ffprobe Documentation

Table of Contents

1. Synopsis

The generic syntax is:

```
ffprobe [options] ['input_file']
```

2. Description

ffprobe gathers information from multimedia streams and prints it in human- and machine-readable fashion.

For example it can be used to check the format of the container used by a multimedia stream and the format and type of each media stream contained in it.

If a filename is specified in input, ffprobe will try to open and probe the file content. If the file cannot be opened or recognized as a multimedia file, a positive exit code is returned.

ffprobe may be employed both as a standalone application or in combination with a textual filter, which may perform more sophisticated processing, e.g. statistical processing or plotting.

Options are used to list some of the formats supported by ffprobe or for specifying which information to display, and for setting how ffprobe will show it.

ffprobe output is designed to be easily parsable by a textual filter, and consists of one or more sections of a form defined by the selected writer, which is specified by the '`print_format`' option.

Sections may contain other nested sections, and are identified by a name (which may be shared by other sections), and an unique name. See the output of '`sections`'.

Metadata tags stored in the container or in the streams are recognized and printed in the corresponding "FORMAT" or "STREAM" section.

3. Options

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

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

3.1 Stream specifiers

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

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

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

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

Possible forms of stream specifiers are:

‘*`stream_index`*’

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

‘*`stream_type`*`[:`*`stream_index`*`]`’

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

‘`p:`*`program_id`*`[:`*`stream_index`*`]`’

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

‘`#stream_id`’

Matches the stream by format-specific ID.

3.2 Generic options

These options are shared amongst the av* tools.

‘`-L`’

Show license.

‘`-h, -?, -help, --help [arg]`’

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

Possible values of *arg* are:

‘`decoder=decoder_name`’

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

‘`encoder=encoder_name`’

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

‘`demuxer=demuxer_name`’

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

‘`muxer=muxer_name`’

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

‘`-version`’

Show version.

‘`-formats`’

Show available formats.

The fields preceding the format names have the following meanings:

‘`D`’

Decoding available

‘`E`’

Encoding available

‘`-codecs`’

Show all codecs known to libavcodec.

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

‘`-decoders`’

Show available decoders.

‘`-encoders`’

Show all available encoders.

‘`-bsfs`’

Show available bitstream filters.

‘`-protocols`’

Show available protocols.

‘`-filters`’

Show available libavfilter filters.

‘`-pix_fmts`’

Show available pixel formats.

‘`-sample_fmts`’

Show available sample formats.

‘`-layouts`’

Show channel names and standard channel layouts.

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

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

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

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

‘`-report`’

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

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

‘`file`’

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

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

‘`-cpuflags flags (`*`global`*`)`’

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

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

3.3 AVOptions

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

'`generic`'

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

'`private`'

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

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

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

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

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

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

3.4 Codec AVOptions

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

> set bitrate (in bits/s)

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

> set bitrate (in bits/s)

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

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

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

Possible values:

‘`mv4`’

use four motion vector by macroblock (mpeg4)

‘`qpel`’

use 1/4 pel motion compensation

‘`loop`’

use loop filter

‘`qscale`’

use fixed qscale

‘`gmc`’

use gmc

‘`mv0`’

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

‘`input_preserved`’
‘`pass1`’

use internal 2pass ratecontrol in first pass mode

‘`pass2`’

use internal 2pass ratecontrol in second pass mode

‘`gray`’

only decode/encode grayscale

‘`emu_edge`’

don’t draw edges

‘`psnr`’

error[?] variables will be set during encoding

‘`truncated`’

‘`naq`’

normalize adaptive quantization

‘`ildct`’

use interlaced dct

‘`low_delay`’

force low delay

‘`global_header`’

place global headers in extradata instead of every keyframe

‘`bitexact`’

use only bitexact stuff (except (i)dct)

‘`aic`’

h263 advanced intra coding / mpeg4 ac prediction

‘`cbp`’

Deprecated, use mpegvideo private options instead

‘`qprd`’

Deprecated, use mpegvideo private options instead

‘`ilme`’

interlaced motion estimation

‘`cgop`’

closed gop

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

set motion estimation method

Possible values:

‘`zero`’

zero motion estimation (fastest)

‘`full`’

full motion estimation (slowest)

‘`epzs`’

EPZS motion estimation (default)

‘`esa`’

esa motion estimation (alias for full)

‘`tesa`’

tesa motion estimation

‘`dia`’

dia motion estimation (alias for epzs)

‘`log`’

log motion estimation

‘`phods`’

phods motion estimation

‘`x1`’

X1 motion estimation

‘`hex`’

hex motion estimation

‘`umh`’

umh motion estimation

‘`iter`’

iter motion estimation

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

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

set the group of picture size

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

set audio sampling rate (in Hz)

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

set number of audio channels

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

set cutoff bandwidth

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

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

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

video quantizer scale blur (VBR)

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

min video quantizer scale (VBR)

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

max video quantizer scale (VBR)

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

max difference between the quantizer scale (VBR)

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

use ’frames’ B frames

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

qp factor between p and b frames

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

ratecontrol method

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

strategy to choose between I/P/B-frames

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

rtp payload size in bytes

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

workaround not auto detected encoder bugs

Possible values:

‘`autodetect`’
‘`old_msmpeg4`’

some old lavc generated msmpeg4v3 files (no autodetection)

‘`xvid_ilace`’

Xvid interlacing bug (autodetected if fourcc==XVIX)

‘`ump4`’

(autodetected if fourcc==UMP4)

‘`no_padding`’

padding bug (autodetected)

‘`amv`’
‘`ac_vlc`’

illegal vlc bug (autodetected per fourcc)

‘`qpel_chroma`’
‘`std_qpel`’

old standard qpel (autodetected per fourcc/version)

‘`qpel_chroma2`’
‘`direct_blocksize`’

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

‘`edge`’

edge padding bug (autodetected per fourcc/version)

‘`hpel_chroma`’
‘`dc_clip`’
‘`ms`’

workaround various bugs in microsofts broken decoders

‘`trunc`’

trancated frames

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

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

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

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

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

how strictly to follow the standards

Possible values:

‘`very`’

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

‘`strict`’

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

‘`normal`’
‘`unofficial`’

allow unofficial extensions

‘`experimental`’

allow non standardized experimental things

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

qp offset between P and B frames

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

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

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

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

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

‘`-has_b_frames[:stream_specifier] *integer* ()`’

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

use MPEG quantizers instead of H.263

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

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

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

experimental quantizer modulation

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

experimental quantizer modulation

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

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

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

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

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

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

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

set ratecontrol buffer size (in bits)

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

currently useless

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

qp factor between P and I frames

‘`-i_qoffset[:stream_specifier]` *float* (*output,video*)’ qp offset between P and I frames

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

initial complexity for 1-pass encoding

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

DCT algorithm

Possible values:

‘`auto`’

autoselect a good one (default)

‘`fastint`’

fast integer

‘`int`’

accurate integer

‘`mmx`’
‘`altivec`’
‘`faan`’

floating point AAN DCT

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

compresses bright areas stronger than medium ones

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

temporal complexity masking

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

spatial complexity masking

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

inter masking

‘`-dark_mask[:stream_specifier]` *`float`* (*`output,video`*)’ compresses dark areas stronger than medium ones

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

select IDCT implementation

Possible values:

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

floating point AAN IDCT

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

set error concealment strategy

Possible values:

‘`guess_mvs`’

iterative motion vector (MV) search (slow)

‘`deblock`’

use strong deblock filter for damaged MBs

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

prediction method

Possible values:

‘`left`’
‘`plane`’
‘`median`’

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

sample aspect ratio

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

print specific debug info

Possible values:

‘`pict`’

picture info

‘`rc`’

rate control

‘`bitstream`’
‘`mb_type`’

macroblock (MB) type

‘`qp`’

per-block quantization parameter (QP)

‘`mv`’

motion vector

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

error recognition

‘`mmco`’

memory management control operations (H.264)

‘`bugs`’
‘`vis_qp`’

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

‘`vis_mb_type`’

visualize block types

‘`buffers`’

picture buffer allocations

‘`thread_ops`’

threading operations

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

visualize motion vectors (MVs)

Possible values:

‘`pf`’

forward predicted MVs of P-frames

‘`bf`’

forward predicted MVs of B-frames

‘`bb`’

backward predicted MVs of B-frames

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

full pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

sub pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

macroblock compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

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

interlaced dct compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

diamond type & size for motion estimation

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

amount of motion predictors from the previous frame

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

pre motion estimation

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

pre motion estimation compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

diamond type & size for motion estimation pre-pass

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

sub pel motion estimation quality

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

limit motion vectors range (1023 for DivX player)

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

intra quant bias

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

inter quant bias

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

Possible values:

‘`vlc`’

variable length coder / huffman coder

‘`ac`’

arithmetic coder

‘`raw`’

raw (no encoding)

‘`rle`’

run-length coder

‘`deflate`’

deflate-based coder

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

context model

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

macroblock decision algorithm (high quality mode)

Possible values:

‘`simple`’

use mbcmp (default)

‘`bits`’

use fewest bits

‘`rd`’

use best rate distortion

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

scene change threshold

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

min lagrange factor (VBR)

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

max lagrange factor (VBR)

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

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

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

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

Possible values:

‘`fast`’

allow non spec compliant speedup tricks

‘`sgop`’

Deprecated, use mpegvideo private options instead

‘`noout`’

skip bitstream encoding

‘`local_header`’

place global headers at every keyframe instead of in extradata

‘`chunks`’

Frame data might be split into multiple chunks

‘`showall`’

Show all frames before the first keyframe

‘`skiprd`’

Deprecated, use mpegvideo private options instead

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

deprecated, use mpegvideo private options instead

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

Possible values:

‘`auto`’

detect a good number of threads

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

motion estimaton threshold

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

macroblock threshold

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

intra_dc_precision

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

nsse weight

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

number of macroblock rows at the top which are skipped

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

number of macroblock rows at the bottom which are skipped

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

Possible values:

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

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

Possible values:

‘`unknown`’

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

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

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

frame skip threshold

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

frame skip factor

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

frame skip exponent

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

frame skip compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

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

increases the quantizer for macroblocks close to borders

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

min macroblock lagrange factor (VBR)

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

max macroblock lagrange factor (VBR)

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

motion estimation bitrate penalty compensation (1.0 = 256)

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

Possible values:

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

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

Possible values:

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

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

Possible values:

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

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

refine the two motion vectors used in bidirectional macroblocks

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

downscales frames for dynamic B-frame decision

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

minimum interval between IDR-frames

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

reference frames to consider for motion compensation

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

chroma qp offset from luma

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

rate-distortion optimal quantization

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

multiplied by qscale for each frame and added to scene_change_score

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

adjusts sensitivity of b_frame_strategy 1

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

GOP timecode frame start number, in non drop frame format

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

set desired number of audio channels

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

Possible values:

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

Possible values:

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

set the log level offset

‘-slices[:stream_specifier] *integer* (*output,video*)’ number of slices, used in parallelized encoding

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

select multithreading type

Possible values:

‘`slice`’
‘`frame`’

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

audio service type

Possible values:

‘`ma`’

Main Audio Service

‘`ef`’

Effects

‘`vi`’

Visually Impaired

‘`hi`’

Hearing Impaired

‘`di`’

Dialogue

‘`co`’

Commentary

‘`em`’

Emergency

‘`vo`’

Voice Over

‘`ka`’

Karaoke

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

sample format audio decoders should prefer

Possible values:

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

3.5 Format AVOptions

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

Possible values:

‘`direct`’

reduce buffering

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

set probing size

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

set packet size

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

Possible values:

‘`ignidx`’

ignore index

‘`genpts`’

generate pts

‘`nofillin`’

do not fill in missing values that can be exactly calculated

‘`noparse`’

disable AVParsers, this needs nofillin too

‘`igndts`’

ignore dts

‘`discardcorrupt`’

discard corrupted frames

‘`sortdts`’

try to interleave outputted packets by dts

‘`keepside`’

dont merge side data

‘`latm`’

enable RTP MP4A-LATM payload

‘`nobuffer`’

reduce the latency introduced by optional buffering

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

how many microseconds are analyzed to estimate duration

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

decryption key

‘`-indexmem` *`integer`* (*`input`*)’

max memory used for timestamp index (per stream)

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

max memory used for buffering real-time frames

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

print specific debug info

Possible values:

‘`ts`’

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

maximum muxing or demuxing delay in microseconds

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

number of frames used to probe fps

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

microseconds by which audio packets should be interleaved earlier

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

microseconds for each chunk

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

size in bytes for each chunk

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

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

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

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

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

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

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

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

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

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

use wallclock as timestamps

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

avoid negative timestamps

‘`-skip_initial_bytes` *`integer`* `(`*`input`*`)`’ skip initial bytes

3.6 Main options

‘`-f` *`format`*’

Force format to use.

