avconv Documentation

Table of Contents

1. Synopsis (avconv.html#toc-Synopsis)

The generic syntax is:

```
avconv [global options] [[infile options]['-i' infile]]... {[outfile options] outfile}...
```

2. Description (avconv.html#toc-Description)

avconv 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.

avconv 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:

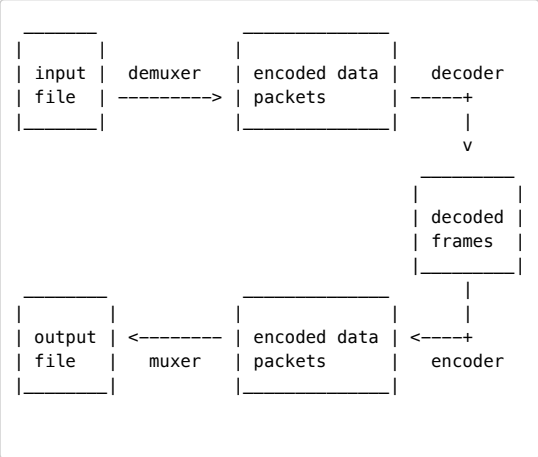
```
avconv -i input.avi -b 64k output.avi
```

- To force the frame rate of the output file to 24 fps:
  ```
  avconv -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:
  ```
  avconv -r 1 -i input.m2v -r 24 output.avi
  ```

The format option may be needed for raw input files.

3. Detailed description (avconv.html#toc-Detailed-description)

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



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

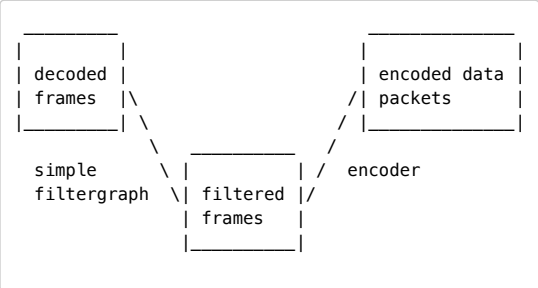
Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering the frames are passed to the 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 (avconv.html#toc-Filtering)

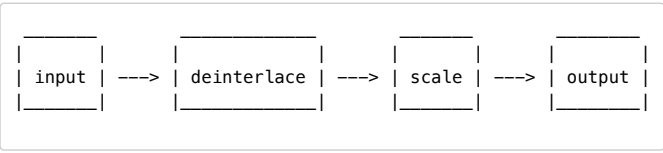
Before encoding, `avconv` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `avconv` distinguishes between two types of filtergraphs - simple and complex.

3.1.1 Simple filtergraphs (avconv.html#toc-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:



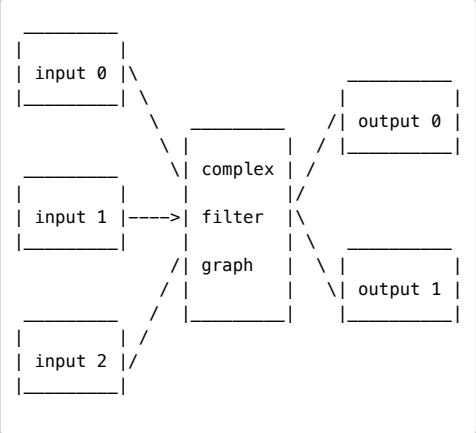
Simple filtergraphs are configured with the per-stream '`-filter`' option (with '`-vf`' and '`-af`' aliases for video and audio respectively). A simple filtergraph for video can look for example like this:



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 (avconv.html#toc-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:



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 (avconv.html#toc-Stream-copy)

Stream copy is a mode selected by supplying the `copy` parameter to the '`-codec`' option. It makes `avconv` 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:

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 (avconv.html#toc-Stream-selection)

By default avconv tries to pick the "best" stream of each type present in input files and add them to each output file. For video, this means the highest resolution, for audio the highest channel count. For subtitle it's simply the first subtitle stream.

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 (avconv.html#toc-Options-2)

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 (avconv.html#toc-Stream-specifiers-1)

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 stream, 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.

'i:*stream_id*'
Match the stream by stream id (e.g. PID in MPEG-TS container).

'm:*key*[:*value*]'
Matches streams with the metadata tag *key* having the specified value. If *value* is not given, matches streams that contain the given tag with any value.

'u'
Matches streams with usable configuration, the codec must be defined and the essential information such as video dimension or audio sample rate must be present.

Note that in `avconv` , matching by metadata will only work properly for input files.

5.2 Generic options (avconv.html#toc-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.

'filter=*filter_name*'
Print detailed information about the filter name *filter_name*. Use the '`-filters`' option to get a list of all filters.

'-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.

‘-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’
‘trace’

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 Libav version.

‘-cpuflags mask (*global*)’
Set a mask that’s applied to autodetected CPU flags. This option is intended for testing. Do not use it unless you know what you’re doing.

5.3 AVOptions (avconv.html#toc-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:

```
avconv -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 (avconv.html#toc-Codec-AVOptions)

‘-b[:stream_specifier] *integer* (*output,audio,video*)’
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 minimum/maximum bitrate. Lowering tolerance too much has an adverse effect on quality.

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

‘unaligned’
allow decoders to produce unaligned output

‘mv4’
use four motion vectors per macroblock (MPEG-4)

‘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 2-pass ratecontrol in first pass mode

‘pass2’
use internal 2-pass ratecontrol in second pass mode

‘gray’
only decode/encode grayscale

‘emu_edge’
do not 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 functions (except (I)DCT)

‘aic’
H.263 advanced intra coding / MPEG-4 AC prediction

‘ilme’
interlaced motion estimation

‘cgop’
closed GOP

‘output_corrupt’
Output even potentially corrupted frames

‘-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’
diamond 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

‘-g[:stream_specifier] *integer* (*output,video*)’
set the group of picture (GOP) 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*)’
‘-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*)’
minimum video quantizer scale (VBR)

‘-qmax[:stream_specifier] *integer* (*output,video*)’
maximum video quantizer scale (VBR)

‘-qdiff[:stream_specifier] *integer* (*output,video*)’
maximum difference between the quantizer scales (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

‘-bug[:stream_specifier] *flags* (*input,video*)’
work around not autodetected 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’
work around various bugs in Microsoft’s broken decoders

‘trunc’
truncated frames

‘-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 the 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

‘-mpeg_quant[:stream_specifier] *integer* (*output,video*)’
use MPEG quantizers instead of H.263

‘-qsquish[:stream_specifier] *float* (*output,video*)’
deprecated, use encoder private options instead

‘-rc_qmod_amp[:stream_specifier] *float* (*output,video*)’
deprecated, use encoder private options instead

‘-rc_qmod_freq[:stream_specifier] *integer* (*output,video*)’
deprecated, use encoder private options instead

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

deprecated, use encoder private options instead

‘-maxrate[:stream_specifier] *integer* (*output,audio,video*)’
Set maximum bitrate tolerance (in bits/s). Requires bufsize to be set.

‘-minrate[:stream_specifier] *integer* (*output,audio,video*)’
Set minimum bitrate tolerance (in bits/s). Most useful in setting up a CBR encode. It is of little use otherwise.

‘-bufsize[:stream_specifier] *integer* (*output,audio,video*)’
set ratecontrol buffer size (in bits)

‘-rc_buf_aggressivity[:stream_specifier] *float* (*output,video*)’
deprecated, use encoder private options instead

‘-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*)’
deprecated, use encoder private options instead

‘-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’
‘arm’
‘altivec’
‘sh4’
‘simplearm’
‘simplearmv5te’
‘simplearmv6’
‘simpleneon’
‘simplealpha’
‘ipp’
‘xvid’

‘xvidmmx’
‘faani’
floating point AAN IDCT

‘-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

‘-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

‘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

‘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

‘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

‘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

‘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

‘-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

‘-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

‘-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

‘-sc_threshold[:stream_specifier] *integer* (*output,video*)’
scene change threshold

‘-lmin[:stream_specifier] *integer* (*output,video*)’
deprecated, use encoder private options instead

‘-lmax[:stream_specifier] *integer* (*output,video*)’
deprecated, use encoder private options instead

‘-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

‘-flags2[:stream_specifier] *flags* (*input/output,audio,video*)’
Possible values:

‘fast’
allow non-spec-compliant speedup tricks

‘noout’
skip bitstream encoding

‘ignorecrop’
ignore cropping information from sps

‘local_header’
place global headers at every keyframe instead of in extradata

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

‘auto’
autodetect a suitable number of threads to use

‘-me_threshold[:stream_specifier] *integer* (*output,video*)’
motion estimation 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’
‘mpeg2_aac_low’
‘mpeg2_aac_he’
‘dts’
‘dts_es’
‘dts_96_24’
‘dts_hd_hra’
‘dts_hd_ma’
‘main10’

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

‘unknown’

‘-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

‘dctmax’
‘chroma’

‘-border_mask[:stream_specifier] *float* (*output,video*)’
deprecated, use encoder private options instead

‘-mblmin[:stream_specifier] *integer* (*output,video*)’
minimum macroblock Lagrange factor (VBR)

‘-mblmax[:stream_specifier] *integer* (*output,video*)’
maximum 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*)’
downscale frames for dynamic B-frame decision

‘-keyint_min[:stream_specifier] *integer* (*output,video*)’
minimum interval between IDR-frames (x264)

‘-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*)’
adjust 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

‘-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 primaries

Possible values:

‘bt709’
BT.709

‘unknown’
Unspecified

‘bt470m’
BT.470 M

‘bt470bg’
BT.470 BG

‘smpte170m’
SMPTE 170 M

‘smpte240m’
SMPTE 240 M

‘film’
Film

‘bt2020’
BT.2020

‘smpte428’
SMPTE 428-1

‘smpte431’
SMPTE 431-2

‘smpte432’
SMPTE 422-1

‘jedec-p22’
JEDEC P22

‘unspecified’
Unspecified

‘smptest428_1’
SMPTE 428-1

‘-color_trc[:stream_specifier] *integer* (*input/output,video*)’
color transfer characteristics

Possible values:

‘bt709’
BT.709

‘unknown’

Unspecified

‘gamma22’
BT.470 M

‘gamma28’
BT.470 BG

‘smpte170m’
SMPTE 170 M

‘smpte240m’
SMPTE 240 M

‘linear’
Linear

‘log100’
Log

‘log316’
Log square root

‘iec61966-2-4’
IEC 61966-2-4

‘bt1361e’
BT.1361

‘iec61966-2-1’
IEC 61966-2-1

‘bt2020-10’
BT.2020 - 10 bit

‘bt2020-12’
BT.2020 - 12 bit

‘smpte2084’
SMPTE 2084

‘smpte428’
SMPTE 428-1

‘arib-std-b67’
ARIB STD-B67

‘unspecified’
Unspecified

‘log’
Log

‘log_sqrt’
Log square root

‘iec61966_2_4’
IEC 61966-2-4

‘bt1361’
BT.1361

‘iec61966_2_1’
IEC 61966-2-1

‘bt2020_10bit’
BT.2020 - 10 bit

‘bt2020_12bit’
BT.2020 - 12 bit

‘smptest2084’
SMPTE 2084

‘smptest428_1’
SMPTE 428-1

‘-colorspace[:stream_specifier] *integer* (*input/output,video*)’
color space

Possible values:

‘rgb’
RGB

‘bt709’
BT.709

‘unknown’
Unspecified

‘fcc’
FCC

‘bt470bg’
BT.470 BG

‘smpte170m’
SMPTE 170 M

‘smpte240m’
SMPTE 240 M

‘ycgco’
YCGCO

‘bt2020nc’
BT.2020 NCL

‘bt2020c’
BT.2020 CL

‘smpte2085’
SMPTE 2085

‘unspecified’
Unspecified

‘ycocg’
YCGCO

‘bt2020_ncl’
BT.2020 NCL

‘bt2020_cl’
BT.2020 CL

‘-color_range[:stream_specifier] *integer* (*input/output,video*)’
color range

Possible values:

‘unknown’
Unspecified

‘tv’
MPEG (219*2^(n-8))

‘pc’
JPEG (2^n-1)

‘unspecified’
Unspecified

‘mpeg’
MPEG (219*2^(n-8))

‘jpeg’
JPEG (2^n-1)

‘-chroma_sample_location[:stream_specifier] *integer* (*input/output,video*)’
chroma sample location

Possible values:

‘unknown’
Unspecified

‘left’

Left

‘center’
Center

‘topleft’
Top-left

‘top’
Top

‘bottomleft’
Bottom-left

‘bottom’
Bottom

‘unspecified’
Unspecified

‘-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] *integer* (*input,audio*)’
Possible values:

‘u8’
8-bit unsigned integer

‘s16’
16-bit signed integer

‘s32’
32-bit signed integer

‘flt’
32-bit float

‘dbl’

64-bit double

‘u8p’
8-bit unsigned integer planar

‘s16p’
16-bit signed integer planar

‘s32p’
32-bit signed integer planar

‘fltp’
32-bit float planar

‘dblp’
64-bit double planar

‘-refcounted_frames[:stream_specifier] *integer* (*input,audio,video*)’
‘-side_data_only_packets[:stream_specifier] *integer* (*output,audio,video*)’

5.5 Format AVOptions (avconv.html#toc-Format-AVOptions)

‘-probesize *integer* (*input*)’
set probing size

‘-packetsize *integer* (*output*)’
set packet size

‘-fflags *flags* (*input/output*)’
Possible values:

‘flush_packets’
reduce the latency by flushing out packets immediately

‘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

‘nobuffer’
reduce the latency introduced by optional buffering

‘bitexact’
do not write random/volatile data

‘-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

‘-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

‘-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

‘-max_interleave_delta *integer* (*output*)’
maximum buffering duration for interleaving

‘-f_strict *integer* (*input/output*)’
how strictly to follow the standards (deprecated; use strict, save via avconv)

Possible values:

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

‘normal’
‘experimental’
allow non-standardized experimental variants

‘-strict *integer* (*input/output*)’
how strictly to follow the standards

Possible values:

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

‘normal’
‘experimental’
allow non-standardized experimental variants

‘-max_ts_probe *integer* (*input*)’
maximum number of packets to read while waiting for the first timestamp

‘-avoid_negative_ts *integer* (*output*)’
shift timestamps so they start at 0

Possible values:

‘auto’
enabled when required by target format

‘make_non_negative’

shift timestamps so they are non negative

‘make_zero’
shift timestamps so they start at 0

‘-protocol_blacklist *string* (*input/output*)’
A comma-separated list of blacklisted protocols used for opening files internally by lavf

‘-protocol_whitelist *string* (*input/output*)’
A comma-separated list of whitelisted protocols used for opening files internally by lavf

5.6 Main options (avconv.html#toc-Main-options)

‘-f *fmt* (*input/output*)’
Force input or output file format. The format is normally autodetected 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*)’
Immediately exit when output files already exist.

‘-loop *number* (*input*)’
Set number of times input stream shall be looped. Loop 0 means no loop, loop -1 means infinite loop.

‘-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 reencoded.

For example

```
avconv -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

```
avconv -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.

