precise (1) avconv.1.gz
Provided by: libav-tools_0.8.1-0ubuntu1_i386

NAME

avconv - avconv video converter

SYNOPSIS

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

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.

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.

OPTIONS

All the numerical options, if not specified otherwise, accept in input

All the numerical options, if not specified otherwise, accept in input a string representing a number, which may contain one of the International System number postfixes, for example 'K', 'M', 'G'. 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.

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".

Stream specifiers

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

A stream specifier is a string generally appended to the option name and separated from it by a colon. E.g. "-codec:a:1 ac3" option contains "a:1" stream specifer, 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.

Generic options

These options are shared amongst the av* tools.

-L Show license.

-h, -?, -help, --help
: Show help.

-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 available codecs.

The fields preceding the codec names have the following meanings:

D Decoding available

E Encoding available

V/A/S
: Video/audio/subtitle codec

S Codec supports slices

D Codec supports direct rendering

D Codec supports direct rendering

T Codec can handle input truncated at random locations instead of only at frame boundaries

-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

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.

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.

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.

-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. When used as an output option (before an output filename), decodes but discards input until the timestamps reach position. This is slower, but more accurate.

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

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

-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 alias for "-frames:d".

-frames[:stream_specifier**]** framecount **(**output,per-stream**)**
Stop writing to the stream after framecount frames.

-q[:stream_specifier**]** q **(**output,per-stream**)**
-qscale[:stream_specifier**]** q **(**output,per-stream**)**
Use fixed quality scale (VBR). The meaning of q is codec-dependent.

-filter[:stream_specifier**]** filter_graph **(**output,per-stream**)**
filter_graph is a description of the filter graph to apply to the stream. Use "-filters" to show all the available filters (including also sources and sinks).

-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.

Video Options

-vframes number **(**output**)**
Set the number of video frames to record. This is an alias for "-frames:v".

-r[:stream_specifier**]** fps **(**input/output,per-stream**)**
Set frame rate (Hz value, fraction or abbreviation), (default = 25).

-s[:stream_specifier**]** size **(**input/output,per-stream**)**
Set frame size. 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

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

-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.

-bt tolerance
Set video bitrate tolerance (in bits, default 4000k). Has a minimum value of: (target_bitrate/target_framerate). In 1-pass

mode, bitrate tolerance specifies how far ratecontrol is willing to deviate from the target average bitrate value. This is not related to min/max bitrate. Lowering tolerance too much has an adverse effect on quality.

-maxrate bitrate
Set max video bitrate (in bit/s). Requires -bufsize to be set.

-minrate bitrate
Set min video bitrate (in bit/s). Most useful in setting up a CBR encode:

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

It is of little use elsewise.

-bufsize size
Set video buffer verifier buffer size (in bits).

-vcodec codec **(**output**)**
Set the video codec. This is an alias for "-codec:v".

-same_quant
Use same quantizer as source (implies VBR).

Note that this is NOT SAME QUALITY. Do not use this option unless you know you need it.

-pass n
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 prefix **(**global**)**
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".

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.

-g gop_size
Set the group of pictures size.

-vdt n
Discard threshold.

-qmin q
minimum video quantizer scale (VBR)

-qmax q
maximum video quantizer scale (VBR)

-qdiff q
maximum difference between the quantizer scales (VBR)

-qblur blur
video quantizer scale blur (VBR) (range 0.0 - 1.0)

-qcomp compression
video quantizer scale compression (VBR) (default 0.5). Constant of ratecontrol equation. Recommended range for default rc_eq: 0.0-1.0

-lmin lambda

minimum video lagrange factor (VBR)

-lmax lambda
max video lagrange factor (VBR)

-mblmin lambda
minimum macroblock quantizer scale (VBR)

-mblmax lambda
maximum macroblock quantizer scale (VBR)

These four options (lmin, lmax, mblmin, 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
```

-rc_init_cplx complexity
initial complexity for single pass encoding

-b_qfactor factor
qp factor between P- and B-frames

-i_qfactor factor
qp factor between P- and I-frames

-b_qoffset offset
qp offset between P- and B-frames

-i_qoffset offset
qp offset between P- and I-frames

-rc_eq equation
Set rate control equation (see section "Expression Evaluation") (default = "tex^qComp").

When computing the rate control equation expression, besides the standard functions defined in the section "Expression Evaluation", the following functions are available:

bits2qp(bits)
qp2bits(qp)

and the following constants are available:

iTex
pTex
tex
mv
fCode
iCount
mcVar
var
isI
isP
isB
avgQP
qComp
avgIITex
avgPITex
avgPPTex
avgBPTex
avgTex

-rc_override[:stream_specifier**]** override **(**output,per-stream**)**
rate control override for specific intervals

-me_method method
Set motion estimation method to method. Available methods are (from lowest to best quality):

zero
Try just the (0, 0) vector.

phods
log
x1
hex
umh
epzs
(default method)

full
exhaustive search (slow and marginally better than epzs)

-er n
Set error resilience to n.

1 FF_ER_CAREFUL (default)

2 FF_ER_COMPLIANT

3 FF_ER_AGGRESSIVE

4 FF_ER_VERY_AGGRESSIVE

-ec bit_mask
Set error concealment to bit_mask. bit_mask is a bit mask of the following values:

1 FF_EC_GUESS_MVS (default = enabled)

2 FF_EC_DEBLOCK (default = enabled)

-bf frames
Use 'frames' B-frames (supported for MPEG-1, MPEG-2 and MPEG-4).

-mbd mode
macroblock decision

0 FF_MB_DECISION_SIMPLE: Use mb_cmp (cannot change it yet in avconv).

1 FF_MB_DECISION_BITS: Choose the one which needs the fewest bits.

2 FF_MB_DECISION_RD: rate distortion

-bug param
Work around encoder bugs that are not auto-detected.

-strict strictness
How strictly to follow the standards.

-deinterlace
Deinterlace pictures.

-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.

Audio Options

-aframes number **(**output**)**
Set the number of audio frames to record. This is an alias for "-frames:a".

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

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

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

-ac[:stream_specifier**]** channels **(**input/output,per-stream**)**
Set the number of audio channels. For output streams it is set by default to the number of input audio channels. For input streams this option only makes sense for audio grabbing devices and raw demuxers and is mapped to the corresponding demuxer options.

-an (output**)**
Disable audio recording.

-acodec codec **(**input/output**)**
Set the audio codec. This is an alias for "-codec:a".

-sample_fmt[:stream_specifier**]** sample_fmt **(**output,per-stream**)**
Set the audio sample format. Use "-sample_fmts" to get a list of supported sample formats.

Advanced Audio options:

-atag fourcc/tag **(**output**)**
Force audio tag/fourcc. This is an alias for "-tag:a".

-audio_service_type type
Set the type of service that the audio stream contains.

ma Main Audio Service (default)

ef Effects

vi Visually Impaired

hi Hearing Impaired

di Dialogue

co Commentary

em Emergency

vo Voice Over

ka Karaoke

Subtitle options:

-scodec codec **(**input/output**)**
Set the subtitle codec. This is an alias for "-codec:s".

-sn (output**)**
Disable subtitle recording.

Audio/Video grab options

-isync (global**)**
Synchronize read on input.

Advanced options

-map [-]input_file_id**[:**stream_specifier**][,**sync_file_id**[:**stream_specifier**]] (**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.

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

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.

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.

-ps size
Set RTP payload size in bytes.

-re (input**)**
Read input at native frame rate. Mainly used to simulate a grab device.

-threads count
Thread count.

-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.

-copyts
Copy timestamps from input to output.

-copytb
Copy input stream time base from input to output when stream copying.

-shortest
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. bistream_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 -vbsf h264_mp4toannexb -an out.h264
```

