *`duration`*’

Create fragments that are *duration* microseconds long.

‘`-frag_size` *`size`*’

Create fragments that contain up to *size* bytes of payload data.

‘`-movflags frag_custom`’

Allow the caller to manually choose when to cut fragments, by calling `av_write_frame(ctx, NULL)` to write a fragment with the packets written so far. (This is only useful with other applications integrating libavformat, not from `ffmpeg`.)

‘`-min_frag_duration` *`duration`*’

Don’t create fragments that are shorter than *duration* microseconds long.

If more than one condition is specified, fragments are cut when one of the specified conditions is fulfilled. The exception to this is `-min_frag_duration`, which has to be fulfilled for any of the other conditions to apply.

Additionally, the way the output file is written can be adjusted through a few other options:

‘`-movflags empty_moov`’

Write an initial moov atom directly at the start of the file, without describing any samples in it. Generally, an mdat/moov pair is written at the start of the file, as a normal MOV/MP4 file, containing only a short portion of the file. With this option set, there is no initial mdat atom, and the moov atom only describes the tracks but has a zero duration.

Files written with this option set do not work in QuickTime. This option is implicitly set when writing ismv (Smooth Streaming) files.

‘`-movflags separate_moof`’

Write a separate moof (movie fragment) atom for each track. Normally, packets for all tracks are written in a moof atom (which is slightly more efficient), but with this option set, the muxer writes one moof/mdat pair for each track, making it easier to separate tracks.

This option is implicitly set when writing ismv (Smooth Streaming) files.

‘`-movflags faststart`’

Run a second pass moving the moov atom on top of the file. This operation can take a while, and will not work in various situations such as fragmented output, thus it is not enabled by default.

Smooth Streaming content can be pushed in real time to a publishing point on IIS with this muxer. Example:

```
ffmpeg -re <normal input/transcoding options> -movflags isml+frag_keyframe -f ismv http://server/publishingpoint.isml/Streams(Encoder1)
```

17.9 mpegts

MPEG transport stream muxer.

This muxer implements ISO 13818-1 and part of ETSI EN 300 468.

The muxer options are:

‘`-mpegts_original_network_id` *`number`*’

> Set the original_network_id (default 0x0001). This is unique identifier of a network in DVB. Its main use is in the unique identification of a service through the path Original_Network_ID, Transport_Stream_ID.

‘`-mpegts_transport_stream_id` *`number`*’

> Set the transport_stream_id (default 0x0001). This identifies a transponder in DVB.

‘`-mpegts_service_id` *`number`*’

> Set the service_id (default 0x0001) also known as program in DVB.

‘`-mpegts_pmt_start_pid` *`number`*’

> Set the first PID for PMT (default 0x1000, max 0x1f00).

‘`-mpegts_start_pid` *`number`*’

> Set the first PID for data packets (default 0x0100, max 0x0f00).

The recognized metadata settings in mpegts muxer are `service_provider` and `service_name`. If they are not set the default for `service_provider` is "FFmpeg" and the default for `service_name` is "Service01".

```
ffmpeg -i file.mpg -c copy \
     -mpegts_original_network_id 0x1122 \
     -mpegts_transport_stream_id 0x3344 \
     -mpegts_service_id 0x5566 \
     -mpegts_pmt_start_pid 0x1500 \
     -mpegts_start_pid 0x150 \
     -metadata service_provider="Some provider" \
     -metadata service_name="Some Channel" \
     -y out.ts
```

17.10 null

Null muxer.

This muxer does not generate any output file, it is mainly useful for testing or benchmarking purposes.

For example to benchmark decoding with `ffmpeg` you can use the command:

```
ffmpeg -benchmark -i INPUT -f null out.null
```

Note that the above command does not read or write the ‘`out.null`’ file, but specifying the output file is required by the `ffmpeg` syntax.

Alternatively you can write the command as:

```
ffmpeg -benchmark -i INPUT -f null -
```

17.11 matroska

Matroska container muxer.

This muxer implements the matroska and webm container specs.

The recognized metadata settings in this muxer are:

‘`title=`*`title name`*’

Name provided to a single track

‘`language=`*`language name`*’

Specifies the language of the track in the Matroska languages form

‘`stereo_mode=`*`mode`*’

Stereo 3D video layout of two views in a single video track

‘`mono`’

video is not stereo

‘`left_right`’

Both views are arranged side by side, Left-eye view is on the left

‘`bottom_top`’

Both views are arranged in top-bottom orientation, Left-eye view is at bottom

‘`top_bottom`’

Both views are arranged in top-bottom orientation, Left-eye view is on top

‘`checkerboard_rl`’

Each view is arranged in a checkerboard interleaved pattern, Left-eye view being first ‘`checkerboard_lr`’

  Each view is arranged in a checkerboard interleaved pattern, Right-eye view being first

‘`row_interleaved_rl`’

  Each view is constituted by a row based interleaving, Right-eye view is first row

‘`row_interleaved_lr`’

  Each view is constituted by a row based interleaving, Left-eye view is first row

‘`col_interleaved_rl`’

  Both views are arranged in a column based interleaving manner, Right-eye view is first column

‘`col_interleaved_lr`’

  Both views are arranged in a column based interleaving manner, Left-eye view is first column

‘`anaglyph_cyan_red`’

  All frames are in anaglyph format viewable through red-cyan filters

‘`right_left`’

  Both views are arranged side by side, Right-eye view is on the left

‘`anaglyph_green_magenta`’

  All frames are in anaglyph format viewable through green-magenta filters

‘`block_lr`’

  Both eyes laced in one Block, Left-eye view is first

‘`block_rl`’

  Both eyes laced in one Block, Right-eye view is first

For example a 3D WebM clip can be created using the following command line:

```
ffmpeg -i sample_left_right_clip.mpg -an -c:v libvpx -metadata stereo_mode=left_right -y stereo_clip.webm
```

17.12 segment, stream_segment, ssegment

Basic stream segmenter.

The segmenter muxer outputs streams to a number of separate files of nearly fixed duration. Output filename pattern can be set in a fashion similar to image2.

`stream_segment` is a variant of the muxer used to write to streaming output formats, i.e. which do not require global headers, and is recommended for outputting e.g. to MPEG transport stream segments. `ssegment` is a shorter alias for `stream_segment`.

Every segment starts with a video keyframe, if a video stream is present. Note that if you want accurate splitting for a video file, you need to make the input key frames correspond to the exact splitting times expected by the segmenter, or the segment muxer will start the new segment with the key frame found next after the specified start time.

The segment muxer works best with a single constant frame rate video.

Optionally it can generate a list of the created segments, by setting the option *segment_list*. The list type is specified by the *segment_list_type* option.

The segment muxer supports the following options:

‘`segment_format` *`format`*’

> Override the inner container format, by default it is guessed by the filename extension.

‘`segment_list` *`name`*’

> Generate also a listfile named *name*. If not specified no listfile is generated.

‘`segment_list_flags` *`flags`*’

> Set flags affecting the segment list generation.
>
> It currently supports the following flags:
>
> *cache*
>
> > Allow caching (only affects M3U8 list files).
>
> *live*
>
> > Allow live-friendly file generation.
> >
> > This currently only affects M3U8 lists. In particular, write a fake EXT-X-TARGETDURATION duration field at the top of the file, based on the specified *segment_time*.
>
> Default value is `cache`.

‘`segment_list_size` *`size`*’

Overwrite the listfile once it reaches *size* entries. If 0 the listfile is never overwritten. Default value is 0.

'`segment_list type` *`type`*'

Specify the format for the segment list file.

The following values are recognized:

'`flat`'

Generate a flat list for the created segments, one segment per line.

'`csv, ext`'

Generate a list for the created segments, one segment per line, each line matching the format (comma-separated values):

```
segment_filename,segment_start_time,segment_end_time
```

segment_filename is the name of the output file generated by the muxer according to the provided pattern. CSV escaping (according to RFC4180) is applied if required.

segment_start_time and *segment_end_time* specify the segment start and end time expressed in seconds.

A list file with the suffix `".csv"` or `".ext"` will auto-select this format.

`ext` is deprecated in favor or `csv`.

'`m3u8`'

Generate an extended M3U8 file, version 4, compliant with http://tools.ietf.org/id/draft-pantos-http-live-streaming-08.txt.

A list file with the suffix `".m3u8"` will auto-select this format.

If not specified the type is guessed from the list file name suffix.

'`segment_time` *`time`*'

Set segment duration to *time*. Default value is "2".

'`segment_time_delta` *`delta`*'

Specify the accuracy time when selecting the start time for a segment. Default value is "0".

When delta is specified a key-frame will start a new segment if its PTS satisfies the relation:

```
PTS >= start_time - time_delta
```

This option is useful when splitting video content, which is always split at GOP boundaries, in case a key frame is found just before the specified split time.

In particular may be used in combination with the ‘`ffmpeg`’ option *force_key_frames*. The key frame times specified by *force_key_frames* may not be set accurately because of rounding issues, with the consequence that a key frame time may result set just before the specified time. For constant frame rate videos a value of 1/2**frame_rate* should address the worst case mismatch between the specified time and the time set by *force_key_frames*.

‘`segment_times` *`times`*’

Specify a list of split points. *times* contains a list of comma separated duration specifications, in increasing order.

‘`segment_wrap` *`limit`*’

Wrap around segment index once it reaches *limit*.

Some examples follow.

- To remux the content of file ‘`in.mkv`’ to a list of segments ‘`out-000.nut`’, ‘`out-001.nut`’, etc., and write the list of generated segments to ‘`out.list`’:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.list out%03d.nut
```

- As the example above, but segment the input file according to the split points specified by the *segment_times* option:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 out%03d.nut
```

- As the example above, but use the `ffmpeg` *force_key_frames* option to force key frames in the input at the specified location, together with the segment option *segment_time_delta* to account for possible roundings operated when setting key frame times.

```
ffmpeg -i in.mkv -force_key_frames 1,2,3,5,8,13,21 -vcodec mpeg4 -acodec pcm_s16le -map 0 \
-f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 -segment_time_delta 0.05 out%03d.nut
```

  In order to force key frames on the input file, transcoding is required.

- To convert the ‘`in.mkv`’ to TS segments using the `libx264` and `libfaac` encoders:

```
ffmpeg -i in.mkv -map 0 -codec:v libx264 -codec:a libfaac -f ssegment -segment_list out.list out%03d.ts
```

- Segment the input file, and create an M3U8 live playlist (can be used as live HLS source):

```
ffmpeg -re -i in.mkv -codec copy -map 0 -f segment -segment_list playlist.m3u8 \
-segment_list_flags +live -segment_time 10 out%03d.mkv
```

17.13 mp3

The MP3 muxer writes a raw MP3 stream with an ID3v2 header at the beginning and optionally an ID3v1 tag at the end. ID3v2.3 and ID3v2.4 are supported, the `id3v2_version` option controls which one is used. The legacy ID3v1 tag is not written by default, but may be enabled with the `write_id3v1` option.

For seekable output the muxer also writes a Xing frame at the beginning, which contains the number of frames in the file. It is useful for computing duration of VBR files.

The muxer supports writing ID3v2 attached pictures (APIC frames). The pictures are supplied to the muxer in form of a video stream with a single packet. There can be any number of those streams, each will correspond to a single APIC frame. The stream metadata tags *title* and *comment* map to APIC *description* and *picture type* respectively. See http://id3.org/id3v2.4.0-frames for allowed picture types.

Note that the APIC frames must be written at the beginning, so the muxer will buffer the audio frames until it gets all the pictures. It is therefore advised to provide the pictures as soon as possible to avoid excessive buffering.

Examples:

Write an mp3 with an ID3v2.3 header and an ID3v1 footer:

```
ffmpeg -i INPUT -id3v2_version 3 -write_id3v1 1 out.mp3
```

Attach a picture to an mp3:

```
ffmpeg -i input.mp3 -i cover.png -c copy -metadata:s:v title="Album cover"
-metadata:s:v comment="Cover (Front)" out.mp3
```

18. Input Devices

Input devices are configured elements in FFmpeg which allow to access the data coming from a multimedia device attached to your system.

When you configure your FFmpeg build, all the supported input devices are enabled by default. You can list all available ones using the configure option "–list-indevs".

You can disable all the input devices using the configure option "–disable-indevs", and selectively enable an input device using the option "–enable-indev=*INDEV*", or you can disable a particular input device using the option "–disable-indev=*INDEV*".

The option "-formats" of the ff* tools will display the list of supported input devices (amongst the demuxers).

A description of the currently available input devices follows.

18.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

To enable this input device during configuration you need libasound installed on your system.

This device allows capturing from an ALSA device. The name of the device to capture has to be an ALSA card identifier.

An ALSA identifier has the syntax:

```
hw:CARD[,DEV[,SUBDEV]]
```

where the *DEV* and *SUBDEV* components are optional.

The three arguments (in order: *CARD*,*DEV*,*SUBDEV*) specify card number or identifier, device number and subdevice number (-1 means any).

To see the list of cards currently recognized by your system check the files '`/proc/asound/cards`' and '`/proc/asound/devices`'.

For example to capture with `ffmpeg` from an ALSA device with card id 0, you may run the command:

```
ffmpeg -f alsa -i hw:0 alsaout.wav
```

For more information see: http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html

18.2 bktr

BSD video input device.

18.3 dshow

Windows DirectShow input device.

DirectShow support is enabled when FFmpeg is built with the mingw-w64 project. Currently only audio and video devices are supported.

Multiple devices may be opened as separate inputs, but they may also be opened on the same input, which should improve synchronism between them.

The input name should be in the format:

```
TYPE=NAME[:TYPE=NAME]
```

where *TYPE* can be either *audio* or *video*, and *NAME* is the device’s name.

18.3.1 Options

If no options are specified, the device’s defaults are used. If the device does not support the requested options, it will fail to open.

‘`video_size`’

Set the video size in the captured video.

‘`framerate`’

Set the framerate in the captured video.

‘`sample_rate`’

Set the sample rate (in Hz) of the captured audio.

‘`sample_size`’

Set the sample size (in bits) of the captured audio.

‘`channels`’

Set the number of channels in the captured audio.

‘`list_devices`’

If set to ‘`true`’, print a list of devices and exit.

‘`list_options`’

If set to ‘`true`’, print a list of selected device’s options and exit.

‘`video_device_number`’

Set video device number for devices with same name (starts at 0, defaults to 0).

‘`audio_device_number`’

Set audio device number for devices with same name (starts at 0, defaults to 0).

‘`pixel_format`’

Select pixel format to be used by DirectShow. This may only be set when the video codec is not set or set to rawvideo.

‘`audio_buffer_size`’

Set audio device buffer size in milliseconds (which can directly impact latency, depending on the device). Defaults to using the audio device’s default buffer size (typically some multiple of 500ms). Setting this value too low can degrade performance. See also http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582(v=vs.85).aspx

18.3.2 Examples

- Print the list of DirectShow supported devices and exit:

  ```
  $ ffmpeg -list_devices true -f dshow -i dummy
  ```

- Open video device *Camera*:

  ```
  $ ffmpeg -f dshow -i video="Camera"
  ```

- Open second video device with name *Camera*:

  ```
  $ ffmpeg -f dshow -video_device_number 1 -i video="Camera"
  ```

- Open video device *Camera* and audio device *Microphone*:

  ```
  $ ffmpeg -f dshow -i video="Camera":audio="Microphone"
  ```

- Print the list of supported options in selected device and exit:

  ```
  $ ffmpeg -list_options true -f dshow -i video="Camera"
  ```

18.4 dv1394

Linux DV 1394 input device.

18.5 fbdev

Linux framebuffer input device.

The Linux framebuffer is a graphic hardware-independent abstraction layer to show graphics on a computer monitor, typically on the console. It is accessed through a file device node, usually '`/dev/fb0`'.

For more detailed information read the file Documentation/fb/framebuffer.txt included in the Linux source tree.

To record from the framebuffer device '`/dev/fb0`' with `ffmpeg`:

```
ffmpeg -f fbdev -r 10 -i /dev/fb0 out.avi
```

You can take a single screenshot image with the command:

```
ffmpeg -f fbdev -frames:v 1 -r 1 -i /dev/fb0 screenshot.jpeg
```

See also http://linux-fbdev.sourceforge.net/, and fbset(1).

18.6 iec61883

FireWire DV/HDV input device using libiec61883.

To enable this input device, you need libiec61883, libraw1394 and libavc1394 installed on your system. Use the configure option `--enable-libiec61883` to compile with the device enabled.

The iec61883 capture device supports capturing from a video device connected via IEEE1394 (FireWire), using libiec61883 and the new Linux FireWire stack (juju). This is the default DV/HDV input method in Linux Kernel 2.6.37 and later, since the old FireWire stack was removed.

Specify the FireWire port to be used as input file, or "auto" to choose the first port connected.

18.6.1 Options

'`dvtype`'

Override autodetection of DV/HDV. This should only be used if auto detection does not work, or if usage of a different device type should be prohibited. Treating a DV device as HDV (or vice versa) will not work and result in undefined behavior. The values '`auto`', '`dv`' and '`hdv`' are supported.

'`dvbuffer`'

Set maxiumum size of buffer for incoming data, in frames. For DV, this is an exact value. For HDV, it is not frame exact, since HDV does not have a fixed frame size.

‘`dvguid`’

Select the capture device by specifying it’s GUID. Capturing will only be performed from the specified device and fails if no device with the given GUID is found. This is useful to select the input if multiple devices are connected at the same time. Look at /sys/bus/firewire/devices to find out the GUIDs.

18.6.2 Examples

- Grab and show the input of a FireWire DV/HDV device.

```
ffplay -f iec61883 -i auto
```

- Grab and record the input of a FireWire DV/HDV device, using a packet buffer of 100000 packets if the source is HDV.

```
ffmpeg -f iec61883 -i auto -hdvbuffer 100000 out.mpg
```

18.7 jack

JACK input device.

To enable this input device during configuration you need libjack installed on your system.

A JACK input device creates one or more JACK writable clients, one for each audio channel, with name *client_name*:input_*N*, where *client_name* is the name provided by the application, and *N* is a number which identifies the channel. Each writable client will send the acquired data to the FFmpeg input device.

Once you have created one or more JACK readable clients, you need to connect them to one or more JACK writable clients.

To connect or disconnect JACK clients you can use the `jack_connect` and `jack_disconnect` programs, or do it through a graphical interface, for example with `qjackctl`.

To list the JACK clients and their properties you can invoke the command `jack_lsp`.

Follows an example which shows how to capture a JACK readable client with `ffmpeg`.

```
# Create a JACK writable client with name "ffmpeg".
$ ffmpeg -f jack -i ffmpeg -y out.wav

# Start the sample jack_metro readable client.
$ jack_metro -b 120 -d 0.2 -f 4000

# List the current JACK clients.
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

# Connect metro to the ffmpeg writable client.
$ jack_connect metro:120_bpm ffmpeg:input_1
```

For more information read: http://jackaudio.org/

18.8 lavfi

Libavfilter input virtual device.

This input device reads data from the open output pads of a libavfilter filtergraph.

For each filtergraph open output, the input device will create a corresponding stream which is mapped to the generated output. Currently only video data is supported. The filtergraph is specified through the option ‘`graph`’.

18.8.1 Options

‘`graph`’

Specify the filtergraph to use as input. Each video open output must be labelled by a unique string of the form "out*N*", where *N* is a number starting from 0 corresponding to the mapped input stream generated by the device. The first unlabelled output is automatically assigned to the "out0" label, but all the others need to be specified explicitly.

If not specified defaults to the filename specified for the input device.

‘`graph_file`’

Set the filename of the filtergraph to be read and sent to the other filters. Syntax of the filtergraph is the same as the one specified by the option *graph*.

18.8.2 Examples

- Create a color video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi -graph "color=pink [out0]" dummy
  ```

- As the previous example, but use filename for specifying the graph description, and omit the "out0" label:

  ```
  ffplay -f lavfi color=pink
  ```

- Create three different video test filtered sources and play them:

  ```
  ffplay -f lavfi -graph "testsrc [out0]; testsrc,hflip [out1]; testsrc,negate [out2]" test3
  ```

- Read an audio stream from a file using the amovie source and play it back with `ffplay`:

  ```
  ffplay -f lavfi "amovie=test.wav"
  ```

- Read an audio stream and a video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi "movie=test.avi[out0];amovie=test.wav[out1]"
  ```

18.9 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

18.10 openal

The OpenAL input device provides audio capture on all systems with a working OpenAL 1.1 implementation.

To enable this input device during configuration, you need OpenAL headers and libraries installed on your system, and need to configure FFmpeg with `--enable-openal`.

OpenAL headers and libraries should be provided as part of your OpenAL implementation, or as an additional download (an SDK). Depending on your installation you may need to specify additional flags via the `--extra-cflags` and `--extra-ldflags` for allowing the build system to locate the OpenAL headers and libraries.

An incomplete list of OpenAL implementations follows:

Creative

> The official Windows implementation, providing hardware acceleration with supported devices and software fallback. See http://openal.org/.

OpenAL Soft

> Portable, open source (LGPL) software implementation. Includes backends for the most common sound APIs on the Windows, Linux, Solaris, and BSD operating systems. See http://kcat.strangesoft.net/openal.html.

Apple

> OpenAL is part of Core Audio, the official Mac OS X Audio interface. See http://developer.apple.com/technologies/mac/audio-and-video.html

This device allows to capture from an audio input device handled through OpenAL.

You need to specify the name of the device to capture in the provided filename. If the empty string is provided, the device will automatically select the default device. You can get the list of the supported devices by using the option *list_devices*.

18.10.1 Options

'`channels`'

> Set the number of channels in the captured audio. Only the values '`1`' (monaural) and '`2`' (stereo) are currently supported. Defaults to '`2`'.

'`sample_size`'

> Set the sample size (in bits) of the captured audio. Only the values '`8`' and '`16`' are currently supported. Defaults to '`16`'.

'`sample_rate`'

> Set the sample rate (in Hz) of the captured audio. Defaults to '`44.1k`'.

'`list_devices`'

> If set to '`true`', print a list of devices and exit. Defaults to '`false`'.

18.10.2 Examples

Print the list of OpenAL supported devices and exit:

```
$ ffmpeg -list_devices true -f openal -i dummy out.ogg
```

Capture from the OpenAL device ‘`DR-BT101 via PulseAudio`’:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out.ogg
```

Capture from the default device (note the empty string ” as filename):

```
$ ffmpeg -f openal -i ’’ out.ogg
```

Capture from two devices simultaneously, writing to two different files, within the same `ffmpeg` command:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out1.ogg -f openal -i ’ALSA Default’ out2.ogg
```

Note: not all OpenAL implementations support multiple simultaneous capture - try the latest OpenAL Soft if the above does not work.

18.11 oss

Open Sound System input device.

The filename to provide to the input device is the device node representing the OSS input device, and is usually set to ‘`/dev/dsp`’.

For example to grab from ‘`/dev/dsp`’ using `ffmpeg` use the command:

```
ffmpeg -f oss -i /dev/dsp /tmp/oss.wav
```

For more information about OSS see: http://manuals.opensound.com/usersguide/dsp.html

18.12 pulse

pulseaudio input device.

To enable this input device during configuration you need libpulse-simple installed in your system.

The filename to provide to the input device is a source device or the string "default"

To list the pulse source devices and their properties you can invoke the command `pactl list sources`.

```
ffmpeg -f pulse -i default /tmp/pulse.wav
```

18.12.1 *server* AVOption

The syntax is:

```
-server server name
```

Connects to a specific server.

18.12.2 *name* AVOption

The syntax is:

```
-name application name
```

Specify the application name pulse will use when showing active clients, by default it is the LIBAVFORMAT_IDENT string

18.12.3 *stream_name* AVOption

The syntax is:

```
-stream_name stream name
```

Specify the stream name pulse will use when showing active streams, by default it is "record"

18.12.4 *sample_rate* AVOption

The syntax is:

```
-sample_rate samplerate
```

Specify the samplerate in Hz, by default 48kHz is used.

18.12.5 *channels* AVOption

The syntax is:

```
-channels N
```

Specify the channels in use, by default 2 (stereo) is set.

18.12.6 *frame_size* AVOption

The syntax is:

```
-frame_size bytes
```

Specify the number of byte per frame, by default it is set to 1024.

18.12.7 *fragment_size* AVOption

The syntax is:

```
-fragment_size bytes
```

Specify the minimal buffering fragment in pulseaudio, it will affect the audio latency. By default it is unset.

18.13 sndio

sndio input device.

To enable this input device during configuration you need libsndio installed on your system.

The filename to provide to the input device is the device node representing the sndio input device, and is usually set to ‘`/dev/audio0`’.

For example to grab from ‘`/dev/audio0`’ using `ffmpeg` use the command:

```
ffmpeg -f sndio -i /dev/audio0 /tmp/oss.wav
```

18.14 video4linux2

Video4Linux2 input video device.

The name of the device to grab is a file device node, usually Linux systems tend to automatically create such nodes when the device (e.g. an USB webcam) is plugged into the system, and has a name of the kind ‘`/dev/videoN`’, where *N* is a number associated to the device.

Video4Linux2 devices usually support a limited set of *width*x*height* sizes and framerates. You can check which are supported using `-list_formats all` for Video4Linux2 devices.

Some usage examples of the video4linux2 devices with ffmpeg and ffplay:

The time base for the timestamps is 1 microsecond. Depending on the kernel version and configuration, the timestamps may be derived from the real time clock (origin at the Unix Epoch) or the monotonic clock (origin usually at boot time, unaffected by NTP or manual changes to the clock). The ‘`-timestamps abs`’ or ‘`-ts abs`’ option can be used to force conversion into the real time clock.

Note that if FFmpeg is build with v4l-utils support ("–enable-libv4l2" option), it will always be used.

```
# Grab and show the input of a video4linux2 device.
ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0

# Grab and record the input of a video4linux2 device, leave the
framerate and size as previously set.
ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
```

"v4l" and "v4l2" can be used as aliases for the respective "video4linux" and "video4linux2".

18.15 vfwcap

VfW (Video for Windows) capture input device.

The filename passed as input is the capture driver number, ranging from 0 to 9. You may use "list" as filename to print a list of drivers. Any other filename will be interpreted as device number 0.

18.16 x11grab

X11 video input device.

This device allows to capture a region of an X11 display.

The filename passed as input has the syntax:

```
[hostname]:display_number.screen_number[+x_offset,y_offset]
```

hostname:*display_number*.*screen_number* specifies the X11 display name of the screen to grab from. *hostname* can be omitted, and defaults to "localhost". The environment variable `DISPLAY` contains the default display name.

x_offset and *y_offset* specify the offsets of the grabbed area with respect to the top-left border of the X11 screen. They default to 0.

Check the X11 documentation (e.g. man X) for more detailed information.

Use the `dpyinfo` program for getting basic information about the properties of your X11 display (e.g. grep for "name" or "dimensions").

For example to grab from ‘`:0.0`’ using `ffmpeg`:

```
ffmpeg -f x11grab -r 25 -s cif -i :0.0 out.mpg
```

Grab at position `10,20`:

```
ffmpeg -f x11grab -r 25 -s cif -i :0.0+10,20 out.mpg
```

18.16.1 Options

‘`draw_mouse`’

Specify whether to draw the mouse pointer. A value of `0` specify not to draw the pointer. Default value is `1`.

‘`follow_mouse`’

Make the grabbed area follow the mouse. The argument can be `centered` or a number of pixels *PIXELS*.

When it is specified with "centered", the grabbing region follows the mouse pointer and keeps the pointer at the center of region; otherwise, the region follows only when the mouse pointer reaches within *PIXELS* (greater than zero) to the edge of region.

For example:

```
ffmpeg -f x11grab -follow_mouse centered -r 25 -s cif -i :0.0 out.mpg
```

To follow only when the mouse pointer reaches within 100 pixels to edge:

```
ffmpeg -f x11grab -follow_mouse 100 -r 25 -s cif -i :0.0 out.mpg
```

‘`framerate`’

Set the grabbing frame rate. Default value is `ntsc`, corresponding to a framerate of `30000/1001`.

‘`show_region`’

Show grabbed region on screen.

If *show_region* is specified with `1`, then the grabbing region will be indicated on screen. With this option, it is easy to know what is being grabbed if only a portion of the screen is grabbed.

For example:

```
ffmpeg -f x11grab -show_region 1 -r 25 -s cif -i :0.0+10,20 out.mpg
```

With *follow_mouse*:

```
ffmpeg -f x11grab -follow_mouse centered -show_region 1 -r 25 -s cif -i :0.0 out.mpg
```

‘`video_size`’

Set the video frame size. Default value is `vga`.

19. Output Devices

Output devices are configured elements in FFmpeg which allow to write multimedia data to an output device attached to your system.

When you configure your FFmpeg build, all the supported output devices are enabled by default. You can list all available ones using the configure option "–list-outdevs".

You can disable all the output devices using the configure option "–disable-outdevs", and selectively enable an output device using the option "–enable-outdev=*OUTDEV*", or you can disable a particular input device using the option "–disable-outdev=*OUTDEV*".

The option "-formats" of the ff* tools will display the list of enabled output devices (amongst the muxers).

A description of the currently available output devices follows.

19.1 alsa

ALSA (Advanced Linux Sound Architecture) output device.

19.2 caca

CACA output device.

This output devices allows to show a video stream in CACA window. Only one CACA window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need to configure FFmpeg with `--enable-libcaca`. libcaca is a graphics library that outputs text instead of pixels.

For more information about libcaca, check: http://caca.zoy.org/wiki/libcaca

19.2.1 Options

'`window_title`'

Set the CACA window title, if not specified default to the filename specified for the output device.

'`window_size`'

Set the CACA window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video.

'`driver`'

Set display driver.

'`algorithm`'

Set dithering algorithm. Dithering is necessary because the picture being rendered has usually far more colours than the available palette. The accepted values are listed with `-list_dither algorithms`.

'`antialias`'

Set antialias method. Antialiasing smoothens the rendered image and avoids the commonly seen staircase effect. The accepted values are listed with `-list_dither antialiases`.

'`charset`'

Set which characters are going to be used when rendering text. The accepted values are listed with `-list_dither charsets`.

'`color`'

Set color to be used when rendering text. The accepted values are listed with `-list_dither colors`.

'`list_drivers`'

If set to '`true`', print a list of available drivers and exit.

'`list_dither`'

List available dither options related to the argument. The argument must be one of `algorithms`, `antialiases`, `charsets`, `colors`.

19.2.2 Examples

- The following command shows the `ffmpeg` output is an CACA window, forcing its size to 80x25:

  ```
  ffmpeg -i INPUT -vcodec rawvideo -pix_fmt rgb24 -window_size 80x25 -f caca -
  ```

- Show the list of available drivers and exit:

  ```
  ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_drivers true -
  ```

- Show the list of available dither colors and exit:

  ```
  ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_dither colors -
  ```

19.3 oss

OSS (Open Sound System) output device.

19.4 sdl

SDL (Simple DirectMedia Layer) output device.

This output devices allows to show a video stream in an SDL window. Only one SDL window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need libsdl installed on your system when configuring your build.

For more information about SDL, check: http://www.libsdl.org/

19.4.1 Options

‘`window_title`’

Set the SDL window title, if not specified default to the filename specified for the output device.

‘`icon_title`’

Set the name of the iconified SDL window, if not specified it is set to the same value of *window_title*.

‘`window_size`’

Set the SDL window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video, downscaled according to the aspect ratio.

19.4.2 Examples

The following command shows the `ffmpeg` output is an SDL window, forcing its size to the qcif format:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt yuv420p -window_size qcif -f sdl "SDL output"
```

19.5 sndio

sndio audio output device.

20. Protocols

Protocols are configured elements in FFmpeg which allow to access resources which require the use of a particular protocol.

When you configure your FFmpeg build, all the supported protocols are enabled by default. You can list all available ones using the configure option "–list-protocols".

You can disable all the protocols using the configure option "–disable-protocols", and selectively enable a protocol using the option "–enable-protocol=*PROTOCOL*", or you can disable a particular protocol using the option "–disable-protocol=*PROTOCOL*".

The option "-protocols" of the ff* tools will display the list of supported protocols.

A description of the currently available protocols follows.

20.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

BluRay angle

‘`chapter`’

Start chapter (1...N)

‘`playlist`’

Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

20.2 concat

Physical concatenation protocol.

Allow to read and seek from many resource in sequence as if they were a unique resource.

A URL accepted by this protocol has the syntax:

```
concat:URL1|URL2|...|URLN
```

where *URL1*, *URL2*, ..., *URLN* are the urls of the resource to be concatenated, each one possibly specifying a distinct protocol.

For example to read a sequence of files ‘`split1.mpeg`’, ‘`split2.mpeg`’, ‘`split3.mpeg`’ with `ffplay` use the command:

```
ffplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

Note that you may need to escape the character "|" which is special for many shells.

20.3 file

File access protocol.

Allow to read from or read to a file.

For example to read from a file ‘`input.mpeg`’ with `ffmpeg` use the command:

```
ffmpeg -i file:input.mpeg output.mpeg
```

The ff* tools default to the file protocol, that is a resource specified with the name "FILE.mpeg" is interpreted as the URL "file:FILE.mpeg".

20.4 gopher

Gopher protocol.

20.5 hls

Read Apple HTTP Live Streaming compliant segmented stream as a uniform one. The M3U8 playlists describing the segments can be remote HTTP resources or local files, accessed using the standard file protocol. The nested protocol is declared by specifying "+*proto*" after the hls URI scheme name, where *proto* is either "file" or "http".

```
hls+http://host/path/to/remote/resource.m3u8
hls+file://path/to/local/resource.m3u8
```

Using this protocol is discouraged - the hls demuxer should work just as well (if not, please report the issues) and is more complete. To use the hls demuxer instead, simply use the direct URLs to the m3u8 files.

20.6 http

HTTP (Hyper Text Transfer Protocol).

20.7 mmst

MMS (Microsoft Media Server) protocol over TCP.

20.8 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

```
mmsh://server[:port][/app][/playpath]
```

20.9 md5

MD5 output protocol.

Computes the MD5 hash of the data to be written, and on close writes this to the designated output or stdout if none is specified. It can be used to test muxers without writing an actual file.

Some examples follow.

```
# Write the MD5 hash of the encoded AVI file to the file output.avi.md5.
ffmpeg -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
ffmpeg -i input.flv -f avi -y md5:
```

Note that some formats (typically MOV) require the output protocol to be seekable, so they will fail with the MD5 output protocol.

20.10 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

number is the number corresponding to the file descriptor of the pipe (e.g. 0 for stdin, 1 for stdout, 2 for stderr). If *number* is not specified, by default the stdout file descriptor will be used for writing, stdin for reading.

For example to read from stdin with `ffmpeg`:

```
cat test.wav | ffmpeg -i pipe:0
# ...this is the same as...
cat test.wav | ffmpeg -i pipe:
```

For writing to stdout with `ffmpeg`:

```
ffmpeg -i test.wav -f avi pipe:1 | cat > test.avi
# ...this is the same as...
ffmpeg -i test.wav -f avi pipe: | cat > test.avi
```

Note that some formats (typically MOV), require the output protocol to be seekable, so they will fail with the pipe output protocol.

20.11 rtmp

Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimedia content across a TCP/IP network.

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

The number of the TCP port to use (by default is 1935).

‘`app`’

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. ‘`/ondemand/`’, ‘`/flash/live/`’, etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

‘`playpath`’

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

Maximum time to wait for the incoming connection. Implies listen.

Additionally, the following parameters can be set via command line options (or in code via `AVOptions`):

‘`rtmp_app`’

Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

‘`rtmp_buffer`’

Set the client buffer time in milliseconds. The default is 3000.

‘`rtmp_conn`’

Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with ’N’ and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘`rtmp_flashver`’

Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘`rtmp_flush_interval`’

Number of packets flushed in the same request (RTMPT only). The default is 10.

‘`rtmp_live`’

Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is `any`, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are `live` and `recorded`.

‘`rtmp_pageurl`’

URL of the web page in which the media was embedded. By default no value will be sent.

‘`rtmp_playpath`’

Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘`rtmp_subscribe`’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘`rtmp_swfhash`’

SHA256 hash of the decompressed SWF file (32 bytes).

‘`rtmp_swfsize`’

Size of the decompressed SWF file, required for SWFVerification.

‘`rtmp_swfurl`’

URL of the SWF player for the media. By default no value will be sent.

‘`rtmp_swfverify`’

URL to player swf file, compute hash/size automatically.

‘`rtmp_tcurl`’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with `ffplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
ffplay rtmp://myserver/vod/sample
```

20.12 rtmpe

Encrypted Real-Time Messaging Protocol.

The Encrypted Real-Time Messaging Protocol (RTMPE) is used for streaming multimedia content within standard cryptographic primitives, consisting of Diffie-Hellman key exchange and HMACSHA256, generating a pair of RC4 keys.

20.13 rtmps

Real-Time Messaging Protocol over a secure SSL connection.

The Real-Time Messaging Protocol (RTMPS) is used for streaming multimedia content across an encrypted connection.

20.14 rtmpt

Real-Time Messaging Protocol tunneled through HTTP.

The Real-Time Messaging Protocol tunneled through HTTP (RTMPT) is used for streaming multimedia content within HTTP requests to traverse firewalls.

20.15 rtmpte

Encrypted Real-Time Messaging Protocol tunneled through HTTP.

The Encrypted Real-Time Messaging Protocol tunneled through HTTP (RTMPTE) is used for streaming multimedia content within HTTP requests to traverse firewalls.

20.16 rtmpts

Real-Time Messaging Protocol tunneled through HTTPS.

The Real-Time Messaging Protocol tunneled through HTTPS (RTMPTS) is used for streaming multimedia content within HTTPS requests to traverse firewalls.

20.17 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

Requires the presence of the librtmp headers and library during configuration. You need to explicitly configure the build with "–enable-librtmp". If enabled this will replace the native RTMP protocol.

This protocol provides most client functions and a few server functions needed to support RTMP, RTMP tunneled in HTTP (RTMPT), encrypted RTMP (RTMPE), RTMP over SSL/TLS (RTMPS) and tunneled variants of these encrypted types (RTMPTE, RTMPTS).

The required syntax is:

```
rtmp_proto://server[:port][/app][/playpath] options
```

where *rtmp_proto* is one of the strings "rtmp", "rtmpt", "rtmpe", "rtmps", "rtmpte", "rtmpts" corresponding to each RTMP variant, and *server*, *port*, *app* and *playpath* have the same meaning as specified for the RTMP native protocol. *options* contains a list of space-separated options of the form *key*=*val*.

See the librtmp manual page (man 3 librtmp) for more information.

For example, to stream a file in real-time to an RTMP server using `ffmpeg`:

```
ffmpeg -re -i myfile -f flv rtmp://myserver/live/mystream
```

To play the same stream using `ffplay`:

```
ffplay "rtmp://myserver/live/mystream live=1"
```

20.18 rtp

Real-Time Protocol.

20.19 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock's RTSP server).

The required syntax for a RTSP url is:

```
rtsp://hostname[:port]/path
```

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

'`udp`'

Use UDP as lower transport protocol.

'`tcp`'

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

'`udp_multicast`'

Use UDP multicast as lower transport protocol.

'`http`'

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

'`filter_src`'

Accept packets only from negotiated peer address and port.

'`listen`'

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

When watching multi-bitrate Real-RTSP streams with `ffplay`, the streams to display can be chosen with `-vst` *n* and `-ast` *n* for video and audio respectively, and can be switched on the fly by pressing `v` and `a`.

Example command lines:

To watch a stream over UDP, with a max reordering delay of 0.5 seconds:

```
ffplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

To watch a stream tunneled over HTTP:

```
ffplay -rtsp_transport http rtsp://server/video.mp4
```

To send a stream in realtime to a RTSP server, for others to watch:

```
ffmpeg -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

To receive a stream in realtime:

```
ffmpeg -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

20.20 sap

Session Announcement Protocol (RFC 2974). This is not technically a protocol handler in libavformat, it is a muxer and demuxer. It is used for signalling of RTP streams, by announcing the SDP for the streams regularly on a separate port.

20.20.1 Muxer

The syntax for a SAP url given to the muxer is:

```
sap://destination[:port][?options]
```

The RTP packets are sent to *destination* on port *port*, or to port 5004 if no port is specified. *options* is a &-separated list. The following options are supported:

‘`announce_addr=`*`address`*’

Specify the destination IP address for sending the announcements to. If omitted, the announcements are sent to the commonly used SAP announcement multicast address 224.2.127.254 (sap.mcast.net), or ff0e::2:7ffe if *destination* is an IPv6 address.

'`announce_port=`*`port`*'

Specify the port to send the announcements on, defaults to 9875 if not specified.

'`ttl=`*`ttl`*'

Specify the time to live value for the announcements and RTP packets, defaults to 255.

'`same_port=`*`0|1`*'

If set to 1, send all RTP streams on the same port pair. If zero (the default), all streams are sent on unique ports, with each stream on a port 2 numbers higher than the previous. VLC/Live555 requires this to be set to 1, to be able to receive the stream. The RTP stack in libavformat for receiving requires all streams to be sent on unique ports.

Example command lines follow.

To broadcast a stream on the local subnet, for watching in VLC:

```
ffmpeg -re -i input -f sap sap://224.0.0.255?same_port=1
```

Similarly, for watching in `ffplay`:

```
ffmpeg -re -i input -f sap sap://224.0.0.255
```

And for watching in `ffplay`, over IPv6:

```
ffmpeg -re -i input -f sap sap://[ff0e::1:2:3:4]
```

20.20.2 Demuxer

The syntax for a SAP url given to the demuxer is:

```
sap://[address][:port]
```

address is the multicast address to listen for announcements on, if omitted, the default 224.2.127.254 (sap.mcast.net) is used. *port* is the port that is listened on, 9875 if omitted.

The demuxers listens for announcements on the given address and port. Once an announcement is received, it tries to receive that particular stream.

Example command lines follow.

To play back the first stream announced on the normal SAP multicast address:

```
ffplay sap://
```

To play back the first stream announced on one the default IPv6 SAP multicast address:

```
ffplay sap://[ff0e::2:7ffe]
```

20.21 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

```
tcp://hostname:port[?options]
```

‘`listen`’

Listen for an incoming connection

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

20.22 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

```
tls://hostname:port[?options]
```

‘`listen`’

Act as a server, listening for an incoming connection.

‘`cafile=`*`filename`*’

Certificate authority file. The file must be in OpenSSL PEM format.

‘`cert=`*`filename`*’

Certificate file. The file must be in OpenSSL PEM format.

‘`key=`*`filename`*’

Private key file.

‘`verify=`*`0`*`|`*`1`*’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

```
ffplay tls://hostname:port
```

20.23 udp

User Datagram Protocol.

The required syntax for a UDP url is:

```
udp://hostname:port[?options]
```

options contains a list of &-separated options of the form *key*=*val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

‘`buffer_size=`*`size`*’

Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

‘`localport=`*`port`*’

Override the local UDP port to bind with.

‘`localaddr=`*`addr`*’

Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

‘`pkt_size=`*`size`*’

Set the size in bytes of UDP packets.

‘`reuse=`*`1|0`*’

Explicitly allow or disallow reusing UDP sockets.

‘`ttl=`*`ttl`*’

Set the time to live value (for multicast only).

‘`connect=`*`1|0`*’

Initialize the UDP socket with `connect()`. In this case, the destination address can’t be changed with ff_udp_set_remote_url later. If the destination address isn’t known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

‘`sources=`*`address`*`[,`*`address`*`]`’

Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘`block=`*`address`*`[,`*`address`*`]`’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘`fifo_size=`*`units`*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘`overrun_nonfatal=`*`1|0`*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

```
ffmpeg -i input -f format udp://hostname:port
```

To stream in mpegts format over UDP using 188 sized UDP packets, using a large input buffer:

```
ffmpeg -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

To receive over UDP from a remote endpoint:

```
ffmpeg -i udp://[multicast-address]:port
```

21. Bitstream Filters

When you configure your FFmpeg build, all the supported bitstream filters are enabled by default. You can list all available ones using the configure option `--list-bsfs`.

You can disable all the bitstream filters using the configure option `--disable-bsfs`, and selectively enable any bitstream filter using the option `--enable-bsf=BSF`, or you can disable a particular bitstream filter using the option `--disable-bsf=BSF`.

The option `-bsfs` of the ff* tools will display the list of all the supported bitstream filters included in your build.

Below is a description of the currently available bitstream filters.

21.1 aac_adtstoasc

21.2 chomp

21.3 dump_extradata

21.4 h264_mp4toannexb

Convert an H.264 bitstream from length prefixed mode to start code prefixed mode (as defined in the Annex B of the ITU-T H.264 specification).

This is required by some streaming formats, typically the MPEG-2 transport stream format ("mpegts").

For example to remux an MP4 file containing an H.264 stream to mpegts format with `ffmpeg`, you can use the command:

```
ffmpeg -i INPUT.mp4 -codec copy -bsf:v h264_mp4toannexb OUTPUT.ts
```

21.5 imx_dump_header

21.6 mjpeg2jpeg

Convert MJPEG/AVI1 packets to full JPEG/JFIF packets.

MJPEG is a video codec wherein each video frame is essentially a JPEG image. The individual frames can be extracted without loss, e.g. by

```
ffmpeg -i ../some_mjpeg.avi -c:v copy frames_%d.jpg
```

Unfortunately, these chunks are incomplete JPEG images, because they lack the DHT segment required for decoding. Quoting from http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml:

Avery Lee, writing in the rec.video.desktop newsgroup in 2001, commented that "MJPEG, or at least the MJPEG in AVIs having the MJPG fourcc, is restricted JPEG with a fixed – and *omitted* – Huffman table. The JPEG must be YCbCr colorspace, it must be 4:2:2, and it must use basic Huffman encoding, not arithmetic or progressive. . . . You can indeed extract the MJPEG frames and decode them with a regular JPEG decoder, but you have to prepend the DHT segment to them, or else the decoder won't have any idea how to decompress the data. The exact table necessary is given in the OpenDML spec."

This bitstream filter patches the header of frames extracted from an MJPEG stream (carrying the AVI1 header ID and lacking a DHT segment) to produce fully qualified JPEG images.

```
ffmpeg -i mjpeg-movie.avi -c:v copy -bsf:v mjpeg2jpeg frame_%d.jpg
exiftran -i -9 frame*.jpg
ffmpeg -i frame_%d.jpg -c:v copy rotated.avi
```

21.7 mjpega_dump_header

21.8 movsub

21.9 mp3_header_compress

21.10 mp3_header_decompress

21.11 noise

21.12 remove_extradata

22. Filtering Introduction

Filtering in FFmpeg is enabled through the libavfilter library.

Libavfilter is the filtering API of FFmpeg. It is the substitute of the now deprecated ’vhooks’ and started as a Google Summer of Code project.

Audio filtering integration into the main FFmpeg repository is a work in progress, so audio API and ABI should not be considered stable yet.

In libavfilter, it is possible for filters to have multiple inputs and multiple outputs. To illustrate the sorts of things that are possible, we can use a complex filter graph. For example, the following one:

```
input --> split --> fifo -----------------------> overlay --> output
            |                                        ^
            |                                        |
            +------> fifo --> crop --> vflip --------+
```

splits the stream in two streams, sends one stream through the crop filter and the vflip filter before merging it back with the other stream by overlaying it on top. You can use the following command to achieve this:

```
ffmpeg -i input -vf "[in] split [T1], fifo, [T2] overlay=0:H/2 [out]; [T1] fifo, crop=iw:ih/2:0:ih/2, vflip [T2]" output
```

The result will be that in output the top half of the video is mirrored onto the bottom half.

Filters are loaded using the *-vf* or *-af* option passed to `ffmpeg` or to `ffplay`. Filters in the same linear chain are separated by commas. In our example, *split, fifo, overlay* are in one linear chain, and *fifo, crop, vflip* are in another. The points where the linear chains join are labeled by names enclosed in square brackets. In our example, that is *[T1]* and *[T2]*. The special labels *[in]* and *[out]* are the points where video is input and output.

Some filters take in input a list of parameters: they are specified after the filter name and an equal sign, and are separated from each other by a colon.

There exist so-called *source filters* that do not have an audio/video input, and *sink filters* that will not have audio/video output.

23. graph2dot

The ‘`graph2dot`’ program included in the FFmpeg ‘`tools`’ directory can be used to parse a filter graph description and issue a corresponding textual representation in the dot language.

Invoke the command:

```
graph2dot -h
```

to see how to use ‘`graph2dot`’.

You can then pass the dot description to the ‘`dot`’ program (from the graphviz suite of programs) and obtain a graphical representation of the filter graph.

For example the sequence of commands:

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

can be used to create and display an image representing the graph described by the *GRAPH_DESCRIPTION* string. Note that this string must be a complete self-contained graph, with its inputs and outputs explicitly defined. For example if your command line is of the form:

```
ffmpeg -i infile -vf scale=640:360 outfile
```

your *GRAPH_DESCRIPTION* string will need to be of the form:

```
nullsrc,scale=640:360,nullsink
```

you may also need to set the *nullsrc* parameters and add a *format* filter in order to simulate a specific input file.

24. Filtergraph description

A filtergraph is a directed graph of connected filters. It can contain cycles, and there can be multiple links between a pair of filters. Each link has one input pad on one side connecting it to one filter from which it takes its input, and one output pad on the other side connecting it to the one filter accepting its output.

Each filter in a filtergraph is an instance of a filter class registered in the application, which defines the features and the number of input and output pads of the filter.

A filter with no input pads is called a "source", a filter with no output pads is called a "sink".

24.1 Filtergraph syntax

A filtergraph can be represented using a textual representation, which is recognized by the '`-filter`'/'`-vf`' and '`-filter_complex`' options in `ffmpeg` and '`-vf`' in `ffplay`, and by the `avfilter_graph_parse()`/`avfilter_graph_parse2()` function defined in '`libavfilter/avfiltergraph.h`'.

A filterchain consists of a sequence of connected filters, each one connected to the previous one in the sequence. A filterchain is represented by a list of ","-separated filter descriptions.

A filtergraph consists of a sequence of filterchains. A sequence of filterchains is represented by a list of ";"-separated filterchain descriptions.

A filter is represented by a string of the form:
[*in_link_1*]...[*in_link_N*]*filter_name*=*arguments*[*out_link_1*]...[*out_link_M*]

filter_name is the name of the filter class of which the described filter is an instance of, and has to be the name of one of the filter classes registered in the program. The name of the filter class is optionally followed by a string "=*arguments*".

arguments is a string which contains the parameters used to initialize the filter instance, and are described in the filter descriptions below.

The list of arguments can be quoted using the character "'" as initial and ending mark, and the character '\' for escaping the characters within the quoted text; otherwise the argument string is considered terminated when the next special character (belonging to the set "[]=;,") is encountered.

The name and arguments of the filter are optionally preceded and followed by a list of link labels. A link label allows to name a link and associate it to a filter output or input pad. The preceding labels *in_link_1* ... *in_link_N*, are associated to the filter input pads, the following labels *out_link_1* ... *out_link_M*, are associated to the output pads.

When two link labels with the same name are found in the filtergraph, a link between the corresponding input and output pad is created.

If an output pad is not labelled, it is linked by default to the first unlabelled input pad of the next filter in the filterchain. For example in the filterchain:

```
nullsrc, split[L1], [L2]overlay, nullsink
```

the split filter instance has two output pads, and the overlay filter instance two input pads. The first output pad of split is labelled "L1", the first input pad of overlay is labelled "L2", and the second output pad of split is linked to the second input pad of overlay, which are both unlabelled.

In a complete filterchain all the unlabelled filter input and output pads must be connected. A filtergraph is considered valid if all the filter input and output pads of all the filterchains are connected.

Libavfilter will automatically insert scale filters where format conversion is required. It is possible to specify swscale flags for those automatically inserted scalers by prepending `sws_flags=flags;` to the filtergraph description.

Follows a BNF description for the filtergraph syntax:

```
NAME             ::= sequence of alphanumeric characters and ’_’
LINKLABEL        ::= "[" NAME "]"
LINKLABELS       ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (eventually quoted)
FILTER           ::= [LINKNAMES] NAME ["=" ARGUMENTS] [LINKNAMES]
FILTERCHAIN      ::= FILTER [,FILTERCHAIN]
FILTERGRAPH      ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

24.2 Notes on filtergraph escaping

Some filter arguments require the use of special characters, typically `:` to separate key=value pairs in a named options list. In this case the user should perform a first level escaping when specifying the filter arguments. For example, consider the following literal string to be embedded in the drawtext filter arguments:

```
this is a ’string’: may contain one, or more, special characters
```

Since `:` is special for the filter arguments syntax, it needs to be escaped, so you get:

```
text=this is a \’string\’\: may contain one, or more, special characters
```

A second level of escaping is required when embedding the filter arguments in a filtergraph description, in order to escape all the filtergraph special characters. Thus the example above becomes:

```
drawtext=text=this is a \\\’string\\\’\\: may contain one\, or more\, special characters
```

Finally an additional level of escaping may be needed when writing the filtergraph description in a shell command, which depends on the escaping rules of the adopted shell. For example, assuming that `\` is special and needs to be escaped with another `\`, the previous string will finally result in:

```
-vf "drawtext=text=this is a \\\\\\’string\\\\\\’\\\\: may contain one\\, or more\\, special characters"
```

Sometimes, it might be more convenient to employ quoting in place of escaping. For example the string:

```
Caesar: tu quoque, Brute, fili mi
```

Can be quoted in the filter arguments as:

```
text=’Caesar: tu quoque, Brute, fili mi’
```

And finally inserted in a filtergraph like:

```
drawtext=text=\’Caesar: tu quoque\, Brute\, fili mi\’
```

See the Quoting and escaping section for more information about the escaping and quoting rules adopted by FFmpeg.