‘-ss *position* (*input/output*)’
When used as an input option (before `-i`), seeks in this input file to *position*. Note the in most formats it is not possible to seek exactly, so `avconv` will seek to the closest seek point before *position*. When transcoding and ‘`-accurate_seek`’ is enabled (the default), this extra segment between the seek point and *position* will be decoded and discarded. When doing stream copy or when ‘`-noaccurate_seek`’ is used, it will be preserved.

When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*.

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.

‘-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:

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

To set the language of the first audio stream:

```
avconv -i INPUT -metadata:s:a:0 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:

```
avconv -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:

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

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

Set the number of data frames to record. This is an obsolete alias for `-frames:d`, which you should use instead.

'-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.

'-filter_script[:*stream_specifier*] *filename* (*output,per-stream*)'

This option is similar to '`-filter`', the only difference is that its argument is the name of the file from which a filtergraph description is to be read.

'-pre[:*stream_specifier*] *preset_name* (*output,per-stream*)'

Specify the preset for matching stream(s).

'-stats (*global*)'

Print encoding progress/statistics. On by default.

'-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:

```
avconv -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':

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

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

```
avconv -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.

'-noautorotate'

Disable automatically rotating video based on file metadata.

5.7 Video Options (avconv.html#toc-Video-Options)

'-vframes *number* (*output*)'

Set the number of video frames to record. This is an obsolete alias for `-frames:v`, which you should use instead.

'-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). 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

‘2kdci’
2048x1080

‘4kdci’
4096x2160

‘uhd2160’
3840x2160

‘uhd4320’
7680x4320

‘-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.

‘-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:

```
avconv -i foo.mov -c:v libxvid -pass 1 -an -f rawvideo -y NUL
avconv -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 “av2pass”. The complete file name will be ‘`PREFIX-N.log`’, where N is a number specific to the output 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 (avconv.html#toc-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.

‘-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

‘-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.

‘-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.

‘-hwaccel[:*stream_specifier*] *hwaccel* (*input,per-stream*)’
Use hardware acceleration to decode the matching stream(s). The allowed values of *hwaccel* are:

‘none’
Do not use any hardware acceleration (the default).

‘auto’
Automatically select the hardware acceleration method.

‘vda’
Use Apple VDA hardware acceleration.

‘vdpau’
Use VDPAU (Video Decode and Presentation API for Unix) hardware acceleration.

‘dxva2’
Use DXVA2 (DirectX Video Acceleration) hardware acceleration.

‘qsv’
Use the Intel QuickSync Video acceleration for video transcoding.

Unlike most other values, this option does not enable accelerated decoding (that is used automatically whenever a qsv decoder is selected), but accelerated transcoding, without copying the frames into the system memory.

For it to work, both the decoder and the encoder must support QSV acceleration and no filters must be used.

This option has no effect if the selected hwaccel is not available or not supported by the chosen decoder.

Note that most acceleration methods are intended for playback and will not be faster than software decoding on modern CPUs. Additionally, avconv will usually need to copy the decoded frames from the GPU memory into the system memory, resulting in further performance loss. This option is thus mainly useful for testing.

‘-hwaccel_device[:*stream_specifier*] *hwaccel_device* (*input,per-stream*)’
Select a device to use for hardware acceleration.

This option only makes sense when the ‘-hwaccel’ option is also specified. Its exact meaning depends on the specific hardware acceleration method chosen.

‘vdpau’
For VDPAU, this option specifies the X11 display/screen to use. If this option is not specified, the value of the *DISPLAY* environment variable is used

‘dxva2’
For DXVA2, this option should contain the number of the display adapter to use. If this option is not specified, the default adapter is used.

‘qsv’
For QSV, this option corresponds to the values of MFX_IMPL_* . Allowed values are:

‘auto’
‘sw’
‘hw’
‘auto_any’
‘hw_any’

‘hw2’
‘hw3’
‘hw4’

‘–hwaccels’
List all hardware acceleration methods supported in this build of avconv.

5.9 Audio Options (avconv.html#toc-Audio-Options)

‘–aframes *number* (*output*)’
Set the number of audio frames to record. This is an obsolete alias for `–frames:a` , which you should use instead.

‘–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: (avconv.html#toc-Advanced-Audio-options_003a)

‘–atag *fourcc/tag* (*output*)’
Force audio tag/fourcc. This is an alias for `–tag:a` .

5.11 Subtitle options: (avconv.html#toc-Subtitle-options_003a)

‘–scodec *codec* (*input/output*)’
Set the subtitle codec. This is an alias for `–codec:s` .

‘–sn (*output*)’
Disable subtitle recording.

5.12 Advanced options (avconv.html#toc-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

```
avconv –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:

```
avconv -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’:

```
avconv -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:

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

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

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

To pick the English audio stream:

```
avconv -i INPUT -map 0:m:language:eng OUTPUT
```

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

‘-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:

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

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

```
avconv -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.

‘-debug’
Print specific debug info.

‘-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.

‘-timelimit *duration* (*global*)’
Exit after avconv 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 or live input stream (e.g. when reading from a file). Should not be used with actual grab devices or live input streams (where it can cause packet loss).

‘-vsync *parameter*’
Video sync method.

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

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

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

‘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’
Copy input stream time base from input to output when stream copying.

‘-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:

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

‘-bsf[:*stream_specifier*] *bitstream_filters* (*output,per-stream*)’
Set bitstream filters for matching streams. *bitstream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

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

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

‘-tag[:*stream_specifier*] *codec_tag* (*input/output,per-stream*)’
Force a tag/fourcc for matching streams.

‘-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

```
avconv -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

```
avconv -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

```
avconv -i video.mkv -i image.png -filter_complex 'overlay' out.mkv
```

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

```
avconv -filter_complex 'color=red' -t 5 out.mkv
```

'-filter_complex_script *filename* (*global*)'
This option is similar to '`-filter_complex`', the only difference is that its argument is the name of the file from which a complex filtergraph description is to be read.

'-accurate_seek (*input*)'
This option enables or disables accurate seeking in input files with the '`-ss`' option. It is enabled by default, so seeking is accurate when transcoding. Use '`-noaccurate_seek`' to disable it, which may be useful e.g. when copying some streams and transcoding the others.

'-max_muxing_queue_size *packets* (*output,per-stream*)'
When transcoding audio and/or video streams, avconv will not begin writing into the output until it has one packet for each such stream. While waiting for that to happen, packets for other streams are buffered. This option sets the size of this buffer, in packets, for the matching output stream.

The default value of this option should be high enough for most uses, so only touch this option if you are sure that you need it.

6. Tips (avconv.html#toc-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:
  ```
  avconv -g 3 -r 3 -t 10 -b 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 (avconv.html#toc-Examples-1)

7.1 Preset files (avconv.html#toc-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 Libav source tree for examples.

Preset files are specified with the `pre` option, this option takes a preset name as input. Avconv searches for a file named *preset_name*.avpreset in the directories '`$AVCONV_DATADIR`' (if set), and '`$HOME/.avconv`', and in the data directory defined at configuration time (usually '`$PREFIX/share/avconv`') 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 (avconv.html#toc-Video-and-Audio-grabbing)

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

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

Note that you must activate the right video source and channel before launching avconv with any TV viewer such as xawtv (http://linux.bytesex.org/xawtv/) by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing (avconv.html#toc-X11-grabbing)

Grab the X11 display with avconv via

```
avconv -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.

```
avconv -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 (avconv.html#toc-Video-and-Audio-file-format-conversion)

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

Examples:

- You can use YUV files as input:

  ```
  avconv -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 avconv cannot guess it.

- You can input from a raw YUV420P file:

  ```
  avconv -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:

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

- You can set several input files and output files:

  ```
  avconv -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:

  ```
  avconv -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:

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

  ```
  avconv -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 `avconv -formats`.

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

```
avconv -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 `-frames:v` or `-t` option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

```
avconv -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.

- You can put many streams of the same type in the output:
  ```
  avconv -i test1.avi -i test2.avi -map 1:1 -map 1:0 -map 0:1 -map 0:0 -c copy -y test12.nut
  ```
  The resulting output file ‘`test12.nut`’ will contain the first four streams from the input files in reverse order.
- To force CBR video output:
  ```
  avconv -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:
  ```
  avconv -i src.ext -lmax 21*QP2LAMBDA dst.ext
  ```

8. Expression Evaluation (avconv.html#toc-Expression-Evaluation)

When evaluating an arithmetic expression, Libav 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)’
‘cosh(x)’
‘tanh(x)’
‘sin(x)’
‘cos(x)’
‘tan(x)’
‘atan(x)’
‘asin(x)’
‘acos(x)’
‘exp(x)’
‘log(x)’
‘abs(x)’
‘squish(x)’
‘gauss(x)’
‘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)’
‘max(x, y)’
‘min(x, y)’
‘eq(x, y)’
‘gte(x, y)’
‘gt(x, y)’
‘lte(x, y)’

‘lt(x, y)’

‘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.

‘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.

Note that:

∗ works like AND

\+ works like OR

thus

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

is equivalent to

```
A∗B + not(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

9. Decoders (avconv.html#toc-Decoders)

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

When you configure your Libav 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 `-decoders` of the av* tools will display the list of enabled decoders.

10. Audio Decoders (avconv.html#toc-Audio-Decoders)

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

10.1 ac3 (avconv.html#toc-ac3)

AC-3 audio decoder.

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

10.1.1 AC-3 Decoder Options (avconv.html#toc-AC_002d3-Decoder-Options)

‘`-drc_scale` *`value`*’
Dynamic Range Scale Factor. The factor to apply to dynamic range values from the AC-3 stream. This factor is applied exponentially. There are 3 notable scale factor ranges:

‘`drc_scale == 0`’
DRC disabled. Produces full range audio.

‘`0 < drc_scale <= 1`’
DRC enabled. Applies a fraction of the stream DRC value. Audio reproduction is between full range and full compression.

‘`drc_scale > 1`’
DRC enabled. Applies drc_scale asymmetrically. Loud sounds are fully compressed. Soft sounds are enhanced.

11. Encoders (avconv.html#toc-Encoders)

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

When you configure your Libav 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 `-encoders` of the av* tools will display the list of enabled encoders.

12. Audio Encoders (avconv.html#toc-Audio-Encoders)

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

12.1 ac3 and ac3_fixed (avconv.html#toc-ac3-and-ac3_005ffixed)

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.

12.1.1 AC-3 Metadata (avconv.html#toc-AC_002d3-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 (http://www.atsc.org/cms/standards/a_52-2010.pdf)
- A/54 - Guide to the Use of the ATSC Digital Television Standard (http://www.atsc.org/cms/standards/a_54a_with_corr_1.pdf)
- Dolby Metadata Guide (http://www.dolby.com/uploadedFiles/zz-_Shared_Assets/English_PDFs/Professional/18_Metadata.Guide.pdf)
- Dolby Digital Professional Encoding Guidelines (http://www.dolby.com/uploadedFiles/zz-_Shared_Assets/English_PDFs/Professional/46_DDEncodingGuidelines.pdf)

12.1.1.1 Metadata Control Options (avconv.html#toc-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.

12.1.1.2 Downmix Levels (avconv.html#toc-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)

12.1.1.3 Audio Production Information (avconv.html#toc-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

12.1.1.4 Other Metadata Options (avconv.html#toc-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)

12.1.2 Extended Bitstream Information (avconv.html#toc-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.

12.1.2.1 Extended Bitstream Information - Part 1 (avconv.html#toc-Extended-Bitstream-Information-_002d-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)

12.1.2.2 Extended Bitstream Information - Part 2 (avconv.html#toc-Extended-Bitstream-Information-_002d-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

12.1.3 Other AC-3 Encoding Options (avconv.html#toc-Other-AC_002d3-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.

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)

12.2 libwavpack (avconv.html#toc-libwavpack)

A wrapper providing WavPack encoding through libwavpack.

Only lossless mode using 32-bit integer samples is supported currently. The ‘`compression_level`’ option can be used to control speed vs. compression tradeoff, with the values mapped to libwavpack as follows:

‘`0`’
Fast mode - corresponding to the wavpack ‘`-f`’ option.

‘`1`’
Normal (default) settings.

‘`2`’
High quality - corresponding to the wavpack ‘-h’ option.

‘`3`’
Very high quality - corresponding to the wavpack ‘-hh’ option.

‘`4-8`’
Same as 3, but with extra processing enabled - corresponding to the wavpack ‘-x’ option. I.e. 4 is the same as ‘-x2’ and 8 is the same as ‘-x6’.

13. Video Encoders (avconv.html#toc-Video-Encoders)

13.1 libwebp (avconv.html#toc-libwebp)

libwebp WebP Image encoder wrapper

libwebp is Google’s official encoder for WebP images. It can encode in either lossy or lossless mode. Lossy images are essentially a wrapper around a VP8 frame. Lossless images are a separate codec developed by Google.

13.1.1 Pixel Format (avconv.html#toc-Pixel-Format)

Currently, libwebp only supports YUV420 for lossy and RGB for lossless due to limitations of the format and libwebp. Alpha is supported for either mode. Because of API limitations, if RGB is passed in when encoding lossy or YUV is passed in for encoding lossless, the pixel format will automatically be converted using functions from libwebp. This is not ideal and is done only for convenience.

13.1.2 Options (avconv.html#toc-Options-1)

‘`-lossless` *`boolean`*’
Enables/Disables use of lossless mode. Default is 0.

‘`-compression_level` *`integer`*’
For lossy, this is a quality/speed tradeoff. Higher values give better quality for a given size at the cost of increased encoding time. For lossless, this is a size/speed tradeoff. Higher values give smaller size at the cost of increased encoding time. More specifically, it controls the number of extra algorithms and compression tools used, and varies the combination of these tools. This maps to the *method* option in libwebp. The valid range is 0 to 6. Default is 4.

‘`-qscale` *`float`*’
For lossy encoding, this controls image quality, 0 to 100. For lossless encoding, this controls the effort and time spent at compressing more. The default value is 75. Note that for usage via libavcodec, this option is called *global_quality* and must be multiplied by *FF_QP2LAMBDA*.

‘`-preset` *`type`*’
Configuration preset. This does some automatic settings based on the general type of the image.

‘`none`’
Do not use a preset.

‘`default`’
Use the encoder default.

‘`picture`’
Digital picture, like portrait, inner shot

‘`photo`’
Outdoor photograph, with natural lighting

‘`drawing`’
Hand or line drawing, with high-contrast details

‘`icon`’
Small-sized colorful images

‘`text`’
Text-like

‘`lumi_aq`’
Enable lumi masking adaptive quantization when set to 1. Default is 0 (disabled).

‘`variance_aq`’
Enable variance adaptive quantization when set to 1. Default is 0 (disabled).

When combined with ‘`lumi_aq`’, the resulting quality will not be better than any of the two specified individually. In other words, the resulting quality will be the worse one of the two effects.

‘`ssim`’
Set structural similarity (SSIM) displaying method. Possible values:

‘`off`’
Disable displaying of SSIM information.

‘`avg`’
Output average SSIM at the end of encoding to stdout. The format of showing the average SSIM is:

```
Average SSIM: %f
```

For users who are not familiar with C, %f means a float number, or a decimal (e.g. 0.939232).

‘`frame`’
Output both per-frame SSIM data during encoding and average SSIM at the end of encoding to stdout. The format of per-frame information is:

```
       SSIM: avg: %1.3f min: %1.3f max: %1.3f
```

For users who are not familiar with C, %1.3f means a float number rounded to 3 digits after the dot (e.g. 0.932).

‘`ssim_acc`’
Set SSIM accuracy. Valid options are integers within the range of 0-4, while 0 gives the most accurate result and 4 computes the fastest.

13.2 libx264 (avconv.html#toc-libx264)

x264 H.264/MPEG-4 AVC encoder wrapper

x264 supports an impressive number of features, including 8x8 and 4x4 adaptive spatial transform, adaptive B-frame placement, CAVLC/CABAC entropy coding, interlacing (MBAFF), lossless mode, psy optimizations for detail retention (adaptive quantization, psy-RD, psy-trellis).

The Libav wrapper provides a mapping for most of them using global options that match those of the encoders and provides private options for the unique encoder options. Additionally an expert override is provided to directly pass a list of key=value tuples as accepted by x264_param_parse.

13.2.1 Option Mapping (avconv.html#toc-Option-Mapping)

The following options are supported by the x264 wrapper, the x264-equivalent options follow the Libav ones.

b	bitrate	Libav `b` option is expressed in bits/s, x264 `bitrate` in kilobits/s.
bf	bframes	Maximum number of B-frames.
g	keyint	Maximum GOP size.
qmin	qpmin	Minimum quantizer scale.
qmax	qpmax	Maximum quantizer scale.
qdiff	qpstep	Maximum difference between quantizer scales.
qblur	qblur	Quantizer curve blur
qcomp	qcomp	Quantizer curve compression factor
refs	ref	Number of reference frames each P-frame can use. The range is from *0-16*.
sc_threshold	scenecut	Sets the threshold for the scene change detection.
trellis	trellis	Performs Trellis quantization to increase efficiency. Enabled by default.
nr	nr	Noise reduction.
me_range	merange	Maximum range of the motion search in pixels.
subq	subme	Sub-pixel motion estimation method.
b_strategy	b-adapt	Adaptive B-frame placement decision algorithm. Use only on first-pass.
keyint_min	min-keyint	Minimum GOP size.
coder	cabac	Set coder to `ac` to use CABAC.
cmp	chroma-me	Set to `chroma` to use chroma motion estimation.
threads	threads	Number of encoding threads.
thread_type	sliced_threads	Set to `slice` to use sliced threading instead of frame threading.
flags -cgop	open-gop	Set `-cgop` to use recovery points to close GOPs.
rc_init_occupancy	vbv-init	Initial buffer occupancy.

13.2.2 Private Options (avconv.html#toc-Private-Options)

‘`-preset` *`string`*’
Set the encoding preset (cf. x264 –fullhelp).

‘`-tune` *`string`*’
Tune the encoding params (cf. x264 –fullhelp).

‘`-profile` *`string`*’
Set profile restrictions (cf. x264 –fullhelp).

‘`-fastfirstpass` *`integer`*’
Use fast settings when encoding first pass.

‘`-crf` *`float`*’
Select the quality for constant quality mode.

‘`-crf_max` *`float`*’
In CRF mode, prevents VBV from lowering quality beyond this point.

‘`-qp` *`integer`*’
Constant quantization parameter rate control method.

‘`-aq-mode` *`integer`*’
AQ method

Possible values:

‘`none`’
‘`variance`’
Variance AQ (complexity mask).

‘`autovariance`’
Auto-variance AQ (experimental).

‘`-aq-strength` *`float`*’
AQ strength, reduces blocking and blurring in flat and textured areas.

‘-psy *integer*’
Use psychovisual optimizations.

‘-psy-rd *string*’
Strength of psychovisual optimization, in <psy-rd>:<psy-trellis> format.

‘-rc-lookahead *integer*’
Number of frames to look ahead for frametype and ratecontrol.

‘-weightb *integer*’
Weighted prediction for B-frames.

‘-weightp *integer*’
Weighted prediction analysis method.

Possible values:

‘none’
‘simple’
‘smart’

‘-ssim *integer*’
Calculate and print SSIM stats.

‘-intra-refresh *integer*’
Use Periodic Intra Refresh instead of IDR frames.

‘-bluray-compat *integer*’
Configure the encoder to be compatible with the bluray standard. It is a shorthand for setting "bluray-compat=1 force-cfr=1".

‘-b-bias *integer*’
Influences how often B-frames are used.

‘-b-pyramid *integer*’
Keep some B-frames as references.

Possible values:

‘none’
‘strict’
Strictly hierarchical pyramid.

‘normal’
Non-strict (not Blu-ray compatible).

‘-mixed-refs *integer*’
One reference per partition, as opposed to one reference per macroblock.

‘-8x8dct *integer*’
High profile 8x8 transform.

‘-fast-pskip *integer*’
‘-aud *integer*’
Use access unit delimiters.

‘-mbtree *integer*’
Use macroblock tree ratecontrol.

‘-deblock *string*’
Loop filter parameters, in <alpha:beta> form.

‘-cplxblur *float*’
Reduce fluctuations in QP (before curve compression).

‘-partitions *string*’
A comma-separated list of partitions to consider, possible values: p8x8, p4x4, b8x8, i8x8, i4x4, none, all.

‘-direct-pred *integer*’
Direct MV prediction mode

Possible values:

‘none’
‘spatial’
‘temporal’
‘auto’

‘-slice-max-size *integer*’
Limit the size of each slice in bytes.

‘-stats *string*’
Filename for 2 pass stats.

‘-nal-hrd *integer*’
Signal HRD information (requires vbv-bufsize; cbr not allowed in .mp4).

Possible values:

‘none’
‘vbr’
‘cbr’

‘-x264-params *string*’
Override the x264 configuration using a :-separated list of key=value parameters.

```
-x264-params level=30:bframes=0:weightp=0:cabac=0:ref=1:vbv-maxrate=768:vbv-bufsize=2000:analyse=all:me=umh:no-fast-pskip=1:subq=6:8x8
```

Encoding avpresets for common usages are provided so they can be used with the general presets system (e.g. passing the `-pre` option).

13.3 ProRes (avconv.html#toc-ProRes)

Apple ProRes encoder.

13.3.1 Private Options (avconv.html#toc-Private-Options-1)

‘profile *integer*’
Select the ProRes profile to encode

‘proxy’
‘lt’
‘standard’
‘hq’
‘4444’

‘quant_mat *integer*’
Select quantization matrix.

‘auto’
‘default’
‘proxy’
‘lt’
‘standard’
‘hq’

If set to *auto*, the matrix matching the profile will be picked. If not set, the matrix providing the highest quality, *default*, will be picked.

‘bits_per_mb *integer*’
How many bits to allot for coding one macroblock. Different profiles use between 200 and 2400 bits per macroblock, the maximum is 8000.

‘mbs_per_slice *integer*’
Number of macroblocks in each slice (1-8); the default value (8) should be good in almost all situations.

‘vendor *string*’
Override the 4-byte vendor ID. A custom vendor ID like *apl0* would claim the stream was produced by the Apple encoder.

‘alpha_bits *integer*’
Specify number of bits for alpha component. Possible values are *0*, *8* and *16*. Use *0* to disable alpha plane coding.

13.3.2 Speed considerations (avconv.html#toc-Speed-considerations)

In the default mode of operation the encoder has to honor frame constraints (i.e. not produce frames with a size larger than requested) while still making the output picture as good as possible. A frame containing a lot of small details is harder to compress and the encoder would spend more time searching for appropriate quantizers for each slice.

Setting a higher ‘`bits_per_mb`’ limit will improve the speed.

For the fastest encoding speed set the ‘`qscale`’ parameter (4 is the recommended value) and do not set a size constraint.

13.4 libkvazaar (avconv.html#toc-libkvazaar)

Kvazaar H.265/HEVC encoder.

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

13.4.1 Options (avconv.html#toc-Options)

'b'
Set target video bitrate in bit/s and enable rate control.

'`kvazaar-params`'
Set kvazaar parameters as a list of *name*=*value* pairs separated by commas (,). See kvazaar documentation for a list of options.

13.5 QSV encoders (avconv.html#toc-QSV-encoders)

The family of Intel QuickSync Video encoders (MPEG-2, H.264 and HEVC)

The ratecontrol method is selected as follows:

- When '`global_quality`' is specified, a quality-based mode is used. Specifically this means either
  - - *CQP* - constant quantizer scale, when the '`qscale`' codec flag is also set (the '`-qscale`' avconv option).
  - - *LA_ICQ* - intelligent constant quality with lookahead, when the '`la_depth`' option is also set.
  - - *ICQ* – intelligent constant quality otherwise.
- Otherwise, a bitrate-based mode is used. For all of those, you should specify at least the desired average bitrate with the 'b' option.
  - - *LA* - VBR with lookahead, when the '`la_depth`' option is specified.
  - - *VCM* - video conferencing mode, when the '`vcm`' option is set.
  - - *CBR* - constant bitrate, when '`maxrate`' is specified and equal to the average bitrate.
  - - *VBR* - variable bitrate, when '`maxrate`' is specified, but is higher than the average bitrate.
  - - *AVBR* - average VBR mode, when '`maxrate`' is not specified. This mode is further configured by the '`avbr_accuracy`' and '`avbr_convergence`' options.

Note that depending on your system, a different mode than the one you specified may be selected by the encoder. Set the verbosity level to *verbose* or higher to see the actual settings used by the QSV runtime.

Additional libavcodec global options are mapped to MSDK options as follows:

- '`g/gop_size`' -> '`GopPicSize`'
- '`bf/max_b_frames`'+1 -> '`GopRefDist`'
- '`rc_init_occupancy/rc_initial_buffer_occupancy`' -> '`InitialDelayInKB`'
- '`slices`' -> '`NumSlice`'
- '`refs`' -> '`NumRefFrame`'
- '`b_strategy/b_frame_strategy`' -> '`BRefType`'
- '`cgop/CLOSED_GOP`' codec flag -> '`GopOptFlag`'
- For the *CQP* mode, the '`i_qfactor/i_qoffset`' and '`b_qfactor/b_qoffset`' set the difference between *QPP* and *QPI*, and *QPP* and *QPB* respectively.
- Setting the '`coder`' option to the value *vlc* will make the H.264 encoder use CAVLC instead of CABAC.

14. Demuxers (avconv.html#toc-Demuxers)

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

When you configure your Libav 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 av* tools will display the list of enabled demuxers.

The description of some of the currently available demuxers follows.

14.1 image2 (avconv.html#toc-image2-1)

Image file demuxer.

This demuxer reads from a list of image files specified by a pattern.

The 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 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 0 and 4, all the following numbers must be sequential. This limitation may be hopefully fixed.

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

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.

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

The following example shows how to use `avconv` for creating a video from the images in the file sequence ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., assuming an input framerate of 10 frames per second:

```
avconv -i 'img-%03d.jpeg' -r 10 out.mkv
```

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:

```
avconv -i img.jpeg img.png
```

‘`-pixel_format` *`format`*’
Set the pixel format (for raw image)

‘`-video_size` *`size`*’
Set the frame size (for raw image)

‘`-framerate` *`rate`*’
Set the frame rate

‘`-loop` *`bool`*’
Loop over the images

‘`-start_number` *`start`*’
Specify the first number in the sequence

14.2 applehttp (avconv.html#toc-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 avplay), 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".

14.3 flv (avconv.html#toc-flv)

Adobe Flash Video Format demuxer.

This demuxer is used to demux FLV files and RTMP network streams.

‘`-flv_metadata` *`bool`*’
Allocate the streams according to the onMetaData array content.

14.4 asf (avconv.html#toc-asf)

Advanced Systems Format demuxer.

This demuxer is used to demux ASF files and MMS network streams.

‘`-no_resync_search` *`bool`*’
Do not try to resynchronize by looking for a certain optional start code.

15. Muxers (avconv.html#toc-Muxers)

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

When you configure your Libav 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 av* tools will display the list of enabled muxers.

A description of some of the currently available muxers follows.

15.1 crc (avconv.html#toc-crc-1)

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`':

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

You can print the CRC to stdout with the command:

```
avconv -i INPUT -f crc -
```

You can select the output format of each frame with `avconv` 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:

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

See also the framecrc muxer.

15.2 framecrc (avconv.html#toc-framecrc-1)

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

This muxer computes and prints the Adler-32 CRC for each decoded audio and video frame. 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 frame of the form: *stream_index*, *frame_dts*, *frame_size*, 0x*CRC*, where *CRC* is a hexadecimal number 0-padded to 8 digits containing the CRC of the decoded frame.

For example to compute the CRC of each decoded frame in the input, and store it in the file '`out.crc`':

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

You can print the CRC of each decoded frame to stdout with the command:

```
avconv -i INPUT -f framecrc -
```

You can select the output format of each frame with `avconv` by specifying the audio and video codec and format. 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:

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

See also the crc muxer.

15.3 hls (avconv.html#toc-hls-1)

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.

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

'`-hls_time` *`seconds`*'
Set the segment length in seconds.

'`-hls_list_size` *`size`*'
Set the maximum number of playlist entries.

'`-hls_wrap` *`wrap`*'
Set the number after which index wraps.

'`-start_number` *`number`*'
Start the sequence from *number*.

'`-hls_base_url` *`baseurl`*'
Append *baseurl* to every entry in the playlist. Useful to generate playlists with absolute paths.

'`-hls_allow_cache` *`allowcache`*'
Explicitly set whether the client MAY (1) or MUST NOT (0) cache media segments

'`-hls_version` *`version`*'

Set the protocol version. Enables or disables version-specific features such as the integer (version 2) or decimal EXTINF values (version 3).

‘-hls_enc *enc*’
Enable (1) or disable (0) the AES128 encryption. When enabled every segment generated is encrypted and the encryption key is saved as *playlist name*.key.

‘-hls_enc_key *key*’
Use the specified hex-coded 16byte key to encrypt the segments, by default it is randomly generated.

‘-hls_enc_key_url *keyurl*’
If set, *keyurl* is prepended instead of *baseurl* to the key filename in the playlist.

‘-hls_enc_iv *iv*’
Use a specified hex-coded 16byte initialization vector for every segment instead of the autogenerated ones.

15.4 image2 (avconv.html#toc-image2-2)

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 `avconv` for creating a sequence of files ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., taking one image every second from the input video:

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

Note that with `avconv` , 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:

```
avconv -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:

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

‘-start_number *number*’
Start the sequence from *number*.

‘-update *number*’
If *number* is nonzero, the filename will always be interpreted as just a filename, not a pattern, and this file will be continuously overwritten with new images.

15.5 matroska (avconv.html#toc-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:

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

This muxer supports the following options:

‘reserve_index_space’
By default, this muxer writes the index for seeking (called cues in Matroska terms) at the end of the file, because it cannot know in advance how much space to leave for the index at the beginning of the file. However for some use cases – e.g. streaming where seeking is possible but slow – it is useful to put the index at the beginning of the file.