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

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

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 '-intra' 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).

EXAMPLES

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.

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.

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.

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
  ```

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 -vframes or -t option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

    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 0.3 -map 0.2 -map 0.1 -map 0.0 -c copy test12.nut

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

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)
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

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 "-codecs" of the ff* tools will display the list of enabled encoders.

AUDIO ENCODERS

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

ac3 and ac3_fixed

AC-3 audio encoders.

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

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

AC-3 Metadata

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

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

*<A/52:2010 - Digital Audio Compression (AC-3) (E-AC-3) Standard ("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")>

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.

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)

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

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).

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)

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.

Extended Bitstream Information - Part 1

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

0
notindicated
Not Indicated (default)

1
ltrt
Lt/Rt Downmix Preferred

2
loro
Lo/Ro Downmix Preferred

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

1.414
Apply +3dB gain

1.189
Apply +1.5dB gain

1.000
Apply 0dB gain

0.841
Apply -1.5dB gain

0.707

Apply -3.0dB gain

0.595
Apply -4.5dB gain (default)

0.500
Apply -6.0dB gain

0.000
Silence Center Channel

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

0.841
Apply -1.5dB gain

0.707
Apply -3.0dB gain

0.595
Apply -4.5dB gain

0.500
Apply -6.0dB gain (default)

0.000
Silence Surround Channel(s)

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

1.414
Apply +3dB gain

1.189
Apply +1.5dB gain

1.000
Apply 0dB gain

0.841
Apply -1.5dB gain

0.707
Apply -3.0dB gain

0.595
Apply -4.5dB gain (default)

0.500
Apply -6.0dB gain

0.000
Silence Center Channel

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

0.841
Apply -1.5dB gain

0.707
Apply -3.0dB gain

0.595
Apply -4.5dB gain

0.500
Apply -6.0dB gain (default)

0.000
Silence Surround Channel(s)

Extended Bitstream Information - Part 2

-dsurex_mode mode
Dolby Surround EX Mode. Indicates whether the stream uses Dolby Surround EX (7.1 matrixed to 5.1). Using this option does **NOT** mean the encoder will actually apply Dolby Surround EX processing.

0
notindicated
Not Indicated (default)

1
on Dolby Surround EX Off

2
off Dolby Surround EX On

-dheadphone_mode mode
Dolby Headphone Mode. Indicates whether the stream uses Dolby Headphone encoding (multi-channel matrixed to 2.0 for use with headphones). Using this option does **NOT** mean the encoder will actually apply Dolby Headphone processing.

0
notindicated
Not Indicated (default)

1
on Dolby Headphone Off

2
off Dolby Headphone On

-ad_conv_type type
A/D Converter Type. Indicates whether the audio has passed through HDCD A/D conversion.

0
standard
Standard A/D Converter (default)

1
hdcd
HDCD A/D Converter

Other AC-3 Encoding Options

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

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

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)

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 ff* tools will display the list of enabled demuxers.

The description of some of the currently available demuxers follows.

image2

Image file demuxer.

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

The pattern may contain the string "%d" or "%0Nd", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0Nd" 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 "%0Nd", 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 "%0Nd", for example to convert a single image file img.jpeg you can employ the command:

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

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".

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 ff* tools will display the list of enabled muxers.

A description of some of the currently available muxers follows.

crc

CRC (Cyclic Redundancy Check) testing format.

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

The output of the muxer consists of a single line of the form: CRC=0xCRC, 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.

framecrc

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, 0xCRC, 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.

image2

Image file muxer.

The image file muxer writes video frames to image files.

The output filenames are specified by a pattern, which can be used to produce sequentially numbered series of files. The pattern may contain the string "%d" or "%0Nd", this string specifies the position of the characters representing a numbering in the filenames. If the form "%0Nd" 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 "%0Nd", 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 "%0Nd", 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
```

mpegts

MPEG transport stream muxer.

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

The muxer options are:

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

-mpegts_transport_stream_id number
: Set the transport_stream_id (default 0x0001). This identifies a transponder in DVB.

-mpegts_service_id number
: Set the service_id (default 0x0001) also known as program in DVB.

-mpegts_pmt_start_pid number
: Set the first PID for PMT (default 0x1000, max 0x1f00).

-mpegts_start_pid number
: Set the first PID for data packets (default 0x0100, max 0x0f00).

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

null

Null muxer.

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

For example to benchmark decoding with **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 -
```

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
```

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_size size
Overwrite the listfile once it reaches size entries.

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

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 ff* tools will display the list of supported input devices (amongst the demuxers).

A description of the currently available input devices follows.

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**>

bktr
BSD video input device.

dv1394
Linux DV 1394 input device.

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/**>, and fbset(1).

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/**>

libdc1394
IIDC1394 input device, based on libdc1394 and libraw1394.

oss
Open Sound System input device.

The filename to provide to the input device is the device node representing the OSS input device, and is usually set to /dev/dsp.