25. Audio Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the audio filters included in your build.

Below is a description of the currently available audio filters.

25.1 aconvert

Convert the input audio format to the specified formats.

The filter accepts a string of the form: "*sample_format*:*channel_layout*".

sample_format specifies the sample format, and can be a string or the corresponding numeric value defined in ‘`libavutil/samplefmt.h`’. Use ’p’ suffix for a planar sample format.

channel_layout specifies the channel layout, and can be a string or the corresponding number value defined in ‘`libavutil/channel_layout.h`’.

The special parameter "auto", signifies that the filter will automatically select the output format depending on the output filter.

Some examples follow.

- Convert input to float, planar, stereo:

  ```
  aconvert=fltp:stereo
  ```

- Convert input to unsigned 8-bit, automatically select out channel layout:

  ```
  aconvert=u8:auto
  ```

25.2 aformat

Convert the input audio to one of the specified formats. The framework will negotiate the most appropriate format to minimize conversions.

The filter accepts the following named parameters:

‘`sample_fmts`’

A comma-separated list of requested sample formats.

‘`sample_rates`’

A comma-separated list of requested sample rates.

‘`channel_layouts`’

A comma-separated list of requested channel layouts.

If a parameter is omitted, all values are allowed.

For example to force the output to either unsigned 8-bit or signed 16-bit stereo:

```
aformat=sample_fmts\=u8\,s16:channel_layouts\=stereo
```

25.3 amerge

Merge two or more audio streams into a single multi-channel stream.

The filter accepts the following named options:

‘`inputs`’

Set the number of inputs. Default is 2.

If the channel layouts of the inputs are disjoint, and therefore compatible, the channel layout of the output will be set accordingly and the channels will be reordered as necessary. If the channel layouts of the inputs are not disjoint, the output will have all the channels of the first input then all the channels of the second input, in that order, and the channel layout of the output will be the default value corresponding to the total number of channels.

For example, if the first input is in 2.1 (FL+FR+LF) and the second input is FC+BL+BR, then the output will be in 5.1, with the channels in the following order: a1, a2, b1, a3, b2, b3 (a1 is the first channel of the first input, b1 is the first channel of the second input).

On the other hand, if both input are in stereo, the output channels will be in the default order: a1, a2, b1, b2, and the channel layout will be arbitrarily set to 4.0, which may or may not be the expected value.

All inputs must have the same sample rate, and format.

If inputs do not have the same duration, the output will stop with the shortest.

Example: merge two mono files into a stereo stream:

```
amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
```

Example: multiple merges:

```
ffmpeg -f lavfi -i "
amovie=input.mkv:si=0 [a0];
amovie=input.mkv:si=1 [a1];
amovie=input.mkv:si=2 [a2];
amovie=input.mkv:si=3 [a3];
amovie=input.mkv:si=4 [a4];
amovie=input.mkv:si=5 [a5];
[a0][a1][a2][a3][a4][a5] amerge=inputs=6" -c:a pcm_s16le output.mkv
```

25.4 amix

Mixes multiple audio inputs into a single output.

For example

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

will mix 3 input audio streams to a single output with the same duration as the first input and a dropout transition time of 3 seconds.

The filter accepts the following named parameters:

‘`inputs`’

Number of inputs. If unspecified, it defaults to 2.

‘`duration`’

How to determine the end-of-stream.

‘`longest`’

Duration of longest input. (default)

‘`shortest`’

Duration of shortest input.

‘`first`’

Duration of first input.

‘`dropout_transition`’

Transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

25.5 anull

Pass the audio source unchanged to the output.

25.6 aresample

Resample the input audio to the specified sample rate.

The filter accepts exactly one parameter, the output sample rate. If not specified then the filter will automatically convert between its input and output sample rates.

For example, to resample the input audio to 44100Hz:

```
aresample=44100
```

25.7 asetnsamples

Set the number of samples per each output audio frame.

The last output packet may contain a different number of samples, as the filter will flush all the remaining samples when the input audio signal its end.

The filter accepts parameters as a list of *key*=*value* pairs, separated by ":".

‘`nb_out_samples, n`’

> Set the number of frames per each output audio frame. The number is intended as the number of samples *per each channel*. Default value is 1024.

‘`pad, p`’

> If set to 1, the filter will pad the last audio frame with zeroes, so that the last frame will contain the same number of samples as the previous ones. Default value is 1.

For example, to set the number of per-frame samples to 1234 and disable padding for the last frame, use:

```
asetnsamples=n=1234:p=0
```

25.8 ashowinfo

Show a line containing various information for each input audio frame. The input audio is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

A description of each shown parameter follows:

‘`n`’

> sequential number of the input frame, starting from 0

‘`pts`’

> Presentation timestamp of the input frame, in time base units; the time base depends on the filter input pad, and is usually 1/*sample_rate*.

‘`pts_time`’

> presentation timestamp of the input frame in seconds

‘`pos`’

> position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic audio)

‘`fmt`’

> sample format

‘`chlayout`’

channel layout

‘`rate`’

sample rate for the audio frame

‘`nb_samples`’

number of samples (per channel) in the frame

‘`checksum`’

Adler-32 checksum (printed in hexadecimal) of the audio data. For planar audio the data is treated as if all the planes were concatenated.

‘`plane_checksums`’

A list of Adler-32 checksums for each data plane.

25.9 asplit

Split input audio into several identical outputs.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

For example:

```
[in] asplit [out0][out1]
```

will create two separate outputs from the same input.

To create 3 or more outputs, you need to specify the number of outputs, like in:

```
[in] asplit=3 [out0][out1][out2]
```

```
ffmpeg -i INPUT -filter_complex asplit=5 OUTPUT
```

will create 5 copies of the input audio.

25.10 astreamsync

Forward two audio streams and control the order the buffers are forwarded.

The argument to the filter is an expression deciding which stream should be forwarded next: if the result is negative, the first stream is forwarded; if the result is positive or zero, the second stream is forwarded. It can use the following variables:

b1 b2

number of buffers forwarded so far on each stream

s1 s2

number of samples forwarded so far on each stream

t1 t2

current timestamp of each stream

The default value is `t1-t2`, which means to always forward the stream that has a smaller timestamp.

Example: stress-test `amerge` by randomly sending buffers on the wrong input, while avoiding too much of a desynchronization:

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

25.11 atempo

Adjust audio tempo.

The filter accepts exactly one parameter, the audio tempo. If not specified then the filter will assume nominal 1.0 tempo. Tempo must be in the [0.5, 2.0] range.

For example, to slow down audio to 80% tempo:

```
atempo=0.8
```

For example, to speed up audio to 125% tempo:

```
atempo=1.25
```

25.12 earwax

Make audio easier to listen to on headphones.

This filter adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Ported from SoX.

25.13 pan

Mix channels with specific gain levels. The filter accepts the output channel layout followed by a set of channels definitions.

This filter is also designed to remap efficiently the channels of an audio stream.

The filter accepts parameters of the form: "*l*:*outdef*:*outdef*:..."

‘`l`’

output channel layout or number of channels

‘`outdef`’

output channel specification, of the form: "*out_name*=[*gain**]*in_name*[+[*gain**]*in_name*...]"

‘`out_name`’

output channel to define, either a channel name (FL, FR, etc.) or a channel number (c0, c1, etc.)

‘`gain`’

multiplicative coefficient for the channel, 1 leaving the volume unchanged

‘`in_name`’

input channel to use, see out_name for details; it is not possible to mix named and numbered input channels

If the ‘=’ in a channel specification is replaced by ‘<’, then the gains for that specification will be renormalized so that the total is 1, thus avoiding clipping noise.

25.13.1 Mixing examples

For example, if you want to down-mix from stereo to mono, but with a bigger factor for the left channel:

```
pan=1:c0=0.9*c0+0.1*c1
```

A customized down-mix to stereo that works automatically for 3-, 4-, 5- and 7-channels surround:

```
pan=stereo: FL < FL + 0.5*FC + 0.6*BL + 0.6*SL : FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

Note that `ffmpeg` integrates a default down-mix (and up-mix) system that should be preferred (see "-ac" option) unless you have very specific needs.

25.13.2 Remapping examples

The channel remapping will be effective if, and only if:

- gain coefficients are zeroes or ones,
- only one input per channel output,

If all these conditions are satisfied, the filter will notify the user ("Pure channel mapping detected"), and use an optimized and lossless method to do the remapping.

For example, if you have a 5.1 source and want a stereo audio stream by dropping the extra channels:

```
pan="stereo: c0=FL : c1=FR"
```

Given the same source, you can also switch front left and front right channels and keep the input channel layout:

```
pan="5.1: c0=c1 : c1=c0 : c2=c2 : c3=c3 : c4=c4 : c5=c5"
```

If the input is a stereo audio stream, you can mute the front left channel (and still keep the stereo channel layout) with:

```
pan="stereo:c1=c1"
```

Still with a stereo audio stream input, you can copy the right channel in both front left and right:

```
pan="stereo: c0=FR : c1=FR"
```

25.14 silencedetect

Detect silence in an audio stream.

This filter logs a message when it detects that the input audio volume is less or equal to a noise tolerance value for a duration greater or equal to the minimum detected noise duration.

The printed times and duration are expressed in seconds.

‘`duration, d`’

> Set silence duration until notification (default is 2 seconds).

‘`noise, n`’

> Set noise tolerance. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default is -60dB, or 0.001.

Detect 5 seconds of silence with -50dB noise tolerance:

```
silencedetect=n=-50dB:d=5
```

Complete example with `ffmpeg` to detect silence with 0.0001 noise tolerance in ‘`silence.mp3`’:

```
ffmpeg -f lavfi -i amovie=silence.mp3,silencedetect=noise=0.0001 -f null -
```

25.15 volume

Adjust the input audio volume.

The filter accepts exactly one parameter *vol*, which expresses how the audio volume will be increased or decreased.

Output values are clipped to the maximum value.

If *vol* is expressed as a decimal number, the output audio volume is given by the relation:

```
output_volume = vol * input_volume
```

If *vol* is expressed as a decimal number followed by the string "dB", the value represents the requested change in decibels of the input audio power, and the output audio volume is given by the relation:

```
output_volume = 10^(vol/20) * input_volume
```

Otherwise *vol* is considered an expression and its evaluated value is used for computing the output audio volume according to the first relation.

Default value for *vol* is 1.0.

25.15.1 Examples

- Half the input audio volume:

  ```
  volume=0.5
  ```

  The above example is equivalent to:

  ```
  volume=1/2
  ```

- Decrease input audio power by 12 decibels:

  ```
  volume=-12dB
  ```

25.16 volumedetect

Detect the volume of the input video.

The filter has no parameters. The input is not modified. Statistics about the volume will be printed in the log when the input stream end is reached.

In particular it will show the mean volume (root mean square), maximum volume (on a per-sample basis), and the beginning of an histogram of the registered volume values (from the maximum value to a cumulated 1/1000 of the samples).

All volumes are in decibels relative to the maximum PCM value.

Here is an excerpt of the output:

```
[Parsed_volumedetect_0  0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0  0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0  0xa23120] histogram_4db: 6
[Parsed_volumedetect_0  0xa23120] histogram_5db: 62
[Parsed_volumedetect_0  0xa23120] histogram_6db: 286
[Parsed_volumedetect_0  0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0  0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0  0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0  0xa23120] histogram_10db: 8409
```

It means that:

- The mean square energy is approximately -27 dB, or 10^-2.7.
- The largest sample is at -4 dB, or more precisely between -4 dB and -5 dB.
- There are 6 samples at -4 dB, 62 at -5 dB, 286 at -6 dB, etc.

In other words, raising the volume by +4 dB does not cause any clipping, raising it by +5 dB causes clipping for 6 samples, etc.

25.17 asyncts

Synchronize audio data with timestamps by squeezing/stretching it and/or dropping samples/adding silence when needed.

The filter accepts the following named parameters:

‘`compensate`’

: Enable stretching/squeezing the data to make it match the timestamps. Disabled by default. When disabled, time gaps are covered with silence.

‘`min_delta`’

: Minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. Default value is 0.1. If you get non-perfect sync with this filter, try setting this parameter to 0.

‘`max_comp`’

: Maximum compensation in samples per second. Relevant only with compensate=1. Default value 500.

‘`first_pts`’

: Assume the first pts should be this value. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame’s expected pts, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream.

25.18 channelsplit

Split each channel in input audio stream into a separate output stream.

This filter accepts the following named parameters:

‘`channel_layout`’

: Channel layout of the input stream. Default is "stereo".

For example, assuming a stereo input MP3 file

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

will create an output Matroska file with two audio streams, one containing only the left channel and the other the right channel.

To split a 5.1 WAV file into per-channel files

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

25.19 channelmap

Remap input channels to new locations.

This filter accepts the following named parameters:

‘`channel_layout`’

Channel layout of the output stream.

‘`map`’

Map channels from input to output. The argument is a comma-separated list of mappings, each in the *`in_channel-out_channel`* or *in_channel* form. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the input channel layout. *out_channel* is the name of the output channel or its index in the output channel layout. If *out_channel* is not given then it is implicitly an index, starting with zero and increasing by one for each mapping.

If no mapping is present, the filter will implicitly map input channels to output channels preserving index.

For example, assuming a 5.1+downmix input MOV file

```
ffmpeg -i in.mov -filter 'channelmap=map=DL-FL\,DR-FR' out.wav
```

will create an output WAV file tagged as stereo from the downmix channels of the input.

To fix a 5.1 WAV improperly encoded in AAC’s native channel order

```
ffmpeg -i in.wav -filter 'channelmap=1\,2\,0\,5\,3\,4:channel_layout=5.1' out.wav
```

25.20 join

Join multiple input streams into one multi-channel stream.

The filter accepts the following named parameters:

'`inputs`'

Number of input streams. Defaults to 2.

'`channel_layout`'

Desired output channel layout. Defaults to stereo.

'`map`'

Map channels from inputs to output. The argument is a comma-separated list of mappings, each in the `input_idx.in_channel-out_channel` form. *input_idx* is the 0-based index of the input stream. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the specified input stream. *out_channel* is the name of the output channel.

The filter will attempt to guess the mappings when those are not specified explicitly. It does so by first trying to find an unused matching input channel and if that fails it picks the first unused input channel.

E.g. to join 3 inputs (with properly set channel layouts)

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

To build a 5.1 output from 6 single-channel streams:

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex
'join=inputs=6:channel_layout=5.1:map=0.0-FL\,1.0-FR\,2.0-FC\,3.0-SL\,4.0-SR\,5.0-LFE'
out
```

25.21 resample

Convert the audio sample format, sample rate and channel layout. This filter is not meant to be used directly.

26. Audio Sources

Below is a description of the currently available audio sources.

26.1 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in '`libavfilter/asrc_abuffer.h`'.

It accepts the following mandatory parameters: *sample_rate*:*sample_fmt*:*channel_layout*

'`sample_rate`'

> The sample rate of the incoming audio buffers.

'`sample_fmt`'

> The sample format of the incoming audio buffers. Either a sample format name or its corresponging integer representation from the enum AVSampleFormat in '`libavutil/samplefmt.h`'

'`channel_layout`'

> The channel layout of the incoming audio buffers. Either a channel layout name from channel_layout_map in '`libavutil/channel_layout.c`' or its corresponding integer representation from the AV_CH_LAYOUT_* macros in '`libavutil/channel_layout.h`'

For example:

```
abuffer=44100:s16p:stereo
```

will instruct the source to accept planar 16bit signed stereo at 44100Hz. Since the sample format with name "s16p" corresponds to the number 6 and the "stereo" channel layout corresponds to the value 0x3, this is equivalent to:

```
abuffer=44100:6:0x3
```

26.2 aevalsrc

Generate an audio signal specified by an expression.

This source accepts in input one or more expressions (one for each channel), which are evaluated and used to generate a corresponding audio signal.

It accepts the syntax: *exprs*[::*options*]. *exprs* is a list of expressions separated by ":", one for each separate channel. In case the *channel_layout* is not specified, the selected channel layout depends on the number of provided expressions.

options is an optional sequence of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

'`channel_layout, c`'

> Set the channel layout. The number of channels in the specified layout must be equal to the number of specified expressions.

‘`duration, d`’

Set the minimum duration of the sourced audio. See the function `av_parse_time()` for the accepted format. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame.

If not specified, or the expressed duration is negative, the audio is supposed to be generated forever.

‘`nb_samples, n`’

Set the number of samples per channel per each output frame, default to 1024.

‘`sample_rate, s`’

Specify the sample rate, default to 44100.

Each expression in *exprs* can contain the following constants:

‘`n`’

number of the evaluated sample, starting from 0

‘`t`’

time of the evaluated sample expressed in seconds, starting from 0

‘`s`’

sample rate

26.2.1 Examples

- Generate silence:

  ```
  aevalsrc=0
  ```

- Generate a sin signal with frequency of 440 Hz, set sample rate to 8000 Hz:

  ```
  aevalsrc="sin(440*2*PI*t)::s=8000"
  ```

- Generate a two channels signal, specify the channel layout (Front Center + Back Center) explicitly:

  ```
  aevalsrc="sin(420*2*PI*t):cos(430*2*PI*t)::c=FC|BC"
  ```

- Generate white noise:

```
aevalsrc="-2+random(0)"
```

- Generate an amplitude modulated signal:

  ```
  aevalsrc="sin(10*2*PI*t)*sin(880*2*PI*t)"
  ```

- Generate 2.5 Hz binaural beats on a 360 Hz carrier:

  ```
  aevalsrc="0.1*sin(2*PI*(360-2.5/2)*t) : 0.1*sin(2*PI*(360+2.5/2)*t)"
  ```

26.3 anullsrc

Null audio source, return unprocessed audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools, or as the source for filters which ignore the input data (for example the sox synth filter).

It accepts an optional sequence of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`sample_rate, s`’

Specify the sample rate, and defaults to 44100.

‘`channel_layout, cl`’

Specify the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is "stereo".

Check the channel_layout_map definition in ‘`libavutil/channel_layout.c`’ for the mapping between strings and channel layout values.

‘`nb_samples, n`’

Set the number of samples per requested frames.

Follow some examples:

```
#  set the sample rate to 48000 Hz and the channel layout to AV_CH_LAYOUT_MONO.
anullsrc=r=48000:cl=4

# same as
anullsrc=r=48000:cl=mono
```

26.4 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is not intended to be part of user-supplied graph descriptions but for insertion by calling programs through the interface defined in ‘`libavfilter/buffersrc.h`’.

It accepts the following named parameters:

‘`time_base`’

Timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

‘`sample_rate`’

Audio sample rate.

‘`sample_fmt`’

Name of the sample format, as returned by `av_get_sample_fmt_name()`.

‘`channel_layout`’

Channel layout of the audio data, in the form that can be accepted by `av_get_channel_layout()`.

All the parameters need to be explicitly defined.

26.5 flite

Synthesize a voice utterance using the libflite library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libflite`.

Note that the flite library is not thread-safe.

The source accepts parameters as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`list_voices`’

If set to 1, list the names of the available voices and exit immediately. Default value is 0.

‘`nb_samples, n`’

Set the maximum number of samples per frame. Default value is 512.

‘`textfile`’

Set the filename containing the text to speak.

‘`text`’

Set the text to speak.

‘`voice, v`’

Set the voice to use for the speech synthesis. Default value is `kal`. See also the *list_voices* option.

26.5.1 Examples

- Read from file ‘`speech.txt`’, and synthetize the text using the standard flite voice:

  ```
  flite=textfile=speech.txt
  ```

- Read the specified text selecting the `slt` voice:

  ```
  flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Input text to ffmpeg:

  ```
  ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Make ‘`ffplay`’ speak the specified text, using `flite` and the `lavfi` device:

  ```
  ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
  ```

For more information about libflite, check: http://www.speech.cs.cmu.edu/flite/

27. Audio Sinks

Below is a description of the currently available audio sinks.

27.1 abuffersink

Buffer audio frames, and make them available to the end of filter chain.

This sink is mainly intended for programmatic use, in particular through the interface defined in ‘`libavfilter/buffersink.h`’.

It requires a pointer to an AVABufferSinkContext structure, which defines the incoming buffers' formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

27.2 anullsink

Null audio sink, do absolutely nothing with the input audio. It is mainly useful as a template and to be employed in analysis / debugging tools.

27.3 abuffersink

This sink is intended for programmatic use. Frames that arrive on this sink can be retrieved by the calling program using the interface defined in '`libavfilter/buffersink.h`'.

This filter accepts no parameters.

28. Video Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the video filters included in your build.

Below is a description of the currently available video filters.

28.1 alphaextract

Extract the alpha component from the input as a grayscale video. This is especially useful with the *alphamerge* filter.

28.2 alphamerge

Add or replace the alpha component of the primary input with the grayscale value of a second input. This is intended for use with *alphaextract* to allow the transmission or storage of frame sequences that have alpha in a format that doesn't support an alpha channel.

For example, to reconstruct full frames from a normal YUV-encoded video and a separate video created with *alphaextract*, you might use:

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

Since this filter is designed for reconstruction, it operates on frame sequences without considering timestamps, and terminates when either input reaches end of stream. This will cause problems if your encoding pipeline drops frames. If you're trying to apply an image as an overlay to a video stream, consider the *overlay* filter instead.

28.3 ass

Draw ASS (Advanced Substation Alpha) subtitles on top of input video using the libass library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libass`.

This filter accepts the following named options, expressed as a sequence of *key*=*value* pairs, separated by ":".

‘`filename, f`’

Set the filename of the ASS file to read. It must be specified.

‘`original_size`’

Specify the size of the original video, the video for which the ASS file was composed. Due to a misdesign in ASS aspect ratio arithmetic, this is necessary to correctly scale the fonts if the aspect ratio has been changed.

If the first key is not specified, it is assumed that the first value specifies the ‘`filename`’.

For example, to render the file ‘`sub.ass`’ on top of the input video, use the command:

```
ass=sub.ass
```

which is equivalent to:

```
ass=filename=sub.ass
```

28.4 bbox

Compute the bounding box for the non-black pixels in the input frame luminance plane.

This filter computes the bounding box containing all the pixels with a luminance value greater than the minimum allowed value. The parameters describing the bounding box are printed on the filter log.

28.5 blackdetect

Detect video intervals that are (almost) completely black. Can be useful to detect chapter transitions, commercials, or invalid recordings. Output lines contains the time for the start, end and duration of the detected black interval expressed in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

This filter accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

'`black_min_duration, d`'

Set the minimum detected black duration expressed in seconds. It must be a non-negative floating point number.

Default value is 2.0.

'`picture_black_ratio_th, pic_th`'

Set the threshold for considering a picture "black". Express the minimum value for the ratio:

```
nb_black_pixels / nb_pixels
```

for which a picture is considered black. Default value is 0.98.

'`pixel_black_th, pix_th`'

Set the threshold for considering a pixel "black".

The threshold expresses the maximum pixel luminance value for which a pixel is considered "black". The provided value is scaled according to the following equation:

```
absolute_threshold = luminance_minimum_value + pixel_black_th * luminance_range_size
```

luminance_range_size and *luminance_minimum_value* depend on the input video format, the range is [0-255] for YUV full-range formats and [16-235] for YUV non full-range formats.

Default value is 0.10.

The following example sets the maximum pixel threshold to the minimum value, and detects only black intervals of 2 or more seconds:

```
blackdetect=d=2:pix_th=0.00
```

28.6 blackframe

Detect frames that are (almost) completely black. Can be useful to detect chapter transitions or commercials. Output lines consist of the frame number of the detected frame, the percentage of blackness, the position in the file if known or -1 and the timestamp in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the syntax:

```
blackframe[=amount:[threshold]]
```

amount is the percentage of the pixels that have to be below the threshold, and defaults to 98.

threshold is the threshold below which a pixel value is considered black, and defaults to 32.

28.7 boxblur

Apply boxblur algorithm to the input video.

This filter accepts the parameters:
luma_radius:*luma_power*:*chroma_radius*:*chroma_power*:*alpha_radius*:*alpha_power*

Chroma and alpha parameters are optional, if not specified they default to the corresponding values set for *luma_radius* and *luma_power*.

luma_radius, *chroma_radius*, and *alpha_radius* represent the radius in pixels of the box used for blurring the corresponding input plane. They are expressions, and can contain the following constants:

‘`w, h`’

the input width and height in pixels

‘`cw, ch`’

the input chroma image width and height in pixels

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

The radius must be a non-negative number, and must not be greater than the value of the expression `min(w,h)/2` for the luma and alpha planes, and of `min(cw,ch)/2` for the chroma planes.

luma_power, *chroma_power*, and *alpha_power* represent how many times the boxblur filter is applied to the corresponding plane.

Some examples follow:

- Apply a boxblur filter with luma, chroma, and alpha radius set to 2:

  ```
  boxblur=2:1
  ```

- Set luma radius to 2, alpha and chroma radius to 0

  ```
  boxblur=2:1:0:0:0:0
  ```

- Set luma and chroma radius to a fraction of the video dimension

  ```
  boxblur=min(h\,w)/10:1:min(cw\,ch)/10:1
  ```

28.8 colormatrix

The colormatrix filter allows conversion between any of the following color space: BT.709 (*bt709*), BT.601 (*bt601*), SMPTE-240M (*smpte240m*) and FCC (*fcc*).

The syntax of the parameters is *source*:*destination*:

```
colormatrix=bt601:smpte240m
```

28.9 copy

Copy the input source unchanged to the output. Mainly useful for testing purposes.

28.10 crop

Crop the input video to *out_w*:*out_h*:*x*:*y*:*keep_aspect*

The *keep_aspect* parameter is optional, if specified and set to a non-zero value will force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

The *out_w*, *out_h*, *x*, *y* parameters are expressions containing the following constants:

‘`x, y`’

  the computed values for *x* and *y*. They are evaluated for each new frame.

‘`in_w, in_h`’

  the input width and height

‘`iw, ih`’

  same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘`n`’

the number of input frame, starting from 0

‘`pos`’

the position in the file of the input frame, NAN if unknown

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

The *out_w* and *out_h* parameters specify the expressions for the width and height of the output (cropped) video. They are evaluated just at the configuration of the filter.

The default value of *out_w* is "in_w", and the default value of *out_h* is "in_h".

The expression for *out_w* may depend on the value of *out_h*, and the expression for *out_h* may depend on *out_w*, but they cannot depend on *x* and *y*, as *x* and *y* are evaluated after *out_w* and *out_h*.

The *x* and *y* parameters specify the expressions for the position of the top-left corner of the output (non-cropped) area. They are evaluated for each frame. If the evaluated value is not valid, it is approximated to the nearest valid value.

The default value of *x* is "(in_w-out_w)/2", and the default value for *y* is "(in_h-out_h)/2", which set the cropped area at the center of the input image.

The expression for *x* may depend on *y*, and the expression for *y* may depend on *x*.

Follow some examples:

```
# crop the central input area with size 100x100
crop=100:100

# crop the central input area with size 2/3 of the input video
"crop=2/3*in_w:2/3*in_h"

# crop the input video central square
crop=in_h

# delimit the rectangle with the top-left corner placed at position
# 100:100 and the right-bottom corner corresponding to the right-bottom
# corner of the input image.
crop=in_w-100:in_h-100:100:100

# crop 10 pixels from the left and right borders, and 20 pixels from
# the top and bottom borders
"crop=in_w-2*10:in_h-2*20"

# keep only the bottom right quarter of the input image
"crop=in_w/2:in_h/2:in_w/2:in_h/2"

# crop height for getting Greek harmony
"crop=in_w:1/PHI*in_w"

# trembling effect
"crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)"

# erratic camera effect depending on timestamp
"crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"

# set x depending on the value of y
"crop=in_w/2:in_h/2:y:10+10*sin(n/10)"
```

28.11 cropdetect

Auto-detect crop size.

Calculate necessary cropping parameters and prints the recommended parameters through the logging system. The detected dimensions correspond to the non-black area of the input video.

It accepts the syntax:

```
cropdetect[=limit[:round[:reset]]]
```

‘`limit`’

Threshold, which can be optionally specified from nothing (0) to everything (255), defaults to 24.

‘`round`’

Value which the width/height should be divisible by, defaults to 16. The offset is automatically adjusted to center the video. Use 2 to get only even dimensions (needed for 4:2:2 video). 16 is best when encoding to most video codecs.

‘`reset`’

Counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Defaults to 0.

This can be useful when channel logos distort the video area. 0 indicates never reset and return the largest area encountered during playback.

28.12 decimate

This filter drops frames that do not differ greatly from the previous frame in order to reduce framerate. The main use of this filter is for very-low-bitrate encoding (e.g. streaming over dialup modem), but it could in theory be used for fixing movies that were inverse-telecined incorrectly.

It accepts the following parameters: *max*:*hi*:*lo*:*frac*.

‘`max`’

Set the maximum number of consecutive frames which can be dropped (if positive), or the minimum interval between dropped frames (if negative). If the value is 0, the frame is dropped unregarding the number of previous sequentially dropped frames.

Default value is 0.

‘`hi, lo, frac`’

Set the dropping threshold values.

Values for *hi* and *lo* are for 8x8 pixel blocks and represent actual pixel value differences, so a threshold of 64 corresponds to 1 unit of difference for each pixel, or the same spread out differently over the block.

A frame is a candidate for dropping if no 8x8 blocks differ by more than a threshold of *hi*, and if no more than *frac* blocks (1 meaning the whole image) differ by more than a threshold of *lo*.

Default value for *hi* is 64*12, default value for *lo* is 64*5, and default value for *frac* is 0.33.

28.13 delogo

Suppress a TV station logo by a simple interpolation of the surrounding pixels. Just set a rectangle covering the logo and watch it disappear (and sometimes something even uglier appear - your mileage may vary).

The filter accepts parameters as a string of the form "*x*:*y*:*w*:*h*:*band*", or as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`x, y`’

Specify the top left corner coordinates of the logo. They must be specified.

‘`w, h`’

Specify the width and height of the logo to clear. They must be specified.

‘`band, t`’

Specify the thickness of the fuzzy edge of the rectangle (added to *w* and *h*). The default value is 4.

‘`show`’

When set to 1, a green rectangle is drawn on the screen to simplify finding the right *x*, *y*, *w*, *h* parameters, and *band* is set to 4. The default value is 0.

Some examples follow.

- Set a rectangle covering the area with top left corner coordinates 0,0 and size 100x77, setting a band of size 10:

  ```
  delogo=0:0:100:77:10
  ```

- As the previous example, but use named options:

  ```
  delogo=x=0:y=0:w=100:h=77:band=10
  ```

28.14 deshake

Attempt to fix small changes in horizontal and/or vertical shift. This filter helps remove camera shake from hand-holding a camera, bumping a tripod, moving on a vehicle, etc.

The filter accepts parameters as a string of the form "*x*:*y*:*w*:*h*:*rx*:*ry*:*edge*:*blocksize*:*contrast*:*search*:*filename*"

A description of the accepted parameters follows.

‘`x, y, w, h`’

Specify a rectangular area where to limit the search for motion vectors. If desired the search for motion vectors can be limited to a rectangular area of the frame defined by its top left corner, width and height. These parameters have the same meaning as the drawbox filter which can be used to visualise the position of the bounding box.

This is useful when simultaneous movement of subjects within the frame might be confused for camera motion by the motion vector search.

If any or all of *x*, *y*, *w* and *h* are set to -1 then the full frame is used. This allows later options to be set without specifying the bounding box for the motion vector search.

Default - search the whole frame.

‘`rx, ry`’

Specify the maximum extent of movement in x and y directions in the range 0-64 pixels. Default 16.

‘`edge`’

Specify how to generate pixels to fill blanks at the edge of the frame. An integer from 0 to 3 as follows:

‘`0`’

Fill zeroes at blank locations

‘`1`’

Original image at blank locations

‘`2`’

Extruded edge value at blank locations

‘`3`’

Mirrored edge at blank locations

The default setting is mirror edge at blank locations.

‘`blocksize`’

Specify the blocksize to use for motion search. Range 4-128 pixels, default 8.

‘`contrast`’

Specify the contrast threshold for blocks. Only blocks with more than the specified contrast (difference between darkest and lightest pixels) will be considered. Range 1-255, default 125.

‘`search`’

Specify the search strategy 0 = exhaustive search, 1 = less exhaustive search. Default - exhaustive search.

‘`filename`’

: If set then a detailed log of the motion search is written to the specified file.

28.15 drawbox

Draw a colored box on the input image.

The filter accepts parameters as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`x, y`’

: Specify the top left corner coordinates of the box. Default to 0.

‘`width, w`’
‘`height, h`’

: Specify the width and height of the box, if 0 they are interpreted as the input width and height. Default to 0.

‘`color, c`’

: Specify the color of the box to write, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence. If the special value `invert` is used, the box edge color is the same as the video with inverted luma.

‘`thickness, t`’

: Set the thickness of the box edge. Default value is `4`.

If the key of the first options is omitted, the arguments are interpreted according to the following syntax:

```
drawbox=x:y:width:height:color:thickness
```

Some examples follow:

- Draw a black box around the edge of the input image:

  ```
  drawbox
  ```

- Draw a box with color red and an opacity of 50%:

  ```
  drawbox=10:20:200:60:red@0.5
  ```

The previous example can be specified as:

```
drawbox=x=10:y=20:w=200:h=60:color=red@0.5
```

- Fill the box with pink color:

```
drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max
```

28.16 drawtext

Draw text string or text from specified file on top of video using the libfreetype library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libfreetype`.

28.16.1 Syntax

The filter accepts parameters as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`box`’

Used to draw a box around text using background color. Value should be either 1 (enable) or 0 (disable). The default value of *box* is 0.

‘`boxcolor`’

The color to be used for drawing box around text. Either a string (e.g. "yellow") or in 0xRRGGBB[AA] format (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *boxcolor* is "white".

‘`draw`’

Set an expression which specifies if the text should be drawn. If the expression evaluates to 0, the text is not drawn. This is useful for specifying that the text should be drawn only when specific conditions are met.

Default value is "1".

See below for the list of accepted constants and functions.

‘`expansion`’

Select how the *text* is expanded. Can be either `none`, `strftime` (default for compatibity reasons but deprecated) or `normal`. See the Text expansion section below for details.

‘`fix_bounds`’

If true, check and fix text coords to avoid clipping.

‘`fontcolor`’

The color to be used for drawing fonts. Either a string (e.g. "red") or in 0xRRGGBB[AA] format (e.g. "0xff000033"), possibly followed by an alpha specifier. The default value of *fontcolor* is "black".

‘`fontfile`’

The font file to be used for drawing text. Path must be included. This parameter is mandatory.

‘`fontsize`’

The font size to be used for drawing text. The default value of *fontsize* is 16.

‘`ft_load_flags`’

Flags to be used for loading the fonts.

The flags map the corresponding flags supported by libfreetype, and are a combination of the following values:

default
no_scale
no_hinting
render
no_bitmap
vertical_layout
force_autohint
crop_bitmap
pedantic
ignore_global_advance_width
no_recurse
ignore_transform
monochrome
linear_design
no_autohint
end table

Default value is "render".

For more information consult the documentation for the FT_LOAD_* libfreetype flags.

‘`shadowcolor`’

The color to be used for drawing a shadow behind the drawn text. It can be a color name (e.g. "yellow") or a string in the 0xRRGGBB[AA] form (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *shadowcolor* is "black".

‘`shadowx, shadowy`’

The x and y offsets for the text shadow position with respect to the position of the text. They can be either positive or negative values. Default value for both is "0".

‘`tabsize`’

The size in number of spaces to use for rendering the tab. Default value is 4.

‘`timecode`’

Set the initial timecode representation in "hh:mm:ss[:;.]ff" format. It can be used with or without text parameter. *timecode_rate* option must be specified.

‘`timecode_rate, rate, r`’

Set the timecode frame rate (timecode only).

‘`text`’

The text string to be drawn. The text must be a sequence of UTF-8 encoded characters. This parameter is mandatory if no file is specified with the parameter *textfile*.

‘`textfile`’

A text file containing text to be drawn. The text must be a sequence of UTF-8 encoded characters.

This parameter is mandatory if no text string is specified with the parameter *text*.

If both *text* and *textfile* are specified, an error is thrown.

‘`x, y`’

The expressions which specify the offsets where text will be drawn within the video frame. They are relative to the top/left border of the output image.

The default value of *x* and *y* is "0".

See below for the list of accepted constants and functions.

The parameters for *x* and *y* are expressions containing the following constants and functions:

‘`dar`’

input display aspect ratio, it is the same as (*w* / *h*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘`line_h, lh`’

the height of each text line

‘`main_h, h, H`’

the input height

‘`main_w, w, W`’

the input width

‘`max_glyph_a, ascent`’

the maximum distance from the baseline to the highest/upper grid coordinate used to place a glyph outline point, for all the rendered glyphs. It is a positive value, due to the grid’s orientation with the Y axis upwards.

‘`max_glyph_d, descent`’

the maximum distance from the baseline to the lowest grid coordinate used to place a glyph outline point, for all the rendered glyphs. This is a negative value, due to the grid’s orientation, with the Y axis upwards.

‘`max_glyph_h`’

maximum glyph height, that is the maximum height for all the glyphs contained in the rendered text, it is equivalent to *ascent* - *descent*.

‘`max_glyph_w`’

maximum glyph width, that is the maximum width for all the glyphs contained in the rendered text

‘`n`’

the number of input frame, starting from 0

‘`rand(min, max)`’

return a random number included between *min* and *max*

‘`sar`’

input sample aspect ratio

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘`text_h, th`’

the height of the rendered text

‘`text_w, tw`’

the width of the rendered text

‘`x, y`’

the x and y offset coordinates where the text is drawn.

These parameters allow the *x* and *y* expressions to refer each other, so you can for example specify `y=x/dar`.

If libavfilter was built with `--enable-fontconfig`, then ‘`fontfile`’ can be a fontconfig pattern or omitted.

28.16.2 Text expansion

If ‘`expansion`’ is set to `strftime` (which is the default for now), the filter recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime(). This feature is deprecated.

If ‘`expansion`’ is set to `none`, the text is printed verbatim.

If ‘`expansion`’ is set to `normal` (which will be the default), the following expansion mechanism is used.

The backslash character ’\’, followed by any character, always expands to the second character.

Sequence of the form `%{...}` are expanded. The text between the braces is a function name, possibly followed by arguments separated by ’:’. If the arguments contain special characters or delimiters (’:’ or ’}’), they should be escaped.

Note that they probably must also be escaped as the value for the ‘`text`’ option in the filter argument string and as the filter argument in the filter graph description, and possibly also for the shell, that makes up to four levels of escaping; using a text file avoids these problems.

The following functions are available:

`gmtime`

The time at which the filter is running, expressed in UTC. It can accept an argument: a strftime() format string.

`localtime`

The time at which the filter is running, expressed in the local time zone. It can accept an argument: a strftime() format string.

`n, frame_num`

The frame number, starting from 0.

`pts`

The timestamp of the current frame, in seconds, with microsecond accuracy.

28.16.3 Examples

Some examples follow.

- Draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

  ```
  drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
  ```

- Draw 'Test Text' with font FreeSerif of size 24 at position x=100 and y=50 (counting from the top-left corner of the screen), text is yellow with a red box around it. Both the text and the box have an opacity of 20%.

  ```
  drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
            x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
  ```

  Note that the double quotes are not necessary if spaces are not used within the parameter list.

- Show the text at the center of the video frame:

  ```
  drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
  ```

- Show a text line sliding from right to left in the last row of the video frame. The file '`LONG_LINE`' is assumed to contain a single line with no newlines.

  ```
  drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
  ```

- Show the content of file ‘`CREDITS`’ off the bottom of the frame and scroll up.

  ```
  drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
  ```

- Draw a single green letter "g", at the center of the input video. The glyph baseline is placed at half screen height.

  ```
  drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=g:x=(w-max_glyph_w)/2:y=h/2-ascent"
  ```

- Show text for 1 second every 3 seconds:

  ```
  drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:draw=lt(mod(t\,3)\,1):text='blink'"
  ```

- Use fontconfig to set the font. Note that the colons need to be escaped.

  ```
  drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
  ```

- Print the date of a real-time encoding (see strftime(3)):

  ```
  drawtext='fontfile=FreeSans.ttf:expansion=normal:text=%{localtime:%a %b %d %Y}'
  ```

For more information about libfreetype, check: http://www.freetype.org/.

For more information about fontconfig, check:
http://freedesktop.org/software/fontconfig/fontconfig-user.html.

28.17 edgedetect

Detect and draw edges. The filter uses the Canny Edge Detection algorithm.

This filter accepts the following optional named parameters:

‘`low, high`’

> Set low and high threshold values used by the Canny thresholding algorithm.
>
> The high threshold selects the "strong" edge pixels, which are then connected through 8-connectivity with the "weak" edge pixels selected by the low threshold.
>
> *low* and *high* threshold values must be choosen in the range [0,1], and *low* should be lesser or equal to *high*.
>
> Default value for *low* is `20/255`, and default value for *high* is `50/255`.

Example:

```
edgedetect=low=0.1:high=0.4
```

28.18 fade

Apply fade-in/out effect to input video.

It accepts the parameters: *type*:*start_frame*:*nb_frames*[:*options*]

type specifies if the effect type, can be either "in" for fade-in, or "out" for a fade-out effect.

start_frame specifies the number of the start frame for starting to apply the fade effect.

nb_frames specifies the number of frames for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black.

options is an optional sequence of *key*=*value* pairs, separated by ":". The description of the accepted options follows.

‘`type, t`’

See *type*.

‘`start_frame, s`’

See *start_frame*.

‘`nb_frames, n`’

See *nb_frames*.

‘`alpha`’

If set to 1, fade only alpha channel, if one exists on the input. Default value is 0.

A few usage examples follow, usable too as test scenarios.

```
# fade in first 30 frames of video
fade=in:0:30

# fade out last 45 frames of a 200-frame video
fade=out:155:45

# fade in first 25 frames and fade out last 25 frames of a 1000-frame video
fade=in:0:25, fade=out:975:25

# make first 5 frames black, then fade in from frame 5-24
fade=in:5:20

# fade in alpha over first 25 frames of video
fade=in:0:25:alpha=1
```

28.19 field

Extract a single field from an interlaced image using stride arithmetic to avoid wasting CPU time. The output frames are marked as non-interlaced.

This filter accepts the following named options:

‘`type`’

> Specify whether to extract the top (if the value is `0` or `top`) or the bottom field (if the value is `1` or `bottom`).

If the option key is not specified, the first value sets the *type* option. For example:

```
field=bottom
```

is equivalent to:

```
field=type=bottom
```

28.20 fieldorder

Transform the field order of the input video.

It accepts one parameter which specifies the required field order that the input interlaced video will be transformed to. The parameter can assume one of the following values:

‘`0 or bff`’

> output bottom field first

‘`1 or tff`’

output top field first

Default value is "tff".

Transformation is achieved by shifting the picture content up or down by one line, and filling the remaining line with appropriate picture content. This method is consistent with most broadcast field order converters.

If the input video is not flagged as being interlaced, or it is already flagged as being of the required output field order then this filter does not alter the incoming video.

This filter is very useful when converting to or from PAL DV material, which is bottom field first.

For example:

```
ffmpeg -i in.vob -vf "fieldorder=bff" out.dv
```

28.21 fifo

Buffer input images and send them when they are requested.

This filter is mainly useful when auto-inserted by the libavfilter framework.

The filter does not take parameters.

28.22 format

Convert the input video to one of the specified pixel formats. Libavfilter will try to pick one that is supported for the input to the next filter.

The filter accepts a list of pixel format names, separated by ":", for example "yuv420p:monow:rgb24".

Some examples follow:

```
# convert the input video to the format "yuv420p"
format=yuv420p

# convert the input video to any of the formats in the list
format=yuv420p:yuv444p:yuv410p
```

28.23 fps

Convert the video to specified constant framerate by duplicating or dropping frames as necessary.

This filter accepts the following named parameters:

'`fps`'

Desired output framerate.

'`round`'

Rounding method. The default is `near`.

28.24 framestep

Select one frame every N.

This filter accepts in input a string representing a positive integer. Default argument is `1`.

28.25 frei0r

Apply a frei0r effect to the input video.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

The filter supports the syntax:

```
filter_name[{:|=}param1:param2:...:paramN]
```

filter_name is the name of the frei0r effect to load. If the environment variable `FREI0R_PATH` is defined, the frei0r effect is searched in each one of the directories specified by the colon (or semicolon on Windows platforms) separated list in `FREIOR_PATH`, otherwise in the standard frei0r paths, which are in this order: '`HOME/.frei0r-1/lib/`', '`/usr/local/lib/frei0r-1/`', '`/usr/lib/frei0r-1/`'.

param1, *param2*, ... , *paramN* specify the parameters for the frei0r effect.

A frei0r effect parameter can be a boolean (whose values are specified with "y" and "n"), a double, a color (specified by the syntax *R*/*G*/*B*, *R*, *G*, and *B* being float numbers from 0.0 to 1.0) or by an `av_parse_color()` color description), a position (specified by the syntax *X*/*Y*, *X* and *Y* being float numbers) and a string.

The number and kind of parameters depend on the loaded effect. If an effect parameter is not specified the default value is set.

Some examples follow:

- Apply the distort0r effect, set the first two double parameters:

  ```
  frei0r=distort0r:0.5:0.01
  ```

- Apply the colordistance effect, take a color as first parameter:

  ```
  frei0r=colordistance:0.2/0.3/0.4
  frei0r=colordistance:violet
  frei0r=colordistance:0x112233
  ```

- Apply the perspective effect, specify the top left and top right image positions:

  ```
  frei0r=perspective:0.2/0.2:0.8/0.2
  ```

For more information see: http://frei0r.dyne.org

28.26 geq

The filter takes one, two or three equations as parameter, separated by ':'. The first equation is mandatory and applies to the luma plane. The two following are respectively for chroma blue and chroma red planes.

The filter syntax allows named parameters:

'`lum_expr`'

  the luminance expression

'`cb_expr`'

  the chrominance blue expression

'`cr_expr`'

  the chrominance red expression

If one of the chrominance expression is not defined, it falls back on the other one. If none of them are specified, they will evaluate the luminance expression.

The expressions can use the following variables and functions:

‘`N`’

The sequential number of the filtered frame, starting from `0`.

‘`X, Y`’

The coordinates of the current sample.

‘`W, H`’

The width and height of the image.

‘`SW, SH`’

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

‘`p(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the current plane.

‘`lum(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the luminance plane.

‘`cb(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the blue-difference chroma plane.

‘`cr(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the red-difference chroma plane.

For functions, if *x* and *y* are outside the area, the value will be automatically clipped to the closer edge.

Some examples follow:

- Flip the image horizontally:

```
geq=p(W-X\,Y)
```

- Generate a fancy enigmatic moving light:

```
nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N*0.07)*W/2-W/2\,Y-sin(N*0.09)*H/2-H/2)^2*1000000*sin(N*0.02):128:128
```

28.27 gradfun

Fix the banding artifacts that are sometimes introduced into nearly flat regions by truncation to 8bit color depth. Interpolate the gradients that should go where the bands are, and dither them.

This filter is designed for playback only. Do not use it prior to lossy compression, because compression tends to lose the dither and bring back the bands.

The filter takes two optional parameters, separated by ’:’: *strength*:*radius*

strength is the maximum amount by which the filter will change any one pixel. Also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 255, default value is 1.2, out-of-range values will be clipped to the valid range.

radius is the neighborhood to fit the gradient to. A larger radius makes for smoother gradients, but also prevents the filter from modifying the pixels near detailed regions. Acceptable values are 8-32, default value is 16, out-of-range values will be clipped to the valid range.

```
# default parameters
gradfun=1.2:16

# omitting radius
gradfun=1.2
```

28.28 hflip

Flip the input video horizontally.

