avprobe Documentation

Table of Contents

1. Synopsis

The generic syntax is:

```
avprobe [options] ['input_file']
```

2. Description

avprobe 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, avprobe 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.

avprobe 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 avprobe or for specifying which information to display, and for setting how avprobe will show it.

avprobe output is designed to be easily parsable by any INI or JSON parsers.

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 specifer, which matches the second audio stream. Therefore it would select the ac3 codec for the second audio stream.

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

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

Possible forms of stream specifiers are:

'`stream_index`'

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

'`stream_type[:stream_index]`'

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

'`p:program_id[:stream_index]`'

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

'`i:stream_id`'

> Match the stream by stream id (e.g. PID in MPEG-TS container).

'`m:key[:value]`'

> Matches streams with the metadata tag *key* having the specified value. If *value* is not given, matches streams that contain the given tag with any value.

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

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.

'`filter=filter_name`'

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

'`-version`'

Show version.

'`-formats`'

Show available formats.

The fields preceding the format names have the following meanings:

'`D`'

Decoding available

'`E`'

Encoding available

'`-codecs`'

Show all codecs known to libavcodec.

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

'`-decoders`'

    Show available decoders.

'`-encoders`'

    Show all available encoders.

'`-bsfs`'

    Show available bitstream filters.

'`-protocols`'

    Show available protocols.

'`-filters`'

    Show available libavfilter filters.

'`-pix_fmts`'

    Show available pixel formats.

'`-sample_fmts`'

    Show available sample formats.

'`-loglevel` *loglevel* `| -v` *loglevel*'

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

    '`quiet`'
    '`panic`'
    '`fatal`'
    '`error`'
    '`warning`'
    '`info`'
    '`verbose`'
    '`debug`'

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

'-cpuflags mask (*global*)'

> Set a mask that's applied to autodetected CPU flags. This option is intended for testing. Do not use it unless you know what you're doing.

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:

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

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

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

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

3.4 Codec AVOptions

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

> set bitrate (in bits/s)

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

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

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

Possible values:

'unaligned'

    allow decoders to produce unaligned output

'mv4'

    use four motion vectors per macroblock (MPEG-4)

'qpel'

    use 1/4-pel motion compensation

'loop'

    use loop filter

'qscale'

    use fixed qscale

'gmc'

    use gmc

'mv0'

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

'input_preserved'
'pass1'

    use internal 2-pass ratecontrol in first pass mode

'pass2'

    use internal 2-pass ratecontrol in second pass mode

'gray'

    only decode/encode grayscale

'emu_edge'

    do not draw edges

'psnr'

error[?] variables will be set during encoding

'truncated'

'naq'

> normalize adaptive quantization

'ildct'

> use interlaced DCT

'low_delay'

> force low delay

'global_header'

> place global headers in extradata instead of every keyframe

'bitexact'

> use only bitexact functions (except (I)DCT)

'aic'

> H.263 advanced intra coding / MPEG-4 AC prediction

'ilme'

> interlaced motion estimation

'cgop'

> closed GOP

'output_corrupt'

> Output even potentially corrupted frames

'-me_method[:stream_specifier] integer (output,video)'

> set motion estimation method

> Possible values:

'zero'

> zero motion estimation (fastest)

'full'

> full motion estimation (slowest)

'epzs'

> EPZS motion estimation (default)

'esa'

> esa motion estimation (alias for full)

'tesa'

> tesa motion estimation

'dia'

> diamond motion estimation (alias for EPZS)

'log'

> log motion estimation

'phods'

> phods motion estimation

'x1'

> X1 motion estimation

'hex'

> hex motion estimation

'umh'

> umh motion estimation

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

> set the group of picture (GOP) size

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

> set audio sampling rate (in Hz)

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

set number of audio channels

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

set cutoff bandwidth

`'-frame_size[:stream_specifier] integer (output,audio)'`
`'-qcomp[:stream_specifier] float (output,video)'`

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

`'-qblur[:stream_specifier] float (output,video)'`

video quantizer scale blur (VBR)

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

minimum video quantizer scale (VBR)

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

maximum video quantizer scale (VBR)

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

maximum difference between the quantizer scales (VBR)

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

use 'frames' B frames

`'-b_qfactor[:stream_specifier] float (output,video)'`

QP factor between P- and B-frames

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

ratecontrol method

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

strategy to choose between I/P/B-frames

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

RTP payload size in bytes

'-bug[:stream_specifier] *flags* (*input,video*)'

> work around not autodetected encoder bugs

> Possible values:

> 'autodetect'
> 'old_msmpeg4'

>> some old lavc-generated MSMPEG4v3 files (no autodetection)

> 'xvid_ilace'

>> Xvid interlacing bug (autodetected if FOURCC == XVIX)

> 'ump4'

>> (autodetected if FOURCC == UMP4)

> 'no_padding'

>> padding bug (autodetected)

> 'amv'
> 'ac_vlc'

>> illegal VLC bug (autodetected per FOURCC)

> 'qpel_chroma'
> 'std_qpel'

>> old standard qpel (autodetected per FOURCC/version)

> 'qpel_chroma2'
> 'direct_blocksize'

>> direct-qpel-blocksize bug (autodetected per FOURCC/version)

> 'edge'

>> edge padding bug (autodetected per FOURCC/version)

> 'hpel_chroma'
> 'dc_clip'
> 'ms'

>> work around various bugs in Microsoft's broken decoders

'trunc'

> truncated frames

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

> how strictly to follow the standards
>
> Possible values:
>
> 'very'
>
> > strictly conform to a older more strict version of the spec or reference software
>
> 'strict'
>
> > strictly conform to all the things in the spec no matter what the consequences
>
> 'normal'
> 'unofficial'
>
> > allow unofficial extensions
>
> 'experimental'
>
> > allow non-standardized experimental things

'-b_qoffset[:stream_specifier] *float (output,video)*'

> QP offset between P- and B-frames

'-err_detect[:stream_specifier] *flags (input,audio,video)*'

> set error detection flags
>
> Possible values:
>
> 'crccheck'
>
> > verify embedded CRCs
>
> 'bitstream'
>
> > detect bitstream specification deviations
>
> 'buffer'
>
> > detect improper bitstream length

‘explode’

       abort decoding on minor error detection

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

    use MPEG quantizers instead of H.263

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

    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_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 maximum bitrate tolerance (in bits/s). Requires bufsize to be set.

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

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

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

    set ratecontrol buffer size (in bits)

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

    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`'
    '`arm`'
    '`altivec`'
    '`sh4`'
    '`simplearm`'
    '`simplearmv5te`'
    '`simplearmv6`'
    '`simpleneon`'
    '`simplealpha`'
    '`ipp`'
    '`xvid`'
    '`xvidmmx`'
    '`faani`'

        floating point AAN IDCT

'`-ec[:stream_specifier]` *flags (input,video)*'

    set error concealment strategy

    Possible values:

    '`guess_mvs`'

        iterative motion vector (MV) search (slow)

    '`deblock`'

        use strong deblock filter for damaged MBs

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

    prediction method

Possible values:

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

sample aspect ratio

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

print specific debug info

Possible values:

'pict'

>picture info

'rc'

>rate control

'bitstream'
'mb_type'

>macroblock (MB) type

'qp'

>per-block quantization parameter (QP)

'mv'

>motion vector

'dct_coeff'
'skip'
'startcode'
'pts'
'er'

>error recognition

'mmco'

>memory management control operations (H.264)

'bugs'

'vis_qp'

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

'vis_mb_type'

>   visualize block types

'buffers'

>   picture buffer allocations

'thread_ops'

>   threading operations

'-vismv[:stream_specifier] integer (input,video)'

>   visualize motion vectors (MVs)

>   Possible values:

>   'pf'

>>   forward predicted MVs of P-frames

>   'bf'

>>   forward predicted MVs of B-frames

>   'bb'

>>   backward predicted MVs of B-frames

'-cmp[:stream_specifier] integer (output,video)'

>   full-pel ME compare function

>   Possible values:

>   'sad'

>>   sum of absolute differences, fast (default)

>   'sse'

>>   sum of squared errors

'satd'

>   sum of absolute Hadamard transformed differences

'dct'

>   sum of absolute DCT transformed differences

'psnr'

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

'bit'

>   number of bits needed for the block

'rd'

>   rate distortion optimal, slow

'zero'

>   0

'vsad'

>   sum of absolute vertical differences

'vsse'

>   sum of squared vertical differences

'nsse'

