FFmpeg Documentation

Table of Contents

FFmpeg Documentation

1. Synopsis

The generic syntax is:

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

2. Description

FFmpeg is a very fast video and audio converter. It can also grab from a live audio/video source.

The command line interface is designed to be intuitive, in the sense that FFmpeg tries to figure out all parameters that can possibly be derived automatically. You usually only have to specify the target bitrate you want.

FFmpeg can also convert from any sample rate to any other, and resize video on the fly with a high quality polyphase filter.

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.

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

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

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

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

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

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

The format option may be needed for raw input files.

By default, FFmpeg tries to convert as losslessly as possible: It uses the same audio and video parameters for the outputs as the one specified for the inputs.

3. Options

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 commandline will set to false the boolean option with name "foo".

3.1 Generic options

These options are shared amongst the ff* 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
`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.

`-loglevel *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 @env{FFMPEG_FORCE_NOCOLOR} or @env{NO_COLOR}, or can be forced setting the environment variable @env{FFMPEG_FORCE_COLOR}. The use of the environment variable @env{NO_COLOR} is deprecated and will be dropped in a following FFmpeg version.

3.2 Main options

`-f *fmt*'
Force format.

`-i *filename*'
input file name

`-y'
Overwrite output files.

`-t *duration*'
Restrict the transcoded/captured video sequence to the duration specified in seconds. `hh:mm:ss[.xxx]` syntax is also supported.

`-fs *limit_size*'
Set the file size limit.

`-ss *position*'
Seek to given time position in seconds. `hh:mm:ss[.xxx]` syntax is also supported.

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

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

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

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

`-metadata *key=value*'
Set a metadata key/value pair. For example, for setting the title in the output file:

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

`-v *number*'
Set the logging verbosity level.

`-target *type*'
Specify target file type ("vcd", "svcd", "dvd", "dv", "dv50", "pal-vcd", "ntsc-svcd", ...). All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

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

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

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

`-dframes *number*'
Set the number of data frames to record.

`-scodec *codec*'
Force subtitle codec ('copy' to copy stream).

`-newsubtitle'

Add a new subtitle stream to the current output stream.

`-slang *code*'
Set the ISO 639 language code (3 letters) of the current subtitle stream.

3.3 Video Options

`-b *bitrate*'
Set the video bitrate in bit/s (default = 200 kb/s).

`-vframes *number*'
Set the number of video frames to record.

`-r *fps*'
Set frame rate (Hz value, fraction or abbreviation), (default = 25).

`-s *size*'
Set frame size. The format is `wxh' (ffserver default = 160x128, ffmpeg default = same as source). The following abbreviations are recognized:

`sqcif'
128x96

`qcif'
176x144

`cif'
352x288

`4cif'
704x576

`16cif'
1408x1152

`qqvga'
160x120

`qvga'
320x240

`vga'
640x480

`svga'
800x600

`xga'
1024x768

`uxga'
1600x1200

`qxga'
2048x1536

`sxga'
1280x1024

`qsxga'
2560x2048

`hsxga'
5120x4096

`wvga'
852x480

`wxga'
1366x768

`wsxga'
1600x1024

`wuxga'
1920x1200

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

`-aspect *aspect*'
Set aspect ratio (4:3, 16:9 or 1.3333, 1.7777).

`-croptop *size*'
`-cropbottom *size*'
`-cropleft *size*'
`-cropright *size*'
All the crop options have been removed. Use -vf crop=width:height:x:y instead.

`-padtop *size*'
`-padbottom *size*'
`-padleft *size*'
`-padright *size*'
`-padcolor *hex_color*'
All the pad options have been removed. Use -vf pad=width:height:x:y:color instead.