‘`-unit`’

Show the unit of the displayed values.

‘`-prefix`’

Use SI prefixes for the displayed values. Unless the "-byte_binary_prefix" option is used all the prefixes are decimal.

‘`-byte_binary_prefix`’

Force the use of binary prefixes for byte values.

‘`-sexagesimal`’

Use sexagesimal format HH:MM:SS.MICROSECONDS for time values.

‘`-pretty`’

Prettify the format of the displayed values, it corresponds to the options "-unit -prefix -byte_binary_prefix -sexagesimal".

‘`-of, -print_format` *`writer_name`*`[=`*`writer_options`*`]`’

Set the output printing format.

writer_name specifies the name of the writer, and *writer_options* specifies the options to be passed to the writer.

For example for printing the output in JSON format, specify:

```
-print_format json
```

For more details on the available output printing formats, see the Writers section below.

‘`-sections`’

Print sections structure and section information, and exit. The output is not meant to be parsed by a machine.

‘-select_streams *stream_specifier*’

Select only the streams specified by *stream_specifier*. This option affects only the options related to streams (e.g. show_streams, show_packets, etc.).

For example to show only audio streams, you can use the command:

```
ffprobe -show_streams -select_streams a INPUT
```

To show only video packets belonging to the video stream with index 1:

```
ffprobe -show_packets -select_streams v:1 INPUT
```

‘-show_data’

Show payload data, as an hexadecimal and ASCII dump. Coupled with ‘-show_packets’, it will dump the packets’ data. Coupled with ‘-show_streams’, it will dump the codec extradata.

The dump is printed as the "data" field. It may contain newlines.

‘-show_error’

Show information about the error found when trying to probe the input.

The error information is printed within a section with name "ERROR".

‘-show_format’

Show information about the container format of the input multimedia stream.

All the container format information is printed within a section with name "FORMAT".

‘-show_format_entry *name*’

Like ‘-show_format’, but only prints the specified entry of the container format information, rather than all. This option may be given more than once, then all specified entries will be shown.

This option is deprecated, use show_entries instead.

‘-show_entries *section_entries*’

Set list of entries to show.

Entries are specified according to the following syntax. *section_entries* contains a list of section entries separated by `:`. Each section entry is composed by a section name (or unique name), optionally followed by a list of entries local to that section, separated by `,`.

If section name is specified but is followed by no `=`, all entries are printed to output, together with all the contained sections. Otherwise only the entries specified in the local section entries list are printed. In particular, if `=` is specified but the list of local entries is empty, then no entries will be shown for that section.

Note that the order of specification of the local section entries is not honored in the output, and the usual display order will be retained.

The formal syntax is given by:

```
LOCAL_SECTION_ENTRIES ::= SECTION_ENTRY_NAME[,LOCAL_SECTION_ENTRIES]
SECTION_ENTRY         ::= SECTION_NAME[=[LOCAL_SECTION_ENTRIES]]
SECTION_ENTRIES       ::= SECTION_ENTRY[:SECTION_ENTRIES]
```

For example, to show only the index and type of each stream, and the PTS time, duration time, and stream index of the packets, you can specify the argument:

```
packet=pts_time,duration_time,stream_index : stream=index,codec_type
```

To show all the entries in the section "format", but only the codec type in the section "stream", specify the argument:

```
format : stream=codec_type
```

To show all the tags in the stream and format sections:

```
format_tags : format_tags
```

To show only the `title` tag (if available) in the stream sections:

```
stream_tags=title
```

‘`-show_packets`’

Show information about each packet contained in the input multimedia stream.

The information for each single packet is printed within a dedicated section with name "PACKET".

‘`-show_frames`’

Show information about each frame contained in the input multimedia stream.

The information for each single frame is printed within a dedicated section with name "FRAME".

‘-show_streams’

Show information about each media stream contained in the input multimedia stream.

Each media stream information is printed within a dedicated section with name "STREAM".

‘-count_frames’

Count the number of frames per stream and report it in the corresponding stream section.

‘-count_packets’

Count the number of packets per stream and report it in the corresponding stream section.

‘-show_private_data, -private’

Show private data, that is data depending on the format of the particular shown element. This option is enabled by default, but you may need to disable it for specific uses, for example when creating XSD-compliant XML output.

‘-show_program_version’

Show information related to program version.

Version information is printed within a section with name "PROGRAM_VERSION".

‘-show_library_versions’

Show information related to library versions.

Version information for each library is printed within a section with name "LIBRARY_VERSION".

‘-show_versions’

Show information related to program and library versions. This is the equivalent of setting both ‘-show_program_version’ and ‘-show_library_versions’ options.

‘-bitexact’

Force bitexact output, useful to produce output which is not dependent on the specific build.

‘-i *input_file*’

Read *input_file*.

4. Writers

A writer defines the output format adopted by `ffprobe`, and will be used for printing all the parts of the output.

A writer may accept one or more arguments, which specify the options to adopt. The options are specified as a list of *key*=*value* pairs, separated by ":".

A description of the currently available writers follows.

4.1 default

Default format.

Print each section in the form:

```
[SECTION]
key1=val1
...
keyN=valN
[/SECTION]
```

Metadata tags are printed as a line in the corresponding FORMAT or STREAM section, and are prefixed by the string "TAG:".

A description of the accepted options follows.

‘`nokey, nk`’

If set to 1 specify not to print the key of each field. Default value is 0.

‘`noprint_wrappers, nw`’

If set to 1 specify not to print the section header and footer. Default value is 0.

4.2 compact, csv

Compact and CSV format.

The `csv` writer is equivalent to `compact`, but supports different defaults.

Each section is printed on a single line. If no option is specifid, the output has the form:

```
section|key1=val1| ... |keyN=valN
```

Metadata tags are printed in the corresponding "format" or "stream" section. A metadata tag key, if printed, is prefixed by the string "tag:".

The description of the accepted options follows.

‘`item_sep, s`’

Specify the character to use for separating fields in the output line. It must be a single printable character, it is "|" by default ("," for the `csv` writer).

‘`nokey, nk`’

If set to 1 specify not to print the key of each field. Its default value is 0 (1 for the `csv` writer).

‘`escape, e`’

Set the escape mode to use, default to "c" ("csv" for the `csv` writer).

It can assume one of the following values:

‘`c`’

Perform C-like escaping. Strings containing a newline (’\n’), carriage return (’\r’), a tab (’\t’), a form feed (’\f’), the escaping character (’\’) or the item separator character *SEP* are escaped using C-like fashioned escaping, so that a newline is converted to the sequence "\n", a carriage return to "\r", ’\’ to "\\" and the separator *SEP* is converted to "*SEP*".

‘`csv`’

Perform CSV-like escaping, as described in RFC4180. Strings containing a newline (’\n’), a carriage return (’\r’), a double quote (’"’), or *SEP* are enclosed in double-quotes.

‘`none`’

Perform no escaping.

‘`print_section, p`’

Print the section name at the begin of each line if the value is `1`, disable it with value set to `0`. Default value is `1`.

4.3 flat

Flat format.

A free-form output where each line contains an explicit key=value, such as "streams.stream.3.tags.foo=bar". The output is shell escaped, so it can be directly embedded in sh scripts as long as the separator character is an alphanumeric character or an underscore (see *sep_char* option).

The description of the accepted options follows.

‘`sep_char, s`’

> Separator character used to separate the chapter, the section name, IDs and potential tags in the printed field key.
>
> Default value is ’.’.

‘`hierarchical, h`’

> Specify if the section name specification should be hierarchical. If set to 1, and if there is more than one section in the current chapter, the section name will be prefixed by the name of the chapter. A value of 0 will disable this behavior.
>
> Default value is 1.

4.4 ini

INI format output.

Print output in an INI based format.

The following conventions are adopted:

- all key and values are UTF-8
- ’.’ is the subgroup separator
- newline, ’\t’, ’\f’, ’\b’ and the following characters are escaped
- ’\’ is the escape character
- ’#’ is the comment indicator
- ’=’ is the key/value separator
- ’:’ is not used but usually parsed as key/value separator

This writer accepts options as a list of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`hierarchical, h`’

> Specify if the section name specification should be hierarchical. If set to 1, and if there is more than one section in the current chapter, the section name will be prefixed by the name of the chapter. A value of 0 will disable this behavior.
>
> Default value is 1.

4.5 json

JSON based format.

Each section is printed using JSON notation.

The description of the accepted options follows.

'`compact, c`'

> If set to 1 enable compact output, that is each section will be printed on a single line. Default value is 0.

For more information about JSON, see http://www.json.org/.

4.6 xml

XML based format.

The XML output is described in the XML schema description file '`ffprobe.xsd`' installed in the FFmpeg datadir.

An updated version of the schema can be retrieved at the url http://www.ffmpeg.org/schema/ffprobe.xsd, which redirects to the latest schema committed into the FFmpeg development source code tree.

Note that the output issued will be compliant to the '`ffprobe.xsd`' schema only when no special global output options ('`unit`', '`prefix`', '`byte_binary_prefix`', '`sexagesimal`' etc.) are specified.

The description of the accepted options follows.

'`fully_qualified, q`'

> If set to 1 specify if the output should be fully qualified. Default value is 0. This is required for generating an XML file which can be validated through an XSD file.

'`xsd_compliant, x`'

> If set to 1 perform more checks for ensuring that the output is XSD compliant. Default value is 0. This option automatically sets '`fully_qualified`' to 1.

For more information about the XML format, see http://www.w3.org/XML/.

5. Timecode

`ffprobe` supports Timecode extraction:

- MPEG1/2 timecode is extracted from the GOP, and is available in the video stream details (‘`-show_streams`’, see *timecode*).
- MOV timecode is extracted from tmcd track, so is available in the tmcd stream metadata (‘`-show_streams`’, see *TAG:timecode*).
- DV, GXF and AVI timecodes are available in format metadata (‘`-show_format`’, see *TAG:timecode*).