>   noise preserving sum of squared differences

'dctmax'
'chroma'
'-subcmp[:stream_specifier] integer (output,video)'

>   sub-pel ME compare function

>   Possible values:

'sad'

>   sum of absolute differences, fast (default)

'sse'

>   sum of squared errors

'satd'

>   sum of absolute Hadamard transformed differences

'dct'

>   sum of absolute DCT transformed differences

'psnr'

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

'bit'

>   number of bits needed for the block

'rd'

>   rate distortion optimal, slow

'zero'

>   0

'vsad'

>   sum of absolute vertical differences

'vsse'

>   sum of squared vertical differences

'nsse'

>   noise preserving sum of squared differences

'dctmax'
'chroma'
'-mbcmp[:stream_specifier] *integer* (*output,video*)'

>   macroblock compare function

>   Possible values:

‘sad’

　　sum of absolute differences, fast (default)

‘sse’

　　sum of squared errors

‘satd’

　　sum of absolute Hadamard transformed differences

‘dct’

　　sum of absolute DCT transformed differences

‘psnr’

　　sum of squared quantization errors (avoid, low quality)

‘bit’

　　number of bits needed for the block

‘rd’

　　rate distortion optimal, slow

‘zero’

　　0

‘vsad’

　　sum of absolute vertical differences

‘vsse’

　　sum of squared vertical differences

‘nsse’

　　noise preserving sum of squared differences

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

interlaced DCT compare function

Possible values:

'sad'

    sum of absolute differences, fast (default)

'sse'

    sum of squared errors

'satd'

    sum of absolute Hadamard transformed differences

'dct'

    sum of absolute DCT transformed differences

'psnr'

    sum of squared quantization errors (avoid, low quality)

'bit'

    number of bits needed for the block

'rd'

    rate distortion optimal, slow

'zero'

    0

'vsad'

    sum of absolute vertical differences

'vsse'

    sum of squared vertical differences

'nsse'

    noise preserving sum of squared differences

'dctmax'

'chroma'

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

    diamond type & size for motion estimation

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

    amount of motion predictors from the previous frame

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

    pre motion estimation

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

    pre motion estimation compare function

    Possible values:

    'sad'

        sum of absolute differences, fast (default)

    'sse'

        sum of squared errors

    'satd'

        sum of absolute Hadamard transformed differences

    'dct'

        sum of absolute DCT transformed differences

    'psnr'

        sum of squared quantization errors (avoid, low quality)

    'bit'

        number of bits needed for the block

    'rd'

        rate distortion optimal, slow

    'zero'

> 0

'vsad'

> sum of absolute vertical differences

'vsse'

> sum of squared vertical differences

'nsse'

> noise preserving sum of squared differences

'dctmax'
'chroma'

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

> diamond type & size for motion estimation pre-pass

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

> sub-pel motion estimation quality

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

> limit motion vectors range (1023 for DivX player)

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

> intra quant bias

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

> inter quant bias

'-global_quality[:stream_specifier] *integer* (*output,audio,video*)'
'-coder[:stream_specifier] *integer* (*output,video*)'

> Possible values:

'vlc'

> variable length coder / Huffman coder

'ac'

> arithmetic coder

'`raw`'

> raw (no encoding)

'`rle`'

> run-length coder

'`deflate`'

> deflate-based coder

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

> context model

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

> macroblock decision algorithm (high quality mode)

> Possible values:

> '`simple`'

>> use mbcmp (default)

> '`bits`'

>> use fewest bits

> '`rd`'

>> use best rate distortion

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

> scene change threshold

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

> minimum Lagrange factor (VBR)

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

> maximum 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

'-flags2[:stream_specifier] *flags (input/output,audio,video)*'

    Possible values:

    'fast'

        allow non-spec-compliant speedup tricks

    'noout'

        skip bitstream encoding

    'ignorecrop'

        ignore cropping information from sps

    'local_header'

        place global headers at every keyframe instead of in extradata

'-error[:stream_specifier] *integer (output,video)*'
'-threads[:stream_specifier] *integer (input/output,video)*'

    Possible values:

    'auto'

        autodetect a suitable number of threads to use

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

    motion estimation threshold

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

    macroblock threshold

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

    intra_dc_precision

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

nsse weight

'-skip_top[:stream_specifier] *integer (input,video)*'

number of macroblock rows at the top which are skipped

'-skip_bottom[:stream_specifier] *integer (input,video)*'

number of macroblock rows at the bottom which are skipped

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

Possible values:

'unknown'
'aac_main'
'aac_low'
'aac_ssr'
'aac_ltp'
'aac_he'
'aac_he_v2'
'aac_ld'
'aac_eld'
'mpeg2_aac_low'
'mpeg2_aac_he'
'dts'
'dts_es'
'dts_96_24'
'dts_hd_hra'
'dts_hd_ma'
'-level[:stream_specifier] *integer (output,audio,video)*'

Possible values:

'unknown'
'-skip_threshold[:stream_specifier] *integer (output,video)*'

frame skip threshold

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

frame skip factor

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

frame skip exponent

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

> frame skip compare function
>
> Possible values:
>
> '`sad`'
>
> > sum of absolute differences, fast (default)
>
> '`sse`'
>
> > sum of squared errors
>
> '`satd`'
>
> > sum of absolute Hadamard transformed differences
>
> '`dct`'
>
> > sum of absolute DCT transformed differences
>
> '`psnr`'
>
> > sum of squared quantization errors (avoid, low quality)
>
> '`bit`'
>
> > number of bits needed for the block
>
> '`rd`'
>
> > rate distortion optimal, slow
>
> '`zero`'
>
> > 0
>
> '`vsad`'
>
> > sum of absolute vertical differences
>
> '`vsse`'
>
> > sum of squared vertical differences
>
> '`nsse`'

noise preserving sum of squared differences

'dctmax'
'chroma'
'-border_mask[:stream_specifier] *float* (*output,video*)'

increase the quantizer for macroblocks close to borders

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

minimum macroblock Lagrange factor (VBR)

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

maximum macroblock Lagrange factor (VBR)

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

motion estimation bitrate penalty compensation (1.0 = 256)

'-skip_loop_filter[:stream_specifier] *integer* (*input,video*)'

Possible values:

'none'
'default'
'noref'
'bidir'
'nokey'
'all'
'-skip_idct[:stream_specifier] *integer* (*input,video*)'

Possible values:

'none'
'default'
'noref'
'bidir'
'nokey'
'all'
'-skip_frame[:stream_specifier] *integer* (*input,video*)'

Possible values:

'none'
'default'
'noref'

      'bidir'
      'nokey'
      'all'

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

    refine the two motion vectors used in bidirectional macroblocks

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

    downscale frames for dynamic B-frame decision

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

    minimum interval between IDR-frames (x264)

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

    reference frames to consider for motion compensation

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

    chroma QP offset from luma

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

    rate-distortion optimal quantization

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

    multiplied by qscale for each frame and added to scene_change_score

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

    adjust sensitivity of b_frame_strategy 1

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

    GOP timecode frame start number, in non-drop-frame format

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

    set desired number of audio channels

'-channel_layout[:stream_specifier] *integer (input/output,audio)*'

Possible values:

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

Possible values:

'-rc_max_vbv_use[:stream_specifier] *float (output,video)*'
'-rc_min_vbv_use[:stream_specifier] *float (output,video)*'
'-ticks_per_frame[:stream_specifier] *integer (input/output,audio,video)*'
'-color_primaries[:stream_specifier] *integer (input/output,video)*'

color primaries

Possible values:

'bt709'

BT.709

'unspecified'

Unspecified

'bt470m'

BT.470 M

'bt470bg'

BT.470 BG

'smpte170m'

SMPTE 170 M

'smpte240m'

SMPTE 240 M

'film'

Film

'bt2020'

BT.2020

'-color_trc[:stream_specifier] *integer (input/output,video)*'

color transfert characteristic

Possible values:

'bt709'

> BT.709

'unspecfied'

> Unspecified

'gamma22'

> Gamma 2.2

'gamma28'

> Gamma 2.8

'smpte170m'

> SMPTE 170 M

'smpte240m'

> SMPTE 240 M

'linear'

> Linear

'log'

> SMPTE 240 M

'log_sqrt'

> SMPTE 240 M

'iec61966_2_4'

> SMPTE 240 M

'bt1361'

> BT.1361

'iec61966_2_1'

SMPTE 240 M

'bt2020_10bit'

BT.2020 - 10 bit

'bt2020_12bit'