`-vn'
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:

```
ffmpeg -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*'
Force video codec to *codec*. Use the `copy` special value to tell that the raw codec data must be copied as is.

`-sameq'

Use same video quality as source (implies VBR).

`-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:

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

`-passlogfile *prefix*'
: Set two-pass log file name prefix to *prefix*, the default file name prefix is "ffmpeg2pass". The complete file name will be `PREFIX-N.log', where N is a number specific to the output stream.

`-newvideo'
: Add a new video stream to the current output stream.

`-vlang *code*'
: Set the ISO 639 language code (3 letters) of the current video stream.

`-vf *filter_graph*'
: *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).

3.4 Advanced Video Options

`-pix_fmt *format*'
: Set pixel format. Use 'list' as parameter to show all the supported pixel formats.

`-sws_flags *flags*'
: Set SwScaler flags.

`-g *gop_size*'
: Set the group of pictures size.

`-intra'
: Use only intra frames.

`-vdt *n*'
: Discard threshold.

`-qscale *q*'
: Use fixed video quantizer scale (VBR).

`-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:

```
ffmpeg -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 *override*'
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)

`-dct_algo *algo*'
Set DCT algorithm to *algo*. Available values are:
`0'
FF_DCT_AUTO (default)
`1'
FF_DCT_FASTINT
`2'
FF_DCT_INT
`3'
FF_DCT_MMX
`4'
FF_DCT_MLIB
`5'
FF_DCT_ALTIVEC
`-idct_algo *algo*'
Set IDCT algorithm to *algo*. Available values are:
`0'
FF_IDCT_AUTO (default)
`1'
FF_IDCT_INT
`2'
FF_IDCT_SIMPLE
`3'
FF_IDCT_SIMPLEMMX
`4'
FF_IDCT_LIBMPEG2MMX
`5'
FF_IDCT_PS2
`6'
FF_IDCT_MLIB
`7'
FF_IDCT_ARM
`8'
FF_IDCT_ALTIVEC
`9'
FF_IDCT_SH4
`10'
FF_IDCT_SIMPLEARM
`-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 FFmpeg).

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

`2'
FF_MB_DECISION_RD: rate distortion

`-4mv'
Use four motion vector by macroblock (MPEG-4 only).

`-part'
Use data partitioning (MPEG-4 only).

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

`-strict *strictness*'
How strictly to follow the standards.

`-aic'
Enable Advanced intra coding (h263+).

`-umv'
Enable Unlimited Motion Vector (h263+)

`-deinterlace'
Deinterlace pictures.

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

`-psnr'
Calculate PSNR of compressed frames.

`-vstats'
Dump video coding statistics to `vstats_HHMMSS.log'.

`-vstats_file *file*'
Dump video coding statistics to *file*.

`-top *n*'
top=1/bottom=0/auto=-1 field first

`-dc *precision*'
Intra_dc_precision.

`-vtag *fourcc/tag*'
Force video tag/fourcc.

`-qphist'
Show QP histogram.

`-vbsf *bitstream_filter*'
Bitstream filters available are "dump_extra", "remove_extra", "noise", "h264_mp4toannexb", "imxdump", "mjpegadump", "mjpeg2jpeg".

```
ffmpeg -i h264.mp4 -vcodec copy -vbsf h264_mp4toannexb -an out.h264
```

`-force_key_frames *time*[,*time*...]'
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.

3.5 Audio Options

`-aframes *number*'
Set the number of audio frames to record.

`-ar *freq*'
Set the audio sampling frequency (default = 44100 Hz).

`-ab *bitrate*'
Set the audio bitrate in bit/s (default = 64k).

`-aq *q*'
Set the audio quality (codec-specific, VBR).

`-ac *channels*'
Set the number of audio channels. For input streams it is set by default to 1, for output streams it is set by default to the same number of audio channels in input. If the input file has audio streams with different channel count, the behaviour is undefined.

`-an'
Disable audio recording.

`-acodec *codec*'
Force audio codec to *codec*. Use the `copy` special value to specify that the raw codec data must be copied as is.

`-newaudio'
Add a new audio track to the output file. If you want to specify parameters, do so before `-newaudio` (`-acodec`, `-ab`, etc..). Mapping will be done automatically, if the number of output streams is equal to the number of input streams, else it will pick the first one that matches. You can override the mapping using `-map` as usual. Example:

```
ffmpeg -i file.mpg -vcodec copy -acodec ac3 -ab 384k test.mpg -acodec mp2 -ab 192k -newaudio
```

`-alang *code*'
Set the ISO 639 language code (3 letters) of the current audio stream.

3.6 Advanced Audio options:

`-atag *fourcc/tag*'
Force audio tag/fourcc.