If this option is set to a non-zero value, the muxer will reserve a given amount of space in the file header and then try to write the cues there when the muxing finishes. If the available space does not suffice, muxing will fail. A safe size for most use cases should be about 50kB per hour of video.

Note that cues are only written if the output is seekable and this option will have no effect if it is not.

15.6 mov, mp4, ismv (avconv.html#toc-mov_002c-mp4_002c-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 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:

‘-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 `avconv`.)

'`-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.

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 index (moov atom) to the beginning 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.

'`-movflags disable_chpl`'
Disable Nero chapter markers (chpl atom). Normally, both Nero chapters and a QuickTime chapter track are written to the file. With this option set, only the QuickTime chapter track will be written. Nero chapters can cause failures when the file is reprocessed with certain tagging programs.

'`-movflags omit_tfhd_offset`'
Do not write any absolute base_data_offset in tfhd atoms. This avoids tying fragments to absolute byte positions in the file/streams.

'`-movflags default_base_moof`'
Similarly to the omit_tfhd_offset, this flag avoids writing the absolute base_data_offset field in tfhd atoms, but does so by using the new default-base-is-moof flag instead. This flag is new from 14496-12:2012. This may make the fragments easier to parse in certain circumstances (avoiding basing track fragment location calculations on the implicit end of the previous track fragment).

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

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

15.7 mp3 (avconv.html#toc-mp3)

The MP3 muxer writes a raw MP3 stream with the following optional features:

- An ID3v2 metadata header at the beginning (enabled by default). Versions 2.3 and 2.4 are supported, the `id3v2_version` private option controls which one is used (3 or 4). Setting `id3v2_version` to 0 disables the ID3v2 header completely.
  The muxer supports writing attached pictures (APIC frames) to the ID3v2 header. 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 (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.
- A Xing/LAME frame right after the ID3v2 header (if present). It is enabled by default, but will be written only if the output is seekable. The `write_xing` private option can be used to disable it. The frame contains various information that may be useful to the decoder, like the audio duration or encoder delay.
- A legacy ID3v1 tag at the end of the file (disabled by default). It may be enabled with the `write_id3v1` private option, but as its capabilities are very limited, its usage is not recommended.

Examples:

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

```
avconv -i INPUT -id3v2_version 3 -write_id3v1 1 out.mp3
```

Attach a picture to an mp3:

```
avconv -i input.mp3 -i cover.png -c copy -metadata:s:v title="Album cover"
-metadata:s:v comment="Cover (Front)" out.mp3
```

Write a "clean" MP3 without any extra features:

```
avconv -i input.wav -write_xing 0 -id3v2_version 0 out.mp3
```

15.8 mpegts (avconv.html#toc-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).

'-muxrate *number*'
Set a constant muxrate (default VBR).

'-pcr_period *numer*'
Override the default PCR retransmission time (default 20ms), ignored if variable muxrate is selected.

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

```
avconv -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
```

15.9 null (avconv.html#toc-null-1)

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 `avconv` you can use the command:

```
avconv -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 `avconv` syntax.

Alternatively you can write the command as:

```
avconv -benchmark -i INPUT -f null -
```

15.10 nut (avconv.html#toc-nut)

'-syncpoints *flags*'
Change the syncpoint usage in nut:

'*default* use the normal low-overhead seeking aids.'
'*none* do not use the syncpoints at all, reducing the overhead but making the stream non-seekable;'
'*timestamped* extend the syncpoint with a wallclock field.'

The *none* and *timestamped* flags are experimental.

```
avconv -i INPUT -f_strict experimental -syncpoints none - | processor
```

15.11 ogg (avconv.html#toc-ogg)

Ogg container muxer.

'-page_duration *duration*'
Preferred page duration, in microseconds. The muxer will attempt to create pages that are approximately *duration* microseconds long. This allows the user to compromise between seek granularity and container overhead. The default is 1 second. A value of 0 will fill all segments, making pages as large as possible. A value of 1 will effectively use 1 packet-per-page in most situations, giving a small seek granularity at the cost of additional container overhead.

'-serial_offset *value*'
Serial value from which to set the streams serial number. Setting it to different and sufficiently large values ensures that the produced ogg files can be safely chained.

15.12 segment (avconv.html#toc-segment)

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.

Every segment starts with a video keyframe, if a video stream is present. The segment muxer works best with a single constant frame rate video.

Optionally it can generate a flat list of the created segments, one segment per line.

'segment_format *format*'
Override the inner container format, by default it is guessed by the filename extension.

'segment_time *t*'
Set segment duration to *t* seconds.

'segment_list *name*'
Generate also a listfile named *name*.

'segment_list_type *type*'
Select the listing format.

'*flat* use a simple flat list of entries.'
'*hls* use a m3u8-like structure.'

'segment_list_size *size*'
Overwrite the listfile once it reaches *size* entries.

'segment_list_entry_prefix *prefix*'
Prepend *prefix* to each entry. Useful to generate absolute paths.

'segment_wrap *limit*'
Wrap around segment index once it reaches *limit*.

```
avconv -i in.mkv -c copy -map 0 -f segment -list out.list out%03d.nut
```

16. Input Devices (avconv.html#toc-Input-Devices)

Input devices are configured elements in Libav which allow to access the data coming from a multimedia device attached to your system.

When you configure your Libav 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 av* tools will display the list of supported input devices (amongst the demuxers).

A description of the currently available input devices follows.

16.1 alsa (avconv.html#toc-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 `avconv` from an ALSA device with card id 0, you may run the command:

```
avconv -f alsa -i hw:0 alsaout.wav
```

For more information see: http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html (http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html)

16.2 bktr (avconv.html#toc-bktr)

BSD video input device.

16.3 dv1394 (avconv.html#toc-dv1394)

Linux DV 1394 input device.

16.4 fbdev (avconv.html#toc-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 `avconv` :

```
avconv -f fbdev -r 10 -i /dev/fb0 out.avi
```

You can take a single screenshot image with the command:

```
avconv -f fbdev -frames:v 1 -r 1 -i /dev/fb0 screenshot.jpeg
```

See also http://linux-fbdev.sourceforge.net/ (http://linux-fbdev.sourceforge.net/), and fbset(1).

16.5 jack (avconv.html#toc-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 Libav 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 `avconv` .

```
# Create a JACK writable client with name "libav".
$ avconv -f jack -i libav -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
libav:input_1
metro:120_bpm

# Connect metro to the avconv writable client.
$ jack_connect metro:120_bpm libav:input_1
```

For more information read: http://jackaudio.org/ (http://jackaudio.org/)

16.6 libdc1394 (avconv.html#toc-libdc1394)

IIDC1394 input device, based on libdc1394 and libraw1394.

16.7 oss (avconv.html#toc-oss-1)

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 `avconv` use the command:

```
avconv -f oss -i /dev/dsp /tmp/oss.wav
```

For more information about OSS see: http://manuals.opensound.com/usersguide/dsp.html (http://manuals.opensound.com/usersguide/dsp.html)

16.8 pulse (avconv.html#toc-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`'.

```
avconv -f pulse -i default /tmp/pulse.wav
```

16.8.1 *server* AVOption (avconv.html#toc-server-AVOption)

The syntax is:

```
-server server name
```

Connects to a specific server.

16.8.2 *name* AVOption (avconv.html#toc-name-AVOption)

The syntax is:

```
-name application name
```

Specify the application name pulse will use when showing active clients, by default it is "libav"

16.8.3 *stream_name* AVOption (avconv.html#toc-stream_005fname-AVOption)

The syntax is:

```
-stream_name stream name
```

Specify the stream name pulse will use when showing active streams, by default it is "record"

16.8.4 *sample_rate* AVOption (avconv.html#toc-sample_005frate-AVOption)

The syntax is:

```
-sample_rate samplerate
```

Specify the samplerate in Hz, by default 48kHz is used.

16.8.5 *channels* AVOption (avconv.html#toc-channels-AVOption)

The syntax is:

```
-channels N
```

Specify the channels in use, by default 2 (stereo) is set.

16.8.6 *frame_size* AVOption (avconv.html#toc-frame_005fsize-AVOption)

The syntax is:

```
-frame_size bytes
```

Specify the number of byte per frame, by default it is set to 1024.

16.8.7 *fragment_size* AVOption (avconv.html#toc-fragment_005fsize-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.

16.9 sndio (avconv.html#toc-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 `avconv` use the command:

```
avconv -f sndio -i /dev/audio0 /tmp/oss.wav
```

16.10 video4linux2 (avconv.html#toc-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 *widthxheight* sizes and framerates. You can check which are supported using `-list_formats all` for Video4Linux2 devices.

Some usage examples of the video4linux2 devices with avconv and avplay:

```
# List supported formats for a video4linux2 device.
avplay -f video4linux2 -list_formats all /dev/video0

# Grab and show the input of a video4linux2 device.
avplay -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.
avconv -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
```

16.11 vfwcap (avconv.html#toc-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.

16.12 x11grab (avconv.html#toc-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 `avconv`:

```
avconv -f x11grab -r 25 -s cif -i :0.0 out.mpg

# Grab at position 10,20.
avconv -f x11grab -r 25 -s cif -i :0.0+10,20 out.mpg
```

16.12.1 *follow_mouse* AVOption (avconv.html#toc-follow_005fmouse-AVOption)

The syntax is:

```
-follow_mouse centered|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:

```
avconv -f x11grab -follow_mouse centered -r 25 -s cif -i :0.0 out.mpg

# Follows only when the mouse pointer reaches within 100 pixels to edge
avconv -f x11grab -follow_mouse 100 -r 25 -s cif -i :0.0 out.mpg
```

16.12.2 *show_region* AVOption (avconv.html#toc-show_005fregion-AVOption)

The syntax is:

```
-show_region 1
```

If *show_region* AVOption is specified with *1*, then the grabbing region will be indicated on screen. With this option, it's easy to know what is being grabbed if only a portion of the screen is grabbed.

For example:

```
avconv -f x11grab -show_region 1 -r 25 -s cif -i :0.0+10,20 out.mpg

# With follow_mouse
avconv -f x11grab -follow_mouse centered -show_region 1  -r 25 -s cif -i :0.0 out.mpg
```

16.12.3 *grab_x grab_y* AVOption (avconv.html#toc-grab_005fx-grab_005fy-AVOption)

The syntax is:

```
-grab_x x_offset -grab_y y_offset
```

Set the grabbing region coordinates. The are expressed as offset from the top left corner of the X11 window. The default value is 0.

17. Output Devices (avconv.html#toc-Output-Devices)

Output devices are configured elements in Libav which allow to write multimedia data to an output device attached to your system.

When you configure your Libav 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 av* tools will display the list of enabled output devices (amongst the muxers).

A description of the currently available output devices follows.

17.1 alsa (avconv.html#toc-alsa-1)

ALSA (Advanced Linux Sound Architecture) output device.

17.2 oss (avconv.html#toc-oss)

OSS (Open Sound System) output device.

17.3 sndio (avconv.html#toc-sndio-1)

sndio audio output device.

18. Protocols (avconv.html#toc-Protocols)

Protocols are configured elements in Libav which allow to access resources which require the use of a particular protocol.

When you configure your Libav 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 av* tools will display the list of supported protocols.

All protocols accept the following options:

‘rw_timeout’
Maximum time to wait for (network) read/write operations to complete, in microseconds.

A description of the currently available protocols follows.

18.1 concat (avconv.html#toc-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 `avplay` use the command:

```
avplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

Note that you may need to escape the character "|" which is special for many shells.

18.2 file (avconv.html#toc-file)

File access protocol.

Allow to read from or read to a file.

For example to read from a file ‘input.mpeg’ with `avconv` use the command:

```
avconv -i file:input.mpeg output.mpeg
```

The av* tools default to the file protocol, that is a resource specified with the name "FILE.mpeg" is interpreted as the URL "file:FILE.mpeg".

This protocol accepts the following options:

‘follow’
If set to 1, the protocol will retry reading at the end of the file, allowing reading files that still are being written. In order for this to terminate, you either need to use the rw_timeout option, or use the interrupt callback (for API users).

18.3 gopher (avconv.html#toc-gopher)

Gopher protocol.

18.4 hls (avconv.html#toc-hls-2)

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.

18.5 http (avconv.html#toc-http)

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options:

‘`chunked_post`’
If set to 1 use chunked Transfer-Encoding for posts, default is 1.

‘`content_type`’
Set a specific content type for the POST messages.

‘`headers`’
Set custom HTTP headers, can override built in default headers. The value must be a string encoding the headers.

‘`multiple_requests`’
Use persistent connections if set to 1, default is 0.

‘`post_data`’
Set custom HTTP post data.

‘`user_agent`’
Override the User-Agent header. If not specified a string of the form "Lavf/<version>" will be used.

‘`mime_type`’
Export the MIME type.

‘`icy`’
If set to 1 request ICY (SHOUTcast) metadata from the server. If the server supports this, the metadata has to be retrieved by the application by reading the ‘`icy_metadata_headers`’ and ‘`icy_metadata_packet`’ options. The default is 1.

‘`icy_metadata_headers`’
If the server supports ICY metadata, this contains the ICY-specific HTTP reply headers, separated by newline characters.

‘`icy_metadata_packet`’
If the server supports ICY metadata, and ‘`icy`’ was set to 1, this contains the last non-empty metadata packet sent by the server. It should be polled in regular intervals by applications interested in mid-stream metadata updates.

‘`offset`’
Set initial byte offset.

‘`end_offset`’
Try to limit the request to bytes preceding this offset.

18.6 Icecast (avconv.html#toc-Icecast)

Icecast (stream to Icecast servers)

This protocol accepts the following options:

‘`ice_genre`’
Set the stream genre.

‘`ice_name`’
Set the stream name.

‘`ice_description`’
Set the stream description.

‘`ice_url`’
Set the stream website URL.

‘`ice_public`’
Set if the stream should be public or not. The default is 0 (not public).

‘user_agent’
Override the User-Agent header. If not specified a string of the form "Lavf/<version>" will be used.

‘password’
Set the Icecast mountpoint password.

‘content_type’
Set the stream content type. This must be set if it is different from audio/mpeg.

‘legacy_icecast’
This enables support for Icecast versions < 2.4.0, that do not support the HTTP PUT method but the SOURCE method.

18.7 mmst (avconv.html#toc-mmst)

MMS (Microsoft Media Server) protocol over TCP.

18.8 mmsh (avconv.html#toc-mmsh)

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

```
mmsh://server[:port][/app][/playpath]
```

18.9 md5 (avconv.html#toc-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.
avconv -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
avconv -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.

18.10 pipe (avconv.html#toc-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 `avconv` :

```
cat test.wav | avconv -i pipe:0
# ...this is the same as...
cat test.wav | avconv -i pipe:
```

For writing to stdout with `avconv` :

```
avconv -i test.wav -f avi pipe:1 | cat > test.avi
# ...this is the same as...
avconv -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.

18.11 rtmp (avconv.html#toc-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://[username:password@]server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘username’
An optional username (mostly for publishing).

‘password’
An optional password (mostly for publishing).

‘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 `AVOption`s):

‘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. (When publishing, the default is FMLE/3.0 (compatible; <libavformat version>).)