For example to horizontally flip the input video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "hflip" out.avi
```

28.29 hqdn3d

High precision/quality 3d denoise filter. This filter aims to reduce image noise producing smooth images and making still images really still. It should enhance compressibility.

It accepts the following optional parameters: *luma_spatial*:*chroma_spatial*:*luma_tmp*:*chroma_tmp*

‘`luma_spatial`’

a non-negative float number which specifies spatial luma strength, defaults to 4.0

‘`chroma_spatial`’

a non-negative float number which specifies spatial chroma strength, defaults to 3.0**luma_spatial*/4.0

‘`luma_tmp`’

a float number which specifies luma temporal strength, defaults to 6.0**luma_spatial*/4.0

‘`chroma_tmp`’

a float number which specifies chroma temporal strength, defaults to *luma_tmp***chroma_spatial*/*luma_spatial*

28.30 hue

Modify the hue and/or the saturation of the input.

This filter accepts the following optional named options:

‘`h`’

Specify the hue angle as a number of degrees. It accepts a float number or an expression, and defaults to 0.0.

‘`H`’

Specify the hue angle as a number of degrees. It accepts a float number or an expression, and defaults to 0.0.

‘`s`’

Specify the saturation in the [-10,10] range. It accepts a float number and defaults to 1.0.

The *h*, *H* and *s* parameters are expressions containing the following constants:

‘`n`’

frame count of the input frame starting from 0

‘`pts`’

presentation timestamp of the input frame expressed in time base units

‘`r`’

frame rate of the input video, NAN if the input frame rate is unknown

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘`tb`’

time base of the input video

The options can also be set using the syntax: *hue*:*saturation*

In this case *hue* is expressed in degrees.

Some examples follow:

- Set the hue to 90 degrees and the saturation to 1.0:

  ```
  hue=h=90:s=1
  ```

- Same command but expressing the hue in radians:

  ```
  hue=H=PI/2:s=1
  ```

- Same command without named options, hue must be expressed in degrees:

  ```
  hue=90:1
  ```

- Note that "h:s" syntax does not support expressions for the values of h and s, so the following example will issue an error:

  ```
  hue=PI/2:1
  ```

- Rotate hue and make the saturation swing between 0 and 2 over a period of 1 second:

  ```
  hue="H=2*PI*t: s=sin(2*PI*t)+1"
  ```

- Apply a 3 seconds saturation fade-in effect starting at 0:

  ```
  hue="s=min(t/3\,1)"
  ```

  The general fade-in expression can be written as:

  ```
  hue="s=min(0\, max((t-START)/DURATION\, 1))"
  ```

- Apply a 3 seconds saturation fade-out effect starting at 5 seconds:

```
hue="s=max(0\, min(1\, (8-t)/3))"
```

The general fade-out expression can be written as:

```
hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
```

28.30.1 Commands

This filter supports the following command:

‘`reinit`’

Modify the hue and/or the saturation of the input video. The command accepts the same named options and syntax than when calling the filter from the command-line.

If a parameter is omitted, it is kept at its current value.

28.31 idet

Interlaceing detect filter. This filter tries to detect if the input is interlaced or progressive. Top or bottom field first.

28.32 lut, lutrgb, lutyuv

Compute a look-up table for binding each pixel component input value to an output value, and apply it to input video.

lutyuv applies a lookup table to a YUV input video, *lutrgb* to an RGB input video.

These filters accept in input a ":"-separated list of options, which specify the expressions used for computing the lookup table for the corresponding pixel component values.

The *lut* filter requires either YUV or RGB pixel formats in input, and accepts the options:

‘*c0* `(first pixel component)`’
‘*c1* `(second pixel component)`’
‘*c2* `(third pixel component)`’
‘*c3* `(fourth pixel component, corresponds to the alpha component)`’

The exact component associated to each option depends on the format in input.

The *lutrgb* filter requires RGB pixel formats in input, and accepts the options:

‘*r* `(red component)`’

‘*g* (green component)’
‘*b* (blue component)’
‘*a* (alpha component)’

The *lutyuv* filter requires YUV pixel formats in input, and accepts the options:

‘*y* (Y/luminance component)’
‘*u* (U/Cb component)’
‘*v* (V/Cr component)’
‘*a* (alpha component)’

The expressions can contain the following constants and functions:

‘`w, h`’

the input width and height

‘`val`’

input value for the pixel component

‘`clipval`’

the input value clipped in the *minval*-*maxval* range

‘`maxval`’

maximum value for the pixel component

‘`minval`’

minimum value for the pixel component

‘`negval`’

the negated value for the pixel component value clipped in the *minval*-*maxval* range , it corresponds to the expression "maxval-clipval+minval"

‘`clip(val)`’

the computed value in *val* clipped in the *minval*-*maxval* range

‘`gammaval(gamma)`’

the computed gamma correction value of the pixel component value clipped in the *minval*-*maxval* range, corresponds to the expression "pow((clipval-minval)/(maxval-minval)\,*gamma*)*(maxval-minval)+minval"

All expressions default to "val".

Some examples follow:

```
# negate input video
lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val"
lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"

# the above is the same as
lutrgb="r=negval:g=negval:b=negval"
lutyuv="y=negval:u=negval:v=negval"

# negate luminance
lutyuv=y=negval

# remove chroma components, turns the video into a graytone image
lutyuv="u=128:v=128"

# apply a luma burning effect
lutyuv="y=2*val"

# remove green and blue components
lutrgb="g=0:b=0"

# set a constant alpha channel value on input
format=rgba,lutrgb=a="maxval-minval/2"

# correct luminance gamma by a 0.5 factor
lutyuv=y=gammaval(0.5)
```

28.33 mp

Apply an MPlayer filter to the input video.

This filter provides a wrapper around most of the filters of MPlayer/MEncoder.

This wrapper is considered experimental. Some of the wrapped filters may not work properly and we may drop support for them, as they will be implemented natively into FFmpeg. Thus you should avoid depending on them when writing portable scripts.

The filters accepts the parameters: *filter_name*[:=]*filter_params*

filter_name is the name of a supported MPlayer filter, *filter_params* is a string containing the parameters accepted by the named filter.

The list of the currently supported filters follows:

denoise3d
detc

dint
divtc
down3dright
dsize
eq2
eq
fil
fspp
harddup
il
ilpack
ivtc
kerndeint
mcdeint
noise
ow
perspective
phase
pp7
pullup
qp
sab
softpulldown
softskip
spp
telecine
tinterlace
unsharp
uspp

The parameter syntax and behavior for the listed filters are the same of the corresponding MPlayer filters. For detailed instructions check the "VIDEO FILTERS" section in the MPlayer manual.

Some examples follow:

- Adjust gamma, brightness, contrast:

  ```
  mp=eq2=1.0:2:0.5
  ```

- Add temporal noise to input video:

  ```
  mp=noise=20t
  ```

See also mplayer(1), http://www.mplayerhq.hu/.

28.34 negate

Negate input video.

This filter accepts an integer in input, if non-zero it negates the alpha component (if available). The default value in input is 0.

28.35 noformat

Force libavfilter not to use any of the specified pixel formats for the input to the next filter.

The filter accepts a list of pixel format names, separated by ":", for example "yuv420p:monow:rgb24".

Some examples follow:

```
# force libavfilter to use a format different from "yuv420p" for the
# input to the vflip filter
noformat=yuv420p,vflip

# convert the input video to any of the formats not contained in the list
noformat=yuv420p:yuv444p:yuv410p
```

28.36 null

Pass the video source unchanged to the output.

28.37 ocv

Apply video transform using libopencv.

To enable this filter install libopencv library and headers and configure FFmpeg with `--enable-libopencv`.

The filter takes the parameters: *filter_name*{:=}*filter_params*.

filter_name is the name of the libopencv filter to apply.

filter_params specifies the parameters to pass to the libopencv filter. If not specified the default values are assumed.

Refer to the official libopencv documentation for more precise information: http://opencv.willowgarage.com/documentation/c/image_filtering.html

Follows the list of supported libopencv filters.

28.37.1 dilate

Dilate an image by using a specific structuring element. This filter corresponds to the libopencv function `cvDilate`.

It accepts the parameters: *struct_el*:*nb_iterations*.

struct_el represents a structuring element, and has the syntax: *cols*x*rows*+*anchor_x*x*anchor_y*/*shape*

cols and *rows* represent the number of columns and rows of the structuring element, *anchor_x* and *anchor_y* the anchor point, and *shape* the shape for the structuring element, and can be one of the values "rect", "cross", "ellipse", "custom".

If the value for *shape* is "custom", it must be followed by a string of the form "=*filename*". The file with name *filename* is assumed to represent a binary image, with each printable character corresponding to a bright pixel. When a custom *shape* is used, *cols* and *rows* are ignored, the number or columns and rows of the read file are assumed instead.

The default value for *struct_el* is "3x3+0x0/rect".

nb_iterations specifies the number of times the transform is applied to the image, and defaults to 1.

Follow some example:

```
# use the default values
ocv=dilate

# dilate using a structuring element with a 5x5 cross, iterate two times
ocv=dilate=5x5+2x2/cross:2

# read the shape from the file diamond.shape, iterate two times
# the file diamond.shape may contain a pattern of characters like this:
#   *
#  ***
# *****
#  ***
#   *
# the specified cols and rows are ignored (but not the anchor point coordinates)
ocv=0x0+2x2/custom=diamond.shape:2
```

28.37.2 erode

Erode an image by using a specific structuring element. This filter corresponds to the libopencv function `cvErode`.

The filter accepts the parameters: *struct_el*:*nb_iterations*, with the same syntax and semantics as the dilate filter.

28.37.3 smooth

Smooth the input video.

The filter takes the following parameters: *type*:*param1*:*param2*:*param3*:*param4*.

type is the type of smooth filter to apply, and can be one of the following values: "blur", "blur_no_scale", "median", "gaussian", "bilateral". The default value is "gaussian".

param1, *param2*, *param3*, and *param4* are parameters whose meanings depend on smooth type. *param1* and *param2* accept integer positive values or 0, *param3* and *param4* accept float values.

The default value for *param1* is 3, the default value for the other parameters is 0.

These parameters correspond to the parameters assigned to the libopencv function `cvSmooth`.

28.38 overlay

Overlay one video on top of another.

It takes two inputs and one output, the first input is the "main" video on which the second input is overlayed.

It accepts the parameters: *x*:*y*[:*options*].

x is the x coordinate of the overlayed video on the main video, *y* is the y coordinate. *x* and *y* are expressions containing the following parameters:

‘`main_w, main_h`’

    main input width and height

‘`W, H`’

    same as *main_w* and *main_h*

‘`overlay_w, overlay_h`’

    overlay input width and height

‘`w, h`’

    same as *overlay_w* and *overlay_h*

options is an optional list of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`rgb`’

If set to 1, force the filter to accept inputs in the RGB color space. Default value is 0.

Be aware that frames are taken from each input video in timestamp order, hence, if their initial timestamps differ, it is a a good idea to pass the two inputs through a *setpts=PTS-STARTPTS* filter to have them begin in the same zero timestamp, as it does the example for the *movie* filter.

Follow some examples:

```
# draw the overlay at 10 pixels from the bottom right
# corner of the main video.
overlay=main_w-overlay_w-10:main_h-overlay_h-10

# insert a transparent PNG logo in the bottom left corner of the input
ffmpeg -i input -i logo -filter_complex ’overlay=10:main_h-overlay_h-10’ output

# insert 2 different transparent PNG logos (second logo on bottom
# right corner):
ffmpeg -i input -i logo1 -i logo2 -filter_complex
’overlay=10:H-h-10,overlay=W-w-10:H-h-10’ output

# add a transparent color layer on top of the main video,
# WxH specifies the size of the main input to the overlay filter
color=red@.3:WxH [over]; [in][over] overlay [out]

# play an original video and a filtered version (here with the deshake filter)
# side by side
ffplay input.avi -vf ’split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w’

# the previous example is the same as:
ffplay input.avi -vf ’split[b], pad=iw*2[src], [b]deshake, [src]overlay=w’
```

You can chain together more overlays but the efficiency of such approach is yet to be tested.

28.39 pad

Add paddings to the input image, and places the original input at the given coordinates *x*, *y*.

It accepts the following parameters: *width*:*height*:*x*:*y*:*color*.

The parameters *width*, *height*, *x*, and *y* are expressions containing the following constants:

‘`in_w, in_h`’

the input video width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output width and height, that is the size of the padded area as specified by the *width* and *height* expressions

‘`ow, oh`’

same as *out_w* and *out_h*

‘`x, y`’

x and y offsets as specified by the *x* and *y* expressions, or NAN if not yet specified

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

Follows the description of the accepted parameters.

‘`width, height`’

Specify the size of the output image with the paddings added. If the value for *width* or *height* is 0, the corresponding input size is used for the output.

The *width* expression can reference the value set by the *height* expression, and vice versa.

The default value of *width* and *height* is 0.

‘`x, y`’

Specify the offsets where to place the input image in the padded area with respect to the top/left border of the output image.

The *x* expression can reference the value set by the *y* expression, and vice versa.

The default value of *x* and *y* is 0.

‘`color`’

Specify the color of the padded area, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence.

The default value of *color* is "black".

28.39.1 Examples

- Add paddings with color "violet" to the input video. Output video size is 640x480, the top-left corner of the input video is placed at column 0, row 40:

  ```
  pad=640:480:0:40:violet
  ```

- Pad the input to get an output with dimensions increased by 3/2, and put the input video at the center of the padded area:

  ```
  pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a squared output with size equal to the maximum value between the input width and height, and put the input video at the center of the padded area:

  ```
  pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a final w/h ratio of 16:9:

  ```
  pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- In case of anamorphic video, in order to set the output display aspect correctly, it is necessary to use *sar* in the expression, according to the relation:

  ```
  (ih * X / ih) * sar = output_dar
  X = output_dar / sar
  ```

  Thus the previous example needs to be modified to:

  ```
  pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Double output size and put the input video in the bottom-right corner of the output padded area:

  ```
  pad="2*iw:2*ih:ow-iw:oh-ih"
  ```

28.40 pixdesctest

Pixel format descriptor test filter, mainly useful for internal testing. The output video should be equal to the input video.

For example:

```
format=monow, pixdesctest
```

can be used to test the monowhite pixel format descriptor definition.

28.41 removelogo

Suppress a TV station logo, using an image file to determine which pixels comprise the logo. It works by filling in the pixels that comprise the logo with neighboring pixels.

This filter requires one argument which specifies the filter bitmap file, which can be any image format supported by libavformat. The width and height of the image file must match those of the video stream being processed.

Pixels in the provided bitmap image with a value of zero are not considered part of the logo, non-zero pixels are considered part of the logo. If you use white (255) for the logo and black (0) for the rest, you will be safe. For making the filter bitmap, it is recommended to take a screen capture of a black frame with the logo visible, and then using a threshold filter followed by the erode filter once or twice.

If needed, little splotches can be fixed manually. Remember that if logo pixels are not covered, the filter quality will be much reduced. Marking too many pixels as part of the logo does not hurt as much, but it will increase the amount of blurring needed to cover over the image and will destroy more information than necessary, and extra pixels will slow things down on a large logo.

28.42 scale

Scale (resize) the input video, using the libswscale library.

The scale filter forces the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

This filter accepts a list of named options in the form of *key*=*value* pairs separated by ":". If the key for the first two options is not specified, the assumed keys for the first two values are `w` and `h`. If the first option has no key and can be interpreted like a video size specification, it will be used to set the video size.

A description of the accepted options follows.

‘`width, w`’

Set the video width expression, default value is `iw`. See below for the list of accepted constants.

‘`height, h`’

Set the video heiht expression, default value is `ih`. See below for the list of accepted constants.

‘`interl`’

Set the interlacing. It accepts the following values:

‘`1`’

force interlaced aware scaling

‘`0`’

do not apply interlaced scaling

‘`-1`’

select interlaced aware scaling depending on whether the source frames are flagged as interlaced or not

Default value is `0`.

‘`flags`’

Set libswscale scaling flags. If not explictly specified the filter applies a bilinear scaling algorithm.

‘`size, s`’

Set the video size, the value must be a valid abbreviation or in the form *width*x*height*.

The values of the *w* and *h* options are expressions containing the following constants:

‘`in_w, in_h`’

the input width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

If the input image format is different from the format requested by the next filter, the scale filter will convert the input to the requested format.

If the value for *width* or *height* is 0, the respective input size is used for the output.

If the value for *width* or *height* is -1, the scale filter will use, for the respective output size, a value that maintains the aspect ratio of the input image.

28.42.1 Examples

- Scale the input video to a size of 200x100:

  ```
  scale=200:100
  ```

  This is equivalent to:

  ```
  scale=w=200:h=100
  ```

  or:

  ```
  scale=200x100
  ```

- Specify a size abbreviation for the output size:

  ```
  scale=qcif
  ```

which can also be written as:

```
scale=size=qcif
```

- Scale the input to 2x:

```
scale=2*iw:2*ih
```

- The above is the same as:

```
scale=2*in_w:2*in_h
```

- Scale the input to 2x with forced interlaced scaling:

```
scale=2*iw:2*ih:interl=1
```

- Scale the input to half size:

```
scale=iw/2:ih/2
```

- Increase the width, and set the height to the same size:

```
scale=3/2*iw:ow
```

- Seek for Greek harmony:

```
scale=iw:1/PHI*iw
scale=ih*PHI:ih
```

- Increase the height, and set the width to 3/2 of the height:

```
scale=3/2*oh:3/5*ih
```

- Increase the size, but make the size a multiple of the chroma:

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- Increase the width to a maximum of 500 pixels, keep the same input aspect ratio:

```
scale='min(500\, iw*3/2):-1'
```

28.43 select

Select frames to pass in output.

It accepts in input an expression, which is evaluated for each input frame. If the expression is evaluated to a non-zero value, the frame is selected and passed to the output, otherwise it is discarded.

The expression can contain the following constants:

‘`n`’

the sequential number of the filtered frame, starting from 0

‘`selected_n`’

the sequential number of the selected frame, starting from 0

‘`prev_selected_n`’

the sequential number of the last selected frame, NAN if undefined

‘`TB`’

timebase of the input timestamps

‘`pts`’

the PTS (Presentation TimeStamp) of the filtered video frame, expressed in *TB* units, NAN if undefined

‘`t`’

the PTS (Presentation TimeStamp) of the filtered video frame, expressed in seconds, NAN if undefined

‘`prev_pts`’

the PTS of the previously filtered video frame, NAN if undefined

‘`prev_selected_pts`’

the PTS of the last previously filtered video frame, NAN if undefined

‘`prev_selected_t`’

the PTS of the last previously selected video frame, NAN if undefined

‘`start_pts`’

the PTS of the first video frame in the video, NAN if undefined

‘`start_t`’

the time of the first video frame in the video, NAN if undefined

‘`pict_type`’

the type of the filtered frame, can assume one of the following values:

‘`I`’
‘`P`’
‘`B`’
‘`S`’
‘`SI`’
‘`SP`’
‘`BI`’

‘`interlace_type`’

the frame interlace type, can assume one of the following values:

‘`PROGRESSIVE`’

the frame is progressive (not interlaced)

‘`TOPFIRST`’

the frame is top-field-first

‘`BOTTOMFIRST`’

the frame is bottom-field-first

‘`key`’

1 if the filtered frame is a key-frame, 0 otherwise

‘`pos`’

the position in the file of the filtered frame, -1 if the information is not available (e.g. for synthetic video)

‘`scene`’

value between 0 and 1 to indicate a new scene; a low value reflects a low probability for the current frame to introduce a new scene, while a higher value means the current frame is more likely to be one (see the example below)

The default value of the select expression is "1".

Some examples follow:

```
# select all frames in input
select

# the above is the same as:
select=1

# skip all frames:
select=0

# select only I-frames
select='eq(pict_type\,I)'

# select one frame every 100
select='not(mod(n\,100))'

# select only frames contained in the 10-20 time interval
select='gte(t\,10)*lte(t\,20)'

# select only I frames contained in the 10-20 time interval
select='gte(t\,10)*lte(t\,20)*eq(pict_type\,I)'

# select frames with a minimum distance of 10 seconds
select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
```

Complete example to create a mosaic of the first scenes:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

Comparing *scene* against a value between 0.3 and 0.5 is generally a sane choice.

28.44 setdar, setsar

The `setdar` filter sets the Display Aspect Ratio for the filter output video.

This is done by changing the specified Sample (aka Pixel) Aspect Ratio, according to the following equation:

```
DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR
```

Keep in mind that the `setdar` filter does not modify the pixel dimensions of the video frame. Also the display aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. in case of scaling or if another "setdar" or a "setsar" filter is applied.

The `setsar` filter sets the Sample (aka Pixel) Aspect Ratio for the filter output video.

Note that as a consequence of the application of this filter, the output display aspect ratio will change according to the equation above.

Keep in mind that the sample aspect ratio set by the `setsar` filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

The `setdar` and `setsar` filters accept a string in the form *num*:*den* expressing an aspect ratio, or the following named options, expressed as a sequence of *key*=*value* pairs, separated by ":".

‘`max`’

> Set the maximum integer value to use for expressing numerator and denominator when reducing the expressed aspect ratio to a rational. Default value is `100`.

‘`r, ratio:`’

> Set the aspect ratio used by the filter.
>
> The parameter can be a floating point number string, an expression, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0". In case the form "*num*:*den*" the `:` character should be escaped.

If the keys are omitted in the named options list, the specifed values are assumed to be *ratio* and *max* in that order.

For example to change the display aspect ratio to 16:9, specify:

```
setdar='16:9'
```

The example above is equivalent to:

```
setdar=1.77777
```

To change the sample aspect ratio to 10:11, specify:

```
setsar='10:11'
```

To set a display aspect ratio of 16:9, and specify a maximum integer value of 1000 in the aspect ratio reduction, use the command:

```
setdar=ratio='16:9':max=1000
```

28.45 setfield

Force field for the output video frame.

The `setfield` filter marks the interlace type field for the output frames. It does not change the input frame, but only sets the corresponding property, which affects how the frame is treated by following filters (e.g. `fieldorder` or `yadif`).

It accepts a string parameter, which can assume the following values:

‘`auto`’

    Keep the same field property.

‘`bff`’

    Mark the frame as bottom-field-first.

‘`tff`’

    Mark the frame as top-field-first.

‘`prog`’

    Mark the frame as progressive.

28.46 showinfo

Show a line containing various information for each input video frame. The input video is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

A description of each shown parameter follows:

‘`n`’

    sequential number of the input frame, starting from 0

‘`pts`’

    Presentation TimeStamp of the input frame, expressed as a number of time base units. The time base unit depends on the filter input pad.

‘`pts_time`’

    Presentation TimeStamp of the input frame, expressed as a number of seconds

‘`pos`’

: position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic video)

‘`fmt`’

: pixel format name

‘`sar`’

: sample aspect ratio of the input frame, expressed in the form *num*/*den*

‘`s`’

: size of the input frame, expressed in the form *width*x*height*

‘`i`’

: interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first)

‘`iskey`’

: 1 if the frame is a key frame, 0 otherwise

‘`type`’

: picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, "?" for unknown type). Check also the documentation of the `AVPictureType` enum and of the `av_get_picture_type_char` function defined in ‘`libavutil/avutil.h`’.

‘`checksum`’

: Adler-32 checksum (printed in hexadecimal) of all the planes of the input frame

‘`plane_checksum`’

: Adler-32 checksum (printed in hexadecimal) of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]"

28.47 slicify

Pass the images of input video on to next video filter as multiple slices.

```
ffmpeg -i in.avi -vf "slicify=32" out.avi
```

The filter accepts the slice height as parameter. If the parameter is not specified it will use the default value of 16.

Adding this in the beginning of filter chains should make filtering faster due to better use of the memory cache.

28.48 smartblur

Blur the input video without impacting the outlines.

The filter accepts the following parameters:
luma_radius:*luma_strength*:*luma_threshold*[:*chroma_radius*:*chroma_strength*:*chroma_threshold*]

Parameters prefixed by *luma* indicate that they work on the luminance of the pixels whereas parameters prefixed by *chroma* refer to the chrominance of the pixels.

If the chroma parameters are not set, the luma parameters are used for either the luminance and the chrominance of the pixels.

luma_radius or *chroma_radius* must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger).

luma_strength or *chroma_strength* must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image.

luma_threshold or *chroma_threshold* must be an integer in the range [-30,30] that is used as a coefficient to determine whether a pixel should be blurred or not. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges.

28.49 split

Split input video into several identical outputs.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

For example

```
ffmpeg -i INPUT -filter_complex split=5 OUTPUT
```

will create 5 copies of the input video.

For example:

```
[in] split [splitout1][splitout2];
[splitout1] crop=100:100:0:0    [cropout];
[splitout2] pad=200:200:100:100 [padout];
```

will create two separate outputs from the same input, one cropped and one padded.

28.50 super2xsai

Scale the input by 2x and smooth using the Super2xSaI (Scale and Interpolate) pixel art scaling algorithm.

Useful for enlarging pixel art images without reducing sharpness.

28.51 swapuv

Swap U & V plane.

28.52 thumbnail

Select the most representative frame in a given sequence of consecutive frames.

It accepts as argument the frames batch size to analyze (default *N*=100); in a set of *N* frames, the filter will pick one of them, and then handle the next batch of *N* frames until the end.

Since the filter keeps track of the whole frames sequence, a bigger *N* value will result in a higher memory usage, so a high value is not recommended.

The following example extract one picture each 50 frames:

```
thumbnail=50
```

Complete example of a thumbnail creation with `ffmpeg`:

```
ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
```

28.53 tile

Tile several successive frames together.

It accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`layout`’

Set the grid size (i.e. the number of lines and columns) in the form "*w*x*h*".

‘`margin`’

Set the outer border margin in pixels.

‘`padding`’

Set the inner border thickness (i.e. the number of pixels between frames). For more advanced padding options (such as having different values for the edges), refer to the pad video filter.

‘`nb_frames`’

Set the maximum number of frames to render in the given area. It must be less than or equal to *w*x*h*. The default value is `0`, meaning all the area will be used.

Alternatively, the options can be specified as a flat string:

layout[:*nb_frames*[:*margin*[:*padding*]]]

For example, produce 8Ã8 PNG tiles of all keyframes (‘`-skip_frame nokey`’) in a movie:

```
ffmpeg -skip_frame nokey -i file.avi -vf 'scale=128:72,tile=8x8' -an -vsync 0 keyframes%03d.png
```

The ‘`-vsync 0`’ is necessary to prevent `ffmpeg` from duplicating each output frame to accomodate the originally detected frame rate.

Another example to display `5` pictures in an area of `3x2` frames, with `7` pixels between them, and `2` pixels of initial margin, using mixed flat and named options:

```
tile=3x2:nb_frames=5:padding=7:margin=2
```

28.54 tinterlace

Perform various types of temporal field interlacing.

Frames are counted starting from 1, so the first input frame is considered odd.

This filter accepts a single parameter specifying the mode. Available modes are:

‘`merge, 0`’

Move odd frames into the upper field, even into the lower field, generating a double height frame at half framerate.

‘`drop_odd, 1`’

Only output even frames, odd frames are dropped, generating a frame with unchanged height at half framerate.

‘`drop_even, 2`’

Only output odd frames, even frames are dropped, generating a frame with unchanged height at half framerate.

‘`pad, 3`’

Expand each frame to full height, but pad alternate lines with black, generating a frame with double height at the same input framerate.

‘`interleave_top, 4`’

Interleave the upper field from odd frames with the lower field from even frames, generating a frame with unchanged height at half framerate.

‘`interleave_bottom, 5`’

Interleave the lower field from odd frames with the upper field from even frames, generating a frame with unchanged height at half framerate.

‘`interlacex2, 6`’

Double frame rate with unchanged height. Frames are inserted each containing the second temporal field from the previous input frame and the first temporal field from the next input frame. This mode relies on the top_field_first flag. Useful for interlaced video displays with no field synchronisation.

Numeric values are deprecated but are accepted for backward compatibility reasons.

Default mode is `merge`.

28.55 transpose

Transpose rows with columns in the input video and optionally flip it.

This filter accepts the following named parameters:

‘`dir`’

Specify the transposition direction. Can assume the following values:

‘`0, 4`’

Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

‘1, 5’

Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

‘2, 6’

Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

‘3, 7’

Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

For values between 4-7, the transposition is only done if the input video geometry is portrait and not landscape. These values are deprecated, the `passthrough` option should be used instead.

‘passthrough’

Do not apply the transposition if the input geometry matches the one specified by the specified value. It accepts the following values:

‘none’

Always apply transposition.

‘portrait’

Preserve portrait geometry (when *height* >= *width*).

‘landscape’

Preserve landscape geometry (when *width* >= *height*).

Default value is `none`.

28.56 unsharp

Sharpen or blur the input video.

It accepts the following parameters:
luma_msize_x:*luma_msize_y*:*luma_amount*:*chroma_msize_x*:*chroma_msize_y*:*chroma_amount*

Negative values for the amount will blur the input video, while positive values will sharpen. All parameters are optional and default to the equivalent of the string ’5:5:1.0:5:5:0.0’.

‘`luma_msize_x`’

Set the luma matrix horizontal size. It can be an integer between 3 and 13, default value is 5.

‘`luma_msize_y`’

Set the luma matrix vertical size. It can be an integer between 3 and 13, default value is 5.

‘`luma_amount`’

Set the luma effect strength. It can be a float number between -2.0 and 5.0, default value is 1.0.

‘`chroma_msize_x`’

Set the chroma matrix horizontal size. It can be an integer between 3 and 13, default value is 5.

‘`chroma_msize_y`’

Set the chroma matrix vertical size. It can be an integer between 3 and 13, default value is 5.

‘`chroma_amount`’

Set the chroma effect strength. It can be a float number between -2.0 and 5.0, default value is 0.0.

```
# Strong luma sharpen effect parameters
unsharp=7:7:2.5

# Strong blur of both luma and chroma parameters
unsharp=7:7:-2:7:7:-2

# Use the default values with ffmpeg
ffmpeg -i in.avi -vf "unsharp" out.mp4
```

28.57 vflip

Flip the input video vertically.

```
ffmpeg -i in.avi -vf "vflip" out.avi
```

28.58 yadif

Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

It accepts the optional parameters: *mode*:*parity*:*auto*.

mode specifies the interlacing mode to adopt, accepts one of the following values:

‘`0`’

    output 1 frame for each frame

‘`1`’

    output 1 frame for each field

‘`2`’

    like 0 but skips spatial interlacing check

‘`3`’

    like 1 but skips spatial interlacing check

Default value is 0.

parity specifies the picture field parity assumed for the input interlaced video, accepts one of the following values:

‘`0`’

    assume top field first

‘`1`’

    assume bottom field first

‘`-1`’

    enable automatic detection

Default value is -1. If interlacing is unknown or decoder does not export this information, top field first will be assumed.

auto specifies if deinterlacer should trust the interlaced flag and only deinterlace frames marked as interlaced

‘`0`’

deinterlace all frames

‘`1`’

only deinterlace frames marked as interlaced

Default value is 0.

29. Video Sources

Below is a description of the currently available video sources.

29.1 buffer

Buffer video frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in ‘`libavfilter/vsrc_buffer.h`’.

It accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`video_size`’

Specify the size (width and height) of the buffered video frames.

‘`pix_fmt`’

A string representing the pixel format of the buffered video frames. It may be a number corresponding to a pixel format, or a pixel format name.

‘`time_base`’

Specify the timebase assumed by the timestamps of the buffered frames.

‘`time_base`’

Specify the frame rate expected for the video stream.

‘`pixel_aspect`’

Specify the sample aspect ratio assumed by the video frames.

‘`sws_param`’

Specify the optional parameters to be used for the scale filter which is automatically inserted when an input change is detected in the input size or format.

For example:

```
buffer=size=320x240:pix_fmt=yuv410p:time_base=1/24:pixel_aspect=1/1
```

will instruct the source to accept video frames with size 320x240 and with format "yuv410p", assuming 1/24 as the timestamps timebase and square pixels (1:1 sample aspect ratio). Since the pixel format with name "yuv410p" corresponds to the number 6 (check the enum AVPixelFormat definition in '`libavutil/pixfmt.h`'), this example corresponds to:

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

Alternatively, the options can be specified as a flat string, but this syntax is deprecated:

width:*height*:*pix_fmt*:*time_base.num*:*time_base.den*:*pixel_aspect.num*:*pixel_aspect.den*[:*sws_param*]

29.2 cellauto

Create a pattern generated by an elementary cellular automaton.

The initial state of the cellular automaton can be defined through the '`filename`', and '`pattern`' options. If such options are not specified an initial state is created randomly.

At each new frame a new row in the video is filled with the result of the cellular automaton next generation. The behavior when the whole frame is filled is defined by the '`scroll`' option.

This source accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

'`filename, f`'

Read the initial cellular automaton state, i.e. the starting row, from the specified file. In the file, each non-whitespace character is considered an alive cell, a newline will terminate the row, and further characters in the file will be ignored.

'`pattern, p`'

Read the initial cellular automaton state, i.e. the starting row, from the specified string.

Each non-whitespace character in the string is considered an alive cell, a newline will terminate the row, and further characters in the string will be ignored.

'`rate, r`'

Set the video rate, that is the number of frames generated per second. Default is 25.

‘`random_fill_ratio, ratio`’

Set the random fill ratio for the initial cellular automaton row. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI.

This option is ignored when a file or a pattern is specified.

‘`random_seed, seed`’

Set the seed for filling randomly the initial row, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

‘`rule`’

Set the cellular automaton rule, it is a number ranging from 0 to 255. Default value is 110.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ or ‘`pattern`’ is specified, the size is set by default to the width of the specified initial state row, and the height is set to *width* * PHI.

If ‘`size`’ is set, it must contain the width of the specified pattern string, and the specified pattern will be centered in the larger row.

If a filename or a pattern string is not specified, the size value defaults to "320x518" (used for a randomly generated initial state).

‘`scroll`’

If set to 1, scroll the output upward when all the rows in the output have been already filled. If set to 0, the new generated row will be written over the top row just after the bottom row is filled. Defaults to 1.

‘`start_full, full`’

If set to 1, completely fill the output with generated rows before outputting the first frame. This is the default behavior, for disabling set the value to 0.

‘`stitch`’

If set to 1, stitch the left and right row edges together. This is the default behavior, for disabling set the value to 0.

29.2.1 Examples

- Read the initial state from ‘`pattern`’, and specify an output of size 200x400.

  ```
  cellauto=f=pattern:s=200x400
  ```

- Generate a random initial row with a width of 200 cells, with a fill ratio of 2/3:

  ```
  cellauto=ratio=2/3:s=200x200
  ```

- Create a pattern generated by rule 18 starting by a single alive cell centered on an initial row with width 100:

  ```
  cellauto=p=@:s=100x400:full=0:rule=18
  ```

- Specify a more elaborated initial pattern:

  ```
  cellauto=p='@@ @ @@':s=100x400:full=0:rule=18
  ```

29.3 mandelbrot

Generate a Mandelbrot set fractal, and progressively zoom towards the point specified with *start_x* and *start_y*.

This source accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`end_pts`’

  Set the terminal pts value. Default value is 400.

‘`end_scale`’

  Set the terminal scale value. Must be a floating point value. Default value is 0.3.

‘`inner`’

  Set the inner coloring mode, that is the algorithm used to draw the Mandelbrot fractal internal region.

  It shall assume one of the following values:

  ‘`black`’

Set black mode.

‘`convergence`’

Show time until convergence.

‘`mincol`’

Set color based on point closest to the origin of the iterations.

‘`period`’

Set period mode.

Default value is *mincol*.

‘`bailout`’

Set the bailout value. Default value is 10.0.

‘`maxiter`’

Set the maximum of iterations performed by the rendering algorithm. Default value is 7189.

‘`outer`’

Set outer coloring mode. It shall assume one of following values:

‘`iteration_count`’

Set iteration cound mode.

‘`normalized_iteration_count`’

set normalized iteration count mode.

Default value is *normalized_iteration_count*.

‘`rate, r`’

Set frame rate, expressed as number of frames per second. Default value is "25".

‘`size, s`’

Set frame size. Default value is "640x480".

‘`start_scale`’

Set the initial scale value. Default value is 3.0.

‘`start_x`’

Set the initial x position. Must be a floating point value between -100 and 100. Default value is -0.743643887037158704752191506114774.

‘`start_y`’

Set the initial y position. Must be a floating point value between -100 and 100. Default value is -0.131825904205311970493132056385139.

29.4 mptestsrc

Generate various test patterns, as generated by the MPlayer test filter.

The size of the generated video is fixed, and is 256x256. This source is useful in particular for testing encoding features.

This source accepts an optional sequence of *key*=*value* pairs, separated by ":". The description of the accepted options follows.

‘`rate, r`’

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

‘`duration, d`’

Set the video duration of the sourced video. The accepted syntax is:

```
[-]HH:MM:SS[.m...]
[-]S+[.m...]
```

See also the function `av_parse_time()`.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

‘`test, t`’

Set the number or the name of the test to perform. Supported tests are:

‘`dc_luma`’
‘`dc_chroma`’
‘`freq_luma`’

‘`freq_chroma`’
‘`amp_luma`’
‘`amp_chroma`’
‘`cbp`’
‘`mv`’
‘`ring1`’
‘`ring2`’
‘`all`’

Default value is "all", which will cycle through the list of all tests.

For example the following:

```
testsrc=t=dc_luma
```

will generate a "dc_luma" test pattern.

29.5 frei0r_src

Provide a frei0r source.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

The source supports the syntax:

```
size:rate:src_name[{=|:}param1:param2:...:paramN]
```

size is the size of the video to generate, may be a string of the form *width*x*height* or a frame size abbreviation. *rate* is the rate of the video to generate, may be a string of the form *num*/*den* or a frame rate abbreviation. *src_name* is the name to the frei0r source to load. For more information regarding frei0r and how to set the parameters read the section frei0r in the description of the video filters.

For example, to generate a frei0r partik0l source with size 200x200 and frame rate 10 which is overlayed on the overlay filter main input:

```
frei0r_src=200x200:10:partik0l=1234 [overlay]; [in][overlay] overlay
```

29.6 life

Generate a life pattern.

This source is based on a generalization of John Conway’s life game.

The sourced input represents a life grid, each pixel represents a cell which can be in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.

At each interaction the grid evolves according to the adopted rule, which specifies the number of neighbor alive cells which will make a cell stay alive or born. The '`rule`' option allows to specify the rule to adopt.

This source accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

'`filename, f`'

> Set the file from which to read the initial grid state. In the file, each non-whitespace character is considered an alive cell, and newline is used to delimit the end of each row.
>
> If this option is not specified, the initial grid is generated randomly.

'`rate, r`'

> Set the video rate, that is the number of frames generated per second. Default is 25.

'`random_fill_ratio, ratio`'

> Set the random fill ratio for the initial random grid. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI. It is ignored when a file is specified.

'`random_seed, seed`'

> Set the seed for filling the initial random grid, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

'`rule`'

> Set the life rule.
>
> A rule can be specified with a code of the kind "S*NS*/B*NB*", where *NS* and *NB* are sequences of numbers in the range 0-8, *NS* specifies the number of alive neighbor cells which make a live cell stay alive, and *NB* the number of alive neighbor cells which make a dead cell to become alive (i.e. to "born"). "s" and "b" can be used in place of "S" and "B", respectively.
>
> Alternatively a rule can be specified by an 18-bits integer. The 9 high order bits are used to encode the next cell state if it is alive for each number of neighbor alive cells, the low order bits specify the rule for "borning" new cells. Higher order bits encode for an higher number of neighbor cells. For example the number 6153 = `(12<<9)+9` specifies a stay alive rule of 12 and a born rule of 9, which corresponds to "S23/B03".

Default value is "S23/B3", which is the original Conway's game of life rule, and will keep a cell alive if it has 2 or 3 neighbor alive cells, and will born a new cell if there are three alive cells around a dead cell.

'`size, s`'

Set the size of the output video.

If '`filename`' is specified, the size is set by default to the same size of the input file. If '`size`' is set, it must contain the size specified in the input file, and the initial grid defined in that file is centered in the larger resulting area.

If a filename is not specified, the size value defaults to "320x240" (used for a randomly generated initial grid).

'`stitch`'

If set to 1, stitch the left and right grid edges together, and the top and bottom edges also. Defaults to 1.

'`mold`'

Set cell mold speed. If set, a dead cell will go from '`death_color`' to '`mold_color`' with a step of '`mold`'. '`mold`' can have a value from 0 to 255.

'`life_color`'

Set the color of living (or new born) cells.

'`death_color`'

Set the color of dead cells. If '`mold`' is set, this is the first color used to represent a dead cell.

'`mold_color`'

Set mold color, for definitely dead and moldy cells.

29.6.1 Examples

- Read a grid from '`pattern`', and center it on a grid of size 300x300 pixels:

```
life=f=pattern:s=300x300
```

- Generate a random grid of size 200x200, with a fill ratio of 2/3:

```
life=ratio=2/3:s=200x200
```

- Specify a custom rule for evolving a randomly generated grid:

```
life=rule=S14/B34
```

- Full example with slow death effect (mold) using `ffplay`:

```
ffplay -f lavfi life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
```

29.7 color, nullsrc, rgbtestsrc, smptebars, testsrc

The `color` source provides an uniformly colored input.

The `nullsrc` source returns unprocessed video frames. It is mainly useful to be employed in analysis / debugging tools, or as the source for filters which ignore the input data.

The `rgbtestsrc` source generates an RGB test pattern useful for detecting RGB vs BGR issues. You should see a red, green and blue stripe from top to bottom.

The `smptebars` source generates a color bars pattern, based on the SMPTE Engineering Guideline EG 1-1990.

The `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

These sources accept an optional sequence of *key*=*value* pairs, separated by ":". The description of the accepted options follows.

‘`color, c`’

> Specify the color of the source, only used in the `color` source. It can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence, possibly followed by an alpha specifier. The default value is "black".

‘`size, s`’

> Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation. The default value is "320x240".

‘`rate, r`’

> Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

‘`sar`’

Set the sample aspect ratio of the sourced video.

‘`duration, d`’

Set the video duration of the sourced video. The accepted syntax is:

```
[-]HH[:MM[:SS[.m...]]]
[-]S+[.m...]
```

See also the function `av_parse_time()`.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

‘`decimals, n`’

Set the number of decimals to show in the timestamp, only used in the `testsrc` source.

The displayed timestamp value will correspond to the original timestamp value multiplied by the power of 10 of the specified value. Default value is 0.

For example the following:

```
testsrc=duration=5.3:size=qcif:rate=10
```

will generate a video with a duration of 5.3 seconds, with size 176x144 and a frame rate of 10 frames per second.

The following graph description will generate a red source with an opacity of 0.2, with size "qcif" and a frame rate of 10 frames per second.

```
color=c=red@0.2:s=qcif:r=10
```

If the input content is to be ignored, `nullsrc` can be used. The following command generates noise in the luminance plane by employing the `geq` filter:

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

30. Video Sinks

Below is a description of the currently available video sinks.

30.1 buffersink

Buffer video frames, and make them available to the end of the filter graph.

This sink is mainly intended for a programmatic use, in particular through the interface defined in ‘`libavfilter/buffersink.h`’.

It does not require a string parameter in input, but you need to specify a pointer to a list of supported pixel formats terminated by -1 in the opaque parameter provided to `avfilter_init_filter` when initializing this sink.

30.2 nullsink

Null video sink, do absolutely nothing with the input video. It is mainly useful as a template and to be employed in analysis / debugging tools.

31. Multimedia Filters

Below is a description of the currently available multimedia filters.

31.1 asendcmd, sendcmd

Send commands to filters in the filtergraph.

These filters read commands to be sent to other filters in the filtergraph.

`asendcmd` must be inserted between two audio filters, `sendcmd` must be inserted between two video filters, but apart from that they act the same way.

The specification of commands can be provided in the filter arguments with the *commands* option, or in a file specified by the *filename* option.

These filters accept the following options:

‘`commands, c`’

Set the commands to be read and sent to the other filters.

‘`filename, f`’

Set the filename of the commands to be read and sent to the other filters.

31.1.1 Commands syntax

A commands description consists of a sequence of interval specifications, comprising a list of commands to be executed when a particular event related to that interval occurs. The occurring event is typically the current frame time entering or leaving a given time interval.

An interval is specified by the following syntax:

```
START[-END] COMMANDS;
```

The time interval is specified by the *START* and *END* times. *END* is optional and defaults to the maximum time.

The current frame time is considered within the specified interval if it is included in the interval [*START*, *END*), that is when the time is greater or equal to *START* and is lesser than *END*.

COMMANDS consists of a sequence of one or more command specifications, separated by ",", relating to that interval. The syntax of a command specification is given by:

```
[FLAGS] TARGET COMMAND ARG
```

FLAGS is optional and specifies the type of events relating to the time interval which enable sending the specified command, and must be a non-null sequence of identifier flags separated by "+" or "|" and enclosed between "[" and "]".

The following flags are recognized:

‘`enter`’

> The command is sent when the current frame timestamp enters the specified interval. In other words, the command is sent when the previous frame timestamp was not in the given interval, and the current is.

‘`leave`’

> The command is sent when the current frame timestamp leaves the specified interval. In other words, the command is sent when the previous frame timestamp was in the given interval, and the current is not.

If *FLAGS* is not specified, a default value of `[enter]` is assumed.

TARGET specifies the target of the command, usually the name of the filter class or a specific filter instance name.

COMMAND specifies the name of the command for the target filter.

ARG is optional and specifies the optional list of argument for the given *COMMAND*.

Between one interval specification and another, whitespaces, or sequences of characters starting with # until the end of line, are ignored and can be used to annotate comments.

A simplified BNF description of the commands specification syntax follows:

```
COMMAND_FLAG  ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND       ::= ["[" COMMAND_FLAGS "]"] TARGET COMMAND [ARG]
COMMANDS      ::= COMMAND [,COMMANDS]
INTERVAL      ::= START[-END] COMMANDS
INTERVALS     ::= INTERVAL[;INTERVALS]
```

31.1.2 Examples

- Specify audio tempo change at second 4:

  ```
  asendcmd=c='4.0 atempo tempo 1.5',atempo
  ```

- Specify a list of drawtext and hue commands in a file.

  ```
  # show text in the interval 5-10
  5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
           [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';

  # desaturate the image in the interval 15-20
  15.0-20.0 [enter] hue reinit s=0,
            [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
            [leave] hue reinit s=1,
            [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';

  # apply an exponential saturation fade-out effect, starting from time 25
  25 [enter] hue s=exp(t-25)
  ```

  A filtergraph allowing to read and process the above command list stored in a file '`test.cmd`', can be specified with:

  ```
  sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text='',hue
  ```

31.2 asetpts, setpts

Change the PTS (presentation timestamp) of the input frames.

`asetpts` works on audio frames, `setpts` on video frames.

Accept in input an expression evaluated through the eval API, which can contain the following constants:

‘`FRAME_RATE`’

frame rate, only defined for constant frame-rate video

‘`PTS`’

the presentation timestamp in input

‘`N`’

the count of the input frame, starting from 0.

‘`NB_CONSUMED_SAMPLES`’

the number of consumed samples, not including the current frame (only audio)

‘`NB_SAMPLES`’

the number of samples in the current frame (only audio)

‘`SAMPLE_RATE`’

audio sample rate

‘`STARTPTS`’

the PTS of the first frame

‘`STARTT`’

the time in seconds of the first frame

‘`INTERLACED`’

tell if the current frame is interlaced

‘`T`’

the time in seconds of the current frame

‘`TB`’

the time base

‘`POS`’

original position in the file of the frame, or undefined if undefined for the current frame

‘PREV_INPTS’

previous input PTS

‘PREV_INT’

previous input time in seconds

‘PREV_OUTPTS’

previous output PTS

‘PREV_OUTT’

previous output time in seconds

31.2.1 Examples

- Start counting PTS from zero

  ```
  setpts=PTS-STARTPTS
  ```

- Apply fast motion effect:

  ```
  setpts=0.5*PTS
  ```

- Apply slow motion effect:

  ```
  setpts=2.0*PTS
  ```

- Set fixed rate of 25 frames per second:

  ```
  setpts=N/(25*TB)
  ```

- Set fixed rate 25 fps with some jitter:

  ```
  setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'
  ```

- Apply an offset of 10 seconds to the input PTS:

  ```
  setpts=PTS+10/TB
  ```

31.3 ebur128

EBU R128 scanner filter. This filter takes an audio stream as input and outputs it unchanged. By default, it logs a message at a frequency of 10Hz with the Momentary loudness (identified by `M`), Short-term loudness (`S`), Integrated loudness (`I`) and Loudness Range (`LRA`).

The filter also has a video output (see the *video* option) with a real time graph to observe the loudness evolution. The graphic contains the logged message mentioned above, so it is not printed anymore when this option is set, unless the verbose logging is set. The main graphing area contains the short-term loudness (3 seconds of analysis), and the gauge on the right is for the momentary loudness (400 milliseconds).

More information about the Loudness Recommendation EBU R128 on http://tech.ebu.ch/loudness.

The filter accepts the following named parameters:

‘`video`’

> Activate the video output. The audio stream is passed unchanged whether this option is set or no. The video stream will be the first output stream if activated. Default is `0`.

‘`size`’

> Set the video size. This option is for video only. Default and minimum resolution is `640x480`.

‘`meter`’

> Set the EBU scale meter. Default is `9`. Common values are `9` and `18`, respectively for EBU scale meter +9 and EBU scale meter +18. Any other integer value between this range is allowed.

Example of real-time graph using `ffplay`, with a EBU scale meter +18:

```
ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
```

Run an analysis with `ffmpeg`:

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

31.4 settb, asettb

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

It accepts in input an arithmetic expression representing a rational. The expression can contain the constants "AVTB" (the default timebase), "intb" (the input timebase) and "sr" (the sample rate, audio only).

The default value for the input is "intb".

31.4.1 Examples

- Set the timebase to 1/25:

  ```
  settb=1/25
  ```

- Set the timebase to 1/10:

  ```
  settb=0.1
  ```

- Set the timebase to 1001/1000:

  ```
  settb=1+0.001
  ```

- Set the timebase to 2*intb:

  ```
  settb=2*intb
  ```

- Set the default timebase value:

  ```
  settb=AVTB
  ```

31.5 concat

Concatenate audio and video streams, joining them together one after the other.

The filter works on segments of synchronized video and audio streams. All segments must have the same number of streams of each type, and that will also be the number of streams at output.

The filter accepts the following named parameters:

‘`n`’

   Set the number of segments. Default is 2.

‘`v`’

   Set the number of output video streams, that is also the number of video streams in each segment. Default is 1.

‘a’

Set the number of output audio streams, that is also the number of video streams in each segment. Default is 0.

‘unsafe’

Activate unsafe mode: do not fail if segments have a different format.

The filter has *v*+*a* outputs: first *v* video outputs, then *a* audio outputs.

There are *n*Ã(*v*+*a*) inputs: first the inputs for the first segment, in the same order as the outputs, then the inputs for the second segment, etc.

Related streams do not always have exactly the same duration, for various reasons including codec frame size or sloppy authoring. For that reason, related synchronized streams (e.g. a video and its audio track) should be concatenated at once. The concat filter will use the duration of the longest stream in each segment (except the last one), and if necessary pad shorter audio streams with silence.

For this filter to work correctly, all segments must start at timestamp 0.

All corresponding streams must have the same parameters in all segments; the filtering system will automatically select a common pixel format for video streams, and a common sample format, sample rate and channel layout for audio streams, but other settings, such as resolution, must be converted explicitly by the user.

Different frame rates are acceptable but will result in variable frame rate at output; be sure to configure the output file to handle it.

Examples:

- Concatenate an opening, an episode and an ending, all in bilingual version (video in stream 0, audio in streams 1 and 2):

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
  ’[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2]
   concat=n=3:v=1:a=2 [v] [a1] [a2]’ \
  -map ’[v]’ -map ’[a1]’ -map ’[a2]’ output.mkv
```

- Concatenate two parts, handling audio and video separately, using the (a)movie sources, and adjusting the resolution:

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;
movie=part2.mp4, scale=512:288 [v2] ; amovie=part2.mp4 [a2] ;
[v1] [v2] concat [outv] ; [a1] [a2] concat=v=0:a=1 [outa]
```

Note that a desync will happen at the stitch if the audio and video streams do not have exactly the same duration in the first file.

31.6 showspectrum

Convert input audio to a video output, representing the audio frequency spectrum.

The filter accepts the following named parameters:

‘`size, s`’

Specify the video size for the output. Default value is `640x480`.

‘`slide`’

Specify if the spectrum should slide along the window. Default value is `0`.

The usage is very similar to the showwaves filter; see the examples in that section.

31.7 showwaves

Convert input audio to a video output, representing the samples waves.

The filter accepts the following named parameters:

‘`n`’

Set the number of samples which are printed on the same column. A larger value will decrease the frame rate. Must be a positive integer. This option can be set only if the value for *rate* is not explicitly specified.

‘`rate, r`’

Set the (approximate) output frame rate. This is done by setting the option *n*. Default value is "25".

‘`size, s`’

Specify the video size for the output. Default value is "600x240".

Some examples follow.

- Output the input file audio and the corresponding video representation at the same time:

  ```
  amovie=a.mp3,asplit[out0],showwaves[out1]
  ```

- Create a synthetic signal and show it with showwaves, forcing a framerate of 30 frames per second:

```
        aevalsrc=sin(1*2*PI*t)*sin(880*2*PI*t):cos(2*PI*200*t),asplit[out0],showwaves=r=30[out1]
```

32. Multimedia Sources

Below is a description of the currently available multimedia sources.

32.1 amovie

This is the same as src_movie source, except it selects an audio stream by default.

32.2 movie

Read audio and/or video stream(s) from a movie container.

It accepts the syntax: *movie_name*[:*options*] where *movie_name* is the name of the resource to read (not necessarily a file but also a device or a stream accessed through some protocol), and *options* is an optional sequence of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`format_name, f`’

> Specifies the format assumed for the movie to read, and can be either the name of a container or an input device. If not specified the format is guessed from *movie_name* or by probing.

‘`seek_point, sp`’

> Specifies the seek point in seconds, the frames will be output starting from this seek point, the parameter is evaluated with `av_strtod` so the numerical value may be suffixed by an IS postfix. Default value is "0".

‘`streams, s`’

> Specifies the streams to read. Several streams can be specified, separated by "+". The source will then have as many outputs, in the same order. The syntax is explained in the Stream specifiers chapter. Two special names, "dv" and "da" specify respectively the default (best suited) video and audio stream. Default is "dv", or "da" if the filter is called as "amovie".

‘`stream_index, si`’

> Specifies the index of the video stream to read. If the value is -1, the best suited video stream will be automatically selected. Default value is "-1". Deprecated. If the filter is called "amovie", it will select audio instead of video.

‘loop’

Specifies how many times to read the stream in sequence. If the value is less than 1, the stream will be read again and again. Default value is "1".

Note that when the movie is looped the source timestamps are not changed, so it will generate non monotonically increasing timestamps.

This filter allows to overlay a second video on top of main input of a filtergraph as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                    ^
                                    |
movie --> scale--> deltapts1 -------+
```

Some examples follow.