`-absf *bitstream_filter*'
Bitstream filters available are "dump_extra", "remove_extra", "noise", "mp3comp", "mp3decomp".

3.7 Subtitle options:

`-scodec *codec*'
Force subtitle codec ('copy' to copy stream).

`-newsubtitle'
Add a new subtitle stream to the current output stream.

`-slang *code*'
Set the ISO 639 language code (3 letters) of the current subtitle stream.

`-sn'
Disable subtitle recording.

`-sbsf *bitstream_filter*'
Bitstream filters available are "mov2textsub", "text2movsub".

```
ffmpeg -i file.mov -an -vn -sbsf mov2textsub -scodec copy -f rawvideo sub.txt
```

3.8 Audio/Video grab options

`-vc *channel*'
Set video grab channel (DV1394 only).

`-tvstd *standard*'
Set television standard (NTSC, PAL (SECAM)).

`-isync'
Synchronize read on input.

3.9 Advanced options

`-map *input_stream_id*[:*sync_stream_id*]'
Set stream mapping from input streams to output streams. Just enumerate the input streams in the order you want them in the output. *sync_stream_id* if specified sets the input stream to sync against.

`-map_meta_data *outfile*[,*metadata*]:*infile*[,*metadata*]'
Set meta data information of *outfile* from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata* parameters specify, which metadata to copy - (g)lobal (i.e. metadata that applies to the whole file), per-(s)tream, per-(c)hapter or per-(p)rogram. All metadata specifiers other than global must be followed by the stream/chapter/program number. If metadata specifier is omitted, it defaults to global. By default, global metadata is copied from the first input file to all output files, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying. For example to copy metadata from the first stream of the input file to global metadata of the output file:

```
ffmpeg -i in.ogg -map_meta_data 0:0,s0 out.mp3
```

`-map_chapters *outfile*:*infile*'
Copy chapters from *infile* to *outfile*. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter to all output files. Use a negative file index to disable any chapter copying.

`-debug'
Print specific debug info.

`-benchmark'
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.

`-dump'
Dump each input packet.

`-hex'
When dumping packets, also dump the payload.

`-bitexact'
Only use bit exact algorithms (for codec testing).

`-ps *size*'
Set RTP payload size in bytes.

`-re'
Read input at native frame rate. Mainly used to simulate a grab device.

`-loop_input'
Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing.

`-loop_output *number_of_times*'
Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely).

`-threads *count*'
Thread count.

`-vsync *parameter*'

Video sync method. 0 Each frame is passed with its timestamp from the demuxer to the muxer 1 Frames will be duplicated and dropped to achieve exactly the requested constant framerate. 2 Frames are passed through with their timestamp or dropped so as to prevent 2 frames from having the same timestamp -1 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.

`-shortest'
Finish encoding when the shortest input stream ends.

`-dts_delta_threshold'
Timestamp discontinuity delta threshold.

`-muxdelay *seconds*'
Set the maximum demux-decode delay.

`-muxpreload *seconds*'
Set the initial demux-decode delay.

`-streamid *output-stream-index*:*new-value*'
Assign a new value to a stream's stream-id field in the next output file. All stream-id fields are reset to default for each output file. For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

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

3.10 Preset files

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

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

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

First ffmpeg searches for a file named *arg*.ffpreset in the directories `$FFMPEG_DATADIR' (if set), and `$HOME/.ffmpeg', and in the datadir defined at configuration time (usually `PREFIX/share/ffmpeg') in that order. For example, if the argument is `libx264-max`, it will search for the file `libx264-max.ffpreset'.

If no such file is found, then ffmpeg will search for a file named *codec_name*-*arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libx264` and use `-vpre max`, then it will search for the file `libx264-max.ffpreset'.

4. Tips

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

  ```
  ffmpeg -g 3 -r 3 -t 10 -b 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).
- When converting video files, you can use the '-sameq' option which uses the same quality factor in the encoder as in the decoder. It allows almost lossless encoding.

5. Examples

5.1 Video and Audio grabbing

FFmpeg can grab video and audio from devices given that you specify the input format and device.

```
ffmpeg -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 FFmpeg 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.

5.2 X11 grabbing

FFmpeg can grab the X11 display.