6. Syntax

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

6.1 Quoting and escaping

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

- `'` and `\` are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.
- A special character is escaped by prefixing it with a ’\’.
- All characters enclosed between ” are included literally in the parsed string. The quote character `'` itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

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

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

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

6.1.1 Examples

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

  ```
  Crime d\'Amour
  ```

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

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

- Include leading or trailing whitespaces using quoting:

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

- Escaping and quoting can be mixed together:

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

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

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

6.2 Date

The accepted syntax is:

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

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

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

6.3 Time duration

The accepted syntax is:

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

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

6.4 Video size

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

The following abbreviations are recognized:

'`sqcif`'

128x96

‘`qcif`’

176x144

‘`cif`’

352x288

‘`4cif`’

704x576

‘`16cif`’

1408x1152

‘`qqvga`’

160x120

‘`qvga`’

320x240

‘`vga`’

640x480

‘`svga`’

800x600

‘`xga`’

1024x768

‘`uxga`’

1600x1200

‘`qxga`’

2048x1536

‘`sxga`’

1280x1024

‘`qsxga`’

2560x2048

‘`hsxga`’

5120x4096

‘`wvga`’

852x480

‘`wxga`’

1366x768

‘`wsxga`’

1600x1024

‘`wuxga`’

1920x1200

‘`woxga`’

2560x1600

‘`wqsxga`’

3200x2048

‘`wquxga`’

3840x2400

‘`whsxga`’

6400x4096

‘`whuxga`’

7680x4800

‘`cga`’

320x200

‘`ega`’

640x350

‘`hd480`’

852x480

‘`hd720`’

1280x720

‘`hd1080`’

1920x1080

6.5 Video rate

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

The following abbreviations are recognized:

‘`ntsc`’

30000/1001

‘`pal`’

25/1

‘`qntsc`’

30000/1

‘`qpal`’

25/1

‘`sntsc`’

30000/1

‘`spal`’

25/1

‘`film`’

24/1

‘`ntsc-film`’

24000/1

6.6 Ratio

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

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

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

6.7 Color

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

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

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

7. Decoders

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

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

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

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

8. Video Decoders

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

8.1 rawvideo

Raw video decoder.

This decoder decodes rawvideo streams.

8.1.1 Options

‘`top` *`top_field_first`*’

> Specify the assumed field type of the input video.
>
> ‘`-1`’
>
> > the video is assumed to be progressive (default)
>
> ‘`0`’
>
> > bottom-field-first is assumed
>
> ‘`1`’
>
> > top-field-first is assumed

9. Audio Decoders

9.1 ffwavesynth

Internal wave synthetizer.

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

10. Demuxers

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

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

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

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

The description of some of the currently available demuxers follows.

10.1 image2

Image file demuxer.

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

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

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

This demuxer accepts the following options:

‘`framerate`’

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

‘`loop`’

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

‘`pattern_type`’

Select the pattern type used to interpret the provided filename.

pattern_type accepts one of the following values.

‘`sequence`’

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

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

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

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

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

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

‘`glob`’

Select a glob wildcard pattern type.

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

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

Select a mixed glob wildcard/sequence pattern.

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

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

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

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

Default value is *glob_sequence*.

‘`pixel_format`’

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

‘`start_number`’

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

‘`start_number_range`’

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

‘`video_size`’

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

10.1.1 Examples

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

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

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

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

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

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

10.2 applehttp

Apple HTTP Live Streaming demuxer.

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

10.3 sbg

SBaGen script demuxer.

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

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

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

11. 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.

11.1 bluray

Read BluRay playlist.

The accepted options are:

'`angle`'

BluRay angle

‘`chapter`’

Start chapter (1...N)

‘`playlist`’

Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

11.2 concat

Physical concatenation protocol.

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

A URL accepted by this protocol has the syntax:

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

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

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

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

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

11.3 file

File access protocol.

Allow to read from or read to a file.

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

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

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

11.4 gopher

Gopher protocol.

11.5 hls

Read Apple HTTP Live Streaming compliant segmented stream as a uniform one. The M3U8 playlists describing the segments can be remote HTTP resources or local files, accessed using the standard file protocol. The nested protocol is declared by specifying "+*proto*" after the hls URI scheme name, where *proto* is either "file" or "http".

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

Using this protocol is discouraged - the hls demuxer should work just as well (if not, please report the issues) and is more complete. To use the hls demuxer instead, simply use the direct URLs to the m3u8 files.

11.6 http

HTTP (Hyper Text Transfer Protocol).

11.7 mmst

MMS (Microsoft Media Server) protocol over TCP.

11.8 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

11.9 md5

MD5 output protocol.

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

Some examples follow.

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

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

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

11.10 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

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

For example to read from stdin with `ffmpeg`:

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

For writing to stdout with `ffmpeg`:

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

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

11.11 rtmp

Real-Time Messaging Protocol.

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

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

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

‘`app`’

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. ‘`/ondemand/`’, ‘`/flash/live/`’, etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

‘`playpath`’

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

Maximum time to wait for the incoming connection. Implies listen.

Additionally, the following parameters can be set via command line options (or in code via `AVOptions`):

‘`rtmp_app`’

Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

‘`rtmp_buffer`’

Set the client buffer time in milliseconds. The default is 3000.

‘`rtmp_conn`’

Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with ’N’ and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘`rtmp_flashver`’

Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘`rtmp_flush_interval`’

Number of packets flushed in the same request (RTMPT only). The default is 10.

‘`rtmp_live`’

Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is `any`, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are `live` and `recorded`.

‘`rtmp_pageurl`’

URL of the web page in which the media was embedded. By default no value will be sent.

‘`rtmp_playpath`’

Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘`rtmp_subscribe`’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘`rtmp_swfhash`’

SHA256 hash of the decompressed SWF file (32 bytes).

‘`rtmp_swfsize`’

Size of the decompressed SWF file, required for SWFVerification.

‘`rtmp_swfurl`’

URL of the SWF player for the media. By default no value will be sent.

‘`rtmp_swfverify`’

URL to player swf file, compute hash/size automatically.

‘`rtmp_tcurl`’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with `ffplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

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

11.12 rtmpe

Encrypted Real-Time Messaging Protocol.

The Encrypted Real-Time Messaging Protocol (RTMPE) is used for streaming multimedia content within standard cryptographic primitives, consisting of Diffie-Hellman key exchange and HMACSHA256, generating a pair of RC4 keys.

11.13 rtmps

Real-Time Messaging Protocol over a secure SSL connection.

The Real-Time Messaging Protocol (RTMPS) is used for streaming multimedia content across an encrypted connection.

11.14 rtmpt

Real-Time Messaging Protocol tunneled through HTTP.

The Real-Time Messaging Protocol tunneled through HTTP (RTMPT) is used for streaming multimedia content within HTTP requests to traverse firewalls.

11.15 rtmpte

Encrypted Real-Time Messaging Protocol tunneled through HTTP.

The Encrypted Real-Time Messaging Protocol tunneled through HTTP (RTMPTE) is used for streaming multimedia content within HTTP requests to traverse firewalls.

11.16 rtmpts

Real-Time Messaging Protocol tunneled through HTTPS.

The Real-Time Messaging Protocol tunneled through HTTPS (RTMPTS) is used for streaming multimedia content within HTTPS requests to traverse firewalls.

11.17 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

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

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

The required syntax is:

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

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

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

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

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

To play the same stream using `ffplay`:

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

11.18 rtp

Real-Time Protocol.

11.19 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock's RTSP server).

The required syntax for a RTSP url is:

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

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

'`udp`'

Use UDP as lower transport protocol.

'`tcp`'

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

'`udp_multicast`'

Use UDP multicast as lower transport protocol.

'`http`'

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

‘`filter_src`’

Accept packets only from negotiated peer address and port.

‘`listen`’

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

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

Example command lines:

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

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

To watch a stream tunneled over HTTP:

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

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

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

To receive a stream in realtime:

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

11.20 sap

Session Announcement Protocol (RFC 2974). This is not technically a protocol handler in libavformat, it is a muxer and demuxer. It is used for signalling of RTP streams, by announcing the SDP for the streams regularly on a separate port.

11.20.1 Muxer

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

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

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

'`announce_addr=address`'

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

'`announce_port=port`'

Specify the port to send the announcements on, defaults to 9875 if not specified.

'`ttl=ttl`'

Specify the time to live value for the announcements and RTP packets, defaults to 255.

'`same_port=0|1`'

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

Example command lines follow.

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

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

Similarly, for watching in `ffplay`:

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

And for watching in `ffplay`, over IPv6:

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

11.20.2 Demuxer

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

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

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

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

Example command lines follow.

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

```
ffplay sap://
```

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

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

11.21 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

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

‘`listen`’

Listen for an incoming connection

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

11.22 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

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

‘`listen`’

Act as a server, listening for an incoming connection.

‘`cafile=filename`’

Certificate authority file. The file must be in OpenSSL PEM format.

‘`cert=filename`’

Certificate file. The file must be in OpenSSL PEM format.

‘`key=filename`’

Private key file.

‘`verify=0|1`’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

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

11.23 udp

User Datagram Protocol.

The required syntax for a UDP url is:

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

options contains a list of &-separated options of the form *key*=*val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

'`buffer_size=`*`size`*'

> Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

'`localport=`*`port`*'

> Override the local UDP port to bind with.

'`localaddr=`*`addr`*'

> Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

'`pkt_size=`*`size`*'

> Set the size in bytes of UDP packets.

'`reuse=`*`1`*`|`*`0`*'

> Explicitly allow or disallow reusing UDP sockets.

'`ttl=`*`ttl`*'

> Set the time to live value (for multicast only).

'`connect=`*`1`*`|`*`0`*'

> Initialize the UDP socket with `connect()`. In this case, the destination address can't be changed with ff_udp_set_remote_url later. If the destination address isn't known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

'`sources=`*`address`*`[,`*`address`*`]`'

> Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘`block=`*`address`*`[,`*`address`*`]`’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘`fifo_size=`*`units`*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘`overrun_nonfatal=`*`1|0`*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

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

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

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

To receive over UDP from a remote endpoint:

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

12. 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.

12.1 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

12.2 bktr

BSD video input device.

12.3 dshow

Windows DirectShow input device.

DirectShow support is enabled when FFmpeg is built with the mingw-w64 project. Currently only audio and video devices are supported.

Multiple devices may be opened as separate inputs, but they may also be opened on the same input, which should improve synchronism between them.

The input name should be in the format:

```
TYPE=NAME[:TYPE=NAME]
```

where *TYPE* can be either *audio* or *video*, and *NAME* is the device’s name.

12.3.1 Options

If no options are specified, the device’s defaults are used. If the device does not support the requested options, it will fail to open.

‘`video_size`’

Set the video size in the captured video.

‘`framerate`’

Set the framerate in the captured video.

‘`sample_rate`’

Set the sample rate (in Hz) of the captured audio.

‘`sample_size`’

Set the sample size (in bits) of the captured audio.

‘`channels`’

Set the number of channels in the captured audio.

‘`list_devices`’

If set to ‘`true`’, print a list of devices and exit.

‘`list_options`’

If set to ‘`true`’, print a list of selected device’s options and exit.

‘`video_device_number`’

Set video device number for devices with same name (starts at 0, defaults to 0).

‘`audio_device_number`’

Set audio device number for devices with same name (starts at 0, defaults to 0).

‘`pixel_format`’

Select pixel format to be used by DirectShow. This may only be set when the video codec is not set or set to rawvideo.