For example to grab from /dev/dsp using **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**>

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
```

server AVOption

The syntax is:

```
-server <server name>
```

Connects to a specific server.

name AVOption

The syntax is:

```
-name <application name>
```

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

stream_name AVOption

The syntax is:

```
-stream_name <stream name>
```

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

sample_rate AVOption

The syntax is:

```
-sample_rate <samplerate>
```

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

channels AVOption

The syntax is:

```
-channels <N>
```

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

frame_size AVOption

The syntax is:

```
-frame_size <bytes>
```

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

fragment_size AVOption

The syntax is:

```
-fragment_size &lt;bytes&gt;
```

Specify the minimal buffering fragment in pulseaudio, it will affect the audio latency. By default it is unset.

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
```

video4linux and video4linux2

Video4Linux and Video4Linux2 input video devices.

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.

Video4Linux and Video4Linux2 devices only support a limited set of widthxheight sizes and framerates. You can check which are supported for example with the command dov4l for Video4Linux devices and using **-list_formats all** for Video4Linux2 devices.

If the size for the device is set to 0x0, the input device will try to autodetect the size to use. Only for the video4linux2 device, if the frame rate is set to 0/0 the input device will use the frame rate value already set in the driver.

Video4Linux support is deprecated since Linux 2.6.30, and will be dropped in later versions.

Follow some usage examples of the video4linux devices with the ff* tools.

```
# Grab and show the input of a video4linux device, frame rate is set
# to the default of 25/1.
avplay -s 320x240 -f video4linux /dev/video0

# Grab and show the input of a video4linux2 device, autoadjust size.
avplay -f video4linux2 /dev/video0

# Grab and record the input of a video4linux2 device, autoadjust size,
# frame rate value defaults to 0/0 so it is read from the video4linux2
# driver.
avconv -f video4linux2 -i /dev/video0 out.mpeg
```

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.

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 ommitted, 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
```

follow_mouse 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
```

show_region 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
```

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 ff* tools will display the list of enabled output devices (amongst the muxers).

A description of the currently available output devices follows.

alsa

ALSA (Advanced Linux Sound Architecture) output device.

oss

OSS (Open Sound System) output device.

sndio

sndio audio output device.

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 enabled by default. You can list all available ones using the configure option "--list-protocols".

You can disable all the protocols using the configure option "--disable-protocols", and selectively enable a protocol using the option "--enable-protocol=PROTOCOL", or you can disable a particular protocol using the option "--disable-protocol=PROTOCOL".

The option "-protocols" of the ff* tools will display the list of supported protocols.

A description of the currently available protocols follows.

applehttp

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. HTTP is default, specific protocol can be declared by specifying "+proto" after the applehttp URI scheme name, where proto is either "file" or "http".

```
applehttp://host/path/to/remote/resource.m3u8
applehttp+http://host/path/to/remote/resource.m3u8
applehttp+file://path/to/local/resource.m3u8
```

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.

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

gopher

Gopher protocol.

http

HTTP (Hyper Text Transfer Protocol).

mmst

MMS (Microsoft Media Server) protocol over TCP.

mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

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.

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.

rtmp
Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimedia content across a TCP/IP network.

The required syntax is:

```
rtmp://<server>[:<port>][/<app>][/<playpath>]
```

The accepted parameters are:

server
The address of the RTMP server.

port
The number of the TCP port to use (by default is 1935).

app It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. /ondemand/, /flash/live/, etc.).

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:".

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
```

rtmp, rtmpe, rtmps, rtmpt, rtmpte
Real-Time Messaging Protocol and its variants supported through librtmp.

Requires the presence of the librtmp headers and library during configuration. You need to explicitly configure the build with "--enable-librtmp". If enabled this will replace the native RTMP protocol.

This protocol provides most client functions and a few server functions needed to support RTMP, RTMP tunneled in HTTP (RTMPT), encrypted RTMP (RTMPE), RTMP over SSL/TLS (RTMPS) and tunneled variants of these encrypted types (RTMPTE, RTMPTS).

The required syntax is:

    <rtmp_proto>://<server>[:<port>][/<app>][/<playpath>] <options>

where rtmp_proto is one of the strings "rtmp", "rtmpt", "rtmpe", "rtmps", "rtmpte", "rtmpts" corresponding to each RTMP variant, and server, port, app and playpath have the same meaning as specified for the RTMP native protocol. options contains a list of space-separated options of the form key=val.

See the librtmp manual page (man 3 librtmp) for more information.

For example, to stream a file in real-time to an RTMP server using **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"

rtp
Real-Time Protocol.

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.

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). In order for this to be enabled, a maximum delay must be specified in 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
```

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.

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]
```

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

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]
```

tcp

Trasmission 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>
```

udp

User Datagram Protocol.

The required syntax for a UDP url is:

```
udp://<hostname>:<port>[?<options>]
```

options contains a list of &-seperated 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.

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>
```

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 ff* tools will display the list of all the supported bitstream filters included in your build.

Below is a description of the currently available bitstream filters.

aac_adtstoasc
chomp
dump_extradata
h264_mp4toannexb
imx_dump_header
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**>:

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

mjpega_dump_header
movsub
mp3_header_compress
mp3_header_decompress
noise
remove_extradata

FILTERGRAPH DESCRIPTION

A filtergraph is a directed graph of connected filters. It can contain cycles, and there can be multiple links between a pair of filters. Each link has one input pad on one side connecting it to one filter from which it takes its input, and one output pad on the other side connecting it to the one filter accepting its output.

Each filter in a filtergraph is an instance of a filter class registered in the application, which defines the features and the number of input and output pads of the filter.

A filter with no input pads is called a "source", a filter with no output pads is called a "sink".

Filtergraph syntax

A filtergraph can be represented using a textual representation, which is recognized by the "-vf" and "-af" options in **avconv** and **avplay**, and by the "av_parse_graph()" function defined in libavfilter/avfiltergraph.

A filterchain consists of a sequence of connected filters, each one connected to the previous one in the sequence. A filterchain is represented by a list of ","-separated filter descriptions.

A filtergraph consists of a sequence of filterchains. A sequence of filterchains is represented by a list of ";"-separated filterchain descriptions.

A filter is represented by a string of the form:
[in_link_1]...[in_link_N]filter_name=arguments[out_link_1]...[out_link_M]

filter_name is the name of the filter class of which the described filter is an instance of, and has to be the name of one of the filter classes registered in the program. The name of the filter class is optionally followed by a string "=arguments".

arguments is a string which contains the parameters used to initialize the filter instance, and are described in the filter descriptions below.

The list of arguments can be quoted using the character "'" as initial and ending mark, and the character '\' for escaping the characters within the quoted text; otherwise the argument string is considered terminated when the next special character (belonging to the set "[]=;,") is encountered.

The name and arguments of the filter are optionally preceded and followed by a list of link labels. A link label allows to name a link and associate it to a filter output or input pad. The preceding labels in_link_1 ... in_link_N, are associated to the filter input pads, the following labels out_link_1 ... out_link_M, are associated to the output pads.

When two link labels with the same name are found in the filtergraph, a link between the corresponding input and output pad is created.

If an output pad is not labelled, it is linked by default to the first unlabelled input pad of the next filter in the filterchain. For example in the filterchain:

```
nullsrc, split[L1], [L2]overlay, nullsink
```

the split filter instance has two output pads, and the overlay filter instance two input pads. The first output pad of split is labelled "L1", the first input pad of overlay is labelled "L2", and the second output pad of split is linked to the second input pad of overlay, which are both unlabelled.

In a complete filterchain all the unlabelled filter input and output pads must be connected. A filtergraph is considered valid if all the filter input and output pads of all the filterchains are connected.

Follows a BNF description for the filtergraph syntax:

```
<NAME>             ::= sequence of alphanumeric characters and '_'
<LINKLABEL>        ::= "[" <NAME> "]"
<LINKLABELS>       ::= <LINKLABEL> [<LINKLABELS>]
<FILTER_ARGUMENTS> ::= sequence of chars (eventually quoted)
<FILTER>           ::= [<LINKNAMES>] <NAME> ["=" <ARGUMENTS>] [<LINKNAMES>]
<FILTERCHAIN>      ::= <FILTER> [,<FILTERCHAIN>]
<FILTERGRAPH>      ::= <FILTERCHAIN> [;<FILTERGRAPH>]
```

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.

anull

Pass the audio source unchanged to the output.

AUDIO SOURCES

Below is a description of the currently available audio sources.

anullsrc

Null audio source, never return audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools.

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

sample_rate specify the sample rate, and defaults to 44100.

channel_layout specify 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 libavcodec/audioconvert.c
for the mapping between strings and channel layout values.

Follow some examples:

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

# same as
anullsrc=48000:mono
```

AUDIO SINKS

Below is a description of the currently available audio sinks.

anullsink

Null audio sink, do absolutely nothing with the input audio. It is mainly useful as a template and to be employed in analysis / debugging tools.

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.

blackframe

Detect frames that are (almost) completely black. Can be useful to detect chapter transitions or commercials. Output lines consist of the frame number of the detected frame, the percentage of blackness, the position in the file if known or -1 and the timestamp in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the syntax:

```
blackframe[=<amount>:[<threshold>]]
```

amount is the percentage of the pixels that have to be below the threshold, and defaults to 98.

threshold is the threshold below which a pixel value is considered black, and defaults to 32.

boxblur

Apply boxblur algorithm to the input video.

This filter accepts the parameters:
luma_power:luma_radius:chroma_radius:chroma_power:alpha_radius:alpha_power

Chroma and alpha parameters are optional, if not specified they default to the corresponding values set for luma_radius and luma_power.

luma_radius, chroma_radius, and alpha_radius represent the radius in pixels of the box used for blurring the corresponding input plane. They are expressions, and can contain the following constants:

w, h
the input width and height in pixels

cw, ch
the input chroma image width and height in pixels

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

The radius must be a non-negative number, and must not be greater than the value of the expression "min(w,h)/2" for the luma and alpha planes, and of "min(cw,ch)/2" for the chroma planes.

luma_power, chroma_power, and alpha_power represent how many times the boxblur filter is applied to the corresponding plane.

Some examples follow:

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

```
boxblur=2:1
```

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

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

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

```
boxblur=min(h,w)/10:1:min(cw,ch)/10:1
```

copy
Copy the input source unchanged to the output. Mainly useful for testing purposes.

crop
Crop the input video to out_w:out_h:x:y.

The parameters are expressions containing the following constants:

E, PI, PHI
the corresponding mathematical approximated values for e (euler number), pi (greek PI), PHI (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
same as in_w and in_h

out_w, out_h
the output (cropped) width and height

ow, oh
same as out_w and out_h

n the number of input frame, starting from 0

pos the position in the file of the input frame, NAN if unknown

t timestamp expressed in seconds, NAN if the input timestamp is unknown

The out_w and out_h parameters specify the expressions for the width and height of the output (cropped) video. They are evaluated just at the configuration of the filter.

The default value of out_w is "in_w", and the default value of out_h is "in_h".

The expression for out_w may depend on the value of out_h, and the expression for out_h may depend on out_w, but they cannot depend on x and y, as x and y are evaluated after out_w and out_h.

The x and y parameters specify the expressions for the position of the top-left corner of the output (non-cropped) area. They are evaluated for each frame. If the evaluated value is not valid, it is approximated to the nearest valid value.

The default value of x is "(in_w-out_w)/2", and the default value for y is "(in_h-out_h)/2", which set the cropped area at the center of the input image.

The expression for x may depend on y, and the expression for y may depend on x.

Follow some examples:

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

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

# crop the input video central square
crop=in_h

# delimit the rectangle with the top-left corner placed at position
```

```
# 100:100 and the right-bottom corner corresponding to the right-bottom
# corner of the input image.
crop=in_w-100:in_h-100:100:100

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

# keep only the bottom right quarter of the input image
"crop=in_w/2:in_h/2:in_w/2:in_h/2"

# crop height for getting Greek harmony
"crop=in_w:1/PHI*in_w"

# trembling effect
"crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)"

# erratic camera effect depending on timestamp
"crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"

# set x depending on the value of y
"crop=in_w/2:in_h/2:y:10+10*sin(n/10)"
```

cropdetect

Auto-detect crop size.

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

It accepts the syntax:

```
cropdetect[=<limit>[:<round>[:<reset>]]]
```

limit

Threshold, which can be optionally specified from nothing (0) to everything (255), defaults to 24.

round

Value which the width/height should be divisible by, defaults to 16. The offset is automatically adjusted to center the video. Use 2 to get only even dimensions (needed for 4:2:2 video). 16 is best when encoding to most video codecs.

reset

Counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Defaults to 0.

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

delogo

Suppress a TV station logo by a simple interpolation of the surrounding pixels. Just set a rectangle covering the logo and watch it disappear (and sometimes something even uglier appear - your mileage may vary).

The filter accepts parameters as a string of the form "x:y:w:h:band", or as a list of key=value pairs, separated by ":".

The description of the accepted parameters follows.

x, y

Specify the top left corner coordinates of the logo. They must be specified.

w, h

Specify the width and height of the logo to clear. They must be specified.

band, t

Specify the thickness of the fuzzy edge of the rectangle (added to w and h). The default value is 4.

show

When set to 1, a green rectangle is drawn on the screen to simplify finding the right x, y, w, h parameters, and band is set to 4. The default value is 0.

Some examples follow.

Some examples follow:

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

  ```
  delogo=0:0:100:77:10
  ```

- As the previous example, but use named options:

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

drawbox

Draw a colored box on the input image.

It accepts the syntax:

```
drawbox=<x>:<y>:<width>:<height>:<color>
```

x, y
: Specify the top left corner coordinates of the box. Default to 0.

width, height
: Specify the width and height of the box, if 0 they are interpreted as the input width and height. Default 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.

Follow 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=10:20:200:60:red@0.5"
```

drawtext

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

To enable compilation of this filter you need to configure Libav with "--enable-libfreetype".

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

The filter accepts parameters as a list of key=value pairs, separated by ":".

The description of the accepted parameters follows.

fontfile
: The font file to be used for drawing text. Path must be included. This parameter is mandatory.

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. 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
  : main input width and height

W, H
same as main_w and main_h

text_w, text_h
rendered text width and height

w, h
same as text_w and text_h

n the number of frames processed, starting from 0

t timestamp expressed in seconds, NAN if the input timestamp is unknown

The default value of x and y is 0.

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. 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. 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 background color. Value should 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. 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
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.

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/**>.

fade

Apply fade-in/out effect to input video.

It accepts the parameters: type:start_frame:nb_frames

type specifies if the effect type, can be either "in" for fade-in, or "out" for a fade-out effect.

start_frame specifies the number of the start frame for starting to apply the fade effect.

nb_frames specifies the number of frames for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black.

A few usage examples follow, usable too as test scenarios.

```
# fade in first 30 frames of video
fade=in:0:30

# fade out last 45 frames of a 200-frame video
fade=out:155:45

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

# make first 5 frames black, then fade in from frame 5-24
fade=in:5:20
```

fieldorder

Transform the field order of the input video.

It accepts one parameter which specifies the required field order that the input interlaced video will be transformed to. The parameter can assume one of the following values:

0 or bff

output bottom field first

1 or tff

output top field first

Default value is "tff".

Transformation is achieved by shifting the picture content up or down by one line, and filling the remaining line with appropriate picture content. This method is consistent with most broadcast field order converters.

If the input video is not flagged as being interlaced, or it is already flagged as being of the required output field order then this filter does not alter the incoming video.

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

For example:

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

fifo

Buffer input images and send them when they are requested.

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

The filter does not take parameters.

format

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

The filter accepts a list of pixel format names, separated by ":", for example "yuv420p:monow:rgb24".

Some examples follow:

```
# convert the input video to the format "yuv420p"
format=yuv420p

# convert the input video to any of the formats in the list
format=yuv420p:yuv444p:yuv410p
```

frei0r

Apply a frei0r effect to the input video.

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

The filter supports the syntax:

```
<filter_name>[{:|=}<param1>:<param2>:...:<paramN>]
```

filter_name is the name to the frei0r effect to load. If the environment variable **FREIOR_PATH** is defined, the frei0r effect is searched in each one of the directories specified by the colon separated list in **FREIOR_PATH**, otherwise in the standard frei0r paths, which are in this order: HOME/.frei0r-1/lib/, /usr/local/lib/frei0r-1/, /usr/lib/frei0r-1/.

param1, param2, ... , paramN specify the parameters for the frei0r effect.

A frei0r effect parameter can be a boolean (whose values are specified with "y" and "n"), a double, a color (specified by the syntax R/G/B, R, G, and B being float numbers from 0.0 to 1.0) or by an "av_parse_color()" color description), a position (specified by the syntax X/Y, X and Y being float numbers) and a string.

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

Some examples follow:

```
# apply the distort0r effect, set the first two double parameters
frei0r=distort0r:0.5:0.01

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

# apply the perspective effect, specify the top left and top right
# image positions
frei0r=perspective:0.2/0.2:0.8/0.2
```

For more information see: <**http://piksel.org/frei0r**>

gradfun

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

This filter is designed for playback only. Do not use it prior to lossy compression, because compression tends to lose the dither and bring back the bands.

The filter takes two optional parameters, separated by ':':
strength:radius

strength is the maximum amount by which the filter will change any one pixel. Also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 255, default value is 1.2, out-of-range values will be clipped to the valid range.

radius is the neighborhood to fit the gradient to. A larger radius makes for smoother gradients, but also prevents the filter from modifying the pixels near detailed regions. Acceptable values are 8-32, default value is 16, out-of-range values will be clipped to the valid range.

```
# default parameters
gradfun=1.2:16

# omitting radius
gradfun=1.2
```

hflip

Flip the input video horizontally.

For example to horizontally flip the input video with **avconv**:

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

hqdn3d

High precision/quality 3d denoise filter. This filter aims to reduce image noise producing smooth images and making still images really still. It should enhance compressibility.

It accepts the following optional parameters:
luma_spatial:chroma_spatial:luma_tmp:chroma_tmp

luma_spatial
a non-negative float number which specifies spatial luma strength, defaults to 4.0

chroma_spatial
a non-negative float number which specifies spatial chroma strength, defaults to 3.0*luma_spatial/4.0

luma_tmp
a float number which specifies luma temporal strength, defaults to 6.0*luma_spatial/4.0

chroma_tmp
a float number which specifies chroma temporal strength, defaults to luma_tmp*chroma_spatial/luma_spatial

lut, lutrgb, lutyuv

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

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

These filters accept in input a ":"-separated list of options, which specify the expressions used for computing the lookup table for the corresponding pixel component values.

The lut filter requires either YUV or RGB pixel formats in input, and accepts the options:

c0 (first pixel component) c1 (second pixel component) c2 (third pixel component) c3 (fourth pixel component, corresponds to the alpha component)

The exact component associated to each option depends on the format in input.