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

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

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

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

5.3 Video and Audio file format conversion

* FFmpeg can use any supported file format and protocol as input:

Examples:

* You can use YUV files as input:

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

It will use the files:

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

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

* You can input from a raw YUV420P file:

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

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

* You can output to a raw YUV420P file:

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

* You can set several input files and output files:

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

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

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

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

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

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

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

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

* You can transcode decrypted VOBs:

```
ffmpeg -i snatch_1.vob -f avi -vcodec mpeg4 -b 800k -g 300 -bf 2 -acodec libmp3lame -ab 128k snatch.avi
```

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

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

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

For extracting images from a video:

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

This will extract one video frame per second from the video and will output them in files named `foo-001.jpeg',

`foo-002.jpeg', etc. Images will be rescaled to fit the new WxH values.

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

For creating a video from many images:

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

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

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

```
ffmpeg -i test1.avi -i test2.avi -vcodec copy -acodec copy -vcodec copy -acodec copy test12.avi -newvideo
```

In addition to the first video and audio streams, the resulting output file `test12.avi' will contain the second video and the second audio stream found in the input streams list.

The `-newvideo`, `-newaudio` and `-newsubtitle` options have to be specified immediately after the name of the output file to which you want to add them.

6. Expression Evaluation

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

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

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

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

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

The following functions are available:

`sinh(x)'
`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 previosly 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.

Note that:

* works like AND

+ works like OR

thus

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

is equivalent to

```
A*B + not(A)*C
```

When A evaluates to either 1 or 0, that is the same as

```
A*B + eq(A,0)*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

7. Input Devices

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

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

You can disable all the input devices using the configure option "--disable-indevs", and selectively enable an input device using the option "--enable-indev=*INDEV*", or you can disable a particular input device using the option "--disable-indev=*INDEV*".

The option "-formats" of the ff* tools will display the list of supported input devices (amongst the demuxers).

A description of the currently available input devices follows.

7.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

To enable this input device during configuration you need libasound installed on your system.

This device allows capturing from an ALSA device. The name of the device to capture has to be an ALSA card identifier.

An ALSA identifier has the syntax:

`hw:`*`CARD`*`[,`*`DEV`*`[,`*`SUBDEV`*`]]`

where the *DEV* and *SUBDEV* components are optional.

The three arguments (in order: *CARD*,*DEV*,*SUBDEV*) specify card number or identifier, device number and subdevice number (-1 means any).

To see the list of cards currently recognized by your system check the files `` `/proc/asound/cards' `` and `` `/proc/asound/devices' ``.

For example to capture with `` `ffmpeg' `` from an ALSA device with card id 0, you may run the command:

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

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

7.2 bktr

BSD video input device.

7.3 dv1394

Linux DV 1394 input device.

7.4 jack

JACK input device.

To enable this input device during configuration you need libjack installed on your system.

A JACK input device creates one or more JACK writable clients, one for each audio channel, with name *client_name*:input_*N*, where *client_name* is the name provided by the application, and *N* is a number which identifies the channel. Each writable client will send the acquired data to the FFmpeg input device.

Once you have created one or more JACK readable clients, you need to connect them to one or more JACK writable clients.

To connect or disconnect JACK clients you can use the `` `jack_connect' `` and `` `jack_disconnect' `` programs, or do it through a graphical interface, for example with `` `qjackctl' ``.

To list the JACK clients and their properties you can invoke the command `` `jack_lsp' ``.

Follows an example which shows how to capture a JACK readable client with `` `ffmpeg' ``.

```
# Create a JACK writable client with name "ffmpeg".
$ ffmpeg -f jack -i ffmpeg -y out.wav

# Start the sample jack_metro readable client.
$ jack_metro -b 120 -d 0.2 -f 4000

# List the current JACK clients.
```

```
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

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

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

7.5 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

7.6 oss

Open Sound System input device.

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

For example to grab from `/dev/dsp' using `ffmpeg' use the command:

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

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

7.7 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/video*N*', 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 the command `v4l-info' for Video4Linux2 devices.

If the size for the device is set to 0x0, the input device will try to autodetect the size to use.