- Skip 3.2 seconds from the start of the avi file in.avi, and overlay it on top of the input labelled as "in":

  ```
  movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [movie];
  [in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]
  ```

- Read from a video4linux2 device, and overlay it on top of the input labelled as "in":

  ```
  movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [movie];
  [in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]
  ```

- Read the first video stream and the audio stream with id 0x81 from dvd.vob; the video is connected to the pad named "video" and the audio is connected to the pad named "audio":

  ```
  movie=dvd.vob:s=v:0+#0x81 [video] [audio]
  ```

33. Metadata

FFmpeg is able to dump metadata from media files into a simple UTF-8-encoded INI-like text file and then load it back using the metadata muxer/demuxer.

The file format is as follows:

1. A file consists of a header and a number of metadata tags divided into sections, each on its own line.
2. The header is a ’;FFMETADATA’ string, followed by a version number (now 1).
3. Metadata tags are of the form ’key=value’
4. Immediately after header follows global metadata
5. After global metadata there may be sections with per-stream/per-chapter metadata.
6. A section starts with the section name in uppercase (i.e. STREAM or CHAPTER) in brackets (’[’, ’]’) and ends with next section or end of file.

7. At the beginning of a chapter section there may be an optional timebase to be used for start/end values. It must be in form ’TIMEBASE=num/den’, where num and den are integers. If the timebase is missing then start/end times are assumed to be in milliseconds. Next a chapter section must contain chapter start and end times in form ’START=num’, ’END=num’, where num is a positive integer.
8. Empty lines and lines starting with ’;’ or ’#’ are ignored.
9. Metadata keys or values containing special characters (’=’, ’;’, ’#’, ’\’ and a newline) must be escaped with a backslash ’\’.
10. Note that whitespace in metadata (e.g. foo = bar) is considered to be a part of the tag (in the example above key is ’foo ’, value is ’ bar’).

A ffmetadata file might look like this:

```
;FFMETADATA1
title=bike\\shed
;this is a comment
artist=FFmpeg troll team

[CHAPTER]
TIMEBASE=1/1000
START=0
#chapter ends at 0:01:00
END=60000
title=chapter \#1
[STREAM]
title=multi\
line
```

This document was generated by *john* on *November 26, 2012* using *texi2html 1.82*.

ffmpeg Documentation

Table of Contents

1. Synopsis

The generic syntax is:

```
ffmpeg [global options] [[infile options]['-i' infile]]... {[outfile options] outfile}...
```

2. Description

ffmpeg is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

ffmpeg reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output filename. Anything found on the command line which cannot be interpreted as an option is considered to be an output filename.

Each input or output file can in principle contain any number of streams of different types (video/audio/subtitle/attachment/data). Allowed number and/or types of streams can be limited by the container format. Selecting, which streams from which inputs go into output, is done either automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1` etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. See also the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

- To set the video bitrate of the output file to 64kbit/s:

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

- To force the frame rate of the output file to 24 fps:

```
ffmpeg -i input.avi -r 24 output.avi
```

- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

```
ffmpeg -r 1 -i input.m2v -r 24 output.avi
```

The format option may be needed for raw input files.

3. Detailed description

The transcoding process in `ffmpeg` for each output can be described by the following diagram:

```
 _______              ______________               _________              ______________            ________
|       |            |              |             |         |            |              |          |        |
| input |  demuxer   | encoded data |   decoder   | decoded |  encoder   | encoded data |  muxer   | output |
| file  | ---------> | packets      |  ---------> | frames  | ---------> | packets      | -------> | file   |
|_______|            |______________|             |_________|            |______________|          |________|
```

`ffmpeg` calls the libavformat library (containing demuxers) to read input files and get packets containing encoded data from them. When there are multiple input files, `ffmpeg` tries to keep them synchronized by tracking lowest timestamp on any active input stream.

Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering the frames are passed to the encoder, which encodes them and outputs encoded packets again. Finally those are passed to the muxer, which writes the encoded packets to the output file.

3.1 Filtering

Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs - simple and complex.

3.1.1 Simple filtergraphs

Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:

```
 _________                        __________              ______________
|         |                      |          |            |              |
| decoded |  simple filtergraph  | filtered |  encoder   | encoded data |
| frames  | -------------------> | frames   | ---------> | packets      |
|_________|                      |__________|            |______________|
```

Simple filtergraphs are configured with the per-stream '`-filter`' option (with '`-vf`' and '`-af`' aliases for video and audio respectively). A simple filtergraph for video can look for example like this:

```
 _______        _____________        _______        _____        ________
|       |      |             |      |       |      |     |      |        |
| input | ---> | deinterlace | ---> | scale | ---> | fps | ---> | output |
|_______|      |_____________|      |_______|      |_____|      |________|
```

Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

3.1.2 Complex filtergraphs

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case e.g. when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:

```
 _________
|         |
| input 0 |\                    __________
|_________| \                  |          |
             \   _________    /| output 0 |
              \ |         |  / |__________|
 _________     \| complex | /
|         |     |         |/
| input 1 |---->| filter  |\
|_________|     |         | \   __________
               /| graph   |  \ |          |
              / |         |   \| output 1 |
 _________   /  |_________|    |__________|
|         | /
| input 2 |/
|_________|
```

Complex filtergraphs are configured with the '`-filter_complex`' option. Note that this option is global, since a complex filtergraph by its nature cannot be unambiguously associated with a single stream or file.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

3.2 Stream copy

Stream copy is a mode selected by supplying the `copy` parameter to the ‘`-codec`’ option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will in this case simplify to this:

```
 _______              ______________            ________
|       |            |              |          |        |
| input |  demuxer   | encoded data |  muxer   | output |
| file  | ---------> | packets      | -------> | file   |
|_______|            |______________|          |________|
```

Since there is no decoding or encoding, it is very fast and there is no quality loss. However it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

4. Stream selection

By default ffmpeg includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria; for video it is the stream with the highest resolution, for audio the stream with the most channels, for subtitle it’s the first subtitle stream. In the case where several streams of the same type rate equally, the lowest numbered stream is chosen.

You can disable some of those defaults by using `-vn/-an/-sn` options. For full manual control, use the `-map` option, which disables the defaults just described.

5. Options

All the numerical options, if not specified otherwise, accept in input a string representing a number, which may contain one of the SI unit prefixes, for example ’K’, ’M’, ’G’. If ’i’ is appended after the prefix, binary prefixes are used, which are based on powers of 1024 instead of powers of 1000. The ’B’ postfix multiplies the value by 8, and can be appended after a unit prefix or used alone. This allows using for example ’KB’, ’MiB’, ’G’ and ’B’ as number postfix.

Options which do not take arguments are boolean options, and set the corresponding value to true. They can be set to false by prefixing with "no" the option name, for example using "-nofoo" in the command line will set to false the boolean option with name "foo".

5.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) does a given option belong to.

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. `-codec:a:1 ac3` option contains `a:1` stream specifier, which matches the second audio stream. Therefore it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, the option is then applied to all of them. E.g. the stream specifier in `-b:a 128k` matches all audio streams.

An empty stream specifier matches all streams, for example `-codec copy` or `-codec: copy` would copy all the streams without reencoding.

Possible forms of stream specifiers are:

‘*stream_index*’

Matches the stream with this index. E.g. `-threads:1 4` would set the thread count for the second stream to 4.

‘*stream_type*[:*stream_index*]’

stream_type is one of: ’v’ for video, ’a’ for audio, ’s’ for subtitle, ’d’ for data and ’t’ for attachments. If *stream_index* is given, then matches stream number *stream_index* of this type. Otherwise matches all streams of this type.

‘p:*program_id*[:*stream_index*]’

If *stream_index* is given, then matches stream number *stream_index* in program with id *program_id*. Otherwise matches all streams in this program.

‘#*stream_id*’

Matches the stream by format-specific ID.

5.2 Generic options

These options are shared amongst the av* tools.

‘`-L`’

Show license.

‘`-h, -?, -help, --help [`*arg*`]`’

Show help. An optional parameter may be specified to print help about a specific item.

Possible values of *arg* are:

‘`decoder=`*`decoder_name`*’

Print detailed information about the decoder named *decoder_name*. Use the ‘`-decoders`’ option to get a list of all decoders.

‘`encoder=`*`encoder_name`*’

Print detailed information about the encoder named *encoder_name*. Use the ‘`-encoders`’ option to get a list of all encoders.

‘`demuxer=`*`demuxer_name`*’

Print detailed information about the demuxer named *demuxer_name*. Use the ‘`-formats`’ option to get a list of all demuxers and muxers.

‘`muxer=`*`muxer_name`*’

Print detailed information about the muxer named *muxer_name*. Use the ‘`-formats`’ option to get a list of all muxers and demuxers.

‘`-version`’

Show version.

‘`-formats`’

Show available formats.

The fields preceding the format names have the following meanings:

‘`D`’

Decoding available

‘`E`’

Encoding available

‘`-codecs`’

Show all codecs known to libavcodec.

Note that the term ’codec’ is used throughout this documentation as a shortcut for what is more correctly called a media bitstream format.

‘`-decoders`’

Show available decoders.

‘`-encoders`’

Show all available encoders.

‘`-bsfs`’

Show available bitstream filters.

‘`-protocols`’

Show available protocols.

‘`-filters`’

Show available libavfilter filters.

‘`-pix_fmts`’

Show available pixel formats.

‘`-sample_fmts`’

Show available sample formats.

‘`-layouts`’

Show channel names and standard channel layouts.

‘`-loglevel` *`loglevel`* `| -v` *`loglevel`*’

Set the logging level used by the library. *loglevel* is a number or a string containing one of the following values:

‘`quiet`’
‘`panic`’
‘`fatal`’
‘`error`’
‘`warning`’
‘`info`’
‘`verbose`’
‘`debug`’

By default the program logs to stderr, if coloring is supported by the terminal, colors are used to mark errors and warnings. Log coloring can be disabled setting the environment variable `AV_LOG_FORCE_NOCOLOR` or `NO_COLOR`, or can be forced setting the environment variable `AV_LOG_FORCE_COLOR`. The use of the environment variable `NO_COLOR` is deprecated and will be dropped in a following FFmpeg version.

‘`-report`’

Dump full command line and console output to a file named `*program*-*YYYYMMDD*-*HHMMSS*.log` in the current directory. This file can be useful for bug reports. It also implies `-loglevel verbose`.

Setting the environment variable `FFREPORT` to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ’:’. The following option is recognized:

‘`file`’

set the file name to use for the report; `%p` is expanded to the name of the program, `%t` is expanded to a timestamp, `%%` is expanded to a plain `%`

Errors in parsing the environment variable are not fatal, and will not appear in the report.

‘`-cpuflags flags (`*`global`*`)`’

Allows setting and clearing cpu flags. This option is intended for testing. Do not use it unless you know what you’re doing.

```
ffmpeg -cpuflags -sse+mmx ...
ffmpeg -cpuflags mmx ...
ffmpeg -cpuflags 0 ...
```

5.3 AVOptions

These options are provided directly by the libavformat, libavdevice and libavcodec libraries. To see the list of available AVOptions, use the ‘`-help`’ option. They are separated into two categories:

‘`generic`’

These options can be set for any container, codec or device. Generic options are listed under AVFormatContext options for containers/devices and under AVCodecContext options for codecs.

‘`private`’

These options are specific to the given container, device or codec. Private options are listed under their corresponding containers/devices/codecs.

For example to write an ID3v2.3 header instead of a default ID3v2.4 to an MP3 file, use the ‘`id3v2_version`’ private option of the MP3 muxer:

```
ffmpeg -i input.flac -id3v2_version 3 out.mp3
```

All codec AVOptions are obviously per-stream, so the chapter on stream specifiers applies to them

Note '`-nooption`' syntax cannot be used for boolean AVOptions, use '`-option 0`'/'`-option 1`'.

Note2 old undocumented way of specifying per-stream AVOptions by prepending v/a/s to the options name is now obsolete and will be removed soon.

5.4 Codec AVOptions

'`-b[:stream_specifier]` *`integer`* `(`*`output,audio,video`*`)`'

set bitrate (in bits/s)

'`-ab[:stream_specifier]` *`integer`* `(`*`output,audio`*`)`'

set bitrate (in bits/s)

'`-bt[:stream_specifier]` *`integer`* `(`*`output,video`*`)`'

Set video bitrate tolerance (in bits/s). In 1-pass mode, bitrate tolerance specifies how far ratecontrol is willing to deviate from the target average bitrate value. This is not related to min/max bitrate. Lowering tolerance too much has an adverse effect on quality.

'`-flags[:stream_specifier]` *`flags`* `(`*`input/output,audio,video,subtitles`*`)`'

Possible values:

'`mv4`'

use four motion vector by macroblock (mpeg4)

'`qpel`'

use 1/4 pel motion compensation

'`loop`'

use loop filter

'`qscale`'

use fixed qscale

'`gmc`'

use gmc

'`mv0`'

always try a mb with mv=<0,0>

‘`input_preserved`’
‘`pass1`’

use internal 2pass ratecontrol in first pass mode

‘`pass2`’

use internal 2pass ratecontrol in second pass mode

‘`gray`’

only decode/encode grayscale

‘`emu_edge`’

don’t draw edges

‘`psnr`’

error[?] variables will be set during encoding

‘`truncated`’
‘`naq`’

normalize adaptive quantization

‘`ildct`’

use interlaced dct

‘`low_delay`’

force low delay

‘`global_header`’

place global headers in extradata instead of every keyframe

‘`bitexact`’

use only bitexact stuff (except (i)dct)

‘`aic`’

h263 advanced intra coding / mpeg4 ac prediction ‘`cbp`’

Deprecated, use mpegvideo private options instead

‘`qprd`’

Deprecated, use mpegvideo private options instead

‘`ilme`’

interlaced motion estimation

‘`cgop`’

closed gop

‘`-sub_id[:stream_specifier] `*`integer`*` ()`’
‘`-me_method[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

set motion estimation method

Possible values:

‘`zero`’

zero motion estimation (fastest)

‘`full`’

full motion estimation (slowest)

‘`epzs`’

EPZS motion estimation (default)

‘`esa`’

esa motion estimation (alias for full)

‘`tesa`’

tesa motion estimation

‘`dia`’

dia motion estimation (alias for epzs)

‘`log`’

log motion estimation

‘`phods`’

phods motion estimation

‘`x1`’

X1 motion estimation

‘`hex`’

hex motion estimation

‘`umh`’

umh motion estimation

‘`iter`’

iter motion estimation

‘`-extradata_size[:stream_specifier] `*`integer`*` ()`’
‘`-time_base[:stream_specifier] `*`rational number`*` ()`’
‘`-g[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

set the group of picture size

‘`-ar[:stream_specifier] `*`integer`*` (`*`input/output,audio`*`)`’

set audio sampling rate (in Hz)

‘`-ac[:stream_specifier] `*`integer`*` (`*`input/output,audio`*`)`’

set number of audio channels

‘`-cutoff[:stream_specifier] `*`integer`*` (`*`output,audio`*`)`’

set cutoff bandwidth

‘`-frame_size[:stream_specifier] `*`integer`*` (`*`output,audio`*`)`’
‘`-frame_number[:stream_specifier] `*`integer`*` ()`’
‘`-delay[:stream_specifier] `*`integer`*` ()`’
‘`-qcomp[:stream_specifier] `*`float`*` (`*`output,video`*`)`’

video quantizer scale compression (VBR). Constant of ratecontrol equation. Recommended range for default rc_eq: 0.0-1.0

‘-qblur[:stream_specifier] *float* (*output,video*)’

video quantizer scale blur (VBR)

‘-qmin[:stream_specifier] *integer* (*output,video*)’

min video quantizer scale (VBR)

‘-qmax[:stream_specifier] *integer* (*output,video*)’

max video quantizer scale (VBR)

‘-qdiff[:stream_specifier] *integer* (*output,video*)’

max difference between the quantizer scale (VBR)

‘-bf[:stream_specifier] *integer* (*output,video*)’

use ’frames’ B frames

‘-b_qfactor[:stream_specifier] *float* (*output,video*)’

qp factor between p and b frames

‘-rc_strategy[:stream_specifier] *integer* (*output,video*)’

ratecontrol method

‘-b_strategy[:stream_specifier] *integer* (*output,video*)’

strategy to choose between I/P/B-frames

‘-ps[:stream_specifier] *integer* (*output,video*)’

rtp payload size in bytes

‘-mv_bits[:stream_specifier] *integer* ()’
‘-header_bits[:stream_specifier] *integer* ()’
‘-i_tex_bits[:stream_specifier] *integer* ()’
‘-p_tex_bits[:stream_specifier] *integer* ()’
‘-i_count[:stream_specifier] *integer* ()’
‘-p_count[:stream_specifier] *integer* ()’
‘-skip_count[:stream_specifier] *integer* ()’
‘-misc_bits[:stream_specifier] *integer* ()’
‘-frame_bits[:stream_specifier] *integer* ()’
‘-codec_tag[:stream_specifier] *integer* ()’
‘-bug[:stream_specifier] *flags* (*input,video*)’

workaround not auto detected encoder bugs

Possible values:

‘`autodetect`’
‘`old_msmpeg4`’

> some old lavc generated msmpeg4v3 files (no autodetection)

‘`xvid_ilace`’

> Xvid interlacing bug (autodetected if fourcc==XVIX)

‘`ump4`’

> (autodetected if fourcc==UMP4)

‘`no_padding`’

> padding bug (autodetected)

‘`amv`’
‘`ac_vlc`’

> illegal vlc bug (autodetected per fourcc)

‘`qpel_chroma`’
‘`std_qpel`’

> old standard qpel (autodetected per fourcc/version)

‘`qpel_chroma2`’
‘`direct_blocksize`’

> direct-qpel-blocksize bug (autodetected per fourcc/version)

‘`edge`’

> edge padding bug (autodetected per fourcc/version)

‘`hpel_chroma`’
‘`dc_clip`’
‘`ms`’

> workaround various bugs in microsofts broken decoders

‘`trunc`’

trancated frames

‘`-lelim[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

single coefficient elimination threshold for luminance (negative values also consider dc coefficient)

‘`-celim[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

single coefficient elimination threshold for chrominance (negative values also consider dc coefficient)

‘`-strict[:stream_specifier] `*`integer`*` (`*`input/output,audio,video`*`)`’

how strictly to follow the standards

Possible values:

‘`very`’

strictly conform to a older more strict version of the spec or reference software

‘`strict`’

strictly conform to all the things in the spec no matter what consequences

‘`normal`’
‘`unofficial`’

allow unofficial extensions

‘`experimental`’

allow non standardized experimental things

‘`-b_qoffset[:stream_specifier] `*`float`*` (`*`output,video`*`)`’

qp offset between P and B frames

‘`-err_detect[:stream_specifier] `*`flags`*` (`*`input,audio,video`*`)`’

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

consider things that violate the spec and have not been seen in the wild as errors

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder should not do as an error

‘`-has_b_frames[:stream_specifier] *integer* ()`’
‘`-block_align[:stream_specifier] *integer* ()`’
‘`-mpeg_quant[:stream_specifier] *integer* (*output,video*)`’

use MPEG quantizers instead of H.263

‘`-qsquish[:stream_specifier] *float* (*output,video*)`’

how to keep quantizer between qmin and qmax (0 = clip, 1 = use differentiable function)

‘`-rc_qmod_amp[:stream_specifier] *float* (*output,video*)`’

experimental quantizer modulation

‘`-rc_qmod_freq[:stream_specifier] *integer* (*output,video*)`’

experimental quantizer modulation

‘`-rc_override_count[:stream_specifier] *integer* ()`’
‘`-rc_eq[:stream_specifier] *string* (*output,video*)`’

Set rate control equation. When computing the expression, besides the standard functions defined in the section ’Expression Evaluation’, the following functions are available: bits2qp(bits), qp2bits(qp). Also the following constants are available: iTex pTex tex mv fCode iCount mcVar var isI isP isB avgQP qComp avgIITex avgPITex avgPPTex avgBPTex avgTex.

‘`-maxrate[:stream_specifier]` *integer* (*output,audio,video*)’

Set max bitrate tolerance (in bits/s). Requires bufsize to be set.

‘`-minrate[:stream_specifier]` *integer* (*output,audio,video*)’

Set min bitrate tolerance (in bits/s). Most useful in setting up a CBR encode. It is of little use elsewise.

‘`-bufsize[:stream_specifier]` *integer* (*output,audio,video*)’

set ratecontrol buffer size (in bits)

‘`-rc_buf_aggressivity[:stream_specifier]` *float* (*output,video*)’

currently useless

‘`-i_qfactor[:stream_specifier]` *float* (*output,video*)’

qp factor between P and I frames

‘`-i_qoffset[:stream_specifier]` *float* (*output,video*)’

qp offset between P and I frames

‘`-rc_init_cplx[:stream_specifier]` *float* (*output,video*)’

initial complexity for 1-pass encoding

‘`-dct[:stream_specifier]` *integer* (*output,video*)’

DCT algorithm

Possible values:

‘`auto`’

autoselect a good one (default)

‘`fastint`’

fast integer

‘`int`’

accurate integer

‘`mmx`’

‘`altivec`’
‘`faan`’

floating point AAN DCT

‘`-lumi_mask[:stream_specifier] *float* (*output,video*)`’

compresses bright areas stronger than medium ones

‘`-tcplx_mask[:stream_specifier] *float* (*output,video*)`’

temporal complexity masking

‘`-scplx_mask[:stream_specifier] *float* (*output,video*)`’

spatial complexity masking

‘`-p_mask[:stream_specifier] *float* (*output,video*)`’

inter masking

‘`-dark_mask[:stream_specifier] *float* (*output,video*)`’

compresses dark areas stronger than medium ones

‘`-idct[:stream_specifier] *integer* (*input/output,video*)`’

select IDCT implementation

Possible values:

‘`auto`’
‘`int`’
‘`simple`’
‘`simplemmx`’
‘`libmpeg2mmx`’
‘`mmi`’
‘`arm`’
‘`altivec`’
‘`sh4`’
‘`simplearm`’
‘`simplearmv5te`’
‘`simplearmv6`’
‘`simpleneon`’
‘`simplealpha`’
‘`h264`’
‘`vp3`’

‘`ipp`’
‘`xvidmmx`’
‘`faani`’

floating point AAN IDCT

‘`-slice_count[:stream_specifier] *integer* ()`’
‘`-ec[:stream_specifier] *flags* (*input,video*)`’

set error concealment strategy

Possible values:

‘`guess_mvs`’

iterative motion vector (MV) search (slow)

‘`deblock`’

use strong deblock filter for damaged MBs

‘`-bits_per_coded_sample[:stream_specifier] *integer* ()`’
‘`-pred[:stream_specifier] *integer* (*output,video*)`’

prediction method

Possible values:

‘`left`’
‘`plane`’
‘`median`’
‘`-aspect[:stream_specifier] *rational number* (*output,video*)`’

sample aspect ratio

‘`-debug[:stream_specifier] *flags* (*input/output,audio,video,subtitles*)`’

print specific debug info

Possible values:

‘`pict`’

picture info

‘`rc`’

rate control

‘`bitstream`’
‘`mb_type`’

macroblock (MB) type

‘`qp`’

per-block quantization parameter (QP)

‘`mv`’

motion vector

‘`dct_coeff`’
‘`skip`’
‘`startcode`’
‘`pts`’
‘`er`’

error recognition

‘`mmco`’

memory management control operations (H.264)

‘`bugs`’
‘`vis_qp`’

visualize quantization parameter (QP), lower QP are tinted greener

‘`vis_mb_type`’

visualize block types

‘`buffers`’

picture buffer allocations

‘`thread_ops`’

threading operations

‘`-vismv[:stream_specifier]` *`integer`* `(`*`input,video`*`)`’

visualize motion vectors (MVs)

Possible values:

‘`pf`’

forward predicted MVs of P-frames

‘`bf`’

forward predicted MVs of B-frames

‘`bb`’

backward predicted MVs of B-frames

‘`-cmp[:stream_specifier] *integer* (*output,video*)`’

full pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

‘`-subcmp[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

sub pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’
‘`-mbcmp[:stream_specifier] *integer* (*output,video*)`’

macroblock compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

‘`-ildctcmp[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

interlaced dct compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’
‘`-dia_size[:stream_specifier] *integer* (*output,video*)`’

diamond type & size for motion estimation

‘`-last_pred[:stream_specifier] *integer* (*output,video*)`’

amount of motion predictors from the previous frame

‘`-preme[:stream_specifier] *integer* (*output,video*)`’

pre motion estimation

‘`-precmp[:stream_specifier] *integer* (*output,video*)`’

pre motion estimation compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

'`bit`'

number of bits needed for the block

'`rd`'

rate distortion optimal, slow

'`zero`'

0

'`vsad`'

sum of absolute vertical differences

'`vsse`'

sum of squared vertical differences

'`nsse`'

noise preserving sum of squared differences

'`w53`'

5/3 wavelet, only used in snow

'`w97`'

9/7 wavelet, only used in snow

'`dctmax`'
'`chroma`'

'`-pre_dia_size[:stream_specifier]` *integer* (*output,video*)'

diamond type & size for motion estimation pre-pass

'`-subq[:stream_specifier]` *integer* (*output,video*)'

sub pel motion estimation quality

'`-dtg_active_format[:stream_specifier]` *integer* ()'
'`-me_range[:stream_specifier]` *integer* (*output,video*)'

limit motion vectors range (1023 for DivX player)

‘-ibias[:stream_specifier] *integer* (*output,video*)’

intra quant bias

‘-pbias[:stream_specifier] *integer* (*output,video*)’

inter quant bias

‘-color_table_id[:stream_specifier] *integer* ()’
‘-global_quality[:stream_specifier] *integer* (*output,audio,video*)’
‘-coder[:stream_specifier] *integer* (*output,video*)’

Possible values:

‘vlc’

variable length coder / huffman coder

‘ac’

arithmetic coder

‘raw’

raw (no encoding)

‘rle’

run-length coder

‘deflate’

deflate-based coder

‘-context[:stream_specifier] *integer* (*output,video*)’

context model

‘-slice_flags[:stream_specifier] *integer* ()’
‘-xvmc_acceleration[:stream_specifier] *integer* ()’
‘-mbd[:stream_specifier] *integer* (*output,video*)’

macroblock decision algorithm (high quality mode)

Possible values:

‘simple’

use mbcmp (default)

‘`bits`’

use fewest bits

‘`rd`’

use best rate distortion

‘`-stream_codec_tag[:stream_specifier]` *`integer`* `()`’
‘`-sc_threshold[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

scene change threshold

‘`-lmin[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

min lagrange factor (VBR)

‘`-lmax[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

max lagrange factor (VBR)

‘`-nr[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

noise reduction

‘`-rc_init_occupancy[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’

number of bits which should be loaded into the rc buffer before decoding starts

‘`-inter_threshold[:stream_specifier]` *`integer`* `(`*`output,video`*`)`’
‘`-flags2[:stream_specifier]` *`flags`* `(`*`input/output,audio,video`*`)`’

Possible values:

‘`fast`’

allow non spec compliant speedup tricks

‘`sgop`’

Deprecated, use mpegvideo private options instead

‘`noout`’

skip bitstream encoding

‘`local_header`’

place global headers at every keyframe instead of in extradata

‘`chunks`’

Frame data might be split into multiple chunks

‘`showall`’

Show all frames before the first keyframe

‘`skiprd`’

Deprecated, use mpegvideo private options instead

‘`-error[:stream_specifier]` *`integer`* (*`output,video`*)’
‘`-qns[:stream_specifier]` *`integer`* (*`output,video`*)’

deprecated, use mpegvideo private options instead

‘`-threads[:stream_specifier]` *`integer`* (*`input/output,video`*)’

Possible values:

‘`auto`’

detect a good number of threads

‘`-me_threshold[:stream_specifier]` *`integer`* (*`output,video`*)’

motion estimaton threshold

‘`-mb_threshold[:stream_specifier]` *`integer`* (*`output,video`*)’

macroblock threshold

‘`-dc[:stream_specifier]` *`integer`* (*`output,video`*)’

intra_dc_precision

‘`-nssew[:stream_specifier]` *`integer`* (*`output,video`*)’

nsse weight

‘`-skip_top[:stream_specifier]` *`integer`* (*`input,video`*)’

number of macroblock rows at the top which are skipped

‘`-skip_bottom[:stream_specifier]` *integer* (*input,video*)’

number of macroblock rows at the bottom which are skipped

‘`-profile[:stream_specifier]` *integer* (*output,audio,video*)’

Possible values:

‘`unknown`’
‘`aac_main`’
‘`aac_low`’
‘`aac_ssr`’
‘`aac_ltp`’
‘`aac_he`’
‘`aac_he_v2`’
‘`aac_ld`’
‘`aac_eld`’
‘`dts`’
‘`dts_es`’
‘`dts_96_24`’
‘`dts_hd_hra`’
‘`dts_hd_ma`’

‘`-level[:stream_specifier]` *integer* (*output,audio,video*)’

Possible values:

‘`unknown`’

‘`-lowres[:stream_specifier]` *integer* (*input,audio,video*)’

decode at 1= 1/2, 2=1/4, 3=1/8 resolutions

‘`-skip_threshold[:stream_specifier]` *integer* (*output,video*)’

frame skip threshold

‘`-skip_factor[:stream_specifier]` *integer* (*output,video*)’

frame skip factor

‘`-skip_exp[:stream_specifier]` *integer* (*output,video*)’

frame skip exponent

‘`-skipcmp[:stream_specifier]` *integer* (*output,video*)’

frame skip compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

‘`-border_mask[:stream_specifier] `*`float`*` (`*`output,video`*`)`’

increases the quantizer for macroblocks close to borders

‘`-mblmin[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

min macroblock lagrange factor (VBR)

‘`-mblmax[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

max macroblock lagrange factor (VBR)

‘`-mepc[:stream_specifier] `*`integer`*` (`*`output,video`*`)`’

motion estimation bitrate penalty compensation (1.0 = 256)

‘`-skip_loop_filter[:stream_specifier] `*`integer`*` (`*`input,video`*`)`’

Possible values:

‘`none`’
‘`default`’
‘`noref`’
‘`bidir`’
‘`nokey`’
‘`all`’

‘`-skip_idct[:stream_specifier] `*`integer`*` (`*`input,video`*`)`’

Possible values:

‘`none`’
‘`default`’
‘`noref`’
‘`bidir`’
‘`nokey`’
‘`all`’

‘`-skip_frame[:stream_specifier] `*`integer`*` (`*`input,video`*`)`’

Possible values:

`‘none’`
`‘default’`
`‘noref’`
`‘bidir’`
`‘nokey’`
`‘all’`

‘`-bidir_refine[:stream_specifier]` *integer* (*output,video*)’

refine the two motion vectors used in bidirectional macroblocks

‘`-brd_scale[:stream_specifier]` *integer* (*output,video*)’

downscales frames for dynamic B-frame decision

‘`-keyint_min[:stream_specifier]` *integer* (*output,video*)’

minimum interval between IDR-frames

‘`-refs[:stream_specifier]` *integer* (*output,video*)’

reference frames to consider for motion compensation

‘`-chromaoffset[:stream_specifier]` *integer* (*output,video*)’

chroma qp offset from luma

‘`-trellis[:stream_specifier]` *integer* (*output,audio,video*)’

rate-distortion optimal quantization

‘`-sc_factor[:stream_specifier]` *integer* (*output,video*)’

multiplied by qscale for each frame and added to scene_change_score

‘`-mv0_threshold[:stream_specifier]` *integer* (*output,video*)’
‘`-b_sensitivity[:stream_specifier]` *integer* (*output,video*)’

adjusts sensitivity of b_frame_strategy 1

‘`-compression_level[:stream_specifier]` *integer* (*output,audio,video*)’
‘`-min_prediction_order[:stream_specifier]` *integer* (*output,audio*)’
‘`-max_prediction_order[:stream_specifier]` *integer* (*output,audio*)’
‘`-timecode_frame_start[:stream_specifier]` *integer* (*output,video*)’

GOP timecode frame start number, in non drop frame format

‘`-request_channels[:stream_specifier]` *integer* (*input,audio*)’ set desired number of audio channels

‘-bits_per_raw_sample[:stream_specifier] *integer* ()’
‘-channel_layout[:stream_specifier] *integer* (*input/output,audio*)’

Possible values:

‘-request_channel_layout[:stream_specifier] *integer* (*input,audio*)’

Possible values:

‘-rc_max_vbv_use[:stream_specifier] *float* (*output,video*)’
‘-rc_min_vbv_use[:stream_specifier] *float* (*output,video*)’
‘-ticks_per_frame[:stream_specifier] *integer* (*input/output,audio,video*)’
‘-color_primaries[:stream_specifier] *integer* (*input/output,video*)’
‘-color_trc[:stream_specifier] *integer* (*input/output,video*)’
‘-colorspace[:stream_specifier] *integer* (*input/output,video*)’
‘-color_range[:stream_specifier] *integer* (*input/output,video*)’
‘-chroma_sample_location[:stream_specifier] *integer* (*input/output,video*)’
‘-log_level_offset[:stream_specifier] *integer* ()’

set the log level offset

‘-slices[:stream_specifier] *integer* (*output,video*)’

number of slices, used in parallelized encoding

‘-thread_type[:stream_specifier] *flags* (*input/output,video*)’

select multithreading type

Possible values:

‘slice’
‘frame’
‘-audio_service_type[:stream_specifier] *integer* (*output,audio*)’

audio service type

Possible values:

‘ma’

Main Audio Service

‘ef’

Effects

‘`vi`’

Visually Impaired

‘`hi`’

Hearing Impaired

‘`di`’

Dialogue

‘`co`’

Commentary

‘`em`’

Emergency

‘`vo`’

Voice Over

‘`ka`’

Karaoke

‘`-request_sample_fmt[:stream_specifier]` *`value`* `(`*`input,audio`*`)`’

sample format audio decoders should prefer

Possible values:

‘`-pkt_timebase[:stream_specifier]` *`rational number`* `()`’

5.5 Format AVOptions

‘`-avioflags` *`flags`* `(`*`input/output`*`)`’

Possible values:

‘`direct`’

reduce buffering

‘`-probesize` *`integer`* `(`*`input`*`)`’

set probing size

‘`-packetsize` *`integer`* `(`*`output`*`)`’

set packet size

‘`-fflags` *`flags`* `(`*`input/output`*`)`’

Possible values:

‘`ignidx`’

ignore index

‘`genpts`’

generate pts

‘`nofillin`’

do not fill in missing values that can be exactly calculated

‘`noparse`’

disable AVParsers, this needs nofillin too

‘`igndts`’

ignore dts

‘`discardcorrupt`’

discard corrupted frames

‘`sortdts`’

try to interleave outputted packets by dts

‘`keepside`’

dont merge side data

‘`latm`’

enable RTP MP4A-LATM payload

‘`nobuffer`’

reduce the latency introduced by optional buffering

‘`-analyzeduration` *`integer`* (*`input`*)’

how many microseconds are analyzed to estimate duration

‘`-cryptokey` *`hexadecimal string`* (*`input`*)’

decryption key

‘`-indexmem` *`integer`* (*`input`*)’

max memory used for timestamp index (per stream)

‘`-rtbufsize` *`integer`* (*`input`*)’

max memory used for buffering real-time frames

‘`-fdebug` *`flags`* (*`input/output`*)’

print specific debug info

Possible values:

‘`ts`’

‘`-max_delay` *`integer`* (*`input/output`*)’

maximum muxing or demuxing delay in microseconds

‘`-fpsprobesize` *`integer`* (*`input`*)’

number of frames used to probe fps

‘`-audio_preload` *`integer`* (*`output`*)’

microseconds by which audio packets should be interleaved earlier

‘`-chunk_duration` *`integer`* (*`output`*)’

microseconds for each chunk

‘`-chunk_size` *`integer`* (*`output`*)’

size in bytes for each chunk

‘`-f_err_detect` *`flags`* (*`input`*)’

set error detection flags (deprecated; use err_detect, save via avconv)

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

consider things that violate the spec and have not been seen in the wild as errors

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

‘`-err_detect` *`flags`* `(`*`input`*`)`’

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

consider things that violate the spec and have not been seen in the wild as errors

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

‘`-use_wallclock_as_timestamps` *integer* (*input*)’

use wallclock as timestamps

‘`-avoid_negative_ts` *integer* (*output*)’

avoid negative timestamps

‘`-skip_initial_bytes` *integer* (*input*)’

skip initial bytes

5.6 Main options

‘`-f` *fmt* (*input/output*)’

Force input or output file format. The format is normally auto detected for input files and guessed from file extension for output files, so this option is not needed in most cases.

‘`-i` *filename* (*input*)’

input file name

‘`-y` (*global*)’

Overwrite output files without asking.

‘`-n` (*global*)’

Do not overwrite output files but exit if file exists.

‘`-c[:`*stream_specifier*`]` *codec* (*input/output,per-stream*)’
‘`-codec[:`*stream_specifier*`]` *codec* (*input/output,per-stream*)’

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

'`-t` *`duration`* (*`output`*)'

Stop writing the output after its duration reaches *duration*. *duration* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

'`-fs` *`limit_size`* (*`output`*)'

Set the file size limit, expressed in bytes.

'`-ss` *`position`* (*`input/output`*)'

When used as an input option (before `-i`), seeks in this input file to *position*. When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*. This is slower, but more accurate.

position may be either in seconds or in `hh:mm:ss[.xxx]` form.

'`-itsoffset` *`offset`* (*`input`*)'

Set the input time offset in seconds. `[-]hh:mm:ss[.xxx]` syntax is also supported. The offset is added to the timestamps of the input files. Specifying a positive offset means that the corresponding streams are delayed by *offset* seconds.

'`-timestamp` *`time`* (*`output`*)'

Set the recording timestamp in the container. The syntax for *time* is:

```
now|([(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...])|(HHMMSS[.m...]))[Z|z])
```

If the value is "now" it takes the current time. Time is local time unless 'Z' or 'z' is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

'`-metadata[:metadata_specifier]` *key*`=`*value* (*output,per-metadata*)'

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams or chapters. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

```
ffmpeg -i in.avi -metadata title="my title" out.flv
```

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:1 language=eng OUTPUT
```

'`-target` *type* (*output*)'

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

```
ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg
```

Nevertheless you can specify additional options as long as you know they do not conflict with the standard, as in:

```
ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg
```

'`-dframes` *number* (*output*)'

Set the number of data frames to record. This is an alias for `-frames:d`.

'`-frames[:`*stream_specifier*`]` *framecount* (*output,per-stream*)'

Stop writing to the stream after *framecount* frames.

'`-q[:`*stream_specifier*`]` *q* (*output,per-stream*)'

‘`-qscale[:`*`stream_specifier`*`]` *`q`* `(`*`output,per-stream`*`)`’

Use fixed quality scale (VBR). The meaning of *q* is codec-dependent.

‘`-filter[:`*`stream_specifier`*`]` *`filter_graph`* `(`*`output,per-stream`*`)`’

filter_graph is a description of the filter graph to apply to the stream. Use `-filters` to show all the available filters (including also sources and sinks).

See also the ‘`-filter_complex`’ option if you want to create filter graphs with multiple inputs and/or outputs.

‘`-pre[:`*`stream_specifier`*`]` *`preset_name`* `(`*`output,per-stream`*`)`’

Specify the preset for matching stream(s).

‘`-stats (`*`global`*`)`’

Print encoding progress/statistics. On by default.

‘`-progress` *`url`* `(`*`global`*`)`’

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key*=*value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

‘`-stdin`’

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

‘`-debug_ts (`*`global`*`)`’

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option `-fdebug ts`.

‘`-attach` *`filename`* `(`*`output`*`)`’

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

‘`-dump_attachment[:`*`stream_specifier`*`]` *`filename`* `(`*`input,per-stream`*`)`’

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the `filename` metadata tag will be used.

E.g. to extract the first attachment to a file named ’out.ttf’:

```
ffmpeg -dump_attachment:t:0 out.ttf INPUT
```

To extract all attachments to files determined by the `filename` tag:

```
ffmpeg -dump_attachment:t "" INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

5.7 Video Options

‘`-vframes` *`number`* `(`*`output`*`)`’

Set the number of video frames to record. This is an alias for `-frames:v`.

‘`-r[:`*`stream_specifier`*`]` *`fps`* `(`*`input/output,per-stream`*`)`’

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps* (note that this actually causes the `fps` filter to be inserted to the end of the corresponding filtergraph).

‘`-s[:`*`stream_specifier`*`]` *`size`* `(`*`input/output,per-stream`*`)`’

Set frame size.

As an input option, this is a shortcut for the ‘`video_size`’ private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the `scale` video filter to the *end* of the corresponding filtergraph. Please use the `scale` filter directly to insert it at the beginning or some other place.

The format is ‘`wxh`’ (default - same as source).

‘`-aspect[:`*`stream_specifier`*`]` *`aspect`* `(`*`output,per-stream`*`)`’

Set the video display aspect ratio specified by *aspect*.

aspect can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

‘`-croptop` *`size`*’
‘`-cropbottom` *`size`*’
‘`-cropleft` *`size`*’
‘`-cropright` *`size`*’

All the crop options have been removed. Use -vf crop=width:height:x:y instead.

‘`-padtop` *`size`*’
‘`-padbottom` *`size`*’
‘`-padleft` *`size`*’
‘`-padright` *`size`*’
‘`-padcolor` *`hex_color`*’

All the pad options have been removed. Use -vf pad=width:height:x:y:color instead.

‘`-vn (`*`output`*`)`’

Disable video recording.

‘`-vcodec` *`codec`* `(`*`output`*`)`’

Set the video codec. This is an alias for `-codec:v`.

‘`-pass[:`*`stream_specifier`*`]` *`n`* `(`*`output,per-stream`*`)`’

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

```
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
ffmpeg -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y /dev/null
```

‘`-passlogfile[:`*`stream_specifier`*`]` *`prefix`* `(`*`output,per-stream`*`)`’

Set two-pass log file name prefix to *prefix*, the default file name prefix is “ffmpeg2pass”. The complete file name will be ‘`PREFIX-N.log`’, where N is a number specific to the output stream

‘`-vlang` *`code`*’

Set the ISO 639 language code (3 letters) of the current video stream.

‘`-vf` *`filter_graph`* `(`*`output`*`)`’

filter_graph is a description of the filter graph to apply to the input video. Use the option "-filters" to show all the available filters (including also sources and sinks). This is an alias for `-filter:v`.

5.8 Advanced Video Options

‘`-pix_fmt[:`*`stream_specifier`*`]` *`format`* `(`*`input/output,per-stream`*`)`’

Set pixel format. Use `-pix_fmts` to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a `+`, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions inside filter graphs are disabled. If *pix_fmt* is a single `+`, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

‘`-sws_flags` *`flags`* `(`*`input/output`*`)`’

Set SwScaler flags.

‘`-vdt` *`n`*’

Discard threshold.

‘`-rc_override[:`*`stream_specifier`*`]` *`override`* `(`*`output,per-stream`*`)`’

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

‘`-deinterlace`’

Deinterlace pictures. This option is deprecated since the deinterlacing is very low quality. Use the yadif filter with `-filter:v yadif`.

‘`-ilme`’

Force interlacing support in encoder (MPEG-2 and MPEG-4 only). Use this option if your input file is interlaced and you want to keep the interlaced format for minimum losses. The alternative is to deinterlace the input stream with ‘`-deinterlace`’, but deinterlacing introduces losses.

‘`-psnr`’

Calculate PSNR of compressed frames.

‘`-vstats`’

Dump video coding statistics to ‘`vstats_HHMMSS.log`’.

‘`-vstats_file` *`file`*’

Dump video coding statistics to *file*.

‘`-top[:`*`stream_specifier`*`]` *`n`* `(`*`output,per-stream`*`)`’

top=1/bottom=0/auto=-1 field first

‘`-dc` *`precision`*’

Intra_dc_precision.

‘`-vtag` *`fourcc/tag`* `(`*`output`*`)`’

Force video tag/fourcc. This is an alias for `-tag:v`.

‘`-qphist (`*`global`*`)`’

Show QP histogram

‘`-vbsf` *`bitstream_filter`*’

Deprecated see -bsf

‘`-force_key_frames[:`*`stream_specifier`*`]` *`time`*`[,`*`time`*`...]`
`(`*`output,per-stream`*`)`’

Force key frames at the specified timestamps, more precisely at the first frames after each specified time. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file. The timestamps must be specified in ascending order.

‘`-copyinkf[:`*`stream_specifier`*`] (`*`output,per-stream`*`)`’

When doing stream copy, copy also non-key frames found at the beginning.

5.9 Audio Options

‘`-aframes` *number* (*output*)’

> Set the number of audio frames to record. This is an alias for `-frames:a`.

‘`-ar[:`*stream_specifier*`]` *freq* (*input/output,per-stream*)’

> Set the audio sampling frequency. For output streams it is set by default to the frequency of the corresponding input stream. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

‘`-aq` *q* (*output*)’

> Set the audio quality (codec-specific, VBR). This is an alias for -q:a.

‘`-ac[:`*stream_specifier*`]` *channels* (*input/output,per-stream*)’

> Set the number of audio channels. For output streams it is set by default to the number of input audio channels. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

‘`-an` (*output*)’

> Disable audio recording.

‘`-acodec` *codec* (*input/output*)’

> Set the audio codec. This is an alias for `-codec:a`.

‘`-sample_fmt[:`*stream_specifier*`]` *sample_fmt* (*output,per-stream*)’

> Set the audio sample format. Use `-sample_fmts` to get a list of supported sample formats.

‘`-af` *filter_graph* (*output*)’

> *filter_graph* is a description of the filter graph to apply to the input audio. Use the option "-filters" to show all the available filters (including also sources and sinks). This is an alias for `-filter:a`.

5.10 Advanced Audio options:

‘`-atag` *fourcc/tag* (*output*)’

> Force audio tag/fourcc. This is an alias for `-tag:a`.

‘`-absf` *bitstream_filter*’

Deprecated, see -bsf

5.11 Subtitle options:

‘`-slang` *`code`*’

Set the ISO 639 language code (3 letters) of the current subtitle stream.

‘`-scodec` *`codec`* `(`*`input/output`*`)`’

Set the subtitle codec. This is an alias for `-codec:s`.

‘`-sn (`*`output`*`)`’

Disable subtitle recording.

‘`-sbsf` *`bitstream_filter`*’

Deprecated, see -bsf

5.12 Advanced Subtitle options:

‘`-fix_sub_duration`’

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

5.13 Advanced options

‘`-map [-]`*`input_file_id`*`[:`*`stream_specifier`*`][,`*`sync_file_id`*`[:`*`stream_specifier`*`]] |` *`[linklabel]`* `(`*`output`*`)`’

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A `-` character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the ‘`-filter_complex`’ option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in ‘`INPUT`’ identified by "0:1" to the (single) output stream in ‘`out.wav`’.

For example, to select the stream with index 2 from input file ‘`a.mov`’ (specified by the identifier "0:2"), and stream with index 6 from input ‘`b.mov`’ (specified by the identifier "1:6"), and copy them to the output file ‘`out.mov`’:

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

```
ffmpeg -i INPUT -map 0:v -map 0:a:2 OUTPUT
```

To map all the streams except the second audio, use negative mappings

```
ffmpeg -i INPUT -map 0 -map -0:a:1 OUTPUT
```

Note that using this option disables the default mappings for this output file.

‘`-map_channel [`*`input_file_id.stream_specifier.channel_id`*`|-1][:`*`output_file_id.stream_specifier`*`]`’

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don't match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can't for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here '`input.mkv`') with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

'`-map_metadata[:`*`metadata_spec_out`*`]` *`infile`*`[:`*`metadata_spec_in`*`] (`*`output,per-metadata`*`)`'

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

'*`g`*'

global metadata, i.e. metadata that applies to the whole file

'`s[:`*`stream_spec`*`]`'

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

'`c:`*`chapter_index`*'

per-chapter metadata. *chapter_index* is the zero-based chapter index.

'`p:`*`program_index`*'

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

```
ffmpeg -i in.ogg -map_metadata 0:s:0 out.mp3
```

To do the reverse, i.e. copy global metadata to all audio streams:

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

'`-map_chapters` *`input_file_index`* `(`*`output`*`)`'

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

'`-benchmark (`*`global`*`)`'

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

‘`-benchmark_all (`*`global`*`)`’

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

‘`-timelimit` *`duration`* `(`*`global`*`)`’

Exit after ffmpeg has been running for *duration* seconds.

‘`-dump (`*`global`*`)`’

Dump each input packet to stderr.

‘`-hex (`*`global`*`)`’

When dumping packets, also dump the payload.

‘`-re (`*`input`*`)`’

Read input at native frame rate. Mainly used to simulate a grab device. By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming). If your input(s) is coming from some other live streaming source (through HTTP or UDP for example) the server might already be in real-time, thus the option will likely not be required. On the other hand, this is meaningful if your input(s) is a file you are trying to push in real-time.