‘`audio_buffer_size`’

Set audio device buffer size in milliseconds (which can directly impact latency, depending on the device). Defaults to using the audio device’s default buffer size (typically some multiple of 500ms). Setting this value too low can degrade performance. See also http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582(v=vs.85).aspx

12.3.2 Examples

- Print the list of DirectShow supported devices and exit:

  ```
  $ ffmpeg -list_devices true -f dshow -i dummy
  ```

- Open video device *Camera*:

  ```
  $ ffmpeg -f dshow -i video="Camera"
  ```

- Open second video device with name *Camera*:

  ```
  $ ffmpeg -f dshow -video_device_number 1 -i video="Camera"
  ```

- Open video device *Camera* and audio device *Microphone*:

  ```
  $ ffmpeg -f dshow -i video="Camera":audio="Microphone"
  ```

- Print the list of supported options in selected device and exit:

  ```
  $ ffmpeg -list_options true -f dshow -i video="Camera"
  ```

12.4 dv1394

Linux DV 1394 input device.

12.5 fbdev

Linux framebuffer input device.

The Linux framebuffer is a graphic hardware-independent abstraction layer to show graphics on a computer monitor, typically on the console. It is accessed through a file device node, usually '`/dev/fb0`'.

For more detailed information read the file Documentation/fb/framebuffer.txt included in the Linux source tree.

To record from the framebuffer device '`/dev/fb0`' with `ffmpeg`:

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

You can take a single screenshot image with the command:

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

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

12.6 iec61883

FireWire DV/HDV input device using libiec61883.

To enable this input device, you need libiec61883, libraw1394 and libavc1394 installed on your system. Use the configure option `--enable-libiec61883` to compile with the device enabled.

The iec61883 capture device supports capturing from a video device connected via IEEE1394 (FireWire), using libiec61883 and the new Linux FireWire stack (juju). This is the default DV/HDV input method in Linux Kernel 2.6.37 and later, since the old FireWire stack was removed.

Specify the FireWire port to be used as input file, or "auto" to choose the first port connected.

12.6.1 Options

'`dvtype`'

Override autodetection of DV/HDV. This should only be used if auto detection does not work, or if usage of a different device type should be prohibited. Treating a DV device as HDV (or vice versa) will not work and result in undefined behavior. The values '`auto`', '`dv`' and '`hdv`' are supported.

'`dvbuffer`'

Set maxiumum size of buffer for incoming data, in frames. For DV, this is an exact value. For HDV, it is not frame exact, since HDV does not have a fixed frame size.

'`dvguid`'

Select the capture device by specifying it’s GUID. Capturing will only be performed from the specified device and fails if no device with the given GUID is found. This is useful to select the input if multiple devices are connected at the same time. Look at /sys/bus/firewire/devices to find out the GUIDs.

12.6.2 Examples

- Grab and show the input of a FireWire DV/HDV device.

  ```
  ffplay -f iec61883 -i auto
  ```

- Grab and record the input of a FireWire DV/HDV device, using a packet buffer of 100000 packets if the source is HDV.

  ```
  ffmpeg -f iec61883 -i auto -hdvbuffer 100000 out.mpg
  ```

12.7 jack

JACK input device.

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

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

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

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

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

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

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

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

# List the current JACK clients.
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

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

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

12.8 lavfi

Libavfilter input virtual device.

This input device reads data from the open output pads of a libavfilter filtergraph.

For each filtergraph open output, the input device will create a corresponding stream which is mapped to the generated output. Currently only video data is supported. The filtergraph is specified through the option ‘`graph`’.

12.8.1 Options

‘`graph`’

Specify the filtergraph to use as input. Each video open output must be labelled by a unique string of the form "out*N*", where *N* is a number starting from 0 corresponding to the mapped input stream generated by the device. The first unlabelled output is automatically assigned to the "out0" label, but all the others need to be specified explicitly.

If not specified defaults to the filename specified for the input device.

‘`graph_file`’

Set the filename of the filtergraph to be read and sent to the other filters. Syntax of the filtergraph is the same as the one specified by the option *graph*.

12.8.2 Examples

- Create a color video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi -graph "color=pink [out0]" dummy
  ```

- As the previous example, but use filename for specifying the graph description, and omit the "out0" label:

  ```
  ffplay -f lavfi color=pink
  ```

- Create three different video test filtered sources and play them:

  ```
  ffplay -f lavfi -graph "testsrc [out0]; testsrc,hflip [out1]; testsrc,negate [out2]" test3
  ```

- Read an audio stream from a file using the amovie source and play it back with `ffplay`:

  ```
  ffplay -f lavfi "amovie=test.wav"
  ```

- Read an audio stream and a video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi "movie=test.avi[out0];amovie=test.wav[out1]"
  ```

12.9 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

12.10 openal

The OpenAL input device provides audio capture on all systems with a working OpenAL 1.1 implementation.

To enable this input device during configuration, you need OpenAL headers and libraries installed on your system, and need to configure FFmpeg with `--enable-openal`.

OpenAL headers and libraries should be provided as part of your OpenAL implementation, or as an additional download (an SDK). Depending on your installation you may need to specify additional flags via the `--extra-cflags` and `--extra-ldflags` for allowing the build system to locate the OpenAL headers and libraries.

An incomplete list of OpenAL implementations follows:

Creative

The official Windows implementation, providing hardware acceleration with supported devices and software fallback. See http://openal.org/.

OpenAL Soft

Portable, open source (LGPL) software implementation. Includes backends for the most common sound APIs on the Windows, Linux, Solaris, and BSD operating systems. See http://kcat.strangesoft.net/openal.html.

Apple

OpenAL is part of Core Audio, the official Mac OS X Audio interface. See http://developer.apple.com/technologies/mac/audio-and-video.html

This device allows to capture from an audio input device handled through OpenAL.

You need to specify the name of the device to capture in the provided filename. If the empty string is provided, the device will automatically select the default device. You can get the list of the supported devices by using the option *list_devices*.

12.10.1 Options

‘`channels`’

Set the number of channels in the captured audio. Only the values ‘`1`’ (monaural) and ‘`2`’ (stereo) are currently supported. Defaults to ‘`2`’.

‘`sample_size`’

Set the sample size (in bits) of the captured audio. Only the values ‘`8`’ and ‘`16`’ are currently supported. Defaults to ‘`16`’.

‘`sample_rate`’

Set the sample rate (in Hz) of the captured audio. Defaults to ‘`44.1k`’.

‘`list_devices`’

If set to ‘`true`’, print a list of devices and exit. Defaults to ‘`false`’.

12.10.2 Examples

Print the list of OpenAL supported devices and exit:

```
$ ffmpeg -list_devices true -f openal -i dummy out.ogg
```

Capture from the OpenAL device ‘`DR-BT101 via PulseAudio`’:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out.ogg
```

Capture from the default device (note the empty string ” as filename):

```
$ ffmpeg -f openal -i ’’ out.ogg
```

Capture from two devices simultaneously, writing to two different files, within the same `ffmpeg` command:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out1.ogg -f openal -i ’ALSA Default’ out2.ogg
```

Note: not all OpenAL implementations support multiple simultaneous capture - try the latest OpenAL Soft if the above does not work.

12.11 oss

Open Sound System input device.

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

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

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

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

12.12 pulse

pulseaudio input device.

To enable this input device during configuration you need libpulse-simple installed in your system.

The filename to provide to the input device is a source device or the string "default"

To list the pulse source devices and their properties you can invoke the command `pactl list sources`.

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

12.12.1 *server* AVOption

The syntax is:

```
-server server name
```

Connects to a specific server.

12.12.2 *name* AVOption

The syntax is:

```
-name application name
```

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

12.12.3 *stream_name* AVOption

The syntax is:

```
-stream_name stream name
```

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

12.12.4 *sample_rate* AVOption

The syntax is:

```
-sample_rate samplerate
```

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

12.12.5 *channels* AVOption

The syntax is:

```
-channels N
```

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

12.12.6 *frame_size* AVOption

The syntax is:

```
-frame_size bytes
```

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

12.12.7 *fragment_size* AVOption

The syntax is:

```
-fragment_size bytes
```

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

12.13 sndio

sndio input device.

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

The filename to provide to the input device is the device node representing the sndio input device, and is usually set to '`/dev/audio0`'.

For example to grab from '`/dev/audio0`' using `ffmpeg` use the command:

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

12.14 video4linux2

Video4Linux2 input video device.

The name of the device to grab is a file device node, usually Linux systems tend to automatically create such nodes when the device (e.g. an USB webcam) is plugged into the system, and has a name of the kind '`/dev/video`*N*', where *N* is a number associated to the device.

Video4Linux2 devices usually support a limited set of *width*x*height* sizes and framerates. You can check which are supported using `-list_formats all` for Video4Linux2 devices.

Some usage examples of the video4linux2 devices with ffmpeg and ffplay:

The time base for the timestamps is 1 microsecond. Depending on the kernel version and configuration, the timestamps may be derived from the real time clock (origin at the Unix Epoch) or the monotonic clock (origin usually at boot time, unaffected by NTP or manual changes to the clock). The '`-timestamps abs`' or '`-ts abs`' option can be used to force conversion into the real time clock.

Note that if FFmpeg is build with v4l-utils support ("–enable-libv4l2" option), it will always be used.

```
# Grab and show the input of a video4linux2 device.
ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0

# Grab and record the input of a video4linux2 device, leave the
framerate and size as previously set.
ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
```

"v4l" and "v4l2" can be used as aliases for the respective "video4linux" and "video4linux2".

12.15 vfwcap

VfW (Video for Windows) capture input device.

The filename passed as input is the capture driver number, ranging from 0 to 9. You may use "list" as filename to print a list of drivers. Any other filename will be interpreted as device number 0.

12.16 x11grab

X11 video input device.

This device allows to capture a region of an X11 display.

The filename passed as input has the syntax:

```
[hostname]:display_number.screen_number[+x_offset,y_offset]
```

hostname:*display_number*.*screen_number* specifies the X11 display name of the screen to grab from. *hostname* can be omitted, and defaults to "localhost". The environment variable `DISPLAY` contains the default display name.

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

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

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

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

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

Grab at position `10,20`:

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

12.16.1 Options

'`draw_mouse`'

Specify whether to draw the mouse pointer. A value of `0` specify not to draw the pointer. Default value is `1`.

'`follow_mouse`'

Make the grabbed area follow the mouse. The argument can be `centered` or a number of pixels *PIXELS*.

When it is specified with "centered", the grabbing region follows the mouse pointer and keeps the pointer at the center of region; otherwise, the region follows only when the mouse pointer reaches within *PIXELS* (greater than zero) to the edge of region.

For example:

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

To follow only when the mouse pointer reaches within 100 pixels to edge:

```
ffmpeg -f x11grab -follow_mouse 100 -r 25 -s cif -i :0.0 out.mpg
```

'`framerate`'

Set the grabbing frame rate. Default value is `ntsc`, corresponding to a framerate of `30000/1001`.

'`show_region`'

Show grabbed region on screen.

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

For example:

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

With *follow_mouse*:

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

‘`video_size`’

Set the video frame size. Default value is `vga`.

This document was generated by *john* on *November 26, 2012* using *texi2html 1.82*.

ffprobe Documentation

Table of Contents

1. Synopsis

The generic syntax is:

```
ffprobe [options] ['input_file']
```

2. Description

ffprobe gathers information from multimedia streams and prints it in human- and machine-readable fashion.

For example it can be used to check the format of the container used by a multimedia stream and the format and type of each media stream contained in it.

If a filename is specified in input, ffprobe will try to open and probe the file content. If the file cannot be opened or recognized as a multimedia file, a positive exit code is returned.

ffprobe may be employed both as a standalone application or in combination with a textual filter, which may perform more sophisticated processing, e.g. statistical processing or plotting.