BT.2020 - 12 bit

'-colorspace[:stream_specifier] *integer* (*input/output,video*)'

colorspace

Possible values:

'rgb'

RGB

'bt709'

BT.709

'unspecified'

Unspecified

'fcc'

FourCC

'bt470bg'

BT.470 BG

'smpte170m'

SMPTE 170 M

'smpte240m'

SMPTE 240 M

'ycocg'

YCOCG

'bt2020_ncl'

BT.2020 NCL

'bt2020_cl'

    BT.2020 CL

'-color_range[:stream_specifier] *integer* (*input/output,video*)'

color range

Possible values:

'unspecified'

    Unspecified

'mpeg'

    MPEG (219*2^(n-8))

'jpeg'

    JPEG (2^n-1)

'-chroma_sample_location[:stream_specifier] *integer* (*input/output,video*)'
'-slices[:stream_specifier] *integer* (*output,video*)'

number of slices, used in parallelized encoding

'-thread_type[:stream_specifier] *flags* (*input/output,video*)'

select multithreading type

Possible values:

'slice'
'frame'
'-audio_service_type[:stream_specifier] *integer* (*output,audio*)'

audio service type

Possible values:

'ma'

    Main Audio Service

'ef'

    Effects

'vi'

    Visually Impaired

'hi'

    Hearing Impaired

'di'

    Dialogue

'co'

    Commentary

'em'

    Emergency

'vo'

    Voice Over

'ka'

    Karaoke

'-request_sample_fmt[:stream_specifier] integer (input,audio)'

    Possible values:

    'u8'

        8-bit unsigned integer

    's16'

        16-bit signed integer

    's32'

        32-bit signed integer

    'flt'

32-bit float

'dbl'

64-bit double

'u8p'

8-bit unsigned integer planar

's16p'

16-bit signed integer planar

's32p'

32-bit signed integer planar

'fltp'

32-bit float planar

'dblp'

64-bit double planar

'-refcounted_frames[:stream_specifier] integer (input,audio,video)'
'-side_data_only_packets[:stream_specifier] integer
(output,audio,video)'

3.5 Format AVOptions

'-probesize integer (input)'

set probing size

'-packetsize integer (output)'

set packet size

'-fflags flags (input/output)'

Possible values:

'flush_packets'

reduce the latency by flushing out packets immediately

'ignidx'

ignore index

'genpts'

generate pts

'nofillin'

do not fill in missing values that can be exactly calculated

'noparse'

disable AVParsers, this needs nofillin too

'igndts'

ignore dts

'discardcorrupt'

discard corrupted frames

'nobuffer'

reduce the latency introduced by optional buffering

'bitexact'

do not write random/volatile data

'-analyzeduration *integer* (*input*)'

how many microseconds are analyzed to estimate duration

'-cryptokey *hexadecimal string* (*input*)'

decryption key

'-indexmem *integer* (*input*)'

max memory used for timestamp index (per stream)

'-rtbufsize *integer* (*input*)'

max memory used for buffering real-time frames

'-fdebug *flags* (*input/output*)'

print specific debug info

Possible values:

'ts'

'-max_delay *integer* (*input/output*)'

maximum muxing or demuxing delay in microseconds

'-fpsprobesize *integer* (*input*)'

number of frames used to probe fps

'-f_err_detect *flags* (*input*)'

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

Possible values:

'crccheck'

verify embedded CRCs

'bitstream'

detect bitstream specification deviations

'buffer'

detect improper bitstream length

'explode'

abort decoding on minor error detection

'-err_detect *flags* (*input*)'

set error detection flags

Possible values:

'crccheck'

verify embedded CRCs

'bitstream'

detect bitstream specification deviations

‘buffer’

  detect improper bitstream length

‘explode’

  abort decoding on minor error detection

‘-max_interleave_delta *integer* (*output*)’

  maximum buffering duration for interleaving

‘-f_strict *integer* (*input/output*)’

  how strictly to follow the standards (deprecated; use strict, save via avconv)

  Possible values:

  ‘strict’

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

  ‘normal’
  ‘experimental’

    allow non-standardized experimental variants

‘-strict *integer* (*input/output*)’

  how strictly to follow the standards

  Possible values:

  ‘strict’

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

  ‘normal’
  ‘experimental’

    allow non-standardized experimental variants

3.6 Main options

‘-f *format*’

  Force format to use.

'`-of` *formatter*'

> Use a specific formatter to output the document. The following formatters are available

> '`ini`'
> '`json`'
> '`old`'

>> Pseudo-INI format that used to be the only one available in old avprobe versions.

'`-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".

'`-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.

'`-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_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".

4. Demuxers

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

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

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

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

The description of some of the currently available demuxers follows.

4.1 image2

Image file demuxer.

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

The pattern may contain the string "%d" or "%0Nd", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0Nd" is used, the string representing the number in each filename is 0-padded and N is the total number of 0-padded digits representing the number. The literal character '%' can be specified in the pattern with the string "%%".

If the pattern contains "%d" or "%0Nd", the first filename of the file list specified by the pattern must contain a number inclusively contained between 0 and 4, all the following numbers must be sequential. This limitation may be hopefully fixed.

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

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

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

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

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

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

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

'`-pixel_format` *format*'

    Set the pixel format (for raw image)

'`-video_size` *size*'

    Set the frame size (for raw image)

'`-framerate` *rate*'

    Set the frame rate

'`-loop` *bool*'

    Loop over the images

'`-start_number` *start*'

    Specify the first number in the sequence

4.2 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 avplay), the caller can decide which variant streams to actually receive. The total bitrate of the variant that the stream belongs to is available in a metadata key named "variant_bitrate".

4.3 flv

Adobe Flash Video Format demuxer.

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

'`-flv_metadata` *bool*'

Allocate the streams according to the onMetaData array content.

4.4 asf

Advanced Systems Format demuxer.

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

'`-no_resync_search` *bool*'

Do not try to resynchronize by looking for a certain optional start code.

5. Muxers

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

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

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

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

A description of some of the currently available muxers follows.

5.1 crc

CRC (Cyclic Redundancy Check) testing format.

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

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

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

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

You can print the CRC to stdout with the command:

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

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

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

See also the framecrc muxer.

5.2 framecrc

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

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

The output of the muxer consists of a line for each audio and video frame of the form: *stream_index*, *frame_dts*, *frame_size*, 0x*CRC*, where *CRC* is a hexadecimal number 0-padded to 8 digits containing the CRC of the decoded frame.

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

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

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

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

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

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

See also the crc muxer.

5.3 hls

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

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

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

'-hls_time *seconds*'

Set the segment length in seconds.

'-hls_list_size *size*'

Set the maximum number of playlist entries.

'-hls_wrap *wrap*'

Set the number after which index wraps.

'-start_number *number*'

Start the sequence from *number*.

'-hls_base_url *baseurl*'

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

5.4 image2

Image file muxer.

The image file muxer writes video frames to image files.

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

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

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

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

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

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

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

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

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

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

'-start_number *number*'

> Start the sequence from *number*.

'-update *number*'

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

5.5 matroska

Matroska container muxer.

This muxer implements the matroska and webm container specs.

The recognized metadata settings in this muxer are:

'title=*title name*'

> Name provided to a single track

'language=*language name*'

    Specifies the language of the track in the Matroska languages form

'STEREO_MODE=*mode*'

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

    'mono'

        video is not stereo

    'left_right'

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

    'bottom_top'

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

    'top_bottom'

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

    'checkerboard_rl'

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

    'checkerboard_lr'

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

    'row_interleaved_rl'

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

    'row_interleaved_lr'

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

    'col_interleaved_rl'

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

    'col_interleaved_lr'

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

    'anaglyph_cyan_red'

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

'`right_left`'

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

'`anaglyph_green_magenta`'

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

'`block_lr`'

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

'`block_rl`'

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

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

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

This muxer supports the following options:

'`reserve_index_space`'

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

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

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

5.6 mov, mp4, ismv

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

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

'-movflags frag_keyframe'

Start a new fragment at each video keyframe.

'-frag_duration *duration*'

Create fragments that are *duration* microseconds long.

'-frag_size *size*'

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

'-movflags frag_custom'

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

'-min_frag_duration *duration*'

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

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

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

'-movflags empty_moov'

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

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

'-movflags separate_moof'

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

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

'`-movflags faststart`'

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

'`-movflags disable_chpl`'

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

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

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

5.7 mp3

The MP3 muxer writes a raw MP3 stream with an ID3v2 header at the beginning and optionally an ID3v1 tag at the end. ID3v2.3 and ID3v2.4 are supported, the `id3v2_version` option controls which one is used. Setting `id3v2_version` to 0 will disable the ID3v2 header completely. The legacy ID3v1 tag is not written by default, but may be enabled with the `write_id3v1` option.