‘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 `avplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
avplay rtmp://myserver/vod/sample
```

To publish to a password protected server, passing the playpath and app names separately:

```
avconv -re -i <input> -f flv -rtmp_playpath some/long/path -rtmp_app long/app/name rtmp://username:password@myserver/
```

18.12 rtmpe (avconv.html#toc-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.

18.13 rtmps (avconv.html#toc-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.

18.14 rtmpt (avconv.html#toc-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.

18.15 rtmpte (avconv.html#toc-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.

18.16 rtmpts (avconv.html#toc-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.

18.17 librtmp rtmp, rtmpe, rtmps, rtmpt, rtmpte (avconv.html#toc-librtmp-rtmp_002c-rtmpe_002c-rtmps_002c-rtmpt_002c-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 `avconv` :

```
avconv -re -i myfile -f flv rtmp://myserver/live/mystream
```

To play the same stream using `avplay` :

```
avplay "rtmp://myserver/live/mystream live=1"
```

18.18 rtp (avconv.html#toc-rtp)

Real-Time Protocol.

18.19 rtsp (avconv.html#toc-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 (http://github.com/revmischa/rtsp-server)).

The required syntax for a RTSP url is:

```
rtsp://hostname[:port]/path
```

The following options (set on the `avconv` / `avplay` 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 `avplay` , 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:

```
avplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

To watch a stream tunneled over HTTP:

```
avplay -rtsp_transport http rtsp://server/video.mp4
```

To send a stream in realtime to a RTSP server, for others to watch:

```
avconv -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

To receive a stream in realtime:

```
avconv -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

18.20 sap (avconv.html#toc-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.

18.20.1 Muxer (avconv.html#toc-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:

```
avconv -re -i input -f sap sap://224.0.0.255?same_port=1
```

Similarly, for watching in avplay:

```
avconv -re -i input -f sap sap://224.0.0.255
```

And for watching in avplay, over IPv6:

```
avconv -re -i input -f sap sap://[ff0e::1:2:3:4]
```

18.20.2 Demuxer (avconv.html#toc-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:

```
avplay sap://
```

To play back the first stream announced on one the default IPv6 SAP multicast address:

```
avplay sap://[ff0e::2:7ffe]
```

18.21 tcp (avconv.html#toc-tcp)

Transmission Control Protocol.

The required syntax for a TCP url is:

```
tcp://hostname:port[?options]
```

'listen'
: Listen for an incoming connection

```
avconv -i input -f format tcp://hostname:port?listen
avplay tcp://hostname:port
```

18.22 tls (avconv.html#toc-tls)

Transport Layer Security (TLS) / Secure Sockets Layer (SSL)

The required syntax for a TLS url is:

```
tls://hostname:port
```

The following parameters can be set via command line options (or in code via `AVOption`s):

‘`ca_file`’
A file containing certificate authority (CA) root certificates to treat as trusted. If the linked TLS library contains a default this might not need to be specified for verification to work, but not all libraries and setups have defaults built in.

‘`tls_verify=1|0`’
If enabled, try to verify the peer that we are communicating with. Note, if using OpenSSL, this currently only makes sure that the peer certificate is signed by one of the root certificates in the CA database, but it does not validate that the certificate actually matches the host name we are trying to connect to. (With GnuTLS, the host name is validated as well.)

This is disabled by default since it requires a CA database to be provided by the caller in many cases.

‘`cert_file`’
A file containing a certificate to use in the handshake with the peer. (When operating as server, in listen mode, this is more often required by the peer, while client certificates only are mandated in certain setups.)

‘`key_file`’
A file containing the private key for the certificate.

‘`listen=1|0`’
If enabled, listen for connections on the provided port, and assume the server role in the handshake instead of the client role.

18.23 udp (avconv.html#toc-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*. Follow the list of supported options.

‘`buffer_size=size`’
set the UDP buffer size in bytes

‘`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.

Some usage examples of the udp protocol with `avconv` follow.

To stream over UDP to a remote endpoint:

```
avconv -i input -f format udp://hostname:port
```

To stream in mpegts format over UDP using 188 sized UDP packets, using a large input buffer:

```
avconv -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

To receive over UDP from a remote endpoint:

```
avconv -i udp://[multicast-address]:port
```

18.24 unix (avconv.html#toc-unix)

Unix local socket

The required syntax for a Unix socket URL is:

```
unix://filepath
```

The following parameters can be set via command line options (or in code via `AVOption` s):

'timeout'
Timeout in ms.

'listen'
Create the Unix socket in listening mode.

19. Bitstream Filters (avconv.html#toc-Bitstream-Filters)

When you configure your Libav 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 av* 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.

19.1 aac_adtstoasc (avconv.html#toc-aac_005fadtstoasc)

19.2 chomp (avconv.html#toc-chomp)

19.3 dump_extradata (avconv.html#toc-dump_005fextradata)

19.4 extract_extradata (avconv.html#toc-extract_005fextradata)

Extract the in-band extradata.

Certain codecs allow the long-term headers (e.g. MPEG-2 sequence headers, or H.264/HEVC (VPS/)SPS/PPS) to be transmitted either "in-band" (i.e. as a part of the bitstream containing the coded frames) or "out of band" (e.g. on the container level). This latter form is called "extradata" in Libav terminology.

This bitstream filter detects the in-band headers and makes them available as extradata.

'remove'
When this option is enabled, the long-term headers are removed from the bitstream after extraction.

19.5 h264_mp4toannexb (avconv.html#toc-h264_005fmp4toannexb)

19.6 imx_dump_header (avconv.html#toc-imx_005fdump_005fheader)

19.7 mjpeg2jpeg (avconv.html#toc-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

```
avconv -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 (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.

```
avconv -i mjpeg-movie.avi -c:v copy -bsf:v mjpeg2jpeg frame_%d.jpg
exiftran -i -9 frame*.jpg
avconv -i frame_%d.jpg -c:v copy rotated.avi
```

19.8 mjpega_dump_header (avconv.html#toc-mjpega_005fdump_005fheader)

19.9 movsub (avconv.html#toc-movsub)

19.10 mp3_header_compress (avconv.html#toc-mp3_005fheader_005fcompress)

19.11 mp3_header_decompress (avconv.html#toc-mp3_005fheader_005fdecompress)

19.12 noise (avconv.html#toc-noise)

19.13 remove_extradata (avconv.html#toc-remove_005fextradata)

20. Filtergraph description (avconv.html#toc-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 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", and a filter with no output pads is called a "sink".

20.1 Filtergraph syntax (avconv.html#toc-Filtergraph-syntax-1)

A filtergraph has a textual representation, which is recognized by the '`-filter`'/'`-vf`' and '`-filter_complex`' options in `avconv` and '`-vf`' in `avplay` , and by the `avfilter_graph_parse()` / `avfilter_graph_parse2()` functions defined in '`libavfilter/avfilter.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. It may have one of two forms:

- A ':'-separated list of *key*=*value* pairs.
- A ':'-separated list of *value*. In this case, the keys are assumed to be the option names in the order they are declared. E.g. the `fade` filter declares three options in this order – '`type`', '`start_frame`' and '`nb_frames`'. Then the parameter list *in:0:30* means that the value *in* is assigned to the option '`type`', *0* to '`start_frame`' and *30* to '`nb_frames`'.

If the option value itself is a list of items (e.g. the `format` filter takes a list of pixel formats), the items in the list are usually separated by '|'.

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.

Here is a BNF description of the filtergraph syntax:

```
NAME             ::= sequence of alphanumeric characters and '_'
LINKLABEL        ::= "[" NAME "]"
LINKLABELS       ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (possibly quoted)
FILTER           ::= [LINKLABELS] NAME ["=" FILTER_ARGUMENTS] [LINKLABELS]
FILTERCHAIN      ::= FILTER [,FILTERCHAIN]
FILTERGRAPH      ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

21. Audio Filters (avconv.html#toc-Audio-Filters)

When you configure your Libav 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.

21.1 aformat (avconv.html#toc-aformat)

Convert the input audio to one of the specified formats. The framework will negotiate the most appropriate format to minimize conversions.

It accepts the following parameters:

‘`sample_fmts`’
A ’|’-separated list of requested sample formats.

‘`sample_rates`’
A ’|’-separated list of requested sample rates.

‘`channel_layouts`’
A ’|’-separated list of requested channel layouts.

If a parameter is omitted, all values are allowed.

Force the output to either unsigned 8-bit or signed 16-bit stereo

```
aformat=sample_fmts=u8|s16:channel_layouts=stereo
```

21.2 amix (avconv.html#toc-amix)

Mixes multiple audio inputs into a single output.

For example

```
avconv -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.

It accepts the following parameters:

‘`inputs`’
The number of inputs. If unspecified, it defaults to 2.

‘`duration`’
How to determine the end-of-stream.

‘longest’
The duration of the longest input. (default)

‘shortest’
The duration of the shortest input.

‘first’
The duration of the first input.

‘dropout_transition’
The transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

21.3 anull (avconv.html#toc-anull)

Pass the audio source unchanged to the output.

21.4 asetpts (avconv.html#toc-asetpts)

Change the PTS (presentation timestamp) of the input audio frames.

It accepts the following parameters:

‘expr’
The expression which is evaluated for each frame to construct its timestamp.

The expression is evaluated through the eval API and can contain the following constants:

‘FRAME_RATE’
frame rate, only defined for constant frame-rate video

‘PTS’
the presentation timestamp in input

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘N’
The number of audio samples passed through the filter so far, starting at 0.

‘S’
The number of audio samples in the current frame.

‘SR’
The audio sample rate.

‘STARTPTS’
The PTS of the first frame.

‘PREV_INPTS’
The previous input PTS.

‘PREV_OUTPTS’
The previous output PTS.

‘RTCTIME’
The wallclock (RTC) time in microseconds.

‘RTCSTART’
The wallclock (RTC) time at the start of the movie in microseconds.

Some examples:

```
# Start counting PTS from zero
asetpts=expr=PTS-STARTPTS

# Generate timestamps by counting samples
asetpts=expr=N/SR/TB

# Generate timestamps from a "live source" and rebase onto the current timebase
asetpts='(RTCTIME - RTCSTART) / (TB * 1000000)'
```

21.5 asettb (avconv.html#toc-asettb)

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

This filter accepts the following parameters:

‘expr’
The expression which is evaluated into the output timebase.

The expression can contain the constants *PI*, *E*, *PHI*, *AVTB* (the default timebase), *intb* (the input timebase), and *sr* (the sample rate, audio only).

The default value for the input is *intb*.

Some 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

# Set the timebase to twice the sample rate:
asettb=sr*2
```

21.6 ashowinfo (avconv.html#toc-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*.

It accepts the following parameters:

‘n’
The (sequential) number of the input frame, starting from 0.

‘pts’
The 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’
The presentation timestamp of the input frame in seconds.

‘fmt’
The sample format.

‘chlayout’
The channel layout.

‘rate’
The sample rate for the audio frame.

‘nb_samples’
The number of samples (per channel) in the frame.

‘checksum’
The 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.

21.7 asplit (avconv.html#toc-asplit)

Split input audio into several identical outputs.

It accepts a single parameter, which specifies the number of outputs. If unspecified, it defaults to 2.

For example,

```
avconv -i INPUT -filter_complex asplit=5 OUTPUT
```

will create 5 copies of the input audio.

21.8 asyncts (avconv.html#toc-asyncts)

Synchronize audio data with timestamps by squeezing/stretching it and/or dropping samples/adding silence when needed.

It accepts the following 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’
The minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. The default value is 0.1. If you get an imperfect sync with this filter, try setting this parameter to 0.

‘max_comp’
The maximum compensation in samples per second. Only relevant with compensate=1. The default value is 500.

‘first_pts’
Assume that the first PTS should be this value. The time base is 1 / sample rate. This allows for padding/trimming at the start of the 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 or to trim any samples with a negative PTS due to encoder delay.

21.9 atrim (avconv.html#toc-atrim)

Trim the input so that the output contains one continuous subpart of the input.

It accepts the following parameters:

‘start’
Timestamp (in seconds) of the start of the section to keep. I.e. the audio sample with the timestamp *start* will be the first sample in the output.

‘end’
Timestamp (in seconds) of the first audio sample that will be dropped. I.e. the audio sample immediately preceding the one with the timestamp *end* will be the last sample in the output.

‘start_pts’
Same as *start*, except this option sets the start timestamp in samples instead of seconds.

‘end_pts’
Same as *end*, except this option sets the end timestamp in samples instead of seconds.

‘duration’
The maximum duration of the output in seconds.

‘start_sample’
The number of the first sample that should be output.

‘end_sample’
The number of the first sample that should be dropped.

Note that the first two sets of the start/end options and the ‘`duration`’ option look at the frame timestamp, while the _sample options simply count the samples that pass through the filter. So start/end_pts and start/end_sample will give different results when the timestamps are wrong, inexact or do not start at zero. Also note that this filter does not modify the timestamps. If you wish to have the output timestamps start at zero, insert the asetpts filter after the atrim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all samples that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple atrim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- Drop everything except the second minute of input:
  ```
  avconv -i INPUT -af atrim=60:120
  ```
- Keep only the first 1000 samples:
  ```
  avconv -i INPUT -af atrim=end_sample=1000
  ```

21.10 bs2b (avconv.html#toc-bs2b)

Bauer stereo to binaural transformation, which improves headphone listening of stereo audio records.

It accepts the following parameters:

‘profile’
Pre-defined crossfeed level.

‘default’
Default level (fcut=700, feed=50).

‘cmoy’
Chu Moy circuit (fcut=700, feed=60).

‘jmeier’
Jan Meier circuit (fcut=650, feed=95).

‘fcut’
Cut frequency (in Hz).

‘feed’
Feed level (in Hz).

21.11 channelsplit (avconv.html#toc-channelsplit)

Split each channel from an input audio stream into a separate output stream.

It accepts the following parameters:

‘channel_layout’
The channel layout of the input stream. The default is "stereo".

For example, assuming a stereo input MP3 file,

```
avconv -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.

Split a 5.1 WAV file into per-channel files:

```
avconv -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]' low_frequency_effects.wav
-map '[SL]' side_left.wav -map '[SR]' side_right.wav
```

21.12 channelmap (avconv.html#toc-channelmap)

Remap input channels to new locations.

It accepts the following parameters:

‘channel_layout’
The channel layout of the output stream.

‘map’
Map channels from input to output. The argument is a ’|’-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 indices.