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.
ffplay -s 320x240 -f video4linux /dev/video0

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

# Grab and record the input of a video4linux2 device, autoadjust size.
ffmpeg -f video4linux2 -i /dev/video0 out.mpeg
```

7.8 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.

7.9 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 @env{DISPLAY} contains the default display name.

x_offset and *y_offset* specify the offsets of the grabbed area with respect to the top-left border of the X11 screen. They default to 0.

Check the X11 documentation (e.g. man X) for more detailed information.

Use the `dpyinfo' program for getting basic information about the properties of your X11 display (e.g. grep for "name" or "dimensions").

For example to grab from `:0.0' using `ffmpeg':

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

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

8. Output Devices

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

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

You can disable all the output devices using the configure option "--disable-outdevs", and selectively enable an output device using the option "--enable-outdev=*OUTDEV*", or you can disable a particular input device using the option "--disable-outdev=*OUTDEV*".

The option "-formats" of the ff* tools will display the list of enabled output devices (amongst the muxers).

A description of the currently available output devices follows.

8.1 alsa

ALSA (Advanced Linux Sound Architecture) output device.

8.2 oss

OSS (Open Sound System) output device.

9. Protocols

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

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

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

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

A description of the currently available protocols follows.

9.1 concat

Physical concatenation protocol.

Allow to read and seek from many resource in sequence as if they were a unique resource.

A URL accepted by this protocol has the syntax:

`concat:`*URL1*`|`*URL2*`|...|`*URLN*

where *URL1*, *URL2*, ..., *URLN* are the urls of the resource to be concatenated, each one possibly specifying a distinct protocol.

For example to read a sequence of files `split1.mpeg', `split2.mpeg', `split3.mpeg' with `ffplay' use the command:

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

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

9.2 file

File access protocol.

Allow to read from or read to a file.

For example to read from a file `input.mpeg' with `ffmpeg' use the command:

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

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

9.3 gopher

Gopher protocol.

9.4 http

HTTP (Hyper Text Transfer Protocol).

9.5 mmst

MMS (Microsoft Media Server) protocol over TCP.

9.6 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

9.7 md5

MD5 output protocol.

Computes the MD5 hash of the data to be written, and on close writes this to the designated output or stdout if none is specified. It can be used to test muxers without writing an actual file.

Some examples follow.

```
# Write the MD5 hash of the encoded AVI file to the file output.avi.md5.
ffmpeg -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
ffmpeg -i input.flv -f avi -y md5:
```

Note that some formats (typically MOV) require the output protocol to be seekable, so they will fail with the MD5 output protocol.

9.8 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

`pipe:[`*`number`*`]`

number is the number corresponding to the file descriptor of the pipe (e.g. 0 for stdin, 1 for stdout, 2 for stderr). If *number* is not specified, by default the stdout file descriptor will be used for writing, stdin for reading.

For example to read from stdin with `ffmpeg':

```
cat test.wav | ffmpeg -i pipe:0
```

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

For writing to stdout with `ffmpeg':

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

Note that some formats (typically MOV), require the output protocol to be seekable, so they will fail with the pipe output protocol.

9.9 rtmp

Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimeâ€□ dia 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 `ffplay' a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
ffplay rtmp://myserver/vod/sample
```

9.10 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 explicitely configure the build with "--enable-librtmp". If enabled this will replace the native RTMP protocol.

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

The required syntax is:

```
rtmp_proto://server[:port][/app][/playpath] options
```

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

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

For example, to stream a file in real-time to an RTMP server using `ffmpeg':

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

To play the same stream using `ffplay':

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

9.11 rtp

Real-Time Protocol.

9.12 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[?options]
```

options is a `&`-separated list. The following options are supported:

`udp'
: Use UDP as lower transport protocol.

`tcp'
: Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

`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.

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 `ffplay', the streams to display can be chosen with `-vst` *n* and `-ast` *n* for video and audio respectively, and can be switched on the fly by pressing `v` and `a`.

Example command lines:

To watch a stream over UDP, with a max reordering delay of 0.5 seconds:

```
ffplay -max_delay 500000 rtsp://server/video.mp4?udp
```

To watch a stream tunneled over HTTP:

```
ffplay rtsp://server/video.mp4?http
```

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

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

9.13 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.

9.13.1 Muxer

The syntax for a SAP url given to the muxer is:

```
sap://destination[:port][?options]
```

The RTP packets are sent to *destination* on port *port*, or to port 5004 if no port is specified. *options* is a &-separated list. The following options are supported:

`announce_addr=address'
: Specify the destination IP address for sending the announcements to. If omitted, the announcements are sent to the commonly used SAP announcement multicast address 224.2.127.254 (sap.mcast.net), or ff0e::2:7ffe if *destination* is an IPv6 address.

`announce_port=port'
: Specify the port to send the announcements on, defaults to 9875 if not specified.

`ttl=ttl'
: Specify the time to live value for the announcements and RTP packets, defaults to 255.

`same_port=0|1'
: If set to 1, send all RTP streams on the same port pair. If zero (the default), all streams are sent on unique ports, with each stream on a port 2 numbers higher than the previous. VLC/Live555 requires this to be set to 1, to be able to receive the stream. The RTP stack in libavformat for receiving requires all streams to be sent on unique ports.

Example command lines follow.

To broadcast a stream on the local subnet, for watching in VLC:

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

Similarly, for watching in ffplay:

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

And for watching in ffplay, over IPv6:

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

9.13.2 Demuxer

The syntax for a SAP url given to the demuxer is:

```
sap://[address][:port]
```

address is the multicast address to listen for announcements on, if omitted, the default 224.2.127.254 (sap.mcast.net) is used. *port* is the port that is listened on, 9875 if omitted.

The demuxers listens for announcements on the given address and port. Once an announcement is received, it tries to receive that particular stream.

Example command lines follow.

To play back the first stream announced on the normal SAP multicast address:

```
ffplay sap://
```

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

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

9.14 tcp

Trasmission Control Protocol.

9.15 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

`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 udp_set_remote_url later. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received.

Some usage examples of the udp protocol with `ffmpeg' follow.

To stream over UDP to a remote endpoint:

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

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

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

To receive over UDP from a remote endpoint:

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

10. Bitstream Filters

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

You can disable all the bitstream filters using the configure option `--disable-bsfs`, and selectively enable any bitstream filter using the option `--enable-bsf=BSF`, or you can disable a particular bitstream filter using the option `--disable-bsf=BSF`.

The option `-bsfs` of the ff* tools will display the list of all the supported bitstream filters included in your build.

Below is a description of the currently available bitstream filters.

10.1 aac_adtstoasc

10.2 chomp

10.3 dump_extradata

10.4 h264_mp4toannexb

10.5 imx_dump_header

10.6 mjpeg2jpeg

Convert MJPEG/AVI1 packets to full JPEG/JFIF packets.

MJPEG is a video codec wherein each video frame is essentially a JPEG image. The individual frames can be extracted without loss, e.g. by

```
ffmpeg -i ../some_mjpeg.avi -vcodec copy frames_%d.jpg
```

Unfortunately, these chunks are incomplete JPEG images, because they lack the DHT segment required for decoding. Quoting from http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml:

Avery Lee, writing in the rec.video.desktop newsgroup in 2001, commented that "MJPEG, or at least the MJPEG in AVIs having the MJPG fourcc, is restricted JPEG with a fixed -- and *omitted* -- Huffman table. The JPEG must be YCbCr colorspace, it must be 4:2:2, and it must use basic Huffman encoding, not arithmetic or progressive. . . . You can indeed extract the MJPEG frames and decode them with a regular JPEG decoder, but you have to prepend the DHT segment to them, or else the decoder won't have any idea how to decompress the data. The exact table necessary is given in the OpenDML spec."

This bitstream filter patches the header of frames extracted from an MJPEG stream (carrying the AVI1 header ID and lacking a DHT segment) to produce fully qualified JPEG images.

```
ffmpeg -i mjpeg-movie.avi -vcodec copy -vbsf mjpeg2jpeg frame_%d.jpg
exiftran -i -9 frame*.jpg
ffmpeg -i frame_%d.jpg -vcodec copy rotated.avi
```

10.7 mjpega_dump_header

10.8 movsub

10.9 mp3_header_compress

10.10 mp3_header_decompress

10.11 noise

10.12 remove_extradata

11. 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".

11.1 Filtergraph syntax

A filtergraph can be represented using a textual representation, which is recognized by the `-vf` and `-af` options of the ff* tools, 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]
```

12. Audio Filters

When you configure your FFmpeg build, you can disable any of the existing filters using --disable-filters. The configure output will show the audio filters included in your build.