The muxer may also write a Xing frame at the beginning, which contains the number of frames in the file. It is useful for computing duration of VBR files. The Xing frame is written if the output stream is seekable and if the `write_xing` option is set to 1 (the default).

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

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

Examples:

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

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

Attach a picture to an mp3:

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

Write a "clean" MP3 without any extra features:

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

5.8 mpegts

MPEG transport stream muxer.

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

The muxer options are:

'-mpegts_original_network_id *number*'

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

'-mpegts_transport_stream_id *number*'

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

'-mpegts_service_id *number*'

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

'-mpegts_pmt_start_pid *number*'

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

'-mpegts_start_pid *number*'

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

'-muxrate *number*'

Set a constant muxrate (default VBR).

'-pcr_period *numer*'

Override the default PCR retransmission time (default 20ms), ignored if variable muxrate is selected.

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

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

5.9 null

Null muxer.

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

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

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

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

Alternatively you can write the command as:

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

5.10 nut

'`-syncpoints flags`'

Change the syncpoint usage in nut:

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

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

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

5.11 ogg

Ogg container muxer.

'`-page_duration duration`'

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

5.12 segment

Basic stream segmenter.

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

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

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

'`segment_format format`'

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

'`segment_time t`'

> Set segment duration to *t* seconds.

'`segment_list name`'

> Generate also a listfile named *name*.

'`segment_list_type type`'

> Select the listing format.
>
> '`flat use a simple flat list of entries.`'
> '`hls use a m3u8-like structure.`'

'`segment_list_size size`'

> Overwrite the listfile once it reaches *size* entries.

'`segment_list_entry_prefix prefix`'

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

'segment_wrap *limit*'

Wrap around segment index once it reaches *limit*.

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

6. Protocols

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

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

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

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

A description of the currently available protocols follows.

6.1 concat

Physical concatenation protocol.

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

A URL accepted by this protocol has the syntax:

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

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

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

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

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

6.2 file

File access protocol.

Allow to read from or read to a file.

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

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

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

6.3 gopher

Gopher protocol.

6.4 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.

6.5 http

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options:

'chunked_post'

> If set to 1 use chunked Transfer-Encoding for posts, default is 1.

'content_type'

Set a specific content type for the POST messages.

'headers'

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

'multiple_requests'

Use persistent connections if set to 1, default is 0.

'post_data'

Set custom HTTP post data.

'user_agent'

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

'mime_type'

Export the MIME type.

'icy'

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

'icy_metadata_headers'

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

'icy_metadata_packet'

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

'offset'

Set initial byte offset.

'end_offset'

Try to limit the request to bytes preceding this offset.

6.6 Icecast

Icecast (stream to Icecast servers)

This protocol accepts the following options:

`ice_genre`

> Set the stream genre.

`ice_name`

> Set the stream name.

`ice_description`

> Set the stream description.

`ice_url`

> Set the stream website URL.

`ice_public`

> Set if the stream should be public or not. The default is 0 (not public).

`user_agent`

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

`password`

> Set the Icecast mountpoint password.

`content_type`

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

`legacy_icecast`

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

6.7 mmst

MMS (Microsoft Media Server) protocol over TCP.

6.8 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

6.9 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.
avconv -i input.flv -f avi -y md5:output.avi.md5

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

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

6.10 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 `avconv`:

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

For writing to stdout with `avconv`:

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

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

6.11 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://[username:password@]server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

'`username`'

    An optional username (mostly for publishing).

'`password`'

    An optional password (mostly for publishing).

'`server`'

    The address of the RTMP server.

'`port`'

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

'`app`'

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

'`playpath`'

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

'`listen`'

Act as a server, listening for an incoming connection.

'`timeout`'

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

Additionally, the following parameters can be set via command line options (or in code via `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. (When publishing, the default is FMLE/3.0 (compatible; <libavformat version>).)

'`rtmp_flush_interval`'

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

'`rtmp_live`'

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

'rtmp_pageurl'

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

'rtmp_playpath'

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

'rtmp_subscribe'

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

'rtmp_swfhash'

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

'rtmp_swfsize'

Size of the decompressed SWF file, required for SWFVerification.

'rtmp_swfurl'

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

'rtmp_swfverify'

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

'rtmp_tcurl'

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

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

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

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

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

6.12 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.

6.13 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.

6.14 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.

6.15 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.

6.16 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.

6.17 librtmp rtmp, rtmpe, rtmps, rtmpt, rtmpte

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

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

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

The required syntax is:

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

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

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

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

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

To play the same stream using `avplay`:

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

6.18 rtp

Real-Time Protocol.

6.19 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 `avconv/avplay` command line, or set in code via `AVOptions` or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

'udp'

Use UDP as lower transport protocol.

'`tcp`'

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

'`udp_multicast`'

Use UDP multicast as lower transport protocol.

'`http`'

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

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

Flags for `rtsp_flags`:

'`filter_src`'

Accept packets only from negotiated peer address and port.

'`listen`'

Act as a server, listening for an incoming connection.

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

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

Example command lines:

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

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

To watch a stream tunneled over HTTP:

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

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

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

To receive a stream in realtime:

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

6.20 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.

6.20.1 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:

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

Similarly, for watching in avplay:

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

And for watching in avplay, over IPv6:

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

6.20.2 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:

```
avplay sap://
```

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

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

6.21 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

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

'listen'

Listen for an incoming connection

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

6.22 tls

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

The required syntax for a TLS url is:

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

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

'`ca_file`'

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

'`tls_verify=1|0`'

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

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

'`cert_file`'

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

'`key_file`'

A file containing the private key for the certificate.

'`listen=1|0`'

If enabled, listen for connections on the provided port, and assume the server role in the handshake instead of the client role.

6.23 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*. Follow the list of supported options.

'`buffer_size=size`'

> set the UDP buffer size in bytes

'`localport=port`'

> override the local UDP port to bind with

'`localaddr=addr`'

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

'`pkt_size=size`'

> set the size in bytes of UDP packets

'`reuse=1|0`'

> explicitly allow or disallow reusing UDP sockets

'`ttl=ttl`'

> set the time to live value (for multicast only)

'`connect=1|0`'

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

'`sources=address[,address]`'

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

'block=*address*[,*address*]'

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

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

To stream over UDP to a remote endpoint:

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

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

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

To receive over UDP from a remote endpoint:

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

6.24 unix

Unix local socket

The required syntax for a Unix socket URL is:

```
unix://filepath
```

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

'timeout'

Timeout in ms.

'listen'

Create the Unix socket in listening mode.

7. Input Devices

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

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

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

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

A description of the currently available input devices follows.

7.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

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

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

An ALSA identifier has the syntax:

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

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

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

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

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

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

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

7.2 bktr

BSD video input device.

7.3 dv1394

Linux DV 1394 input device.

7.4 fbdev

Linux framebuffer input device.

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

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

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

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

You can take a single screenshot image with the command:

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

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

7.5 jack

JACK input device.

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

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

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

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

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

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

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

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

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

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

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

7.6 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

7.7 oss

Open Sound System input device.

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

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

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

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

7.8 pulse

pulseaudio input device.

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

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

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

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

7.8.1 *server* **AVOption**

The syntax is:

```
-server server name
```

Connects to a specific server.

7.8.2 *name* **AVOption**

The syntax is:

```
-name application name
```

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

7.8.3 *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"

7.8.4 *sample_rate* **AVOption**

The syntax is:

```
-sample_rate samplerate
```

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

7.8.5 *channels* **AVOption**

The syntax is:

```
-channels N
```

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

7.8.6 *frame_size* **AVOption**

The syntax is:

```
-frame_size bytes
```

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

7.8.7 *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.

7.9 sndio

sndio input device.

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

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

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

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

7.10 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 avconv and avplay:

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

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

7.11 vfwcap

VfW (Video for Windows) capture input device.

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

7.12 x11grab

X11 video input device.

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

The filename passed as input has the syntax:

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

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

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

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

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

For example to grab from ':0.0' using avconv:

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

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

7.12.1 *follow_mouse* **AVOption**

The syntax is:

```
-follow_mouse centered|PIXELS
```

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

For example:

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

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

7.12.2 *show_region* **AVOption**

The syntax is:

```
-show_region 1
```

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

For example:

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

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

This document was generated by *john* on *June 1, 2015* using *texi2html 1.82*.

avprobe Documentation

Table of Contents

1. Synopsis

The generic syntax is:

```
avprobe [options] ['input_file']
```

2. Description

avprobe 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, avprobe 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.

avprobe 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 avprobe or for specifying which information to display, and for setting how avprobe will show it.

avprobe output is designed to be easily parsable by any INI or JSON parsers.