Options are used to list some of the formats supported by ffprobe or for specifying which information to display, and for setting how ffprobe will show it.

ffprobe output is designed to be easily parsable by a textual filter, and consists of one or more sections of a form defined by the selected writer, which is specified by the '`print_format`' option.

Sections may contain other nested sections, and are identified by a name (which may be shared by other sections), and an unique name. See the output of ‘`sections`’.

Metadata tags stored in the container or in the streams are recognized and printed in the corresponding "FORMAT" or "STREAM" section.

3. Options

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

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

3.1 Stream specifiers

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

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

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

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

Possible forms of stream specifiers are:

‘*`stream_index`*’

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

‘*`stream_type`*`[:`*`stream_index`*`]`’

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

‘`p:`*`program_id`*`[:`*`stream_index`*`]`’

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

‘`#`*`stream_id`*’

Matches the stream by format-specific ID.

3.2 Generic options

These options are shared amongst the av* tools.

‘`-L`’

Show license.

‘`-h, -?, -help, --help [`*`arg`*`]`’

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

Possible values of *arg* are:

‘`decoder=`*`decoder_name`*’

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

‘`encoder=`*`encoder_name`*’

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

‘`demuxer=`*`demuxer_name`*’

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

‘`muxer=`*`muxer_name`*’

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

‘`-version`’

Show version.

‘`-formats`’

Show available formats.

The fields preceding the format names have the following meanings:

‘`D`’

Decoding available

‘`E`’

Encoding available

‘`-codecs`’

Show all codecs known to libavcodec.

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

‘`-decoders`’

Show available decoders.

‘`-encoders`’

Show all available encoders.

‘`-bsfs`’

Show available bitstream filters.

‘`-protocols`’

Show available protocols.

‘`-filters`’

Show available libavfilter filters.

‘`-pix_fmts`’

Show available pixel formats.

‘`-sample_fmts`’

Show available sample formats.

‘`-layouts`’

Show channel names and standard channel layouts.

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

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

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

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

‘`-report`’

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

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

‘`file`’

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

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

‘`-cpuflags flags (`*`global`*`)`’

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

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

3.3 AVOptions

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

'`generic`'

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

'`private`'

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

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

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

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

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

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

3.4 Codec AVOptions

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

> set bitrate (in bits/s)

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

> set bitrate (in bits/s)

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

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

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

Possible values:

‘`mv4`’

use four motion vector by macroblock (mpeg4)

‘`qpel`’

use 1/4 pel motion compensation

‘`loop`’

use loop filter

‘`qscale`’

use fixed qscale

‘`gmc`’

use gmc

‘`mv0`’

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

‘`input_preserved`’
‘`pass1`’

use internal 2pass ratecontrol in first pass mode

‘`pass2`’

use internal 2pass ratecontrol in second pass mode

‘`gray`’

only decode/encode grayscale

‘`emu_edge`’

don’t draw edges

‘`psnr`’

error[?] variables will be set during encoding

‘`truncated`’

‘`naq`’

normalize adaptive quantization

‘`ildct`’

use interlaced dct

‘`low_delay`’

force low delay

‘`global_header`’

place global headers in extradata instead of every keyframe

‘`bitexact`’

use only bitexact stuff (except (i)dct)

‘`aic`’

h263 advanced intra coding / mpeg4 ac prediction

‘`cbp`’

Deprecated, use mpegvideo private options instead

‘`qprd`’

Deprecated, use mpegvideo private options instead

‘`ilme`’

interlaced motion estimation

‘`cgop`’

closed gop

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

set motion estimation method

Possible values:

‘`zero`’

zero motion estimation (fastest)

‘`full`’

full motion estimation (slowest)

‘`epzs`’

EPZS motion estimation (default)

‘`esa`’

esa motion estimation (alias for full)

‘`tesa`’

tesa motion estimation

‘`dia`’

dia motion estimation (alias for epzs)

‘`log`’

log motion estimation

‘`phods`’

phods motion estimation

‘`x1`’

X1 motion estimation

‘`hex`’

hex motion estimation

‘`umh`’

umh motion estimation

‘`iter`’

iter motion estimation

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

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

set the group of picture size

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

set audio sampling rate (in Hz)

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

set number of audio channels

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

set cutoff bandwidth

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

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

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

video quantizer scale blur (VBR)

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

min video quantizer scale (VBR)

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

max video quantizer scale (VBR)

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

max difference between the quantizer scale (VBR)

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

use ’frames’ B frames

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

qp factor between p and b frames

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

ratecontrol method

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

strategy to choose between I/P/B-frames

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

rtp payload size in bytes

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

workaround not auto detected encoder bugs

Possible values:

‘`autodetect`’
‘`old_msmpeg4`’

some old lavc generated msmpeg4v3 files (no autodetection)

‘`xvid_ilace`’

Xvid interlacing bug (autodetected if fourcc==XVIX)

‘`ump4`’

(autodetected if fourcc==UMP4)

‘`no_padding`’

padding bug (autodetected)

‘`amv`’
‘`ac_vlc`’

illegal vlc bug (autodetected per fourcc)

‘`qpel_chroma`’
‘`std_qpel`’

old standard qpel (autodetected per fourcc/version)

‘`qpel_chroma2`’
‘`direct_blocksize`’

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

‘`edge`’

edge padding bug (autodetected per fourcc/version)

‘`hpel_chroma`’
‘`dc_clip`’
‘`ms`’

workaround various bugs in microsofts broken decoders

‘`trunc`’

trancated frames

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

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

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

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

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

how strictly to follow the standards

Possible values:

‘`very`’

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

‘`strict`’

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

‘`normal`’
‘`unofficial`’

allow unofficial extensions

‘`experimental`’

allow non standardized experimental things

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

qp offset between P and B frames

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

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

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

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

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

‘`-has_b_frames[:stream_specifier] *integer* ()`’

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

use MPEG quantizers instead of H.263

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

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

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

experimental quantizer modulation

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

experimental quantizer modulation

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

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

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

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

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

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

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

set ratecontrol buffer size (in bits)

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

currently useless

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

qp factor between P and I frames

‘`-i_qoffset[:stream_specifier]` *float* (*output,video*)’ qp offset between P and I frames

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

initial complexity for 1-pass encoding

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

DCT algorithm

Possible values:

‘`auto`’

autoselect a good one (default)

‘`fastint`’

fast integer

‘`int`’

accurate integer

‘`mmx`’
‘`altivec`’
‘`faan`’

floating point AAN DCT

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

compresses bright areas stronger than medium ones

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

temporal complexity masking

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

spatial complexity masking

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

inter masking

‘`-dark_mask[:stream_specifier]` *float* (*output,video*)’ compresses dark areas stronger than medium ones

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

select IDCT implementation

Possible values:

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

floating point AAN IDCT

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

set error concealment strategy

Possible values:

‘`guess_mvs`’

iterative motion vector (MV) search (slow)

‘`deblock`’

use strong deblock filter for damaged MBs

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

prediction method

Possible values:

‘`left`’
‘`plane`’
‘`median`’

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

sample aspect ratio

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

print specific debug info

Possible values:

‘`pict`’

picture info

‘`rc`’

rate control

‘`bitstream`’
‘`mb_type`’

macroblock (MB) type

‘`qp`’

per-block quantization parameter (QP)

‘`mv`’

motion vector

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

error recognition

‘`mmco`’

memory management control operations (H.264)

‘`bugs`’
‘`vis_qp`’

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

‘`vis_mb_type`’

visualize block types

‘`buffers`’

picture buffer allocations

‘`thread_ops`’

threading operations

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

visualize motion vectors (MVs)

Possible values:

‘`pf`’

forward predicted MVs of P-frames

‘`bf`’

forward predicted MVs of B-frames

‘`bb`’

backward predicted MVs of B-frames

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

full pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

sub pel me compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

macroblock compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

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

interlaced dct compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

diamond type & size for motion estimation

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

amount of motion predictors from the previous frame

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

pre motion estimation

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

pre motion estimation compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

‘`dctmax`’
‘`chroma`’

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

diamond type & size for motion estimation pre-pass

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

sub pel motion estimation quality

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

limit motion vectors range (1023 for DivX player)

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

intra quant bias

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

inter quant bias

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

Possible values:

‘`vlc`’

variable length coder / huffman coder

‘`ac`’

arithmetic coder

‘`raw`’

raw (no encoding)

‘`rle`’

run-length coder

‘`deflate`’

deflate-based coder

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

context model

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

macroblock decision algorithm (high quality mode)

Possible values:

‘`simple`’

use mbcmp (default)

‘`bits`’

use fewest bits

‘`rd`’

use best rate distortion

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

scene change threshold

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

min lagrange factor (VBR)

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

max lagrange factor (VBR)

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

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

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

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

Possible values:

‘`fast`’

allow non spec compliant speedup tricks

‘`sgop`’

Deprecated, use mpegvideo private options instead

‘`noout`’

skip bitstream encoding

‘`local_header`’

place global headers at every keyframe instead of in extradata

‘`chunks`’

Frame data might be split into multiple chunks

‘`showall`’

Show all frames before the first keyframe

‘`skiprd`’

Deprecated, use mpegvideo private options instead

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

deprecated, use mpegvideo private options instead

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

Possible values:

‘`auto`’

detect a good number of threads

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

motion estimaton threshold

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

macroblock threshold

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

intra_dc_precision

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

nsse weight

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

number of macroblock rows at the top which are skipped

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

number of macroblock rows at the bottom which are skipped

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

Possible values:

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

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

Possible values:

‘`unknown`’

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

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

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

frame skip threshold

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

frame skip factor

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

frame skip exponent

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

frame skip compare function

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘`w53`’

5/3 wavelet, only used in snow

‘`w97`’

9/7 wavelet, only used in snow

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

increases the quantizer for macroblocks close to borders

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

min macroblock lagrange factor (VBR)

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

max macroblock lagrange factor (VBR)

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

motion estimation bitrate penalty compensation (1.0 = 256)

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

Possible values:

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

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

Possible values:

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

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

Possible values:

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

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

refine the two motion vectors used in bidirectional macroblocks

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

downscales frames for dynamic B-frame decision

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

minimum interval between IDR-frames

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

reference frames to consider for motion compensation

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

chroma qp offset from luma

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

rate-distortion optimal quantization

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

multiplied by qscale for each frame and added to scene_change_score

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

adjusts sensitivity of b_frame_strategy 1

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

GOP timecode frame start number, in non drop frame format

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

set desired number of audio channels

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

Possible values:

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

Possible values:

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

set the log level offset

‘`-slices[:stream_specifier]` *integer* (*output,video*)’ number of slices, used in parallelized encoding

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

select multithreading type

Possible values:

‘`slice`’
‘`frame`’

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

audio service type

Possible values:

‘`ma`’

Main Audio Service

‘`ef`’

Effects

‘`vi`’

Visually Impaired

‘`hi`’

Hearing Impaired

‘`di`’

Dialogue

‘`co`’

Commentary

‘`em`’

Emergency

‘`vo`’

Voice Over

‘`ka`’

Karaoke

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

sample format audio decoders should prefer

Possible values:

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

3.5 Format AVOptions

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

Possible values:

‘`direct`’

reduce buffering

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

set probing size

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

set packet size

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

Possible values:

‘`ignidx`’

ignore index

‘`genpts`’

generate pts

‘`nofillin`’

do not fill in missing values that can be exactly calculated

‘`noparse`’

disable AVParsers, this needs nofillin too

‘`igndts`’

ignore dts

‘`discardcorrupt`’

discard corrupted frames

‘`sortdts`’

try to interleave outputted packets by dts

‘`keepside`’

dont merge side data

‘`latm`’

enable RTP MP4A-LATM payload

‘`nobuffer`’

reduce the latency introduced by optional buffering

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

how many microseconds are analyzed to estimate duration

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

decryption key

‘`-indexmem` *`integer`* (*`input`*)’

max memory used for timestamp index (per stream)

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

max memory used for buffering real-time frames

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

print specific debug info

Possible values:

‘`ts`’

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

maximum muxing or demuxing delay in microseconds

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

number of frames used to probe fps

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

microseconds by which audio packets should be interleaved earlier

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

microseconds for each chunk

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

size in bytes for each chunk

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

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

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

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

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

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

set error detection flags

Possible values:

‘`crccheck`’

verify embedded CRCs

‘`bitstream`’

detect bitstream specification deviations

‘`buffer`’

detect improper bitstream length

‘`explode`’

abort decoding on minor error detection

‘`careful`’

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

‘`compliant`’

consider all spec non compliancies as errors

‘`aggressive`’

consider things that a sane encoder shouldnt do as an error

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

use wallclock as timestamps

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

avoid negative timestamps

‘`-skip_initial_bytes` *`integer`* `(`*`input`*`)`’ skip initial bytes

3.6 Main options

‘`-f` *`format`*’

Force format to use.