For example, assuming a 5.1+downmix input MOV file,

```
avconv -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

```
avconv -i in.wav -filter 'channelmap=1|2|0|5|3|4:5.1' out.wav
```

21.13 compand (avconv.html#toc-compand)

Compress or expand the audio’s dynamic range.

It accepts the following parameters:

‘attacks’
‘decays’
A list of times in seconds for each channel over which the instantaneous level of the input signal is averaged to determine its volume. *attacks* refers to increase of volume and *decays* refers to decrease of volume. For most situations, the attack time (response to the audio getting louder) should be shorter than the decay time, because the human ear is more sensitive to sudden loud audio than sudden soft audio. A typical value for attack is

0.3 seconds and a typical value for decay is 0.8 seconds.

'points'
A list of points for the transfer function, specified in dB relative to the maximum possible signal amplitude. Each key points list must be defined using the following syntax: `x0/y0|x1/y1|x2/y2|....`

The input values must be in strictly increasing order but the transfer function does not have to be monotonically rising. The point `0/0` is assumed but may be overridden (by `0/out-dBn`). Typical values for the transfer function are `-70/-70|-60/-20`.

'soft-knee'
Set the curve radius in dB for all joints. It defaults to 0.01.

'gain'
Set the additional gain in dB to be applied at all points on the transfer function. This allows for easy adjustment of the overall gain. It defaults to 0.

'volume'
Set an initial volume, in dB, to be assumed for each channel when filtering starts. This permits the user to supply a nominal level initially, so that, for example, a very large gain is not applied to initial signal levels before the companding has begun to operate. A typical value for audio which is initially quiet is -90 dB. It defaults to 0.

'delay'
Set a delay, in seconds. The input audio is analyzed immediately, but audio is delayed before being fed to the volume adjuster. Specifying a delay approximately equal to the attack/decay times allows the filter to effectively operate in predictive rather than reactive mode. It defaults to 0.

21.13.1 Examples (avconv.html#toc-Examples)

- Make music with both quiet and loud passages suitable for listening to in a noisy environment:
```
compand=.3|.3:1|1:-90/-60|-60/-40|-40/-30|-20/-20:6:0:-90:0.2
```
- A noise gate for when the noise is at a lower level than the signal:
```
compand=.1|.1:.2|.2:-900/-900|-50.1/-900|-50/-50:.01:0:-90:.1
```
- Here is another noise gate, this time for when the noise is at a higher level than the signal (making it, in some ways, similar to squelch):
```
compand=.1|.1:.1|.1:-45.1/-45.1|-45/-900|0/-900:.01:45:-90:.1
```

21.14 join (avconv.html#toc-join)

Join multiple input streams into one multi-channel stream.

It accepts the following parameters:

'inputs'
The number of input streams. It defaults to 2.

'channel_layout'
The desired output channel layout. It defaults to stereo.

'map'
Map channels from inputs to output. The argument is a '|'-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 they 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.

Join 3 inputs (with properly set channel layouts):

```
avconv -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

Build a 5.1 output from 6 single-channel streams:

```
avconv -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
```

21.15 hdcd (avconv.html#toc-hdcd)

Decodes High Definition Compatible Digital (HDCD) data. A 16-bit PCM stream with embedded HDCD codes is expanded into a 20-bit PCM stream.

The filter supports the Peak Extend and Low-level Gain Adjustment features of HDCD, and detects the Transient Filter flag.

```
avconv -i HDCD16.flac -af hdcd OUT24.flac
```

When using the filter with WAV, note that the default encoding for WAV is 16-bit, so the resulting 20-bit stream will be truncated back to 16-bit. Use something like `-acodec pcm_s24le` after the filter to get 24-bit PCM output.

```
avconv -i HDCD16.wav -af hdcd OUT16.wav
avconv -i HDCD16.wav -af hdcd -acodec pcm_s24le OUT24.wav
```

The filter accepts the following options:

'analyze_mode'
Replace audio with a solid tone and adjust the amplitude to signal some specific aspect of the decoding process. The output file can be loaded in an audio editor alongside the original to aid analysis.

Modes are:

'0, off'
Disabled

'1, lle'
Gain adjustment level at each sample

'2, pe'
Samples where peak extend occurs

'3, cdt'
Samples where the code detect timer is active

'4, tgm'
Samples where the target gain does not match between channels

'5, pel'
Any samples above peak extend level

'6, ltgm'
Gain adjustment level at each sample, in each channel

21.16 resample (avconv.html#toc-resample)

Convert the audio sample format, sample rate and channel layout. It is not meant to be used directly; it is inserted automatically by libavfilter whenever conversion is needed. Use the *aformat* filter to force a specific conversion.

21.17 volume (avconv.html#toc-volume)

Adjust the input audio volume.

It accepts the following parameters:

'volume'
This expresses how the audio volume will be increased or decreased.

Output values are clipped to the maximum value.

The output audio volume is given by the relation:

```
output_volume = volume * input_volume
```

The default value for *volume* is 1.0.

'precision'
This parameter represents the mathematical precision.

It determines which input sample formats will be allowed, which affects the precision of the volume scaling.

'fixed'
8-bit fixed-point; this limits input sample format to U8, S16, and S32.

'float'
32-bit floating-point; this limits input sample format to FLT. (default)

'double'
64-bit floating-point; this limits input sample format to DBL.

'replaygain'
Choose the behaviour on encountering ReplayGain side data in input frames.

'drop'

Remove ReplayGain side data, ignoring its contents (the default).

‘ignore’
Ignore ReplayGain side data, but leave it in the frame.

‘track’
Prefer the track gain, if present.

‘album’
Prefer the album gain, if present.

‘replaygain_preamp’
Pre-amplification gain in dB to apply to the selected replaygain gain.

Default value for *replaygain_preamp* is 0.0.

‘replaygain_noclip’
Prevent clipping by limiting the gain applied.

Default value for *replaygain_noclip* is 1.

21.17.1 Examples (avconv.html#toc-Examples-2)

- Halve the input audio volume:
```
volume=volume=0.5
volume=volume=1/2
volume=volume=-6.0206dB
```
- Increase input audio power by 6 decibels using fixed-point precision:
```
volume=volume=6dB:precision=fixed
```

22. Audio Sources (avconv.html#toc-Audio-Sources)

Below is a description of the currently available audio sources.

22.1 anullsrc (avconv.html#toc-anullsrc)

The null audio source; it never returns audio frames. It is mainly useful as a template and for use in analysis / debugging tools.

It accepts, as an optional parameter, a string of the form *sample_rate*:*channel_layout*.

sample_rate specifies the sample rate, and defaults to 44100.

channel_layout specifies the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is 3, which corresponds to CH_LAYOUT_STEREO.

Check the channel_layout_map definition in ‘`libavutil/channel_layout.c`’ for the mapping between strings and channel layout values.

Some examples:

```
# Set the sample rate to 48000 Hz and the channel layout to CH_LAYOUT_MONO
anullsrc=48000:4

# The same as above
anullsrc=48000:mono
```

22.2 abuffer (avconv.html#toc-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; it is for insertion by calling programs, through the interface defined in ‘`libavfilter/buffersrc.h`’.

It accepts the following parameters:

‘time_base’
The 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’
The audio sample rate.

‘sample_fmt’
The name of the sample format, as returned by `av_get_sample_fmt_name()`.

‘channel_layout’

The 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.

23. Audio Sinks (avconv.html#toc-Audio-Sinks)

Below is a description of the currently available audio sinks.

23.1 anullsink (avconv.html#toc-anullsink)

Null audio sink; do absolutely nothing with the input audio. It is mainly useful as a template and for use in analysis / debugging tools.

23.2 abuffersink (avconv.html#toc-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`'.

It does not accept any parameters.

24. Video Filters (avconv.html#toc-Video-Filters)

When you configure your Libav 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.

24.1 blackframe (avconv.html#toc-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.

It accepts the following parameters:

'`amount`'
The percentage of the pixels that have to be below the threshold; it defaults to 98.

'`threshold`'
The threshold below which a pixel value is considered black; it defaults to 32.

24.2 boxblur (avconv.html#toc-boxblur)

Apply a boxblur algorithm to the input video.

It accepts the following parameters:

'`luma_radius`'
'`luma_power`'
'`chroma_radius`'
'`chroma_power`'
'`alpha_radius`'
'`alpha_power`'

The 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`'
The 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:

- Apply a boxblur filter with the luma, chroma, and alpha radii set to 2:

```
boxblur=luma_radius=2:luma_power=1
```

- Set the luma radius to 2, and alpha and chroma radius to 0:

```
boxblur=2:1:0:0:0:0
```

- Set the luma and chroma radii to a fraction of the video dimension:

```
boxblur=luma_radius=min(h\,w)/10:luma_power=1:chroma_radius=min(cw\,ch)/10:chroma_power=1
```

24.3 copy (avconv.html#toc-copy)

Copy the input source unchanged to the output. This is mainly useful for testing purposes.

24.4 crop (avconv.html#toc-crop)

Crop the input video to given dimensions.

It accepts the following parameters:

‘out_w’
The width of the output video.

‘out_h’
The height of the output video.

‘x’
The horizontal position, in the input video, of the left edge of the output video.

‘y’
The vertical position, in the input video, of the top edge of the output video.

The parameters are expressions containing the following constants:

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘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’
These are the same as *in_w* and *in_h*.

‘out_w, out_h’
The output (cropped) width and height.

‘ow, oh’
These are the same as *out_w* and *out_h*.

‘n’
The number of the input frame, starting from 0.

‘t’
The timestamp expressed in seconds. It’s 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 only evaluated during 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*.

Some examples:

```
# Crop the central input area with size 100x100
crop=out_w=100:out_h=100

# Crop the central input area with size 2/3 of the input video
"crop=out_w=2/3*in_w:out_h=2/3*in_h"

# Crop the input video central square
crop=out_w=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=out_w=in_w-100:out_h=in_h-100:x=100:y=100

# Crop 10 pixels from the left and right borders, and 20 pixels from
# the top and bottom borders
"crop=out_w=in_w-2*10:out_h=in_h-2*20"

# Keep only the bottom right quarter of the input image
"crop=out_w=in_w/2:out_h=in_h/2:x=in_w/2:y=in_h/2"

# Crop height for getting Greek harmony
"crop=out_w=in_w:out_h=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=out_w=in_w/2:out_h=in_h/2:x=(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):y=(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)"
```

24.5 cropdetect (avconv.html#toc-cropdetect)

Auto-detect the crop size.

It calculates the necessary cropping parameters and prints the recommended parameters via the logging system. The detected dimensions correspond to the non-black area of the input video.

It accepts the following parameters:

‘limit’
: The threshold, an optional parameter between nothing (0) and everything (255). It defaults to 24.

‘round’
: The value which the width/height should be divisible by. It 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’
: A counter that determines how many frames cropdetect will reset the previously detected largest video area after. It will then start over and detect the current optimal crop area. It defaults to 0.

  This can be useful when channel logos distort the video area. 0 indicates ’never reset’, and returns the largest area encountered during playback.

24.6 delogo (avconv.html#toc-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).

It accepts the following parameters:

‘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.

An example:

- Set a rectangle covering the area with top left corner coordinates 0,0 and size 100x77, and a band of size 10:

```
delogo=x=0:y=0:w=100:h=77:band=10
```

24.7 drawbox (avconv.html#toc-drawbox)

Draw a colored box on the input image.

It accepts the following parameters:

‘x, y’
Specify the top left corner coordinates of the box. It defaults to 0.

‘width, height’
Specify the width and height of the box; if 0 they are interpreted as the input width and height. It defaults to 0.

‘color’
Specify the color of the box to write. It can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence.

Some examples:

```
# Draw a black box around the edge of the input image
drawbox

# Draw a box with color red and an opacity of 50%
drawbox=x=10:y=20:width=200:height=60:color=red@0.5"
```

24.8 drawtext (avconv.html#toc-drawtext)

Draw a text string or text from a specified file on top of a video, using the libfreetype library.

To enable compilation of this filter, you need to configure Libav with `--enable-libfreetype`. To enable default font fallback and the *font* option you need to configure Libav with `--enable-libfontconfig`.

The filter also recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime().

It accepts the following parameters:

‘font’
The font family to be used for drawing text. By default Sans.

‘fontfile’
The font file to be used for drawing text. The path must be included. This parameter is mandatory if the fontconfig support is disabled.

‘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 offsets where text will be drawn within the video frame. It is relative to the top/left border of the output image. They accept expressions similar to the overlay filter:

‘x, y’
The computed values for *x* and *y*. They are evaluated for each new frame.

‘main_w, main_h’
The main input width and height.

‘W, H’
These are the same as *main_w* and *main_h*.

‘text_w, text_h’
The rendered text’s width and height.

‘w, h’
These are the same as *text_w* and *text_h*.

‘n’
The number of frames processed, starting from 0.

‘t’
The timestamp, expressed in seconds. It’s NAN if the input timestamp is unknown.

The default value of *x* and *y* is 0.

‘draw’
Draw the text only if the expression evaluates as non-zero. The expression accepts the same variables *x, y* do. The default value is 1.

‘alpha’
Draw the text applying alpha blending. The value can be either a number between 0.0 and 1.0 The expression accepts the same variables *x, y* do. The default value is 1.

‘fontsize’
The font size to be used for drawing text. The default value of *fontsize* is 16.

‘fontcolor’
The color to be used for drawing fonts. It is 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".

‘boxcolor’
The color to be used for drawing box around text. It is 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".

‘box’
Used to draw a box around text using the background color. The value must be either 1 (enable) or 0 (disable). The default value of *box* is 0.

‘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. The default value for both is "0".

‘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".

‘ft_load_flags’
The 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.

‘tabsize’
The size in number of spaces to use for rendering the tab. Default value is 4.

‘fix_bounds’
If true, check and fix text coords to avoid clipping.