The lutrgb filter requires RGB pixel formats in input, and accepts the options:

r (red component) g (green component) b (blue component) a (alpha component)

The lutyuv filter requires YUV pixel formats in input, and accepts the options:

y (Y/luminance component) u (U/Cb component) v (V/Cr component) a (alpha component)

The expressions can contain the following constants and functions:

E, PI, PHI
the corresponding mathematical approximated values for e (euler number), pi (greek PI), PHI (golden ratio)

w, h
the input width and height

val input value for the pixel component

clipval
the input value clipped in the minval-maxval range

maxval
maximum value for the pixel component

minval
minimum value for the pixel component

negval
the negated value for the pixel component value clipped in the minval-maxval range , it corresponds to the expression "maxval-clipval+minval"

clip(val)
the computed value in val clipped in the minval-maxval range

gammaval(gamma)
the computed gamma correction value of the pixel component value clipped in the minval-maxval range, corresponds to the expression "pow((clipval-minval)/(maxval-minval),gamma)*(maxval-minval)+minval"

All expressions default to "val".

Some examples follow:

```
# negate input video
lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val"
lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"

# the above is the same as
lutrgb="r=negval:g=negval:b=negval"
lutyuv="y=negval:u=negval:v=negval"

# negate luminance
lutyuv=negval

# remove chroma components, turns the video into a graytone image
lutyuv="u=128:v=128"

# apply a luma burning effect
lutyuv="y=2*val"

# remove green and blue components
lutrgb="g=0:b=0"

# set a constant alpha channel value on input
format=rgba,lutrgb=a="maxval-minval/2"

# correct luminance gamma by a 0.5 factor
lutyuv=y=gammaval(0.5)
```

negate
Negate input video.

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

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

The filter accepts a list of pixel format names, separated by ":", for example "yuv420p:monow:rgb24".

Some examples follow:

```
# force libavfilter to use a format different from "yuv420p" for the
```

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

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

null

Pass the video source unchanged to the output.

ocv

Apply video transform using libopencv.

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

The filter takes the parameters: filter_name{:=}filter_params.

filter_name is the name of the libopencv filter to apply.

filter_params specifies the parameters to pass to the libopencv filter. If not specified the default values are assumed.

Refer to the official libopencv documentation for more precise information:
<**http://opencv.willowgarage.com/documentation/c/image_filtering.html**>

Follows the list of supported libopencv filters.

dilate

Dilate an image by using a specific structuring element. This filter corresponds to the libopencv function "cvDilate".

It accepts the parameters: struct_el:nb_iterations.

struct_el represents a structuring element, and has the syntax:
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, and can be one of the values "rect", "cross", "ellipse", "custom".

If the value for shape is "custom", it must be followed by a string of the form "=filename". The file with name filename is assumed to represent a binary image, with each printable character corresponding to a bright pixel. When a custom shape is used, cols and rows are ignored, the number or columns and rows of the read file are assumed instead.

The default value for struct_el is "3x3+0x0/rect".

nb_iterations specifies the number of times the transform is applied to the image, and defaults to 1.

Follow some example:

```
# use the default values
ocv=dilate

# dilate using a structuring element with a 5x5 cross, iterate two times
ocv=dilate=5x5+2x2/cross:2

# read the shape from the file diamond.shape, iterate two times
# the file diamond.shape may contain a pattern of characters like this:
#   *
#  ***
# *****
#  ***
#   *
# the specified cols and rows are ignored (but not the anchor point coordinates)
ocv=0x0+2x2/custom=diamond.shape:2
```

erode

Erode an image by using a specific structuring element. This filter corresponds to the libopencv function "cvErode".

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

smooth

Smooth the input video.

The filter takes the following parameters:
type:param1:param2:param3:param4.

type is the type of smooth filter to apply, and can be one of the following values: "blur", "blur_no_scale", "median", "gaussian", "bilateral". The default value is "gaussian".

param1, param2, param3, and param4 are parameters whose meanings depend on smooth type. param1 and param2 accept integer positive values or 0, param3 and param4 accept float values.

The default value for param1 is 3, the default value for the other parameters is 0.

These parameters correspond to the parameters assigned to the libopencv function "cvSmooth".

overlay

Overlay one video on top of another.

It takes two inputs and one output, the first input is the "main" video on which the second input is overlayed.

It accepts the parameters: x:y.

x is the x coordinate of the overlayed video on the main video, y is the y coordinate. The parameters are expressions containing the following parameters:

main_w, main_h
main input width and height

W, H
same as main_w and main_h

overlay_w, overlay_h
overlay input width and height

w, h
same as overlay_w and overlay_h

Be aware that frames are taken from each input video in timestamp order, hence, if their initial timestamps differ, it is a a good idea to pass the two inputs through a setpts=PTS-STARTPTS filter to have them begin in the same zero timestamp, as it does the example for the movie filter.

Follow some examples:

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

# insert a transparent PNG logo in the bottom left corner of the input
movie=logo.png [logo];
[in][logo] overlay=10:main_h-overlay_h-10 [out]

# insert 2 different transparent PNG logos (second logo on bottom
# right corner):
movie=logo1.png [logo1];
movie=logo2.png [logo2];
[in][logo1]       overlay=10:H-h-10 [in+logo1];
[in+logo1][logo2] overlay=W-w-10:H-h-10 [out]

# 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]
```

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

pad

Add paddings to the input image, and places the original input at the given coordinates x, y.

It accepts the following parameters: width:height:x:y:color.

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

E, PI, PHI
the corresponding mathematical approximated values for e (euler number), pi (greek PI), phi (golden ratio)

in_w, in_h
the input video width and height

iw, ih
same as in_w and in_h

out_w, out_h
the output width and height, that is the size of the padded area as specified by the width and height expressions

ow, oh
same as out_w and out_h

x, y
x and y offsets as specified by the x and y expressions, or NAN if not yet specified

a input display aspect ratio, same as iw / ih

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

Follows the description of the accepted parameters.

width, height
Specify the size of the output image with the paddings added. If the value for width or height is 0, the corresponding input size is used for the output.

The width expression can reference the value set by the height expression, and vice versa.

The default value of width and height is 0.

x, y
Specify the offsets where to place the input image in the padded area with respect to the top/left border of the output image.

The x expression can reference the value set by the y expression, and vice versa.

The default value of x and y is 0.

color
Specify the color of the padded area, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence.

The default value of color is "black".

Some examples follow:

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

# pad the input to get an output with dimensions increased bt 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 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"
```

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.

scale

Scale the input video to *width*:*height* and/or convert the image format.