‘`-unit`’

Show the unit of the displayed values.

‘`-prefix`’

Use SI prefixes for the displayed values. Unless the "-byte_binary_prefix" option is used all the prefixes are decimal.

‘`-byte_binary_prefix`’

Force the use of binary prefixes for byte values.

‘`-sexagesimal`’

Use sexagesimal format HH:MM:SS.MICROSECONDS for time values.

‘`-pretty`’

Prettify the format of the displayed values, it corresponds to the options "-unit -prefix -byte_binary_prefix -sexagesimal".

‘`-of, -print_format` *`writer_name`*`[=`*`writer_options`*`]`’

Set the output printing format.

writer_name specifies the name of the writer, and *writer_options* specifies the options to be passed to the writer.

For example for printing the output in JSON format, specify:

```
-print_format json
```

For more details on the available output printing formats, see the Writers section below.

‘`-sections`’

Print sections structure and section information, and exit. The output is not meant to be parsed by a machine.

‘`-select_streams` *`stream_specifier`*’

Select only the streams specified by *stream_specifier*. This option affects only the options related to streams (e.g. `show_streams`, `show_packets`, etc.).

For example to show only audio streams, you can use the command:

```
ffprobe -show_streams -select_streams a INPUT
```

To show only video packets belonging to the video stream with index 1:

```
ffprobe -show_packets -select_streams v:1 INPUT
```

‘`-show_data`’

Show payload data, as an hexadecimal and ASCII dump. Coupled with ‘`-show_packets`’, it will dump the packets’ data. Coupled with ‘`-show_streams`’, it will dump the codec extradata.

The dump is printed as the "data" field. It may contain newlines.

‘`-show_error`’

Show information about the error found when trying to probe the input.

The error information is printed within a section with name "ERROR".

‘`-show_format`’

Show information about the container format of the input multimedia stream.

All the container format information is printed within a section with name "FORMAT".

‘`-show_format_entry` *`name`*’

Like ‘`-show_format`’, but only prints the specified entry of the container format information, rather than all. This option may be given more than once, then all specified entries will be shown.

This option is deprecated, use `show_entries` instead.

‘`-show_entries` *`section_entries`*’

Set list of entries to show.

Entries are specified according to the following syntax. *section_entries* contains a list of section entries separated by `:`. Each section entry is composed by a section name (or unique name), optionally followed by a list of entries local to that section, separated by `,`.

If section name is specified but is followed by no `=`, all entries are printed to output, together with all the contained sections. Otherwise only the entries specified in the local section entries list are printed. In particular, if `=` is specified but the list of local entries is empty, then no entries will be shown for that section.

Note that the order of specification of the local section entries is not honored in the output, and the usual display order will be retained.

The formal syntax is given by:

```
LOCAL_SECTION_ENTRIES ::= SECTION_ENTRY_NAME[,LOCAL_SECTION_ENTRIES]
SECTION_ENTRY         ::= SECTION_NAME[=[LOCAL_SECTION_ENTRIES]]
SECTION_ENTRIES       ::= SECTION_ENTRY[:SECTION_ENTRIES]
```

For example, to show only the index and type of each stream, and the PTS time, duration time, and stream index of the packets, you can specify the argument:

```
packet=pts_time,duration_time,stream_index : stream=index,codec_type
```

To show all the entries in the section "format", but only the codec type in the section "stream", specify the argument:

```
format : stream=codec_type
```

To show all the tags in the stream and format sections:

```
format_tags : format_tags
```

To show only the `title` tag (if available) in the stream sections:

```
stream_tags=title
```

‘`-show_packets`’

Show information about each packet contained in the input multimedia stream.

The information for each single packet is printed within a dedicated section with name "PACKET".

‘`-show_frames`’

Show information about each frame contained in the input multimedia stream.

The information for each single frame is printed within a dedicated section with name "FRAME".

‘`-show_streams`’

Show information about each media stream contained in the input multimedia stream.

Each media stream information is printed within a dedicated section with name "STREAM".

‘`-count_frames`’

Count the number of frames per stream and report it in the corresponding stream section.

‘`-count_packets`’

Count the number of packets per stream and report it in the corresponding stream section.

‘`-show_private_data, -private`’

Show private data, that is data depending on the format of the particular shown element. This option is enabled by default, but you may need to disable it for specific uses, for example when creating XSD-compliant XML output.

‘`-show_program_version`’

Show information related to program version.

Version information is printed within a section with name "PROGRAM_VERSION".

‘`-show_library_versions`’

Show information related to library versions.

Version information for each library is printed within a section with name "LIBRARY_VERSION".

‘`-show_versions`’

Show information related to program and library versions. This is the equivalent of setting both ‘`-show_program_version`’ and ‘`-show_library_versions`’ options.

‘`-bitexact`’

Force bitexact output, useful to produce output which is not dependent on the specific build.

‘`-i` *`input_file`*’

Read *input_file*.

4. Writers

A writer defines the output format adopted by `ffprobe`, and will be used for printing all the parts of the output.

A writer may accept one or more arguments, which specify the options to adopt. The options are specified as a list of *key*=*value* pairs, separated by ":".

A description of the currently available writers follows.

4.1 default

Default format.

Print each section in the form:

```
[SECTION]
key1=val1
...
keyN=valN
[/SECTION]
```

Metadata tags are printed as a line in the corresponding FORMAT or STREAM section, and are prefixed by the string "TAG:".

A description of the accepted options follows.

‘`nokey, nk`’

If set to 1 specify not to print the key of each field. Default value is 0.

‘`noprint_wrappers, nw`’

If set to 1 specify not to print the section header and footer. Default value is 0.

4.2 compact, csv

Compact and CSV format.

The `csv` writer is equivalent to `compact`, but supports different defaults.

Each section is printed on a single line. If no option is specifid, the output has the form:

```
section|key1=val1| ... |keyN=valN
```

Metadata tags are printed in the corresponding "format" or "stream" section. A metadata tag key, if printed, is prefixed by the string "tag:".

The description of the accepted options follows.

‘`item_sep, s`’

> Specify the character to use for separating fields in the output line. It must be a single printable character, it is "|" by default ("," for the `csv` writer).

‘`nokey, nk`’

> If set to 1 specify not to print the key of each field. Its default value is 0 (1 for the `csv` writer).

‘`escape, e`’

> Set the escape mode to use, default to "c" ("csv" for the `csv` writer).
>
> It can assume one of the following values:
>
> ‘`c`’
>
> > Perform C-like escaping. Strings containing a newline (’\n’), carriage return (’\r’), a tab (’\t’), a form feed (’\f’), the escaping character (’\’) or the item separator character *SEP* are escaped using C-like fashioned escaping, so that a newline is converted to the sequence "\n", a carriage return to "\r", ’\’ to "\\" and the separator *SEP* is converted to "\SEP".
>
> ‘`csv`’
>
> > Perform CSV-like escaping, as described in RFC4180. Strings containing a newline (’\n’), a carriage return (’\r’), a double quote (’"’), or *SEP* are enclosed in double-quotes.
>
> ‘`none`’
>
> > Perform no escaping.

‘`print_section, p`’

> Print the section name at the begin of each line if the value is `1`, disable it with value set to `0`. Default value is `1`.

4.3 flat

Flat format.

A free-form output where each line contains an explicit key=value, such as "streams.stream.3.tags.foo=bar". The output is shell escaped, so it can be directly embedded in sh scripts as long as the separator character is an alphanumeric character or an underscore (see *sep_char* option).

The description of the accepted options follows.

‘`sep_char, s`’

> Separator character used to separate the chapter, the section name, IDs and potential tags in the printed field key.
>
> Default value is ’.’.

‘`hierarchical, h`’

> Specify if the section name specification should be hierarchical. If set to 1, and if there is more than one section in the current chapter, the section name will be prefixed by the name of the chapter. A value of 0 will disable this behavior.
>
> Default value is 1.

4.4 ini

INI format output.

Print output in an INI based format.

The following conventions are adopted:

- all key and values are UTF-8
- ’.’ is the subgroup separator
- newline, ’\t’, ’\f’, ’\b’ and the following characters are escaped
- ’\’ is the escape character
- ’#’ is the comment indicator
- ’=’ is the key/value separator
- ’:’ is not used but usually parsed as key/value separator

This writer accepts options as a list of *key*=*value* pairs, separated by ":".

The description of the accepted options follows.

‘`hierarchical, h`’

> Specify if the section name specification should be hierarchical. If set to 1, and if there is more than one section in the current chapter, the section name will be prefixed by the name of the chapter. A value of 0 will disable this behavior.
>
> Default value is 1.

4.5 json

JSON based format.

Each section is printed using JSON notation.

The description of the accepted options follows.

‘`compact, c`’

> If set to 1 enable compact output, that is each section will be printed on a single line. Default value is 0.

For more information about JSON, see http://www.json.org/.

4.6 xml

XML based format.

The XML output is described in the XML schema description file ‘`ffprobe.xsd`’ installed in the FFmpeg datadir.

An updated version of the schema can be retrieved at the url http://www.ffmpeg.org/schema/ffprobe.xsd, which redirects to the latest schema committed into the FFmpeg development source code tree.

Note that the output issued will be compliant to the ‘`ffprobe.xsd`’ schema only when no special global output options (‘`unit`’, ‘`prefix`’, ‘`byte_binary_prefix`’, ‘`sexagesimal`’ etc.) are specified.

The description of the accepted options follows.

‘`fully_qualified, q`’

> If set to 1 specify if the output should be fully qualified. Default value is 0. This is required for generating an XML file which can be validated through an XSD file.

‘`xsd_compliant, x`’

> If set to 1 perform more checks for ensuring that the output is XSD compliant. Default value is 0. This option automatically sets ‘`fully_qualified`’ to 1.

For more information about the XML format, see http://www.w3.org/XML/.

5. Timecode

`ffprobe` supports Timecode extraction:

- MPEG1/2 timecode is extracted from the GOP, and is available in the video stream details (‘`-show_streams`’, see *timecode*).
- MOV timecode is extracted from tmcd track, so is available in the tmcd stream metadata (‘`-show_streams`’, see *TAG:timecode*).
- DV, GXF and AVI timecodes are available in format metadata (‘`-show_format`’, see *TAG:timecode*).

