ffmpeg Documentation

Table of Contents

1. Synopsis

ffmpeg [*global_options*] {[*input_file_options*] -i ‘`input_file`’} ... {[*output_file_options*] ‘`output_file`’} ...

2. Description

`ffmpeg` is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

`ffmpeg` reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output filename. Anything found on the command line which cannot be interpreted as an option is considered to be an output filename.

Each input or output file can, in principle, contain any number of streams of different types (video/audio/subtitle/attachment/data). The allowed number and/or types of streams may be limited by the container format. Selecting which streams from which inputs will go into which output is either done automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1`, etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. Also see the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

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

  ```
  ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
  ```

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

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

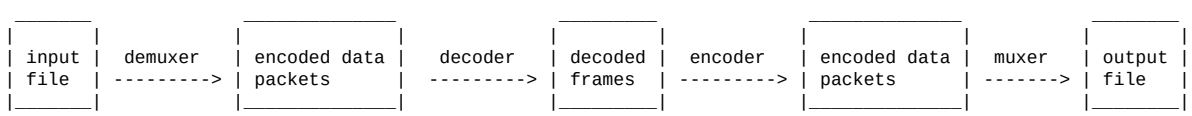
- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

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

The format option may be needed for raw input files.

3. Detailed description

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



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

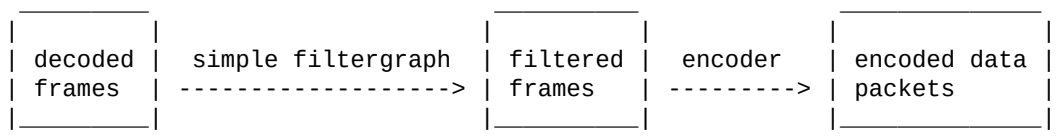
Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering, the frames are passed to the encoder, which encodes them and outputs encoded packets. Finally those are passed to the muxer, which writes the encoded packets to the output file.

3.1 Filtering

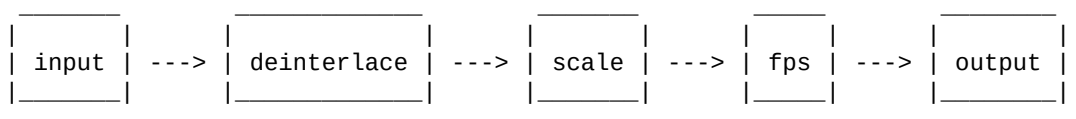
Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs: simple and complex.

3.1.1 Simple filtergraphs

Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:



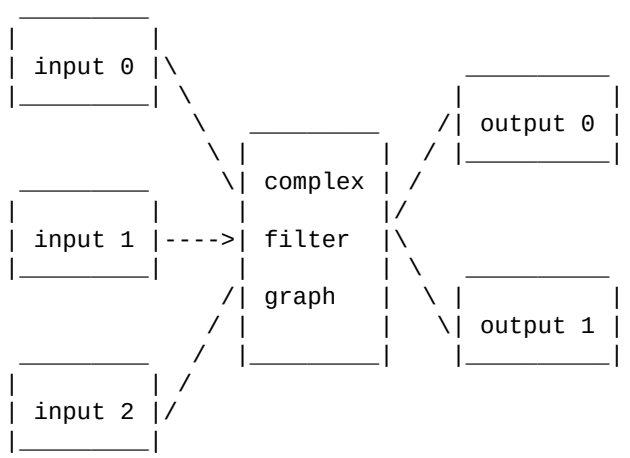
Simple filtergraphs are configured with the per-stream ‘`-filter`’ option (with ‘`-vf`’ and ‘`-af`’ aliases for video and audio respectively). A simple filtergraph for video can look for example like this:



Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

3.1.2 Complex filtergraphs

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case, for example, when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:



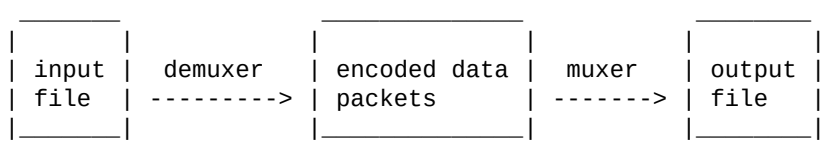
Complex filtergraphs are configured with the '`-filter_complex`' option. Note that this option is global, since a complex filtergraph, by its nature, cannot be unambiguously associated with a single stream or file.

The '`-lavfi`' option is equivalent to '`-filter_complex`'.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

3.2 Stream copy

Stream copy is a mode selected by supplying the `copy` parameter to the '`-codec`' option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will, in this case, simplify to this:



Since there is no decoding or encoding, it is very fast and there is no quality loss. However, it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

4. Stream selection

By default, `ffmpeg` includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria: for video, it is the stream with the highest resolution, for audio, it is the stream with the most channels, for subtitles, it is the first subtitle stream. In the case where several streams of the same type rate equally, the stream with the lowest index is chosen.

You can disable some of those defaults by using the `-vn/-an/-sn` options. For full manual control, use the `-map` option, which disables the defaults just described.

5. Options

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: ’K’, ’M’, or ’G’.

If ’i’ is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiplies, which are based on powers of 1024 instead of powers of 1000. Appending ’B’ to the SI unit prefix multiplies the value by 8. This allows using, for example: ’KB’, ’MiB’, ’G’ and ’B’ as number suffixes.

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

5.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs 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` contains the `a:1` stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is 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 following: ’v’ for video, ’a’ for audio, ’s’ for subtitle, ’d’ for data, and ’t’ for attachments. If *stream_index* is given, then it matches stream number *stream_index* of this type. Otherwise, it matches all streams of this type.

‘p:*program_id*[:*stream_index*]’

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

‘#*stream_id*’

Matches the stream by a format-specific ID.

5.2 Generic options

These options are shared amongst the ff* 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.

‘`-codecs`’

Show all codecs known to libavcodec.

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

‘`-decoders`’

Show available decoders.

‘`-encoders`’

Show all available encoders.

‘`-bsfs`’

Show available bitstream filters.

‘`-protocols`’

Show available protocols.

‘`-filters`’

Show available libavfilter filters.

‘`-pix_fmts`’

Show available pixel formats.

‘`-sample_fmts`’

Show available sample formats.

‘`-layouts`’

Show channel names and standard channel layouts.

‘`-loglevel [repeat+]`*`loglevel`*` | -v [repeat+]`*`loglevel`*’

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using ’repeat’ will not change the loglevel. *loglevel* is a number or a string containing one of the following values:

‘`quiet`’

Show nothing at all; be silent.

‘`panic`’

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

‘`fatal`’

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

‘`error`’

Show all errors, including ones which can be recovered from.

‘`warning`’

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

‘`info`’

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

‘`verbose`’

Same as `info`, except more verbose.

‘`debug`’

Show everything, including debugging information.

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

‘`-report`’

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

Setting the environment variable `FFREPORT` to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ’:’ (see the “Quoting and escaping” section in the ffmpeg-utils manual). The following option is recognized:

‘`file`’

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

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

‘`-cpuflags flags (`*`global`*`)`’

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

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

Possible flags for this option are:

‘`x86`’
- ‘`mmx`’
- ‘`mmxext`’
- ‘`sse`’
- ‘`sse2`’
- ‘`sse2slow`’

‘sse3’
‘sse3slow’
‘ssse3’
‘atom’
‘sse4.1’
‘sse4.2’
‘avx’
‘xop’
‘fma4’
‘3dnow’
‘3dnowext’
‘cmov’
‘ARM’
‘armv5te’
‘armv6’
‘armv6t2’
‘vfp’
‘vfpv3’
‘neon’
‘PowerPC’
‘altivec’
‘Specific Processors’
‘pentium2’
‘pentium3’
‘pentium4’
‘k6’
‘k62’
‘athlon’
‘athlonxp’
‘k8’

‘-opencl_options options (*global*)’

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

options must be a list of *key=value* option pairs separated by ’:’. See the “OpenCL Options” section in the ffmpeg-utils manual for the list of supported options.

5.3 AVOptions

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

‘generic’

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

‘`private`’

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

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

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

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

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

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

5.4 Main options

‘`-f` *`fmt`* (*`input/output`*)’

Force input or output file format. The format is normally auto detected for input files and guessed from the file extension for output files, so this option is not needed in most cases.

‘`-i` *`filename`* (*`input`*)’

input file name

‘`-y` (*`global`*)’

Overwrite output files without asking.

‘`-n` (*`global`*)’

Do not overwrite output files, and exit immediately if a specified output file already exists.

‘`-c[:`*`stream_specifier`*`]` *`codec`* (*`input/output,per-stream`*)’
‘`-codec[:`*`stream_specifier`*`]` *`codec`* (*`input/output,per-stream`*)’

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

‘`-t` *`duration`* (*`output`*)’

Stop writing the output after its duration reaches *duration*. *duration* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

‘`-to` *`position`* (*`output`*)’

Stop writing the output at *position*. *position* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

‘`-fs` *`limit_size`* (*`output`*)’

Set the file size limit, expressed in bytes.

‘`-ss` *`position`* (*`input/output`*)’

When used as an input option (before `-i`), seeks in this input file to *position*. When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*. This is slower, but more accurate.

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

‘`-itsoffset` *`offset`* (*`input`*)’

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

‘`-timestamp` *`time`* (*`output`*)’

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

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

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

'`-metadata[:metadata_specifier] key=value (output,per-metadata)`'

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams or chapters. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

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

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:1 language=eng OUTPUT
```

'`-target type (output)`'

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

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

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

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

'`-dframes number (output)`'

Set the number of data frames to record. This is an alias for `-frames:d`.

‘`-frames[:stream_specifier] framecount (output,per-stream)`’

Stop writing to the stream after *framecount* frames.

‘`-q[:stream_specifier] q (output,per-stream)`’
‘`-qscale[:stream_specifier] q (output,per-stream)`’

Use fixed quality scale (VBR). The meaning of *q* is codec-dependent.

‘`-filter[:stream_specifier] filtergraph (output,per-stream)`’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

filtergraph is a description of the filtergraph to apply to the stream, and must have a single input and a single output of the same type of the stream. In the filtergraph, the input is associated to the label `in`, and the output to the label `out`. See the ffmpeg-filters manual for more information about the filtergraph syntax.

See the -filter_complex option if you want to create filtergraphs with multiple inputs and/or outputs.

‘`-filter_script[:stream_specifier] filename (output,per-stream)`’

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

‘`-pre[:stream_specifier] preset_name (output,per-stream)`’

Specify the preset for matching stream(s).

‘`-stats (global)`’

Print encoding progress/statistics. It is on by default, to explicitly disable it you need to specify `-nostats`.

‘`-progress url (global)`’

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key=value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

‘`-stdin`’

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

‘`-debug_ts (`*`global`*`)`’

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option `-fdebug ts`.

‘`-attach `*`filename`*` (`*`output`*`)`’

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

‘`-dump_attachment[:`*`stream_specifier`*`] `*`filename`*` (`*`input,per-stream`*`)`’

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the `filename` metadata tag will be used.

E.g. to extract the first attachment to a file named ’out.ttf’:

```
ffmpeg -dump_attachment:t:0 out.ttf -i INPUT
```

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

```
ffmpeg -dump_attachment:t "" -i INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

5.5 Video Options

‘`-vframes` *`number`* `(`*`output`*`)`’

Set the number of video frames to record. This is an alias for `-frames:v`.

‘`-r[:`*`stream_specifier`*`]` *`fps`* `(`*`input/output,per-stream`*`)`’

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps*.

‘`-s[:`*`stream_specifier`*`]` *`size`* `(`*`input/output,per-stream`*`)`’

Set frame size.

As an input option, this is a shortcut for the ‘`video_size`’ private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the `scale` video filter to the *end* of the corresponding filtergraph. Please use the `scale` filter directly to insert it at the beginning or some other place.

The format is ‘`wxh`’ (default - same as source).

‘`-aspect[:`*`stream_specifier`*`]` *`aspect`* `(`*`output,per-stream`*`)`’

Set the video display aspect ratio specified by *aspect*.

aspect can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

If used together with ‘`-vcodec copy`’, it will affect the aspect ratio stored at container level, but not the aspect ratio stored in encoded frames, if it exists.

‘`-vn (`*`output`*`)`’

Disable video recording.

‘`-vcodec` *`codec`* `(`*`output`*`)`’

Set the video codec. This is an alias for `-codec:v`.

‘-pass[:*stream_specifier*] *n* (*output,per-stream*)’

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

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

‘-passlogfile[:*stream_specifier*] *prefix* (*output,per-stream*)’

Set two-pass log file name prefix to *prefix*, the default file name prefix is “ffmpeg2pass”. The complete file name will be ‘PREFIX-N.log’, where N is a number specific to the output stream

‘-vlang *code*’

Set the ISO 639 language code (3 letters) of the current video stream.

‘-vf *filtergraph* (*output*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for -filter:v, see the -filter option.

5.6 Advanced Video Options

‘-pix_fmt[:*stream_specifier*] *format* (*input/output,per-stream*)’

Set pixel format. Use -pix_fmts to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a +, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions inside filtergraphs are disabled. If *pix_fmt* is a single +, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

‘-sws_flags *flags* (*input/output*)’

Set SwScaler flags.

‘-vdt *n*’

Discard threshold.

‘-rc_override[:*stream_specifier*] *override* (*output,per-stream*)’

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

‘`-deinterlace`’

Deinterlace pictures. This option is deprecated since the deinterlacing is very low quality. Use the yadif filter with `-filter:v yadif`.

‘`-ilme`’

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

‘`-psnr`’

Calculate PSNR of compressed frames.

‘`-vstats`’

Dump video coding statistics to ‘`vstats_HHMMSS.log`’.

‘`-vstats_file` *`file`*’

Dump video coding statistics to *file*.

‘`-top[:`*`stream_specifier`*`]` *`n`* `(`*`output,per-stream`*`)`’

top=1/bottom=0/auto=-1 field first

‘`-dc` *`precision`*’

Intra_dc_precision.

‘`-vtag` *`fourcc/tag`* `(`*`output`*`)`’

Force video tag/fourcc. This is an alias for `-tag:v`.

‘`-qphist (`*`global`*`)`’

Show QP histogram

‘`-vbsf` *`bitstream_filter`*’

Deprecated see -bsf

‘`-force_key_frames[:`*`stream_specifier`*`]` *`time`*`[,`*`time`*`...]`
`(`*`output,per-stream`*`)`’

‘-force_key_frames[:*stream_specifier*] expr:*expr* (*output,per-stream*)’

Force key frames at the specified timestamps, more precisely at the first frames after each specified time.

If the argument is prefixed with `expr:`, the string *expr* is interpreted like an expression and is evaluated for each frame. A key frame is forced in case the evaluation is non-zero.

If one of the times is "`chapters`[*delta*]", it is expanded into the time of the beginning of all chapters in the file, shifted by *delta*, expressed as a time in seconds. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file.

For example, to insert a key frame at 5 minutes, plus key frames 0.1 second before the beginning of every chapter:

```
-force_key_frames 0:05:00,chapters-0.1
```

The expression in *expr* can contain the following constants:

‘`n`’

the number of current processed frame, starting from 0

‘`n_forced`’

the number of forced frames

‘`prev_forced_n`’

the number of the previous forced frame, it is `NAN` when no keyframe was forced yet

‘`prev_forced_t`’

the time of the previous forced frame, it is `NAN` when no keyframe was forced yet

‘`t`’

the time of the current processed frame

For example to force a key frame every 5 seconds, you can specify:

```
-force_key_frames expr:gte(t,n_forced*5)
```

To force a key frame 5 seconds after the time of the last forced one, starting from second 13:

```
-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
```

Note that forcing too many keyframes is very harmful for the lookahead algorithms of certain encoders: using fixed-GOP options or similar would be more efficient.

‘-copyinkf[:*stream_specifier*] (*output,per-stream*)’

When doing stream copy, copy also non-key frames found at the beginning.

5.7 Audio Options

‘-aframes *number* (*output*)’

Set the number of audio frames to record. This is an alias for -frames:a.

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

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

‘-aq *q* (*output*)’

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

‘-ac[:*stream_specifier*] *channels* (*input/output,per-stream*)’

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

‘-an (*output*)’

Disable audio recording.

‘-acodec *codec* (*input/output*)’

Set the audio codec. This is an alias for -codec:a.

‘-sample_fmt[:*stream_specifier*] *sample_fmt* (*output,per-stream*)’

Set the audio sample format. Use -sample_fmts to get a list of supported sample formats.

‘-af *filtergraph* (*output*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for -filter:a, see the -filter option.

5.8 Advanced Audio options:

‘-atag *fourcc/tag* (*output*)’

Force audio tag/fourcc. This is an alias for `-tag:a`.

‘-absf *bitstream_filter*’

Deprecated, see -bsf

‘-guess_layout_max *channels* (*input,per-stream*)’

If some input channel layout is not known, try to guess only if it corresponds to at most the specified number of channels. For example, 2 tells to `ffmpeg` to recognize 1 channel as mono and 2 channels as stereo but not 6 channels as 5.1. The default is to always try to guess. Use 0 to disable all guessing.

5.9 Subtitle options:

‘-slang *code*’

Set the ISO 639 language code (3 letters) of the current subtitle stream.

‘-scodec *codec* (*input/output*)’

Set the subtitle codec. This is an alias for `-codec:s`.

‘-sn (*output*)’

Disable subtitle recording.

‘-sbsf *bitstream_filter*’

Deprecated, see -bsf

5.10 Advanced Subtitle options:

‘-fix_sub_duration’

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

‘-canvas_size *size*’

Set the size of the canvas used to render subtitles.

5.11 Advanced options

‘-map [-]*input_file_id*[:*stream_specifier*][,*sync_file_id*[:*stream_specifier*]] | *[linklabel]* (*output*)’

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A `-` character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the ‘`-filter_complex`’ option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in ‘`INPUT`’ identified by "0:1" to the (single) output stream in ‘`out.wav`’.