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

E, PI, PHI
the corresponding mathematical approximated values for e (euler number), pi (greek PI), phi (golden ratio)

in_w, in_h
the input width and height

iw, ih
same as *in_w* and *in_h*

out_w, out_h
the output (cropped) width and height

ow, oh
same as *out_w* and *out_h*

dar, a
input display aspect ratio, same as *iw* / *ih*

sar input sample aspect ratio

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

If the input image format is different from the format requested by the next filter, the scale filter will convert the input to the requested format.

If the value for *width* or *height* is 0, the respective input size is used for the output.

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

The default value of *width* and *height* is 0.

Some examples follow:

```
# scale the input video to a size of 200x100.
scale=200:100

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

# scale the input to half size
scale=iw/2:ih/2

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

# seek for Greek harmony
scale=iw:1/PHI*iw
scale=ih*PHI:ih

# increase the height, and set the width to 3/2 of the height
scale=3/2*oh:3/5*ih

# increase the size, but make the size a multiple of the chroma
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"

# increase the width to a maximum of 500 pixels, keep the same input aspect ratio
```

```
# increase the width to a maximum of 500 pixels, keep the same input aspect ratio
scale='min(500, iw*3/2):-1'
```

select

Select frames to pass in output.

It accepts in input an expression, which is evaluated for each input frame. If the expression is evaluated to a non-zero value, the frame is selected and passed to the output, otherwise it is discarded.

The expression can contain the following constants:

PI Greek PI

PHI golden ratio

E Euler number

n the sequential number of the filtered frame, starting from 0

selected_n
the sequential number of the selected frame, starting from 0

prev_selected_n
the sequential number of the last selected frame, NAN if undefined

TB timebase of the input timestamps

pts the PTS (Presentation TimeStamp) of the filtered video frame, expressed in TB units, NAN if undefined

t the PTS (Presentation TimeStamp) of the filtered video frame, expressed in seconds, NAN if undefined

prev_pts
the PTS of the previously filtered video frame, NAN if undefined

prev_selected_pts
the PTS of the last previously filtered video frame, NAN if undefined

prev_selected_t
the PTS of the last previously selected video frame, NAN if undefined

start_pts
the PTS of the first video frame in the video, NAN if undefined

start_t
the time of the first video frame in the video, NAN if undefined

pict_type
the type of the filtered frame, can assume one of the following values:

I
P
B
S
SI
SP
BI

interlace_type
the frame interlace type, can assume one of the following values:

PROGRESSIVE
the frame is progressive (not interlaced)

TOPFIRST
the frame is top-field-first

BOTTOMFIRST
the frame is bottom-field-first

key 1 if the filtered frame is a key-frame, 0 otherwise

pos the position in the file of the filtered frame, -1 if the information is not available (e.g. for synthetic video)

The default value of the select expression is "1".

Some examples follow:

```
# select all frames in input
select

# the above is the same as:
select=1

# skip all frames:
select=0

# select only I-frames
select='eq(pict_type,I)'

# select one frame every 100
select='not(mod(n,100))'

# select only frames contained in the 10-20 time interval
select='gte(t,10)*lte(t,20)'

# select only I frames contained in the 10-20 time interval
select='gte(t,10)*lte(t,20)*eq(pict_type,I)'

# select frames with a minimum distance of 10 seconds
select='isnan(prev_selected_t)+gte(t-prev_selected_t,10)'
```

setdar

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.

The filter accepts a parameter string which represents the wanted display aspect ratio. The parameter can be a floating point number string, or an expression of the form num:den, where num and den are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0:1".

For example to change the display aspect ratio to 16:9, specify:

```
setdar=16:9
# the above is equivalent to
setdar=1.77777
```

See also the setsar filter documentation.

setpts

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

Accept in input an expression evaluated through the eval API, which can contain the following constants:

PTS the presentation timestamp in input

PI Greek PI

PHI golden ratio

E Euler number

N the count of the input frame, starting from 0.

STARTPTS
the PTS of the first video frame

INTERLACED
tell if the current frame is interlaced

POS original position in the file of the frame, or undefined if undefined for the current frame

PREV_INPTS
previous input PTS

PREV_OUTPTS

previous output PTS

Some examples follow:

```
# start counting PTS from zero
setpts=PTS-STARTPTS

# fast motion
setpts=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))'
```

setsar

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.

The filter accepts a parameter string which represents the wanted sample aspect ratio. The parameter can be a floating point number string, or an expression of the form num:den, where num and den are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0:1".

For example to change the sample aspect ratio to 10:11, specify:

```
setsar=10:11
```

settb

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

It accepts in input an arithmetic expression representing a rational. The expression can contain the constants "PI", "E", "PHI", "AVTB" (the default timebase), and "intb" (the input timebase).

The default value for the input is "intb".

Follow 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
```

showinfo

Show a line containing various information for each input video frame. The input video is not modified.

The shown line contains a sequence of key/value pairs of the form key:value.

A description of each shown parameter follows:

n sequential number of the input frame, starting from 0

pts Presentation TimeStamp of the input frame, expressed as a number of time base units. The time base unit depends on the filter input
pad.

pad.

pts_time
Presentation TimeStamp of the input frame, expressed as a number of seconds

pos position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic video)

fmt pixel format name

sar sample aspect ratio of the input frame, expressed in the form num/den

s size of the input frame, expressed in the form widthxheight

i interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first)

iskey
1 if the frame is a key frame, 0 otherwise

type
picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, "?" for unknown type). Check also the documentation of the "AVPictureType" enum and of the "av_get_picture_type_char" function defined in libavutil/avutil.h.

checksum
Adler-32 checksum of all the planes of the input frame

plane_checksum
Adler-32 checksum of each plane of the input frame, expressed in the form "[c0 c1 c2 c3]"

slicify
Pass the images of input video on to next video filter as multiple slices.