6. Syntax

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

6.1 Quoting and escaping

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

- `'` and `\` are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.
- A special character is escaped by prefixing it with a ’\’.
- All characters enclosed between ” are included literally in the parsed string. The quote character `'` itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

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

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

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

6.1.1 Examples

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

  ```
  Crime d\'Amour
  ```

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

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

- Include leading or trailing whitespaces using quoting:

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

- Escaping and quoting can be mixed together:

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

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

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

6.2 Date

The accepted syntax is:

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

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

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

6.3 Time duration

The accepted syntax is:

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

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

6.4 Video size

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

The following abbreviations are recognized:

'`sqcif`'

128x96

‘`qcif`’

176x144

‘`cif`’

352x288

‘`4cif`’

704x576

‘`16cif`’

1408x1152

‘`qqvga`’

160x120

‘`qvga`’

320x240

‘`vga`’

640x480

‘`svga`’

800x600

‘`xga`’

1024x768

‘`uxga`’

1600x1200

‘`qxga`’

2048x1536

‘`sxga`’

1280x1024

‘qsxga’

2560x2048

‘hsxga’

5120x4096

‘wvga’

852x480

‘wxga’

1366x768

‘wsxga’

1600x1024

‘wuxga’

1920x1200

‘woxga’

2560x1600

‘wqsxga’

3200x2048

‘wquxga’

3840x2400

‘whsxga’

6400x4096

‘whuxga’

7680x4800

‘cga’

320x200

‘`ega`’

640x350

‘`hd480`’

852x480

‘`hd720`’

1280x720

‘`hd1080`’

1920x1080

6.5 Video rate

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

The following abbreviations are recognized:

‘`ntsc`’

30000/1001

‘`pal`’

25/1

‘`qntsc`’

30000/1

‘`qpal`’

25/1

‘`sntsc`’

30000/1

‘`spal`’

25/1

‘`film`’

24/1

‘`ntsc-film`’

24000/1

6.6 Ratio

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

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

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

6.7 Color

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

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

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

7. Decoders

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

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

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

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

8. Video Decoders

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

8.1 rawvideo

Raw video decoder.

This decoder decodes rawvideo streams.

8.1.1 Options

‘`top` *`top_field_first`*’

Specify the assumed field type of the input video.

‘`-1`’

the video is assumed to be progressive (default)

‘`0`’

bottom-field-first is assumed

‘`1`’

top-field-first is assumed

9. Audio Decoders

9.1 ffwavesynth

Internal wave synthetizer.

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

10. Demuxers

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

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

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

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

The description of some of the currently available demuxers follows.

10.1 image2

Image file demuxer.

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

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

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

This demuxer accepts the following options:

‘`framerate`’

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

‘`loop`’

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

‘`pattern_type`’

> Select the pattern type used to interpret the provided filename.
>
> *pattern_type* accepts one of the following values.
>
> ‘`sequence`’
>
> > Select a sequence pattern type, used to specify a sequence of files indexed by sequential numbers.
> >
> > A sequence pattern may contain the string "%d" or "%0*N*d", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0*N*d" is used, the string representing the number in each filename is 0-padded and *N* is the total number of 0-padded digits representing the number. The literal character ’%’ can be specified in the pattern with the string "%%".

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

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

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

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

'`glob`'

Select a glob wildcard pattern type.

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

'`glob_sequence` *`(deprecated, will be removed)`*'

Select a mixed glob wildcard/sequence pattern.

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

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

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

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

Default value is *glob_sequence*.

'`pixel_format`'

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

‘`start_number`’

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

‘`start_number_range`’

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

‘`video_size`’

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

10.1.1 Examples

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

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

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

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

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

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

10.2 applehttp

Apple HTTP Live Streaming demuxer.

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

10.3 sbg

SBaGen script demuxer.

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

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

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

11. 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.

11.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

BluRay angle

‘`chapter`’

Start chapter (1...N)

‘`playlist`’

Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

11.2 concat

Physical concatenation protocol.

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

A URL accepted by this protocol has the syntax:

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

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

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

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

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

11.3 file

File access protocol.

Allow to read from or read to a file.

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

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

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

11.4 gopher

Gopher protocol.

11.5 hls

Read Apple HTTP Live Streaming compliant segmented stream as a uniform one. The M3U8 playlists describing the segments can be remote HTTP resources or local files, accessed using the standard file protocol. The nested protocol is declared by specifying "+*proto*" after the hls URI scheme name, where *proto* is either "file" or "http".

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

Using this protocol is discouraged - the hls demuxer should work just as well (if not, please report the issues) and is more complete. To use the hls demuxer instead, simply use the direct URLs to the m3u8 files.

11.6 http

HTTP (Hyper Text Transfer Protocol).

11.7 mmst

MMS (Microsoft Media Server) protocol over TCP.

11.8 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

11.9 md5

MD5 output protocol.

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

Some examples follow.

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

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

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

11.10 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

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

For example to read from stdin with `ffmpeg`:

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

For writing to stdout with `ffmpeg`:

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

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

11.11 rtmp

Real-Time Messaging Protocol.

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

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

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

‘`app`’

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. ‘`/ondemand/`’, ‘`/flash/live/`’, etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

‘`playpath`’

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

> Maximum time to wait for the incoming connection. Implies listen.

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

‘`rtmp_app`’

> Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

‘`rtmp_buffer`’

> Set the client buffer time in milliseconds. The default is 3000.

‘`rtmp_conn`’

> Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with ’N’ and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘`rtmp_flashver`’

> Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘`rtmp_flush_interval`’

> Number of packets flushed in the same request (RTMPT only). The default is 10.

‘`rtmp_live`’

> Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is `any`, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are `live` and `recorded`.

‘`rtmp_pageurl`’

> URL of the web page in which the media was embedded. By default no value will be sent.

‘`rtmp_playpath`’

> Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘`rtmp_subscribe`’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘`rtmp_swfhash`’

SHA256 hash of the decompressed SWF file (32 bytes).

‘`rtmp_swfsize`’

Size of the decompressed SWF file, required for SWFVerification.

‘`rtmp_swfurl`’

URL of the SWF player for the media. By default no value will be sent.

‘`rtmp_swfverify`’

URL to player swf file, compute hash/size automatically.

‘`rtmp_tcurl`’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with `ffplay` a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

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

11.12 rtmpe

Encrypted Real-Time Messaging Protocol.

The Encrypted Real-Time Messaging Protocol (RTMPE) is used for streaming multimedia content within standard cryptographic primitives, consisting of Diffie-Hellman key exchange and HMACSHA256, generating a pair of RC4 keys.

11.13 rtmps

Real-Time Messaging Protocol over a secure SSL connection.

The Real-Time Messaging Protocol (RTMPS) is used for streaming multimedia content across an encrypted connection.

11.14 rtmpt

Real-Time Messaging Protocol tunneled through HTTP.

The Real-Time Messaging Protocol tunneled through HTTP (RTMPT) is used for streaming multimedia content within HTTP requests to traverse firewalls.

11.15 rtmpte

Encrypted Real-Time Messaging Protocol tunneled through HTTP.

The Encrypted Real-Time Messaging Protocol tunneled through HTTP (RTMPTE) is used for streaming multimedia content within HTTP requests to traverse firewalls.

11.16 rtmpts

Real-Time Messaging Protocol tunneled through HTTPS.

The Real-Time Messaging Protocol tunneled through HTTPS (RTMPTS) is used for streaming multimedia content within HTTPS requests to traverse firewalls.

11.17 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

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

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

The required syntax is:

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

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

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

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

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

To play the same stream using `ffplay`:

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

11.18 rtp

Real-Time Protocol.

11.19 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock's RTSP server).

The required syntax for a RTSP url is:

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

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

'`udp`'

Use UDP as lower transport protocol.

'`tcp`'

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

'`udp_multicast`'

Use UDP multicast as lower transport protocol.

'`http`'

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

‘`filter_src`’

Accept packets only from negotiated peer address and port.

‘`listen`’

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

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

Example command lines:

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

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

To watch a stream tunneled over HTTP:

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

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

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

To receive a stream in realtime:

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

11.20 sap

Session Announcement Protocol (RFC 2974). This is not technically a protocol handler in libavformat, it is a muxer and demuxer. It is used for signalling of RTP streams, by announcing the SDP for the streams regularly on a separate port.

11.20.1 Muxer

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

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

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

'`announce_addr=`*`address`*'

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

'`announce_port=`*`port`*'

> Specify the port to send the announcements on, defaults to 9875 if not specified.

'`ttl=`*`ttl`*'

> Specify the time to live value for the announcements and RTP packets, defaults to 255.

'`same_port=`*`0|1`*'

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

Example command lines follow.

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

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

Similarly, for watching in `ffplay`:

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

And for watching in `ffplay`, over IPv6:

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

11.20.2 Demuxer

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

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

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

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

Example command lines follow.

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

```
ffplay sap://
```

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

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

11.21 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

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

‘`listen`’

Listen for an incoming connection

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

11.22 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

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

‘`listen`’

Act as a server, listening for an incoming connection.

‘`cafile=`*`filename`*’

Certificate authority file. The file must be in OpenSSL PEM format.

‘`cert=`*`filename`*’

Certificate file. The file must be in OpenSSL PEM format.

‘`key=`*`filename`*’

Private key file.

‘`verify=`*`0|1`*’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

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

11.23 udp

User Datagram Protocol.

The required syntax for a UDP url is:

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

options contains a list of &-separated options of the form *key*=*val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

'`buffer_size=`*`size`*'

Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

'`localport=`*`port`*'

Override the local UDP port to bind with.

'`localaddr=`*`addr`*'

Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

'`pkt_size=`*`size`*'

Set the size in bytes of UDP packets.

'`reuse=`*`1`*`|`*`0`*'

Explicitly allow or disallow reusing UDP sockets.

'`ttl=`*`ttl`*'

Set the time to live value (for multicast only).

'`connect=`*`1`*`|`*`0`*'

Initialize the UDP socket with `connect()`. In this case, the destination address can't be changed with ff_udp_set_remote_url later. If the destination address isn't known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

'`sources=`*`address`*`[,`*`address`*`]`'

Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘`block=`*`address`*`[,`*`address`*`]`’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘`fifo_size=`*`units`*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘`overrun_nonfatal=`*`1|0`*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

‘`timeout=`*`microseconds`*’

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

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

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

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

To receive over UDP from a remote endpoint:

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

12. 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.

12.1 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

12.2 bktr

BSD video input device.

12.3 dshow

Windows DirectShow input device.

DirectShow support is enabled when FFmpeg is built with the mingw-w64 project. Currently only audio and video devices are supported.

Multiple devices may be opened as separate inputs, but they may also be opened on the same input, which should improve synchronism between them.

The input name should be in the format:

```
TYPE=NAME[:TYPE=NAME]
```

where *TYPE* can be either *audio* or *video*, and *NAME* is the device's name.

12.3.1 Options

If no options are specified, the device's defaults are used. If the device does not support the requested options, it will fail to open.

'`video_size`'

Set the video size in the captured video.

'`framerate`'

Set the framerate in the captured video.

'`sample_rate`'

Set the sample rate (in Hz) of the captured audio.

'`sample_size`'

Set the sample size (in bits) of the captured audio.

'`channels`'

Set the number of channels in the captured audio.

'`list_devices`'

If set to '`true`', print a list of devices and exit.

'`list_options`'

If set to '`true`', print a list of selected device's options and exit.

'`video_device_number`'

Set video device number for devices with same name (starts at 0, defaults to 0).