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 specifer, which matches the second audio stream. Therefore it would select the ac3 codec for the second audio stream.

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

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

Possible forms of stream specifiers are:

'`stream_index`'

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

'`stream_type[:stream_index]`'

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

'`p:program_id[:stream_index]`'

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

'`i:stream_id`'

> Match the stream by stream id (e.g. PID in MPEG-TS container).

'`m:key[:value]`'

> Matches streams with the metadata tag *key* having the specified value. If *value* is not given, matches streams that contain the given tag with any value.

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

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.

    'filter=*filter_name*'

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

'-version'

    Show version.

'-formats'

    Show available formats.

    The fields preceding the format names have the following meanings:

    'D'

        Decoding available

    'E'

        Encoding available

'`-codecs`'

    Show all codecs known to libavcodec.

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

'`-decoders`'

    Show available decoders.

'`-encoders`'

    Show all available encoders.

'`-bsfs`'

    Show available bitstream filters.

'`-protocols`'

    Show available protocols.

'`-filters`'

    Show available libavfilter filters.

'`-pix_fmts`'

    Show available pixel formats.

'`-sample_fmts`'

    Show available sample formats.

'`-loglevel` *`loglevel`* `|` `-v` *`loglevel`*'

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

    '`quiet`'
    '`panic`'
    '`fatal`'
    '`error`'
    '`warning`'
    '`info`'
    '`verbose`'
    '`debug`'

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

'`-cpuflags mask (`*`global`*`)`'

Set a mask that's applied to autodetected CPU flags. This option is intended for testing. Do not use it unless you know what you're doing.

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:

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

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

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

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

3.4 Codec AVOptions

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

set bitrate (in bits/s)

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

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

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

Possible values:

'`unaligned`'

allow decoders to produce unaligned output

'`mv4`'

use four motion vectors per macroblock (MPEG-4)

'`qpel`'

use 1/4-pel motion compensation

'`loop`'

use loop filter

'`qscale`'

use fixed qscale

'`gmc`'

use gmc

'`mv0`'

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

'`input_preserved`'
'`pass1`'

use internal 2-pass ratecontrol in first pass mode

'`pass2`'

use internal 2-pass ratecontrol in second pass mode

'`gray`'

only decode/encode grayscale

'emu_edge'

   do not draw edges

'psnr'

   error[?] variables will be set during encoding

'truncated'
'naq'

   normalize adaptive quantization

'ildct'

   use interlaced DCT

'low_delay'

   force low delay

'global_header'

   place global headers in extradata instead of every keyframe

'bitexact'

   use only bitexact functions (except (I)DCT)

'aic'

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

'ilme'

   interlaced motion estimation

'cgop'

   closed GOP

'output_corrupt'

   Output even potentially corrupted frames

'-me_method[:stream_specifier] integer (output,video)'

   set motion estimation method

Possible values:

'zero'

    zero motion estimation (fastest)

'full'

    full motion estimation (slowest)

'epzs'

    EPZS motion estimation (default)

'esa'

    esa motion estimation (alias for full)

'tesa'

    tesa motion estimation

'dia'

    diamond motion estimation (alias for EPZS)

'log'

    log motion estimation

'phods'

    phods motion estimation

'x1'

    X1 motion estimation

'hex'

    hex motion estimation

'umh'

    umh motion estimation

'-g[:stream_specifier] integer (output,video)'

set the group of picture (GOP) size

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

set audio sampling rate (in Hz)

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

set number of audio channels

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

set cutoff bandwidth

'-frame_size[:stream_specifier] *integer (output,audio)*'
'-qcomp[:stream_specifier] *float (output,video)*'

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

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

video quantizer scale blur (VBR)

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

minimum video quantizer scale (VBR)

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

maximum video quantizer scale (VBR)

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

maximum difference between the quantizer scales (VBR)

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

use 'frames' B frames

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

QP factor between P- and B-frames

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

ratecontrol method

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

    strategy to choose between I/P/B-frames

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

    RTP payload size in bytes

'`-bug[:stream_specifier]` *flags (input,video)*'

    work around not autodetected encoder bugs

    Possible values:

    '`autodetect`'
    '`old_msmpeg4`'

        some old lavc-generated MSMPEG4v3 files (no autodetection)

    '`xvid_ilace`'

        Xvid interlacing bug (autodetected if FOURCC == XVIX)

    '`ump4`'

        (autodetected if FOURCC == UMP4)

    '`no_padding`'

        padding bug (autodetected)

    '`amv`'
    '`ac_vlc`'

        illegal VLC bug (autodetected per FOURCC)

    '`qpel_chroma`'
    '`std_qpel`'

        old standard qpel (autodetected per FOURCC/version)

    '`qpel_chroma2`'
    '`direct_blocksize`'

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

    '`edge`'

edge padding bug (autodetected per FOURCC/version)

'hpel_chroma'
'dc_clip'
'ms'

work around various bugs in Microsoft's broken decoders

'trunc'

truncated frames

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

how strictly to follow the standards

Possible values:

'very'

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

'strict'

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

'normal'
'unofficial'

allow unofficial extensions

'experimental'

allow non-standardized experimental things

'-b_qoffset[:stream_specifier] *float (output,video*)'

QP offset between P- and B-frames

'-err_detect[:stream_specifier] *flags (input,audio,video*)'

set error detection flags

Possible values:

'crccheck'

verify embedded CRCs

'bitstream'

> detect bitstream specification deviations

'buffer'

> detect improper bitstream length

'explode'

> abort decoding on minor error detection

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

> use MPEG quantizers instead of H.263

'-qsquish[:stream_specifier] *float (output,video)*'

> 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_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 maximum bitrate tolerance (in bits/s). Requires bufsize to be set.

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

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

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

> set ratecontrol buffer size (in bits)

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

    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`'
    '`arm`'
    '`altivec`'
    '`sh4`'
    '`simplearm`'
    '`simplearmv5te`'
    '`simplearmv6`'
    '`simpleneon`'
    '`simplealpha`'
    '`ipp`'
    '`xvid`'
    '`xvidmmx`'
    '`faani`'

        floating point AAN IDCT

'`-ec[:stream_specifier]` *flags* (*input,video*)'

    set error concealment strategy

    Possible values:

    '`guess_mvs`'

        iterative motion vector (MV) search (slow)

'deblock'

 use strong deblock filter for damaged MBs

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

 prediction method

 Possible values:

 'left'
 'plane'
 'median'

'-aspect[:stream_specifier] *rational number (output,video)*'

 sample aspect ratio

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

 print specific debug info

 Possible values:

 'pict'

  picture info

 'rc'

  rate control

 'bitstream'
 'mb_type'

  macroblock (MB) type

 'qp'

  per-block quantization parameter (QP)

 'mv'

  motion vector

 'dct_coeff'
 'skip'
 'startcode'
 'pts'

‘er’

    error recognition

‘mmco’

    memory management control operations (H.264)

‘bugs’
‘vis_qp’

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

‘vis_mb_type’

    visualize block types

‘buffers’

    picture buffer allocations

‘thread_ops’

    threading operations

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

    visualize motion vectors (MVs)

    Possible values:

    ‘pf’

        forward predicted MVs of P-frames

    ‘bf’

        forward predicted MVs of B-frames

    ‘bb’

        backward predicted MVs of B-frames

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

    full-pel ME compare function

    Possible values:

'sad'

> sum of absolute differences, fast (default)

'sse'

> sum of squared errors

'satd'

> sum of absolute Hadamard transformed differences

'dct'

> sum of absolute DCT transformed differences

'psnr'

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

'bit'

> number of bits needed for the block

'rd'

> rate distortion optimal, slow

'zero'

> 0

'vsad'

> sum of absolute vertical differences

'vsse'

> sum of squared vertical differences

'nsse'

> noise preserving sum of squared differences

'dctmax'
'chroma'
'-subcmp[:stream_specifier] integer (output,video)'

sub-pel ME compare function

Possible values:

'sad'

    sum of absolute differences, fast (default)

'sse'

    sum of squared errors

'satd'

    sum of absolute Hadamard transformed differences

'dct'

    sum of absolute DCT transformed differences

'psnr'

    sum of squared quantization errors (avoid, low quality)