```
./avconv -i in.avi -vf "slicify=32" out.avi
```

The filter accepts the slice height as parameter. If the parameter is not specified it will use the default value of 16.

Adding this in the beginning of filter chains should make filtering faster due to better use of the memory cache.

transpose
Transpose rows with columns in the input video and optionally flip it.

It accepts a parameter representing an integer, which can assume the values:

0 Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

1 Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

2 Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

3 Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

unsharp
Sharpen or blur the input video.

It accepts the following parameters:
luma_msize_x:luma_msize_y:luma_amount:chroma_msize_x:chroma_msize_y:chroma_amount

Negative values for the amount will blur the input video, while positive values will sharpen. All parameters are optional and default to the equivalent of the string '5:5:1.0:5:5:0.0'.

luma_msize_x
Set the luma matrix horizontal size. It can be an integer between 3 and 13, default value is 5.

luma_msize_y
Set the luma matrix vertical size. It can be an integer between 3 and 13, default value is 5.

luma_amount
Set the luma effect strength. It can be a float number between -2.0 and 5.0, default value is 1.0.

chroma_msize_x
Set the chroma matrix horizontal size. It can be an integer between 3 and 13, default value is 5.

chroma_msize_y
Set the chroma matrix vertical size. It can be an integer between 3 and 13, default value is 5.

luma_amount
Set the chroma effect strength. It can be a float number between -2.0 and 5.0, default value is 0.0.

```
# Strong luma sharpen effect parameters
unsharp=7:7:2.5

# Strong blur of both luma and chroma parameters
unsharp=7:7:-2:7:7:-2

# Use the default values with B<avconv>
./avconv -i in.avi -vf "unsharp" out.mp4
```

vflip
Flip the input video vertically.

```
./avconv -i in.avi -vf "vflip" out.avi
```

yadif
Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

It accepts the optional parameters: mode:parity:auto.

mode specifies the interlacing mode to adopt, accepts one of the following values:

0 output 1 frame for each frame

1 output 1 frame for each field

2 like 0 but skips spatial interlacing check

3 like 1 but skips spatial interlacing check

Default value is 0.

parity specifies the picture field parity assumed for the input interlaced video, accepts one of the following values:

0 assume top field first

1 assume bottom field first

-1 enable automatic detection

Default value is -1. If interlacing is unknown or decoder does not export this information, top field first will be assumed.

auto specifies if deinterlacer should trust the interlaced flag and only deinterlace frames marked as interlaced

0 deinterlace all frames

1 only deinterlace frames marked as interlaced

Default value is 0.

VIDEO SOURCES

Below is a description of the currently available video sources.

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:height:pix_fmt_string:timebase_num:timebase_den:sample_aspect_ratio_num:sample_aspect_ratio.den

All the parameters need to be explicitly defined.

Follows the list of the accepted parameters.

width, height
Specify the width and height of the buffered video frames.

pix_fmt_string
A string representing the pixel format of the buffered video frames. It may be a number corresponding to a pixel format, or a pixel format name.

timebase_num, timebase_den
Specify numerator and denomitor of the timebase assumed by the timestamps of the buffered frames.

sample_aspect_ratio.num, sample_aspect_ratio.den
Specify numerator and denominator of the sample aspect ratio assumed by the video frames.

For example:

```
buffer=320:240:yuv410p:1:24:1:1
```

will instruct the source to accept video frames with size 320x240 and with format "yuv410p", assuming 1/24 as the timestamps timebase and square pixels (1:1 sample aspect ratio). Since the pixel format with name "yuv410p" corresponds to the number 6 (check the enum PixelFormat definition in libavutil/pixfmt.h), this example corresponds to:

```
buffer=320:240:6:1:24
```

color

Provide an uniformly colored input.

It accepts the following parameters: color:frame_size:frame_rate

Follows the description of the accepted 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".

frame_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".

frame_rate
Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format frame_rate_num/frame_rate_den, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

For example 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 overlayed over the source connected to the pad with identifier "in".

```
"color=red@0.2:qcif:10 [color]; [in][color] overlay [out]"
```

movie

Read a video stream from a movie container.

Read a video stream from a movie container.

It accepts the syntax: movie_name[:options] where movie_name is the name of the resource to read (not necessarily a file but also a device or a stream accessed through some protocol), and options is an optional sequence of key=value pairs, separated by ":".

The description of the accepted options follows.

format_name, f
: Specifies the format assumed for the movie to read, and can be either the name of a container or an input device. If not specified the format is guessed from movie_name or by probing.

seek_point, sp
: Specifies the seek point in seconds, the frames will be output starting from this seek point, the parameter is evaluated with "av_strtod" so the numerical value may be suffixed by an IS postfix. Default value is "0".

stream_index, si
: Specifies the index of the video stream to read. If the value is -1, the best suited video stream will be automatically selected. Default value is "-1".

This filter allows to overlay a second video on top of main input of a filtergraph as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                    ^
                                    |
movie --> scale--> deltapts1 -------+
```

Some examples follow:

```
# skip 3.2 seconds from the start of the avi file in.avi, and overlay it
# on top of the input labelled as "in".
movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [movie];
[in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]

# read from a video4linux2 device, and overlay it on top of the input
# labelled as "in"
movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [movie];
[in] setpts=PTS-STARTPTS, [movie] overlay=16:16 [out]
```

nullsrc

Null video source, never return images. It is mainly useful as a template and to be employed in analysis / debugging tools.

It accepts as optional parameter a string of the form width:height:timebase.

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", "AVTB" (the default timebase), and defaults to the value "AVTB".

frei0r_src

Provide a frei0r source.

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

The source supports the syntax:

```
<size>:<rate>:<src_name>[{=|:}<param1>:<param2>:...:<paramN>]
```

size is the size of the video to generate, may be a string of the form widthxheight or a frame size abbreviation. rate is the rate of the video to generate, may be a string of the form num/den or a frame rate abbreviation. src_name is the name to the frei0r source to load. For more information regarding frei0r and how to set the parameters read the section frei0r in the description of the video filters.

Some examples follow:

```
# generate a frei0r partik0l source with size 200x200 and framerate 10
# which is overlayed on the overlay filter main input
```

```
frei0r_src=200x200:10:partik0l=1234 [overlay]; [in][overlay] overlay
```

rgbtestsrc, 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.

Both sources accept an optional sequence of key=value pairs, separated by ":". The description of the accepted options follows.

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 float number or a valid video frame rate abbreviation. The default value is "25".

sar Set the sample aspect ratio of the sourced video.

duration
Set the video duration of the sourced video. The accepted syntax is:

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

See also the function "av_parse_time()".

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

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.

VIDEO SINKS

Below is a description of the currently available video sinks.

nullsink

Null video sink, do absolutely nothing with the input video. It is mainly useful as a template and to be employed in analysis / debugging tools.

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
```

SEE ALSO

avplay(1), avprobe(1) and the Libav HTML documentation

AUTHORS

The Libav developers