‘`-loop_input`’

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

‘`-loop_output` *`number_of_times`*’

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

‘`-vsync` *`parameter`*’

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

‘`0, passthrough`’

Each frame is passed with its timestamp from the demuxer to the muxer.

‘`1, cfr`’

Frames will be duplicated and dropped to achieve exactly the requested constant framerate.

‘`2, vfr`’

Frames are passed through with their timestamp or dropped so as to prevent 2 frames from having the same timestamp.

‘`drop`’

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

‘`-1, auto`’

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

‘`-async` *`samples_per_second`*’

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction. This option has been deprecated. Use the `asyncts` audio filter instead.

‘`-copyts`’

Copy timestamps from input to output.

‘`-copytb` *`mode`*’

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

‘`1`’

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid non monotonically increasing timestamps when copying video streams with variable frame rate.

‘`0`’

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

‘-1’

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

‘-shortest (*output*)’

Finish encoding when the shortest input stream ends.

‘-dts_delta_threshold’

Timestamp discontinuity delta threshold.

‘-muxdelay *seconds* (*input*)’

Set the maximum demux-decode delay.

‘-muxpreload *seconds* (*input*)’

Set the initial demux-decode delay.

‘-streamid *output-stream-index*:*new-value* (*output*)’

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

```
ffmpeg -i infile -streamid 0:33 -streamid 1:36 out.ts
```

‘-bsf[:*stream_specifier*] *bitstream_filters* (*output,per-stream*)’

Set bitstream filters for matching streams. *bistream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

```
ffmpeg -i h264.mp4 -c:v copy -bsf:v h264_mp4toannexb -an out.h264
```

```
ffmpeg -i file.mov -an -vn -bsf:s mov2textsub -c:s copy -f rawvideo sub.txt
```

‘-tag[:*stream_specifier*] *codec_tag* (*per-stream*)’

Force a tag/fourcc for matching streams.

'-timecode *hh*:*mm*:*ss*SEP*ff*'

Specify Timecode for writing. *SEP* is ':' for non drop timecode and ';' (or '.') for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

'-filter_complex *filtergraph* (*global*)'

Define a complex filter graph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the '-filter' options. *filtergraph* is a description of the filter graph, as described in Filtergraph syntax.

Input link labels must refer to input streams using the `[file_index:stream_specifier]` syntax (i.e. the same as '-map' uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with '-map'. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map
'[out]' out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map
'[out]' out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
```

To generate 5 seconds of pure red video using lavfi `color` source:

```
ffmpeg -filter_complex 'color=red' -t 5 out.mkv
```

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720Ã576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

5.14 Preset files

A preset file contains a sequence of *option*=*value* pairs, one for each line, specifying a sequence of options which would be awkward to specify on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `vpre`, `apre`, `spre`, and `fpre` options. The `fpre` option takes the filename of the preset instead of a preset name as input and can be used for any kind of codec. For the `vpre`, `apre`, and `spre` options, the options specified in a preset file are applied to the currently selected codec of the same type as the preset option.

The argument passed to the `vpre`, `apre`, and `spre` preset options identifies the preset file to use according to the following rules:

First ffmpeg searches for a file named *arg*.ffpreset in the directories '`$FFMPEG_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the datadir defined at configuration time (usually '`PREFIX/share/ffmpeg`') or in a '`ffpresets`' folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file '`libvpx-1080p.ffpreset`'.

If no such file is found, then ffmpeg will search for a file named *codec_name*-*arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file '`libvpx-1080p.ffpreset`'.

6. Tips

- For streaming at very low bitrate application, use a low frame rate and a small GOP size. This is especially true for RealVideo where the Linux player does not seem to be very fast, so it can miss frames. An example is:

```
        ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
```

- The parameter ’q’ which is displayed while encoding is the current quantizer. The value 1 indicates that a very good quality could be achieved. The value 31 indicates the worst quality. If q=31 appears too often, it means that the encoder cannot compress enough to meet your bitrate. You must either increase the bitrate, decrease the frame rate or decrease the frame size.
- If your computer is not fast enough, you can speed up the compression at the expense of the compression ratio. You can use ’-me zero’ to speed up motion estimation, and ’-g 0’ to disable motion estimation completely (you have only I-frames, which means it is about as good as JPEG compression).
- To have very low audio bitrates, reduce the sampling frequency (down to 22050 Hz for MPEG audio, 22050 or 11025 for AC-3).
- To have a constant quality (but a variable bitrate), use the option ’-qscale n’ when ’n’ is between 1 (excellent quality) and 31 (worst quality).

7. Examples

7.1 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which can be specified also on the command line. Lines starting with the hash (’#’) character are ignored and are used to provide comments. Empty lines are also ignored. Check the ‘`presets`’ directory in the FFmpeg source tree for examples.

Preset files are specified with the `pre` option, this option takes a preset name as input. FFmpeg searches for a file named *preset_name*.avpreset in the directories ‘`$AVCONV_DATADIR`’ (if set), and ‘`$HOME/.ffmpeg`’, and in the data directory defined at configuration time (usually ‘`$PREFIX/share/ffmpeg`’) in that order. For example, if the argument is `libx264-max`, it will search for the file ‘`libx264-max.avpreset`’.

7.2 Video and Audio grabbing

If you specify the input format and device then ffmpeg can grab video and audio directly.

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing

Grab the X11 display with ffmpeg via

```
ffmpeg -f x11grab -s cif -r 25 -i :0.0 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable.

```
ffmpeg -f x11grab -s cif -r 25 -i :0.0+10,20 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable. 10 is the x-offset and 20 the y-offset for the grabbing.

7.4 Video and Audio file format conversion

Any supported file format and protocol can serve as input to ffmpeg:

Examples:

- You can use YUV files as input:

  ```
  ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
  ```

  It will use the files:

  ```
  /tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
  /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
  ```

  The Y files use twice the resolution of the U and V files. They are raw files, without header. They can be generated by all decent video decoders. You must specify the size of the image with the ‘`-s`’ option if ffmpeg cannot guess it.

- You can input from a raw YUV420P file:

  ```
  ffmpeg -i /tmp/test.yuv /tmp/out.avi
  ```

  test.yuv is a file containing raw YUV planar data. Each frame is composed of the Y plane followed by the U and V planes at half vertical and horizontal resolution.

- You can output to a raw YUV420P file:

  ```
  ffmpeg -i mydivx.avi hugefile.yuv
  ```

- You can set several input files and output files:

  ```
  ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
  ```

  Converts the audio file a.wav and the raw YUV video file a.yuv to MPEG file a.mpg.

- You can also do audio and video conversions at the same time:

  ```
  ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
  ```

  Converts a.wav to MPEG audio at 22050 Hz sample rate.

- You can encode to several formats at the same time and define a mapping from input stream to output streams:

  ```
  ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
  ```

  Converts a.wav to a.mp2 at 64 kbits and to b.mp2 at 128 kbits. '-map file:index' specifies which input stream is used for each output stream, in the order of the definition of output streams.

- You can transcode decrypted VOBs:

  ```
  ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
  ```

  This is a typical DVD ripping example; the input is a VOB file, the output an AVI file with MPEG-4 video and MP3 audio. Note that in this command we use B-frames so the MPEG-4 stream is DivX5 compatible, and GOP size is 300 which means one intra frame every 10 seconds for 29.97fps input video. Furthermore, the audio stream is MP3-encoded so you need to enable LAME support by passing `--enable-libmp3lame` to configure. The mapping is particularly useful for DVD transcoding to get the desired audio language.

  NOTE: To see the supported input formats, use `ffmpeg -formats`.

- You can extract images from a video, or create a video from many images:

  For extracting images from a video:

  ```
  ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
  ```

  This will extract one video frame per second from the video and will output them in files named '`foo-001.jpeg`', '`foo-002.jpeg`', etc. Images will be rescaled to fit the new WxH values.

  If you want to extract just a limited number of frames, you can use the above command in combination with the -vframes or -t option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

```
ffmpeg -f image2 -i foo-%03d.jpeg -r 12 -s WxH foo.avi
```

The syntax `foo-%03d.jpeg` specifies to use a decimal number composed of three digits padded with zeroes to express the sequence number. It is the same syntax supported by the C printf function, but only formats accepting a normal integer are suitable.

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -i 'foo-*.jpeg' -r 12 -s WxH foo.avi
```

- You can put many streams of the same type in the output:

  ```
  ffmpeg -i test1.avi -i test2.avi -map 0.3 -map 0.2 -map 0.1 -map 0.0 -c copy test12.nut
  ```

  The resulting output file ‘`test12.avi`’ will contain first four streams from the input file in reverse order.

- To force CBR video output:

  ```
  ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
  ```

- The four options lmin, lmax, mblmin and mblmax use ’lambda’ units, but you may use the QP2LAMBDA constant to easily convert from ’q’ units:

  ```
  ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
  ```

8. Syntax

This section documents the syntax and formats employed by the FFmpeg libraries and tools.

8.1 Quoting and escaping

FFmpeg adopts the following quoting and escaping mechanism, unless explicitly specified. The following rules are applied:

- `'` and `\` are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.

- A special character is escaped by prefixing it with a '\'.
- All characters enclosed between '' are included literally in the parsed string. The quote character `'` itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

Note that you may need to add a second level of escaping when using the command line or a script, which depends on the syntax of the adopted shell language.

The function `av_get_token` defined in '`libavutil/avstring.h`' can be used to parse a token quoted or escaped according to the rules defined above.

The tool '`tools/ffescape`' in the FFmpeg source tree can be used to automatically quote or escape a string in a script.

8.1.1 Examples

- Escape the string `Crime d'Amour` containing the `'` special character:

  ```
  Crime d\'Amour
  ```

- The string above contains a quote, so the `'` needs to be escaped when quoting it:

  ```
  'Crime d'\''Amour'
  ```

- Include leading or trailing whitespaces using quoting:

  ```
  '  this string starts and ends with whitespaces  '
  ```

- Escaping and quoting can be mixed together:

  ```
  ' The string '\'string\'' is a string '
  ```

- To include a literal `\` you can use either escaping or quoting:

  ```
  'c:\foo' can be written as c:\\foo
  ```

8.2 Date

The accepted syntax is:

```
[(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...]]])|(HHMMSS[.m...]]]))[Z]
now
```

If the value is "now" it takes the current time.

Time is local time unless Z is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

8.3 Time duration

The accepted syntax is:

```
[-]HH:MM:SS[.m...]
[-]S+[.m...]
```

HH expresses the number of hours, *MM* the number a of minutes and *SS* the number of seconds.

8.4 Video size

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation.

The following abbreviations are recognized:

‘`sqcif`’

   128x96

‘`qcif`’

   176x144

‘`cif`’

   352x288

‘`4cif`’

   704x576

‘`16cif`’

   1408x1152

‘`qqvga`’

   160x120

‘`qvga`’

320x240

‘`vga`’

640x480

‘`svga`’

800x600

‘`xga`’

1024x768

‘`uxga`’

1600x1200

‘`qxga`’

2048x1536

‘`sxga`’

1280x1024

‘`qsxga`’

2560x2048

‘`hsxga`’

5120x4096

‘`wvga`’

852x480

‘`wxga`’

1366x768

‘`wsxga`’

1600x1024

‘`wuxga`’

1920x1200

‘`woxga`’

2560x1600

‘`wqsxga`’

3200x2048

‘`wquxga`’

3840x2400

‘`whsxga`’

6400x4096

‘`whuxga`’

7680x4800

‘`cga`’

320x200

‘`ega`’

640x350

‘`hd480`’

852x480

‘`hd720`’

1280x720

‘`hd1080`’

1920x1080

8.5 Video rate

Specify the frame rate of a video, expressed as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation.

The following abbreviations are recognized:

‘`ntsc`’

 30000/1001

‘`pal`’

 25/1

‘`qntsc`’

 30000/1

‘`qpal`’

 25/1

‘`sntsc`’

 30000/1

‘`spal`’

 25/1

‘`film`’

 24/1

‘`ntsc-film`’

 24000/1

8.6 Ratio

A ratio can be expressed as an expression, or in the form *numerator*:*denominator*.

Note that a ratio with infinite (1/0) or negative value is considered valid, so you should check on the returned value if you want to exclude those values.

The undefined value can be expressed using the "0:0" string.

8.7 Color

It can be the name of a color (case insensitive match) or a [0x|#]RRGGBB[AA] sequence, possibly followed by "@" and a string representing the alpha component.

The alpha component may be a string composed by "0x" followed by an hexadecimal number or a decimal number between 0.0 and 1.0, which represents the opacity value (0x00/0.0 means completely transparent, 0xff/1.0 completely opaque). If the alpha component is not specified then 0xff is assumed.

The string "random" will result in a random color.

9. Expression Evaluation

When evaluating an arithmetic expression, FFmpeg uses an internal formula evaluator, implemented through the ‘`libavutil/eval.h`’ interface.

An expression may contain unary, binary operators, constants, and functions.

Two expressions *expr1* and *expr2* can be combined to form another expression "*expr1*;*expr2*". *expr1* and *expr2* are evaluated in turn, and the new expression evaluates to the value of *expr2*.

The following binary operators are available: `+`, `-`, `*`, `/`, `^`.

The following unary operators are available: `+`, `-`.

The following functions are available:

‘`sinh(x)`’

Compute hyperbolic sine of *x*.

‘`cosh(x)`’

Compute hyperbolic cosine of *x*.

‘`tanh(x)`’

Compute hyperbolic tangent of *x*.

‘`sin(x)`’

Compute sine of *x*.

‘`cos(x)`’

Compute cosine of *x*.

‘`tan(x)`’

Compute tangent of *x*.

‘`atan(x)`’

Compute arctangent of *x*.

‘`asin(x)`’

Compute arcsine of *x*.

‘`acos(x)`’

Compute arccosine of *x*.

‘`exp(x)`’

Compute exponential of *x* (with base `e`, the Euler’s number).

‘`log(x)`’

Compute natural logarithm of *x*.

‘`abs(x)`’

Compute absolute value of *x*.

‘`squish(x)`’

Compute expression `1/(1 + exp(4*x))`.

‘`gauss(x)`’

Compute Gauss function of *x*, corresponding to `exp(-x*x/2) / sqrt(2*PI)`.

‘`isinf(x)`’

Return 1.0 if *x* is +/-INFINITY, 0.0 otherwise.

‘`isnan(x)`’

Return 1.0 if *x* is NAN, 0.0 otherwise.

‘`mod(x, y)`’

Compute the remainder of division of *x* by *y*.

‘`max(x, y)`’

Return the maximum between *x* and *y*.

‘`min(x, y)`’

Return the maximum between *x* and *y*.

‘`eq(x, y)`’

Return 1 if *x* and *y* are equivalent, 0 otherwise.

‘`gte(x, y)`’

Return 1 if *x* is greater than or equal to *y*, 0 otherwise.

‘`gt(x, y)`’

Return 1 if *x* is greater than *y*, 0 otherwise.

‘`lte(x, y)`’

Return 1 if *x* is lesser than or equal to *y*, 0 otherwise.

‘`lt(x, y)`’

Return 1 if *x* is lesser than *y*, 0 otherwise.

‘`st(var, expr)`’

Allow to store the value of the expression *expr* in an internal variable. *var* specifies the number of the variable where to store the value, and it is a value ranging from 0 to 9. The function returns the value stored in the internal variable. Note, Variables are currently not shared between expressions.

‘`ld(var)`’

Allow to load the value of the internal variable with number *var*, which was previously stored with st(*var*, *expr*). The function returns the loaded value.

‘`while(cond, expr)`’

Evaluate expression *expr* while the expression *cond* is non-zero, and returns the value of the last *expr* evaluation, or NAN if *cond* was always false.

‘`ceil(expr)`’

Round the value of expression *expr* upwards to the nearest integer. For example, "ceil(1.5)" is "2.0".

‘`floor(expr)`’

Round the value of expression *expr* downwards to the nearest integer. For example, "floor(-1.5)" is "-2.0".

‘`trunc(expr)`’

Round the value of expression *expr* towards zero to the nearest integer. For example, "trunc(-1.5)" is "-1.0".

‘`sqrt(expr)`’

Compute the square root of *expr*. This is equivalent to "(*expr*)^.5".

‘`not(expr)`’

Return 1.0 if *expr* is zero, 0.0 otherwise.

‘`pow(x, y)`’

Compute the power of *x* elevated *y*, it is equivalent to "(*x*)^(*y*)".

‘`random(x)`’

Return a pseudo random value between 0.0 and 1.0. *x* is the index of the internal variable which will be used to save the seed/state.

‘`hypot(x, y)`’

This function is similar to the C function with the same name; it returns "sqrt(*x***x* + *y***y*)", the length of the hypotenuse of a right triangle with sides of length *x* and *y*, or the distance of the point (*x*, *y*) from the origin.

‘`gcd(x, y)`’

Return the greatest common divisor of *x* and *y*. If both *x* and *y* are 0 or either or both are less than zero then behavior is undefined.

‘`if(x, y)`’

Evaluate *x*, and if the result is non-zero return the result of the evaluation of *y*, return 0 otherwise.

‘`ifnot(x, y)`’

Evaluate *x*, and if the result is zero return the result of the evaluation of *y*, return 0 otherwise.

‘`taylor(expr, x) taylor(expr, x, id)`’

Evaluate a taylor series at x. expr represents the LD(id)-th derivates of f(x) at 0. If id is not specified then 0 is assumed. note, when you have the derivatives at y instead of 0 taylor(expr, x-y) can be used When the series does not converge the results are undefined.

‘`root(expr, max)`’

Finds x where f(x)=0 in the interval 0..max. f() must be continuous or the result is undefined.

The following constants are available:

'PI'

area of the unit disc, approximately 3.14

'E'

exp(1) (Euler's number), approximately 2.718

'PHI'

golden ratio (1+sqrt(5))/2, approximately 1.618

Assuming that an expression is considered "true" if it has a non-zero value, note that:

`*` works like AND

`+` works like OR

and the construct:

```
if A then B else C
```

is equivalent to

```
if(A,B) + ifnot(A,C)
```

In your C code, you can extend the list of unary and binary functions, and define recognized constants, so that they are available for your expressions.

The evaluator also recognizes the International System number postfixes. If 'i' is appended after the postfix, powers of 2 are used instead of powers of 10. The 'B' postfix multiplies the value for 8, and can be appended after another postfix or used alone. This allows using for example 'KB', 'MiB', 'G' and 'B' as postfix.

Follows the list of available International System postfixes, with indication of the corresponding powers of 10 and of 2.

'y'

-24 / -80

'z'

-21 / -70

‘a’

-18 / -60

‘f’

-15 / -50

‘p’

-12 / -40

‘n’

-9 / -30

‘u’

-6 / -20

‘m’

-3 / -10

‘c’

-2

‘d’

-1

‘h’

2

‘k’

3 / 10

‘K’

3 / 10

‘M’

6 / 20

‘G’

9 / 30

‘`T`’

12 / 40

‘`P`’

15 / 40

‘`E`’

18 / 50

‘`Z`’

21 / 60

‘`Y`’

24 / 70

10. Decoders

Decoders are configured elements in FFmpeg which allow the decoding of multimedia streams.

When you configure your FFmpeg build, all the supported native decoders are enabled by default. Decoders requiring an external library must be enabled manually via the corresponding `--enable-lib` option. You can list all available decoders using the configure option `--list-decoders`.

You can disable all the decoders with the configure option `--disable-decoders` and selectively enable / disable single decoders with the options `--enable-decoder=`*`DECODER`* / `--disable-decoder=`*`DECODER`*.

The option `-codecs` of the ff* tools will display the list of enabled decoders.

11. Video Decoders

A description of some of the currently available video decoders follows.

11.1 rawvideo

Raw video decoder.

This decoder decodes rawvideo streams.

11.1.1 Options

‘`top` *`top_field_first`*’

Specify the assumed field type of the input video.

‘`-1`’

the video is assumed to be progressive (default)

‘`0`’

bottom-field-first is assumed

‘`1`’

top-field-first is assumed

12. Audio Decoders

12.1 ffwavesynth

Internal wave synthetizer.

This decoder generates wave patterns according to predefined sequences. Its use is purely internal and the format of the data it accepts is not publicly documented.

13. Encoders

Encoders are configured elements in FFmpeg which allow the encoding of multimedia streams.

When you configure your FFmpeg build, all the supported native encoders are enabled by default. Encoders requiring an external library must be enabled manually via the corresponding `--enable-lib` option. You can list all available encoders using the configure option `--list-encoders`.

You can disable all the encoders with the configure option `--disable-encoders` and selectively enable / disable single encoders with the options `--enable-encoder=`*`ENCODER`* / `--disable-encoder=`*`ENCODER`*.

The option `-codecs` of the ff* tools will display the list of enabled encoders.

14. Audio Encoders

A description of some of the currently available audio encoders follows.

14.1 ac3 and ac3_fixed

AC-3 audio encoders.

These encoders implement part of ATSC A/52:2010 and ETSI TS 102 366, as well as the undocumented RealAudio 3 (a.k.a. dnet).

The *ac3* encoder uses floating-point math, while the *ac3_fixed* encoder only uses fixed-point integer math. This does not mean that one is always faster, just that one or the other may be better suited to a particular system. The floating-point encoder will generally produce better quality audio for a given bitrate. The *ac3_fixed* encoder is not the default codec for any of the output formats, so it must be specified explicitly using the option `-acodec ac3_fixed` in order to use it.

14.1.1 AC-3 Metadata

The AC-3 metadata options are used to set parameters that describe the audio, but in most cases do not affect the audio encoding itself. Some of the options do directly affect or influence the decoding and playback of the resulting bitstream, while others are just for informational purposes. A few of the options will add bits to the output stream that could otherwise be used for audio data, and will thus affect the quality of the output. Those will be indicated accordingly with a note in the option list below.

These parameters are described in detail in several publicly-available documents.

- A/52:2010 - Digital Audio Compression (AC-3) (E-AC-3) Standard
- A/54 - Guide to the Use of the ATSC Digital Television Standard
- Dolby Metadata Guide
- Dolby Digital Professional Encoding Guidelines

14.1.1.1 Metadata Control Options

‘`-per_frame_metadata` *`boolean`*’

Allow Per-Frame Metadata. Specifies if the encoder should check for changing metadata for each frame.

‘`0`’

The metadata values set at initialization will be used for every frame in the stream. (default)

‘`1`’

Metadata values can be changed before encoding each frame.

14.1.1.2 Downmix Levels

‘`-center_mixlev *level*`’

Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo. This field will only be written to the bitstream if a center channel is present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6dB gain

‘`-surround_mixlev *level*`’

Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo. This field will only be written to the bitstream if one or more surround channels are present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.500`’

Apply -6dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

14.1.1.3 Audio Production Information

Audio Production Information is optional information describing the mixing environment. Either none or both of the fields are written to the bitstream.

‘`-mixing_level *number*`’

Mixing Level. Specifies peak sound pressure level (SPL) in the production environment when the mix was mastered. Valid values are 80 to 111, or -1 for unknown or not indicated. The default value is -1, but that value cannot be used if the Audio Production Information is written to the bitstream. Therefore, if the `room_type` option is not the default value, the `mixing_level` option must not be -1.

‘`-room_type` *`type`*’

Room Type. Describes the equalization used during the final mixing session at the studio or on the dubbing stage. A large room is a dubbing stage with the industry standard X-curve equalization; a small room has flat equalization. This field will not be written to the bitstream if both the `mixing_level` option and the `room_type` option have the default values.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`large`’

Large Room

‘`2`’
‘`small`’

Small Room

14.1.1.4 Other Metadata Options

‘`-copyright` *`boolean`*’

Copyright Indicator. Specifies whether a copyright exists for this audio.

‘`0`’
‘`off`’

No Copyright Exists (default)

‘`1`’
‘`on`’

Copyright Exists

‘`-dialnorm` *`value`*’

Dialogue Normalization. Indicates how far the average dialogue level of the program is below digital 100% full scale (0 dBFS). This parameter determines a level shift during audio reproduction that sets the average volume of the dialogue to a preset level. The goal is to match volume level between program sources. A value of -31dB will result in no volume level change, relative to the source volume, during audio reproduction. Valid values are whole numbers in the range -31 to -1, with -31 being the default.

‘`-dsur_mode` *`mode`*’

Dolby Surround Mode. Specifies whether the stereo signal uses Dolby Surround (Pro Logic). This field will only be written to the bitstream if the audio stream is stereo. Using this option does **NOT** mean the encoder will actually apply Dolby Surround processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`off`’

Not Dolby Surround Encoded

‘`2`’
‘`on`’

Dolby Surround Encoded

‘`-original` *`boolean`*’

Original Bit Stream Indicator. Specifies whether this audio is from the original source and not a copy.

‘`0`’
‘`off`’

Not Original Source

‘`1`’
‘`on`’

Original Source (default)

14.1.2 Extended Bitstream Information

The extended bitstream options are part of the Alternate Bit Stream Syntax as specified in Annex D of the A/52:2010 standard. It is grouped into 2 parts. If any one parameter in a group is specified, all values in that group will be written to the bitstream. Default values are used for those that are written but have not been specified. If the mixing levels are written, the decoder will use these values instead of the ones specified in the `center_mixlev` and `surround_mixlev` options if it supports the Alternate Bit Stream Syntax.

14.1.2.1 Extended Bitstream Information - Part 1

‘`-dmix_mode` *`mode`*’

Preferred Stereo Downmix Mode. Allows the user to select either Lt/Rt (Dolby Surround) or Lo/Ro (normal stereo) as the preferred stereo downmix mode.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`ltrt`’

Lt/Rt Downmix Preferred

‘`2`’
‘`loro`’

Lo/Ro Downmix Preferred

‘`-ltrt_cmixlev` *`level`*’

Lt/Rt Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo in Lt/Rt mode.

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-ltrt_surmixlev` *`level`*’

Lt/Rt Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo in Lt/Rt mode.

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

‘`-loro_cmixlev` *`level`*’

Lo/Ro Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo in Lo/Ro mode.

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-loro_surmixlev` *`level`*’

Lo/Ro Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo in Lo/Ro mode.

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

14.1.2.2 Extended Bitstream Information - Part 2

‘`-dsurex_mode` *`mode`*’

Dolby Surround EX Mode. Indicates whether the stream uses Dolby Surround EX (7.1 matrixed to 5.1). Using this option does **NOT** mean the encoder will actually apply Dolby Surround EX processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`on`’

Dolby Surround EX Off

‘`2`’
‘`off`’

Dolby Surround EX On

‘`-dheadphone_mode` *`mode`*’

Dolby Headphone Mode. Indicates whether the stream uses Dolby Headphone encoding (multi-channel matrixed to 2.0 for use with headphones). Using this option does **NOT** mean the encoder will actually apply Dolby Headphone processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`on`’

Dolby Headphone Off

‘`2`’
‘`off`’

Dolby Headphone On

‘`-ad_conv_type` *`type`*’

A/D Converter Type. Indicates whether the audio has passed through HDCD A/D conversion.

‘0’
‘standard’

Standard A/D Converter (default)

‘1’
‘hdcd’

HDCD A/D Converter

14.1.3 Other AC-3 Encoding Options

‘-stereo_rematrixing *boolean*’

Stereo Rematrixing. Enables/Disables use of rematrixing for stereo input. This is an optional AC-3 feature that increases quality by selectively encoding the left/right channels as mid/side. This option is enabled by default, and it is highly recommended that it be left as enabled except for testing purposes.

14.1.4 Floating-Point-Only AC-3 Encoding Options

These options are only valid for the floating-point encoder and do not exist for the fixed-point encoder due to the corresponding features not being implemented in fixed-point.

‘-channel_coupling *boolean*’

Enables/Disables use of channel coupling, which is an optional AC-3 feature that increases quality by combining high frequency information from multiple channels into a single channel. The per-channel high frequency information is sent with less accuracy in both the frequency and time domains. This allows more bits to be used for lower frequencies while preserving enough information to reconstruct the high frequencies. This option is enabled by default for the floating-point encoder and should generally be left as enabled except for testing purposes or to increase encoding speed.

‘-1’
‘auto’

Selected by Encoder (default)

‘0’
‘off’

Disable Channel Coupling

‘1’
‘on’

Enable Channel Coupling

‘`-cpl_start_band` *number*’

Coupling Start Band. Sets the channel coupling start band, from 1 to 15. If a value higher than the bandwidth is used, it will be reduced to 1 less than the coupling end band. If *auto* is used, the start band will be determined by the encoder based on the bit rate, sample rate, and channel layout. This option has no effect if channel coupling is disabled.

‘`-1`’
‘`auto`’

Selected by Encoder (default)

15. Video Encoders

A description of some of the currently available video encoders follows.

15.1 libtheora

Theora format supported through libtheora.

Requires the presence of the libtheora headers and library during configuration. You need to explicitly configure the build with `--enable-libtheora`.

15.1.1 Options

The following global options are mapped to internal libtheora options which affect the quality and the bitrate of the encoded stream.

‘`b`’

Set the video bitrate, only works if the `qscale` flag in ‘`flags`’ is not enabled.

‘`flags`’

Used to enable constant quality mode encoding through the ‘`qscale`’ flag, and to enable the `pass1` and `pass2` modes.

‘`g`’

Set the GOP size.

‘`global_quality`’

Set the global quality in lambda units, only works if the `qscale` flag in ‘`flags`’ is enabled. The value is clipped in the [0 - 10*`FF_QP2LAMBDA`] range, and then multiplied for 6.3 to get a value in the native libtheora range [0-63].

For example, to set maximum constant quality encoding with `ffmpeg`:

```
ffmpeg -i INPUT -flags:v qscale -global_quality:v "10*QP2LAMBDA" -codec:v libtheora OUTPUT.ogg
```

15.2 libvpx

VP8 format supported through libvpx.

Requires the presence of the libvpx headers and library during configuration. You need to explicitly configure the build with `--enable-libvpx`.

15.2.1 Options

Mapping from FFmpeg to libvpx options with conversion notes in parentheses.

‘`threads`’

g_threads

‘`profile`’

g_profile

‘`vb`’

rc_target_bitrate

‘`g`’

kf_max_dist

‘`keyint_min`’

kf_min_dist

‘`qmin`’

rc_min_quantizer

‘`qmax`’

rc_max_quantizer

‘`bufsize, vb`’

rc_buf_sz (`bufsize * 1000 / vb`)

rc_buf_optimal_sz (`bufsize * 1000 / vb * 5 / 6`)

‘`rc_init_occupancy, vb`’

rc_buf_initial_sz (`rc_init_occupancy * 1000 / vb`)

‘`rc_buffer_aggressivity`’

rc_undershoot_pct

‘`skip_threshold`’

rc_dropframe_thresh

‘`qcomp`’

rc_2pass_vbr_bias_pct

‘`maxrate, vb`’

rc_2pass_vbr_maxsection_pct (`maxrate * 100 / vb`)

‘`minrate, vb`’

rc_2pass_vbr_minsection_pct (`minrate * 100 / vb`)

‘`minrate, maxrate, vb`’

`VPX_CBR` (`minrate == maxrate == vb`)

‘`crf`’

`VPX_CQ`, `VP8E_SET_CQ_LEVEL`

‘`quality`’

‘*`best`*’

`VPX_DL_BEST_QUALITY`

‘*`good`*’

`VPX_DL_GOOD_QUALITY`

‘*`realtime`*’

VPX_DL_REALTIME

‘speed’

VP8E_SET_CPUUSED

‘nr’

VP8E_SET_NOISE_SENSITIVITY

‘mb_threshold’

VP8E_SET_STATIC_THRESHOLD

‘slices’

VP8E_SET_TOKEN_PARTITIONS

‘max-intra-rate’

VP8E_SET_MAX_INTRA_BITRATE_PCT

‘force_key_frames’

VPX_EFLAG_FORCE_KF

‘Alternate reference frame related’
‘vp8flags altref’

VP8E_SET_ENABLEAUTOALTREF

‘*arnr_max_frames*’

VP8E_SET_ARNR_MAXFRAMES

‘*arnr_type*’

VP8E_SET_ARNR_TYPE

‘*arnr_strength*’

VP8E_SET_ARNR_STRENGTH

‘*rc_lookahead*’

g_lag_in_frames

‘vp8flags error_resilient’

g_error_resilient

For more information about libvpx see: http://www.webmproject.org/

15.3 libx264

H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10 format supported through libx264.

Requires the presence of the libx264 headers and library during configuration. You need to explicitly configure the build with `--enable-libx264`.

15.3.1 Options

‘`preset` *`preset_name`*’

Set the encoding preset.

‘`tune` *`tune_name`*’

Tune the encoding params.

‘`fastfirstpass` *`bool`*’

Use fast settings when encoding first pass, default value is 1.

‘`profile` *`profile_name`*’

Set profile restrictions.

‘`level` *`level`*’

Specify level (as defined by Annex A). Deprecated in favor of *x264opts*.

‘`passlogfile` *`filename`*’

Specify filename for 2 pass stats. Deprecated in favor of *x264opts* (see *stats* libx264 option).

‘`wpredp` *`wpred_type`*’

Specify Weighted prediction for P-frames. Deprecated in favor of *x264opts* (see *weightp* libx264 option).

‘`x264opts` *`options`*’

Allow to set any x264 option, see x264 –fullhelp for a list.

options is a list of *key*=*value* couples separated by ":". In *filter* and *psy-rd* options that use ":" as a separator themselves, use "," instead. They accept it as well since long ago but this is kept undocumented for some reason.

For example to specify libx264 encoding options with `ffmpeg`:

```
ffmpeg -i foo.mpg -vcodec libx264 -x264opts keyint=123:min-keyint=20 -an out.mkv
```

For more information about libx264 and the supported options see:
http://www.videolan.org/developers/x264.html

16. Demuxers

Demuxers are configured elements in FFmpeg which allow to read the multimedia streams from a particular type of file.

When you configure your FFmpeg build, all the supported demuxers are enabled by default. You can list all available ones using the configure option "–list-demuxers".

You can disable all the demuxers using the configure option "–disable-demuxers", and selectively enable a single demuxer with the option "–enable-demuxer=*DEMUXER*", or disable it with the option "–disable-demuxer=*DEMUXER*".

The option "-formats" of the ff* tools will display the list of enabled demuxers.

The description of some of the currently available demuxers follows.

16.1 image2

Image file demuxer.

This demuxer reads from a list of image files specified by a pattern. The syntax and meaning of the pattern is specified by the option *pattern_type*.

The pattern may contain a suffix which is used to automatically determine the format of the images contained in the files.

The size, the pixel format, and the format of each image must be the same for all the files in the sequence.

This demuxer accepts the following options:

‘`framerate`’

Set the framerate for the video stream. It defaults to 25.

‘`loop`’

If set to 1, loop over the input. Default value is 0.

‘`pattern_type`’

Select the pattern type used to interpret the provided filename.

pattern_type accepts one of the following values.

‘`sequence`’

Select a sequence pattern type, used to specify a sequence of files indexed by sequential numbers.

A sequence pattern may contain the string "%d" or "%0*N*d", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0*N*d" is used, the string representing the number in each filename is 0-padded and *N* is the total number of 0-padded digits representing the number. The literal character ’%’ can be specified in the pattern with the string "%%".

If the sequence pattern contains "%d" or "%0*N*d", the first filename of the file list specified by the pattern must contain a number inclusively contained between *start_number* and *start_number*+*start_number_range*-1, and all the following numbers must be sequential.

For example the pattern "img-%03d.bmp" will match a sequence of filenames of the form ‘`img-001.bmp`’, ‘`img-002.bmp`’, ..., ‘`img-010.bmp`’, etc.; the pattern "i%%m%%g-%d.jpg" will match a sequence of filenames of the form ‘`i%m%g-1.jpg`’, ‘`i%m%g-2.jpg`’, ..., ‘`i%m%g-10.jpg`’, etc.

Note that the pattern must not necessarily contain "%d" or "%0*N*d", for example to convert a single image file ‘`img.jpeg`’ you can employ the command:

```
ffmpeg -i img.jpeg img.png
```

‘`glob`’

Select a glob wildcard pattern type.

The pattern is interpreted like a `glob()` pattern. This is only selectable if libavformat was compiled with globbing support.

‘`glob_sequence` *(deprecated, will be removed)*’

Select a mixed glob wildcard/sequence pattern.

If your version of libavformat was compiled with globbing support, and the provided pattern contains at least one glob meta character among `%*?[]{}` that is preceded by an unescaped "%", the pattern is interpreted like a `glob()` pattern, otherwise it is interpreted like a sequence pattern.

All glob special characters `%*?[]{}` must be prefixed with "%". To escape a literal "%" you shall use "%%".

For example the pattern `foo-%*.jpeg` will match all the filenames prefixed by "foo-" and terminating with ".jpeg", and `foo-%?%?%?.jpeg` will match all the filenames prefixed with "foo-", followed by a sequence of three characters, and terminating with ".jpeg".

This pattern type is deprecated in favor of *glob* and *sequence*.

Default value is *glob_sequence*.

‘`pixel_format`’

Set the pixel format of the images to read. If not specified the pixel format is guessed from the first image file in the sequence.

‘`start_number`’

Set the index of the file matched by the image file pattern to start to read from. Default value is 0.

‘`start_number_range`’

Set the index interval range to check when looking for the first image file in the sequence, starting from *start_number*. Default value is 5.

‘`video_size`’

Set the video size of the images to read. If not specified the video size is guessed from the first image file in the sequence.

16.1.1 Examples

- Use `ffmpeg` for creating a video from the images in the file sequence ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., assuming an input frame rate of 10 frames per second:

```
ffmpeg -i 'img-%03d.jpeg' -r 10 out.mkv
```

- As above, but start by reading from a file with index 100 in the sequence:

```
ffmpeg -start_number 100 -i 'img-%03d.jpeg' -r 10 out.mkv
```

- Read images matching the "*.png" glob pattern , that is all the files terminating with the ".png" suffix:

```
ffmpeg -pattern_type glob -i "*.png" -r 10 out.mkv
```

16.2 applehttp

Apple HTTP Live Streaming demuxer.

This demuxer presents all AVStreams from all variant streams. The id field is set to the bitrate variant index number. By setting the discard flags on AVStreams (by pressing ’a’ or ’v’ in ffplay), the caller can decide which variant streams to actually receive. The total bitrate of the variant that the stream belongs to is available in a metadata key named "variant_bitrate".

16.3 sbg

SBaGen script demuxer.

This demuxer reads the script language used by SBaGen http://uazu.net/sbagen/ to generate binaural beats sessions. A SBG script looks like that:

```
-SE
a: 300-2.5/3 440+4.5/0
b: 300-2.5/0 440+4.5/3
off: -
NOW      == a
+0:07:00 == b
+0:14:00 == a
+0:21:00 == b
+0:30:00    off
```

A SBG script can mix absolute and relative timestamps. If the script uses either only absolute timestamps (including the script start time) or only relative ones, then its layout is fixed, and the conversion is straightforward. On the other hand, if the script mixes both kind of timestamps, then the *NOW* reference for relative timestamps will be taken from the current time of day at the time the script is read, and the script layout will be frozen according to that reference. That means that if the script is directly played, the actual times will match the absolute timestamps up to the sound controller’s clock accuracy, but if the user somehow pauses the playback or seeks, all times will be shifted accordingly.

17. Muxers

Muxers are configured elements in FFmpeg which allow writing multimedia streams to a particular type of file.

When you configure your FFmpeg build, all the supported muxers are enabled by default. You can list all available muxers using the configure option `--list-muxers`.

You can disable all the muxers with the configure option `--disable-muxers` and selectively enable / disable single muxers with the options `--enable-muxer=`*`MUXER`* / `--disable-muxer=`*`MUXER`*.

The option `-formats` of the ff* tools will display the list of enabled muxers.

A description of some of the currently available muxers follows.

17.1 crc

CRC (Cyclic Redundancy Check) testing format.

This muxer computes and prints the Adler-32 CRC of all the input audio and video frames. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the CRC.

The output of the muxer consists of a single line of the form: CRC=0x*CRC*, where *CRC* is a hexadecimal number 0-padded to 8 digits containing the CRC for all the decoded input frames.

For example to compute the CRC of the input, and store it in the file ‘`out.crc`’:

```
ffmpeg -i INPUT -f crc out.crc
```

You can print the CRC to stdout with the command:

```
ffmpeg -i INPUT -f crc -
```

You can select the output format of each frame with `ffmpeg` by specifying the audio and video codec and format. For example to compute the CRC of the input audio converted to PCM unsigned 8-bit and the input video converted to MPEG-2 video, use the command:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f crc -
```

See also the framecrc muxer.

17.2 framecrc

Per-packet CRC (Cyclic Redundancy Check) testing format.

This muxer computes and prints the Adler-32 CRC for each audio and video packet. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the CRC.

The output of the muxer consists of a line for each audio and video packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, 0xCRC
```

CRC is a hexadecimal number 0-padded to 8 digits containing the CRC of the packet.

For example to compute the CRC of the audio and video frames in '`INPUT`', converted to raw audio and video packets, and store it in the file '`out.crc`':

```
ffmpeg -i INPUT -f framecrc out.crc
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framecrc -
```

With `ffmpeg`, you can select the output format to which the audio and video frames are encoded before computing the CRC for each packet by specifying the audio and video codec. For example, to compute the CRC of each decoded input audio frame converted to PCM unsigned 8-bit and of each decoded input video frame converted to MPEG-2 video, use the command:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f framecrc -
```

See also the crc muxer.

17.3 framemd5

Per-packet MD5 testing format.

This muxer computes and prints the MD5 hash for each audio and video packet. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the hash.

The output of the muxer consists of a line for each audio and video packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, MD5
```

MD5 is a hexadecimal number representing the computed MD5 hash for the packet.

For example to compute the MD5 of the audio and video frames in '`INPUT`', converted to raw audio and video packets, and store it in the file '`out.md5`':

```
ffmpeg -i INPUT -f framemd5 out.md5
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framemd5 -
```

See also the md5 muxer.

17.4 hls

Apple HTTP Live Streaming muxer that segments MPEG-TS according to the HTTP Live Streaming specification.

It creates a playlist file and numbered segment files. The output filename specifies the playlist filename; the segment filenames receive the same basename as the playlist, a sequential number and a .ts extension.

```
     ffmpeg -i in.nut out.m3u8
```

‘`-hls_time segment length in seconds`’
‘`-hls_list_size maximum number of playlist entries`’
‘`-hls_wrap number after which index wraps`’

17.5 ico

ICO file muxer.

Microsoft’s icon file format (ICO) has some strict limitations that should be noted:

- Size cannot exceed 256 pixels in any dimension
- Only BMP and PNG images can be stored
- If a BMP image is used, it must be one of the following pixel formats:

```
BMP Bit Depth      FFmpeg Pixel Format
1bit               pal8
4bit               pal8
8bit               pal8
16bit              rgb555le
24bit              bgr24
32bit              bgra
```

- If a BMP image is used, it must use the BITMAPINFOHEADER DIB header
- If a PNG image is used, it must use the rgba pixel format

17.6 image2

Image file muxer.

The image file muxer writes video frames to image files.

The output filenames are specified by a pattern, which can be used to produce sequentially numbered series of files. The pattern may contain the string "%d" or "%0*N*d", this string specifies the position of the characters representing a numbering in the filenames. If the form "%0*N*d" is used, the string representing the number in each filename is 0-padded to *N* digits. The literal character '%' can be specified in the pattern with the string "%%".

If the pattern contains "%d" or "%0*N*d", the first filename of the file list specified will contain the number 1, all the following numbers will be sequential.

The pattern may contain a suffix which is used to automatically determine the format of the image files to write.

For example the pattern "img-%03d.bmp" will specify a sequence of filenames of the form '`img-001.bmp`', '`img-002.bmp`', ..., '`img-010.bmp`', etc. The pattern "img%%-%d.jpg" will specify a sequence of filenames of the form '`img%-1.jpg`', '`img%-2.jpg`', ..., '`img%-10.jpg`', etc.

The following example shows how to use `ffmpeg` for creating a sequence of files '`img-001.jpeg`', '`img-002.jpeg`', ..., taking one image every second from the input video:

```
ffmpeg -i in.avi -vsync 1 -r 1 -f image2 'img-%03d.jpeg'
```

Note that with `ffmpeg`, if the format is not specified with the `-f` option and the output filename specifies an image file format, the image2 muxer is automatically selected, so the previous command can be written as:

```
ffmpeg -i in.avi -vsync 1 -r 1 'img-%03d.jpeg'
```

Note also that the pattern must not necessarily contain "%d" or "%0*N*d", for example to create a single image file '`img.jpeg`' from the input video you can employ the command:

```
ffmpeg -i in.avi -f image2 -frames:v 1 img.jpeg
```

The image muxer supports the .Y.U.V image file format. This format is special in that that each image frame consists of three files, for each of the YUV420P components. To read or write this image file format, specify the name of the '.Y' file. The muxer will automatically open the '.U' and '.V' files as required.

17.7 md5

MD5 testing format.

This muxer computes and prints the MD5 hash of all the input audio and video frames. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the hash.

The output of the muxer consists of a single line of the form: MD5=*MD5*, where *MD5* is a hexadecimal number representing the computed MD5 hash.

For example to compute the MD5 hash of the input converted to raw audio and video, and store it in the file ‘`out.md5`’:

```
ffmpeg -i INPUT -f md5 out.md5
```

You can print the MD5 to stdout with the command:

```
ffmpeg -i INPUT -f md5 -
```

See also the framemd5 muxer.

17.8 MOV/MP4/ISMV

The mov/mp4/ismv muxer supports fragmentation. Normally, a MOV/MP4 file has all the metadata about all packets stored in one location (written at the end of the file, it can be moved to the start for better playback by adding *faststart* to the *movflags*, or using the `qt-faststart` tool). A fragmented file consists of a number of fragments, where packets and metadata about these packets are stored together. Writing a fragmented file has the advantage that the file is decodable even if the writing is interrupted (while a normal MOV/MP4 is undecodable if it is not properly finished), and it requires less memory when writing very long files (since writing normal MOV/MP4 files stores info about every single packet in memory until the file is closed). The downside is that it is less compatible with other applications.

Fragmentation is enabled by setting one of the AVOptions that define how to cut the file into fragments:

‘`-moov_size` *`bytes`*’

Reserves space for the moov atom at the beginning of the file instead of placing the moov atom at the end. If the space reserved is insufficient, muxing will fail.

‘`-movflags frag_keyframe`’

Start a new fragment at each video keyframe.

‘`-frag_duration` *`duration`*’

Create fragments that are *duration* microseconds long.

‘`-frag_size` *`size`*’

Create fragments that contain up to *size* bytes of payload data.

‘`-movflags frag_custom`’

Allow the caller to manually choose when to cut fragments, by calling `av_write_frame(ctx, NULL)` to write a fragment with the packets written so far. (This is only useful with other applications integrating libavformat, not from `ffmpeg`.)

‘`-min_frag_duration` *`duration`*’

> Don’t create fragments that are shorter than *duration* microseconds long.

If more than one condition is specified, fragments are cut when one of the specified conditions is fulfilled. The exception to this is `-min_frag_duration`, which has to be fulfilled for any of the other conditions to apply.

Additionally, the way the output file is written can be adjusted through a few other options:

‘`-movflags empty_moov`’

> Write an initial moov atom directly at the start of the file, without describing any samples in it. Generally, an mdat/moov pair is written at the start of the file, as a normal MOV/MP4 file, containing only a short portion of the file. With this option set, there is no initial mdat atom, and the moov atom only describes the tracks but has a zero duration.
>
> Files written with this option set do not work in QuickTime. This option is implicitly set when writing ismv (Smooth Streaming) files.

‘`-movflags separate_moof`’

> Write a separate moof (movie fragment) atom for each track. Normally, packets for all tracks are written in a moof atom (which is slightly more efficient), but with this option set, the muxer writes one moof/mdat pair for each track, making it easier to separate tracks.
>
> This option is implicitly set when writing ismv (Smooth Streaming) files.

‘`-movflags faststart`’

> Run a second pass moving the moov atom on top of the file. This operation can take a while, and will not work in various situations such as fragmented output, thus it is not enabled by default.

Smooth Streaming content can be pushed in real time to a publishing point on IIS with this muxer. Example:

```
ffmpeg -re <normal input/transcoding options> -movflags isml+frag_keyframe -f ismv http://server/publishingpoint.isml/Streams(Encoder1)
```

17.9 mpegts

MPEG transport stream muxer.

This muxer implements ISO 13818-1 and part of ETSI EN 300 468.

The muxer options are:

‘`-mpegts_original_network_id` *`number`*’

Set the original_network_id (default 0x0001). This is unique identifier of a network in DVB. Its main use is in the unique identification of a service through the path Original_Network_ID, Transport_Stream_ID.

‘`-mpegts_transport_stream_id` *`number`*’

Set the transport_stream_id (default 0x0001). This identifies a transponder in DVB.

‘`-mpegts_service_id` *`number`*’

Set the service_id (default 0x0001) also known as program in DVB.

‘`-mpegts_pmt_start_pid` *`number`*’

Set the first PID for PMT (default 0x1000, max 0x1f00).

‘`-mpegts_start_pid` *`number`*’

Set the first PID for data packets (default 0x0100, max 0x0f00).

The recognized metadata settings in mpegts muxer are `service_provider` and `service_name`. If they are not set the default for `service_provider` is "FFmpeg" and the default for `service_name` is "Service01".

```
ffmpeg -i file.mpg -c copy \
     -mpegts_original_network_id 0x1122 \
     -mpegts_transport_stream_id 0x3344 \
     -mpegts_service_id 0x5566 \
     -mpegts_pmt_start_pid 0x1500 \
     -mpegts_start_pid 0x150 \
     -metadata service_provider="Some provider" \
     -metadata service_name="Some Channel" \
     -y out.ts
```

17.10 null

Null muxer.

This muxer does not generate any output file, it is mainly useful for testing or benchmarking purposes.

For example to benchmark decoding with `ffmpeg` you can use the command:

```
ffmpeg -benchmark -i INPUT -f null out.null
```

Note that the above command does not read or write the '`out.null`' file, but specifying the output file is required by the `ffmpeg` syntax.

Alternatively you can write the command as:

```
ffmpeg -benchmark -i INPUT -f null -
```

17.11 matroska

Matroska container muxer.

This muxer implements the matroska and webm container specs.

The recognized metadata settings in this muxer are:

'`title=`*`title name`*'

Name provided to a single track

'`language=`*`language name`*'

Specifies the language of the track in the Matroska languages form

'`stereo_mode=`*`mode`*'

Stereo 3D video layout of two views in a single video track

'`mono`'

video is not stereo

'`left_right`'

Both views are arranged side by side, Left-eye view is on the left

'`bottom_top`'

Both views are arranged in top-bottom orientation, Left-eye view is at bottom

'`top_bottom`'

Both views are arranged in top-bottom orientation, Left-eye view is on top

'`checkerboard_rl`'

Each view is arranged in a checkerboard interleaved pattern, Left-eye view being first ‘`checkerboard_lr`’

Each view is arranged in a checkerboard interleaved pattern, Right-eye view being first

‘`row_interleaved_rl`’

Each view is constituted by a row based interleaving, Right-eye view is first row

‘`row_interleaved_lr`’

Each view is constituted by a row based interleaving, Left-eye view is first row

‘`col_interleaved_rl`’

Both views are arranged in a column based interleaving manner, Right-eye view is first column

‘`col_interleaved_lr`’

Both views are arranged in a column based interleaving manner, Left-eye view is first column

‘`anaglyph_cyan_red`’

All frames are in anaglyph format viewable through red-cyan filters

‘`right_left`’

Both views are arranged side by side, Right-eye view is on the left

‘`anaglyph_green_magenta`’

All frames are in anaglyph format viewable through green-magenta filters

‘`block_lr`’

Both eyes laced in one Block, Left-eye view is first

‘`block_rl`’

Both eyes laced in one Block, Right-eye view is first

For example a 3D WebM clip can be created using the following command line:

```
ffmpeg -i sample_left_right_clip.mpg -an -c:v libvpx -metadata stereo_mode=left_right -y stereo_clip.webm
```

17.12 segment, stream_segment, ssegment

Basic stream segmenter.