Below is a description of the currently available audio filters.

12.1 anull

Pass the audio source unchanged to the output.

13. Audio Sources

Below is a description of the currently available audio sources.

13.1 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
```

14. Audio Sinks

Below is a description of the currently available audio sinks.

14.1 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.

15. Video Filters

When you configure your FFmpeg build, you can disable any of the existing filters using --disable-filters. The configure output will show the video filters included in your build.

Below is a description of the currently available video filters.

15.1 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.

15.2 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 heigth

`iw, ih'
: same as *in_w* and *in_h*

`out_w, out_h'
: the output (cropped) width and heigth

`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 lefth 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 and position
"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)"
```

15.3 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.

15.4 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"
```

15.5 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.

15.6 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".

The following command:

```
./ffmpeg -i in.avi -vf "format=yuv420p" out.avi
```

will convert the input video to the format "yuv420p".

@anchor{frei0r}

15.7 frei0r

Apply a frei0r effect to the input video.

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

The filter supports the syntax:

filter_name[{:|=}*param1*:*param2*:...:*paramN*]

filter_name is the name to the frei0r effect to load. If the environment variable @env{FREI0R_PATH} is defined, the frei0r effect is searched in each one of the directories specified by the colon separated list in @env{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

15.8 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.

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
```

15.9 hflip

Flip the input video horizontally.

For example to horizontally flip the video in input with `ffmpeg':

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

15.10 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*

15.11 noformat

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

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

The following command:

```
./ffmpeg -i in.avi -vf "noformat=yuv420p, vflip" out.avi
```

will make libavfilter use a format different from "yuv420p" for the input to the vflip filter.

15.12 null

Pass the video source unchanged to the output.

15.13 ocv_smooth

Apply smooth transform using libopencv.

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

The filter accepts 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`. Refer to the official libopencv documentation for the exact meaning of the parameters: http://opencv.willowgarage.com/documentation/c/image_filtering.html

15.14 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=0:png: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=0:png:logo1.png [logo1];
movie=0:png: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 togheter more overlays but the efficiency of such approach is yet to be tested.

15.15 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*.

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 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 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".

For example:

```
# 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
# row 0, column 40.
```

```
pad=640:480:0:40:violet
```

15.16 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.

15.17 scale

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

For example the command:

```
./ffmpeg -i in.avi -vf "scale=200:100" out.avi
```

will scale the input video to a size of 200x100.

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.

15.18 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))'
```

15.19 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
```

15.20 slicify

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

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

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

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

15.21 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
```

15.22 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 '5:5:1.0:0:0:0.0'.

`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 0.
`chroma_msize_y'
Set the chroma matrix vertical size. It can be an integer between 3 and 13, default value is 0.
`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 ffmpeg
./ffmpeg -i in.avi -vf "unsharp" out.mp4
```

15.23 vflip

Flip the input video vertically.

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

15.24 yadif

yadif is "yet another deinterlacing filter".

It accepts the syntax:

```
yadif=[mode[:parity]]
```

`mode'
: Specify 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'
: 0 if is bottom field first, 1 if the interlaced video is top field first, -1 to enable automatic detection.

16. Video Sources

Below is a description of the currently available video sources.

16.1 buffer

Buffer video frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in `libavfilter/vsrc_buffer.h'.

It accepts the following parameters: *width*:*height*:*pix_fmt_string*:*timebase_num*:*timebase_den*

All the parameters need to be explicitely 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.

For example:

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

will instruct the source to accept video frames with size 320x240 and with format "yuv410p" and assuming 1/24 as the timestamps timebase. 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
```

16.2 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 *widthxheigth*, 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]"
```

16.3 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".

16.4 frei0r_src

Provide a frei0r source.

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

The source supports the syntax:

size:*rate*:*src_name*[{=|:}*param1*:*param2*:...:*paramN*]

size is the size of the video to generate, may be a string of the form *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" (@xref{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
```

17. Video Sinks

Below is a description of the currently available video sinks.

17.1 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.

This document was generated on 14 December 2010 using texi2html 1.56k.