For example, to select the stream with index 2 from input file ‘`a.mov`’ (specified by the identifier "0:2"), and stream with index 6 from input ‘`b.mov`’ (specified by the identifier "1:6"), and copy them to the output file ‘`out.mov`’:

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

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

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

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

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

‘`-map_channel` [*input_file_id*`.`*stream_specifier*`.`*channel_id*`|-1][:`*output_file_id*`.`*stream_specifier*`]’`

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don’t match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can't for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here '`input.mkv`') with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

'`-map_metadata[:`*`metadata_spec_out`*`] `*`infile`*`[:`*`metadata_spec_in`*`] (`*`output,per-metadata`*`)`'

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

'*`g`*'

global metadata, i.e. metadata that applies to the whole file

'*`s`*`[:`*`stream_spec`*`]`'

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

'*`c`*`:`*`chapter_index`*'

per-chapter metadata. *chapter_index* is the zero-based chapter index.

'*`p`*`:`*`program_index`*'

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

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

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

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

‘`-map_chapters` *`input_file_index`* (*`output`*)’

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

‘`-benchmark` (*`global`*)’

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

‘`-benchmark_all` (*`global`*)’

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

‘`-timelimit` *`duration`* (*`global`*)’

Exit after ffmpeg has been running for *duration* seconds.

‘`-dump` (*`global`*)’

Dump each input packet to stderr.

‘`-hex` (*`global`*)’

When dumping packets, also dump the payload.

‘`-re` (*`input`*)’

Read input at native frame rate. Mainly used to simulate a grab device. By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming). If your input(s) is coming from some other live streaming source (through HTTP or UDP for example) the server might already be in real-time, thus the option will likely not be required. On the other hand, this is meaningful if your input(s) is a file you are trying to push in real-time.

‘`-loop_input`’

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

‘`-loop_output` *`number_of_times`*’

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

‘`-vsync` *`parameter`*’

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

‘`0, passthrough`’

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

‘`1, cfr`’

Frames will be duplicated and dropped to achieve exactly the requested constant frame rate.

‘`2, vfr`’

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

‘`drop`’

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

‘`-1, auto`’

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option ‘`avoid_negative_ts`’ is enabled.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

‘`-async` *`samples_per_second`*’

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option ‘`avoid_negative_ts`’ is enabled.

This option has been deprecated. Use the `aresample` audio filter instead.

‘`-copyts`’

Do not process input timestamps, but keep their values without trying to sanitize them. In particular, do not remove the initial start time offset value.

Note that, depending on the ‘`vsync`’ option or on specific muxer processing (e.g. in case the format option ‘`avoid_negative_ts`’ is enabled) the output timestamps may mismatch with the input timestamps even when this option is selected.

‘`-copytb` *mode*’

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

‘`1`’

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid non monotonically increasing timestamps when copying video streams with variable frame rate.

‘`0`’

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

‘`-1`’

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

‘`-shortest (`*output*`)`’

Finish encoding when the shortest input stream ends.

‘`-dts_delta_threshold`’

Timestamp discontinuity delta threshold.

‘`-muxdelay` *seconds* `(`*input*`)`’

Set the maximum demux-decode delay.

‘`-muxpreload` *`seconds`* (*`input`*)’

Set the initial demux-decode delay.

‘`-streamid` *`output-stream-index`*`:`*`new-value`* (*`output`*)’

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

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

‘`-bsf[:`*`stream_specifier`*`]` *`bitstream_filters`* (*`output,per-stream`*)’

Set bitstream filters for matching streams. *bitstream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

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

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

‘`-tag[:`*`stream_specifier`*`]` *`codec_tag`* (*`per-stream`*)’

Force a tag/fourcc for matching streams.

‘`-timecode` *`hh`*`:`*`mm`*`:`*`ss`*`SEP`*`ff`*’

Specify Timecode for writing. *SEP* is ’:’ for non drop timecode and ’;’ (or ’.’) for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

‘`-filter_complex` *`filtergraph`* (*`global`*)’

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the ‘`-filter`’ options. *filtergraph* is a description of the filtergraph, as described in the “Filtergraph syntax” section of the ffmpeg-filters manual.

Input link labels must refer to input streams using the `[file_index:stream_specifier]` syntax (i.e. the same as ‘`-map`’ uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with ‘`-map`’. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map
'[out]' out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map
'[out]' out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

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

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

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

‘`-lavfi` *`filtergraph`* (*`global`*)’

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. Equivalent to ‘`-filter_complex`’.

‘`-filter_complex_script` *`filename`* (*`global`*)’

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

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720x576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

5.12 Preset files

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

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

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

First ffmpeg searches for a file named *arg*.ffpreset in the directories '`$FFMPEG_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the datadir defined at configuration time (usually '`PREFIX/share/ffmpeg`') or in a '`ffpresets`' folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file '`libvpx-1080p.ffpreset`'.

If no such file is found, then ffmpeg will search for a file named *codec_name-arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file '`libvpx-1080p.ffpreset`'.

6. Tips

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

  ```
  ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
  ```

- The parameter 'q' which is displayed while encoding is the current quantizer. The value 1 indicates that a very good quality could be achieved. The value 31 indicates the worst quality. If q=31 appears too often, it means that the encoder cannot compress enough to meet your bitrate. You must either increase the bitrate, decrease the frame rate or decrease the frame size.
- If your computer is not fast enough, you can speed up the compression at the expense of the

compression ratio. You can use '-me zero' to speed up motion estimation, and '-g 0' to disable motion estimation completely (you have only I-frames, which means it is about as good as JPEG compression).

- To have very low audio bitrates, reduce the sampling frequency (down to 22050 Hz for MPEG audio, 22050 or 11025 for AC-3).
- To have a constant quality (but a variable bitrate), use the option '-qscale n' when 'n' is between 1 (excellent quality) and 31 (worst quality).

7. Examples

7.1 Preset files

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

Preset files are specified with the `pre` option, this option takes a preset name as input. FFmpeg searches for a file named *preset_name*.avpreset in the directories '`$AVCONV_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the data directory defined at configuration time (usually '`$PREFIX/share/ffmpeg`') in that order. For example, if the argument is `libx264-max`, it will search for the file '`libx264-max.avpreset`'.

7.2 Video and Audio grabbing

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

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

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing

Grab the X11 display with ffmpeg via

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

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

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

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

7.4 Video and Audio file format conversion

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

Examples:

- You can use YUV files as input:

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

  It will use the files:

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

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

- You can input from a raw YUV420P file:

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

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

- You can output to a raw YUV420P file:

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

- You can set several input files and output files:

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

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

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

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

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

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

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

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

- You can transcode decrypted VOBs:

```
ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
```

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

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

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

For extracting images from a video:

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

This will extract one video frame per second from the video and will output them in files named '`foo-001.jpeg`', '`foo-002.jpeg`', etc. Images will be rescaled to fit the new WxH values.

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

For creating a video from many images:

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

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

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -i 'foo-*.jpeg' -r 12 -s WxH foo.avi
```

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

```
ffmpeg -i test1.avi -i test2.avi -map 0:3 -map 0:2 -map 0:1 -map 0:0 -c copy test12.nut
```

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

- To force CBR video output:

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

- The four options lmin, lmax, mblmin and mblmax use 'lambda' units, but you may use the QP2LAMBDA constant to easily convert from 'q' units:

```
ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
```

8. Syntax

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

8.1 Quoting and escaping

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

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

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

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

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

8.1.1 Examples

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

```
Crime d\'Amour
```

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

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

- Include leading or trailing whitespaces using quoting:

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

- Escaping and quoting can be mixed together:

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

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

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

8.2 Date

The accepted syntax is:

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

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

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

8.3 Time duration

The accepted syntax is:

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

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

8.4 Video size

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

The following abbreviations are recognized:

‘`ntsc`’

720x480

‘`pal`’

720x576

‘`qntsc`’

352x240

‘`qpal`’

352x288

‘`sntsc`’

640x480

‘`spal`’

768x576

‘`film`’

352x240

‘`ntsc-film`’

352x240

‘sqcif’

128x96

‘qcif’

176x144

‘cif’

352x288

‘4cif’

704x576

‘16cif’

1408x1152

‘qqvga’

160x120

‘qvga’

320x240

‘vga’

640x480

‘svga’

800x600

‘xga’

1024x768

‘uxga’

1600x1200

‘qxga’

2048x1536

‘sxga’

1280x1024

‘qsxga’

2560x2048

‘hsxga’

5120x4096

‘wvga’

852x480

‘wxga’

1366x768

‘wsxga’

1600x1024

‘wuxga’

1920x1200

‘woxga’

2560x1600

‘wqsxga’

3200x2048

‘wquxga’

3840x2400

‘whsxga’

6400x4096

‘whuxga’

7680x4800

‘cga’

320x200

‘ega’

640x350

‘hd480’

852x480

‘hd720’

1280x720

‘hd1080’

1920x1080

‘2k’

2048x1080

‘2kflat’

1998x1080

‘2kscope’

2048x858

‘4k’

4096x2160

‘4kflat’

3996x2160

‘4kscope’

4096x1716

8.5 Video rate

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

The following abbreviations are recognized:

‘`ntsc`’

30000/1001

‘`pal`’

25/1

‘`qntsc`’

30000/1001

‘`qpal`’

25/1

‘`sntsc`’

30000/1001

‘`spal`’

25/1

‘`film`’

24/1

‘`ntsc-film`’

24000/1001

8.6 Ratio

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

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

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

8.7 Color

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

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

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

9. Expression Evaluation

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

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

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

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

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

The following functions are available:

‘`abs(x)`’

Compute absolute value of *x*.

‘`acos(x)`’

Compute arccosine of *x*.

‘`asin(x)`’

Compute arcsine of *x*.

‘`atan(x)`’

Compute arctangent of *x*.

‘`between(x, min, max)`’

Return 1 if *x* is greater than or equal to *min* and lesser than or equal to *max*, 0 otherwise.

‘`bitand(x, y)`’
‘`bitor(x, y)`’

Compute bitwise and/or operation on *x* and *y*.

The results of the evaluation of *x* and *y* are converted to integers before executing the bitwise operation.

Note that both the conversion to integer and the conversion back to floating point can lose precision. Beware of unexpected results for large numbers (usually 2^53 and larger).

‘`ceil(expr)`’

Round the value of expression *expr* upwards to the nearest integer. For example, "ceil(1.5)" is "2.0".

‘`cos(x)`’

Compute cosine of *x*.

‘`cosh(x)`’

Compute hyperbolic cosine of *x*.

‘`eq(x, y)`’

Return 1 if *x* and *y* are equivalent, 0 otherwise.

‘`exp(x)`’

Compute exponential of *x* (with base `e`, the Euler’s number).

‘`floor(expr)`’

Round the value of expression *expr* downwards to the nearest integer. For example, "floor(-1.5)" is "-2.0".

‘`gauss(x)`’

Compute Gauss function of *x*, corresponding to `exp(-x*x/2) / sqrt(2*PI)`.

‘`gcd(x, y)`’

Return the greatest common divisor of *x* and *y*. If both *x* and *y* are 0 or either or both are less than zero then behavior is undefined.

‘`gt(x, y)`’

Return 1 if *x* is greater than *y*, 0 otherwise.

‘`gte(x, y)`’

Return 1 if *x* is greater than or equal to *y*, 0 otherwise.

‘`hypot(x, y)`’

This function is similar to the C function with the same name; it returns "sqrt(*x***x* + *y***y*)", the length of the hypotenuse of a right triangle with sides of length *x* and *y*, or the distance of the point (*x*, *y*) from the origin.

‘`if(x, y)`’

Evaluate *x*, and if the result is non-zero return the result of the evaluation of *y*, return 0 otherwise.

‘`if(x, y, z)`’

Evaluate *x*, and if the result is non-zero return the evaluation result of *y*, otherwise the evaluation result of *z*.

‘`ifnot(x, y)`’

Evaluate *x*, and if the result is zero return the result of the evaluation of *y*, return 0 otherwise.

‘`ifnot(x, y, z)`’

Evaluate *x*, and if the result is zero return the evaluation result of *y*, otherwise the evaluation result of *z*.

‘`isinf(x)`’

Return 1.0 if *x* is +/-INFINITY, 0.0 otherwise.

‘`isnan(x)`’

Return 1.0 if *x* is NAN, 0.0 otherwise.

‘`ld(var)`’

Allow to load the value of the internal variable with number *var*, which was previously stored with st(*var*, *expr*). The function returns the loaded value.

‘`log(x)`’

Compute natural logarithm of *x*.

‘`lt(x, y)`’

Return 1 if *x* is lesser than *y*, 0 otherwise.

‘`lte(x, y)`’

Return 1 if *x* is lesser than or equal to *y*, 0 otherwise.

‘`max(x, y)`’

Return the maximum between *x* and *y*.

‘`min(x, y)`’

Return the maximum between *x* and *y*.

‘`mod(x, y)`’

Compute the remainder of division of *x* by *y*.

‘`not(expr)`’

Return 1.0 if *expr* is zero, 0.0 otherwise.

‘`pow(x, y)`’

Compute the power of *x* elevated *y*, it is equivalent to "(*x*)^(*y*)".

‘`print(t)`’
‘`print(t, l)`’

Print the value of expression *t* with loglevel *l*. If *l* is not specified then a default log level is used. Returns the value of the expression printed.

Prints t with loglevel l

‘`random(x)`’

Return a pseudo random value between 0.0 and 1.0. *x* is the index of the internal variable which will be used to save the seed/state.

‘`root(expr, max)`’

Find an input value for which the function represented by *expr* with argument *ld(0)* is 0 in the interval 0..*max*.

The expression in *expr* must denote a continuous function or the result is undefined.

ld(0) is used to represent the function input value, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(0)`. When the expression evaluates to 0 then the corresponding input value will be returned.

‘`sin(x)`’

Compute sine of *x*.

‘`sinh(x)`’

Compute hyperbolic sine of *x*.

‘`sqrt(expr)`’

Compute the square root of *expr*. This is equivalent to "(*expr*)^.5".

‘`squish(x)`’

Compute expression `1/(1 + exp(4*x))`.

‘`st(var, expr)`’

Allow to store the value of the expression *expr* in an internal variable. *var* specifies the number of the variable where to store the value, and it is a value ranging from 0 to 9. The function returns the value stored in the internal variable. Note, Variables are currently not shared between expressions.

‘`tan(x)`’

Compute tangent of *x*.

‘`tanh(x)`’

Compute hyperbolic tangent of *x*.

‘`taylor(expr, x)`’
‘`taylor(expr, x, id)`’

Evaluate a Taylor series at *x*, given an expression representing the `ld(id)`-th derivative of a function at 0.

When the series does not converge the result is undefined.

ld(id) is used to represent the derivative order in *expr*, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(id)`. If *id* is not specified then 0 is assumed.

Note, when you have the derivatives at y instead of 0, `taylor(expr, x-y)` can be used.

‘`time(0)`’

Return the current (wallclock) time in seconds.

‘`trunc(expr)`’

Round the value of expression *expr* towards zero to the nearest integer. For example, "trunc(-1.5)" is "-1.0".

‘`while(cond, expr)`’

Evaluate expression *expr* while the expression *cond* is non-zero, and returns the value of the last *expr* evaluation, or NAN if *cond* was always false.

The following constants are available:

‘`PI`’

area of the unit disc, approximately 3.14

‘`E`’

exp(1) (Euler’s number), approximately 2.718

‘`PHI`’

golden ratio (1+sqrt(5))/2, approximately 1.618

Assuming that an expression is considered "true" if it has a non-zero value, note that:

`*` works like AND

`+` works like OR

For example the construct:

```
if (A AND B) then C
```

is equivalent to:

```
if(A*B, C)
```

In your C code, you can extend the list of unary and binary functions, and define recognized constants, so that they are available for your expressions.

The evaluator also recognizes the International System unit prefixes. 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.

The list of available International System prefixes follows, with indication of the corresponding powers of 10 and of 2.

‘y’

10^-24 / 2^-80

‘z’

10^-21 / 2^-70

‘a’

10^-18 / 2^-60

‘f’

10^-15 / 2^-50

‘p’

10^-12 / 2^-40

‘n’

10^-9 / 2^-30

‘u’

10^-6 / 2^-20

‘m’

10^-3 / 2^-10

‘c’

10^-2

‘d’

10^-1

‘h’

10^2

‘k’

10^3 / 2^10

‘K’

10^3 / 2^10

‘M’

10^6 / 2^20

‘G’

10^9 / 2^30

‘T’

10^12 / 2^40

‘P’

10^15 / 2^40

‘E’

10^18 / 2^50

‘Z’

10^21 / 2^60

‘Y’

10^24 / 2^70

10. OpenCL Options

When FFmpeg is configured with `--enable-opencl`, it is possible to set the options for the global OpenCL context.

The list of supported options follows:

‘`build_options`’

Set build options used to compile the registered kernels.

See reference "OpenCL Specification Version: 1.2 chapter 5.6.4".

‘platform_idx’

Select the index of the platform to run OpenCL code.

The specified index must be one of the indexes in the device list which can be obtained with `av_opencl_get_device_list()`.

‘device_idx’

Select the index of the device used to run OpenCL code.

The specifed index must be one of the indexes in the device list which can be obtained with `av_opencl_get_device_list()`.

11. Codec Options

libavcodec provides some generic global options, which can be set on all the encoders and decoders. In addition each codec may support so-called private options, which are specific for a given codec.

Sometimes, a global option may only affect a specific kind of codec, and may be unsensical or ignored by another, so you need to be aware of the meaning of the specified options. Also some options are meant only for decoding or encoding.

Options may be set by specifying *-option value* in the FFmpeg tools, or by setting the value explicitly in the `AVCodecContext` options or using the ‘`libavutil/opt.h`’ API for programmatic use.

The list of supported options follow:

‘b *integer* (*encoding,audio,video*)’

Set bitrate in bits/s. Default value is 200K.

‘ab *integer* (*encoding,audio*)’

Set audio bitrate (in bits/s). Default value is 128K.

‘bt *integer* (*encoding,video*)’

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

‘flags *flags* (*decoding/encoding,audio,video,subtitles*)’

Set generic flags.

Possible values:

‘mv4’

Use four motion vector by macroblock (mpeg4).

‘qpel’

Use 1/4 pel motion compensation.

‘loop’

Use loop filter.

‘qscale’

Use fixed qscale.

‘gmc’

Use gmc.

‘mv0’

Always try a mb with mv=<0,0>.

‘input_preserved’
‘pass1’

Use internal 2pass ratecontrol in first pass mode.

‘pass2’

Use internal 2pass ratecontrol in second pass mode.

‘gray’

Only decode/encode grayscale.

‘emu_edge’

Do not draw edges.

‘psnr’

Set error[?] variables during encoding.

‘truncated’

‘`naq`’

Normalize adaptive quantization.

‘`ildct`’

Use interlaced DCT.

‘`low_delay`’

Force low delay.

‘`global_header`’

Place global headers in extradata instead of every keyframe.

‘`bitexact`’

Use only bitexact stuff (except (I)DCT).

‘`aic`’

Apply H263 advanced intra coding / mpeg4 ac prediction.

‘`cbp`’

Deprecated, use mpegvideo private options instead.

‘`qprd`’

Deprecated, use mpegvideo private options instead.

‘`ilme`’

Apply interlaced motion estimation.

‘`cgop`’

Use closed gop.

‘`sub_id` *`integer`*’

Deprecated, currently unused.

‘`me_method` *`integer`* (*`encoding,video`*)’

Set motion estimation method.

Possible values:

‘zero’

zero motion estimation (fastest)

‘full’

full motion estimation (slowest)

‘epzs’

EPZS motion estimation (default)

‘esa’

esa motion estimation (alias for full)

‘tesa’

tesa motion estimation

‘dia’

dia motion estimation (alias for epzs)

‘log’

log motion estimation

‘phods’

phods motion estimation

‘x1’

X1 motion estimation

‘hex’

hex motion estimation

‘umh’

umh motion estimation

‘iter’

iter motion estimation

‘extradata_size *integer*’

Set extradata size.

‘time_base *rational number*’

Set codec time base.

It is the fundamental unit of time (in seconds) in terms of which frame timestamps are represented. For fixed-fps content, timebase should be `1 / frame_rate` and timestamp increments should be identically 1.

‘g *integer* (*encoding,video*)’

Set the group of picture size. Default value is 12.

‘ar *integer* (*decoding/encoding,audio*)’

Set audio sampling rate (in Hz).

‘ac *integer* (*decoding/encoding,audio*)’

Set number of audio channels.

‘cutoff *integer* (*encoding,audio*)’

Set cutoff bandwidth.

‘frame_size *integer* (*encoding,audio*)’

Set audio frame size.

Each submitted frame except the last must contain exactly frame_size samples per channel. May be 0 when the codec has CODEC_CAP_VARIABLE_FRAME_SIZE set, in that case the frame size is not restricted. It is set by some decoders to indicate constant frame size.

‘frame_number *integer*’

Set the frame number.

‘delay *integer*’
‘qcomp *float* (*encoding,video*)’

Set video quantizer scale compression (VBR). It is used as a constant in the ratecontrol equation. Recommended range for default rc_eq: 0.0-1.0.

‘qblur *float* (*encoding,video*)’

Set video quantizer scale blur (VBR).

‘qmin *integer* (*encoding,video*)’

Set min video quantizer scale (VBR). Must be included between -1 and 69, default value is 2.

‘qmax *integer* (*encoding,video*)’

Set max video quantizer scale (VBR). Must be included between -1 and 1024, default value is 31.

‘qdiff *integer* (*encoding,video*)’

Set max difference between the quantizer scale (VBR).

‘bf *integer* (*encoding,video*)’

Set max number of B frames.

‘b_qfactor *float* (*encoding,video*)’

Set qp factor between P and B frames.

‘rc_strategy *integer* (*encoding,video*)’

Set ratecontrol method.

‘b_strategy *integer* (*encoding,video*)’

Set strategy to choose between I/P/B-frames.

‘ps *integer* (*encoding,video*)’

Set RTP payload size in bytes.

‘mv_bits *integer*’
‘header_bits *integer*’
‘i_tex_bits *integer*’
‘p_tex_bits *integer*’
‘i_count *integer*’
‘p_count *integer*’
‘skip_count *integer*’
‘misc_bits *integer*’
‘frame_bits *integer*’
‘codec_tag *integer*’
‘bug *flags* (*decoding,video*)’

Workaround not auto detected encoder bugs.

Possible values:

‘autodetect’
‘old_msmpeg4’

some old lavc generated msmpeg4v3 files (no autodetection)

‘xvid_ilace’

Xvid interlacing bug (autodetected if fourcc==XVIX)

‘ump4’

(autodetected if fourcc==UMP4)

‘no_padding’

padding bug (autodetected)

‘amv’
‘ac_vlc’

illegal vlc bug (autodetected per fourcc)

‘qpel_chroma’
‘std_qpel’

old standard qpel (autodetected per fourcc/version)

‘qpel_chroma2’
‘direct_blocksize’

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

‘edge’

edge padding bug (autodetected per fourcc/version)

‘hpel_chroma’
‘dc_clip’
‘ms’

Workaround various bugs in microsoft broken decoders.

‘trunc’

trancated frames

‘lelim *integer* (*encoding,video*)’

Set single coefficient elimination threshold for luminance (negative values also consider DC coefficient).

‘celim *integer* (*encoding,video*)’

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

‘strict *integer* (*decoding/encoding,audio,video*)’

Specify how strictly to follow the standards.

Possible values:

‘very’

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

‘strict’

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

‘normal’
‘unofficial’

allow unofficial extensions

‘experimental’

allow non standardized experimental things

‘b_qoffset *float* (*encoding,video*)’

Set QP offset between P and B frames.

‘err_detect *flags* (*decoding,audio,video*)’

Set error detection flags.

Possible values:

‘crccheck’

verify embedded CRCs

‘bitstream’

detect bitstream specification deviations

‘buffer’

detect improper bitstream length

‘explode’

abort decoding on minor error detection

‘careful’

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

‘compliant’

consider all spec non compliancies as errors

‘aggressive’

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

‘has_b_frames *integer*’
‘block_align *integer*’
‘mpeg_quant *integer* (*encoding,video*)’

Use MPEG quantizers instead of H.263.

‘qsquish *float* (*encoding,video*)’

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

‘rc_qmod_amp *float* (*encoding,video*)’

Set experimental quantizer modulation.

‘rc_qmod_freq *integer* (*encoding,video*)’

Set experimental quantizer modulation.

‘rc_override_count *integer*’
‘rc_eq *string* (*encoding,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` *`integer`* (*`encoding,audio,video`*)’

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

‘`minrate` *`integer`* (*`encoding,audio,video`*)’

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

‘`bufsize` *`integer`* (*`encoding,audio,video`*)’

Set ratecontrol buffer size (in bits).

‘`rc_buf_aggressivity` *`float`* (*`encoding,video`*)’

Currently useless.

‘`i_qfactor` *`float`* (*`encoding,video`*)’

Set QP factor between P and I frames.

‘`i_qoffset` *`float`* (*`encoding,video`*)’

Set QP offset between P and I frames.

‘`rc_init_cplx` *`float`* (*`encoding,video`*)’

Set initial complexity for 1-pass encoding.

‘`dct` *`integer`* (*`encoding,video`*)’

Set 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 *float* (*encoding*,*video*)’

Compress bright areas stronger than medium ones.

‘tcplx_mask *float* (*encoding*,*video*)’

Set temporal complexity masking.

‘scplx_mask *float* (*encoding*,*video*)’

Set spatial complexity masking.

‘p_mask *float* (*encoding*,*video*)’

Set inter masking.

‘dark_mask *float* (*encoding*,*video*)’

Compress dark areas stronger than medium ones.

‘idct *integer* (*decoding/encoding*,*video*)’

Select IDCT implementation.

Possible values:

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

‘ipp’
‘xvidmmx’
‘faani’

floating point AAN IDCT

‘slice_count *integer*’
‘ec *flags* (*decoding,video*)’

Set error concealment strategy.

Possible values:

‘guess_mvs’

iterative motion vector (MV) search (slow)

‘deblock’

use strong deblock filter for damaged MBs

‘bits_per_coded_sample *integer*’
‘pred *integer* (*encoding,video*)’

Set prediction method.

Possible values:

‘left’
‘plane’
‘median’
‘aspect *rational number* (*encoding,video*)’

Set sample aspect ratio.

‘debug *flags* (*decoding/encoding,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` *`integer`* (*`decoding`*,*`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` *`integer`* (*`encoding`*`,`*`video`*`)`’

Set full pel me compare function.

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’
‘subcmp *integer* (*encoding,video*)’

Set sub pel me compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’

‘chroma’

‘mbcmp *integer* (*encoding,video*)’

Set macroblock compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’

‘`ildctcmp` *`integer`* (*`encoding,video`*)’

Set interlaced dct compare function.

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’

‘dia_size *integer* (*encoding,video*)’

Set diamond type & size for motion estimation.

‘last_pred *integer* (*encoding,video*)’

Set amount of motion predictors from the previous frame.

‘preme *integer* (*encoding,video*)’

Set pre motion estimation.

‘precmp *integer* (*encoding,video*)’

Set pre motion estimation compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’

‘chroma’

‘pre_dia_size *integer* (*encoding,video*)’

Set diamond type & size for motion estimation pre-pass.

‘subq *integer* (*encoding,video*)’

Set sub pel motion estimation quality.

‘dtg_active_format *integer*’

‘me_range *integer* (*encoding,video*)’

Set limit motion vectors range (1023 for DivX player).

‘ibias *integer* (*encoding,video*)’

Set intra quant bias.

‘pbias *integer* (*encoding,video*)’

Set inter quant bias.

‘color_table_id *integer*’
‘global_quality *integer* (*encoding,audio,video*)’
‘coder *integer* (*encoding,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 *integer* (*encoding,video*)’

Set context model.

‘slice_flags *integer*’
‘xvmc_acceleration *integer*’
‘mbd *integer* (*encoding,video*)’

Set macroblock decision algorithm (high quality mode).

Possible values:

‘simple’

use mbcmp (default)

‘bits’

use fewest bits

‘rd’

use best rate distortion

‘stream_codec_tag *integer*’
‘sc_threshold *integer* (*encoding,video*)’

Set scene change threshold.

‘lmin *integer* (*encoding,video*)’

Set min lagrange factor (VBR).

‘lmax *integer* (*encoding,video*)’

Set max lagrange factor (VBR).

‘nr *integer* (*encoding,video*)’

Set noise reduction.

‘rc_init_occupancy *integer* (*encoding,video*)’

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

‘inter_threshold *integer* (*encoding,video*)’
‘flags2 *flags* (*decoding/encoding,audio,video*)’

Possible values:

‘fast’

allow non spec compliant speedup tricks

‘sgop’

Deprecated, use mpegvideo private options instead

‘noout’

skip bitstream encoding

‘local_header’

place global headers at every keyframe instead of in extradata

‘chunks’

Frame data might be split into multiple chunks

‘showall’

Show all frames before the first keyframe

‘skiprd’

Deprecated, use mpegvideo private options instead

‘error *integer* (*encoding,video*)’
‘qns *integer* (*encoding,video*)’

Deprecated, use mpegvideo private options instead.

‘threads *integer* (*decoding/encoding,video*)’

Possible values:

‘auto’

detect a good number of threads

‘me_threshold *integer* (*encoding,video*)’

Set motion estimation threshold.

‘mb_threshold *integer* (*encoding,video*)’

Set macroblock threshold.

‘dc *integer* (*encoding,video*)’

Set intra_dc_precision.

‘nssew *integer* (*encoding,video*)’

Set nsse weight.

‘skip_top *integer* (*decoding,video*)’

Set number of macroblock rows at the top which are skipped.

‘skip_bottom *integer* (*decoding,video*)’

Set number of macroblock rows at the bottom which are skipped.

‘profile *integer* (*encoding,audio,video*)’

Possible values:

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

‘level *integer* (*encoding,audio,video*)’

Possible values:

‘unknown’

‘lowres *integer* (*decoding,audio,video*)’

Decode at 1= 1/2, 2=1/4, 3=1/8 resolutions.

‘skip_threshold *integer* (*encoding,video*)’

Set frame skip threshold.

‘skip_factor *integer* (*encoding,video*)’

Set frame skip factor.

‘skip_exp *integer* (*encoding,video*)’

Set frame skip exponent.

‘skipcmp *integer* (*encoding,video*)’

Set frame skip compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’
‘border_mask *float* (*encoding,video*)’

Increase the quantizer for macroblocks close to borders.

‘mblmin *integer* (*encoding,video*)’

Set min macroblock lagrange factor (VBR).

‘mblmax *integer* (*encoding,video*)’

Set max macroblock lagrange factor (VBR).

‘mepc *integer* (*encoding,video*)’

Set motion estimation bitrate penalty compensation (1.0 = 256).

‘skip_loop_filter *integer* (*decoding,video*)’
‘skip_idct *integer* (*decoding,video*)’
‘skip_frame *integer* (*decoding,video*)’

Make decoder discard processing depending on the frame type selected by the option value.

‘skip_loop_filter’ skips frame loop filtering, ‘skip_idct’ skips frame IDCT/dequantization, ‘skip_frame’ skips decoding.

Possible values:

‘none’

Discard no frame.

‘default’

Discard useless frames like 0-sized frames.

‘noref’

Discard all non-reference frames.

‘bidir’

Discard all bidirectional frames.

‘`nokey`’

Discard all frames excepts keyframes.

‘`all`’

Discard all frames.

Default value is ‘`default`’.

‘`bidir_refine` *`integer`* (*`encoding,video`*)’

Refine the two motion vectors used in bidirectional macroblocks.

‘`brd_scale` *`integer`* (*`encoding,video`*)’

Downscale frames for dynamic B-frame decision.

‘`keyint_min` *`integer`* (*`encoding,video`*)’

Set minimum interval between IDR-frames.

‘`refs` *`integer`* (*`encoding,video`*)’

Set reference frames to consider for motion compensation.

‘`chromaoffset` *`integer`* (*`encoding,video`*)’

Set chroma qp offset from luma.

‘`trellis` *`integer`* (*`encoding,audio,video`*)’

Set rate-distortion optimal quantization.

‘`sc_factor` *`integer`* (*`encoding,video`*)’

Set value multiplied by qscale for each frame and added to scene_change_score.

‘`mv0_threshold` *`integer`* (*`encoding,video`*)’
‘`b_sensitivity` *`integer`* (*`encoding,video`*)’

Adjust sensitivity of b_frame_strategy 1.

‘`compression_level` *`integer`* (*`encoding,audio,video`*)’
‘`min_prediction_order` *`integer`* (*`encoding,audio`*)’
‘`max_prediction_order` *`integer`* (*`encoding,audio`*)’

‘timecode_frame_start *integer* (*encoding,video*)’

Set GOP timecode frame start number, in non drop frame format.

‘request_channels *integer* (*decoding,audio*)’

Set desired number of audio channels.

‘bits_per_raw_sample *integer*’
‘channel_layout *integer* (*decoding/encoding,audio*)’

Possible values:

‘request_channel_layout *integer* (*decoding,audio*)’

Possible values:

‘rc_max_vbv_use *float* (*encoding,video*)’
‘rc_min_vbv_use *float* (*encoding,video*)’
‘ticks_per_frame *integer* (*decoding/encoding,audio,video*)’
‘color_primaries *integer* (*decoding/encoding,video*)’
‘color_trc *integer* (*decoding/encoding,video*)’
‘colorspace *integer* (*decoding/encoding,video*)’
‘color_range *integer* (*decoding/encoding,video*)’
‘chroma_sample_location *integer* (*decoding/encoding,video*)’
‘log_level_offset *integer*’

Set the log level offset.

‘slices *integer* (*encoding,video*)’

Number of slices, used in parallelized encoding.

‘thread_type *flags* (*decoding/encoding,video*)’

Select multithreading type.

Possible values:

‘slice’
‘frame’
‘audio_service_type *integer* (*encoding,audio*)’

Set 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 *sample_fmt* (*decoding,audio*)’

Set sample format audio decoders should prefer. Default value is none.

‘pkt_timebase *rational number*’
‘sub_charenc *encoding* (*decoding,subtitles*)’

Set the input subtitles character encoding.

12. Decoders

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

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

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

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

13. Video Decoders

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

13.1 rawvideo

Raw video decoder.

This decoder decodes rawvideo streams.

13.1.1 Options

‘`top` *`top_field_first`*’

Specify the assumed field type of the input video.

‘`-1`’

the video is assumed to be progressive (default)

‘`0`’

bottom-field-first is assumed

‘`1`’

top-field-first is assumed

14. Audio Decoders

14.1 ffwavesynth

Internal wave synthetizer.

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

15. Subtitles Decoders

15.1 dvdsub

This codec decodes the bitmap subtitles used in DVDs; the same subtitles can also be found in VobSub file pairs and in some Matroska files.

15.1.1 Options

‘`palette`’

Specify the global palette used by the bitmaps. When stored in VobSub, the palette is normally specified in the index file; in Matroska, the palette is stored in the codec extra-data in the same format as in VobSub. In DVDs, the palette is stored in the IFO file, and therefore not available when reading from dumped VOB files.

The format for this option is a string containing 16 24-bits hexadecimal numbers (without 0x prefix) separated by comas, for example `0d00ee, ee450d, 101010, eaeaea, 0ce60b, ec14ed, ebff0b, 0d617a, 7b7b7b, d1d1d1, 7b2a0e, 0d950c, 0f007b, cf0dec, cfa80c, 7c127b`.

16. Encoders

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

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

You can disable all the encoders with the configure option `--disable-encoders` and selectively enable / disable single encoders with the options `--enable-encoder=`*`ENCODER`* / `--disable-encoder=`*`ENCODER`*.

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

17. Audio Encoders

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

17.1 ac3 and ac3_fixed

AC-3 audio encoders.

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

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

17.1.1 AC-3 Metadata

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

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

- A/52:2010 - Digital Audio Compression (AC-3) (E-AC-3) Standard
- A/54 - Guide to the Use of the ATSC Digital Television Standard
- Dolby Metadata Guide
- Dolby Digital Professional Encoding Guidelines

17.1.1.1 Metadata Control Options

‘`-per_frame_metadata` *`boolean`*’

Allow Per-Frame Metadata. Specifies if the encoder should check for changing metadata for each frame.

‘`0`’

The metadata values set at initialization will be used for every frame in the stream. (default)

‘`1`’

Metadata values can be changed before encoding each frame.

17.1.1.2 Downmix Levels

‘`-center_mixlev` *`level`*’

Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo. This field will only be written to the bitstream if a center channel is present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6dB gain

‘`-surround_mixlev` *`level`*’

Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo. This field will only be written to the bitstream if one or more surround channels are present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.500`’

Apply -6dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

17.1.1.3 Audio Production Information

Audio Production Information is optional information describing the mixing environment. Either none or both of the fields are written to the bitstream.

‘`-mixing_level` *`number`*’

Mixing Level. Specifies peak sound pressure level (SPL) in the production environment when the mix was mastered. Valid values are 80 to 111, or -1 for unknown or not indicated. The default value is -1, but that value cannot be used if the Audio Production Information is written to the bitstream. Therefore, if the `room_type` option is not the default value, the `mixing_level` option must not be -1.

‘`-room_type` *`type`*’

Room Type. Describes the equalization used during the final mixing session at the studio or on the dubbing stage. A large room is a dubbing stage with the industry standard X-curve equalization; a small room has flat equalization. This field will not be written to the bitstream if both the `mixing_level` option and the `room_type` option have the default values.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`large`’

Large Room

‘`2`’
‘`small`’

Small Room

17.1.1.4 Other Metadata Options

‘`-copyright` *`boolean`*’

Copyright Indicator. Specifies whether a copyright exists for this audio.

‘`0`’
‘`off`’

No Copyright Exists (default)

‘`1`’
‘`on`’

Copyright Exists

‘`-dialnorm` *`value`*’

Dialogue Normalization. Indicates how far the average dialogue level of the program is below digital 100% full scale (0 dBFS). This parameter determines a level shift during audio reproduction that sets the average volume of the dialogue to a preset level. The goal is to match volume level between program sources. A value of -31dB will result in no volume level change, relative to the source volume, during audio reproduction. Valid values are whole numbers in the range -31 to -1, with -31 being the default.

‘`-dsur_mode` *`mode`*’

Dolby Surround Mode. Specifies whether the stereo signal uses Dolby Surround (Pro Logic). This field will only be written to the bitstream if the audio stream is stereo. Using this option does **NOT** mean the encoder will actually apply Dolby Surround processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`off`’

Not Dolby Surround Encoded

‘`2`’
‘`on`’

Dolby Surround Encoded

‘`-original` *`boolean`*’

Original Bit Stream Indicator. Specifies whether this audio is from the original source and not a copy.

‘`0`’
‘`off`’

Not Original Source

‘`1`’
‘`on`’

Original Source (default)

17.1.2 Extended Bitstream Information

The extended bitstream options are part of the Alternate Bit Stream Syntax as specified in Annex D of the A/52:2010 standard. It is grouped into 2 parts. If any one parameter in a group is specified, all values in that group will be written to the bitstream. Default values are used for those that are written but have not been specified. If the mixing levels are written, the decoder will use these values instead of the ones specified in the `center_mixlev` and `surround_mixlev` options if it supports the Alternate Bit Stream Syntax.

17.1.2.1 Extended Bitstream Information - Part 1

‘`-dmix_mode` *`mode`*’

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

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`ltrt`’

Lt/Rt Downmix Preferred

‘`2`’
‘`loro`’

Lo/Ro Downmix Preferred

‘`-ltrt_cmixlev` *`level`*’

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

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-ltrt_surmixlev` *`level`*’

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

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

‘`-loro_cmixlev` *`level`*’

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

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-loro_surmixlev` *`level`*’

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

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

17.1.2.2 Extended Bitstream Information - Part 2

‘`-dsurex_mode` *`mode`*’

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

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`on`’

Dolby Surround EX Off

‘`2`’
‘`off`’

Dolby Surround EX On

‘`-dheadphone_mode` *`mode`*’

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

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`on`’

Dolby Headphone Off

‘`2`’
‘`off`’

Dolby Headphone On

‘`-ad_conv_type` *`type`*’

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

‘0’
‘standard’

Standard A/D Converter (default)

‘1’
‘hdcd’

HDCD A/D Converter

17.1.3 Other AC-3 Encoding Options

‘-stereo_rematrixing *boolean*’

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

17.1.4 Floating-Point-Only AC-3 Encoding Options

These options are only valid for the floating-point encoder and do not exist for the fixed-point encoder due to the corresponding features not being implemented in fixed-point.

‘-channel_coupling *boolean*’

Enables/Disables use of channel coupling, which is an optional AC-3 feature that increases quality by combining high frequency information from multiple channels into a single channel. The per-channel high frequency information is sent with less accuracy in both the frequency and time domains. This allows more bits to be used for lower frequencies while preserving enough information to reconstruct the high frequencies. This option is enabled by default for the floating-point encoder and should generally be left as enabled except for testing purposes or to increase encoding speed.

‘-1’
‘auto’

Selected by Encoder (default)

‘0’
‘off’

Disable Channel Coupling

‘1’
‘on’

Enable Channel Coupling

‘-cpl_start_band *number*’

Coupling Start Band. Sets the channel coupling start band, from 1 to 15. If a value higher than the bandwidth is used, it will be reduced to 1 less than the coupling end band. If *auto* is used, the start band will be determined by the encoder based on the bit rate, sample rate, and channel layout. This option has no effect if channel coupling is disabled.

‘-1’
‘auto’

Selected by Encoder (default)

18. Video Encoders

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

18.1 libtheora

Theora format supported through libtheora.

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

18.1.1 Options

The following global options are mapped to internal libtheora options which affect the quality and the bitrate of the encoded stream.

‘b’

Set the video bitrate, only works if the `qscale` flag in ‘`flags`’ is not enabled.

‘`flags`’

Used to enable constant quality mode encoding through the ‘`qscale`’ flag, and to enable the `pass1` and `pass2` modes.

‘g’

Set the GOP size.

‘`global_quality`’

Set the global quality in lambda units, only works if the `qscale` flag in ‘`flags`’ is enabled. The value is clipped in the [0 - 10*`FF_QP2LAMBDA`] range, and then multiplied for 6.3 to get a value in the native libtheora range [0-63]. A higher value corresponds to a higher quality.

For example, to set maximum constant quality encoding with `ffmpeg`:

```
ffmpeg -i INPUT -flags:v qscale -global_quality:v "10*QP2LAMBDA" -codec:v libtheora OUTPUT.ogg
```

18.2 libvpx

VP8 format supported through libvpx.

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

18.2.1 Options

Mapping from FFmpeg to libvpx options with conversion notes in parentheses.

‘`threads`’

g_threads

‘`profile`’

g_profile

‘`vb`’

rc_target_bitrate

‘`g`’

kf_max_dist

‘`keyint_min`’

kf_min_dist

‘`qmin`’

rc_min_quantizer

‘`qmax`’

rc_max_quantizer

‘bufsize, vb’

rc_buf_sz (bufsize * 1000 / vb)

rc_buf_optimal_sz (bufsize * 1000 / vb * 5 / 6)

‘rc_init_occupancy, vb’

rc_buf_initial_sz (rc_init_occupancy * 1000 / vb)

‘rc_buffer_aggressivity’

rc_undershoot_pct

‘skip_threshold’

rc_dropframe_thresh

‘qcomp’

rc_2pass_vbr_bias_pct

‘maxrate, vb’

rc_2pass_vbr_maxsection_pct (maxrate * 100 / vb)

‘minrate, vb’

rc_2pass_vbr_minsection_pct (minrate * 100 / vb)

‘minrate, maxrate, vb’

VPX_CBR (minrate == maxrate == vb)

‘crf’

VPX_CQ, VP8E_SET_CQ_LEVEL

‘quality’

‘*best*’

VPX_DL_BEST_QUALITY

‘*good*’

VPX_DL_GOOD_QUALITY

‘*realtime*’

VPX_DL_REALTIME

‘speed’

VP8E_SET_CPUUSED

‘nr’

VP8E_SET_NOISE_SENSITIVITY

‘mb_threshold’

VP8E_SET_STATIC_THRESHOLD

‘slices’

VP8E_SET_TOKEN_PARTITIONS

‘max-intra-rate’

VP8E_SET_MAX_INTRA_BITRATE_PCT

‘force_key_frames’

VPX_EFLAG_FORCE_KF

‘Alternate reference frame related’

‘vp8flags altref’

VP8E_SET_ENABLEAUTOALTREF

‘*arnr_max_frames*’

VP8E_SET_ARNR_MAXFRAMES

‘*arnr_type*’

VP8E_SET_ARNR_TYPE

‘*arnr_strength*’

VP8E_SET_ARNR_STRENGTH

‘*rc_lookahead*’

g_lag_in_frames

‘vp8flags error_resilient’

g_error_resilient

For more information about libvpx see: http://www.webmproject.org/

18.3 libx264

x264 H.264/MPEG-4 AVC encoder wrapper

Requires the presence of the libx264 headers and library during configuration. You need to explicitly configure the build with `--enable-libx264`.

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

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

18.3.1 Option Mapping

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

b	bitrate FFmpeg `b` option is expressed in bits/s, x264 `bitrate` in kilobits/s.
bf	bframes Maximum number of B-frames.
g	keyint Maximum GOP size.
qmin	qpmin
qmax	qpmax
qdiff	qpstep
qblur	qblur
qcomp	qcomp
refs	ref
sc_threshold	scenecut
trellis	trellis
nr	nr Noise reduction.
me_range	merange
me_method	me
subq	subme
b_strategy	b-adapt
keyint_min	keyint-min
coder	cabac Set coder to `ac` to use CABAC.
cmp	chroma-me Set to `chroma` to use chroma motion estimation.
threads	threads
thread_type	sliced_threads Set to `slice` to use sliced threading instead of frame threading.
flags -cgop	open-gop Set `-cgop` to use recovery points to close GOPs.
rc_init_occupancy	vbv-init Initial buffer occupancy.

18.3.2 Private Options

‘`-preset` *`string`*’

Set the encoding preset (cf. x264 –fullhelp).

‘`-tune` *`string`*’

Tune the encoding params (cf. x264 –fullhelp).

‘`-profile` *`string`*’

Set profile restrictions (cf. x264 –fullhelp).

‘`-fastfirstpass` *`integer`*’

Use fast settings when encoding first pass.

‘`-crf` *`float`*’

Select the quality for constant quality mode.

‘`-crf_max` *`float`*’

In CRF mode, prevents VBV from lowering quality beyond this point.

‘`-qp` *`integer`*’

Constant quantization parameter rate control method.

‘`-aq-mode` *`integer`*’

AQ method

Possible values:

‘`none`’
‘`variance`’

Variance AQ (complexity mask).

‘`autovariance`’

Auto-variance AQ (experimental).

‘`-aq-strength` *`float`*’

AQ strength, reduces blocking and blurring in flat and textured areas.

‘`-psy` *`integer`*’

Use psychovisual optimizations.

‘`-psy-rd` *`string`*’

Strength of psychovisual optimization, in <psy-rd>:<psy-trellis> format.

‘`-rc-lookahead` *`integer`*’

Number of frames to look ahead for frametype and ratecontrol.

‘`-weightb` *`integer`*’

Weighted prediction for B-frames.

‘`-weightp` *`integer`*’

Weighted prediction analysis method.

Possible values:

‘`none`’
‘`simple`’
‘`smart`’

‘`-ssim` *`integer`*’

Calculate and print SSIM stats.

‘`-intra-refresh` *`integer`*’

Use Periodic Intra Refresh instead of IDR frames.

‘`-b-bias` *`integer`*’

Influences how often B-frames are used.

‘`-b-pyramid` *`integer`*’

Keep some B-frames as references.

Possible values:

‘`none`’
‘`strict`’

Strictly hierarchical pyramid.

‘`normal`’

Non-strict (not Blu-ray compatible).

‘`-mixed-refs` *`integer`*’

One reference per partition, as opposed to one reference per macroblock.

‘`-8x8dct` *`integer`*’

High profile 8x8 transform.

‘`-fast-pskip` *`integer`*’
‘`-aud` *`integer`*’

Use access unit delimiters.

‘`-mbtree` *`integer`*’

Use macroblock tree ratecontrol.

‘`-deblock` *`string`*’

Loop filter parameters, in <alpha:beta> form.

‘`-cplxblur` *`float`*’

Reduce fluctuations in QP (before curve compression).

‘`-partitions` *`string`*’

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

‘`-direct-pred` *`integer`*’

Direct MV prediction mode

Possible values:

‘`none`’
‘`spatial`’
‘`temporal`’
‘`auto`’

‘`-slice-max-size` *`integer`*’

Limit the size of each slice in bytes.

‘`-stats` *`string`*’

Filename for 2 pass stats.

‘`-nal-hrd` *`integer`*’

Signal HRD information (requires vbv-bufsize; cbr not allowed in .mp4).

Possible values:

‘none’
‘vbr’
‘cbr’

‘x264opts *options*’

Allow to set any x264 option, see `x264 --fullhelp` for a list.

options is a list of *key=value* couples separated by ":". In *filter* and *psy-rd* options that use ":" as a separator themselves, use "," instead. They accept it as well since long ago but this is kept undocumented for some reason.

For example to specify libx264 encoding options with `ffmpeg`:

```
ffmpeg -i foo.mpg -vcodec libx264 -x264opts keyint=123:min-keyint=20 -an out.mkv
```

For more information about libx264 and the supported options see:
http://www.videolan.org/developers/x264.html

‘-x264-params *string*’

Override the x264 configuration using a :-separated list of key=value parameters.

```
-x264-params level=30:bframes=0:weightp=0:cabac=0:ref=1:vbv-maxrate=768:vbv-bufsize=2000:analyse=all:me=umh:no-fast-pskip=1:subq=6:8x8dct=0:trellis=0
```

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

18.4 ProRes

Apple ProRes encoder.

FFmpeg contains 2 ProRes encoders, the prores-aw and prores-ks encoder. The used encoder can be choosen with the `-vcodec` option.

18.4.1 Private Options for prores-ks

‘profile *integer*’

Select the ProRes profile to encode

‘proxy’

‘lt’
‘standard’
‘hq’

‘quant_mat *integer*’

Select quantization matrix.

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

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

‘bits_per_mb *integer*’

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

‘mbs_per_slice *integer*’

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

‘vendor *string*’

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

18.4.2 Speed considerations

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

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

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

19. Bitstream Filters

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

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

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

Below is a description of the currently available bitstream filters.

19.1 aac_adtstoasc

19.2 chomp

19.3 dump_extradata

19.4 h264_mp4toannexb

Convert an H.264 bitstream from length prefixed mode to start code prefixed mode (as defined in the Annex B of the ITU-T H.264 specification).

This is required by some streaming formats, typically the MPEG-2 transport stream format ("mpegts").

For example to remux an MP4 file containing an H.264 stream to mpegts format with `ffmpeg`, you can use the command:

```
ffmpeg -i INPUT.mp4 -codec copy -bsf:v h264_mp4toannexb OUTPUT.ts
```

19.5 imx_dump_header

19.6 mjpeg2jpeg

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

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