The segmenter muxer outputs streams to a number of separate files of nearly fixed duration. Output filename pattern can be set in a fashion similar to image2.

`stream_segment` is a variant of the muxer used to write to streaming output formats, i.e. which do not require global headers, and is recommended for outputting e.g. to MPEG transport stream segments. `ssegment` is a shorter alias for `stream_segment`.

Every segment starts with a video keyframe, if a video stream is present. Note that if you want accurate splitting for a video file, you need to make the input key frames correspond to the exact splitting times expected by the segmenter, or the segment muxer will start the new segment with the key frame found next after the specified start time.

The segment muxer works best with a single constant frame rate video.

Optionally it can generate a list of the created segments, by setting the option *segment_list*. The list type is specified by the *segment_list_type* option.

The segment muxer supports the following options:

‘`segment_format` *`format`*’

Override the inner container format, by default it is guessed by the filename extension.

‘`segment_list` *`name`*’

Generate also a listfile named *name*. If not specified no listfile is generated.

‘`segment_list_flags` *`flags`*’

Set flags affecting the segment list generation.

It currently supports the following flags:

cache

Allow caching (only affects M3U8 list files).

live

Allow live-friendly file generation.

This currently only affects M3U8 lists. In particular, write a fake EXT-X-TARGETDURATION duration field at the top of the file, based on the specified *segment_time*.

Default value is `cache`.

‘`segment_list_size` *`size`*’

Overwrite the listfile once it reaches *size* entries. If 0 the listfile is never overwritten. Default value is 0.

‘`segment_list type` *`type`*’

Specify the format for the segment list file.

The following values are recognized:

‘`flat`’

Generate a flat list for the created segments, one segment per line.

‘`csv, ext`’

Generate a list for the created segments, one segment per line, each line matching the format (comma-separated values):

```
segment_filename,segment_start_time,segment_end_time
```

segment_filename is the name of the output file generated by the muxer according to the provided pattern. CSV escaping (according to RFC4180) is applied if required.

segment_start_time and *segment_end_time* specify the segment start and end time expressed in seconds.

A list file with the suffix `".csv"` or `".ext"` will auto-select this format.

`ext` is deprecated in favor or `csv`.

‘`m3u8`’

Generate an extended M3U8 file, version 4, compliant with http://tools.ietf.org/id/draft-pantos-http-live-streaming-08.txt.

A list file with the suffix `".m3u8"` will auto-select this format.

If not specified the type is guessed from the list file name suffix.

‘`segment_time` *`time`*’

Set segment duration to *time*. Default value is "2".

‘`segment_time_delta` *`delta`*’

Specify the accuracy time when selecting the start time for a segment. Default value is "0".

When delta is specified a key-frame will start a new segment if its PTS satisfies the relation:

```
PTS >= start_time - time_delta
```

This option is useful when splitting video content, which is always split at GOP boundaries, in case a key frame is found just before the specified split time.

In particular may be used in combination with the ‘`ffmpeg`’ option *force_key_frames*. The key frame times specified by *force_key_frames* may not be set accurately because of rounding issues, with the consequence that a key frame time may result set just before the specified time. For constant frame rate videos a value of 1/2**frame_rate* should address the worst case mismatch between the specified time and the time set by *force_key_frames*.

‘`segment_times` *`times`*’

Specify a list of split points. *times* contains a list of comma separated duration specifications, in increasing order.

‘`segment_wrap` *`limit`*’

Wrap around segment index once it reaches *limit*.

Some examples follow.

- To remux the content of file ‘`in.mkv`’ to a list of segments ‘`out-000.nut`’, ‘`out-001.nut`’, etc., and write the list of generated segments to ‘`out.list`’:

  ```
  ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.list out%03d.nut
  ```

- As the example above, but segment the input file according to the split points specified by the *segment_times* option:

  ```
  ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 out%03d.nut
  ```

- As the example above, but use the `ffmpeg` *force_key_frames* option to force key frames in the input at the specified location, together with the segment option *segment_time_delta* to account for possible roundings operated when setting key frame times.

  ```
  ffmpeg -i in.mkv -force_key_frames 1,2,3,5,8,13,21 -vcodec mpeg4 -acodec pcm_s16le -map 0 \
  -f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 -segment_time_delta 0.05 out%03d.nut
  ```

  In order to force key frames on the input file, transcoding is required.

- To convert the ‘`in.mkv`’ to TS segments using the `libx264` and `libfaac` encoders:

  ```
  ffmpeg -i in.mkv -map 0 -codec:v libx264 -codec:a libfaac -f ssegment -segment_list out.list out%03d.ts
  ```

- Segment the input file, and create an M3U8 live playlist (can be used as live HLS source):

```
ffmpeg -re -i in.mkv -codec copy -map 0 -f segment -segment_list playlist.m3u8 \
-segment_list_flags +live -segment_time 10 out%03d.mkv
```

17.13 mp3

The MP3 muxer writes a raw MP3 stream with an ID3v2 header at the beginning and optionally an ID3v1 tag at the end. ID3v2.3 and ID3v2.4 are supported, the `id3v2_version` option controls which one is used. The legacy ID3v1 tag is not written by default, but may be enabled with the `write_id3v1` option.

For seekable output the muxer also writes a Xing frame at the beginning, which contains the number of frames in the file. It is useful for computing duration of VBR files.

The muxer supports writing ID3v2 attached pictures (APIC frames). The pictures are supplied to the muxer in form of a video stream with a single packet. There can be any number of those streams, each will correspond to a single APIC frame. The stream metadata tags *title* and *comment* map to APIC *description* and *picture type* respectively. See http://id3.org/id3v2.4.0-frames for allowed picture types.

Note that the APIC frames must be written at the beginning, so the muxer will buffer the audio frames until it gets all the pictures. It is therefore advised to provide the pictures as soon as possible to avoid excessive buffering.

Examples:

Write an mp3 with an ID3v2.3 header and an ID3v1 footer:

```
ffmpeg -i INPUT -id3v2_version 3 -write_id3v1 1 out.mp3
```

Attach a picture to an mp3:

```
ffmpeg -i input.mp3 -i cover.png -c copy -metadata:s:v title="Album cover"
-metadata:s:v comment="Cover (Front)" out.mp3
```

18. Input Devices

Input devices are configured elements in FFmpeg which allow to access the data coming from a multimedia device attached to your system.

When you configure your FFmpeg build, all the supported input devices are enabled by default. You can list all available ones using the configure option "–list-indevs".

You can disable all the input devices using the configure option "–disable-indevs", and selectively enable an input device using the option "–enable-indev=*INDEV*", or you can disable a particular input device using the option "–disable-indev=*INDEV*".

The option "-formats" of the ff* tools will display the list of supported input devices (amongst the demuxers).

A description of the currently available input devices follows.

18.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

To enable this input device during configuration you need libasound installed on your system.

This device allows capturing from an ALSA device. The name of the device to capture has to be an ALSA card identifier.

An ALSA identifier has the syntax:

```
hw:CARD[,DEV[,SUBDEV]]
```

where the *DEV* and *SUBDEV* components are optional.

The three arguments (in order: *CARD*,*DEV*,*SUBDEV*) specify card number or identifier, device number and subdevice number (-1 means any).

To see the list of cards currently recognized by your system check the files ‘`/proc/asound/cards`’ and ‘`/proc/asound/devices`’.

For example to capture with `ffmpeg` from an ALSA device with card id 0, you may run the command:

```
ffmpeg -f alsa -i hw:0 alsaout.wav
```

For more information see: http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html

18.2 bktr

BSD video input device.

18.3 dshow

Windows DirectShow input device.

DirectShow support is enabled when FFmpeg is built with the mingw-w64 project. Currently only audio and video devices are supported.

Multiple devices may be opened as separate inputs, but they may also be opened on the same input, which should improve synchronism between them.

The input name should be in the format:

```
TYPE=NAME[:TYPE=NAME]
```

where *TYPE* can be either *audio* or *video*, and *NAME* is the device's name.

18.3.1 Options

If no options are specified, the device's defaults are used. If the device does not support the requested options, it will fail to open.

'`video_size`'

Set the video size in the captured video.

'`framerate`'

Set the framerate in the captured video.

'`sample_rate`'

Set the sample rate (in Hz) of the captured audio.

'`sample_size`'

Set the sample size (in bits) of the captured audio.

'`channels`'

Set the number of channels in the captured audio.

'`list_devices`'

If set to '`true`', print a list of devices and exit.

'`list_options`'

If set to '`true`', print a list of selected device's options and exit.

'`video_device_number`'

Set video device number for devices with same name (starts at 0, defaults to 0).

'`audio_device_number`'

Set audio device number for devices with same name (starts at 0, defaults to 0).

'`pixel_format`'

Select pixel format to be used by DirectShow. This may only be set when the video codec is not set or set to rawvideo.

'`audio_buffer_size`'

Set audio device buffer size in milliseconds (which can directly impact latency, depending on the device). Defaults to using the audio device's default buffer size (typically some multiple of 500ms). Setting this value too low can degrade performance. See also http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582(v=vs.85).aspx

18.3.2 Examples

- Print the list of DirectShow supported devices and exit:

```
$ ffmpeg -list_devices true -f dshow -i dummy
```

- Open video device *Camera*:

```
$ ffmpeg -f dshow -i video="Camera"
```

- Open second video device with name *Camera*:

```
$ ffmpeg -f dshow -video_device_number 1 -i video="Camera"
```

- Open video device *Camera* and audio device *Microphone*:

```
$ ffmpeg -f dshow -i video="Camera":audio="Microphone"
```

- Print the list of supported options in selected device and exit:

```
$ ffmpeg -list_options true -f dshow -i video="Camera"
```

18.4 dv1394

Linux DV 1394 input device.

18.5 fbdev

Linux framebuffer input device.

The Linux framebuffer is a graphic hardware-independent abstraction layer to show graphics on a computer monitor, typically on the console. It is accessed through a file device node, usually '`/dev/fb0`'.

For more detailed information read the file Documentation/fb/framebuffer.txt included in the Linux source tree.

To record from the framebuffer device '`/dev/fb0`' with `ffmpeg`:

```
ffmpeg -f fbdev -r 10 -i /dev/fb0 out.avi
```

You can take a single screenshot image with the command:

```
ffmpeg -f fbdev -frames:v 1 -r 1 -i /dev/fb0 screenshot.jpeg
```

See also http://linux-fbdev.sourceforge.net/, and fbset(1).

18.6 iec61883

FireWire DV/HDV input device using libiec61883.

To enable this input device, you need libiec61883, libraw1394 and libavc1394 installed on your system. Use the configure option `--enable-libiec61883` to compile with the device enabled.

The iec61883 capture device supports capturing from a video device connected via IEEE1394 (FireWire), using libiec61883 and the new Linux FireWire stack (juju). This is the default DV/HDV input method in Linux Kernel 2.6.37 and later, since the old FireWire stack was removed.

Specify the FireWire port to be used as input file, or "auto" to choose the first port connected.

18.6.1 Options

'`dvtype`'

Override autodetection of DV/HDV. This should only be used if auto detection does not work, or if usage of a different device type should be prohibited. Treating a DV device as HDV (or vice versa) will not work and result in undefined behavior. The values '`auto`', '`dv`' and '`hdv`' are supported.

'`dvbuffer`'

Set maxiumum size of buffer for incoming data, in frames. For DV, this is an exact value. For HDV, it is not frame exact, since HDV does not have a fixed frame size.

'`dvguid`'

Select the capture device by specifying it's GUID. Capturing will only be performed from the specified device and fails if no device with the given GUID is found. This is useful to select the input if multiple devices are connected at the same time. Look at /sys/bus/firewire/devices to find out the GUIDs.

18.6.2 Examples

- Grab and show the input of a FireWire DV/HDV device.

```
ffplay -f iec61883 -i auto
```

- Grab and record the input of a FireWire DV/HDV device, using a packet buffer of 100000 packets if the source is HDV.

```
ffmpeg -f iec61883 -i auto -hdvbuffer 100000 out.mpg
```

18.7 jack

JACK input device.

To enable this input device during configuration you need libjack installed on your system.

A JACK input device creates one or more JACK writable clients, one for each audio channel, with name *client_name*:input_*N*, where *client_name* is the name provided by the application, and *N* is a number which identifies the channel. Each writable client will send the acquired data to the FFmpeg input device.

Once you have created one or more JACK readable clients, you need to connect them to one or more JACK writable clients.

To connect or disconnect JACK clients you can use the `jack_connect` and `jack_disconnect` programs, or do it through a graphical interface, for example with `qjackctl`.

To list the JACK clients and their properties you can invoke the command `jack_lsp`.

Follows an example which shows how to capture a JACK readable client with `ffmpeg`.

```
# Create a JACK writable client with name "ffmpeg".
$ ffmpeg -f jack -i ffmpeg -y out.wav

# Start the sample jack_metro readable client.
$ jack_metro -b 120 -d 0.2 -f 4000

# List the current JACK clients.
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

# Connect metro to the ffmpeg writable client.
$ jack_connect metro:120_bpm ffmpeg:input_1
```

For more information read: http://jackaudio.org/

18.8 lavfi

Libavfilter input virtual device.

This input device reads data from the open output pads of a libavfilter filtergraph.

For each filtergraph open output, the input device will create a corresponding stream which is mapped to the generated output. Currently only video data is supported. The filtergraph is specified through the option ‘`graph`’.

18.8.1 Options

‘`graph`’

Specify the filtergraph to use as input. Each video open output must be labelled by a unique string of the form "out*N*", where *N* is a number starting from 0 corresponding to the mapped input stream generated by the device. The first unlabelled output is automatically assigned to the "out0" label, but all the others need to be specified explicitly.

If not specified defaults to the filename specified for the input device.

‘`graph_file`’

Set the filename of the filtergraph to be read and sent to the other filters. Syntax of the filtergraph is the same as the one specified by the option *graph*.

18.8.2 Examples

- Create a color video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi -graph "color=pink [out0]" dummy
  ```

- As the previous example, but use filename for specifying the graph description, and omit the "out0" label:

  ```
  ffplay -f lavfi color=pink
  ```

- Create three different video test filtered sources and play them:

  ```
  ffplay -f lavfi -graph "testsrc [out0]; testsrc,hflip [out1]; testsrc,negate [out2]" test3
  ```

- Read an audio stream from a file using the amovie source and play it back with `ffplay`:

  ```
  ffplay -f lavfi "amovie=test.wav"
  ```

- Read an audio stream and a video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi "movie=test.avi[out0];amovie=test.wav[out1]"
  ```

18.9 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

18.10 openal

The OpenAL input device provides audio capture on all systems with a working OpenAL 1.1 implementation.

To enable this input device during configuration, you need OpenAL headers and libraries installed on your system, and need to configure FFmpeg with `--enable-openal`.

OpenAL headers and libraries should be provided as part of your OpenAL implementation, or as an additional download (an SDK). Depending on your installation you may need to specify additional flags via the `--extra-cflags` and `--extra-ldflags` for allowing the build system to locate the OpenAL headers and libraries.

An incomplete list of OpenAL implementations follows:

Creative

The official Windows implementation, providing hardware acceleration with supported devices and software fallback. See http://openal.org/.

OpenAL Soft

Portable, open source (LGPL) software implementation. Includes backends for the most common sound APIs on the Windows, Linux, Solaris, and BSD operating systems. See http://kcat.strangesoft.net/openal.html.

Apple

OpenAL is part of Core Audio, the official Mac OS X Audio interface. See http://developer.apple.com/technologies/mac/audio-and-video.html

This device allows to capture from an audio input device handled through OpenAL.

You need to specify the name of the device to capture in the provided filename. If the empty string is provided, the device will automatically select the default device. You can get the list of the supported devices by using the option *list_devices*.

18.10.1 Options

'`channels`'

Set the number of channels in the captured audio. Only the values '`1`' (monaural) and '`2`' (stereo) are currently supported. Defaults to '`2`'.

'`sample_size`'

Set the sample size (in bits) of the captured audio. Only the values '`8`' and '`16`' are currently supported. Defaults to '`16`'.

'`sample_rate`'

Set the sample rate (in Hz) of the captured audio. Defaults to '`44.1k`'.

'`list_devices`'

If set to '`true`', print a list of devices and exit. Defaults to '`false`'.

18.10.2 Examples

Print the list of OpenAL supported devices and exit:

```
$ ffmpeg -list_devices true -f openal -i dummy out.ogg
```

Capture from the OpenAL device ‘`DR-BT101 via PulseAudio`’:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out.ogg
```

Capture from the default device (note the empty string ” as filename):

```
$ ffmpeg -f openal -i ’’ out.ogg
```

Capture from two devices simultaneously, writing to two different files, within the same `ffmpeg` command:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out1.ogg -f openal -i ’ALSA Default’ out2.ogg
```

Note: not all OpenAL implementations support multiple simultaneous capture - try the latest OpenAL Soft if the above does not work.

18.11 oss

Open Sound System input device.

The filename to provide to the input device is the device node representing the OSS input device, and is usually set to ‘`/dev/dsp`’.

For example to grab from ‘`/dev/dsp`’ using `ffmpeg` use the command:

```
ffmpeg -f oss -i /dev/dsp /tmp/oss.wav
```

For more information about OSS see: http://manuals.opensound.com/usersguide/dsp.html

18.12 pulse

pulseaudio input device.

To enable this input device during configuration you need libpulse-simple installed in your system.

The filename to provide to the input device is a source device or the string "default"

To list the pulse source devices and their properties you can invoke the command `pactl list sources`.

```
ffmpeg -f pulse -i default /tmp/pulse.wav
```

18.12.1 *server* AVOption

The syntax is:

```
    -server server name
```

Connects to a specific server.

18.12.2 *name* AVOption

The syntax is:

```
    -name application name
```

Specify the application name pulse will use when showing active clients, by default it is the LIBAVFORMAT_IDENT string

18.12.3 *stream_name* AVOption

The syntax is:

```
    -stream_name stream name
```

Specify the stream name pulse will use when showing active streams, by default it is "record"

18.12.4 *sample_rate* AVOption

The syntax is:

```
    -sample_rate samplerate
```

Specify the samplerate in Hz, by default 48kHz is used.

18.12.5 *channels* AVOption

The syntax is:

```
    -channels N
```

Specify the channels in use, by default 2 (stereo) is set.

18.12.6 *frame_size* AVOption

The syntax is:

```
-frame_size bytes
```

Specify the number of byte per frame, by default it is set to 1024.

18.12.7 *fragment_size* AVOption

The syntax is:

```
-fragment_size bytes
```

Specify the minimal buffering fragment in pulseaudio, it will affect the audio latency. By default it is unset.

18.13 sndio

sndio input device.

To enable this input device during configuration you need libsndio installed on your system.

The filename to provide to the input device is the device node representing the sndio input device, and is usually set to '`/dev/audio0`'.

For example to grab from '`/dev/audio0`' using `ffmpeg` use the command:

```
ffmpeg -f sndio -i /dev/audio0 /tmp/oss.wav
```

18.14 video4linux2

Video4Linux2 input video device.

The name of the device to grab is a file device node, usually Linux systems tend to automatically create such nodes when the device (e.g. an USB webcam) is plugged into the system, and has a name of the kind '`/dev/video`*N*', where *N* is a number associated to the device.

Video4Linux2 devices usually support a limited set of *width*x*height* sizes and framerates. You can check which are supported using `-list_formats all` for Video4Linux2 devices.

Some usage examples of the video4linux2 devices with ffmpeg and ffplay:

The time base for the timestamps is 1 microsecond. Depending on the kernel version and configuration, the timestamps may be derived from the real time clock (origin at the Unix Epoch) or the monotonic clock (origin usually at boot time, unaffected by NTP or manual changes to the clock). The '`-timestamps abs`' or '`-ts abs`' option can be used to force conversion into the real time clock.

Note that if FFmpeg is build with v4l-utils support ("–enable-libv4l2" option), it will always be used.

```
# Grab and show the input of a video4linux2 device.
ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0

# Grab and record the input of a video4linux2 device, leave the
framerate and size as previously set.
ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
```

"v4l" and "v4l2" can be used as aliases for the respective "video4linux" and "video4linux2".

18.15 vfwcap

VfW (Video for Windows) capture input device.

The filename passed as input is the capture driver number, ranging from 0 to 9. You may use "list" as filename to print a list of drivers. Any other filename will be interpreted as device number 0.

18.16 x11grab

X11 video input device.

This device allows to capture a region of an X11 display.

The filename passed as input has the syntax:

```
[hostname]:display_number.screen_number[+x_offset,y_offset]
```

hostname:*display_number*.*screen_number* specifies the X11 display name of the screen to grab from. *hostname* can be omitted, and defaults to "localhost". The environment variable `DISPLAY` contains the default display name.

x_offset and *y_offset* specify the offsets of the grabbed area with respect to the top-left border of the X11 screen. They default to 0.

Check the X11 documentation (e.g. man X) for more detailed information.

Use the `dpyinfo` program for getting basic information about the properties of your X11 display (e.g. grep for "name" or "dimensions").

For example to grab from ‘`:0.0`’ using `ffmpeg`:

```
ffmpeg -f x11grab -r 25 -s cif -i :0.0 out.mpg
```

Grab at position `10,20`:

```
ffmpeg -f x11grab -r 25 -s cif -i :0.0+10,20 out.mpg
```

18.16.1 Options

‘`draw_mouse`’

Specify whether to draw the mouse pointer. A value of `0` specify not to draw the pointer. Default value is `1`.

‘`follow_mouse`’

Make the grabbed area follow the mouse. The argument can be `centered` or a number of pixels *PIXELS*.

When it is specified with "centered", the grabbing region follows the mouse pointer and keeps the pointer at the center of region; otherwise, the region follows only when the mouse pointer reaches within *PIXELS* (greater than zero) to the edge of region.

For example:

```
ffmpeg -f x11grab -follow_mouse centered -r 25 -s cif -i :0.0 out.mpg
```

To follow only when the mouse pointer reaches within 100 pixels to edge:

```
ffmpeg -f x11grab -follow_mouse 100 -r 25 -s cif -i :0.0 out.mpg
```

‘`framerate`’

Set the grabbing frame rate. Default value is `ntsc`, corresponding to a framerate of `30000/1001`.

‘`show_region`’

Show grabbed region on screen.

If *show_region* is specified with `1`, then the grabbing region will be indicated on screen. With this option, it is easy to know what is being grabbed if only a portion of the screen is grabbed.

For example:

```
ffmpeg -f x11grab -show_region 1 -r 25 -s cif -i :0.0+10,20 out.mpg
```

With *follow_mouse*:

```
ffmpeg -f x11grab -follow_mouse centered -show_region 1 -r 25 -s cif -i :0.0 out.mpg
```

‘`video_size`’

Set the video frame size. Default value is `vga`.

19. Output Devices

Output devices are configured elements in FFmpeg which allow to write multimedia data to an output device attached to your system.

When you configure your FFmpeg build, all the supported output devices are enabled by default. You can list all available ones using the configure option "–list-outdevs".

You can disable all the output devices using the configure option "–disable-outdevs", and selectively enable an output device using the option "–enable-outdev=*OUTDEV*", or you can disable a particular input device using the option "–disable-outdev=*OUTDEV*".

The option "-formats" of the ff* tools will display the list of enabled output devices (amongst the muxers).

A description of the currently available output devices follows.

19.1 alsa

ALSA (Advanced Linux Sound Architecture) output device.

19.2 caca

CACA output device.

This output devices allows to show a video stream in CACA window. Only one CACA window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need to configure FFmpeg with `--enable-libcaca`. libcaca is a graphics library that outputs text instead of pixels.

For more information about libcaca, check: http://caca.zoy.org/wiki/libcaca

19.2.1 Options

‘`window_title`’

Set the CACA window title, if not specified default to the filename specified for the output device.

‘`window_size`’

Set the CACA window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video.

‘`driver`’

Set display driver.

‘`algorithm`’

Set dithering algorithm. Dithering is necessary because the picture being rendered has usually far more colours than the available palette. The accepted values are listed with `-list_dither algorithms`.

‘`antialias`’

Set antialias method. Antialiasing smoothens the rendered image and avoids the commonly seen staircase effect. The accepted values are listed with `-list_dither antialiases`.

‘`charset`’

Set which characters are going to be used when rendering text. The accepted values are listed with `-list_dither charsets`.

‘`color`’

Set color to be used when rendering text. The accepted values are listed with `-list_dither colors`.

‘`list_drivers`’

If set to ‘`true`’, print a list of available drivers and exit.

‘`list_dither`’

List available dither options related to the argument. The argument must be one of `algorithms`, `antialiases`, `charsets`, `colors`.

19.2.2 Examples

- The following command shows the `ffmpeg` output is an CACA window, forcing its size to 80x25:

  ```
  ffmpeg -i INPUT -vcodec rawvideo -pix_fmt rgb24 -window_size 80x25 -f caca -
  ```

- Show the list of available drivers and exit:

  ```
  ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_drivers true -
  ```

- Show the list of available dither colors and exit:

  ```
  ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_dither colors -
  ```

19.3 oss

OSS (Open Sound System) output device.

19.4 sdl

SDL (Simple DirectMedia Layer) output device.

This output devices allows to show a video stream in an SDL window. Only one SDL window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need libsdl installed on your system when configuring your build.

For more information about SDL, check: http://www.libsdl.org/

19.4.1 Options

‘`window_title`’

  Set the SDL window title, if not specified default to the filename specified for the output device.

‘`icon_title`’

  Set the name of the iconified SDL window, if not specified it is set to the same value of *window_title*.

‘`window_size`’

  Set the SDL window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video, downscaled according to the aspect ratio.

19.4.2 Examples

The following command shows the `ffmpeg` output is an SDL window, forcing its size to the qcif format:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt yuv420p -window_size qcif -f sdl "SDL output"
```

19.5 sndio

sndio audio output device.

20. Protocols

Protocols are configured elements in FFmpeg which allow to access resources which require the use of a particular protocol.

When you configure your FFmpeg build, all the supported protocols are enabled by default. You can list all available ones using the configure option "–list-protocols".

You can disable all the protocols using the configure option "–disable-protocols", and selectively enable a protocol using the option "–enable-protocol=*PROTOCOL*", or you can disable a particular protocol using the option "–disable-protocol=*PROTOCOL*".

The option "-protocols" of the ff* tools will display the list of supported protocols.

A description of the currently available protocols follows.

20.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

  BluRay angle

‘`chapter`’

  Start chapter (1...N)

‘`playlist`’

  Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

20.2 concat

Physical concatenation protocol.

Allow to read and seek from many resource in sequence as if they were a unique resource.

A URL accepted by this protocol has the syntax:

```
concat:URL1|URL2|...|URLN
```

where *URL1*, *URL2*, ..., *URLN* are the urls of the resource to be concatenated, each one possibly specifying a distinct protocol.

For example to read a sequence of files ‘`split1.mpeg`’, ‘`split2.mpeg`’, ‘`split3.mpeg`’ with `ffplay` use the command:

```
ffplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

Note that you may need to escape the character "|" which is special for many shells.

20.3 file

File access protocol.

Allow to read from or read to a file.

For example to read from a file ‘`input.mpeg`’ with `ffmpeg` use the command:

```
ffmpeg -i file:input.mpeg output.mpeg
```

The ff* tools default to the file protocol, that is a resource specified with the name "FILE.mpeg" is interpreted as the URL "file:FILE.mpeg".

20.4 gopher

Gopher protocol.

20.5 hls

Read Apple HTTP Live Streaming compliant segmented stream as a uniform one. The M3U8 playlists describing the segments can be remote HTTP resources or local files, accessed using the standard file protocol. The nested protocol is declared by specifying "+*proto*" after the hls URI scheme name, where *proto* is either "file" or "http".

```
hls+http://host/path/to/remote/resource.m3u8
hls+file://path/to/local/resource.m3u8
```

Using this protocol is discouraged - the hls demuxer should work just as well (if not, please report the issues) and is more complete. To use the hls demuxer instead, simply use the direct URLs to the m3u8 files.

20.6 http

HTTP (Hyper Text Transfer Protocol).

20.7 mmst

MMS (Microsoft Media Server) protocol over TCP.

20.8 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

```
mmsh://server[:port][/app][/playpath]
```

20.9 md5

MD5 output protocol.

Computes the MD5 hash of the data to be written, and on close writes this to the designated output or stdout if none is specified. It can be used to test muxers without writing an actual file.

Some examples follow.

```
# Write the MD5 hash of the encoded AVI file to the file output.avi.md5.
ffmpeg -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
ffmpeg -i input.flv -f avi -y md5:
```

Note that some formats (typically MOV) require the output protocol to be seekable, so they will fail with the MD5 output protocol.

20.10 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

number is the number corresponding to the file descriptor of the pipe (e.g. 0 for stdin, 1 for stdout, 2 for stderr). If *number* is not specified, by default the stdout file descriptor will be used for writing, stdin for reading.

For example to read from stdin with `ffmpeg`:

```
cat test.wav | ffmpeg -i pipe:0
# ...this is the same as...
cat test.wav | ffmpeg -i pipe:
```

For writing to stdout with `ffmpeg`:

```
ffmpeg -i test.wav -f avi pipe:1 | cat > test.avi
# ...this is the same as...
ffmpeg -i test.wav -f avi pipe: | cat > test.avi
```

Note that some formats (typically MOV), require the output protocol to be seekable, so they will fail with the pipe output protocol.

20.11 rtmp

Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimedia content across a TCP/IP network.

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

The number of the TCP port to use (by default is 1935).

‘`app`’

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. ‘`/ondemand/`’, ‘`/flash/live/`’, etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

‘`playpath`’

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

Maximum time to wait for the incoming connection. Implies listen.

Additionally, the following parameters can be set via command line options (or in code via `AVOptions`):

‘`rtmp_app`’

Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

‘`rtmp_buffer`’

Set the client buffer time in milliseconds. The default is 3000.

‘`rtmp_conn`’

Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with ’N’ and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘`rtmp_flashver`’

Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘`rtmp_flush_interval`’

Number of packets flushed in the same request (RTMPT only). The default is 10.

‘`rtmp_live`’

Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is `any`, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are `live` and `recorded`.

‘`rtmp_pageurl`’

URL of the web page in which the media was embedded. By default no value will be sent.

‘`rtmp_playpath`’

Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘`rtmp_subscribe`’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘`rtmp_swfhash`’

SHA256 hash of the decompressed SWF file (32 bytes).

‘`rtmp_swfsize`’

Size of the decompressed SWF file, required for SWFVerification.

‘`rtmp_swfurl`’

URL of the SWF player for the media. By default no value will be sent.

‘`rtmp_swfverify`’

URL to player swf file, compute hash/size automatically.

‘`rtmp_tcurl`’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with `ffplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
ffplay rtmp://myserver/vod/sample
```

20.12 rtmpe

Encrypted Real-Time Messaging Protocol.

The Encrypted Real-Time Messaging Protocol (RTMPE) is used for streaming multimedia content within standard cryptographic primitives, consisting of Diffie-Hellman key exchange and HMACSHA256, generating a pair of RC4 keys.

20.13 rtmps

Real-Time Messaging Protocol over a secure SSL connection.

The Real-Time Messaging Protocol (RTMPS) is used for streaming multimedia content across an encrypted connection.

20.14 rtmpt

Real-Time Messaging Protocol tunneled through HTTP.

The Real-Time Messaging Protocol tunneled through HTTP (RTMPT) is used for streaming multimedia content within HTTP requests to traverse firewalls.

20.15 rtmpte

Encrypted Real-Time Messaging Protocol tunneled through HTTP.

The Encrypted Real-Time Messaging Protocol tunneled through HTTP (RTMPTE) is used for streaming multimedia content within HTTP requests to traverse firewalls.

20.16 rtmpts

Real-Time Messaging Protocol tunneled through HTTPS.

The Real-Time Messaging Protocol tunneled through HTTPS (RTMPTS) is used for streaming multimedia content within HTTPS requests to traverse firewalls.

20.17 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

Requires the presence of the librtmp headers and library during configuration. You need to explicitly configure the build with "–enable-librtmp". If enabled this will replace the native RTMP protocol.

This protocol provides most client functions and a few server functions needed to support RTMP, RTMP tunneled in HTTP (RTMPT), encrypted RTMP (RTMPE), RTMP over SSL/TLS (RTMPS) and tunneled variants of these encrypted types (RTMPTE, RTMPTS).

The required syntax is:

```
rtmp_proto://server[:port][/app][/playpath] options
```

where *rtmp_proto* is one of the strings "rtmp", "rtmpt", "rtmpe", "rtmps", "rtmpte", "rtmpts" corresponding to each RTMP variant, and *server*, *port*, *app* and *playpath* have the same meaning as specified for the RTMP native protocol. *options* contains a list of space-separated options of the form *key*=*val*.

See the librtmp manual page (man 3 librtmp) for more information.

For example, to stream a file in real-time to an RTMP server using `ffmpeg`:

```
ffmpeg -re -i myfile -f flv rtmp://myserver/live/mystream
```

To play the same stream using `ffplay`:

```
ffplay "rtmp://myserver/live/mystream live=1"
```

20.18 rtp

Real-Time Protocol.

20.19 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock's RTSP server).

The required syntax for a RTSP url is:

```
rtsp://hostname[:port]/path
```

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

'`udp`'

Use UDP as lower transport protocol.

'`tcp`'

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

'`udp_multicast`'

Use UDP multicast as lower transport protocol.

'`http`'

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

'`filter_src`'

Accept packets only from negotiated peer address and port.

'`listen`'

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

When watching multi-bitrate Real-RTSP streams with `ffplay`, the streams to display can be chosen with `-vst` *n* and `-ast` *n* for video and audio respectively, and can be switched on the fly by pressing `v` and `a`.

Example command lines:

To watch a stream over UDP, with a max reordering delay of 0.5 seconds:

```
ffplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

To watch a stream tunneled over HTTP:

```
ffplay -rtsp_transport http rtsp://server/video.mp4
```

To send a stream in realtime to a RTSP server, for others to watch:

```
ffmpeg -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

To receive a stream in realtime:

```
ffmpeg -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

20.20 sap

Session Announcement Protocol (RFC 2974). This is not technically a protocol handler in libavformat, it is a muxer and demuxer. It is used for signalling of RTP streams, by announcing the SDP for the streams regularly on a separate port.

20.20.1 Muxer

The syntax for a SAP url given to the muxer is:

```
sap://destination[:port][?options]
```

The RTP packets are sent to *destination* on port *port*, or to port 5004 if no port is specified. *options* is a &-separated list. The following options are supported:

‘`announce_addr=`*`address`*’

Specify the destination IP address for sending the announcements to. If omitted, the announcements are sent to the commonly used SAP announcement multicast address 224.2.127.254 (sap.mcast.net), or ff0e::2:7ffe if *destination* is an IPv6 address.

'`announce_port=`*`port`*'

Specify the port to send the announcements on, defaults to 9875 if not specified.

'`ttl=`*`ttl`*'

Specify the time to live value for the announcements and RTP packets, defaults to 255.

'`same_port=`*`0|1`*'

If set to 1, send all RTP streams on the same port pair. If zero (the default), all streams are sent on unique ports, with each stream on a port 2 numbers higher than the previous. VLC/Live555 requires this to be set to 1, to be able to receive the stream. The RTP stack in libavformat for receiving requires all streams to be sent on unique ports.

Example command lines follow.

To broadcast a stream on the local subnet, for watching in VLC:

```
ffmpeg -re -i input -f sap sap://224.0.0.255?same_port=1
```

Similarly, for watching in `ffplay`:

```
ffmpeg -re -i input -f sap sap://224.0.0.255
```

And for watching in `ffplay`, over IPv6:

```
ffmpeg -re -i input -f sap sap://[ff0e::1:2:3:4]
```

20.20.2 Demuxer

The syntax for a SAP url given to the demuxer is:

```
sap://[address][:port]
```

address is the multicast address to listen for announcements on, if omitted, the default 224.2.127.254 (sap.mcast.net) is used. *port* is the port that is listened on, 9875 if omitted.

The demuxers listens for announcements on the given address and port. Once an announcement is received, it tries to receive that particular stream.

Example command lines follow.

To play back the first stream announced on the normal SAP multicast address:

```
ffplay sap://
```

To play back the first stream announced on one the default IPv6 SAP multicast address:

```
ffplay sap://[ff0e::2:7ffe]
```

20.21 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

```
tcp://hostname:port[?options]
```

‘`listen`’

Listen for an incoming connection

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

20.22 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

```
tls://hostname:port[?options]
```

‘`listen`’

Act as a server, listening for an incoming connection.

‘`cafile=`*`filename`*’

Certificate authority file. The file must be in OpenSSL PEM format.

‘`cert=`*`filename`*’

Certificate file. The file must be in OpenSSL PEM format.

‘`key=`*`filename`*’

Private key file.

‘`verify=`*`0|1`*’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

```
ffplay tls://hostname:port
```

20.23 udp

User Datagram Protocol.

The required syntax for a UDP url is:

```
udp://hostname:port[?options]
```

options contains a list of &-separated options of the form *key*=*val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

‘`buffer_size=`*`size`*’

Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

‘`localport=`*`port`*’

Override the local UDP port to bind with.

‘`localaddr=`*`addr`*’

Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

‘`pkt_size=`*`size`*’

Set the size in bytes of UDP packets.

‘`reuse=`*`1|0`*’

Explicitly allow or disallow reusing UDP sockets.

‘`ttl=`*`ttl`*’

Set the time to live value (for multicast only).

‘`connect=`*`1|0`*’

Initialize the UDP socket with `connect()`. In this case, the destination address can’t be changed with ff_udp_set_remote_url later. If the destination address isn’t known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

‘`sources=`*`address`*`[,`*`address`*`]`’

Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘`block=`*`address`*`[,`*`address`*`]`’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘`fifo_size=`*`units`*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘`overrun_nonfatal=`*`1|0`*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

'`timeout=`*`microseconds`*'

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

```
ffmpeg -i input -f format udp://hostname:port
```

To stream in mpegts format over UDP using 188 sized UDP packets, using a large input buffer:

```
ffmpeg -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

To receive over UDP from a remote endpoint:

```
ffmpeg -i udp://[multicast-address]:port
```

21. Bitstream Filters

When you configure your FFmpeg build, all the supported bitstream filters are enabled by default. You can list all available ones using the configure option `--list-bsfs`.

You can disable all the bitstream filters using the configure option `--disable-bsfs`, and selectively enable any bitstream filter using the option `--enable-bsf=BSF`, or you can disable a particular bitstream filter using the option `--disable-bsf=BSF`.

The option `-bsfs` of the ff* tools will display the list of all the supported bitstream filters included in your build.

Below is a description of the currently available bitstream filters.

21.1 aac_adtstoasc

21.2 chomp

21.3 dump_extradata

21.4 h264_mp4toannexb

Convert an H.264 bitstream from length prefixed mode to start code prefixed mode (as defined in the Annex B of the ITU-T H.264 specification).

This is required by some streaming formats, typically the MPEG-2 transport stream format ("mpegts").

For example to remux an MP4 file containing an H.264 stream to mpegts format with `ffmpeg`, you can use the command:

```
ffmpeg -i INPUT.mp4 -codec copy -bsf:v h264_mp4toannexb OUTPUT.ts
```

21.5 imx_dump_header

21.6 mjpeg2jpeg

Convert MJPEG/AVI1 packets to full JPEG/JFIF packets.

MJPEG is a video codec wherein each video frame is essentially a JPEG image. The individual frames can be extracted without loss, e.g. by

```
ffmpeg -i ../some_mjpeg.avi -c:v copy frames_%d.jpg
```

Unfortunately, these chunks are incomplete JPEG images, because they lack the DHT segment required for decoding. Quoting from http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml:

Avery Lee, writing in the rec.video.desktop newsgroup in 2001, commented that "MJPEG, or at least the MJPEG in AVIs having the MJPG fourcc, is restricted JPEG with a fixed – and *omitted* – Huffman table. The JPEG must be YCbCr colorspace, it must be 4:2:2, and it must use basic Huffman encoding, not arithmetic or progressive. . . . You can indeed extract the MJPEG frames and decode them with a regular JPEG decoder, but you have to prepend the DHT segment to them, or else the decoder won't have any idea how to decompress the data. The exact table necessary is given in the OpenDML spec."

This bitstream filter patches the header of frames extracted from an MJPEG stream (carrying the AVI1 header ID and lacking a DHT segment) to produce fully qualified JPEG images.

```
ffmpeg -i mjpeg-movie.avi -c:v copy -bsf:v mjpeg2jpeg frame_%d.jpg
exiftran -i -9 frame*.jpg
ffmpeg -i frame_%d.jpg -c:v copy rotated.avi
```

21.7 mjpega_dump_header

21.8 movsub

21.9 mp3_header_compress

21.10 mp3_header_decompress

21.11 noise

21.12 remove_extradata

22. Filtering Introduction

Filtering in FFmpeg is enabled through the libavfilter library.

Libavfilter is the filtering API of FFmpeg. It is the substitute of the now deprecated 'vhooks' and started as a Google Summer of Code project.

Audio filtering integration into the main FFmpeg repository is a work in progress, so audio API and ABI should not be considered stable yet.

In libavfilter, it is possible for filters to have multiple inputs and multiple outputs. To illustrate the sorts of things that are possible, we can use a complex filter graph. For example, the following one:

```
input --> split --> fifo -----------------------> overlay --> output
            |                                        ^
            |                                        |
            +------> fifo --> crop --> vflip --------+
```

splits the stream in two streams, sends one stream through the crop filter and the vflip filter before merging it back with the other stream by overlaying it on top. You can use the following command to achieve this:

```
ffmpeg -i input -vf "[in] split [T1], fifo, [T2] overlay=0:H/2 [out]; [T1] fifo, crop=iw:ih/2:0:ih/2, vflip [T2]" output
```

The result will be that in output the top half of the video is mirrored onto the bottom half.

Filters are loaded using the *-vf* or *-af* option passed to `ffmpeg` or to `ffplay`. Filters in the same linear chain are separated by commas. In our example, *split, fifo, overlay* are in one linear chain, and *fifo, crop, vflip* are in another. The points where the linear chains join are labeled by names enclosed in square brackets. In our example, that is *[T1]* and *[T2]*. The special labels *[in]* and *[out]* are the points where video is input and output.

Some filters take in input a list of parameters: they are specified after the filter name and an equal sign, and are separated from each other by a colon.

There exist so-called *source filters* that do not have an audio/video input, and *sink filters* that will not have audio/video output.

23. graph2dot

The ‘`graph2dot`’ program included in the FFmpeg ‘`tools`’ directory can be used to parse a filter graph description and issue a corresponding textual representation in the dot language.

Invoke the command:

```
graph2dot -h
```

to see how to use ‘`graph2dot`’.

You can then pass the dot description to the ‘`dot`’ program (from the graphviz suite of programs) and obtain a graphical representation of the filter graph.

For example the sequence of commands:

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

can be used to create and display an image representing the graph described by the *GRAPH_DESCRIPTION* string. Note that this string must be a complete self-contained graph, with its inputs and outputs explicitly defined. For example if your command line is of the form:

```
ffmpeg -i infile -vf scale=640:360 outfile
```

your *GRAPH_DESCRIPTION* string will need to be of the form:

```
nullsrc,scale=640:360,nullsink
```

you may also need to set the *nullsrc* parameters and add a *format* filter in order to simulate a specific input file.

24. Filtergraph description

A filtergraph is a directed graph of connected filters. It can contain cycles, and there can be multiple links between a pair of filters. Each link has one input pad on one side connecting it to one filter from which it takes its input, and one output pad on the other side connecting it to the one filter accepting its output.

Each filter in a filtergraph is an instance of a filter class registered in the application, which defines the features and the number of input and output pads of the filter.

A filter with no input pads is called a "source", a filter with no output pads is called a "sink".

24.1 Filtergraph syntax

A filtergraph can be represented using a textual representation, which is recognized by the ‘`-filter`’/‘`-vf`’ and ‘`-filter_complex`’ options in `ffmpeg` and ‘`-vf`’ in `ffplay`, and by the `avfilter_graph_parse()`/`avfilter_graph_parse2()` function defined in ‘`libavfilter/avfiltergraph.h`’.

A filterchain consists of a sequence of connected filters, each one connected to the previous one in the sequence. A filterchain is represented by a list of ","-separated filter descriptions.

A filtergraph consists of a sequence of filterchains. A sequence of filterchains is represented by a list of ";"-separated filterchain descriptions.

A filter is represented by a string of the form:
[*in_link_1*]...[*in_link_N*]*filter_name*=*arguments*[*out_link_1*]...[*out_link_M*]

filter_name is the name of the filter class of which the described filter is an instance of, and has to be the name of one of the filter classes registered in the program. The name of the filter class is optionally followed by a string "=*arguments*".

arguments is a string which contains the parameters used to initialize the filter instance, and are described in the filter descriptions below.

The list of arguments can be quoted using the character "’" as initial and ending mark, and the character ’\’ for escaping the characters within the quoted text; otherwise the argument string is considered terminated when the next special character (belonging to the set "[]=;,") is encountered.

The name and arguments of the filter are optionally preceded and followed by a list of link labels. A link label allows to name a link and associate it to a filter output or input pad. The preceding labels *in_link_1* ... *in_link_N*, are associated to the filter input pads, the following labels *out_link_1* ... *out_link_M*, are associated to the output pads.

When two link labels with the same name are found in the filtergraph, a link between the corresponding input and output pad is created.

If an output pad is not labelled, it is linked by default to the first unlabelled input pad of the next filter in the filterchain. For example in the filterchain:

```
nullsrc, split[L1], [L2]overlay, nullsink
```

the split filter instance has two output pads, and the overlay filter instance two input pads. The first output pad of split is labelled "L1", the first input pad of overlay is labelled "L2", and the second output pad of split is linked to the second input pad of overlay, which are both unlabelled.

In a complete filterchain all the unlabelled filter input and output pads must be connected. A filtergraph is considered valid if all the filter input and output pads of all the filterchains are connected.

Libavfilter will automatically insert scale filters where format conversion is required. It is possible to specify swscale flags for those automatically inserted scalers by prepending `sws_flags=flags;` to the filtergraph description.

Follows a BNF description for the filtergraph syntax:

```
NAME             ::= sequence of alphanumeric characters and '_'
LINKLABEL        ::= "[" NAME "]"
LINKLABELS       ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (eventually quoted)
FILTER           ::= [LINKNAMES] NAME ["=" ARGUMENTS] [LINKNAMES]
FILTERCHAIN      ::= FILTER [,FILTERCHAIN]
FILTERGRAPH      ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

24.2 Notes on filtergraph escaping

Some filter arguments require the use of special characters, typically `:` to separate key=value pairs in a named options list. In this case the user should perform a first level escaping when specifying the filter arguments. For example, consider the following literal string to be embedded in the drawtext filter arguments:

```
this is a 'string': may contain one, or more, special characters
```

Since `:` is special for the filter arguments syntax, it needs to be escaped, so you get:

```
text=this is a \'string\'\: may contain one, or more, special characters
```

A second level of escaping is required when embedding the filter arguments in a filtergraph description, in order to escape all the filtergraph special characters. Thus the example above becomes:

```
drawtext=text=this is a \\\'string\\\'\\: may contain one\, or more\, special characters
```

Finally an additional level of escaping may be needed when writing the filtergraph description in a shell command, which depends on the escaping rules of the adopted shell. For example, assuming that `\` is special and needs to be escaped with another `\`, the previous string will finally result in:

```
-vf "drawtext=text=this is a \\\\\\’string\\\\\\’\\\\: may contain one\\, or more\\, special characters"
```

Sometimes, it might be more convenient to employ quoting in place of escaping. For example the string:

```
Caesar: tu quoque, Brute, fili mi
```

Can be quoted in the filter arguments as:

```
text=’Caesar: tu quoque, Brute, fili mi’
```

And finally inserted in a filtergraph like:

```
drawtext=text=\’Caesar: tu quoque\, Brute\, fili mi\’
```

See the Quoting and escaping section for more information about the escaping and quoting rules adopted by FFmpeg.

25. Audio Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the audio filters included in your build.

Below is a description of the currently available audio filters.

25.1 aconvert

Convert the input audio format to the specified formats.

The filter accepts a string of the form: "*sample_format*:*channel_layout*".

sample_format specifies the sample format, and can be a string or the corresponding numeric value defined in ‘`libavutil/samplefmt.h`’. Use ’p’ suffix for a planar sample format.

channel_layout specifies the channel layout, and can be a string or the corresponding number value defined in ‘`libavutil/channel_layout.h`’.

The special parameter "auto", signifies that the filter will automatically select the output format depending on the output filter.

Some examples follow.

- Convert input to float, planar, stereo:

  ```
  aconvert=fltp:stereo
  ```

- Convert input to unsigned 8-bit, automatically select out channel layout:

  ```
  aconvert=u8:auto
  ```

25.2 aformat

Convert the input audio to one of the specified formats. The framework will negotiate the most appropriate format to minimize conversions.

The filter accepts the following named parameters:

‘`sample_fmts`’

A comma-separated list of requested sample formats.

‘`sample_rates`’

A comma-separated list of requested sample rates.

‘`channel_layouts`’

A comma-separated list of requested channel layouts.

If a parameter is omitted, all values are allowed.

For example to force the output to either unsigned 8-bit or signed 16-bit stereo:

```
aformat=sample_fmts\=u8\,s16:channel_layouts\=stereo
```

25.3 amerge

Merge two or more audio streams into a single multi-channel stream.

The filter accepts the following named options:

‘`inputs`’

Set the number of inputs. Default is 2.

If the channel layouts of the inputs are disjoint, and therefore compatible, the channel layout of the output will be set accordingly and the channels will be reordered as necessary. If the channel layouts of the inputs are not disjoint, the output will have all the channels of the first input then all the channels of the second input, in that order, and the channel layout of the output will be the default value corresponding to the total number of channels.

For example, if the first input is in 2.1 (FL+FR+LF) and the second input is FC+BL+BR, then the output will be in 5.1, with the channels in the following order: a1, a2, b1, a3, b2, b3 (a1 is the first channel of the first input, b1 is the first channel of the second input).

On the other hand, if both input are in stereo, the output channels will be in the default order: a1, a2, b1, b2, and the channel layout will be arbitrarily set to 4.0, which may or may not be the expected value.

All inputs must have the same sample rate, and format.

If inputs do not have the same duration, the output will stop with the shortest.