'`audio_device_number`'

Set audio device number for devices with same name (starts at 0, defaults to 0).

‘`pixel_format`’

Select pixel format to be used by DirectShow. This may only be set when the video codec is not set or set to rawvideo.

‘`audio_buffer_size`’

Set audio device buffer size in milliseconds (which can directly impact latency, depending on the device). Defaults to using the audio device’s default buffer size (typically some multiple of 500ms). Setting this value too low can degrade performance. See also http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582(v=vs.85).aspx

12.3.2 Examples

- Print the list of DirectShow supported devices and exit:

```
$ ffmpeg -list_devices true -f dshow -i dummy
```

- Open video device *Camera*:

```
$ ffmpeg -f dshow -i video="Camera"
```

- Open second video device with name *Camera*:

```
$ ffmpeg -f dshow -video_device_number 1 -i video="Camera"
```

- Open video device *Camera* and audio device *Microphone*:

```
$ ffmpeg -f dshow -i video="Camera":audio="Microphone"
```

- Print the list of supported options in selected device and exit:

```
$ ffmpeg -list_options true -f dshow -i video="Camera"
```

12.4 dv1394

Linux DV 1394 input device.

12.5 fbdev

Linux framebuffer input device.

The Linux framebuffer is a graphic hardware-independent abstraction layer to show graphics on a computer monitor, typically on the console. It is accessed through a file device node, usually '`/dev/fb0`'.

For more detailed information read the file Documentation/fb/framebuffer.txt included in the Linux source tree.

To record from the framebuffer device '`/dev/fb0`' with `ffmpeg`:

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

You can take a single screenshot image with the command:

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

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

12.6 iec61883

FireWire DV/HDV input device using libiec61883.

To enable this input device, you need libiec61883, libraw1394 and libavc1394 installed on your system. Use the configure option `--enable-libiec61883` to compile with the device enabled.

The iec61883 capture device supports capturing from a video device connected via IEEE1394 (FireWire), using libiec61883 and the new Linux FireWire stack (juju). This is the default DV/HDV input method in Linux Kernel 2.6.37 and later, since the old FireWire stack was removed.

Specify the FireWire port to be used as input file, or "auto" to choose the first port connected.

12.6.1 Options

'`dvtype`'

Override autodetection of DV/HDV. This should only be used if auto detection does not work, or if usage of a different device type should be prohibited. Treating a DV device as HDV (or vice versa) will not work and result in undefined behavior. The values '`auto`', '`dv`' and '`hdv`' are supported.

'`dvbuffer`'

Set maxiumum size of buffer for incoming data, in frames. For DV, this is an exact value. For HDV, it is not frame exact, since HDV does not have a fixed frame size.

'`dvguid`'

Select the capture device by specifying it’s GUID. Capturing will only be performed from the specified device and fails if no device with the given GUID is found. This is useful to select the input if multiple devices are connected at the same time. Look at /sys/bus/firewire/devices to find out the GUIDs.

12.6.2 Examples

- Grab and show the input of a FireWire DV/HDV device.

  ```
  ffplay -f iec61883 -i auto
  ```

- Grab and record the input of a FireWire DV/HDV device, using a packet buffer of 100000 packets if the source is HDV.

  ```
  ffmpeg -f iec61883 -i auto -hdvbuffer 100000 out.mpg
  ```

12.7 jack

JACK input device.

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

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

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

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

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

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

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

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

# List the current JACK clients.
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

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

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

12.8 lavfi

Libavfilter input virtual device.

This input device reads data from the open output pads of a libavfilter filtergraph.

For each filtergraph open output, the input device will create a corresponding stream which is mapped to the generated output. Currently only video data is supported. The filtergraph is specified through the option ‘`graph`’.

12.8.1 Options

‘`graph`’

> Specify the filtergraph to use as input. Each video open output must be labelled by a unique string of the form "out*N*", where *N* is a number starting from 0 corresponding to the mapped input stream generated by the device. The first unlabelled output is automatically assigned to the "out0" label, but all the others need to be specified explicitly.
>
> If not specified defaults to the filename specified for the input device.

‘`graph_file`’

> Set the filename of the filtergraph to be read and sent to the other filters. Syntax of the filtergraph is the same as the one specified by the option *graph*.

12.8.2 Examples

- Create a color video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi -graph "color=pink [out0]" dummy
  ```

- As the previous example, but use filename for specifying the graph description, and omit the "out0" label:

  ```
  ffplay -f lavfi color=pink
  ```

- Create three different video test filtered sources and play them:

  ```
  ffplay -f lavfi -graph "testsrc [out0]; testsrc,hflip [out1]; testsrc,negate [out2]" test3
  ```

- Read an audio stream from a file using the amovie source and play it back with `ffplay`:

  ```
  ffplay -f lavfi "amovie=test.wav"
  ```

- Read an audio stream and a video stream and play it back with `ffplay`:

  ```
  ffplay -f lavfi "movie=test.avi[out0];amovie=test.wav[out1]"
  ```

12.9 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

12.10 openal

The OpenAL input device provides audio capture on all systems with a working OpenAL 1.1 implementation.

To enable this input device during configuration, you need OpenAL headers and libraries installed on your system, and need to configure FFmpeg with `--enable-openal`.

OpenAL headers and libraries should be provided as part of your OpenAL implementation, or as an additional download (an SDK). Depending on your installation you may need to specify additional flags via the `--extra-cflags` and `--extra-ldflags` for allowing the build system to locate the OpenAL headers and libraries.

An incomplete list of OpenAL implementations follows:

Creative

The official Windows implementation, providing hardware acceleration with supported devices and software fallback. See http://openal.org/.

OpenAL Soft

Portable, open source (LGPL) software implementation. Includes backends for the most common sound APIs on the Windows, Linux, Solaris, and BSD operating systems. See http://kcat.strangesoft.net/openal.html.

Apple

OpenAL is part of Core Audio, the official Mac OS X Audio interface. See http://developer.apple.com/technologies/mac/audio-and-video.html

This device allows to capture from an audio input device handled through OpenAL.

You need to specify the name of the device to capture in the provided filename. If the empty string is provided, the device will automatically select the default device. You can get the list of the supported devices by using the option *list_devices*.

12.10.1 Options

'`channels`'

Set the number of channels in the captured audio. Only the values '`1`' (monaural) and '`2`' (stereo) are currently supported. Defaults to '`2`'.

'`sample_size`'

Set the sample size (in bits) of the captured audio. Only the values '`8`' and '`16`' are currently supported. Defaults to '`16`'.

'`sample_rate`'

Set the sample rate (in Hz) of the captured audio. Defaults to '`44.1k`'.

'`list_devices`'

If set to '`true`', print a list of devices and exit. Defaults to '`false`'.

12.10.2 Examples

Print the list of OpenAL supported devices and exit:

```
$ ffmpeg -list_devices true -f openal -i dummy out.ogg
```

Capture from the OpenAL device ‘`DR-BT101 via PulseAudio`’:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out.ogg
```

Capture from the default device (note the empty string ” as filename):

```
$ ffmpeg -f openal -i ’’ out.ogg
```

Capture from two devices simultaneously, writing to two different files, within the same `ffmpeg` command:

```
$ ffmpeg -f openal -i ’DR-BT101 via PulseAudio’ out1.ogg -f openal -i ’ALSA Default’ out2.ogg
```

Note: not all OpenAL implementations support multiple simultaneous capture - try the latest OpenAL Soft if the above does not work.

12.11 oss

Open Sound System input device.

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

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

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

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

12.12 pulse

pulseaudio input device.

To enable this input device during configuration you need libpulse-simple installed in your system.

The filename to provide to the input device is a source device or the string "default"

To list the pulse source devices and their properties you can invoke the command `pactl list sources`.

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

12.12.1 *server* AVOption

The syntax is:

```
-server server name
```

Connects to a specific server.

12.12.2 *name* AVOption

The syntax is:

```
-name application name
```

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

12.12.3 *stream_name* AVOption

The syntax is:

```
-stream_name stream name
```

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

12.12.4 *sample_rate* AVOption

The syntax is:

```
-sample_rate samplerate
```

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

12.12.5 *channels* AVOption

The syntax is:

```
-channels N
```

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

12.12.6 *frame_size* AVOption

The syntax is:

```
-frame_size bytes
```

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

12.12.7 *fragment_size* AVOption

The syntax is:

```
-fragment_size bytes
```

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

12.13 sndio

sndio input device.

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

The filename to provide to the input device is the device node representing the sndio input device, and is usually set to '`/dev/audio0`'.

For example to grab from '`/dev/audio0`' using `ffmpeg` use the command:

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

12.14 video4linux2

Video4Linux2 input video device.

The name of the device to grab is a file device node, usually Linux systems tend to automatically create such nodes when the device (e.g. an USB webcam) is plugged into the system, and has a name of the kind '`/dev/video`*N*', where *N* is a number associated to the device.

Video4Linux2 devices usually support a limited set of *width*x*height* sizes and framerates. You can check which are supported using `-list_formats all` for Video4Linux2 devices.

Some usage examples of the video4linux2 devices with ffmpeg and ffplay:

The time base for the timestamps is 1 microsecond. Depending on the kernel version and configuration, the timestamps may be derived from the real time clock (origin at the Unix Epoch) or the monotonic clock (origin usually at boot time, unaffected by NTP or manual changes to the clock). The '`-timestamps abs`' or '`-ts abs`' option can be used to force conversion into the real time clock.

Note that if FFmpeg is build with v4l-utils support ("–enable-libv4l2" option), it will always be used.

```
# Grab and show the input of a video4linux2 device.
ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0

# Grab and record the input of a video4linux2 device, leave the
framerate and size as previously set.
ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
```

"v4l" and "v4l2" can be used as aliases for the respective "video4linux" and "video4linux2".

12.15 vfwcap

VfW (Video for Windows) capture input device.

The filename passed as input is the capture driver number, ranging from 0 to 9. You may use "list" as filename to print a list of drivers. Any other filename will be interpreted as device number 0.

12.16 x11grab

X11 video input device.

This device allows to capture a region of an X11 display.

The filename passed as input has the syntax:

```
[hostname]:display_number.screen_number[+x_offset,y_offset]
```

hostname:*display_number*.*screen_number* specifies the X11 display name of the screen to grab from. *hostname* can be omitted, and defaults to "localhost". The environment variable `DISPLAY` contains the default display name.

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

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

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

For example to grab from ‘`:0.0`’ using `ffmpeg`:

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

Grab at position `10,20`:

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

12.16.1 Options

‘`draw_mouse`’

Specify whether to draw the mouse pointer. A value of `0` specify not to draw the pointer. Default value is `1`.

‘`follow_mouse`’

Make the grabbed area follow the mouse. The argument can be `centered` or a number of pixels *PIXELS*.

When it is specified with "centered", the grabbing region follows the mouse pointer and keeps the pointer at the center of region; otherwise, the region follows only when the mouse pointer reaches within *PIXELS* (greater than zero) to the edge of region.

For example:

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

To follow only when the mouse pointer reaches within 100 pixels to edge:

```
ffmpeg -f x11grab -follow_mouse 100 -r 25 -s cif -i :0.0 out.mpg
```

‘`framerate`’

Set the grabbing frame rate. Default value is `ntsc`, corresponding to a framerate of `30000/1001`.

‘`show_region`’

Show grabbed region on screen.

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

For example:

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

With *follow_mouse*:

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

‘`video_size`’

Set the video frame size. Default value is `vga`.

This document was generated by *john* on *November 26, 2012* using *texi2html 1.82*.