For example the command:

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
```

will draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

The command:

```
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"
```

will 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%.

Note that the double quotes are not necessary if spaces are not used within the parameter list.

For more information about libfreetype, check: http://www.freetype.org/ (http://www.freetype.org/).

24.9 fade (avconv.html#toc-fade)

Apply a fade-in/out effect to the input video.

It accepts the following parameters:

'type'
The effect type can be either "in" for a fade-in, or "out" for a fade-out effect.

'start_frame'
The number of the frame to start applying the fade effect at.

'nb_frames'
The number of frames that the fade effect lasts. 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.

Some examples:

```
# Fade in the first 30 frames of video
fade=type=in:nb_frames=30

# Fade out the last 45 frames of a 200-frame video
fade=type=out:start_frame=155:nb_frames=45

# Fade in the first 25 frames and fade out the last 25 frames of a 1000-frame video
fade=type=in:start_frame=0:nb_frames=25, fade=type=out:start_frame=975:nb_frames=25

# Make the first 5 frames black, then fade in from frame 5-24
fade=type=in:start_frame=5:nb_frames=20
```

24.10 fieldorder (avconv.html#toc-fieldorder)

Transform the field order of the input video.

It accepts the following parameters:

'order'
The output field order. Valid values are *tff* for top field first or *bff* for bottom field first.

The default value is "tff".

The transformation is done 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.

It is very useful when converting to or from PAL DV material, which is bottom field first.

For example:

```
./avconv -i in.vob -vf "fieldorder=order=bff" out.dv
```

24.11 fifo (avconv.html#toc-fifo)

Buffer input images and send them when they are requested.

It is mainly useful when auto-inserted by the libavfilter framework.

It does not take parameters.

24.12 format (avconv.html#toc-format)

Convert the input video to one of the specified pixel formats. Libavfilter will try to pick one that is suitable as input to the next filter.

It accepts the following parameters:

‘pix_fmts’
A ’|’-separated list of pixel format names, such as "pix_fmts=yuv420p|monow|rgb24".

Some examples:

```
# Convert the input video to the "yuv420p" format
format=pix_fmts=yuv420p

# Convert the input video to any of the formats in the list
format=pix_fmts=yuv420p|yuv444p|yuv410p
```

24.13 fps (avconv.html#toc-fps-1)

Convert the video to specified constant framerate by duplicating or dropping frames as necessary.

It accepts the following parameters:

‘fps’
The desired output framerate.

‘start_time’
Assume the first PTS should be the given value, in seconds. 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 duplicates of the first frame if a video stream starts after the audio stream or to trim any frames with a negative PTS.

24.14 framepack (avconv.html#toc-framepack)

Pack two different video streams into a stereoscopic video, setting proper metadata on supported codecs. The two views should have the same size and framerate and processing will stop when the shorter video ends. Please note that you may conveniently adjust view properties with the scale and fps filters.

It accepts the following parameters:

‘format’
The desired packing format. Supported values are:

‘sbs’
The views are next to each other (default).

‘tab’
The views are on top of each other.

‘lines’
The views are packed by line.

‘columns’
The views are packed by column.

‘frameseq’
The views are temporally interleaved.

Some examples:

```
# Convert left and right views into a frame-sequential video
avconv -i LEFT -i RIGHT -filter_complex framepack=frameseq OUTPUT

# Convert views into a side-by-side video with the same output resolution as the input
avconv -i LEFT -i RIGHT -filter_complex [0:v]scale=w=iw/2[left],[1:v]scale=w=iw/2[right],[left][right]framepack=sbs OUTPUT
```

24.15 frei0r (avconv.html#toc-frei0r-1)

Apply a frei0r effect to the input video.

To enable the compilation of this filter, you need to install the frei0r header and configure Libav with –enable-frei0r.

It accepts the following parameters:

‘filter_name’
The name of the frei0r effect to load. If the environment variable `FREI0R_PATH` is defined, the frei0r effect is searched for in each of the directories specified by the colon-separated list in `FREI0R_PATH`. Otherwise, the standard frei0r paths are searched, in this order: ‘`HOME/.frei0r-1/lib/`’, ‘`/usr/local/lib/frei0r-1/`’, ‘`/usr/lib/frei0r-1/`’.

‘filter_params’

A '|'-separated list of parameters to pass to the frei0r effect.

A frei0r effect parameter can be a boolean (its value is either "y" or "n"), a double, a color (specified as *R*/*G*/*B*, where *R*, *G*, and *B* are floating point numbers between 0.0 and 1.0, inclusive) or by an `av_parse_color()` color description), a position (specified as *X*/*Y*, where *X* and *Y* are floating point numbers) and/or a string.

The number and types of parameters depend on the loaded effect. If an effect parameter is not specified, the default value is set.

Some examples:

```
# Apply the distort0r effect, setting the first two double parameters
frei0r=filter_name=distort0r:filter_params=0.5|0.01

# Apply the colordistance effect, taking a color as the first parameter
frei0r=colordistance:0.2/0.3/0.4
frei0r=colordistance:violet
frei0r=colordistance:0x112233

# Apply the perspective effect, specifying the top left and top right
# image positions
frei0r=perspective:0.2/0.2|0.8/0.2
```

For more information, see http://piksel.org/frei0r (http://piksel.org/frei0r)

24.16 gradfun (avconv.html#toc-gradfun)

Fix the banding artifacts that are sometimes introduced into nearly flat regions by truncation to 8-bit colordepth. Interpolate the gradients that should go where the bands are, and dither them.

It 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.

It accepts the following parameters:

'`strength`'
The maximum amount by which the filter will change any one pixel. This is also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 64; the default value is 1.2. Out-of-range values will be clipped to the valid range.

'`radius`'
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; the default value is 16. Out-of-range values will be clipped to the valid range.

```
# Default parameters
gradfun=strength=1.2:radius=16

# Omitting the radius
gradfun=1.2
```

24.17 hflip (avconv.html#toc-hflip)

Flip the input video horizontally.

For example, to horizontally flip the input video with `avconv` :

```
avconv -i in.avi -vf "hflip" out.avi
```

24.18 hqdn3d (avconv.html#toc-hqdn3d)

This is a high precision/quality 3d denoise filter. It 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`'
A non-negative floating point number which specifies spatial luma strength. It defaults to 4.0.

'`chroma_spatial`'
A non-negative floating point number which specifies spatial chroma strength. It defaults to 3.0**luma_spatial*/4.0.

'`luma_tmp`'
A floating point number which specifies luma temporal strength. It defaults to 6.0**luma_spatial*/4.0.

'`chroma_tmp`'
A floating point number which specifies chroma temporal strength. It defaults to *luma_tmp***chroma_spatial*/*luma_spatial*.

24.19 hwupload_cuda (avconv.html#toc-hwupload_005fcuda)

Upload system memory frames to a CUDA device.

It accepts the following optional parameters:

‘device’
The number of the CUDA device to use

24.20 interlace (avconv.html#toc-interlace)

Simple interlacing filter from progressive contents. This interleaves upper (or lower) lines from odd frames with lower (or upper) lines from even frames, halving the frame rate and preserving image height.

```
   Original        Original             New Frame
   Frame 'j'      Frame 'j+1'             (tff)
  ==========      ===========       ==================
    Line 0  --------------------->    Frame 'j' Line 0
    Line 1          Line 1  ---->   Frame 'j+1' Line 1
    Line 2 --------------------->    Frame 'j' Line 2
    Line 3          Line 3  ---->   Frame 'j+1' Line 3
     ...             ...                 ...
New Frame + 1 will be generated by Frame 'j+2' and Frame 'j+3' and so on
```

It accepts the following optional parameters:

‘scan’
This determines whether the interlaced frame is taken from the even (tff - default) or odd (bff) lines of the progressive frame.

‘lowpass’
Enable (default) or disable the vertical lowpass filter to avoid twitter interlacing and reduce moire patterns.

24.21 lut, lutrgb, lutyuv (avconv.html#toc-lut_002c-lutrgb_002c-lutyuv)

Compute a look-up table for binding each pixel component input value to an output value, and apply it to the input video.

lutyuv applies a lookup table to a YUV input video, *lutrgb* to an RGB input video.

These filters accept the following parameters:

‘*c0* (first pixel component)’
‘*c1* (second pixel component)’
‘*c2* (third pixel component)’
‘*c3* (fourth pixel component, corresponds to the alpha component)’
‘*r* (red component)’
‘*g* (green component)’
‘*b* (blue component)’
‘*a* (alpha component)’
‘*y* (Y/luminance component)’
‘*u* (U/Cb component)’
‘*v* (V/Cr component)’

Each of them specifies the expression to use for computing the lookup table for the corresponding pixel component values.

The exact component associated to each of the *c** options depends on the format in input.

The *lut* filter requires either YUV or RGB pixel formats in input, *lutrgb* requires RGB pixel formats in input, and *lutyuv* requires YUV.

The expressions can contain the following constants and functions:

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘w, h’
The input width and height.

‘val’
The input value for the pixel component.

‘clipval’
The input value, clipped to the *minval*-*maxval* range.

‘maxval’
The maximum value for the pixel component.

‘minval’

The minimum value for the pixel component.

‘negval’
The negated value for the pixel component value, clipped to the *minval-maxval* range; it corresponds to the expression "maxval-clipval+minval".

‘clip(val)’
The computed value in *val*, clipped to the *minval-maxval* range.

‘gammaval(gamma)’
The computed gamma correction value of the pixel component value, clipped to the *minval-maxval* range. It corresponds to the expression "pow((clipval-minval)/(maxval-minval)\,*gamma*)*(maxval-minval)+minval"

All expressions default to "val".

Some examples:

```
# 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=negval

# Remove chroma components, turning 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 factor of 0.5
lutyuv=y=gammaval(0.5)
```

24.22 negate (avconv.html#toc-negate)

Negate input video.

It accepts an integer in input; if non-zero it negates the alpha component (if available). The default value in input is 0.

24.23 noformat (avconv.html#toc-noformat)

Force libavfilter not to use any of the specified pixel formats for the input to the next filter.

It accepts the following parameters:

‘pix_fmts’
A ’|’-separated list of pixel format names, such as apix_fmts=yuv420p|monow|rgb24".

Some examples:

```
# Force libavfilter to use a format different from "yuv420p" for the
# input to the vflip filter
noformat=pix_fmts=yuv420p,vflip

# Convert the input video to any of the formats not contained in the list
noformat=yuv420p|yuv444p|yuv410p
```

24.24 null (avconv.html#toc-null)

Pass the video source unchanged to the output.

24.25 ocv (avconv.html#toc-ocv)

Apply a video transform using libopencv.

To enable this filter, install the libopencv library and headers and configure Libav with –enable-libopencv.

It accepts the following parameters:

'filter_name'
The name of the libopencv filter to apply.

'filter_params'
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 (http://opencv.willowgarage.com/documentation/c/image_filtering.html)

Several libopencv filters are supported; see the following subsections.

24.25.1 dilate (avconv.html#toc-dilate-1)

Dilate an image by using a specific structuring element. It corresponds to the libopencv function `cvDilate`.

It accepts the parameters: *struct_el|nb_iterations*.

struct_el represents a structuring element, and has the syntax: *colsxrows+anchor_xxanchor_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. *shape* must be "rect", "cross", "ellipse", or "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.

Some examples:

```
# Use the default values
ocv=dilate

# Dilate using a structuring element with a 5x5 cross, iterating two times
ocv=filter_name=dilate:filter_params=5x5+2x2/cross|2

# Read the shape from the file diamond.shape, iterating two times.
# The file diamond.shape may contain a pattern of characters like this
#   *
#  ***
# *****
#  ***
#   *
# The specified columns and rows are ignored
# but the anchor point coordinates are not
ocv=dilate:0x0+2x2/custom=diamond.shape|2
```

24.25.2 erode (avconv.html#toc-erode)

Erode an image by using a specific structuring element. It corresponds to the libopencv function `cvErode`.

It accepts the parameters: *struct_el:nb_iterations*, with the same syntax and semantics as the dilate filter.

24.25.3 smooth (avconv.html#toc-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 must be one of the following values: "blur", "blur_no_scale", "median", "gaussian", or "bilateral". The default value is "gaussian".

The meaning of *param1*, *param2*, *param3*, and *param4* depend on the smooth type. *param1* and *param2* accept integer positive values or 0. *param3* and *param4* accept floating point 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`.

24.26 overlay (avconv.html#toc-overlay-1)

Overlay one video on top of another.

It takes two inputs and has one output. The first input is the "main" video on which the second input is overlaid.

It accepts the following parameters:

‘x’
The horizontal position of the left edge of the overlaid video on the main video.

‘y’
The vertical position of the top edge of the overlaid video on the main video.

The parameters are expressions containing the following parameters:

‘main_w, main_h’
The main input width and height.

‘W, H’
These are the same as *main_w* and *main_h*.

‘overlay_w, overlay_h’
The overlay input width and height.

‘w, h’
These are the same as *overlay_w* and *overlay_h*.

‘eof_action’
The action to take when EOF is encountered on the secondary input; it accepts one of the following values:

‘repeat’
Repeat the last frame (the default).

‘endall’
End both streams.

‘pass’
Pass the main input through.

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 the example for the *movie* filter does.

Some examples:

```
# Draw the overlay at 10 pixels from the bottom right
# corner of the main video
overlay=x=main_w-overlay_w-10:y=main_h-overlay_h-10

# Insert a transparent PNG logo in the bottom left corner of the input
avconv -i input -i logo -filter_complex 'overlay=x=10:y=main_h-overlay_h-10' output

# Insert 2 different transparent PNG logos (second logo on bottom
# right corner)
avconv -i input -i logo1 -i logo2 -filter_complex
'overlay=x=10:y=H-h-10,overlay=x=W-w-10:y=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]

# Mask 10-20 seconds of a video by applying the delogo filter to a section
avconv -i test.avi -codec:v:0 wmv2 -ar 11025 -b:v 9000k
-vf '[in]split[split_main][split_delogo];[split_delogo]trim=start=360:end=371,delogo=0:0:640:480[delogoed];[split_main][delogoed]overlay=e
masked.avi
```

You can chain together more overlays but the efficiency of such approach is yet to be tested.

24.27 pad (avconv.html#toc-pad)

Add paddings to the input image, and place the original input at the provided *x*, *y* coordinates.

It accepts the following 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 to place the input image at within 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 an 0xRRGGBB[AA] sequence.

The default value of *color* is "black".

The parameters *width*, *height*, *x*, and *y* are expressions containing the following constants:

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘in_w, in_h’
The input video width and height.

‘iw, ih’
These are the same as *in_w* and *in_h*.

‘out_w, out_h’
The output width and height (the size of the padded area), as specified by the *width* and *height* expressions.

‘ow, oh’
These are the same as *out_w* and *out_h*.

‘x, y’
The x and y offsets as specified by the *x* and *y* expressions, or NAN if not yet specified.

‘a’
The input display aspect ratio, same as *iw* / *ih*.

‘hsub, vsub’
The horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

Some examples:

```
# Add paddings with the color "violet" to the input video. The output video
# size is 640x480, and the top-left corner of the input video is placed at
# column 0, row 40
pad=width=640:height=480:x=0:y=40:color=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"

# Double the 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"
```

24.28 pixdesctest (avconv.html#toc-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.

24.29 scale (avconv.html#toc-scale-1)

Scale the input video and/or convert the image format.

It accepts the following parameters:

‘w’
The output video width.

‘h’
The output video height.

The parameters *w* and *h* are expressions containing the following constants:

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘in_w, in_h’
The input width and height.

‘iw, ih’
These are the same as *in_w* and *in_h*.

‘out_w, out_h’
The output (cropped) width and height.

‘ow, oh’
These are the same as *out_w* and *out_h*.

‘a’
This is the same as *iw* / *ih*.

‘sar’
input sample aspect ratio

‘dar’
The input display aspect ratio; it is the same as (*iw* / *ih*) * *sar*.

‘hsub, vsub’
The 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 *w* or *h* is 0, the respective input size is used for the output.

If the value for *w* or *h* is -1, the scale filter will use, for the respective output size, a value that maintains the aspect ratio of the input image.

The default value of *w* and *h* is 0.

Some examples:

```
# Scale the input video to a size of 200x100
scale=w=200:h=100

# Scale the input to 2x
scale=w=2*iw:h=2*ih
# The above is the same as
scale=2*in_w:2*in_h

# Scale the input to half the original size
scale=w=iw/2:h=ih/2

# Increase the width, and set the height to the same size
scale=3/2*iw:ow

# Seek 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=w=3/2*oh:h=3/5*ih

# Increase the size, making 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,
# keeping the same aspect ratio as the input
scale=w='min(500\, iw*3/2):h=-1'
```

24.30 scale_npp (avconv.html#toc-scale_005fnpp)

Use the NVIDIA Performance Primitives (libnpp) to perform scaling and/or pixel format conversion on CUDA video frames. Setting the output width and height works in the same way as for the *scale* filter.

The following additional options are accepted:

‘format’
The pixel format of the output CUDA frames. If set to the string "same" (the default), the input format will be kept. Note that automatic format negotiation and conversion is not yet supported for hardware frames

‘interp_algo’

The interpolation algorithm used for resizing. One of the following:

‘nn’
Nearest neighbour.

‘linear’
‘cubic’
‘cubic2p_bspline’
2-parameter cubic (B=1, C=0)

‘cubic2p_catmullrom’
2-parameter cubic (B=0, C=1/2)

‘cubic2p_b05c03’
2-parameter cubic (B=1/2, C=3/10)

‘super’
Supersampling

‘lanczos’

24.31 select (avconv.html#toc-select)

Select frames to pass in output.

It accepts the following parameters:

‘expr’
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:

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘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. It’s NAN if undefined.

‘TB’
The timebase of the input timestamps.

‘pts’
The PTS (Presentation TimeStamp) of the filtered video frame, expressed in *TB* units. It’s NAN if undefined.

‘t’
The PTS of the filtered video frame, expressed in seconds. It’s NAN if undefined.

‘prev_pts’
The PTS of the previously filtered video frame. It’s NAN if undefined.

‘prev_selected_pts’
The PTS of the last previously filtered video frame. It’s NAN if undefined.

‘prev_selected_t’
The PTS of the last previously selected video frame. It’s NAN if undefined.

‘start_pts’
The PTS of the first video frame in the video. It’s NAN if undefined.

‘start_t’
The time of the first video frame in the video. It’s NAN if undefined.

‘pict_type’
The type of the filtered frame. It can assume one of the following values:

‘I’
‘P’
‘B’
‘S’
‘SI’

‘SP’
‘BI’

‘interlace_type’
The frame interlace type. It 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’
This is 1 if the filtered frame is a key-frame, 0 otherwise.

The default value of the select expression is "1".

Some examples:

```
# Select all the frames in input
select

# The above is the same as
select=expr=1

# Skip all frames
select=expr=0

# Select only I-frames
select='expr=eq(pict_type\,I)'

# Select one frame per 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)'
```

24.32 setdar (avconv.html#toc-setdar-1)

Set 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 this 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.

It accepts the following parameters:

‘dar’
The output display aspect ratio.

The parameter *dar* is an expression containing the following constants:

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘w, h’
The input width and height.

‘a’
This is the same as *w* / *h*.

‘sar’
The input sample aspect ratio.

‘dar’
The input display aspect ratio. It is the same as (*w* / *h*) * *sar*.

‘hsub, vsub’
The horizontal and vertical chroma subsample values. For example, for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

To change the display aspect ratio to 16:9, specify:

```
setdar=dar=16/9
# The above is equivalent to
setdar=dar=1.77777
```

Also see the the setsar filter documentation.

24.33 setpts (avconv.html#toc-setpts)

Change the PTS (presentation timestamp) of the input video frames.

It accepts the following parameters:

‘expr’
The expression which is evaluated for each frame to construct its timestamp.

The expression is evaluated through the eval API and can contain the following constants:

‘PTS’
The presentation timestamp in input.

‘E, PI, PHI’
These are approximated values for the mathematical constants e (Euler’s number), pi (Greek pi), and phi (the golden ratio).

‘N’
The count of the input frame, starting from 0.

‘STARTPTS’
The PTS of the first video frame.

‘INTERLACED’
State whether the current frame is interlaced.

‘PREV_INPTS’
The previous input PTS.

‘PREV_OUTPTS’
The previous output PTS.

‘RTCTIME’
The wallclock (RTC) time in microseconds.

‘RTCSTART’
The wallclock (RTC) time at the start of the movie in microseconds.

‘TB’
The timebase of the input timestamps.

Some examples:

```
# Start counting the PTS from zero
setpts=expr=PTS-STARTPTS

# Fast motion
setpts=expr=0.5*PTS

# Slow motion
setpts=2.0*PTS

# Fixed rate 25 fps
setpts=N/(25*TB)

# Fixed rate 25 fps with some jitter
setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'

# Generate timestamps from a "live source" and rebase onto the current timebase
setpts='(RTCTIME - RTCSTART) / (TB * 1000000)'
```

24.34 setsar (avconv.html#toc-setsar-1)

Set 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 following equation: *DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR*

Keep in mind that the sample aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

It accepts the following parameters:

'sar'
The output sample aspect ratio.

The parameter *sar* is an expression containing the following constants:

'E, PI, PHI'
These are approximated values for the mathematical constants e (Euler's number), pi (Greek pi), and phi (the golden ratio).

'w, h'
The input width and height.

'a'
These are the same as *w* / *h*.

'sar'
The input sample aspect ratio.

'dar'
The 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.

To change the sample aspect ratio to 10:11, specify:

```
setsar=sar=10/11
```

24.35 settb (avconv.html#toc-settb)

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

It accepts the following parameters:

'expr'
The expression which is evaluated into the output timebase.

The expression can contain the constants "PI", "E", "PHI", "AVTB" (the default timebase), and "intb" (the input timebase).

The default value for the input is "intb".

Some examples:

```
# Set the timebase to 1/25
settb=expr=1/25

# Set the timebase to 1/10
settb=expr=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
```

24.36 showinfo (avconv.html#toc-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*.

It accepts the following parameters:

'n'
The (sequential) number of the input frame, starting from 0.

'pts'

The 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’
The Presentation TimeStamp of the input frame, expressed as a number of seconds.

‘pos’
The position of the frame in the input stream, or -1 if this information is unavailable and/or meaningless (for example in case of synthetic video).

‘fmt’
The pixel format name.

‘sar’
The sample aspect ratio of the input frame, expressed in the form *num*/*den*.

‘s’
The size of the input frame, expressed in the form *width*x*height*.

‘i’
The type of interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first).

‘iskey’
This is 1 if the frame is a key frame, 0 otherwise.

‘type’
The picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, or "?" for an unknown type). Also refer to the documentation of the `AVPictureType` enum and of the `av_get_picture_type_char` function defined in ‘`libavutil/avutil.h`’.

‘checksum’
The Adler-32 checksum of all the planes of the input frame.

‘plane_checksum’
The Adler-32 checksum of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]".

24.37 shuffleplanes (avconv.html#toc-shuffleplanes)

Reorder and/or duplicate video planes.

It accepts the following parameters:

‘map0’
The index of the input plane to be used as the first output plane.

‘map1’
The index of the input plane to be used as the second output plane.

‘map2’
The index of the input plane to be used as the third output plane.

‘map3’
The index of the input plane to be used as the fourth output plane.

The first plane has the index 0. The default is to keep the input unchanged.

Swap the second and third planes of the input:

```
avconv -i INPUT -vf shuffleplanes=0:2:1:3 OUTPUT
```

24.38 split (avconv.html#toc-split)

Split input video into several identical outputs.

It accepts a single parameter, which specifies the number of outputs. If unspecified, it defaults to 2.

Create 5 copies of the input video:

```
avconv -i INPUT -filter_complex split=5 OUTPUT
```

24.39 transpose (avconv.html#toc-transpose)

Transpose rows with columns in the input video and optionally flip it.

It accepts the following parameters:

‘dir’
The direction of the transpose.

The direction can assume the following values:

‘cclock_flip’
Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

‘clock’
Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

‘cclock’
Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

‘clock_flip’
Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

24.40 trim (avconv.html#toc-trim)

Trim the input so that the output contains one continuous subpart of the input.

It accepts the following parameters:

‘start’
The timestamp (in seconds) of the start of the kept section. The frame with the timestamp *start* will be the first frame in the output.

‘end’
The timestamp (in seconds) of the first frame that will be dropped. The frame immediately preceding the one with the timestamp *end* will be the last frame in the output.

‘start_pts’
This is the same as *start*, except this option sets the start timestamp in timebase units instead of seconds.

‘end_pts’
This is the same as *end*, except this option sets the end timestamp in timebase units instead of seconds.

‘duration’
The maximum duration of the output in seconds.

‘start_frame’
The number of the first frame that should be passed to the output.

‘end_frame’
The number of the first frame that should be dropped.

Note that the first two sets of the start/end options and the ‘`duration`’ option look at the frame timestamp, while the _frame variants simply count the frames that pass through the filter. Also note that this filter does not modify the timestamps. If you wish for the output timestamps to start at zero, insert a setpts filter after the trim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all the frames that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple trim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- Drop everything except the second minute of input:
  ```
  avconv -i INPUT -vf trim=60:120
  ```
- Keep only the first second:

```
avconv -i INPUT -vf trim=duration=1
```

24.41 unsharp (avconv.html#toc-unsharp)

Sharpen or blur the input video.

It accepts the following parameters:

'luma_msize_x'
Set the luma matrix horizontal size. It must be an integer between 3 and 13. The default value is 5.

'luma_msize_y'
Set the luma matrix vertical size. It must be an integer between 3 and 13. The default value is 5.

'luma_amount'
Set the luma effect strength. It must be a floating point number between -2.0 and 5.0. The default value is 1.0.

'chroma_msize_x'
Set the chroma matrix horizontal size. It must be an integer between 3 and 13. The default value is 5.

'chroma_msize_y'
Set the chroma matrix vertical size. It must be an integer between 3 and 13. The default value is 5.

'chroma_amount'
Set the chroma effect strength. It must be a floating point number between -2.0 and 5.0. The default value is 0.0.

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'.

```
# Strong luma sharpen effect parameters
unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5

# A strong blur of both luma and chroma parameters
unsharp=7:7:-2:7:7:-2

# Use the default values with avconv
./avconv -i in.avi -vf "unsharp" out.mp4
```

24.42 vflip (avconv.html#toc-vflip)

Flip the input video vertically.

```
./avconv -i in.avi -vf "vflip" out.avi
```

24.43 yadif (avconv.html#toc-yadif)

Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

It accepts the following parameters:

'mode'
The interlacing mode to adopt. It accepts one of the following values:

'0'
Output one frame for each frame.

'1'
Output one frame for each field.

'2'
Like 0, but it skips the spatial interlacing check.

'3'
Like 1, but it skips the spatial interlacing check.

The default value is 0.

'parity'
The picture field parity assumed for the input interlaced video. It accepts one of the following values:

'0'
Assume the top field is first.

'1'

Assume the bottom field is first.

‘-1’
Enable automatic detection of field parity.

The default value is -1. If the interlacing is unknown or the decoder does not export this information, top field first will be assumed.

‘auto’
Whether the deinterlacer should trust the interlaced flag and only deinterlace frames marked as interlaced.

‘0’
Deinterlace all frames.

‘1’
Only deinterlace frames marked as interlaced.

The default value is 0.

25. Video Sources (avconv.html#toc-Video-Sources)

Below is a description of the currently available video sources.

25.1 buffer (avconv.html#toc-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 the following parameters:

‘width’
The input video width.

‘height’
The input video height.

‘pix_fmt’
The name of the input video pixel format.

‘time_base’
The time base used for input timestamps.

‘sar’
The sample (pixel) aspect ratio of the input video.

‘hw_frames_ctx’
When using a hardware pixel format, this should be a reference to an AVHWFramesContext describing input frames.

For example:

```
buffer=width=320:height=240:pix_fmt=yuv410p:time_base=1/24:sar=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).

25.2 color (avconv.html#toc-color)

Provide an uniformly colored input.

It accepts the following parameters:

‘color’
Specify the color of the 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’
Specify the size of the sourced video, it may be a string of the form *widthxheight*, or the name of a size abbreviation. The default value is "320x240".

‘framerate’
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 floating point number or a valid video frame rate abbreviation. The default value is "25".

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, which will be overlaid over the source connected to the pad with identifier "in":

```
"color=red@0.2:qcif:10 [color]; [in][color] overlay [out]"
```

25.3 movie (avconv.html#toc-movie)

Read a video stream from a movie container.

Note that this source is a hack that bypasses the standard input path. It can be useful in applications that do not support arbitrary filter graphs, but its use is discouraged in those that do. It should never be used with `avconv` ; the '`-filter_complex`' option fully replaces it.

It accepts the following parameters:

'`filename`'
The name of the resource to read (not necessarily a file; it can also be a device or a stream accessed through some protocol).

'`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. The default value is "0".

'`stream_index, si`'
Specifies the index of the video stream to read. If the value is -1, the most suitable video stream will be automatically selected. The default value is "-1".

It allows overlaying a second video on top of the main input of a filtergraph, as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                    ^
                                    |
movie --> scale--> deltapts1 -------+
```

Some examples:

```
# Skip 3.2 seconds from the start of the AVI file in.avi, and overlay it
# on top of the input labelled "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 "in"
movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [movie];
[in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]
```

25.4 nullsrc (avconv.html#toc-nullsrc)

Null video source: never return images. It is mainly useful as a template and to be employed in analysis / debugging tools.

It accepts a string of the form *width*:*height*:*timebase* as an optional parameter.

width and *height* specify the size of the configured source. The default values of *width* and *height* are respectively 352 and 288 (corresponding to the CIF size format).

timebase specifies an arithmetic expression representing a timebase. The expression can contain the constants "PI", "E", "PHI", and "AVTB" (the default timebase), and defaults to the value "AVTB".

25.5 frei0r_src (avconv.html#toc-frei0r_005fsrc)

Provide a frei0r source.

To enable compilation of this filter you need to install the frei0r header and configure Libav with –enable-frei0r.

This source accepts the following parameters:

'`size`'
The size of the video to generate. It may be a string of the form *width*x*height* or a frame size abbreviation.

'`framerate`'
The framerate of the generated video. It may be a string of the form *num*/*den* or a frame rate abbreviation.

'`filter_name`'
The name to the frei0r source to load. For more information regarding frei0r and how to set the parameters, read the frei0r section in the video filters documentation.

‘filter_params’
A ’|’-separated list of parameters to pass to the frei0r source.

An example:

```
# Generate a frei0r partik0l source with size 200x200 and framerate 10
# which is overlaid on the overlay filter's main input
frei0r_src=size=200x200:framerate=10:filter_name=partik0l:filter_params=1234 [overlay]; [in][overlay] overlay
```

25.6 rgbtestsrc, testsrc (avconv.html#toc-rgbtestsrc_002c-testsrc)

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 `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

The sources accept the following parameters:

‘size, s’
Specify the size of the sourced video, it may be a string of the form *widthxheight*, 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 floating point number or a valid video frame rate abbreviation. The default value is "25".

‘sar’
Set the sample aspect ratio of the sourced video.

‘duration’
Set the video duration of the sourced video. The accepted syntax is:

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

Also see the the `av_parse_time()` function.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

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 framerate of 10 frames per second.

26. Video Sinks (avconv.html#toc-Video-Sinks)

Below is a description of the currently available video sinks.

26.1 buffersink (avconv.html#toc-buffersink)

Buffer video frames, and make them available to the end of the filter graph.

This sink is intended for programmatic use through the interface defined in ‘`libavfilter/buffersink.h`’.

26.2 nullsink (avconv.html#toc-nullsink)

Null video sink: do absolutely nothing with the input video. It is mainly useful as a template and for use in analysis / debugging tools.

27. Metadata (avconv.html#toc-Metadata)

Libav 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=Libav 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 *git push/pull user* on *November 28, 2016* using *texi2html 1.82* (http://www.nongnu.org/texi2html/).