Example: merge two mono files into a stereo stream:

```
amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
```

Example: multiple merges:

```
ffmpeg -f lavfi -i "
amovie=input.mkv:si=0 [a0];
amovie=input.mkv:si=1 [a1];
amovie=input.mkv:si=2 [a2];
amovie=input.mkv:si=3 [a3];
amovie=input.mkv:si=4 [a4];
amovie=input.mkv:si=5 [a5];
[a0][a1][a2][a3][a4][a5] amerge=inputs=6" -c:a pcm_s16le output.mkv
```

25.4 amix

Mixes multiple audio inputs into a single output.

For example

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

will mix 3 input audio streams to a single output with the same duration as the first input and a dropout transition time of 3 seconds.

The filter accepts the following named parameters:

‘`inputs`’

Number of inputs. If unspecified, it defaults to 2.

‘`duration`’

How to determine the end-of-stream.

‘`longest`’

Duration of longest input. (default)

‘`shortest`’

Duration of shortest input.

‘`first`’

Duration of first input.

‘`dropout_transition`’

Transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

25.5 anull

Pass the audio source unchanged to the output.

25.6 aresample

Resample the input audio to the specified sample rate.

The filter accepts exactly one parameter, the output sample rate. If not specified then the filter will automatically convert between its input and output sample rates.

For example, to resample the input audio to 44100Hz:

```
aresample=44100
```

25.7 asetnsamples

Set the number of samples per each output audio frame.

The last output packet may contain a different number of samples, as the filter will flush all the remaining samples when the input audio signal its end.

The filter accepts parameters as a list of *key*=*value* pairs, separated by ":".

‘`nb_out_samples, n`’

> Set the number of frames per each output audio frame. The number is intended as the number of samples *per each channel*. Default value is 1024.

‘`pad, p`’

> If set to 1, the filter will pad the last audio frame with zeroes, so that the last frame will contain the same number of samples as the previous ones. Default value is 1.

For example, to set the number of per-frame samples to 1234 and disable padding for the last frame, use:

```
asetnsamples=n=1234:p=0
```

25.8 ashowinfo

Show a line containing various information for each input audio frame. The input audio is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

A description of each shown parameter follows:

‘`n`’

> sequential number of the input frame, starting from 0

‘`pts`’

> Presentation timestamp of the input frame, in time base units; the time base depends on the filter input pad, and is usually 1/*sample_rate*.

‘`pts_time`’

> presentation timestamp of the input frame in seconds

‘`pos`’

> position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic audio)

‘`fmt`’

> sample format

‘`chlayout`’

channel layout

‘`rate`’

sample rate for the audio frame

‘`nb_samples`’

number of samples (per channel) in the frame

‘`checksum`’

Adler-32 checksum (printed in hexadecimal) of the audio data. For planar audio the data is treated as if all the planes were concatenated.

‘`plane_checksums`’

A list of Adler-32 checksums for each data plane.

25.9 asplit

Split input audio into several identical outputs.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

For example:

```
[in] asplit [out0][out1]
```

will create two separate outputs from the same input.

To create 3 or more outputs, you need to specify the number of outputs, like in:

```
[in] asplit=3 [out0][out1][out2]
```

```
ffmpeg -i INPUT -filter_complex asplit=5 OUTPUT
```

will create 5 copies of the input audio.

25.10 astreamsync

Forward two audio streams and control the order the buffers are forwarded.

The argument to the filter is an expression deciding which stream should be forwarded next: if the result is negative, the first stream is forwarded; if the result is positive or zero, the second stream is forwarded. It can use the following variables:

b1 b2

number of buffers forwarded so far on each stream

s1 s2

number of samples forwarded so far on each stream

t1 t2

current timestamp of each stream

The default value is `t1-t2`, which means to always forward the stream that has a smaller timestamp.

Example: stress-test `amerge` by randomly sending buffers on the wrong input, while avoiding too much of a desynchronization:

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

25.11 atempo

Adjust audio tempo.

The filter accepts exactly one parameter, the audio tempo. If not specified then the filter will assume nominal 1.0 tempo. Tempo must be in the [0.5, 2.0] range.

For example, to slow down audio to 80% tempo:

```
atempo=0.8
```

For example, to speed up audio to 125% tempo:

```
atempo=1.25
```

25.12 earwax

Make audio easier to listen to on headphones.

This filter adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Ported from SoX.

25.13 pan

Mix channels with specific gain levels. The filter accepts the output channel layout followed by a set of channels definitions.

This filter is also designed to remap efficiently the channels of an audio stream.

The filter accepts parameters of the form: "*l*:*outdef*:*outdef*:..."

‘`l`’

  output channel layout or number of channels

‘`outdef`’

  output channel specification, of the form: "*out_name*=[*gain**]*in_name*[+[*gain**]*in_name*...]"

‘`out_name`’

  output channel to define, either a channel name (FL, FR, etc.) or a channel number (c0, c1, etc.)

‘`gain`’

  multiplicative coefficient for the channel, 1 leaving the volume unchanged

‘`in_name`’

  input channel to use, see out_name for details; it is not possible to mix named and numbered input channels

If the ‘=’ in a channel specification is replaced by ‘<’, then the gains for that specification will be renormalized so that the total is 1, thus avoiding clipping noise.

25.13.1 Mixing examples

For example, if you want to down-mix from stereo to mono, but with a bigger factor for the left channel:

```
pan=1:c0=0.9*c0+0.1*c1
```

A customized down-mix to stereo that works automatically for 3-, 4-, 5- and 7-channels surround:

```
pan=stereo: FL < FL + 0.5*FC + 0.6*BL + 0.6*SL : FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

Note that `ffmpeg` integrates a default down-mix (and up-mix) system that should be preferred (see "-ac" option) unless you have very specific needs.

25.13.2 Remapping examples

The channel remapping will be effective if, and only if:

- gain coefficients are zeroes or ones,
- only one input per channel output,

If all these conditions are satisfied, the filter will notify the user ("Pure channel mapping detected"), and use an optimized and lossless method to do the remapping.

For example, if you have a 5.1 source and want a stereo audio stream by dropping the extra channels:

```
pan="stereo: c0=FL : c1=FR"
```

Given the same source, you can also switch front left and front right channels and keep the input channel layout:

```
pan="5.1: c0=c1 : c1=c0 : c2=c2 : c3=c3 : c4=c4 : c5=c5"
```

If the input is a stereo audio stream, you can mute the front left channel (and still keep the stereo channel layout) with:

```
pan="stereo:c1=c1"
```

Still with a stereo audio stream input, you can copy the right channel in both front left and right:

```
pan="stereo: c0=FR : c1=FR"
```

25.14 silencedetect

Detect silence in an audio stream.

This filter logs a message when it detects that the input audio volume is less or equal to a noise tolerance value for a duration greater or equal to the minimum detected noise duration.

The printed times and duration are expressed in seconds.

'`duration, d`'

> Set silence duration until notification (default is 2 seconds).

'`noise, n`'

> Set noise tolerance. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default is -60dB, or 0.001.

Detect 5 seconds of silence with -50dB noise tolerance:

```
silencedetect=n=-50dB:d=5
```

Complete example with `ffmpeg` to detect silence with 0.0001 noise tolerance in '`silence.mp3`':

```
ffmpeg -f lavfi -i amovie=silence.mp3,silencedetect=noise=0.0001 -f null -
```

25.15 volume

Adjust the input audio volume.

The filter accepts exactly one parameter *vol*, which expresses how the audio volume will be increased or decreased.

Output values are clipped to the maximum value.

If *vol* is expressed as a decimal number, the output audio volume is given by the relation:

```
output_volume = vol * input_volume
```

If *vol* is expressed as a decimal number followed by the string "dB", the value represents the requested change in decibels of the input audio power, and the output audio volume is given by the relation:

```
output_volume = 10^(vol/20) * input_volume
```

Otherwise *vol* is considered an expression and its evaluated value is used for computing the output audio volume according to the first relation.

Default value for *vol* is 1.0.

25.15.1 Examples

- Half the input audio volume:

  ```
  volume=0.5
  ```

  The above example is equivalent to:

  ```
  volume=1/2
  ```

- Decrease input audio power by 12 decibels:

  ```
  volume=-12dB
  ```

25.16 volumedetect

Detect the volume of the input video.

The filter has no parameters. The input is not modified. Statistics about the volume will be printed in the log when the input stream end is reached.

In particular it will show the mean volume (root mean square), maximum volume (on a per-sample basis), and the beginning of an histogram of the registered volume values (from the maximum value to a cumulated 1/1000 of the samples).

All volumes are in decibels relative to the maximum PCM value.

Here is an excerpt of the output:

```
[Parsed_volumedetect_0  0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0  0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0  0xa23120] histogram_4db: 6
[Parsed_volumedetect_0  0xa23120] histogram_5db: 62
[Parsed_volumedetect_0  0xa23120] histogram_6db: 286
[Parsed_volumedetect_0  0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0  0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0  0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0  0xa23120] histogram_10db: 8409
```

It means that:

- The mean square energy is approximately -27 dB, or 10^-2.7.
- The largest sample is at -4 dB, or more precisely between -4 dB and -5 dB.
- There are 6 samples at -4 dB, 62 at -5 dB, 286 at -6 dB, etc.

In other words, raising the volume by +4 dB does not cause any clipping, raising it by +5 dB causes clipping for 6 samples, etc.

25.17 asyncts

Synchronize audio data with timestamps by squeezing/stretching it and/or dropping samples/adding silence when needed.

The filter accepts the following named parameters:

‘`compensate`’

> Enable stretching/squeezing the data to make it match the timestamps. Disabled by default. When disabled, time gaps are covered with silence.

‘`min_delta`’

> Minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. Default value is 0.1. If you get non-perfect sync with this filter, try setting this parameter to 0.

‘`max_comp`’

> Maximum compensation in samples per second. Relevant only with compensate=1. Default value 500.

‘`first_pts`’

> Assume the first pts should be this value. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame’s expected pts, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream.

25.18 channelsplit

Split each channel in input audio stream into a separate output stream.

This filter accepts the following named parameters:

‘`channel_layout`’

> Channel layout of the input stream. Default is "stereo".

For example, assuming a stereo input MP3 file

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

will create an output Matroska file with two audio streams, one containing only the left channel and the other the right channel.

To split a 5.1 WAV file into per-channel files

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

25.19 channelmap

Remap input channels to new locations.

This filter accepts the following named parameters:

'`channel_layout`'

Channel layout of the output stream.

'`map`'

Map channels from input to output. The argument is a comma-separated list of mappings, each in the `in_channel-out_channel` or *in_channel* form. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the input channel layout. *out_channel* is the name of the output channel or its index in the output channel layout. If *out_channel* is not given then it is implicitly an index, starting with zero and increasing by one for each mapping.

If no mapping is present, the filter will implicitly map input channels to output channels preserving index.

For example, assuming a 5.1+downmix input MOV file

```
ffmpeg -i in.mov -filter 'channelmap=map=DL-FL\,DR-FR' out.wav
```

will create an output WAV file tagged as stereo from the downmix channels of the input.

To fix a 5.1 WAV improperly encoded in AAC's native channel order

```
ffmpeg -i in.wav -filter 'channelmap=1\,2\,0\,5\,3\,4:channel_layout=5.1' out.wav
```

25.20 join

Join multiple input streams into one multi-channel stream.

The filter accepts the following named parameters:

‘`inputs`’

Number of input streams. Defaults to 2.

‘`channel_layout`’

Desired output channel layout. Defaults to stereo.

‘`map`’

Map channels from inputs to output. The argument is a comma-separated list of mappings, each in the *`input_idx.in_channel-out_channel`* form. *input_idx* is the 0-based index of the input stream. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the specified input stream. *out_channel* is the name of the output channel.

The filter will attempt to guess the mappings when those are not specified explicitly. It does so by first trying to find an unused matching input channel and if that fails it picks the first unused input channel.

E.g. to join 3 inputs (with properly set channel layouts)

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

To build a 5.1 output from 6 single-channel streams:

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex
’join=inputs=6:channel_layout=5.1:map=0.0-FL\,1.0-FR\,2.0-FC\,3.0-SL\,4.0-SR\,5.0-LFE’
out
```

25.21 resample

Convert the audio sample format, sample rate and channel layout. This filter is not meant to be used directly.

26. Audio Sources

Below is a description of the currently available audio sources.

26.1 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in ‘`libavfilter/asrc_abuffer.h`’.

It accepts the following mandatory parameters: *sample_rate*:*sample_fmt*:*channel_layout*

‘`sample_rate`’

> The sample rate of the incoming audio buffers.

‘`sample_fmt`’

> The sample format of the incoming audio buffers. Either a sample format name or its corresponging integer representation from the enum AVSampleFormat in ‘`libavutil/samplefmt.h`’

‘`channel_layout`’

> The channel layout of the incoming audio buffers. Either a channel layout name from channel_layout_map in ‘`libavutil/channel_layout.c`’ or its corresponding integer representation from the AV_CH_LAYOUT_* macros in ‘`libavutil/channel_layout.h`’

For example:

```
abuffer=44100:s16p:stereo
```

will instruct the source to accept planar 16bit signed stereo at 44100Hz. Since the sample format with name "s16p" corresponds to the number 6 and the "stereo" channel layout corresponds to the value 0x3, this is equivalent to:

```
abuffer=44100:6:0x3
```

26.2 aevalsrc

Generate an audio signal specified by an expression.

This source accepts in input one or more expressions (one for each channel), which are evaluated and used to generate a corresponding audio signal.

It accepts the syntax: *exprs*[::*options*]. *exprs* is a list of expressions separated by ":", one for each separate channel. In case the *channel_layout* is not specified, the selected channel layout depends on the number of provided expressions.

options is an optional sequence of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`channel_layout, c`’

> Set the channel layout. The number of channels in the specified layout must be equal to the number of specified expressions.

‘`duration, d`’

Set the minimum duration of the sourced audio. See the function `av_parse_time()` for the accepted format. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame.

If not specified, or the expressed duration is negative, the audio is supposed to be generated forever.

‘`nb_samples, n`’

Set the number of samples per channel per each output frame, default to 1024.

‘`sample_rate, s`’

Specify the sample rate, default to 44100.

Each expression in *exprs* can contain the following constants:

‘`n`’

number of the evaluated sample, starting from 0

‘`t`’

time of the evaluated sample expressed in seconds, starting from 0

‘`s`’

sample rate

26.2.1 Examples

- Generate silence:

  ```
  aevalsrc=0
  ```

- Generate a sin signal with frequency of 440 Hz, set sample rate to 8000 Hz:

  ```
  aevalsrc="sin(440*2*PI*t)::s=8000"
  ```

- Generate a two channels signal, specify the channel layout (Front Center + Back Center) explicitly:

  ```
  aevalsrc="sin(420*2*PI*t):cos(430*2*PI*t)::c=FC|BC"
  ```

- Generate white noise:

```
aevalsrc="-2+random(0)"
```

- Generate an amplitude modulated signal:

```
aevalsrc="sin(10*2*PI*t)*sin(880*2*PI*t)"
```

- Generate 2.5 Hz binaural beats on a 360 Hz carrier:

```
aevalsrc="0.1*sin(2*PI*(360-2.5/2)*t) : 0.1*sin(2*PI*(360+2.5/2)*t)"
```

26.3 anullsrc

Null audio source, return unprocessed audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools, or as the source for filters which ignore the input data (for example the sox synth filter).

It accepts an optional sequence of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`sample_rate, s`’

Specify the sample rate, and defaults to 44100.

‘`channel_layout, cl`’

Specify the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is "stereo".

Check the channel_layout_map definition in ‘`libavutil/channel_layout.c`’ for the mapping between strings and channel layout values.

‘`nb_samples, n`’

Set the number of samples per requested frames.

Follow some examples:

```
#  set the sample rate to 48000 Hz and the channel layout to AV_CH_LAYOUT_MONO.
anullsrc=r=48000:cl=4

# same as
anullsrc=r=48000:cl=mono
```

26.4 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is not intended to be part of user-supplied graph descriptions but for insertion by calling programs through the interface defined in '`libavfilter/buffersrc.h`'.

It accepts the following named parameters:

'`time_base`'

> Timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

'`sample_rate`'

> Audio sample rate.

'`sample_fmt`'

> Name of the sample format, as returned by `av_get_sample_fmt_name()`.

'`channel_layout`'

> Channel layout of the audio data, in the form that can be accepted by `av_get_channel_layout()`.

All the parameters need to be explicitly defined.

26.5 flite

Synthesize a voice utterance using the libflite library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libflite`.

Note that the flite library is not thread-safe.

The source accepts parameters as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

'`list_voices`'

> If set to 1, list the names of the available voices and exit immediately. Default value is 0.

'`nb_samples, n`'

Set the maximum number of samples per frame. Default value is 512.

‘`textfile`’

Set the filename containing the text to speak.

‘`text`’

Set the text to speak.

‘`voice, v`’

Set the voice to use for the speech synthesis. Default value is `kal`. See also the *list_voices* option.

26.5.1 Examples

- Read from file ‘`speech.txt`’, and synthetize the text using the standard flite voice:

  ```
  flite=textfile=speech.txt
  ```

- Read the specified text selecting the `slt` voice:

  ```
  flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Input text to ffmpeg:

  ```
  ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
  ```

- Make ‘`ffplay`’ speak the specified text, using `flite` and the `lavfi` device:

  ```
  ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
  ```

For more information about libflite, check: http://www.speech.cs.cmu.edu/flite/

27. Audio Sinks

Below is a description of the currently available audio sinks.

27.1 abuffersink

Buffer audio frames, and make them available to the end of filter chain.

This sink is mainly intended for programmatic use, in particular through the interface defined in ‘`libavfilter/buffersink.h`’.

It requires a pointer to an AVABufferSinkContext structure, which defines the incoming buffers' formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

27.2 anullsink

Null audio sink, do absolutely nothing with the input audio. It is mainly useful as a template and to be employed in analysis / debugging tools.

27.3 abuffersink

This sink is intended for programmatic use. Frames that arrive on this sink can be retrieved by the calling program using the interface defined in '`libavfilter/buffersink.h`'.

This filter accepts no parameters.

28. Video Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the video filters included in your build.

Below is a description of the currently available video filters.

28.1 alphaextract

Extract the alpha component from the input as a grayscale video. This is especially useful with the *alphamerge* filter.

28.2 alphamerge

Add or replace the alpha component of the primary input with the grayscale value of a second input. This is intended for use with *alphaextract* to allow the transmission or storage of frame sequences that have alpha in a format that doesn't support an alpha channel.

For example, to reconstruct full frames from a normal YUV-encoded video and a separate video created with *alphaextract*, you might use:

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

Since this filter is designed for reconstruction, it operates on frame sequences without considering timestamps, and terminates when either input reaches end of stream. This will cause problems if your encoding pipeline drops frames. If you're trying to apply an image as an overlay to a video stream, consider the *overlay* filter instead.

28.3 ass

Draw ASS (Advanced Substation Alpha) subtitles on top of input video using the libass library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libass`.

This filter accepts the following named options, expressed as a sequence of *key*=*value* pairs, separated by ":".

‘`filename, f`’

Set the filename of the ASS file to read. It must be specified.

‘`original_size`’

Specify the size of the original video, the video for which the ASS file was composed. Due to a misdesign in ASS aspect ratio arithmetic, this is necessary to correctly scale the fonts if the aspect ratio has been changed.

If the first key is not specified, it is assumed that the first value specifies the ‘`filename`’.

For example, to render the file ‘`sub.ass`’ on top of the input video, use the command:

```
ass=sub.ass
```

which is equivalent to:

```
ass=filename=sub.ass
```

28.4 bbox

Compute the bounding box for the non-black pixels in the input frame luminance plane.

This filter computes the bounding box containing all the pixels with a luminance value greater than the minimum allowed value. The parameters describing the bounding box are printed on the filter log.

28.5 blackdetect

Detect video intervals that are (almost) completely black. Can be useful to detect chapter transitions, commercials, or invalid recordings. Output lines contains the time for the start, end and duration of the detected black interval expressed in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

This filter accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`black_min_duration, d`’

Set the minimum detected black duration expressed in seconds. It must be a non-negative floating point number.

Default value is 2.0.

‘`picture_black_ratio_th, pic_th`’

Set the threshold for considering a picture "black". Express the minimum value for the ratio:

```
nb_black_pixels / nb_pixels
```

for which a picture is considered black. Default value is 0.98.

‘`pixel_black_th, pix_th`’

Set the threshold for considering a pixel "black".

The threshold expresses the maximum pixel luminance value for which a pixel is considered "black". The provided value is scaled according to the following equation:

```
absolute_threshold = luminance_minimum_value + pixel_black_th * luminance_range_size
```

luminance_range_size and *luminance_minimum_value* depend on the input video format, the range is [0-255] for YUV full-range formats and [16-235] for YUV non full-range formats.

Default value is 0.10.

The following example sets the maximum pixel threshold to the minimum value, and detects only black intervals of 2 or more seconds:

```
blackdetect=d=2:pix_th=0.00
```

28.6 blackframe

Detect frames that are (almost) completely black. Can be useful to detect chapter transitions or commercials. Output lines consist of the frame number of the detected frame, the percentage of blackness, the position in the file if known or -1 and the timestamp in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the syntax:

```
blackframe[=amount:[threshold]]
```

amount is the percentage of the pixels that have to be below the threshold, and defaults to 98.

threshold is the threshold below which a pixel value is considered black, and defaults to 32.

28.7 boxblur

Apply boxblur algorithm to the input video.

This filter accepts the parameters:
luma_radius:*luma_power*:*chroma_radius*:*chroma_power*:*alpha_radius*:*alpha_power*

Chroma and alpha parameters are optional, if not specified they default to the corresponding values set for *luma_radius* and *luma_power*.

luma_radius, *chroma_radius*, and *alpha_radius* represent the radius in pixels of the box used for blurring the corresponding input plane. They are expressions, and can contain the following constants:

‘`w, h`’

the input width and height in pixels

‘`cw, ch`’

the input chroma image width and height in pixels

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

The radius must be a non-negative number, and must not be greater than the value of the expression `min(w,h)/2` for the luma and alpha planes, and of `min(cw,ch)/2` for the chroma planes.

luma_power, *chroma_power*, and *alpha_power* represent how many times the boxblur filter is applied to the corresponding plane.

Some examples follow:

- Apply a boxblur filter with luma, chroma, and alpha radius set to 2:

  ```
  boxblur=2:1
  ```

- Set luma radius to 2, alpha and chroma radius to 0

  ```
  boxblur=2:1:0:0:0:0
  ```

- Set luma and chroma radius to a fraction of the video dimension

  ```
  boxblur=min(h\,w)/10:1:min(cw\,ch)/10:1
  ```

28.8 colormatrix

The colormatrix filter allows conversion between any of the following color space: BT.709 (*bt709*), BT.601 (*bt601*), SMPTE-240M (*smpte240m*) and FCC (*fcc*).

The syntax of the parameters is *source*:*destination*:

```
colormatrix=bt601:smpte240m
```

28.9 copy

Copy the input source unchanged to the output. Mainly useful for testing purposes.

28.10 crop

Crop the input video to *out_w*:*out_h*:*x*:*y*:*keep_aspect*

The *keep_aspect* parameter is optional, if specified and set to a non-zero value will force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

The *out_w*, *out_h*, *x*, *y* parameters are expressions containing the following constants:

‘`x, y`’

  the computed values for *x* and *y*. They are evaluated for each new frame.

‘`in_w, in_h`’

  the input width and height

‘`iw, ih`’

  same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘`n`’

the number of input frame, starting from 0

‘`pos`’

the position in the file of the input frame, NAN if unknown

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

The *out_w* and *out_h* parameters specify the expressions for the width and height of the output (cropped) video. They are evaluated just at the configuration of the filter.

The default value of *out_w* is "in_w", and the default value of *out_h* is "in_h".

The expression for *out_w* may depend on the value of *out_h*, and the expression for *out_h* may depend on *out_w*, but they cannot depend on *x* and *y*, as *x* and *y* are evaluated after *out_w* and *out_h*.

The *x* and *y* parameters specify the expressions for the position of the top-left corner of the output (non-cropped) area. They are evaluated for each frame. If the evaluated value is not valid, it is approximated to the nearest valid value.

The default value of *x* is "(in_w-out_w)/2", and the default value for *y* is "(in_h-out_h)/2", which set the cropped area at the center of the input image.

The expression for *x* may depend on *y*, and the expression for *y* may depend on *x*.

Follow some examples:

```
# crop the central input area with size 100x100
crop=100:100

# crop the central input area with size 2/3 of the input video
"crop=2/3*in_w:2/3*in_h"

# crop the input video central square
crop=in_h

# delimit the rectangle with the top-left corner placed at position
# 100:100 and the right-bottom corner corresponding to the right-bottom
# corner of the input image.
crop=in_w-100:in_h-100:100:100

# crop 10 pixels from the left and right borders, and 20 pixels from
# the top and bottom borders
"crop=in_w-2*10:in_h-2*20"

# keep only the bottom right quarter of the input image
"crop=in_w/2:in_h/2:in_w/2:in_h/2"

# crop height for getting Greek harmony
"crop=in_w:1/PHI*in_w"

# trembling effect
"crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)"

# erratic camera effect depending on timestamp
"crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"

# set x depending on the value of y
"crop=in_w/2:in_h/2:y:10+10*sin(n/10)"
```

28.11 cropdetect

Auto-detect crop size.

Calculate necessary cropping parameters and prints the recommended parameters through the logging system. The detected dimensions correspond to the non-black area of the input video.

It accepts the syntax:

```
cropdetect[=limit[:round[:reset]]]
```

‘`limit`’

Threshold, which can be optionally specified from nothing (0) to everything (255), defaults to 24.

‘`round`’

Value which the width/height should be divisible by, defaults to 16. The offset is automatically adjusted to center the video. Use 2 to get only even dimensions (needed for 4:2:2 video). 16 is best when encoding to most video codecs.

‘`reset`’

> Counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Defaults to 0.
>
> This can be useful when channel logos distort the video area. 0 indicates never reset and return the largest area encountered during playback.

28.12 decimate

This filter drops frames that do not differ greatly from the previous frame in order to reduce framerate. The main use of this filter is for very-low-bitrate encoding (e.g. streaming over dialup modem), but it could in theory be used for fixing movies that were inverse-telecined incorrectly.

It accepts the following parameters: *max*:*hi*:*lo*:*frac*.

‘`max`’

> Set the maximum number of consecutive frames which can be dropped (if positive), or the minimum interval between dropped frames (if negative). If the value is 0, the frame is dropped unregarding the number of previous sequentially dropped frames.
>
> Default value is 0.

‘`hi, lo, frac`’

> Set the dropping threshold values.
>
> Values for *hi* and *lo* are for 8x8 pixel blocks and represent actual pixel value differences, so a threshold of 64 corresponds to 1 unit of difference for each pixel, or the same spread out differently over the block.
>
> A frame is a candidate for dropping if no 8x8 blocks differ by more than a threshold of *hi*, and if no more than *frac* blocks (1 meaning the whole image) differ by more than a threshold of *lo*.
>
> Default value for *hi* is 64*12, default value for *lo* is 64*5, and default value for *frac* is 0.33.

28.13 delogo

Suppress a TV station logo by a simple interpolation of the surrounding pixels. Just set a rectangle covering the logo and watch it disappear (and sometimes something even uglier appear - your mileage may vary).

The filter accepts parameters as a string of the form "*x*:*y*:*w*:*h*:*band*", or as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`x, y`’

Specify the top left corner coordinates of the logo. They must be specified.

‘`w, h`’

Specify the width and height of the logo to clear. They must be specified.

‘`band, t`’

Specify the thickness of the fuzzy edge of the rectangle (added to *w* and *h*). The default value is 4.

‘`show`’

When set to 1, a green rectangle is drawn on the screen to simplify finding the right *x*, *y*, *w*, *h* parameters, and *band* is set to 4. The default value is 0.

Some examples follow.

- Set a rectangle covering the area with top left corner coordinates 0,0 and size 100x77, setting a band of size 10:

  ```
  delogo=0:0:100:77:10
  ```

- As the previous example, but use named options:

  ```
  delogo=x=0:y=0:w=100:h=77:band=10
  ```

28.14 deshake

Attempt to fix small changes in horizontal and/or vertical shift. This filter helps remove camera shake from hand-holding a camera, bumping a tripod, moving on a vehicle, etc.

The filter accepts parameters as a string of the form "*x*:*y*:*w*:*h*:*rx*:*ry*:*edge*:*blocksize*:*contrast*:*search*:*filename*"

A description of the accepted parameters follows.

‘`x, y, w, h`’

Specify a rectangular area where to limit the search for motion vectors. If desired the search for motion vectors can be limited to a rectangular area of the frame defined by its top left corner, width and height. These parameters have the same meaning as the drawbox filter which can be used to visualise the position of the bounding box.

This is useful when simultaneous movement of subjects within the frame might be confused for camera motion by the motion vector search.

If any or all of *x*, *y*, *w* and *h* are set to -1 then the full frame is used. This allows later options to be set without specifying the bounding box for the motion vector search.

Default - search the whole frame.

‘`rx, ry`’

Specify the maximum extent of movement in x and y directions in the range 0-64 pixels. Default 16.

‘`edge`’

Specify how to generate pixels to fill blanks at the edge of the frame. An integer from 0 to 3 as follows:

‘`0`’

Fill zeroes at blank locations

‘`1`’

Original image at blank locations

‘`2`’

Extruded edge value at blank locations

‘`3`’

Mirrored edge at blank locations

The default setting is mirror edge at blank locations.

‘`blocksize`’

Specify the blocksize to use for motion search. Range 4-128 pixels, default 8.

‘`contrast`’

Specify the contrast threshold for blocks. Only blocks with more than the specified contrast (difference between darkest and lightest pixels) will be considered. Range 1-255, default 125.

‘`search`’

Specify the search strategy 0 = exhaustive search, 1 = less exhaustive search. Default - exhaustive search.

‘`filename`’

If set then a detailed log of the motion search is written to the specified file.

28.15 drawbox

Draw a colored box on the input image.

The filter accepts parameters as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`x, y`’

Specify the top left corner coordinates of the box. Default to 0.

‘`width, w`’
‘`height, h`’

Specify the width and height of the box, if 0 they are interpreted as the input width and height. Default to 0.

‘`color, c`’

Specify the color of the box to write, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence. If the special value `invert` is used, the box edge color is the same as the video with inverted luma.

‘`thickness, t`’

Set the thickness of the box edge. Default value is `4`.

If the key of the first options is omitted, the arguments are interpreted according to the following syntax:

```
drawbox=x:y:width:height:color:thickness
```

Some examples follow:

- Draw a black box around the edge of the input image:

  ```
  drawbox
  ```

- Draw a box with color red and an opacity of 50%:

  ```
  drawbox=10:20:200:60:red@0.5
  ```

The previous example can be specified as:

```
drawbox=x=10:y=20:w=200:h=60:color=red@0.5
```

- Fill the box with pink color:

  ```
  drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max
  ```

28.16 drawtext

Draw text string or text from specified file on top of video using the libfreetype library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libfreetype`.

28.16.1 Syntax

The filter accepts parameters as a list of *key*=*value* pairs, separated by ":".

The description of the accepted parameters follows.

‘`box`’

Used to draw a box around text using background color. Value should be either 1 (enable) or 0 (disable). The default value of *box* is 0.

‘`boxcolor`’

The color to be used for drawing box around text. Either a string (e.g. "yellow") or in 0xRRGGBB[AA] format (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *boxcolor* is "white".

‘`draw`’

Set an expression which specifies if the text should be drawn. If the expression evaluates to 0, the text is not drawn. This is useful for specifying that the text should be drawn only when specific conditions are met.

Default value is "1".

See below for the list of accepted constants and functions.

‘`expansion`’

Select how the *text* is expanded. Can be either `none`, `strftime` (default for compatibity reasons but deprecated) or `normal`. See the Text expansion section below for details.

‘`fix_bounds`’

If true, check and fix text coords to avoid clipping.

‘`fontcolor`’

The color to be used for drawing fonts. Either a string (e.g. "red") or in 0xRRGGBB[AA] format (e.g. "0xff000033"), possibly followed by an alpha specifier. The default value of *fontcolor* is "black".

‘`fontfile`’

The font file to be used for drawing text. Path must be included. This parameter is mandatory.

‘`fontsize`’

The font size to be used for drawing text. The default value of *fontsize* is 16.

‘`ft_load_flags`’

Flags to be used for loading the fonts.

The flags map the corresponding flags supported by libfreetype, and are a combination of the following values:

default
no_scale
no_hinting
render
no_bitmap
vertical_layout
force_autohint
crop_bitmap
pedantic
ignore_global_advance_width
no_recurse
ignore_transform
monochrome
linear_design
no_autohint
end table

Default value is "render".

For more information consult the documentation for the FT_LOAD_* libfreetype flags.

‘`shadowcolor`’

The color to be used for drawing a shadow behind the drawn text. It can be a color name (e.g. "yellow") or a string in the 0xRRGGBB[AA] form (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *shadowcolor* is "black".

‘`shadowx, shadowy`’

The x and y offsets for the text shadow position with respect to the position of the text. They can be either positive or negative values. Default value for both is "0".

‘`tabsize`’

The size in number of spaces to use for rendering the tab. Default value is 4.

‘`timecode`’

Set the initial timecode representation in "hh:mm:ss[:;.]ff" format. It can be used with or without text parameter. *timecode_rate* option must be specified.

‘`timecode_rate, rate, r`’

Set the timecode frame rate (timecode only).

‘`text`’

The text string to be drawn. The text must be a sequence of UTF-8 encoded characters. This parameter is mandatory if no file is specified with the parameter *textfile*.

‘`textfile`’

A text file containing text to be drawn. The text must be a sequence of UTF-8 encoded characters.

This parameter is mandatory if no text string is specified with the parameter *text*.

If both *text* and *textfile* are specified, an error is thrown.

‘`x, y`’

The expressions which specify the offsets where text will be drawn within the video frame. They are relative to the top/left border of the output image.

The default value of *x* and *y* is "0".

See below for the list of accepted constants and functions.

The parameters for *x* and *y* are expressions containing the following constants and functions:

‘`dar`’

input display aspect ratio, it is the same as (*w* / *h*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘`line_h, lh`’

the height of each text line

‘`main_h, h, H`’

the input height

‘`main_w, w, W`’

the input width

‘`max_glyph_a, ascent`’

the maximum distance from the baseline to the highest/upper grid coordinate used to place a glyph outline point, for all the rendered glyphs. It is a positive value, due to the grid’s orientation with the Y axis upwards.

‘`max_glyph_d, descent`’

the maximum distance from the baseline to the lowest grid coordinate used to place a glyph outline point, for all the rendered glyphs. This is a negative value, due to the grid’s orientation, with the Y axis upwards.

‘`max_glyph_h`’

maximum glyph height, that is the maximum height for all the glyphs contained in the rendered text, it is equivalent to *ascent* - *descent*.

‘`max_glyph_w`’

maximum glyph width, that is the maximum width for all the glyphs contained in the rendered text

‘`n`’

the number of input frame, starting from 0

‘`rand(min, max)`’

return a random number included between *min* and *max*

‘`sar`’

input sample aspect ratio

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘`text_h, th`’

the height of the rendered text

‘`text_w, tw`’

the width of the rendered text

‘`x, y`’

the x and y offset coordinates where the text is drawn.

These parameters allow the *x* and *y* expressions to refer each other, so you can for example specify `y=x/dar`.

If libavfilter was built with `--enable-fontconfig`, then ‘`fontfile`’ can be a fontconfig pattern or omitted.

28.16.2 Text expansion

If ‘`expansion`’ is set to `strftime` (which is the default for now), the filter recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime(). This feature is deprecated.

If ‘`expansion`’ is set to `none`, the text is printed verbatim.

If ‘`expansion`’ is set to `normal` (which will be the default), the following expansion mechanism is used.

The backslash character ’\’, followed by any character, always expands to the second character.

Sequence of the form `%{...}` are expanded. The text between the braces is a function name, possibly followed by arguments separated by ’:’. If the arguments contain special characters or delimiters (’:’ or ’}’), they should be escaped.

Note that they probably must also be escaped as the value for the ‘`text`’ option in the filter argument string and as the filter argument in the filter graph description, and possibly also for the shell, that makes up to four levels of escaping; using a text file avoids these problems.

The following functions are available:

`gmtime`

The time at which the filter is running, expressed in UTC. It can accept an argument: a strftime() format string.

`localtime`

The time at which the filter is running, expressed in the local time zone. It can accept an argument: a strftime() format string.

`n, frame_num`

The frame number, starting from 0.

`pts`

The timestamp of the current frame, in seconds, with microsecond accuracy.

28.16.3 Examples

Some examples follow.

- Draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
```

- Draw 'Test Text' with font FreeSerif of size 24 at position x=100 and y=50 (counting from the top-left corner of the screen), text is yellow with a red box around it. Both the text and the box have an opacity of 20%.

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
          x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
```

  Note that the double quotes are not necessary if spaces are not used within the parameter list.

- Show the text at the center of the video frame:

```
drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
```

- Show a text line sliding from right to left in the last row of the video frame. The file '`LONG_LINE`' is assumed to contain a single line with no newlines.

```
drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
```

- Show the content of file ‘CREDITS’ off the bottom of the frame and scroll up.

  ```
  drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
  ```

- Draw a single green letter "g", at the center of the input video. The glyph baseline is placed at half screen height.

  ```
  drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=g:x=(w-max_glyph_w)/2:y=h/2-ascent"
  ```

- Show text for 1 second every 3 seconds:

  ```
  drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:draw=lt(mod(t\,3)\,1):text='blink'"
  ```

- Use fontconfig to set the font. Note that the colons need to be escaped.

  ```
  drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
  ```

- Print the date of a real-time encoding (see strftime(3)):

  ```
  drawtext='fontfile=FreeSans.ttf:expansion=normal:text=%{localtime:%a %b %d %Y}'
  ```

For more information about libfreetype, check: http://www.freetype.org/.

For more information about fontconfig, check: http://freedesktop.org/software/fontconfig/fontconfig-user.html.

28.17 edgedetect

Detect and draw edges. The filter uses the Canny Edge Detection algorithm.

This filter accepts the following optional named parameters:

‘`low, high`’

> Set low and high threshold values used by the Canny thresholding algorithm.
>
> The high threshold selects the "strong" edge pixels, which are then connected through 8-connectivity with the "weak" edge pixels selected by the low threshold.
>
> *low* and *high* threshold values must be choosen in the range [0,1], and *low* should be lesser or equal to *high*.
>
> Default value for *low* is `20/255`, and default value for *high* is `50/255`.

Example:

```
edgedetect=low=0.1:high=0.4
```

28.18 fade

Apply fade-in/out effect to input video.

It accepts the parameters: *type*:*start_frame*:*nb_frames*[:*options*]

type specifies if the effect type, can be either "in" for fade-in, or "out" for a fade-out effect.

start_frame specifies the number of the start frame for starting to apply the fade effect.

nb_frames specifies the number of frames for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black.

options is an optional sequence of *key*=*value* pairs, separated by ":". The description of the accepted options follows.

‘`type, t`’

See *type*.

‘`start_frame, s`’

See *start_frame*.

‘`nb_frames, n`’

See *nb_frames*.

‘`alpha`’

If set to 1, fade only alpha channel, if one exists on the input. Default value is 0.

A few usage examples follow, usable too as test scenarios.

```
# fade in first 30 frames of video
fade=in:0:30

# fade out last 45 frames of a 200-frame video
fade=out:155:45

# fade in first 25 frames and fade out last 25 frames of a 1000-frame video
fade=in:0:25, fade=out:975:25

# make first 5 frames black, then fade in from frame 5-24
fade=in:5:20

# fade in alpha over first 25 frames of video
fade=in:0:25:alpha=1
```

28.19 field

Extract a single field from an interlaced image using stride arithmetic to avoid wasting CPU time. The output frames are marked as non-interlaced.

This filter accepts the following named options:

‘`type`’

> Specify whether to extract the top (if the value is `0` or `top`) or the bottom field (if the value is `1` or `bottom`).

If the option key is not specified, the first value sets the *type* option. For example:

```
field=bottom
```

is equivalent to:

```
field=type=bottom
```

28.20 fieldorder

Transform the field order of the input video.

It accepts one parameter which specifies the required field order that the input interlaced video will be transformed to. The parameter can assume one of the following values:

‘`0 or bff`’

> output bottom field first

‘`1 or tff`’

output top field first

Default value is "tff".

Transformation is achieved by shifting the picture content up or down by one line, and filling the remaining line with appropriate picture content. This method is consistent with most broadcast field order converters.

If the input video is not flagged as being interlaced, or it is already flagged as being of the required output field order then this filter does not alter the incoming video.

This filter is very useful when converting to or from PAL DV material, which is bottom field first.

For example:

```
ffmpeg -i in.vob -vf "fieldorder=bff" out.dv
```

28.21 fifo

Buffer input images and send them when they are requested.

This filter is mainly useful when auto-inserted by the libavfilter framework.

The filter does not take parameters.

28.22 format

Convert the input video to one of the specified pixel formats. Libavfilter will try to pick one that is supported for the input to the next filter.

The filter accepts a list of pixel format names, separated by ":", for example "yuv420p:monow:rgb24".

Some examples follow:

```
# convert the input video to the format "yuv420p"
format=yuv420p

# convert the input video to any of the formats in the list
format=yuv420p:yuv444p:yuv410p
```

28.23 fps

Convert the video to specified constant framerate by duplicating or dropping frames as necessary.

This filter accepts the following named parameters:

‘`fps`’

Desired output framerate.

‘`round`’

Rounding method. The default is `near`.

28.24 framestep

Select one frame every N.

This filter accepts in input a string representing a positive integer. Default argument is `1`.

28.25 frei0r

Apply a frei0r effect to the input video.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

The filter supports the syntax:

```
filter_name[{:|=}param1:param2:...:paramN]
```

filter_name is the name of the frei0r effect to load. If the environment variable `FREI0R_PATH` is defined, the frei0r effect is searched in each one of the directories specified by the colon (or semicolon on Windows platforms) separated list in `FREIOR_PATH`, otherwise in the standard frei0r paths, which are in this order: ‘`HOME/.frei0r-1/lib/`’, ‘`/usr/local/lib/frei0r-1/`’, ‘`/usr/lib/frei0r-1/`’.

param1, *param2*, ... , *paramN* specify the parameters for the frei0r effect.

A frei0r effect parameter can be a boolean (whose values are specified with "y" and "n"), a double, a color (specified by the syntax *R*/*G*/*B*, *R*, *G*, and *B* being float numbers from 0.0 to 1.0) or by an `av_parse_color()` color description), a position (specified by the syntax *X*/*Y*, *X* and *Y* being float numbers) and a string.

The number and kind of parameters depend on the loaded effect. If an effect parameter is not specified the default value is set.

Some examples follow:

- Apply the distort0r effect, set the first two double parameters:

  ```
  frei0r=distort0r:0.5:0.01
  ```

- Apply the colordistance effect, take a color as first parameter:

  ```
  frei0r=colordistance:0.2/0.3/0.4
  frei0r=colordistance:violet
  frei0r=colordistance:0x112233
  ```

- Apply the perspective effect, specify the top left and top right image positions:

  ```
  frei0r=perspective:0.2/0.2:0.8/0.2
  ```

For more information see: http://frei0r.dyne.org

28.26 geq

The filter takes one, two or three equations as parameter, separated by ':'. The first equation is mandatory and applies to the luma plane. The two following are respectively for chroma blue and chroma red planes.

The filter syntax allows named parameters:

'`lum_expr`'

  the luminance expression

'`cb_expr`'

  the chrominance blue expression

'`cr_expr`'

  the chrominance red expression

If one of the chrominance expression is not defined, it falls back on the other one. If none of them are specified, they will evaluate the luminance expression.

The expressions can use the following variables and functions:

‘`N`’

The sequential number of the filtered frame, starting from `0`.

‘`X, Y`’

The coordinates of the current sample.

‘`W, H`’

The width and height of the image.

‘`SW, SH`’

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

‘`p(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the current plane.

‘`lum(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the luminance plane.

‘`cb(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the blue-difference chroma plane.

‘`cr(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the red-difference chroma plane.

For functions, if *x* and *y* are outside the area, the value will be automatically clipped to the closer edge.

Some examples follow:

- Flip the image horizontally:

```
geq=p(W-X\,Y)
```

- Generate a fancy enigmatic moving light:

```
nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N*0.07)*W/2-W/2\,Y-sin(N*0.09)*H/2-H/2)^2*1000000*sin(N*0.02):128:128
```

28.27 gradfun

Fix the banding artifacts that are sometimes introduced into nearly flat regions by truncation to 8bit color depth. Interpolate the gradients that should go where the bands are, and dither them.

This filter is designed for playback only. Do not use it prior to lossy compression, because compression tends to lose the dither and bring back the bands.

The filter takes two optional parameters, separated by ':': *strength*:*radius*

strength is the maximum amount by which the filter will change any one pixel. Also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 255, default value is 1.2, out-of-range values will be clipped to the valid range.

radius is the neighborhood to fit the gradient to. A larger radius makes for smoother gradients, but also prevents the filter from modifying the pixels near detailed regions. Acceptable values are 8-32, default value is 16, out-of-range values will be clipped to the valid range.

```
# default parameters
gradfun=1.2:16

# omitting radius
gradfun=1.2
```

28.28 hflip

Flip the input video horizontally.

For example to horizontally flip the input video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "hflip" out.avi
```

28.29 hqdn3d

High precision/quality 3d denoise filter. This filter aims to reduce image noise producing smooth images and making still images really still. It should enhance compressibility.

It accepts the following optional parameters: *luma_spatial*:*chroma_spatial*:*luma_tmp*:*chroma_tmp*

'`luma_spatial`'

a non-negative float number which specifies spatial luma strength, defaults to 4.0

'`chroma_spatial`'

a non-negative float number which specifies spatial chroma strength, defaults to 3.0*_luma_spatial_/4.0

‘`luma_tmp`’

a float number which specifies luma temporal strength, defaults to 6.0*_luma_spatial_/4.0

‘`chroma_tmp`’

a float number which specifies chroma temporal strength, defaults to _luma_tmp_*_chroma_spatial_/_luma_spatial_

28.30 hue

Modify the hue and/or the saturation of the input.

This filter accepts the following optional named options:

‘`h`’

Specify the hue angle as a number of degrees. It accepts a float number or an expression, and defaults to 0.0.

‘`H`’

Specify the hue angle as a number of degrees. It accepts a float number or an expression, and defaults to 0.0.

‘`s`’

Specify the saturation in the [-10,10] range. It accepts a float number and defaults to 1.0.

The _h_, _H_ and _s_ parameters are expressions containing the following constants:

‘`n`’

frame count of the input frame starting from 0

‘`pts`’

presentation timestamp of the input frame expressed in time base units

‘`r`’

frame rate of the input video, NAN if the input frame rate is unknown

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘`tb`’

time base of the input video

The options can also be set using the syntax: *hue*:*saturation*

In this case *hue* is expressed in degrees.

Some examples follow:

- Set the hue to 90 degrees and the saturation to 1.0:

  ```
  hue=h=90:s=1
  ```

- Same command but expressing the hue in radians:

  ```
  hue=H=PI/2:s=1
  ```

- Same command without named options, hue must be expressed in degrees:

  ```
  hue=90:1
  ```

- Note that "h:s" syntax does not support expressions for the values of h and s, so the following example will issue an error:

  ```
  hue=PI/2:1
  ```

- Rotate hue and make the saturation swing between 0 and 2 over a period of 1 second:

  ```
  hue="H=2*PI*t: s=sin(2*PI*t)+1"
  ```

- Apply a 3 seconds saturation fade-in effect starting at 0:

  ```
  hue="s=min(t/3\,1)"
  ```

  The general fade-in expression can be written as:

  ```
  hue="s=min(0\, max((t-START)/DURATION\, 1))"
  ```

- Apply a 3 seconds saturation fade-out effect starting at 5 seconds:

```
hue="s=max(0\, min(1\, (8-t)/3))"
```

The general fade-out expression can be written as:

```
hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
```

28.30.1 Commands

This filter supports the following command:

‘`reinit`’

Modify the hue and/or the saturation of the input video. The command accepts the same named options and syntax than when calling the filter from the command-line.

If a parameter is omitted, it is kept at its current value.

28.31 idet

Interlaceing detect filter. This filter tries to detect if the input is interlaced or progressive. Top or bottom field first.

28.32 lut, lutrgb, lutyuv

Compute a look-up table for binding each pixel component input value to an output value, and apply it to input video.

lutyuv applies a lookup table to a YUV input video, *lutrgb* to an RGB input video.

These filters accept in input a ":"-separated list of options, which specify the expressions used for computing the lookup table for the corresponding pixel component values.

The *lut* filter requires either YUV or RGB pixel formats in input, and accepts the options:

‘*`c0`* `(first pixel component)`’
‘*`c1`* `(second pixel component)`’
‘*`c2`* `(third pixel component)`’
‘*`c3`* `(fourth pixel component, corresponds to the alpha component)`’

The exact component associated to each option depends on the format in input.

The *lutrgb* filter requires RGB pixel formats in input, and accepts the options:

‘*`r`* `(red component)`’

‘`*g* (green component)`’
‘`*b* (blue component)`’
‘`*a* (alpha component)`’

The *lutyuv* filter requires YUV pixel formats in input, and accepts the options:

‘`*y* (Y/luminance component)`’
‘`*u* (U/Cb component)`’
‘`*v* (V/Cr component)`’
‘`*a* (alpha component)`’

The expressions can contain the following constants and functions:

‘`w, h`’

: the input width and height

‘`val`’

: input value for the pixel component

‘`clipval`’

: the input value clipped in the *minval*-*maxval* range

‘`maxval`’

: maximum value for the pixel component

‘`minval`’

: minimum value for the pixel component

‘`negval`’

: the negated value for the pixel component value clipped in the *minval*-*maxval* range , it corresponds to the expression "maxval-clipval+minval"

‘`clip(val)`’

: the computed value in *val* clipped in the *minval*-*maxval* range

‘`gammaval(gamma)`’

: the computed gamma correction value of the pixel component value clipped in the *minval*-*maxval* range, corresponds to the expression "pow((clipval-minval)/(maxval-minval)\,*gamma*)*(maxval-minval)+minval"

All expressions default to "val".

Some examples follow:

```
# negate input video
lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val"
lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"

# the above is the same as
lutrgb="r=negval:g=negval:b=negval"
lutyuv="y=negval:u=negval:v=negval"

# negate luminance
lutyuv=y=negval

# remove chroma components, turns the video into a graytone image
lutyuv="u=128:v=128"

# apply a luma burning effect
lutyuv="y=2*val"

# remove green and blue components
lutrgb="g=0:b=0"

# set a constant alpha channel value on input
format=rgba,lutrgb=a="maxval-minval/2"

# correct luminance gamma by a 0.5 factor
lutyuv=y=gammaval(0.5)
```

28.33 mp

Apply an MPlayer filter to the input video.

This filter provides a wrapper around most of the filters of MPlayer/MEncoder.

This wrapper is considered experimental. Some of the wrapped filters may not work properly and we may drop support for them, as they will be implemented natively into FFmpeg. Thus you should avoid depending on them when writing portable scripts.

The filters accepts the parameters: *filter_name*[:=]*filter_params*

filter_name is the name of a supported MPlayer filter, *filter_params* is a string containing the parameters accepted by the named filter.