```
ffmpeg -i ../some_mjpeg.avi -c:v copy frames_%d.jpg
```

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

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

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

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

19.7 mjpega_dump_header

19.8 movsub

19.9 mp3_header_compress

19.10 mp3_header_decompress

19.11 noise

19.12 remove_extradata

20. Format Options

The libavformat library provides some generic global options, which can be set on all the muxers and demuxers. In addition each muxer or demuxer may support so-called private options, which are specific for that component.

Options may be set by specifying *-option value* in the FFmpeg tools, or by setting the value explicitly in the `AVFormatContext` options or using the ‘`libavutil/opt.h`’ API for programmatic use.

The list of supported options follows:

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

Possible values:

‘`direct`’

Reduce buffering.

‘`probesize` *`integer`* (*`input`*)’

Set probing size in bytes, i.e. the size of the data to analyze to get stream information. A higher value will allow to detect more information in case it is dispersed into the stream, but will increase latency. Must be an integer not lesser than 32. It is 5000000 by default.

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

Set packet size.

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

Set format flags.

Possible values:

‘`ignidx`’

Ignore index.

‘`genpts`’

Generate PTS.

‘`nofillin`’

Do not fill in missing values that can be exactly calculated.

‘`noparse`’

Disable AVParsers, this needs `+nofillin` too.

‘`igndts`’

Ignore DTS.

‘`discardcorrupt`’

Discard corrupted frames.

‘`sortdts`’

Try to interleave output packets by DTS.

‘keepside’

Do not merge side data.

‘latm’

Enable RTP MP4A-LATM payload.

‘nobuffer’

Reduce the latency introduced by optional buffering

‘analyzeduration *integer* (*input*)’

Specify how many microseconds are analyzed to probe the input. A higher value will allow to detect more accurate information, but will increase latency. It defaults to 5,000,000 microseconds = 5 seconds.

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

Set decryption key.

‘indexmem *integer* (*input*)’

Set max memory used for timestamp index (per stream).

‘rtbufsize *integer* (*input*)’

Set max memory used for buffering real-time frames.

‘fdebug *flags* (*input/output*)’

Print specific debug info.

Possible values:

‘ts’

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

Set maximum muxing or demuxing delay in microseconds.

‘fpsprobesize *integer* (*input*)’

Set number of frames used to probe fps.

‘audio_preload *integer* (*output*)’

Set microseconds by which audio packets should be interleaved earlier.

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

Set microseconds for each chunk.

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

Set size in bytes for each chunk.

‘`err_detect, f_err_detect` *`flags`* (*`input`*)’

Set error detection flags. `f_err_detect` is deprecated and should be used only via the `ffmpeg` tool.

Possible values:

‘`crccheck`’

Verify embedded CRCs.

‘`bitstream`’

Detect bitstream specification deviations.

‘`buffer`’

Detect improper bitstream length.

‘`explode`’

Abort decoding on minor error detection.

‘`careful`’

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

‘`compliant`’

Consider all spec non compliancies as errors.

‘`aggressive`’

Consider things that a sane encoder should not do as an error.

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

Use wallclock as timestamps.

‘avoid_negative_ts *integer* (*output*)’

Shift timestamps to make them positive. A value of 1 enables shifting, a value of 0 disables it, the default value of -1 enables shifting when required by the target format.

When shifting is enabled, all output timestamps are shifted by the same amount. Audio, video, and subtitles desynching and relative timestamp differences are preserved compared to how they would have been without shifting.

Also note that this affects only leading negative timestamps, and not non-monotonic negative timestamps.

‘flush_packets *integer* (*output*)’

Flush the underlying I/O stream after each packet. Default 1 enables it, and has the effect of reducing the latency; 0 disables it and may slightly increase performance in some cases.

21. Demuxers

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

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

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

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

The description of some of the currently available demuxers follows.

21.1 applehttp

Apple HTTP Live Streaming demuxer.

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

21.2 concat

Virtual concatenation script demuxer.

This demuxer reads a list of files and other directives from a text file and demuxes them one after the other, as if all their packet had been muxed together.

The timestamps in the files are adjusted so that the first file starts at 0 and each next file starts where the previous one finishes. Note that it is done globally and may cause gaps if all streams do not have exactly the same length.

All files must have the same streams (same codecs, same time base, etc.).

The duration of each file is used to adjust the timestamps of the next file: if the duration is incorrect (because it was computed using the bit-rate or because the file is truncated, for example), it can cause artifacts. The `duration` directive can be used to override the duration stored in each file.

21.2.1 Syntax

The script is a text file in extended-ASCII, with one directive per line. Empty lines, leading spaces and lines starting with '#' are ignored. The following directive is recognized:

'`file` *`path`*'

Path to a file to read; special characters and spaces must be escaped with backslash or single quotes.

All subsequent directives apply to that file.

'`ffconcat version 1.0`'

Identify the script type and version. It also sets the '`safe`' option to 1 if it was to its default -1.

To make FFmpeg recognize the format automatically, this directive must appears exactly as is (no extra space or byte-order-mark) on the very first line of the script.

'`duration` *`dur`*'

Duration of the file. This information can be specified from the file; specifying it here may be more efficient or help if the information from the file is not available or accurate.

If the duration is set for all files, then it is possible to seek in the whole concatenated video.

21.2.2 Options

This demuxer accepts the following option:

‘safe’

If set to 1, reject unsafe file paths. A file path is considered safe if it does not contain a protocol specification and is relative and all components only contain characters from the portable character set (letters, digits, period, underscore and hyphen) and have no period at the beginning of a component.

If set to 0, any file name is accepted.

The default is -1, it is equivalent to 1 if the format was automatically probed and 0 otherwise.

21.3 libquvi

Play media from Internet services using the quvi project.

The demuxer accepts a ‘format’ option to request a specific quality. It is by default set to *best*.

See http://quvi.sourceforge.net/ for more information.

FFmpeg needs to be built with --enable-libquvi for this demuxer to be enabled.

21.4 image2

Image file demuxer.

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

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

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

This demuxer accepts the following options:

‘framerate’

Set the frame rate for the video stream. It defaults to 25.

‘loop’

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

‘pattern_type’

Select the pattern type used to interpret the provided filename.

pattern_type accepts one of the following values.

‘`sequence`’

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

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

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

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

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

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

‘`glob`’

Select a glob wildcard pattern type.

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

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

Select a mixed glob wildcard/sequence pattern.

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

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

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

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

Default value is *glob_sequence*.

‘`pixel_format`’

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

‘`start_number`’

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

‘`start_number_range`’

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

‘`video_size`’

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

21.4.1 Examples

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

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

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

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

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

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

21.5 rawvideo

Raw video demuxer.

This demuxer allows to read raw video data. Since there is no header specifying the assumed video parameters, the user must specify them in order to be able to decode the data correctly.

This demuxer accepts the following options:

‘framerate’

Set input video frame rate. Default value is 25.

‘pixel_format’

Set the input video pixel format. Default value is yuv420p.

‘video_size’

Set the input video size. This value must be specified explicitly.

For example to read a rawvideo file ‘input.raw’ with ffplay, assuming a pixel format of rgb24, a video size of 320x240, and a frame rate of 10 images per second, use the command:

```
ffplay -f rawvideo -pixel_format rgb24 -video_size 320x240 -framerate 10 input.raw
```

21.6 sbg

SBaGen script demuxer.

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

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

A SBG script can mix absolute and relative timestamps. If the script uses either only absolute timestamps (including the script start time) or only relative ones, then its layout is fixed, and the conversion is straightforward. On the other hand, if the script mixes both kind of timestamps, then the *NOW* reference for relative timestamps will be taken from the current time of day at the time the script is read, and the script layout will be frozen according to that reference. That means that if the script is directly played, the

actual times will match the absolute timestamps up to the sound controller's clock accuracy, but if the user somehow pauses the playback or seeks, all times will be shifted accordingly.

21.7 tedcaptions

JSON captions used for TED Talks.

TED does not provide links to the captions, but they can be guessed from the page. The file '`tools/bookmarklets.html`' from the FFmpeg source tree contains a bookmarklet to expose them.

This demuxer accepts the following option:

'`start_time`'

Set the start time of the TED talk, in milliseconds. The default is 15000 (15s). It is used to sync the captions with the downloadable videos, because they include a 15s intro.

Example: convert the captions to a format most players understand:

```
ffmpeg -i http://www.ted.com/talks/subtitles/id/1/lang/en talk1-en.srt
```

22. Muxers

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

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

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

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

A description of some of the currently available muxers follows.

22.1 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`’:

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

You can print the CRC to stdout with the command:

```
ffmpeg -i INPUT -f crc -
```

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

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

See also the framecrc muxer.

22.2 framecrc

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

This muxer computes and prints the Adler-32 CRC for each audio and video packet. 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 packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, 0xCRC
```

CRC is a hexadecimal number 0-padded to 8 digits containing the CRC of the packet.

For example to compute the CRC of the audio and video frames in ‘`INPUT`’, converted to raw audio and video packets, and store it in the file ‘`out.crc`’:

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

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framecrc -
```

With `ffmpeg`, you can select the output format to which the audio and video frames are encoded before computing the CRC for each packet by specifying the audio and video codec. 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:

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

See also the crc muxer.

22.3 framemd5

Per-packet MD5 testing format.

This muxer computes and prints the MD5 hash for each audio and video packet. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the hash.

The output of the muxer consists of a line for each audio and video packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, MD5
```

MD5 is a hexadecimal number representing the computed MD5 hash for the packet.

For example to compute the MD5 of the audio and video frames in '`INPUT`', converted to raw audio and video packets, and store it in the file '`out.md5`':

```
ffmpeg -i INPUT -f framemd5 out.md5
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framemd5 -
```

See also the md5 muxer.

22.4 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.

```
ffmpeg -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*.

22.5 ico

ICO file muxer.

Microsoft’s icon file format (ICO) has some strict limitations that should be noted:

- Size cannot exceed 256 pixels in any dimension
- Only BMP and PNG images can be stored
- If a BMP image is used, it must be one of the following pixel formats:

```
BMP Bit Depth      FFmpeg Pixel Format
1bit               pal8
4bit               pal8
8bit               pal8
16bit              rgb555le
24bit              bgr24
32bit              bgra
```

- If a BMP image is used, it must use the BITMAPINFOHEADER DIB header
- If a PNG image is used, it must use the rgba pixel format

22.6 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 "%0*N*d", this string specifies the position of the characters representing a numbering in the filenames. If the form "%0*N*d" is used, the string representing the number in each filename is 0-padded to *N* digits. The literal character ’%’ can be specified in the pattern with the string "%%".

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

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

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

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

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

Note that with `ffmpeg`, 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:

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

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

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

‘`start_number` *`number`*’

Start the sequence from *number*. Default value is 1. Must be a positive 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.

The image muxer supports the .Y.U.V image file format. This format is special in that that each image frame consists of three files, for each of the YUV420P components. To read or write this image file format, specify the name of the '.Y' file. The muxer will automatically open the '.U' and '.V' files as required.

22.7 md5

MD5 testing format.

This muxer computes and prints the MD5 hash 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 hash.

The output of the muxer consists of a single line of the form: MD5=*MD5*, where *MD5* is a hexadecimal number representing the computed MD5 hash.

For example to compute the MD5 hash of the input converted to raw audio and video, and store it in the file '`out.md5`':

```
ffmpeg -i INPUT -f md5 out.md5
```

You can print the MD5 to stdout with the command:

```
ffmpeg -i INPUT -f md5 -
```

See also the framemd5 muxer.

22.8 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 by adding *faststart* to the *movflags*, or 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:

'`-moov_size` *`bytes`*'

> Reserves space for the moov atom at the beginning of the file instead of placing the moov atom at the end. If the space reserved is insufficient, muxing will fail.

'`-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 `ffmpeg`.)

‘`-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 moov atom on top 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 rtphint`’

Add RTP hinting tracks to the output file.

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

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

22.9 mpegts

MPEG transport stream muxer.

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

The muxer options are:

‘`-mpegts_original_network_id` *number*’

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

‘`-mpegts_transport_stream_id` *number*’

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

‘`-mpegts_service_id` *number*’

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

‘`-mpegts_pmt_start_pid` *number*’

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

‘`-mpegts_start_pid` *number*’

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

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

```
ffmpeg -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
```

22.10 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 `ffmpeg` you can use the command:

```
ffmpeg -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 `ffmpeg` syntax.

Alternatively you can write the command as:

```
ffmpeg -benchmark -i INPUT -f null -
```

22.11 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:

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

22.12 segment, stream_segment, ssegment

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.

`stream_segment` is a variant of the muxer used to write to streaming output formats, i.e. which do not require global headers, and is recommended for outputting e.g. to MPEG transport stream segments. `ssegment` is a shorter alias for `stream_segment`.

Every segment starts with a keyframe of the selected reference stream, which is set through the '`reference_stream`' option.

Note that if you want accurate splitting for a video file, you need to make the input key frames correspond to the exact splitting times expected by the segmenter, or the segment muxer will start the new segment with the key frame found next after the specified start time.

The segment muxer works best with a single constant frame rate video.

Optionally it can generate a list of the created segments, by setting the option *segment_list*. The list type is specified by the *segment_list_type* option.

The segment muxer supports the following options:

'`reference_stream` *`specifier`*'

Set the reference stream, as specified by the string *specifier*. If *specifier* is set to `auto`, the reference is choosen automatically. Otherwise it must be a stream specifier (see the "Stream specifiers" chapter in the ffmpeg manual) which specifies the reference stream. The default value is "auto".

'`segment_format` *`format`*'

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

'`segment_list` *`name`*'

Generate also a listfile named *name*. If not specified no listfile is generated.

'`segment_list_flags` *`flags`*'

Set flags affecting the segment list generation.

It currently supports the following flags:

cache

Allow caching (only affects M3U8 list files).

live

Allow live-friendly file generation.

Default value is `cache`.

‘`segment_list_size` *`size`*’

Update the list file so that it contains at most the last *size* segments. If 0 the list file will contain all the segments. Default value is 0.

‘`segment_list type` *`type`*’

Specify the format for the segment list file.

The following values are recognized:

‘`flat`’

Generate a flat list for the created segments, one segment per line.

‘`csv, ext`’

Generate a list for the created segments, one segment per line, each line matching the format (comma-separated values):

```
segment_filename,segment_start_time,segment_end_time
```

segment_filename is the name of the output file generated by the muxer according to the provided pattern. CSV escaping (according to RFC4180) is applied if required.

segment_start_time and *segment_end_time* specify the segment start and end time expressed in seconds.

A list file with the suffix `".csv"` or `".ext"` will auto-select this format.

`ext` is deprecated in favor or `csv`.

‘`ffconcat`’

Generate an ffconcat file for the created segments. The resulting file can be read using the FFmpeg concat demuxer.

A list file with the suffix `".ffcat"` or `".ffconcat"` will auto-select this format.

‘`m3u8`’

Generate an extended M3U8 file, version 3, compliant with http://tools.ietf.org/id/draft-pantos-http-live-streaming.

A list file with the suffix `".m3u8"` will auto-select this format.

If not specified the type is guessed from the list file name suffix.

‘`segment_time` *`time`*’

Set segment duration to *time*, the value must be a duration specification. Default value is "2". See also the ‘`segment_times`’ option.

Note that splitting may not be accurate, unless you force the reference stream key-frames at the given time. See the introductory notice and the examples below.

‘`segment_time_delta` *`delta`*’

Specify the accuracy time when selecting the start time for a segment, expressed as a duration specification. Default value is "0".

When delta is specified a key-frame will start a new segment if its PTS satisfies the relation:

```
PTS >= start_time - time_delta
```

This option is useful when splitting video content, which is always split at GOP boundaries, in case a key frame is found just before the specified split time.

In particular may be used in combination with the ‘`ffmpeg`’ option *force_key_frames*. The key frame times specified by *force_key_frames* may not be set accurately because of rounding issues, with the consequence that a key frame time may result set just before the specified time. For constant frame rate videos a value of 1/2**frame_rate* should address the worst case mismatch between the specified time and the time set by *force_key_frames*.

‘`segment_times` *`times`*’

Specify a list of split points. *times* contains a list of comma separated duration specifications, in increasing order. See also the ‘`segment_time`’ option.

‘`segment_frames` *`frames`*’

Specify a list of split video frame numbers. *frames* contains a list of comma separated integer numbers, in increasing order.

This option specifies to start a new segment whenever a reference stream key frame is found and the sequential number (starting from 0) of the frame is greater or equal to the next value in the list.

‘`segment_wrap` *`limit`*’

Wrap around segment index once it reaches *limit*.

‘`segment_start_number` *`number`*’

Set the sequence number of the first segment. Defaults to `0`.

‘`reset_timestamps` *`1|0`*’

Reset timestamps at the begin of each segment, so that each segment will start with near-zero timestamps. It is meant to ease the playback of the generated segments. May not work with some combinations of muxers/codecs. It is set to `0` by default.

22.12.1 Examples

- To remux the content of file ‘`in.mkv`’ to a list of segments ‘`out-000.nut`’, ‘`out-001.nut`’, etc., and write the list of generated segments to ‘`out.list`’:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.list out%03d.nut
```

- As the example above, but segment the input file according to the split points specified by the *segment_times* option:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 out%03d.nut
```

- As the example above, but use the `ffmpeg` *force_key_frames* option to force key frames in the input at the specified location, together with the segment option *segment_time_delta* to account for possible roundings operated when setting key frame times.

```
ffmpeg -i in.mkv -force_key_frames 1,2,3,5,8,13,21 -codec:v mpeg4 -codec:a pcm_s16le -map 0 \
-f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 -segment_time_delta 0.05 out%03d.nut
```

In order to force key frames on the input file, transcoding is required.

- Segment the input file by splitting the input file according to the frame numbers sequence specified with the *segment_frames* option:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_frames 100,200,300,500,800 out%03d.nut
```

- To convert the ‘`in.mkv`’ to TS segments using the `libx264` and `libfaac` encoders:

```
ffmpeg -i in.mkv -map 0 -codec:v libx264 -codec:a libfaac -f ssegment -segment_list out.list out%03d.ts
```

- Segment the input file, and create an M3U8 live playlist (can be used as live HLS source):

```
ffmpeg -re -i in.mkv -codec copy -map 0 -f segment -segment_list playlist.m3u8 \
-segment_list_flags +live -segment_time 10 out%03d.mkv
```

22.13 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. The legacy ID3v1 tag is not written by default, but may be enabled with the `write_id3v1` option.

For seekable output the muxer also writes 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 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:

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

To attach a picture to an mp3 file select both the audio and the picture stream with `map`:

```
ffmpeg -i input.mp3 -i cover.png -c copy -map 0 -map 1
-metadata:s:v title="Album cover" -metadata:s:v comment="Cover (Front)" out.mp3
```

22.14 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.

22.15 tee

The tee muxer can be used to write the same data to several files or any other kind of muxer. It can be used, for example, to both stream a video to the network and save it to disk at the same time.

It is different from specifying several outputs to the `ffmpeg` command-line tool because the audio and video data will be encoded only once with the tee muxer; encoding can be a very expensive process. It is not useful when using the libavformat API directly because it is then possible to feed the same packets to several muxers directly.

The slave outputs are specified in the file name given to the muxer, separated by '|'. If any of the slave name contains the '|' separator, leading or trailing spaces or any special character, it must be escaped (see the "Quoting and escaping" section in the ffmpeg-utils manual).

Options can be specified for each slave by prepending them as a list of *key=value* pairs separated by ':', between square brackets. If the options values contain a special character or the ':' separator, they must be escaped; note that this is a second level escaping.

Example: encode something and both archive it in a WebM file and stream it as MPEG-TS over UDP (the streams need to be explicitly mapped):

```
ffmpeg -i ... -c:v libx264 -c:a mp2 -f tee -map 0:v -map 0:a
  "archive-20121107.mkv|[f=mpegts]udp://10.0.1.255:1234/"
```

Note: some codecs may need different options depending on the output format; the auto-detection of this can not work with the tee muxer. The main example is the '`global_header`' flag.

23. Metadata

FFmpeg is able to dump metadata from media files into a simple UTF-8-encoded INI-like text file and then load it back using the metadata muxer/demuxer.

The file format is as follows:

1. A file consists of a header and a number of metadata tags divided into sections, each on its own line.
2. The header is a ';FFMETADATA' string, followed by a version number (now 1).
3. Metadata tags are of the form 'key=value'
4. Immediately after header follows global metadata
5. After global metadata there may be sections with per-stream/per-chapter metadata.
6. A section starts with the section name in uppercase (i.e. STREAM or CHAPTER) in brackets ('[', ']')

and ends with next section or end of file.

7. At the beginning of a chapter section there may be an optional timebase to be used for start/end values. It must be in form ’TIMEBASE=num/den’, where num and den are integers. If the timebase is missing then start/end times are assumed to be in milliseconds. Next a chapter section must contain chapter start and end times in form ’START=num’, ’END=num’, where num is a positive integer.
8. Empty lines and lines starting with ’;’ or ’#’ are ignored.
9. Metadata keys or values containing special characters (’=’, ’;’, ’#’, ’\’ and a newline) must be escaped with a backslash ’\’.
10. Note that whitespace in metadata (e.g. foo = bar) is considered to be a part of the tag (in the example above key is ’foo ’, value is ’ bar’).

A ffmetadata file might look like this:

```
;FFMETADATA1
title=bike\\shed
;this is a comment
artist=FFmpeg troll team

[CHAPTER]
TIMEBASE=1/1000
START=0
#chapter ends at 0:01:00
END=60000
title=chapter \#1
[STREAM]
title=multi\
line
```

24. Protocols

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

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

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

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

A description of the currently available protocols follows.

24.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

BluRay angle

‘`chapter`’

Start chapter (1...N)

‘`playlist`’

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

Examples:

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

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

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

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

24.2 concat

Physical concatenation protocol.

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

A URL accepted by this protocol has the syntax:

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

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

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

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

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

24.3 data

Data in-line in the URI. See http://en.wikipedia.org/wiki/Data_URI_scheme.

For example, to convert a GIF file given inline with `ffmpeg`:

```
ffmpeg -i "" smiley.png
```

24.4 file

File access protocol.

Allow to read from or read to a file.

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

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

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

24.5 gopher

Gopher protocol.

24.6 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.

24.7 http

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options.

‘`seekable`’

Control seekability of connection. If set to 1 the resource is supposed to be seekable, if set to 0 it is assumed not to be seekable, if set to -1 it will try to autodetect if it is seekable. Default value is -1.

‘`chunked_post`’

If set to 1 use chunked transfer-encoding for posts, default is 1.

‘`headers`’

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

‘`content_type`’

Force a content type.

‘`user-agent`’

Override User-Agent header. If not specified the protocol will use a string describing the libavformat build.

‘`multiple_requests`’

Use persistent connections if set to 1. By default it is 0.

‘`post_data`’

Set custom HTTP post data.

‘`timeout`’

Set timeout of socket I/O operations used by the underlying low level operation. By default it is set to -1, which means that the timeout is not specified.

‘`mime_type`’

Set MIME type.

‘`cookies`’

Set the cookies to be sent in future requests. The format of each cookie is the same as the value of a Set-Cookie HTTP response field. Multiple cookies can be delimited by a newline character.

24.7.1 HTTP Cookies

Some HTTP requests will be denied unless cookie values are passed in with the request. The ‘`cookies`’ option allows these cookies to be specified. At the very least, each cookie must specify a value along with a path and domain. HTTP requests that match both the domain and path will automatically include the cookie value in the HTTP Cookie header field. Multiple cookies can be delimited by a newline.

The required syntax to play a stream specifying a cookie is:

```
ffplay -cookies "nlqptid=nltid=tsn; path=/; domain=somedomain.com;" http://somedomain.com/somestream.m3u8
```

24.8 mmst

MMS (Microsoft Media Server) protocol over TCP.

24.9 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

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

24.10 md5

MD5 output protocol.

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

Some examples follow.

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

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

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

24.11 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

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

For example to read from stdin with `ffmpeg`:

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

For writing to stdout with `ffmpeg`:

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

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

24.12 rtmp

Real-Time Messaging Protocol.

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

The required syntax is:

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

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

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

‘`app`’

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

‘`playpath`’

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

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

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

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

‘`rtmp_app`’

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

‘`rtmp_buffer`’

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

‘`rtmp_conn`’

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

‘`rtmp_flashver`’

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

‘rtmp_flush_interval’

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

‘rtmp_live’

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

‘rtmp_pageurl’

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

‘rtmp_playpath’

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

‘rtmp_subscribe’

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

‘rtmp_swfhash’

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

‘rtmp_swfsize’

Size of the decompressed SWF file, required for SWFVerification.

‘rtmp_swfurl’

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

‘rtmp_swfverify’

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

‘rtmp_tcurl’

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

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

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

24.13 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.

24.14 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.

24.15 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.

24.16 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.

24.17 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.

24.18 rtmp, rtmpe, rtmps, rtmpt, rtmpte

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

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

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

The required syntax is:

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

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

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

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

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

To play the same stream using `ffplay`:

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

24.19 rtp

Real-Time Protocol.

24.20 rtsp

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

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

The required syntax for a RTSP url is:

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

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

Flags for `rtsp_transport`:

‘`udp`’

Use UDP as lower transport protocol.

‘`tcp`’

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

‘`udp_multicast`’

Use UDP multicast as lower transport protocol.

‘`http`’

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

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

Flags for `rtsp_flags`:

‘`filter_src`’

Accept packets only from negotiated peer address and port.

‘`listen`’

Act as a server, listening for an incoming connection.

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

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

Example command lines:

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

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

To watch a stream tunneled over HTTP:

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

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

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

To receive a stream in realtime:

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

‘`stimeout`’

Socket IO timeout in micro seconds.

24.21 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.

24.21.1 Muxer

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

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

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

‘`announce_addr=address`’

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

‘`announce_port=port`’

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

‘`ttl=ttl`’

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

‘`same_port=`*`0|1`*’

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

Example command lines follow.

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

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

Similarly, for watching in `ffplay`:

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

And for watching in `ffplay`, over IPv6:

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

24.21.2 Demuxer

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

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

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

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

Example command lines follow.

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

```
ffplay sap://
```

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

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

24.22 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

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

‘`listen`’

Listen for an incoming connection

‘`timeout=microseconds`’

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

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

24.23 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

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

‘`listen`’

Act as a server, listening for an incoming connection.

‘`cafile=filename`’

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

‘`cert=filename`’

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

‘`key=filename`’

Private key file.

‘verify=*0|1*’

Verify the peer’s certificate.

Example command lines:

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

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

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

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

24.24 udp

User Datagram Protocol.

The required syntax for a UDP url is:

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

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

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

The list of supported options follows.

‘buffer_size=*size*’

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

‘localport=*port*’

Override the local UDP port to bind with.

‘localaddr=*addr*’

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

‘pkt_size=*size*’

Set the size in bytes of UDP packets.

‘reuse=*1|0*’

Explicitly allow or disallow reusing UDP sockets.

‘ttl=*ttl*’

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

‘connect=*1|0*’

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

‘sources=*address*[,*address*]’

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

‘block=*address*[,*address*]’

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

‘fifo_size=*units*’

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

‘overrun_nonfatal=*1|0*’

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

‘timeout=*microseconds*’

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

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

To stream over UDP to a remote endpoint:

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

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

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

To receive over UDP from a remote endpoint:

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

25. Device Options

The libavdevice library provides the same interface as libavformat. Namely, an input device is considered like a demuxer, and an output device like a muxer, and the interface and generic device options are the same provided by libavformat (see the ffmpeg-formats manual).

In addition each input or output device may support so-called private options, which are specific for that component.

Options may be set by specifying *-option value* in the FFmpeg tools, or by setting the value explicitly in the device `AVFormatContext` options or using the ‘`libavutil/opt.h`’ API for programmatic use.

26. Input Devices

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

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

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

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

A description of the currently available input devices follows.

26.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

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

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

An ALSA identifier has the syntax:

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

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

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

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

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

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

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

26.2 bktr

BSD video input device.

26.3 dshow

Windows DirectShow input device.

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

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

The input name should be in the format:

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

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

26.3.1 Options

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

‘`video_size`’

Set the video size in the captured video.

‘`framerate`’

Set the frame rate in the captured video.

‘`sample_rate`’

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

‘`sample_size`’

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

‘`channels`’

Set the number of channels in the captured audio.

‘`list_devices`’

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

‘`list_options`’

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

‘`video_device_number`’

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

‘`audio_device_number`’

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

‘`pixel_format`’

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

‘`audio_buffer_size`’

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

26.3.2 Examples

- Print the list of DirectShow supported devices and exit:

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

- Open video device *Camera*:

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

- Open second video device with name *Camera*:

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

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

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

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

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

26.4 dv1394

Linux DV 1394 input device.

26.5 fbdev

Linux framebuffer input device.

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

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

To record from the framebuffer device ‘/dev/fb0’ with ffmpeg:

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

You can take a single screenshot image with the command:

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

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

26.6 iec61883

FireWire DV/HDV input device using libiec61883.

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

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

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

26.6.1 Options

‘dvtype’

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

‘dvbuffer’

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

‘dvguid’

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

26.6.2 Examples

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

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

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

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

26.7 jack

JACK input device.

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

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

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

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

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

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

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

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

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

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

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

26.8 lavfi

Libavfilter input virtual device.

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

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

26.8.1 Options

‘`graph`’

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

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

‘`graph_file`’

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

26.8.2 Examples

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

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

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

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

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

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

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

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

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

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

26.9 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

26.10 openal

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

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

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

An incomplete list of OpenAL implementations follows:

Creative

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

OpenAL Soft

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

Apple

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

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

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

26.10.1 Options

‘`channels`’

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

‘`sample_size`’

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

‘`sample_rate`’

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

‘`list_devices`’

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

26.10.2 Examples

Print the list of OpenAL supported devices and exit:

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

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

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

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

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

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

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

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

26.11 oss

Open Sound System input device.

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

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

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

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

26.12 pulse

pulseaudio input device.

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

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

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

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

26.12.1 *server* AVOption

The syntax is:

```
-server server name
```

Connects to a specific server.

26.12.2 *name* AVOption

The syntax is:

```
-name application name
```

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

26.12.3 *stream_name* AVOption

The syntax is:

```
-stream_name stream name
```

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

26.12.4 *sample_rate* AVOption

The syntax is:

```
-sample_rate samplerate
```

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

26.12.5 *channels* AVOption

The syntax is:

```
-channels N
```

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

26.12.6 *frame_size* AVOption

The syntax is:

```
-frame_size bytes
```

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

26.12.7 *fragment_size* AVOption

The syntax is:

```
-fragment_size bytes
```

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

26.13 sndio

sndio input device.

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

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

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

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

26.14 video4linux2, v4l2

Video4Linux2 input video device.

"v4l2" can be used as alias for "video4linux2".

If FFmpeg is built with v4l-utils support (by using the `--enable-libv4l2` configure option), the device will always rely on libv4l2.

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 frame rates. You can check which are supported using `-list_formats all` for Video4Linux2 devices. Some devices, like TV cards, support one or more standards. It is possible to list all the supported standards using `-list_standards all`.

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

Some usage examples of the video4linux2 device with `ffmpeg` and `ffplay`:

- Grab and show the input of a video4linux2 device:

  ```
  ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0
  ```

- Grab and record the input of a video4linux2 device, leave the frame rate and size as previously set:

  ```
  ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
  ```

For more information about Video4Linux, check http://linuxtv.org/.

26.14.1 Options

‘`standard`’

Set the standard. Must be the name of a supported standard. To get a list of the supported standards, use the ‘`list_standards`’ option.

‘`channel`’

Set the input channel number. Default to -1, which means using the previously selected channel.

‘`video_size`’

Set the video frame size. The argument must be a string in the form *WIDTH*x*HEIGHT* or a valid size abbreviation.

‘`pixel_format`’

Select the pixel format (only valid for raw video input).

‘`input_format`’

Set the preferred pixel format (for raw video) or a codec name. This option allows to select the input format, when several are available.

‘`framerate`’

Set the preferred video frame rate.

‘`list_formats`’

List available formats (supported pixel formats, codecs, and frame sizes) and exit.

Available values are:

‘`all`’

Show all available (compressed and non-compressed) formats.

‘`raw`’

Show only raw video (non-compressed) formats.

‘`compressed`’

Show only compressed formats.

‘list_standards’

List supported standards and exit.

Available values are:

‘all’

Show all supported standards.

‘timestamps, ts’

Set type of timestamps for grabbed frames.

Available values are:

‘default’

Use timestamps from the kernel.

‘abs’

Use absolute timestamps (wall clock).

‘mono2abs’

Force conversion from monotonic to absolute timestamps.

Default value is default.

26.15 vfwcap

VfW (Video for Windows) capture input device.

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

26.16 x11grab

X11 video input device.

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

The filename passed as input has the syntax:

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

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

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

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

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

For example to grab from ‘`:0.0`’ using `ffmpeg`:

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

Grab at position `10,20`:

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

26.16.1 Options

‘`draw_mouse`’

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

‘`follow_mouse`’

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

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

For example:

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

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

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

‘framerate’

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

‘show_region’

Show grabbed region on screen.

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

For example:

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

With *follow_mouse*:

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

‘video_size’

Set the video frame size. Default value is vga.

27. Output Devices

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

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

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

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

A description of the currently available output devices follows.

27.1 alsa

ALSA (Advanced Linux Sound Architecture) output device.

27.2 caca

CACA output device.

This output devices allows to show a video stream in CACA window. Only one CACA window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need to configure FFmpeg with `--enable-libcaca`. libcaca is a graphics library that outputs text instead of pixels.

For more information about libcaca, check: http://caca.zoy.org/wiki/libcaca

27.2.1 Options

‘`window_title`’

Set the CACA window title, if not specified default to the filename specified for the output device.

‘`window_size`’

Set the CACA window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video.

‘`driver`’

Set display driver.

‘`algorithm`’

Set dithering algorithm. Dithering is necessary because the picture being rendered has usually far more colours than the available palette. The accepted values are listed with `-list_dither algorithms`.

‘`antialias`’

Set antialias method. Antialiasing smoothens the rendered image and avoids the commonly seen staircase effect. The accepted values are listed with `-list_dither antialiases`.

‘`charset`’

Set which characters are going to be used when rendering text. The accepted values are listed with `-list_dither charsets`.

‘color’

Set color to be used when rendering text. The accepted values are listed with `-list_dither colors`.

‘list_drivers’

If set to ‘true’, print a list of available drivers and exit.

‘list_dither’

List available dither options related to the argument. The argument must be one of `algorithms`, `antialiases`, `charsets`, `colors`.

27.2.2 Examples

- The following command shows the `ffmpeg` output is an CACA window, forcing its size to 80x25:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt rgb24 -window_size 80x25 -f caca -
```

- Show the list of available drivers and exit:

```
ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_drivers true -
```

- Show the list of available dither colors and exit:

```
ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_dither colors -
```

27.3 oss

OSS (Open Sound System) output device.

27.4 sdl

SDL (Simple DirectMedia Layer) output device.

This output devices allows to show a video stream in an SDL window. Only one SDL window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need libsdl installed on your system when configuring your build.

For more information about SDL, check: http://www.libsdl.org/

27.4.1 Options

‘window_title’

Set the SDL window title, if not specified default to the filename specified for the output device.

‘icon_title’

Set the name of the iconified SDL window, if not specified it is set to the same value of *window_title*.

‘window_size’

Set the SDL window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video, downscaled according to the aspect ratio.