'bit'

    number of bits needed for the block

'rd'

    rate distortion optimal, slow

'zero'

    0

'vsad'

    sum of absolute vertical differences

'vsse'

    sum of squared vertical differences

'nsse'

    noise preserving sum of squared differences

'dctmax'

'chroma'
'-mbcmp[:stream_specifier] *integer* (*output,video*)'

macroblock compare function

Possible values:

'sad'

   sum of absolute differences, fast (default)

'sse'

   sum of squared errors

'satd'

   sum of absolute Hadamard transformed differences

'dct'

   sum of absolute DCT transformed differences

'psnr'

   sum of squared quantization errors (avoid, low quality)

'bit'

   number of bits needed for the block

'rd'

   rate distortion optimal, slow

'zero'

   0

'vsad'

   sum of absolute vertical differences

'vsse'

   sum of squared vertical differences

'nsse'

noise preserving sum of squared differences

'dctmax'
'chroma'
'-ildctcmp[:stream_specifier] *integer (output,video)*'

interlaced DCT compare function

Possible values:

'sad'

sum of absolute differences, fast (default)

'sse'

sum of squared errors

'satd'

sum of absolute Hadamard transformed differences

'dct'

sum of absolute DCT transformed differences

'psnr'

sum of squared quantization errors (avoid, low quality)

'bit'

number of bits needed for the block

'rd'

rate distortion optimal, slow

'zero'

0

'vsad'

sum of absolute vertical differences

'vsse'

sum of squared vertical differences

'nsse'

noise preserving sum of squared differences

'dctmax'
'chroma'
'-dia_size[:stream_specifier] *integer (output,video)*'

diamond type & size for motion estimation

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

amount of motion predictors from the previous frame

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

pre motion estimation

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

pre motion estimation compare function

Possible values:

'sad'

sum of absolute differences, fast (default)

'sse'

sum of squared errors

'satd'

sum of absolute Hadamard transformed differences

'dct'

sum of absolute DCT transformed differences

'psnr'

sum of squared quantization errors (avoid, low quality)

'bit'

number of bits needed for the block

'rd'

rate distortion optimal, slow

'zero'

0

'vsad'

sum of absolute vertical differences

'vsse'

sum of squared vertical differences

'nsse'

noise preserving sum of squared differences

'dctmax'
'chroma'
'-pre_dia_size[:stream_specifier] *integer* (*output,video*)'

diamond type & size for motion estimation pre-pass

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

sub-pel motion estimation quality

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

limit motion vectors range (1023 for DivX player)

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

intra quant bias

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

inter quant bias

'-global_quality[:stream_specifier] *integer* (*output,audio,video*)'
'-coder[:stream_specifier] *integer* (*output,video*)'

Possible values:

'vlc'

> variable length coder / Huffman coder

'ac'

> arithmetic coder

'raw'

> raw (no encoding)

'rle'

> run-length coder

'deflate'

> deflate-based coder

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

> context model

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

> macroblock decision algorithm (high quality mode)

> Possible values:

> 'simple'

>> use mbcmp (default)

> 'bits'

>> use fewest bits

> 'rd'

>> use best rate distortion

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

> scene change threshold

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

minimum Lagrange factor (VBR)

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

maximum 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

'-flags2[:stream_specifier] *flags* (*input/output,audio,video*)'

Possible values:

'fast'

allow non-spec-compliant speedup tricks

'noout'

skip bitstream encoding

'ignorecrop'

ignore cropping information from sps

'local_header'

place global headers at every keyframe instead of in extradata

'-error[:stream_specifier] *integer* (*output,video*)'
'-threads[:stream_specifier] *integer* (*input/output,video*)'

Possible values:

'auto'

autodetect a suitable number of threads to use

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

motion estimation threshold

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

macroblock threshold

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

    intra_dc_precision

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

    nsse weight

'-skip_top[:stream_specifier] *integer* (*input,video*)'

    number of macroblock rows at the top which are skipped

'-skip_bottom[:stream_specifier] *integer* (*input,video*)'

    number of macroblock rows at the bottom which are skipped

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

    Possible values:

    'unknown'
    'aac_main'
    'aac_low'
    'aac_ssr'
    'aac_ltp'
    'aac_he'
    'aac_he_v2'
    'aac_ld'
    'aac_eld'
    'mpeg2_aac_low'
    'mpeg2_aac_he'
    'dts'
    'dts_es'
    'dts_96_24'
    'dts_hd_hra'
    'dts_hd_ma'

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

    Possible values:

    'unknown'

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

    frame skip threshold

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

    frame skip factor

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

    frame skip exponent

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

    frame skip compare function

    Possible values:

    'sad'

        sum of absolute differences, fast (default)

    'sse'

        sum of squared errors

    'satd'

        sum of absolute Hadamard transformed differences

    'dct'

        sum of absolute DCT transformed differences

    'psnr'

        sum of squared quantization errors (avoid, low quality)

    'bit'

        number of bits needed for the block

    'rd'

        rate distortion optimal, slow

    'zero'

        0

    'vsad'

sum of absolute vertical differences

'vsse'

sum of squared vertical differences

'nsse'

noise preserving sum of squared differences

'dctmax'
'chroma'
'-border_mask[:stream_specifier] float (output,video)'

increase the quantizer for macroblocks close to borders

'-mblmin[:stream_specifier] integer (output,video)'

minimum macroblock Lagrange factor (VBR)

'-mblmax[:stream_specifier] integer (output,video)'

maximum macroblock Lagrange factor (VBR)

'-mepc[:stream_specifier] integer (output,video)'

motion estimation bitrate penalty compensation (1.0 = 256)

'-skip_loop_filter[:stream_specifier] integer (input,video)'

Possible values:

'none'
'default'
'noref'
'bidir'
'nokey'
'all'
'-skip_idct[:stream_specifier] integer (input,video)'

Possible values:

'none'
'default'
'noref'
'bidir'
'nokey'

'all'
'-skip_frame[:stream_specifier] *integer* (*input,video*)'

    Possible values:

    'none'
    'default'
    'noref'
    'bidir'
    'nokey'
    'all'
'-bidir_refine[:stream_specifier] *integer* (*output,video*)'

    refine the two motion vectors used in bidirectional macroblocks

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

    downscale frames for dynamic B-frame decision

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

    minimum interval between IDR-frames (x264)

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

    reference frames to consider for motion compensation

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

    chroma QP offset from luma

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

    rate-distortion optimal quantization

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

    multiplied by qscale for each frame and added to scene_change_score

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

    adjust sensitivity of b_frame_strategy 1

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

GOP timecode frame start number, in non-drop-frame format

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

set desired number of audio channels

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

Possible values:

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

Possible values:

'`-rc_max_vbv_use[:stream_specifier]` *float* (*output,video*)'
'`-rc_min_vbv_use[:stream_specifier]` *float* (*output,video*)'
'`-ticks_per_frame[:stream_specifier]` *integer* (*input/output,audio,video*)'
'`-color_primaries[:stream_specifier]` *integer* (*input/output,video*)'

color primaries

Possible values:

'`bt709`'

BT.709

'`unspecified`'

Unspecified

'`bt470m`'

BT.470 M

'`bt470bg`'

BT.470 BG

'`smpte170m`'

SMPTE 170 M

'`smpte240m`'

SMPTE 240 M

'`film`'

Film

'bt2020'

　　BT.2020

'-color_trc[:stream_specifier] *integer (input/output,video)*'

color transfert characteristic

Possible values:

'bt709'

　　BT.709

'unspecified'

　　Unspecified

'gamma22'

　　Gamma 2.2

'gamma28'

　　Gamma 2.8

'smpte170m'

　　SMPTE 170 M

'smpte240m'

　　SMPTE 240 M

'linear'

　　Linear

'log'

　　SMPTE 240 M

'log_sqrt'

　　SMPTE 240 M

'iec61966_2_4'

> SMPTE 240 M

'`bt1361`'

> BT.1361

'`iec61966_2_1`'

> SMPTE 240 M

'`bt2020_10bit`'

> BT.2020 - 10 bit

'`bt2020_12bit`'

> BT.2020 - 12 bit

'`-colorspace[:stream_specifier]` *integer* (*input/output,video*)'

colorspace

Possible values:

'`rgb`'

> RGB

'`bt709`'

> BT.709

'`unspecified`'

> Unspecified

'`fcc`'

> FourCC

'`bt470bg`'

> BT.470 BG

'`smpte170m`'