The list of the currently supported filters follows:

denoise3d
detc

dint
divtc
down3dright
dsize
eq2
eq
fil
fspp
harddup
il
ilpack
ivtc
kerndeint
mcdeint
noise
ow
perspective
phase
pp7
pullup
qp
sab
softpulldown
softskip
spp
telecine
tinterlace
unsharp
uspp

The parameter syntax and behavior for the listed filters are the same of the corresponding MPlayer filters. For detailed instructions check the "VIDEO FILTERS" section in the MPlayer manual.

Some examples follow:

- Adjust gamma, brightness, contrast:

  ```
  mp=eq2=1.0:2:0.5
  ```

- Add temporal noise to input video:

  ```
  mp=noise=20t
  ```

See also mplayer(1), http://www.mplayerhq.hu/.

28.34 negate

Negate input video.

This filter accepts an integer in input, if non-zero it negates the alpha component (if available). The default value in input is 0.

28.35 noformat

Force libavfilter not to use any of the specified pixel formats for the input to the next filter.

The filter accepts a list of pixel format names, separated by ":", for example "yuv420p:monow:rgb24".

Some examples follow:

```
# force libavfilter to use a format different from "yuv420p" for the
# input to the vflip filter
noformat=yuv420p,vflip

# convert the input video to any of the formats not contained in the list
noformat=yuv420p:yuv444p:yuv410p
```

28.36 null

Pass the video source unchanged to the output.

28.37 ocv

Apply video transform using libopencv.

To enable this filter install libopencv library and headers and configure FFmpeg with `--enable-libopencv`.

The filter takes the parameters: *filter_name*{:=}*filter_params*.

filter_name is the name of the libopencv filter to apply.

filter_params specifies the parameters to pass to the libopencv filter. If not specified the default values are assumed.

Refer to the official libopencv documentation for more precise information:
http://opencv.willowgarage.com/documentation/c/image_filtering.html

Follows the list of supported libopencv filters.

28.37.1 dilate

Dilate an image by using a specific structuring element. This filter corresponds to the libopencv function `cvDilate`.

It accepts the parameters: *struct_el*:*nb_iterations*.

struct_el represents a structuring element, and has the syntax: *cols*x*rows*+*anchor_x*x*anchor_y*/*shape*

cols and *rows* represent the number of columns and rows of the structuring element, *anchor_x* and *anchor_y* the anchor point, and *shape* the shape for the structuring element, and can be one of the values "rect", "cross", "ellipse", "custom".

If the value for *shape* is "custom", it must be followed by a string of the form "=*filename*". The file with name *filename* is assumed to represent a binary image, with each printable character corresponding to a bright pixel. When a custom *shape* is used, *cols* and *rows* are ignored, the number or columns and rows of the read file are assumed instead.

The default value for *struct_el* is "3x3+0x0/rect".

nb_iterations specifies the number of times the transform is applied to the image, and defaults to 1.

Follow some example:

```
# use the default values
ocv=dilate

# dilate using a structuring element with a 5x5 cross, iterate two times
ocv=dilate=5x5+2x2/cross:2

# read the shape from the file diamond.shape, iterate two times
# the file diamond.shape may contain a pattern of characters like this:
#   *
#  ***
# *****
#  ***
#   *
# the specified cols and rows are ignored (but not the anchor point coordinates)
ocv=0x0+2x2/custom=diamond.shape:2
```

28.37.2 erode

Erode an image by using a specific structuring element. This filter corresponds to the libopencv function `cvErode`.

The filter accepts the parameters: *struct_el*:*nb_iterations*, with the same syntax and semantics as the dilate filter.

28.37.3 smooth

Smooth the input video.

The filter takes the following parameters: *type*:*param1*:*param2*:*param3*:*param4*.

type is the type of smooth filter to apply, and can be one of the following values: "blur", "blur_no_scale", "median", "gaussian", "bilateral". The default value is "gaussian".

param1, *param2*, *param3*, and *param4* are parameters whose meanings depend on smooth type. *param1* and *param2* accept integer positive values or 0, *param3* and *param4* accept float values.

The default value for *param1* is 3, the default value for the other parameters is 0.

These parameters correspond to the parameters assigned to the libopencv function `cvSmooth`.

28.38 overlay

Overlay one video on top of another.

It takes two inputs and one output, the first input is the "main" video on which the second input is overlayed.

It accepts the parameters: *x*:*y*[:*options*].

x is the x coordinate of the overlayed video on the main video, *y* is the y coordinate. *x* and *y* are expressions containing the following parameters:

‘`main_w, main_h`’

    main input width and height

‘`W, H`’

    same as *main_w* and *main_h*

‘`overlay_w, overlay_h`’

    overlay input width and height

‘`w, h`’

    same as *overlay_w* and *overlay_h*

options is an optional list of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`rgb`’

If set to 1, force the filter to accept inputs in the RGB color space. Default value is 0.

Be aware that frames are taken from each input video in timestamp order, hence, if their initial timestamps differ, it is a a good idea to pass the two inputs through a *setpts=PTS-STARTPTS* filter to have them begin in the same zero timestamp, as it does the example for the *movie* filter.

Follow some examples:

```
# draw the overlay at 10 pixels from the bottom right
# corner of the main video.
overlay=main_w-overlay_w-10:main_h-overlay_h-10

# insert a transparent PNG logo in the bottom left corner of the input
ffmpeg -i input -i logo -filter_complex 'overlay=10:main_h-overlay_h-10' output

# insert 2 different transparent PNG logos (second logo on bottom
# right corner):
ffmpeg -i input -i logo1 -i logo2 -filter_complex
'overlay=10:H-h-10,overlay=W-w-10:H-h-10' output

# add a transparent color layer on top of the main video,
# WxH specifies the size of the main input to the overlay filter
color=red@.3:WxH [over]; [in][over] overlay [out]

# play an original video and a filtered version (here with the deshake filter)
# side by side
ffplay input.avi -vf 'split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w'

# the previous example is the same as:
ffplay input.avi -vf 'split[b], pad=iw*2[src], [b]deshake, [src]overlay=w'
```

You can chain together more overlays but the efficiency of such approach is yet to be tested.

28.39 pad

Add paddings to the input image, and places the original input at the given coordinates *x*, *y*.

It accepts the following parameters: *width*:*height*:*x*:*y*:*color*.

The parameters *width*, *height*, *x*, and *y* are expressions containing the following constants:

‘`in_w, in_h`’

the input video width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output width and height, that is the size of the padded area as specified by the *width* and *height* expressions

‘`ow, oh`’

same as *out_w* and *out_h*

‘`x, y`’

x and y offsets as specified by the *x* and *y* expressions, or NAN if not yet specified

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

Follows the description of the accepted parameters.

‘`width, height`’

Specify the size of the output image with the paddings added. If the value for *width* or *height* is 0, the corresponding input size is used for the output.

The *width* expression can reference the value set by the *height* expression, and vice versa.

The default value of *width* and *height* is 0.

‘`x, y`’

Specify the offsets where to place the input image in the padded area with respect to the top/left border of the output image.

The *x* expression can reference the value set by the *y* expression, and vice versa.

The default value of *x* and *y* is 0.

‘`color`’

Specify the color of the padded area, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence.

The default value of *color* is "black".

28.39.1 Examples

- Add paddings with color "violet" to the input video. Output video size is 640x480, the top-left corner of the input video is placed at column 0, row 40:

  ```
  pad=640:480:0:40:violet
  ```

- Pad the input to get an output with dimensions increased by 3/2, and put the input video at the center of the padded area:

  ```
  pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a squared output with size equal to the maximum value between the input width and height, and put the input video at the center of the padded area:

  ```
  pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a final w/h ratio of 16:9:

  ```
  pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- In case of anamorphic video, in order to set the output display aspect correctly, it is necessary to use *sar* in the expression, according to the relation:

  ```
  (ih * X / ih) * sar = output_dar
  X = output_dar / sar
  ```

  Thus the previous example needs to be modified to:

  ```
  pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Double output size and put the input video in the bottom-right corner of the output padded area:

  ```
  pad="2*iw:2*ih:ow-iw:oh-ih"
  ```

28.40 pixdesctest

Pixel format descriptor test filter, mainly useful for internal testing. The output video should be equal to the input video.

For example:

```
format=monow, pixdesctest
```

can be used to test the monowhite pixel format descriptor definition.

28.41 removelogo

Suppress a TV station logo, using an image file to determine which pixels comprise the logo. It works by filling in the pixels that comprise the logo with neighboring pixels.

This filter requires one argument which specifies the filter bitmap file, which can be any image format supported by libavformat. The width and height of the image file must match those of the video stream being processed.

Pixels in the provided bitmap image with a value of zero are not considered part of the logo, non-zero pixels are considered part of the logo. If you use white (255) for the logo and black (0) for the rest, you will be safe. For making the filter bitmap, it is recommended to take a screen capture of a black frame with the logo visible, and then using a threshold filter followed by the erode filter once or twice.

If needed, little splotches can be fixed manually. Remember that if logo pixels are not covered, the filter quality will be much reduced. Marking too many pixels as part of the logo does not hurt as much, but it will increase the amount of blurring needed to cover over the image and will destroy more information than necessary, and extra pixels will slow things down on a large logo.

28.42 scale

Scale (resize) the input video, using the libswscale library.

The scale filter forces the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

This filter accepts a list of named options in the form of *key*=*value* pairs separated by ":". If the key for the first two options is not specified, the assumed keys for the first two values are `w` and `h`. If the first option has no key and can be interpreted like a video size specification, it will be used to set the video size.

A description of the accepted options follows.

‘`width, w`’

Set the video width expression, default value is `iw`. See below for the list of accepted constants.

‘`height, h`’

Set the video heiht expression, default value is `ih`. See below for the list of accepted constants.

‘`interl`’

Set the interlacing. It accepts the following values:

‘`1`’

force interlaced aware scaling

‘`0`’

do not apply interlaced scaling

‘`-1`’

select interlaced aware scaling depending on whether the source frames are flagged as interlaced or not

Default value is `0`.

‘`flags`’

Set libswscale scaling flags. If not explictly specified the filter applies a bilinear scaling algorithm.

‘`size, s`’

Set the video size, the value must be a valid abbreviation or in the form *width*x*height*.

The values of the *w* and *h* options are expressions containing the following constants:

‘`in_w, in_h`’

the input width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

If the input image format is different from the format requested by the next filter, the scale filter will convert the input to the requested format.

If the value for *width* or *height* is 0, the respective input size is used for the output.

If the value for *width* or *height* is -1, the scale filter will use, for the respective output size, a value that maintains the aspect ratio of the input image.

28.42.1 Examples

- Scale the input video to a size of 200x100:

  ```
  scale=200:100
  ```

  This is equivalent to:

  ```
  scale=w=200:h=100
  ```

  or:

  ```
  scale=200x100
  ```

- Specify a size abbreviation for the output size:

  ```
  scale=qcif
  ```

which can also be written as:

```
scale=size=qcif
```

- Scale the input to 2x:

```
scale=2*iw:2*ih
```

- The above is the same as:

```
scale=2*in_w:2*in_h
```

- Scale the input to 2x with forced interlaced scaling:

```
scale=2*iw:2*ih:interl=1
```

- Scale the input to half size:

```
scale=iw/2:ih/2
```

- Increase the width, and set the height to the same size:

```
scale=3/2*iw:ow
```

- Seek for Greek harmony:

```
scale=iw:1/PHI*iw
scale=ih*PHI:ih
```

- Increase the height, and set the width to 3/2 of the height:

```
scale=3/2*oh:3/5*ih
```

- Increase the size, but make the size a multiple of the chroma:

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- Increase the width to a maximum of 500 pixels, keep the same input aspect ratio:

```
scale='min(500\, iw*3/2):-1'
```

28.43 select

Select frames to pass in output.

It accepts in input an expression, which is evaluated for each input frame. If the expression is evaluated to a non-zero value, the frame is selected and passed to the output, otherwise it is discarded.

The expression can contain the following constants:

‘`n`’

: the sequential number of the filtered frame, starting from 0

‘`selected_n`’

: the sequential number of the selected frame, starting from 0

‘`prev_selected_n`’

: the sequential number of the last selected frame, NAN if undefined

‘`TB`’

: timebase of the input timestamps

‘`pts`’

: the PTS (Presentation TimeStamp) of the filtered video frame, expressed in *TB* units, NAN if undefined

‘`t`’

: the PTS (Presentation TimeStamp) of the filtered video frame, expressed in seconds, NAN if undefined

‘`prev_pts`’

: the PTS of the previously filtered video frame, NAN if undefined

‘`prev_selected_pts`’

: the PTS of the last previously filtered video frame, NAN if undefined

‘`prev_selected_t`’

: the PTS of the last previously selected video frame, NAN if undefined

‘`start_pts`’

the PTS of the first video frame in the video, NAN if undefined

‘`start_t`’

the time of the first video frame in the video, NAN if undefined

‘`pict_type`’

the type of the filtered frame, can assume one of the following values:

‘`I`’
‘`P`’
‘`B`’
‘`S`’
‘`SI`’
‘`SP`’
‘`BI`’

‘`interlace_type`’

the frame interlace type, can assume one of the following values:

‘`PROGRESSIVE`’

the frame is progressive (not interlaced)

‘`TOPFIRST`’

the frame is top-field-first

‘`BOTTOMFIRST`’

the frame is bottom-field-first

‘`key`’

1 if the filtered frame is a key-frame, 0 otherwise

‘`pos`’

the position in the file of the filtered frame, -1 if the information is not available (e.g. for synthetic video)

‘`scene`’

value between 0 and 1 to indicate a new scene; a low value reflects a low probability for the current frame to introduce a new scene, while a higher value means the current frame is more likely to be one (see the example below)

The default value of the select expression is "1".

Some examples follow:

```
# select all frames in input
select

# the above is the same as:
select=1

# skip all frames:
select=0

# select only I-frames
select='eq(pict_type\,I)'

# select one frame every 100
select='not(mod(n\,100))'

# select only frames contained in the 10-20 time interval
select='gte(t\,10)*lte(t\,20)'

# select only I frames contained in the 10-20 time interval
select='gte(t\,10)*lte(t\,20)*eq(pict_type\,I)'

# select frames with a minimum distance of 10 seconds
select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
```

Complete example to create a mosaic of the first scenes:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

Comparing *scene* against a value between 0.3 and 0.5 is generally a sane choice.

28.44 setdar, setsar

The `setdar` filter sets the Display Aspect Ratio for the filter output video.

This is done by changing the specified Sample (aka Pixel) Aspect Ratio, according to the following equation:

```
DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR
```

Keep in mind that the `setdar` filter does not modify the pixel dimensions of the video frame. Also the display aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. in case of scaling or if another "setdar" or a "setsar" filter is applied.

The `setsar` filter sets the Sample (aka Pixel) Aspect Ratio for the filter output video.

Note that as a consequence of the application of this filter, the output display aspect ratio will change according to the equation above.

Keep in mind that the sample aspect ratio set by the `setsar` filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

The `setdar` and `setsar` filters accept a string in the form *num*:*den* expressing an aspect ratio, or the following named options, expressed as a sequence of *key*=*value* pairs, separated by ":".

‘`max`’

Set the maximum integer value to use for expressing numerator and denominator when reducing the expressed aspect ratio to a rational. Default value is `100`.

‘`r, ratio:`’

Set the aspect ratio used by the filter.

The parameter can be a floating point number string, an expression, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0". In case the form "*num*:*den*" the `:` character should be escaped.

If the keys are omitted in the named options list, the specifed values are assumed to be *ratio* and *max* in that order.

For example to change the display aspect ratio to 16:9, specify:

```
setdar='16:9'
```

The example above is equivalent to:

```
setdar=1.77777
```

To change the sample aspect ratio to 10:11, specify:

```
setsar='10:11'
```

To set a display aspect ratio of 16:9, and specify a maximum integer value of 1000 in the aspect ratio reduction, use the command:

```
setdar=ratio='16:9':max=1000
```

28.45 setfield

Force field for the output video frame.

The `setfield` filter marks the interlace type field for the output frames. It does not change the input frame, but only sets the corresponding property, which affects how the frame is treated by following filters (e.g. `fieldorder` or `yadif`).

It accepts a string parameter, which can assume the following values:

‘`auto`’

Keep the same field property.

‘`bff`’

Mark the frame as bottom-field-first.

‘`tff`’

Mark the frame as top-field-first.

‘`prog`’

Mark the frame as progressive.

28.46 showinfo

Show a line containing various information for each input video frame. The input video is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

A description of each shown parameter follows:

‘`n`’

sequential number of the input frame, starting from 0

‘`pts`’

Presentation TimeStamp of the input frame, expressed as a number of time base units. The time base unit depends on the filter input pad.

‘`pts_time`’

Presentation TimeStamp of the input frame, expressed as a number of seconds

‘`pos`’

position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic video)

‘`fmt`’

pixel format name

‘`sar`’

sample aspect ratio of the input frame, expressed in the form *num*/*den*

‘`s`’

size of the input frame, expressed in the form *width*x*height*

‘`i`’

interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first)

‘`iskey`’

1 if the frame is a key frame, 0 otherwise

‘`type`’

picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, "?" for unknown type). Check also the documentation of the `AVPictureType` enum and of the `av_get_picture_type_char` function defined in ‘`libavutil/avutil.h`’.

‘`checksum`’

Adler-32 checksum (printed in hexadecimal) of all the planes of the input frame

‘`plane_checksum`’

Adler-32 checksum (printed in hexadecimal) of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]"

28.47 slicify

Pass the images of input video on to next video filter as multiple slices.

```
ffmpeg -i in.avi -vf "slicify=32" out.avi
```

The filter accepts the slice height as parameter. If the parameter is not specified it will use the default value of 16.

Adding this in the beginning of filter chains should make filtering faster due to better use of the memory cache.

28.48 smartblur

Blur the input video without impacting the outlines.

The filter accepts the following parameters:
luma_radius:*luma_strength*:*luma_threshold*[:*chroma_radius*:*chroma_strength*:*chroma_threshold*]

Parameters prefixed by *luma* indicate that they work on the luminance of the pixels whereas parameters prefixed by *chroma* refer to the chrominance of the pixels.

If the chroma parameters are not set, the luma parameters are used for either the luminance and the chrominance of the pixels.

luma_radius or *chroma_radius* must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger).

luma_strength or *chroma_strength* must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image.

luma_threshold or *chroma_threshold* must be an integer in the range [-30,30] that is used as a coefficient to determine whether a pixel should be blurred or not. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges.

28.49 split

Split input video into several identical outputs.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

For example

```
ffmpeg -i INPUT -filter_complex split=5 OUTPUT
```

will create 5 copies of the input video.

For example:

```
[in] split [splitout1][splitout2];
[splitout1] crop=100:100:0:0    [cropout];
[splitout2] pad=200:200:100:100 [padout];
```

will create two separate outputs from the same input, one cropped and one padded.

28.50 super2xsai

Scale the input by 2x and smooth using the Super2xSaI (Scale and Interpolate) pixel art scaling algorithm.

Useful for enlarging pixel art images without reducing sharpness.

28.51 swapuv

Swap U & V plane.

28.52 thumbnail

Select the most representative frame in a given sequence of consecutive frames.

It accepts as argument the frames batch size to analyze (default *N*=100); in a set of *N* frames, the filter will pick one of them, and then handle the next batch of *N* frames until the end.

Since the filter keeps track of the whole frames sequence, a bigger *N* value will result in a higher memory usage, so a high value is not recommended.

The following example extract one picture each 50 frames:

```
thumbnail=50
```

Complete example of a thumbnail creation with `ffmpeg`:

```
ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
```

28.53 tile

Tile several successive frames together.

It accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`layout`’

Set the grid size (i.e. the number of lines and columns) in the form "*w*x*h*".

‘`margin`’

Set the outer border margin in pixels.

‘`padding`’

Set the inner border thickness (i.e. the number of pixels between frames). For more advanced padding options (such as having different values for the edges), refer to the pad video filter.

‘`nb_frames`’

Set the maximum number of frames to render in the given area. It must be less than or equal to *w*x*h*. The default value is `0`, meaning all the area will be used.

Alternatively, the options can be specified as a flat string:

layout[:*nb_frames*[:*margin*[:*padding*]]]

For example, produce 8Ã8 PNG tiles of all keyframes (‘`-skip_frame nokey`’) in a movie:

```
ffmpeg -skip_frame nokey -i file.avi -vf ’scale=128:72,tile=8x8’ -an -vsync 0 keyframes%03d.png
```

The ‘`-vsync 0`’ is necessary to prevent `ffmpeg` from duplicating each output frame to accomodate the originally detected frame rate.

Another example to display `5` pictures in an area of `3x2` frames, with `7` pixels between them, and `2` pixels of initial margin, using mixed flat and named options:

```
tile=3x2:nb_frames=5:padding=7:margin=2
```

28.54 tinterlace

Perform various types of temporal field interlacing.

Frames are counted starting from 1, so the first input frame is considered odd.

This filter accepts a single parameter specifying the mode. Available modes are:

‘`merge, 0`’

Move odd frames into the upper field, even into the lower field, generating a double height frame at half framerate.

‘`drop_odd, 1`’

Only output even frames, odd frames are dropped, generating a frame with unchanged height at half framerate.

‘`drop_even, 2`’

Only output odd frames, even frames are dropped, generating a frame with unchanged height at half framerate.

‘`pad, 3`’

Expand each frame to full height, but pad alternate lines with black, generating a frame with double height at the same input framerate.

‘`interleave_top, 4`’

Interleave the upper field from odd frames with the lower field from even frames, generating a frame with unchanged height at half framerate.

‘`interleave_bottom, 5`’

Interleave the lower field from odd frames with the upper field from even frames, generating a frame with unchanged height at half framerate.

‘`interlacex2, 6`’

Double frame rate with unchanged height. Frames are inserted each containing the second temporal field from the previous input frame and the first temporal field from the next input frame. This mode relies on the top_field_first flag. Useful for interlaced video displays with no field synchronisation.

Numeric values are deprecated but are accepted for backward compatibility reasons.

Default mode is `merge`.

28.55 transpose

Transpose rows with columns in the input video and optionally flip it.

This filter accepts the following named parameters:

‘`dir`’

Specify the transposition direction. Can assume the following values:

‘`0, 4`’

Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

‘1, 5’

Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

‘2, 6’

Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

‘3, 7’

Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

For values between 4-7, the transposition is only done if the input video geometry is portrait and not landscape. These values are deprecated, the `passthrough` option should be used instead.

‘passthrough’

Do not apply the transposition if the input geometry matches the one specified by the specified value. It accepts the following values:

‘none’

Always apply transposition.

‘portrait’

Preserve portrait geometry (when *height* >= *width*).

‘landscape’

Preserve landscape geometry (when *width* >= *height*).

Default value is `none`.

28.56 unsharp

Sharpen or blur the input video.

It accepts the following parameters:
luma_msize_x:*luma_msize_y*:*luma_amount*:*chroma_msize_x*:*chroma_msize_y*:*chroma_amount*

Negative values for the amount will blur the input video, while positive values will sharpen. All parameters are optional and default to the equivalent of the string ’5:5:1.0:5:5:0.0’.

‘`luma_msize_x`’

Set the luma matrix horizontal size. It can be an integer between 3 and 13, default value is 5.

‘`luma_msize_y`’

Set the luma matrix vertical size. It can be an integer between 3 and 13, default value is 5.

‘`luma_amount`’

Set the luma effect strength. It can be a float number between -2.0 and 5.0, default value is 1.0.

‘`chroma_msize_x`’

Set the chroma matrix horizontal size. It can be an integer between 3 and 13, default value is 5.

‘`chroma_msize_y`’

Set the chroma matrix vertical size. It can be an integer between 3 and 13, default value is 5.

‘`chroma_amount`’

Set the chroma effect strength. It can be a float number between -2.0 and 5.0, default value is 0.0.

```
# Strong luma sharpen effect parameters
unsharp=7:7:2.5

# Strong blur of both luma and chroma parameters
unsharp=7:7:-2:7:7:-2

# Use the default values with ffmpeg
ffmpeg -i in.avi -vf "unsharp" out.mp4
```

28.57 vflip

Flip the input video vertically.

```
ffmpeg -i in.avi -vf "vflip" out.avi
```

28.58 yadif

Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

It accepts the optional parameters: *mode*:*parity*:*auto*.

mode specifies the interlacing mode to adopt, accepts one of the following values:

‘`0`’

    output 1 frame for each frame

‘`1`’

    output 1 frame for each field

‘`2`’

    like 0 but skips spatial interlacing check

‘`3`’

    like 1 but skips spatial interlacing check

Default value is 0.

parity specifies the picture field parity assumed for the input interlaced video, accepts one of the following values:

‘`0`’

    assume top field first

‘`1`’

    assume bottom field first

‘`-1`’

    enable automatic detection

Default value is -1. If interlacing is unknown or decoder does not export this information, top field first will be assumed.

auto specifies if deinterlacer should trust the interlaced flag and only deinterlace frames marked as interlaced

‘`0`’

deinterlace all frames

‘`1`’

only deinterlace frames marked as interlaced

Default value is 0.

29. Video Sources

Below is a description of the currently available video sources.

29.1 buffer

Buffer video frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in ‘`libavfilter/vsrc_buffer.h`’.

It accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`video_size`’

Specify the size (width and height) of the buffered video frames.

‘`pix_fmt`’

A string representing the pixel format of the buffered video frames. It may be a number corresponding to a pixel format, or a pixel format name.

‘`time_base`’

Specify the timebase assumed by the timestamps of the buffered frames.

‘`time_base`’

Specify the frame rate expected for the video stream.

‘`pixel_aspect`’

Specify the sample aspect ratio assumed by the video frames.

‘`sws_param`’

Specify the optional parameters to be used for the scale filter which is automatically inserted when an input change is detected in the input size or format.

For example:

```
buffer=size=320x240:pix_fmt=yuv410p:time_base=1/24:pixel_aspect=1/1
```

will instruct the source to accept video frames with size 320x240 and with format "yuv410p", assuming 1/24 as the timestamps timebase and square pixels (1:1 sample aspect ratio). Since the pixel format with name "yuv410p" corresponds to the number 6 (check the enum AVPixelFormat definition in '`libavutil/pixfmt.h`'), this example corresponds to:

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

Alternatively, the options can be specified as a flat string, but this syntax is deprecated:

width:*height*:*pix_fmt*:*time_base.num*:*time_base.den*:*pixel_aspect.num*:*pixel_aspect.den*[:*sws_param*]

29.2 cellauto

Create a pattern generated by an elementary cellular automaton.

The initial state of the cellular automaton can be defined through the '`filename`', and '`pattern`' options. If such options are not specified an initial state is created randomly.

At each new frame a new row in the video is filled with the result of the cellular automaton next generation. The behavior when the whole frame is filled is defined by the '`scroll`' option.

This source accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

'`filename, f`'

Read the initial cellular automaton state, i.e. the starting row, from the specified file. In the file, each non-whitespace character is considered an alive cell, a newline will terminate the row, and further characters in the file will be ignored.

'`pattern, p`'

Read the initial cellular automaton state, i.e. the starting row, from the specified string.

Each non-whitespace character in the string is considered an alive cell, a newline will terminate the row, and further characters in the string will be ignored.

'`rate, r`'

Set the video rate, that is the number of frames generated per second. Default is 25.

‘`random_fill_ratio, ratio`’

Set the random fill ratio for the initial cellular automaton row. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI.

This option is ignored when a file or a pattern is specified.

‘`random_seed, seed`’

Set the seed for filling randomly the initial row, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

‘`rule`’

Set the cellular automaton rule, it is a number ranging from 0 to 255. Default value is 110.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ or ‘`pattern`’ is specified, the size is set by default to the width of the specified initial state row, and the height is set to *width* * PHI.

If ‘`size`’ is set, it must contain the width of the specified pattern string, and the specified pattern will be centered in the larger row.

If a filename or a pattern string is not specified, the size value defaults to "320x518" (used for a randomly generated initial state).

‘`scroll`’

If set to 1, scroll the output upward when all the rows in the output have been already filled. If set to 0, the new generated row will be written over the top row just after the bottom row is filled. Defaults to 1.

‘`start_full, full`’

If set to 1, completely fill the output with generated rows before outputting the first frame. This is the default behavior, for disabling set the value to 0.

‘`stitch`’

If set to 1, stitch the left and right row edges together. This is the default behavior, for disabling set the value to 0.

29.2.1 Examples

- Read the initial state from ‘`pattern`’, and specify an output of size 200x400.

  ```
  cellauto=f=pattern:s=200x400
  ```

- Generate a random initial row with a width of 200 cells, with a fill ratio of 2/3:

  ```
  cellauto=ratio=2/3:s=200x200
  ```

- Create a pattern generated by rule 18 starting by a single alive cell centered on an initial row with width 100:

  ```
  cellauto=p=@:s=100x400:full=0:rule=18
  ```

- Specify a more elaborated initial pattern:

  ```
  cellauto=p='@@ @ @@':s=100x400:full=0:rule=18
  ```

29.3 mandelbrot

Generate a Mandelbrot set fractal, and progressively zoom towards the point specified with *start_x* and *start_y*.

This source accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

‘`end_pts`’

Set the terminal pts value. Default value is 400.

‘`end_scale`’

Set the terminal scale value. Must be a floating point value. Default value is 0.3.

‘`inner`’

Set the inner coloring mode, that is the algorithm used to draw the Mandelbrot fractal internal region.

It shall assume one of the following values:

‘`black`’

Set black mode.

‘convergence’

Show time until convergence.

‘mincol’

Set color based on point closest to the origin of the iterations.

‘period’

Set period mode.

Default value is *mincol*.

‘bailout’

Set the bailout value. Default value is 10.0.

‘maxiter’

Set the maximum of iterations performed by the rendering algorithm. Default value is 7189.

‘outer’

Set outer coloring mode. It shall assume one of following values:

‘iteration_count’

Set iteration cound mode.

‘normalized_iteration_count’

set normalized iteration count mode.

Default value is *normalized_iteration_count*.

‘rate, r’

Set frame rate, expressed as number of frames per second. Default value is "25".

‘size, s’

Set frame size. Default value is "640x480".

‘start_scale’

Set the initial scale value. Default value is 3.0.

‘start_x’

Set the initial x position. Must be a floating point value between -100 and 100. Default value is -0.743643887037158704752191506114774.

‘start_y’

Set the initial y position. Must be a floating point value between -100 and 100. Default value is -0.131825904205311970493132056385139.

29.4 mptestsrc

Generate various test patterns, as generated by the MPlayer test filter.

The size of the generated video is fixed, and is 256x256. This source is useful in particular for testing encoding features.

This source accepts an optional sequence of *key*=*value* pairs, separated by ":". The description of the accepted options follows.

‘rate, r’

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

‘duration, d’

Set the video duration of the sourced video. The accepted syntax is:

```
[-]HH:MM:SS[.m...]
[-]S+[.m...]
```

See also the function `av_parse_time()`.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

‘test, t’

Set the number or the name of the test to perform. Supported tests are:

‘dc_luma’
‘dc_chroma’
‘freq_luma’

‘freq_chroma’
‘amp_luma’
‘amp_chroma’
‘cbp’
‘mv’
‘ring1’
‘ring2’
‘all’

Default value is "all", which will cycle through the list of all tests.

For example the following:

```
testsrc=t=dc_luma
```

will generate a "dc_luma" test pattern.

29.5 frei0r_src

Provide a frei0r source.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

The source supports the syntax:

```
size:rate:src_name[{=|:}param1:param2:...:paramN]
```

size is the size of the video to generate, may be a string of the form *width*x*height* or a frame size abbreviation. *rate* is the rate of the video to generate, may be a string of the form *num*/*den* or a frame rate abbreviation. *src_name* is the name to the frei0r source to load. For more information regarding frei0r and how to set the parameters read the section frei0r in the description of the video filters.

For example, to generate a frei0r partik0l source with size 200x200 and frame rate 10 which is overlayed on the overlay filter main input:

```
frei0r_src=200x200:10:partik0l=1234 [overlay]; [in][overlay] overlay
```

29.6 life

Generate a life pattern.

This source is based on a generalization of John Conway’s life game.

The sourced input represents a life grid, each pixel represents a cell which can be in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.

At each interaction the grid evolves according to the adopted rule, which specifies the number of neighbor alive cells which will make a cell stay alive or born. The '`rule`' option allows to specify the rule to adopt.

This source accepts a list of options in the form of *key*=*value* pairs separated by ":". A description of the accepted options follows.

'`filename, f`'

> Set the file from which to read the initial grid state. In the file, each non-whitespace character is considered an alive cell, and newline is used to delimit the end of each row.
>
> If this option is not specified, the initial grid is generated randomly.

'`rate, r`'

> Set the video rate, that is the number of frames generated per second. Default is 25.

'`random_fill_ratio, ratio`'

> Set the random fill ratio for the initial random grid. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI. It is ignored when a file is specified.

'`random_seed, seed`'

> Set the seed for filling the initial random grid, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

'`rule`'

> Set the life rule.
>
> A rule can be specified with a code of the kind "S*NS*/B*NB*", where *NS* and *NB* are sequences of numbers in the range 0-8, *NS* specifies the number of alive neighbor cells which make a live cell stay alive, and *NB* the number of alive neighbor cells which make a dead cell to become alive (i.e. to "born"). "s" and "b" can be used in place of "S" and "B", respectively.
>
> Alternatively a rule can be specified by an 18-bits integer. The 9 high order bits are used to encode the next cell state if it is alive for each number of neighbor alive cells, the low order bits specify the rule for "borning" new cells. Higher order bits encode for an higher number of neighbor cells. For example the number 6153 = `(12<<9)+9` specifies a stay alive rule of 12 and a born rule of 9, which corresponds to "S23/B03".

Default value is "S23/B3", which is the original Conway’s game of life rule, and will keep a cell alive if it has 2 or 3 neighbor alive cells, and will born a new cell if there are three alive cells around a dead cell.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ is specified, the size is set by default to the same size of the input file. If ‘`size`’ is set, it must contain the size specified in the input file, and the initial grid defined in that file is centered in the larger resulting area.

If a filename is not specified, the size value defaults to "320x240" (used for a randomly generated initial grid).

‘`stitch`’

If set to 1, stitch the left and right grid edges together, and the top and bottom edges also. Defaults to 1.

‘`mold`’

Set cell mold speed. If set, a dead cell will go from ‘`death_color`’ to ‘`mold_color`’ with a step of ‘`mold`’. ‘`mold`’ can have a value from 0 to 255.

‘`life_color`’

Set the color of living (or new born) cells.

‘`death_color`’

Set the color of dead cells. If ‘`mold`’ is set, this is the first color used to represent a dead cell.

‘`mold_color`’

Set mold color, for definitely dead and moldy cells.

29.6.1 Examples

- Read a grid from ‘`pattern`’, and center it on a grid of size 300x300 pixels:

  ```
  life=f=pattern:s=300x300
  ```

- Generate a random grid of size 200x200, with a fill ratio of 2/3:

  ```
  life=ratio=2/3:s=200x200
  ```

- Specify a custom rule for evolving a randomly generated grid:

  ```
  life=rule=S14/B34
  ```

- Full example with slow death effect (mold) using `ffplay`:

  ```
  ffplay -f lavfi life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
  ```

29.7 color, nullsrc, rgbtestsrc, smptebars, testsrc

The `color` source provides an uniformly colored input.

The `nullsrc` source returns unprocessed video frames. It is mainly useful to be employed in analysis / debugging tools, or as the source for filters which ignore the input data.

The `rgbtestsrc` source generates an RGB test pattern useful for detecting RGB vs BGR issues. You should see a red, green and blue stripe from top to bottom.

The `smptebars` source generates a color bars pattern, based on the SMPTE Engineering Guideline EG 1-1990.

The `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

These sources accept an optional sequence of *key*=*value* pairs, separated by ":". The description of the accepted options follows.

‘`color, c`’

> Specify the color of the source, only used in the `color` source. It can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence, possibly followed by an alpha specifier. The default value is "black".

‘`size, s`’

> Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation. The default value is "320x240".

‘`rate, r`’

> Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

‘`sar`’

Set the sample aspect ratio of the sourced video.

‘`duration, d`’

Set the video duration of the sourced video. The accepted syntax is:

```
[-]HH[:MM[:SS[.m...]]]
[-]S+[.m...]
```

See also the function `av_parse_time()`.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

‘`decimals, n`’

Set the number of decimals to show in the timestamp, only used in the `testsrc` source.

The displayed timestamp value will correspond to the original timestamp value multiplied by the power of 10 of the specified value. Default value is 0.

For example the following:

```
testsrc=duration=5.3:size=qcif:rate=10
```

will generate a video with a duration of 5.3 seconds, with size 176x144 and a frame rate of 10 frames per second.

The following graph description will generate a red source with an opacity of 0.2, with size "qcif" and a frame rate of 10 frames per second.

```
color=c=red@0.2:s=qcif:r=10
```

If the input content is to be ignored, `nullsrc` can be used. The following command generates noise in the luminance plane by employing the `geq` filter:

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

30. Video Sinks

Below is a description of the currently available video sinks.

30.1 buffersink

Buffer video frames, and make them available to the end of the filter graph.

This sink is mainly intended for a programmatic use, in particular through the interface defined in '`libavfilter/buffersink.h`'.

It does not require a string parameter in input, but you need to specify a pointer to a list of supported pixel formats terminated by -1 in the opaque parameter provided to `avfilter_init_filter` when initializing this sink.

30.2 nullsink

Null video sink, do absolutely nothing with the input video. It is mainly useful as a template and to be employed in analysis / debugging tools.

31. Multimedia Filters

Below is a description of the currently available multimedia filters.

31.1 asendcmd, sendcmd

Send commands to filters in the filtergraph.

These filters read commands to be sent to other filters in the filtergraph.

`asendcmd` must be inserted between two audio filters, `sendcmd` must be inserted between two video filters, but apart from that they act the same way.

The specification of commands can be provided in the filter arguments with the *commands* option, or in a file specified by the *filename* option.

These filters accept the following options:

'`commands, c`'

Set the commands to be read and sent to the other filters.

'`filename, f`'

Set the filename of the commands to be read and sent to the other filters.

31.1.1 Commands syntax

A commands description consists of a sequence of interval specifications, comprising a list of commands to be executed when a particular event related to that interval occurs. The occurring event is typically the current frame time entering or leaving a given time interval.

An interval is specified by the following syntax:

```
START[-END] COMMANDS;
```

The time interval is specified by the *START* and *END* times. *END* is optional and defaults to the maximum time.

The current frame time is considered within the specified interval if it is included in the interval [*START*, *END*), that is when the time is greater or equal to *START* and is lesser than *END*.

COMMANDS consists of a sequence of one or more command specifications, separated by ",", relating to that interval. The syntax of a command specification is given by:

```
[FLAGS] TARGET COMMAND ARG
```

FLAGS is optional and specifies the type of events relating to the time interval which enable sending the specified command, and must be a non-null sequence of identifier flags separated by "+" or "|" and enclosed between "[" and "]".

The following flags are recognized:

'`enter`'

> The command is sent when the current frame timestamp enters the specified interval. In other words, the command is sent when the previous frame timestamp was not in the given interval, and the current is.

'`leave`'

> The command is sent when the current frame timestamp leaves the specified interval. In other words, the command is sent when the previous frame timestamp was in the given interval, and the current is not.

If *FLAGS* is not specified, a default value of `[enter]` is assumed.

TARGET specifies the target of the command, usually the name of the filter class or a specific filter instance name.

COMMAND specifies the name of the command for the target filter.

ARG is optional and specifies the optional list of argument for the given *COMMAND*.

Between one interval specification and another, whitespaces, or sequences of characters starting with # until the end of line, are ignored and can be used to annotate comments.

A simplified BNF description of the commands specification syntax follows:

```
COMMAND_FLAG  ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND       ::= ["[" COMMAND_FLAGS "]"] TARGET COMMAND [ARG]
COMMANDS      ::= COMMAND [,COMMANDS]
INTERVAL      ::= START[-END] COMMANDS
INTERVALS     ::= INTERVAL[;INTERVALS]
```

31.1.2 Examples

- Specify audio tempo change at second 4:

  ```
  asendcmd=c='4.0 atempo tempo 1.5',atempo
  ```

- Specify a list of drawtext and hue commands in a file.

  ```
  # show text in the interval 5-10
  5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
           [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';

  # desaturate the image in the interval 15-20
  15.0-20.0 [enter] hue reinit s=0,
            [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
            [leave] hue reinit s=1,
            [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';

  # apply an exponential saturation fade-out effect, starting from time 25
  25 [enter] hue s=exp(t-25)
  ```

  A filtergraph allowing to read and process the above command list stored in a file '`test.cmd`', can be specified with:

  ```
  sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text='',hue
  ```

31.2 asetpts, setpts

Change the PTS (presentation timestamp) of the input frames.

`asetpts` works on audio frames, `setpts` on video frames.

Accept in input an expression evaluated through the eval API, which can contain the following constants:

‘`FRAME_RATE`’

frame rate, only defined for constant frame-rate video

‘`PTS`’

the presentation timestamp in input

‘`N`’

the count of the input frame, starting from 0.

‘`NB_CONSUMED_SAMPLES`’

the number of consumed samples, not including the current frame (only audio)

‘`NB_SAMPLES`’

the number of samples in the current frame (only audio)

‘`SAMPLE_RATE`’

audio sample rate

‘`STARTPTS`’

the PTS of the first frame

‘`STARTT`’

the time in seconds of the first frame

‘`INTERLACED`’

tell if the current frame is interlaced

‘`T`’

the time in seconds of the current frame

‘`TB`’

the time base

‘`POS`’

original position in the file of the frame, or undefined if undefined for the current frame

‘`PREV_INPTS`’

previous input PTS

‘`PREV_INT`’

previous input time in seconds

‘`PREV_OUTPTS`’

previous output PTS

‘`PREV_OUTT`’

previous output time in seconds

31.2.1 Examples

- Start counting PTS from zero

  ```
  setpts=PTS-STARTPTS
  ```

- Apply fast motion effect:

  ```
  setpts=0.5*PTS
  ```

- Apply slow motion effect:

  ```
  setpts=2.0*PTS
  ```

- Set fixed rate of 25 frames per second:

  ```
  setpts=N/(25*TB)
  ```

- Set fixed rate 25 fps with some jitter:

  ```
  setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'
  ```

- Apply an offset of 10 seconds to the input PTS:

  ```
  setpts=PTS+10/TB
  ```

31.3 ebur128

EBU R128 scanner filter. This filter takes an audio stream as input and outputs it unchanged. By default, it logs a message at a frequency of 10Hz with the Momentary loudness (identified by `M`), Short-term loudness (`S`), Integrated loudness (`I`) and Loudness Range (`LRA`).

The filter also has a video output (see the *video* option) with a real time graph to observe the loudness evolution. The graphic contains the logged message mentioned above, so it is not printed anymore when this option is set, unless the verbose logging is set. The main graphing area contains the short-term loudness (3 seconds of analysis), and the gauge on the right is for the momentary loudness (400 milliseconds).

More information about the Loudness Recommendation EBU R128 on http://tech.ebu.ch/loudness.

The filter accepts the following named parameters:

‘`video`’

Activate the video output. The audio stream is passed unchanged whether this option is set or no. The video stream will be the first output stream if activated. Default is `0`.

‘`size`’

Set the video size. This option is for video only. Default and minimum resolution is `640x480`.

‘`meter`’

Set the EBU scale meter. Default is `9`. Common values are `9` and `18`, respectively for EBU scale meter +9 and EBU scale meter +18. Any other integer value between this range is allowed.

Example of real-time graph using `ffplay`, with a EBU scale meter +18:

```
ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
```

Run an analysis with `ffmpeg`:

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

31.4 settb, asettb

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

It accepts in input an arithmetic expression representing a rational. The expression can contain the constants "AVTB" (the default timebase), "intb" (the input timebase) and "sr" (the sample rate, audio only).

The default value for the input is "intb".

31.4.1 Examples

- Set the timebase to 1/25:

  ```
  settb=1/25
  ```

- Set the timebase to 1/10:

  ```
  settb=0.1
  ```

- Set the timebase to 1001/1000:

  ```
  settb=1+0.001
  ```

- Set the timebase to 2*intb:

  ```
  settb=2*intb
  ```

- Set the default timebase value:

  ```
  settb=AVTB
  ```

31.5 concat

Concatenate audio and video streams, joining them together one after the other.

The filter works on segments of synchronized video and audio streams. All segments must have the same number of streams of each type, and that will also be the number of streams at output.

The filter accepts the following named parameters:

‘`n`’

  Set the number of segments. Default is 2.

‘`v`’

  Set the number of output video streams, that is also the number of video streams in each segment. Default is 1.

‘`a`’

Set the number of output audio streams, that is also the number of video streams in each segment. Default is 0.

‘`unsafe`’

Activate unsafe mode: do not fail if segments have a different format.

The filter has *v*+*a* outputs: first *v* video outputs, then *a* audio outputs.

There are *n*Ã(*v*+*a*) inputs: first the inputs for the first segment, in the same order as the outputs, then the inputs for the second segment, etc.

Related streams do not always have exactly the same duration, for various reasons including codec frame size or sloppy authoring. For that reason, related synchronized streams (e.g. a video and its audio track) should be concatenated at once. The concat filter will use the duration of the longest stream in each segment (except the last one), and if necessary pad shorter audio streams with silence.

For this filter to work correctly, all segments must start at timestamp 0.

All corresponding streams must have the same parameters in all segments; the filtering system will automatically select a common pixel format for video streams, and a common sample format, sample rate and channel layout for audio streams, but other settings, such as resolution, must be converted explicitly by the user.

Different frame rates are acceptable but will result in variable frame rate at output; be sure to configure the output file to handle it.

Examples:

- Concatenate an opening, an episode and an ending, all in bilingual version (video in stream 0, audio in streams 1 and 2):

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
  ’[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2]
   concat=n=3:v=1:a=2 [v] [a1] [a2]’ \
  -map ’[v]’ -map ’[a1]’ -map ’[a2]’ output.mkv
```

- Concatenate two parts, handling audio and video separately, using the (a)movie sources, and adjusting the resolution:

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;
movie=part2.mp4, scale=512:288 [v2] ; amovie=part2.mp4 [a2] ;
[v1] [v2] concat [outv] ; [a1] [a2] concat=v=0:a=1 [outa]
```

Note that a desync will happen at the stitch if the audio and video streams do not have exactly the same duration in the first file.

31.6 showspectrum

Convert input audio to a video output, representing the audio frequency spectrum.

The filter accepts the following named parameters:

‘`size, s`’

Specify the video size for the output. Default value is `640x480`.

‘`slide`’

Specify if the spectrum should slide along the window. Default value is `0`.

The usage is very similar to the showwaves filter; see the examples in that section.

31.7 showwaves

Convert input audio to a video output, representing the samples waves.

The filter accepts the following named parameters:

‘`n`’

Set the number of samples which are printed on the same column. A larger value will decrease the frame rate. Must be a positive integer. This option can be set only if the value for *rate* is not explicitly specified.

‘`rate, r`’

Set the (approximate) output frame rate. This is done by setting the option *n*. Default value is "25".

‘`size, s`’

Specify the video size for the output. Default value is "600x240".

Some examples follow.

- Output the input file audio and the corresponding video representation at the same time:

  ```
  amovie=a.mp3,asplit[out0],showwaves[out1]
  ```

- Create a synthetic signal and show it with showwaves, forcing a framerate of 30 frames per second:

```
        aevalsrc=sin(1*2*PI*t)*sin(880*2*PI*t):cos(2*PI*200*t),asplit[out0],showwaves=r=30[out1]
```

32. Multimedia Sources

Below is a description of the currently available multimedia sources.

32.1 amovie

This is the same as src_movie source, except it selects an audio stream by default.

32.2 movie

Read audio and/or video stream(s) from a movie container.

It accepts the syntax: *movie_name*[:*options*] where *movie_name* is the name of the resource to read (not necessarily a file but also a device or a stream accessed through some protocol), and *options* is an optional sequence of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`format_name, f`’

: Specifies the format assumed for the movie to read, and can be either the name of a container or an input device. If not specified the format is guessed from *movie_name* or by probing.

‘`seek_point, sp`’

: Specifies the seek point in seconds, the frames will be output starting from this seek point, the parameter is evaluated with `av_strtod` so the numerical value may be suffixed by an IS postfix. Default value is "0".

‘`streams, s`’

: Specifies the streams to read. Several streams can be specified, separated by "+". The source will then have as many outputs, in the same order. The syntax is explained in the Stream specifiers chapter. Two special names, "dv" and "da" specify respectively the default (best suited) video and audio stream. Default is "dv", or "da" if the filter is called as "amovie".

‘`stream_index, si`’

: Specifies the index of the video stream to read. If the value is -1, the best suited video stream will be automatically selected. Default value is "-1". Deprecated. If the filter is called "amovie", it will select audio instead of video.

‘`loop`’

Specifies how many times to read the stream in sequence. If the value is less than 1, the stream will be read again and again. Default value is "1".

Note that when the movie is looped the source timestamps are not changed, so it will generate non monotonically increasing timestamps.

This filter allows to overlay a second video on top of main input of a filtergraph as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                    ^
                                    |
movie --> scale--> deltapts1 -------+
```

Some examples follow.

- Skip 3.2 seconds from the start of the avi file in.avi, and overlay it on top of the input labelled as "in":

  ```
  movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [movie];
  [in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]
  ```

- Read from a video4linux2 device, and overlay it on top of the input labelled as "in":

  ```
  movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [movie];
  [in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]
  ```

- Read the first video stream and the audio stream with id 0x81 from dvd.vob; the video is connected to the pad named "video" and the audio is connected to the pad named "audio":

  ```
  movie=dvd.vob:s=v:0+#0x81 [video] [audio]
  ```

33. Metadata

FFmpeg is able to dump metadata from media files into a simple UTF-8-encoded INI-like text file and then load it back using the metadata muxer/demuxer.

The file format is as follows:

1. A file consists of a header and a number of metadata tags divided into sections, each on its own line.
2. The header is a ’;FFMETADATA’ string, followed by a version number (now 1).
3. Metadata tags are of the form ’key=value’
4. Immediately after header follows global metadata
5. After global metadata there may be sections with per-stream/per-chapter metadata.
6. A section starts with the section name in uppercase (i.e. STREAM or CHAPTER) in brackets (’[’, ’]’) and ends with next section or end of file.

7. At the beginning of a chapter section there may be an optional timebase to be used for start/end values. It must be in form ’TIMEBASE=num/den’, where num and den are integers. If the timebase is missing then start/end times are assumed to be in milliseconds. Next a chapter section must contain chapter start and end times in form ’START=num’, ’END=num’, where num is a positive integer.
8. Empty lines and lines starting with ’;’ or ’#’ are ignored.
9. Metadata keys or values containing special characters (’=’, ’;’, ’#’, ’\’ and a newline) must be escaped with a backslash ’\’.
10. Note that whitespace in metadata (e.g. foo = bar) is considered to be a part of the tag (in the example above key is ’foo ’, value is ’ bar’).

A ffmetadata file might look like this:

```
;FFMETADATA1
title=bike\\shed
;this is a comment
artist=FFmpeg troll team

[CHAPTER]
TIMEBASE=1/1000
START=0
#chapter ends at 0:01:00
END=60000
title=chapter \#1
[STREAM]
title=multi\
line
```

This document was generated by *john* on *November 26, 2012* using *texi2html 1.82*.