> SMPTE 170 M

'`smpte240m`'

SMPTE 240 M

'ycocg'

YCOCG

'bt2020_ncl'

BT.2020 NCL

'bt2020_cl'

BT.2020 CL

'-color_range[:stream_specifier] *integer* (*input/output,video*)'

color range

Possible values:

'unspecified'

Unspecified

'mpeg'

MPEG ($219*2^{(n-8)}$)

'jpeg'

JPEG (2^n-1)

'-chroma_sample_location[:stream_specifier] *integer* (*input/output,video*)'
'-slices[:stream_specifier] *integer* (*output,video*)'

number of slices, used in parallelized encoding

'-thread_type[:stream_specifier] *flags* (*input/output,video*)'

select multithreading type

Possible values:

'slice'
'frame'
'-audio_service_type[:stream_specifier] *integer* (*output,audio*)'

audio service type

Possible values:

'ma'

   Main Audio Service

'ef'

   Effects

'vi'

   Visually Impaired

'hi'

   Hearing Impaired

'di'

   Dialogue

'co'

   Commentary

'em'

   Emergency

'vo'

   Voice Over

'ka'

   Karaoke

'-request_sample_fmt[:stream_specifier] *integer* (*input,audio*)'

   Possible values:

'u8'

   8-bit unsigned integer

's16'

16-bit signed integer

'`s32`'

32-bit signed integer

'`flt`'

32-bit float

'`dbl`'

64-bit double

'`u8p`'

8-bit unsigned integer planar

'`s16p`'

16-bit signed integer planar

'`s32p`'

32-bit signed integer planar

'`fltp`'

32-bit float planar

'`dblp`'

64-bit double planar

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

3.5 Format AVOptions

'`-probesize` *integer* (*input*)'

set probing size

'`-packetsize` *integer* (*output*)'

set packet size

'`-fflags` *flags* (*input/output*)'

> Possible values:
>
> '`flush_packets`'
>
>> reduce the latency by flushing out packets immediately
>
> '`ignidx`'
>
>> ignore index
>
> '`genpts`'
>
>> generate pts
>
> '`nofillin`'
>
>> do not fill in missing values that can be exactly calculated
>
> '`noparse`'
>
>> disable AVParsers, this needs nofillin too
>
> '`igndts`'
>
>> ignore dts
>
> '`discardcorrupt`'
>
>> discard corrupted frames
>
> '`nobuffer`'
>
>> reduce the latency introduced by optional buffering
>
> '`bitexact`'
>
>> do not write random/volatile data

'`-analyzeduration` *integer* (*input*)'

> how many microseconds are analyzed to estimate duration

'`-cryptokey` *hexadecimal string* (*input*)'

> decryption key

'`-indexmem` *integer* (*input*)'

max memory used for timestamp index (per stream)

'-rtbufsize `integer` (`input`)'

max memory used for buffering real-time frames

'-fdebug `flags` (`input/output`)'

print specific debug info

Possible values:

'ts'

'-max_delay `integer` (`input/output`)'

maximum muxing or demuxing delay in microseconds

'-fpsprobesize `integer` (`input`)'

number of frames used to probe fps

'-f_err_detect `flags` (`input`)'

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

Possible values:

'crccheck'

verify embedded CRCs

'bitstream'

detect bitstream specification deviations

'buffer'

detect improper bitstream length

'explode'

abort decoding on minor error detection

'-err_detect `flags` (`input`)'

set error detection flags

Possible values:

'crccheck'

> verify embedded CRCs

'bitstream'

> detect bitstream specification deviations

'buffer'

> detect improper bitstream length

'explode'

> abort decoding on minor error detection

'-max_interleave_delta *integer* (*output*)'

> maximum buffering duration for interleaving

'-f_strict *integer* (*input/output*)'

> how strictly to follow the standards (deprecated; use strict, save via avconv)

> Possible values:

> 'strict'

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

> 'normal'
> 'experimental'

>> allow non-standardized experimental variants

'-strict *integer* (*input/output*)'

> how strictly to follow the standards

> Possible values:

> 'strict'

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

> 'normal'
> 'experimental'

allow non-standardized experimental variants

3.6 Main options

'-f *format*'

Force format to use.

'-of *formatter*'

Use a specific formatter to output the document. The following formatters are available

'ini'
'json'
'old'

Pseudo-INI format that used to be the only one available in old avprobe versions.

'-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".

'-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.

'`-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_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".

4. Demuxers

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

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

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

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

The description of some of the currently available demuxers follows.

4.1 image2

Image file demuxer.

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

The pattern may contain the string "%d" or "%0Nd", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0Nd" is used, the string representing the number in each filename is 0-padded and N is the total number of 0-padded digits representing the number. The literal character '%' can be specified in the pattern with the string "%%".

If the pattern contains "%d" or "%0Nd", the first filename of the file list specified by the pattern must contain a number inclusively contained between 0 and 4, all the following numbers must be sequential. This limitation may be hopefully fixed.

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

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

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

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

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

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

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

'-pixel_format *format*'

    Set the pixel format (for raw image)

'-video_size *size*'

    Set the frame size (for raw image)

'-framerate *rate*'

    Set the frame rate

'-loop *bool*'

    Loop over the images

'-start_number *start*'

    Specify the first number in the sequence

4.2 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 avplay), the caller can decide which variant streams to actually receive. The total bitrate of the variant that the stream belongs to is available in a metadata key named "variant_bitrate".

4.3 flv

Adobe Flash Video Format demuxer.

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

'`-flv_metadata` *`bool`*'

 Allocate the streams according to the onMetaData array content.

4.4 asf

Advanced Systems Format demuxer.

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

'`-no_resync_search` *`bool`*'

 Do not try to resynchronize by looking for a certain optional start code.

5. Muxers

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

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

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

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

A description of some of the currently available muxers follows.

5.1 crc

CRC (Cyclic Redundancy Check) testing format.

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

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

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

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

You can print the CRC to stdout with the command:

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

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

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

See also the framecrc muxer.

5.2 framecrc

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

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

The output of the muxer consists of a line for each audio and video frame of the form: *stream_index*, *frame_dts*, *frame_size*, 0x*CRC*, where *CRC* is a hexadecimal number 0-padded to 8 digits containing the CRC of the decoded frame.

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

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

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

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

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

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

See also the crc muxer.

5.3 hls

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

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

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

'-hls_time *seconds*'

Set the segment length in seconds.

'-hls_list_size *size*'

Set the maximum number of playlist entries.

'-hls_wrap *wrap*'

Set the number after which index wraps.

'-start_number *number*'

Start the sequence from *number*.

'-hls_base_url *baseurl*'

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

5.4 image2

Image file muxer.

The image file muxer writes video frames to image files.

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

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

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

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

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

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

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

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

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

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

'-start_number *number*'

    Start the sequence from *number*.

'-update *number*'

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

5.5 matroska

Matroska container muxer.

This muxer implements the matroska and webm container specs.

The recognized metadata settings in this muxer are:

'title=*title name*'

    Name provided to a single track

'language=*language name*'

    Specifies the language of the track in the Matroska languages form

'STEREO_MODE=*mode*'

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

    'mono'

        video is not stereo

    'left_right'

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

    'bottom_top'

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

    'top_bottom'

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

    'checkerboard_rl'

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

    'checkerboard_lr'

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

    'row_interleaved_rl'

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

'row_interleaved_lr'

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

'col_interleaved_rl'

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

'col_interleaved_lr'

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

'anaglyph_cyan_red'

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

'right_left'

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

'anaglyph_green_magenta'

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

'block_lr'

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

'block_rl'

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

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

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

This muxer supports the following options:

'reserve_index_space'

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

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

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

5.6 mov, mp4, ismv

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

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

'`-movflags frag_keyframe`'

 Start a new fragment at each video keyframe.

'`-frag_duration duration`'

 Create fragments that are *duration* microseconds long.

'`-frag_size size`'

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

'`-movflags frag_custom`'

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

'`-min_frag_duration duration`'

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

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

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

'`-movflags empty_moov`'

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

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

'-movflags separate_moof'

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

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

'-movflags faststart'

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

'-movflags disable_chpl'

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

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

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

5.7 mp3

The MP3 muxer writes a raw MP3 stream with an ID3v2 header at the beginning and optionally an ID3v1 tag at the end. ID3v2.3 and ID3v2.4 are supported, the `id3v2_version` option controls which one is used. Setting `id3v2_version` to 0 will disable the ID3v2 header completely. The legacy ID3v1 tag is not written by default, but may be enabled with the `write_id3v1` option.

The muxer may also write a Xing frame at the beginning, which contains the number of frames in the file. It is useful for computing duration of VBR files. The Xing frame is written if the output stream is seekable and if the `write_xing` option is set to 1 (the default).

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

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

Examples:

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

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

Attach a picture to an mp3:

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

Write a "clean" MP3 without any extra features:

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

5.8 mpegts

MPEG transport stream muxer.

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

The muxer options are:

'-mpegts_original_network_id *number*'

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

'-mpegts_transport_stream_id *number*'

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

'-mpegts_service_id *number*'

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

'-mpegts_pmt_start_pid *number*'

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

'-mpegts_start_pid *number*'

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

'`-muxrate number`'

    Set a constant muxrate (default VBR).

'`-pcr_period numer`'

    Override the default PCR retransmission time (default 20ms), ignored if variable muxrate is selected.

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

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

5.9 null

Null muxer.

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

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

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

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

Alternatively you can write the command as:

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

5.10 nut

'`-syncpoints flags`'

    Change the syncpoint usage in nut:

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

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

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

5.11 ogg

Ogg container muxer.

'-page_duration *duration*'

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

5.12 segment

Basic stream segmenter.

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

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

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

'segment_format *format*'

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

'segment_time *t*'

Set segment duration to *t* seconds.

'segment_list *name*'

Generate also a listfile named *name*.

'`segment_list_type` *`type`*'

> Select the listing format.
>
> '*`flat`* `use a simple flat list of entries.`'
> '*`hls`* `use a m3u8-like structure.`'

'`segment_list_size` *`size`*'

> Overwrite the listfile once it reaches *size* entries.

'`segment_list_entry_prefix` *`prefix`*'

> Prepend *prefix* to each entry. Useful to generate absolute paths.

'`segment_wrap` *`limit`*'

> Wrap around segment index once it reaches *limit*.

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

6. Protocols

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

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

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

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

A description of the currently available protocols follows.

6.1 concat

Physical concatenation protocol.

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

A URL accepted by this protocol has the syntax:

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

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

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

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

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

6.2 file

File access protocol.

Allow to read from or read to a file.

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

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

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

6.3 gopher

Gopher protocol.

6.4 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.

6.5 http

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options:

'chunked_post'

> If set to 1 use chunked Transfer-Encoding for posts, default is 1.

'content_type'

> Set a specific content type for the POST messages.

'headers'

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

'multiple_requests'

> Use persistent connections if set to 1, default is 0.

'post_data'

> Set custom HTTP post data.

'user_agent'

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

'mime_type'

> Export the MIME type.

'icy'

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

'icy_metadata_headers'

> If the server supports ICY metadata, this contains the ICY-specific HTTP reply headers, separated by newline characters.

'icy_metadata_packet'

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

'offset'

Set initial byte offset.

'end_offset'

Try to limit the request to bytes preceding this offset.

6.6 Icecast

Icecast (stream to Icecast servers)

This protocol accepts the following options:

'ice_genre'

Set the stream genre.

'ice_name'

Set the stream name.

'ice_description'

Set the stream description.

'ice_url'

Set the stream website URL.

'ice_public'

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

'user_agent'

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

'password'

Set the Icecast mountpoint password.

'content_type'

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

'legacy_icecast'

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

6.7 mmst

MMS (Microsoft Media Server) protocol over TCP.

6.8 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

6.9 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.
avconv -i input.flv -f avi -y md5:output.avi.md5

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

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

6.10 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 `avconv`:

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

For writing to stdout with `avconv`:

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

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

6.11 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://[username:password@]server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

'`username`'

    An optional username (mostly for publishing).

'`password`'

    An optional password (mostly for publishing).

'server'

The address of the RTMP server.

'port'

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

'app'

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

'playpath'

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

'listen'

Act as a server, listening for an incoming connection.

'timeout'

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

Additionally, the following parameters can be set via command line options (or in code via `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. (When publishing, the default is FMLE/3.0 (compatible; <libavformat version>).)

'`rtmp_flush_interval`'

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

'`rtmp_live`'

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

'`rtmp_pageurl`'

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

'`rtmp_playpath`'

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

'`rtmp_subscribe`'

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

'`rtmp_swfhash`'

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

'`rtmp_swfsize`'

Size of the decompressed SWF file, required for SWFVerification.

'`rtmp_swfurl`'

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

'`rtmp_swfverify`'

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

'`rtmp_tcurl`'

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

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

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

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

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

6.12 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.

6.13 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.

6.14 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.

6.15 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.

6.16 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.

6.17 librtmp rtmp, rtmpe, rtmps, rtmpt, rtmpte

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

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

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

The required syntax is:

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

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

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

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

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

To play the same stream using `avplay`:

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

6.18 rtp

Real-Time Protocol.

6.19 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 `avconv`/`avplay` command line, or set in code via `AVOptions` or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

'`udp`'

> Use UDP as lower transport protocol.

'`tcp`'

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

'`udp_multicast`'

> Use UDP multicast as lower transport protocol.

'`http`'

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

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

Flags for `rtsp_flags`:

'`filter_src`'

> Accept packets only from negotiated peer address and port.

'`listen`'

> Act as a server, listening for an incoming connection.

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

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

Example command lines:

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

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

To watch a stream tunneled over HTTP:

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

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

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

To receive a stream in realtime:

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

6.20 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.

6.20.1 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:

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

Similarly, for watching in avplay:

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

And for watching in avplay, over IPv6:

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

6.20.2 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:

```
avplay sap://
```

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

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

6.21 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

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

'listen'

Listen for an incoming connection

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

6.22 tls

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

The required syntax for a TLS url is:

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

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

'ca_file'

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

'tls_verify=1|0'

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

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

'cert_file'

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

'key_file'

> A file containing the private key for the certificate.

'listen=*1|0*'

> If enabled, listen for connections on the provided port, and assume the server role in the handshake instead of the client role.

6.23 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*. Follow the list of supported options.

'buffer_size=*size*'

> set the UDP buffer size in bytes

'localport=*port*'

> override the local UDP port to bind with

'localaddr=*addr*'

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

'pkt_size=*size*'

> set the size in bytes of UDP packets

'reuse=*1|0*'

> explicitly allow or disallow reusing UDP sockets

'ttl=*ttl*'

> set the time to live value (for multicast only)

'connect=*1|0*'

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

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

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

'block=*address*[,*address*]'

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

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

To stream over UDP to a remote endpoint:

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

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

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

To receive over UDP from a remote endpoint:

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

6.24 unix

Unix local socket

The required syntax for a Unix socket URL is:

```
unix://filepath
```

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

'`timeout`'

> Timeout in ms.

'`listen`'

> Create the Unix socket in listening mode.

7. Input Devices

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

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

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

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

A description of the currently available input devices follows.

7.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

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

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

An ALSA identifier has the syntax:

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

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

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

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

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

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

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

7.2 bktr

BSD video input device.

7.3 dv1394

Linux DV 1394 input device.

7.4 fbdev

Linux framebuffer input device.

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

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

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

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

You can take a single screenshot image with the command:

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

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

7.5 jack

JACK input device.

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

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

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

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

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

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

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

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

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

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

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

7.6 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

7.7 oss

Open Sound System input device.

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

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

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

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

7.8 pulse

pulseaudio input device.

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

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

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

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

7.8.1 *server* AVOption

The syntax is:

```
-server server name
```

Connects to a specific server.

7.8.2 *name* AVOption

The syntax is:

```
-name application name
```

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

7.8.3 *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"

7.8.4 *sample_rate* **AVOption**

The syntax is:

```
-sample_rate samplerate
```

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

7.8.5 *channels* **AVOption**

The syntax is:

```
-channels N
```

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

7.8.6 *frame_size* **AVOption**

The syntax is:

```
-frame_size bytes
```

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

7.8.7 *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.

7.9 sndio

sndio input device.

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

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

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

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

7.10 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 avconv and avplay:

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

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

7.11 vfwcap

VfW (Video for Windows) capture input device.

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

7.12 x11grab

X11 video input device.

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

The filename passed as input has the syntax:

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

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

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

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

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

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

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

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

7.12.1 *follow_mouse* **AVOption**

The syntax is:

```
-follow_mouse centered|PIXELS
```

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

For example:

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

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

7.12.2 *show_region* **AVOption**

The syntax is:

```
-show_region 1
```

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

For example:

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

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

This document was generated by *john* on *June 1, 2015* using *texi2html 1.82*.