27.4.2 Examples

The following command shows the ffmpeg output is an SDL window, forcing its size to the qcif format:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt yuv420p -window_size qcif -f sdl "SDL output"
```

27.5 sndio

sndio audio output device.

28. Resampler Options

The audio resampler supports the following named options.

Options may be set by specifying *-option value* in the FFmpeg tools, *option=value* for the aresample filter, by setting the value explicitly in the SwrContext options or using the ‘libavutil/opt.h’ API for programmatic use.

‘ich, in_channel_count’

Set the number of input channels. Default value is 0. Setting this value is not mandatory if the corresponding channel layout ‘in_channel_layout’ is set.

‘och, out_channel_count’

Set the number of output channels. Default value is 0. Setting this value is not mandatory if the corresponding channel layout ‘out_channel_layout’ is set.

‘uch, used_channel_count’

Set the number of used input channels. Default value is 0. This option is only used for special remapping.

‘`isr, in_sample_rate`’

Set the input sample rate. Default value is 0.

‘`osr, out_sample_rate`’

Set the output sample rate. Default value is 0.

‘`isf, in_sample_fmt`’

Specify the input sample format. It is set by default to `none`.

‘`osf, out_sample_fmt`’

Specify the output sample format. It is set by default to `none`.

‘`tsf, internal_sample_fmt`’

Set the internal sample format. Default value is `none`. This will automatically be chosen when it is not explicitly set.

‘`icl, in_channel_layout`’

Set the input channel layout.

‘`ocl, out_channel_layout`’

Set the output channel layout.

‘`clev, center_mix_level`’

Set the center mix level. It is a value expressed in deciBel, and must be in the interval [-32,32].

‘`slev, surround_mix_level`’

Set the surround mix level. It is a value expressed in deciBel, and must be in the interval [-32,32].

‘`lfe_mix_level`’

Set LFE mix into non LFE level. It is used when there is a LFE input but no LFE output. It is a value expressed in deciBel, and must be in the interval [-32,32].

‘`rmvol, rematrix_volume`’

Set rematrix volume. Default value is 1.0.

‘flags, swr_flags’

Set flags used by the converter. Default value is 0.

It supports the following individual flags:

‘res’

force resampling, this flag forces resampling to be used even when the input and output sample rates match.

‘dither_scale’

Set the dither scale. Default value is 1.

‘dither_method’

Set dither method. Default value is 0.

Supported values:

‘rectangular’

select rectangular dither

‘triangular’

select triangular dither

‘triangular_hp’

select triangular dither with high pass

‘lipshitz’

select lipshitz noise shaping dither

‘shibata’

select shibata noise shaping dither

‘low_shibata’

select low shibata noise shaping dither

‘high_shibata’

select high shibata noise shaping dither

‘f_weighted’

select f-weighted noise shaping dither

‘modified_e_weighted’

select modified-e-weighted noise shaping dither

‘improved_e_weighted’

select improved-e-weighted noise shaping dither

‘resampler’

Set resampling engine. Default value is swr.

Supported values:

‘swr’

select the native SW Resampler; filter options precision and cheby are not applicable in this case.

‘soxr’

select the SoX Resampler (where available); compensation, and filter options filter_size, phase_shift, filter_type & kaiser_beta, are not applicable in this case.

‘filter_size’

For swr only, set resampling filter size, default value is 32.

‘phase_shift’

For swr only, set resampling phase shift, default value is 10, and must be in the interval [0,30].

‘linear_interp’

Use Linear Interpolation if set to 1, default value is 0.

‘cutoff’

Set cutoff frequency (swr: 6dB point; soxr: 0dB point) ratio; must be a float value between 0 and 1. Default value is 0.97 with swr, and 0.91 with soxr (which, with a sample-rate of 44100, preserves the entire audio band to 20kHz).

‘precision’

For soxr only, the precision in bits to which the resampled signal will be calculated. The default value of 20 (which, with suitable dithering, is appropriate for a destination bit-depth of 16) gives SoX's 'High Quality'; a value of 28 gives SoX's 'Very High Quality'.

'cheby'

For soxr only, selects passband rolloff none (Chebyshev) & higher-precision approximation for 'irrational' ratios. Default value is 0.

'async'

For swr only, simple 1 parameter audio sync to timestamps using stretching, squeezing, filling and trimming. Setting this to 1 will enable filling and trimming, larger values represent the maximum amount in samples that the data may be stretched or squeezed for each second. Default value is 0, thus no compensation is applied to make the samples match the audio timestamps.

'first_pts'

For swr only, assume the first pts should be this value. The time unit is 1 / sample rate. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame's expected pts, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream or to trim any samples with a negative pts due to encoder delay.

'min_comp'

For swr only, set the minimum difference between timestamps and audio data (in seconds) to trigger stretching/squeezing/filling or trimming of the data to make it match the timestamps. The default is that stretching/squeezing/filling and trimming is disabled ('min_comp' = FLT_MAX).

'min_hard_comp'

For swr only, set the minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples to make it match the timestamps. This option effectively is a threshold to select between hard (trim/fill) and soft (squeeze/stretch) compensation. Note that all compensation is by default disabled through 'min_comp'. The default is 0.1.

'comp_duration'

For swr only, set duration (in seconds) over which data is stretched/squeezed to make it match the timestamps. Must be a non-negative double float value, default value is 1.0.

'max_soft_comp'

For swr only, set maximum factor by which data is stretched/squeezed to make it match the timestamps. Must be a non-negative double float value, default value is 0.

‘matrix_encoding’

Select matrixed stereo encoding.

It accepts the following values:

‘none’

select none

‘dolby’

select Dolby

‘dplii’

select Dolby Pro Logic II

Default value is none.

‘filter_type’

For swr only, select resampling filter type. This only affects resampling operations.

It accepts the following values:

‘cubic’

select cubic

‘blackman_nuttall’

select Blackman Nuttall Windowed Sinc

‘kaiser’

select Kaiser Windowed Sinc

‘kaiser_beta’

For swr only, set Kaiser Window Beta value. Must be an integer in the interval [2,16], default value is 9.

29. Scaler Options

The video scaler supports the following named options.

Options may be set by specifying *-option value* in the FFmpeg tools. For programmatic use, they can be set explicitly in the `SwsContext` options or through the ‘`libavutil/opt.h`’ API.

‘`sws_flags`’

Set the scaler flags. This is also used to set the scaling algorithm. Only a single algorithm should be selected.

It accepts the following values:

‘`fast_bilinear`’

Select fast bilinear scaling algorithm.

‘`bilinear`’

Select bilinear scaling algorithm.

‘`bicubic`’

Select bicubic scaling algorithm.

‘`experimental`’

Select experimental scaling algorithm.

‘`neighbor`’

Select nearest neighbor rescaling algorithm.

‘`area`’

Select averaging area rescaling algorithm.

‘`bicubiclin`’

Select bicubic scaling algorithm for the luma component, bilinear for chroma components.

‘`gauss`’

Select Gaussian rescaling algorithm.

‘`sinc`’

Select sinc rescaling algorithm.

‘`lanczos`’

Select lanczos rescaling algorithm.

‘spline’

Select natural bicubic spline rescaling algorithm.

‘print_info’

Enable printing/debug logging.

‘accurate_rnd’

Enable accurate rounding.

‘full_chroma_int’

Enable full chroma interpolation.

‘full_chroma_inp’

Select full chroma input.

‘bitexact’

Enable bitexact output.

‘srcw’

Set source width.

‘srch’

Set source height.

‘dstw’

Set destination width.

‘dsth’

Set destination height.

‘src_format’

Set source pixel format (must be expressed as an integer).

‘dst_format’

Set destination pixel format (must be expressed as an integer).

‘src_range’

Select source range.

‘dst_range’

Select destination range.

‘param0, param1’

Set scaling algorithm parameters. The specified values are specific of some scaling algorithms and ignored by others. The specified values are floating point number values.

30. Filtering Introduction

Filtering in FFmpeg is enabled through the libavfilter library.

In libavfilter, a filter can have multiple inputs and multiple outputs. To illustrate the sorts of things that are possible, we consider the following filtergraph.

```
input --> split ----------------------> overlay --> output
            |                             ^
            |                             |
            +-----> crop --> vflip -------+
```

This filtergraph splits the input stream in two streams, sends one stream through the crop filter and the vflip filter before merging it back with the other stream by overlaying it on top. You can use the following command to achieve this:

```
ffmpeg -i INPUT -vf "split [main][tmp]; [tmp] crop=iw:ih/2:0:0, vflip [flip]; [main][flip] overlay=0:H/2" OUTPUT
```

The result will be that in output the top half of the video is mirrored onto the bottom half.

Filters in the same linear chain are separated by commas, and distinct linear chains of filters are separated by semicolons. In our example, *crop,vflip* are in one linear chain, *split* and *overlay* are separately in another. The points where the linear chains join are labelled by names enclosed in square brackets. In the example, the split filter generates two outputs that are associated to the labels *[main]* and *[tmp]*.

The stream sent to the second output of *split*, labelled as *[tmp]*, is processed through the *crop* filter, which crops away the lower half part of the video, and then vertically flipped. The *overlay* filter takes in input the first unchanged output of the split filter (which was labelled as *[main]*), and overlay on its lower half the output generated by the *crop,vflip* filterchain.

Some filters take in input a list of parameters: they are specified after the filter name and an equal sign, and are separated from each other by a colon.

There exist so-called *source filters* that do not have an audio/video input, and *sink filters* that will not have audio/video output.

31. graph2dot

The ‘`graph2dot`’ program included in the FFmpeg ‘`tools`’ directory can be used to parse a filtergraph description and issue a corresponding textual representation in the dot language.

Invoke the command:

```
graph2dot -h
```

to see how to use ‘`graph2dot`’.

You can then pass the dot description to the ‘`dot`’ program (from the graphviz suite of programs) and obtain a graphical representation of the filtergraph.

For example the sequence of commands:

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

can be used to create and display an image representing the graph described by the *GRAPH_DESCRIPTION* string. Note that this string must be a complete self-contained graph, with its inputs and outputs explicitly defined. For example if your command line is of the form:

```
ffmpeg -i infile -vf scale=640:360 outfile
```

your *GRAPH_DESCRIPTION* string will need to be of the form:

```
nullsrc,scale=640:360,nullsink
```

you may also need to set the *nullsrc* parameters and add a *format* filter in order to simulate a specific input file.

32. Filtergraph description

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

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

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

32.1 Filtergraph syntax

A filtergraph can be represented using a textual representation, which is recognized by the ‘`-filter`’/‘`-vf`’ and ‘`-filter_complex`’ options in `ffmpeg` and ‘`-vf`’ in `ffplay`, and by the `avfilter_graph_parse()`/`avfilter_graph_parse2()` function defined in ‘`libavfilter/avfilter.h`’.

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

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

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

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

arguments is a string which contains the parameters used to initialize the filter instance. It may have one of the following forms:

- A ’:’-separated list of *key=value* pairs.
- A ’:’-separated list of *value*. In this case, the keys are assumed to be the option names in the order they are declared. E.g. the `fade` filter declares three options in this order – ‘`type`’, ‘`start_frame`’ and ‘`nb_frames`’. Then the parameter list *in:0:30* means that the value *in* is assigned to the option ‘`type`’, *0* to ‘`start_frame`’ and *30* to ‘`nb_frames`’.
- A ’:’-separated list of mixed direct *value* and long *key=value* pairs. The direct *value* must precede the *key=value* pairs, and follow the same constraints order of the previous point. The following *key=value* pairs can be set in any preferred order.

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

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

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

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

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

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

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

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

Libavfilter will automatically insert scale filters where format conversion is required. It is possible to specify swscale flags for those automatically inserted scalers by prepending `sws_flags=`*`flags`*`;` to the filtergraph description.

Follows a BNF description for the filtergraph syntax:

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

32.2 Notes on filtergraph escaping

Some filter arguments require the use of special characters, typically `:` to separate key=value pairs in a named options list. In this case the user should perform a first level escaping when specifying the filter arguments. For example, consider the following literal string to be embedded in the drawtext filter arguments:

```
this is a 'string': may contain one, or more, special characters
```

Since : is special for the filter arguments syntax, it needs to be escaped, so you get:

```
text=this is a \'string\'\: may contain one, or more, special characters
```

A second level of escaping is required when embedding the filter arguments in a filtergraph description, in order to escape all the filtergraph special characters. Thus the example above becomes:

```
drawtext=text=this is a \\\'string\\\'\\: may contain one\, or more\, special characters
```

Finally an additional level of escaping may be needed when writing the filtergraph description in a shell command, which depends on the escaping rules of the adopted shell. For example, assuming that \ is special and needs to be escaped with another \, the previous string will finally result in:

```
-vf "drawtext=text=this is a \\\\\\'string\\\\\\'\\\\: may contain one\\, or more\\, special characters"
```

Sometimes, it might be more convenient to employ quoting in place of escaping. For example the string:

```
Caesar: tu quoque, Brute, fili mi
```

Can be quoted in the filter arguments as:

```
text='Caesar: tu quoque, Brute, fili mi'
```

And finally inserted in a filtergraph like:

```
drawtext=text=\'Caesar: tu quoque\, Brute\, fili mi\'
```

See the “Quoting and escaping” section in the ffmpeg-utils manual for more information about the escaping and quoting rules adopted by FFmpeg.

33. Timeline editing

Some filters support a generic ‘`enable`’ option. For the filters supporting timeline editing, this option can be set to an expression which is evaluated before sending a frame to the filter. If the evaluation is non-zero, the filter will be enabled, otherwise the frame will be sent unchanged to the next filter in the filtergraph.

The expression accepts the following values:

‘`t`’

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

‘n’

sequential number of the input frame, starting from 0

‘pos’

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

Additionally, these filters support an ‘enable’ command that can be used to re-define the expression.

Like any other filtering option, the ‘enable’ option follows the same rules.

For example, to enable a blur filter (smartblur) from 10 seconds to 3 minutes, and a curves filter starting at 3 seconds:

```
smartblur = enable='between(t,10,3*60)',
curves    = enable='gte(t,3)' : preset=cross_process
```

34. Audio Filters

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

Below is a description of the currently available audio filters.

34.1 aconvert

Convert the input audio format to the specified formats.

This filter is deprecated. Use aformat instead.

The filter accepts a string of the form: "*sample_format*:*channel_layout*".

sample_format specifies the sample format, and can be a string or the corresponding numeric value defined in ‘libavutil/samplefmt.h’. Use ’p’ suffix for a planar sample format.

channel_layout specifies the channel layout, and can be a string or the corresponding number value defined in ‘libavutil/channel_layout.h’.

The special parameter "auto", signifies that the filter will automatically select the output format depending on the output filter.

34.1.1 Examples

- Convert input to float, planar, stereo:

```
aconvert=fltp:stereo
```

- Convert input to unsigned 8-bit, automatically select out channel layout:

```
aconvert=u8:auto
```

34.2 allpass

Apply a two-pole all-pass filter with central frequency (in Hz) *frequency*, and filter-width *width*. An all-pass filter changes the audio's frequency to phase relationship without changing its frequency to amplitude relationship.

The filter accepts the following options:

'`frequency, f`'

Set frequency in Hz.

'`width_type`'

Set method to specify band-width of filter.

'`h`'

Hz

'`q`'

Q-Factor

'`o`'

octave

'`s`'

slope

'`width, w`'

Specify the band-width of a filter in width_type units.

34.3 highpass

Apply a high-pass filter with 3dB point frequency. The filter can be either single-pole, or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

‘`frequency, f`’

Set frequency in Hz. Default is 3000.

‘`poles, p`’

Set number of poles. Default is 2.

‘`width_type`’

Set method to specify band-width of filter.

‘`h`’

Hz

‘`q`’

Q-Factor

‘`o`’

octave

‘`s`’

slope

‘`width, w`’

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

34.4 lowpass

Apply a low-pass filter with 3dB point frequency. The filter can be either single-pole or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

‘frequency, f’

Set frequency in Hz. Default is 500.

‘poles, p’

Set number of poles. Default is 2.

‘width_type’

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

34.5 bass

Boost or cut the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi’s tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

‘gain, g’

Give the gain at 0 Hz. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

‘frequency, f’

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `100` Hz.

'`width_type`'

Set method to specify band-width of filter.

'`h`'

Hz

'`q`'

Q-Factor

'`o`'

octave

'`s`'

slope

'`width, w`'

Determine how steep is the filter's shelf transition.

34.6 treble

Boost or cut treble (upper) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

'`gain, g`'

Give the gain at whichever is the lower of ~22 kHz and the Nyquist frequency. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

'`frequency, f`'

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `3000` Hz.

'`width_type`'

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Determine how steep is the filter’s shelf transition.

34.7 bandpass

Apply a two-pole Butterworth band-pass filter with central frequency *frequency*, and (3dB-point) band-width width. The *csg* option selects a constant skirt gain (peak gain = Q) instead of the default: constant 0dB peak gain. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

‘frequency, f’

Set the filter’s central frequency. Default is 3000.

‘csg’

Constant skirt gain if set to 1. Defaults to 0.

‘width_type’

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Specify the band-width of a filter in width_type units.

34.8 bandreject

Apply a two-pole Butterworth band-reject filter with central frequency *frequency*, and (3dB-point) band-width *width*. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

‘frequency, f’

Set the filter’s central frequency. Default is `3000`.

‘width_type’

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Specify the band-width of a filter in width_type units.

34.9 biquad

Apply a biquad IIR filter with the given coefficients. Where *b0*, *b1*, *b2* and *a0*, *a1*, *a2* are the numerator and denominator coefficients respectively.

34.10 equalizer

Apply a two-pole peaking equalisation (EQ) filter. With this filter, the signal-level at and around a selected frequency can be increased or decreased, whilst (unlike bandpass and bandreject filters) that at all other frequencies is unchanged.

In order to produce complex equalisation curves, this filter can be given several times, each with a different central frequency.

The filter accepts the following options:

‘`frequency, f`’

Set the filter’s central frequency in Hz.

‘`width_type`’

Set method to specify band-width of filter.

‘`h`’

Hz

‘`q`’

Q-Factor

‘`o`’

octave

‘`s`’

slope

‘`width, w`’

Specify the band-width of a filter in width_type units.

‘`gain, g`’

Set the required gain or attenuation in dB. Beware of clipping when using a positive gain.

34.11 afade

Apply fade-in/out effect to input audio.

A description of the accepted parameters follows.

‘type, t’

Specify the effect type, can be either in for fade-in, or out for a fade-out effect. Default is in.

‘start_sample, ss’

Specify the number of the start sample for starting to apply the fade effect. Default is 0.

‘nb_samples, ns’

Specify the number of samples for which the fade effect has to last. At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. Default is 44100.

‘start_time, st’

Specify time for starting to apply the fade effect. Default is 0. The accepted syntax is:

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

See also the function av_parse_time(). If set this option is used instead of *start_sample* one.

‘duration, d’

Specify the duration for which the fade effect has to last. Default is 0. The accepted syntax is:

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

See also the function av_parse_time(). At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. If set this option is used instead of *nb_samples* one.

‘curve’

Set curve for fade transition.

It accepts the following values:

‘`tri`’

select triangular, linear slope (default)

‘`qsin`’

select quarter of sine wave

‘`hsin`’

select half of sine wave

‘`esin`’

select exponential sine wave

‘`log`’

select logarithmic

‘`par`’

select inverted parabola

‘`qua`’

select quadratic

‘`cub`’

select cubic

‘`squ`’

select square root

‘`cbr`’

select cubic root

34.11.1 Examples

- Fade in first 15 seconds of audio:

```
afade=t=in:ss=0:d=15
```

- Fade out last 25 seconds of a 900 seconds audio:

```
afade=t=out:st=875:d=25
```

34.12 aformat

Set output format constraints for the input audio. The framework will negotiate the most appropriate format to minimize conversions.

The filter accepts the following named parameters:

‘`sample_fmts`’

A ’|’-separated list of requested sample formats.

‘`sample_rates`’

A ’|’-separated list of requested sample rates.

‘`channel_layouts`’

A ’|’-separated list of requested channel layouts.

If a parameter is omitted, all values are allowed.

For example to force the output to either unsigned 8-bit or signed 16-bit stereo:

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

34.13 amerge

Merge two or more audio streams into a single multi-channel stream.

The filter accepts the following options:

‘`inputs`’

Set the number of inputs. Default is 2.

If the channel layouts of the inputs are disjoint, and therefore compatible, the channel layout of the output will be set accordingly and the channels will be reordered as necessary. If the channel layouts of the inputs are not disjoint, the output will have all the channels of the first input then all the channels of the second input, in that order, and the channel layout of the output will be the default value corresponding to the total number of channels.

For example, if the first input is in 2.1 (FL+FR+LF) and the second input is FC+BL+BR, then the output will be in 5.1, with the channels in the following order: a1, a2, b1, a3, b2, b3 (a1 is the first channel of the first input, b1 is the first channel of the second input).

On the other hand, if both input are in stereo, the output channels will be in the default order: a1, a2, b1, b2, and the channel layout will be arbitrarily set to 4.0, which may or may not be the expected value.

All inputs must have the same sample rate, and format.

If inputs do not have the same duration, the output will stop with the shortest.

34.13.1 Examples

- Merge two mono files into a stereo stream:

```
amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
```

- Multiple merges assuming 1 video stream and 6 audio streams in ‘`input.mkv`’:

```
ffmpeg -i input.mkv -filter_complex "[0:1][0:2][0:3][0:4][0:5][0:6] amerge=inputs=6" -c:a pcm_s16le output.mkv
```

34.14 amix

Mixes multiple audio inputs into a single output.

For example

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

will mix 3 input audio streams to a single output with the same duration as the first input and a dropout transition time of 3 seconds.

The filter accepts the following named parameters:

‘`inputs`’

Number of inputs. If unspecified, it defaults to 2.

‘`duration`’

How to determine the end-of-stream.

‘`longest`’

Duration of longest input. (default)

‘shortest’

Duration of shortest input.

‘first’

Duration of first input.

‘dropout_transition’

Transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

34.15 anull

Pass the audio source unchanged to the output.

34.16 apad

Pad the end of a audio stream with silence, this can be used together with -shortest to extend audio streams to the same length as the video stream.

34.17 aphaser

Add a phasing effect to the input audio.

A phaser filter creates series of peaks and troughs in the frequency spectrum. The position of the peaks and troughs are modulated so that they vary over time, creating a sweeping effect.

A description of the accepted parameters follows.

‘in_gain’

Set input gain. Default is 0.4.

‘out_gain’

Set output gain. Default is 0.74

‘delay’

Set delay in milliseconds. Default is 3.0.

‘decay’

Set decay. Default is 0.4.

‘speed’

Set modulation speed in Hz. Default is 0.5.

‘type’

Set modulation type. Default is triangular.

It accepts the following values:

‘triangular, t’
‘sinusoidal, s’

34.18 aresample

Resample the input audio to the specified parameters, using the libswresample library. If none are specified then the filter will automatically convert between its input and output.

This filter is also able to stretch/squeeze the audio data to make it match the timestamps or to inject silence / cut out audio to make it match the timestamps, do a combination of both or do neither.

The filter accepts the syntax [*sample_rate*:]*resampler_options*, where *sample_rate* expresses a sample rate and *resampler_options* is a list of *key*=*value* pairs, separated by ":". See the ffmpeg-resampler manual for the complete list of supported options.

34.18.1 Examples

- Resample the input audio to 44100Hz:

```
aresample=44100
```

- Stretch/squeeze samples to the given timestamps, with a maximum of 1000 samples per second compensation:

```
aresample=async=1000
```

34.19 asetnsamples

Set the number of samples per each output audio frame.

The last output packet may contain a different number of samples, as the filter will flush all the remaining samples when the input audio signal its end.

The filter accepts the following options:

‘nb_out_samples, n’

Set the number of frames per each output audio frame. The number is intended as the number of samples *per each channel*. Default value is 1024.

‘pad, p’

If set to 1, the filter will pad the last audio frame with zeroes, so that the last frame will contain the same number of samples as the previous ones. Default value is 1.

For example, to set the number of per-frame samples to 1234 and disable padding for the last frame, use:

```
asetnsamples=n=1234:p=0
```

34.20 asetpts

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

This filter accepts the following options:

‘expr’

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

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

‘PTS’

the presentation timestamp in input

‘PI’

Greek PI

‘PHI’

golden ratio

‘E’

Euler number

‘N’

Number of the audio samples pass through the filter so far, starting at 0.

‘S’

Number of the audio samples in the current frame.

‘SR’

Audio sample rate.

‘STARTPTS’

the PTS of the first frame

‘PREV_INPTS’

previous input PTS

‘PREV_OUTPTS’

previous output PTS

‘RTCTIME’

wallclock (RTC) time in microseconds

‘RTCSTART’

wallclock (RTC) time at the start of the movie in microseconds

Some examples follow:

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

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

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

34.21 asetrate

Set the sample rate without altering the PCM data. This will result in a change of speed and pitch.

The filter accepts the following options:

‘sample_rate, r’

Set the output sample rate. Default is 44100 Hz.

34.22 ashowinfo

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

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

A description of each shown parameter follows:

‘n’

sequential number of the input frame, starting from 0

‘pts’

Presentation timestamp of the input frame, in time base units; the time base depends on the filter input pad, and is usually 1/*sample_rate*.

‘pts_time’

presentation timestamp of the input frame in seconds

‘pos’

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

‘fmt’

sample format

‘chlayout’

channel layout

‘rate’

sample rate for the audio frame

‘nb_samples’

number of samples (per channel) in the frame

‘checksum’

Adler-32 checksum (printed in hexadecimal) of the audio data. For planar audio the data is treated as if all the planes were concatenated.

‘plane_checksums’

A list of Adler-32 checksums for each data plane.

34.23 astats

Display time domain statistical information about the audio channels. Statistics are calculated and displayed for each audio channel and, where applicable, an overall figure is also given.

The filter accepts the following option:

‘length’

Short window length in seconds, used for peak and trough RMS measurement. Default is `0.05` (50 miliseconds). Allowed range is `[0.1 - 10]`.

A description of each shown parameter follows:

‘DC offset’

Mean amplitude displacement from zero.

‘Min level’

Minimal sample level.

‘Max level’

Maximal sample level.

‘Peak level dB’
‘RMS level dB’

Standard peak and RMS level measured in dBFS.

‘RMS peak dB’
‘RMS trough dB’

Peak and trough values for RMS level measured over a short window.

‘Crest factor’

Standard ratio of peak to RMS level (note: not in dB).

‘Flat factor’

Flatness (i.e. consecutive samples with the same value) of the signal at its peak levels (i.e. either *Min level* or *Max level*).

‘Peak count’

Number of occasions (not the number of samples) that the signal attained either *Min level* or *Max level*.

34.24 astreamsync

Forward two audio streams and control the order the buffers are forwarded.

The filter accepts the following options:

‘expr, e’

Set the expression deciding which stream should be forwarded next: if the result is negative, the first stream is forwarded; if the result is positive or zero, the second stream is forwarded. It can use the following variables:

b1 b2

number of buffers forwarded so far on each stream

s1 s2

number of samples forwarded so far on each stream

t1 t2

current timestamp of each stream

The default value is `t1-t2`, which means to always forward the stream that has a smaller timestamp.

34.24.1 Examples

Stress-test `amerge` by randomly sending buffers on the wrong input, while avoiding too much of a desynchronization:

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

34.25 atempo

Adjust audio tempo.

The filter accepts exactly one parameter, the audio tempo. If not specified then the filter will assume nominal 1.0 tempo. Tempo must be in the [0.5, 2.0] range.

34.25.1 Examples

- Slow down audio to 80% tempo:

```
atempo=0.8
```

- To speed up audio to 125% tempo:

```
atempo=1.25
```

34.26 earwax

Make audio easier to listen to on headphones.

This filter adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Ported from SoX.

34.27 pan

Mix channels with specific gain levels. The filter accepts the output channel layout followed by a set of channels definitions.

This filter is also designed to remap efficiently the channels of an audio stream.

The filter accepts parameters of the form: "*l*:*outdef*:*outdef*:..."

‘`l`’

output channel layout or number of channels

‘`outdef`’

output channel specification, of the form: "*out_name*=[*gain**]*in_name*[+[*gain**]*in_name*...]"

‘out_name’

output channel to define, either a channel name (FL, FR, etc.) or a channel number (c0, c1, etc.)

‘gain’

multiplicative coefficient for the channel, 1 leaving the volume unchanged

‘in_name’

input channel to use, see out_name for details; it is not possible to mix named and numbered input channels

If the ‘=’ in a channel specification is replaced by ‘<’, then the gains for that specification will be renormalized so that the total is 1, thus avoiding clipping noise.

34.27.1 Mixing examples

For example, if you want to down-mix from stereo to mono, but with a bigger factor for the left channel:

```
pan=1:c0=0.9*c0+0.1*c1
```

A customized down-mix to stereo that works automatically for 3-, 4-, 5- and 7-channels surround:

```
pan=stereo: FL < FL + 0.5*FC + 0.6*BL + 0.6*SL : FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

Note that `ffmpeg` integrates a default down-mix (and up-mix) system that should be preferred (see "-ac" option) unless you have very specific needs.

34.27.2 Remapping examples

The channel remapping will be effective if, and only if:

- gain coefficients are zeroes or ones,
- only one input per channel output,

If all these conditions are satisfied, the filter will notify the user ("Pure channel mapping detected"), and use an optimized and lossless method to do the remapping.

For example, if you have a 5.1 source and want a stereo audio stream by dropping the extra channels:

```
pan="stereo: c0=FL : c1=FR"
```

Given the same source, you can also switch front left and front right channels and keep the input channel layout:

```
pan="5.1: c0=c1 : c1=c0 : c2=c2 : c3=c3 : c4=c4 : c5=c5"
```

If the input is a stereo audio stream, you can mute the front left channel (and still keep the stereo channel layout) with:

```
pan="stereo:c1=c1"
```

Still with a stereo audio stream input, you can copy the right channel in both front left and right:

```
pan="stereo: c0=FR : c1=FR"
```

34.28 silencedetect

Detect silence in an audio stream.

This filter logs a message when it detects that the input audio volume is less or equal to a noise tolerance value for a duration greater or equal to the minimum detected noise duration.

The printed times and duration are expressed in seconds.

The filter accepts the following options:

‘`duration, d`’

Set silence duration until notification (default is 2 seconds).

‘`noise, n`’

Set noise tolerance. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default is -60dB, or 0.001.

34.28.1 Examples

- Detect 5 seconds of silence with -50dB noise tolerance:

  ```
  silencedetect=n=-50dB:d=5
  ```

- Complete example with `ffmpeg` to detect silence with 0.0001 noise tolerance in ‘`silence.mp3`’:

  ```
  ffmpeg -i silence.mp3 -af silencedetect=noise=0.0001 -f null -
  ```

34.29 asyncts

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

This filter is not built by default, please use aresample to do squeezing/stretching.

The filter accepts the following named parameters:

‘`compensate`’

Enable stretching/squeezing the data to make it match the timestamps. Disabled by default. When disabled, time gaps are covered with silence.

‘`min_delta`’

Minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. Default value is 0.1. If you get non-perfect sync with this filter, try setting this parameter to 0.

‘`max_comp`’

Maximum compensation in samples per second. Relevant only with compensate=1. Default value 500.

‘`first_pts`’

Assume the first pts should be this value. The time base is 1 / sample rate. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame’s expected pts, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream or to trim any samples with a negative pts due to encoder delay.

34.30 atrim

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

This filter accepts the following options:

‘`start`’

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

‘`end`’

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

‘`start_pts`’

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

‘`end_pts`’

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

‘`duration`’

Maximum duration of the output in seconds.

‘`start_sample`’

Number of the first sample that should be passed to output.

‘`end_sample`’

Number of the first sample that should be dropped.

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

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

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

Examples:

- drop everything except the second minute of input

```
ffmpeg -i INPUT -af atrim=60:120
```

- keep only the first 1000 samples

```
ffmpeg -i INPUT -af atrim=end_sample=1000
```

34.31 channelsplit

Split each channel in input audio stream into a separate output stream.

This filter accepts the following named parameters:

‘channel_layout’

Channel layout of the input stream. Default is "stereo".

For example, assuming a stereo input MP3 file

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

will create an output Matroska file with two audio streams, one containing only the left channel and the other the right channel.

To split a 5.1 WAV file into per-channel files

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

34.32 channelmap

Remap input channels to new locations.

This filter accepts the following named parameters:

‘channel_layout’

Channel layout of the output stream.

‘map’

Map channels from input to output. The argument is a '|'-separated list of mappings, each in the *in_channel*-*out_channel* or *in_channel* form. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the input channel layout. *out_channel* is the name of the output channel or its index in the output channel layout. If *out_channel* is not given then it is implicitly an index, starting with zero and increasing by one for each mapping.

If no mapping is present, the filter will implicitly map input channels to output channels preserving index.

For example, assuming a 5.1+downmix input MOV file

```
ffmpeg -i in.mov -filter 'channelmap=map=DL-FL|DR-FR' out.wav
```

will create an output WAV file tagged as stereo from the downmix channels of the input.

To fix a 5.1 WAV improperly encoded in AAC's native channel order

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

34.33 join

Join multiple input streams into one multi-channel stream.

The filter accepts the following named parameters:

'`inputs`'

Number of input streams. Defaults to 2.

'`channel_layout`'

Desired output channel layout. Defaults to stereo.

'`map`'

Map channels from inputs to output. The argument is a '|'-separated list of mappings, each in the *`input_idx.in_channel-out_channel`* form. *input_idx* is the 0-based index of the input stream. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the specified input stream. *out_channel* is the name of the output channel.

The filter will attempt to guess the mappings when those are not specified explicitly. It does so by first trying to find an unused matching input channel and if that fails it picks the first unused input channel.

E.g. to join 3 inputs (with properly set channel layouts)

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

To build a 5.1 output from 6 single-channel streams:

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex
'join=inputs=6:channel_layout=5.1:map=0.0-FL|1.0-FR|2.0-FC|3.0-SL|4.0-SR|5.0-LFE'
out
```

34.34 resample

Convert the audio sample format, sample rate and channel layout. This filter is not meant to be used directly.

34.35 volume

Adjust the input audio volume.

The filter accepts the following options:

‘`volume`’

Expresses how the audio volume will be increased or decreased.

Output values are clipped to the maximum value.

The output audio volume is given by the relation:

```
output_volume = volume * input_volume
```

Default value for *volume* is 1.0.

‘`precision`’

Set the mathematical precision.

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

‘`fixed`’

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

‘`float`’

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

‘`double`’

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

34.35.1 Examples

- Halve the input audio volume:

```
volume=volume=0.5
volume=volume=1/2
volume=volume=-6.0206dB
```

In all the above example the named key for ‘`volume`’ can be omitted, for example like in:

```
volume=0.5
```

- Increase input audio power by 6 decibels using fixed-point precision:

```
volume=volume=6dB:precision=fixed
```

34.36 volumedetect

Detect the volume of the input video.

The filter has no parameters. The input is not modified. Statistics about the volume will be printed in the log when the input stream end is reached.

In particular it will show the mean volume (root mean square), maximum volume (on a per-sample basis), and the beginning of an histogram of the registered volume values (from the maximum value to a cumulated 1/1000 of the samples).

All volumes are in decibels relative to the maximum PCM value.

34.36.1 Examples

Here is an excerpt of the output:

```
[Parsed_volumedetect_0  0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0  0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0  0xa23120] histogram_4db: 6
[Parsed_volumedetect_0  0xa23120] histogram_5db: 62
[Parsed_volumedetect_0  0xa23120] histogram_6db: 286
[Parsed_volumedetect_0  0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0  0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0  0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0  0xa23120] histogram_10db: 8409
```

It means that:

- The mean square energy is approximately -27 dB, or 10^-2.7.
- The largest sample is at -4 dB, or more precisely between -4 dB and -5 dB.
- There are 6 samples at -4 dB, 62 at -5 dB, 286 at -6 dB, etc.

In other words, raising the volume by +4 dB does not cause any clipping, raising it by +5 dB causes clipping for 6 samples, etc.

35. Audio Sources

Below is a description of the currently available audio sources.

35.1 abuffer

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

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

It accepts the following named parameters:

‘`time_base`’

Timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

‘`sample_rate`’

The sample rate of the incoming audio buffers.

‘`sample_fmt`’

The sample format of the incoming audio buffers. Either a sample format name or its corresponging integer representation from the enum AVSampleFormat in ‘`libavutil/samplefmt.h`’

‘`channel_layout`’

The channel layout of the incoming audio buffers. Either a channel layout name from channel_layout_map in ‘`libavutil/channel_layout.c`’ or its corresponding integer representation from the AV_CH_LAYOUT_* macros in ‘`libavutil/channel_layout.h`’

‘`channels`’

The number of channels of the incoming audio buffers. If both *channels* and *channel_layout* are specified, then they must be consistent.

35.1.1 Examples

```
abuffer=sample_rate=44100:sample_fmt=s16p:channel_layout=stereo
```

will instruct the source to accept planar 16bit signed stereo at 44100Hz. Since the sample format with name "s16p" corresponds to the number 6 and the "stereo" channel layout corresponds to the value 0x3, this is equivalent to:

```
abuffer=sample_rate=44100:sample_fmt=6:channel_layout=0x3
```

35.2 aevalsrc

Generate an audio signal specified by an expression.

This source accepts in input one or more expressions (one for each channel), which are evaluated and used to generate a corresponding audio signal.

This source accepts the following options:

‘`exprs`’

Set the ’|’-separated expressions list for each separate channel. In case the ‘`channel_layout`’ option is not specified, the selected channel layout depends on the number of provided expressions.

‘`channel_layout, c`’

Set the channel layout. The number of channels in the specified layout must be equal to the number of specified expressions.

‘`duration, d`’

Set the minimum duration of the sourced audio. See the function `av_parse_time()` for the accepted format. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame.

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

‘`nb_samples, n`’

Set the number of samples per channel per each output frame, default to 1024.

‘`sample_rate, s`’

Specify the sample rate, default to 44100.

Each expression in *exprs* can contain the following constants:

‘`n`’

number of the evaluated sample, starting from 0

‘t’

time of the evaluated sample expressed in seconds, starting from 0

‘s’

sample rate

35.2.1 Examples

- Generate silence:

```
aevalsrc=0
```

- Generate a sin signal with frequency of 440 Hz, set sample rate to 8000 Hz:

```
aevalsrc="sin(440*2*PI*t):s=8000"
```

- Generate a two channels signal, specify the channel layout (Front Center + Back Center) explicitly:

```
aevalsrc="sin(420*2*PI*t)|cos(430*2*PI*t):c=FC|BC"
```

- Generate white noise:

```
aevalsrc="-2+random(0)"
```

- Generate an amplitude modulated signal:

```
aevalsrc="sin(10*2*PI*t)*sin(880*2*PI*t)"
```

- Generate 2.5 Hz binaural beats on a 360 Hz carrier:

```
aevalsrc="0.1*sin(2*PI*(360-2.5/2)*t) | 0.1*sin(2*PI*(360+2.5/2)*t)"
```

35.3 anullsrc

Null audio source, return unprocessed audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools, or as the source for filters which ignore the input data (for example the sox synth filter).

This source accepts the following options:

‘`channel_layout, cl`’

Specify the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is "stereo".

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

‘`sample_rate, r`’

Specify the sample rate, and defaults to 44100.

‘`nb_samples, n`’

Set the number of samples per requested frames.

35.3.1 Examples

- Set the sample rate to 48000 Hz and the channel layout to AV_CH_LAYOUT_MONO.

```
anullsrc=r=48000:cl=4
```

- Do the same operation with a more obvious syntax:

```
anullsrc=r=48000:cl=mono
```

35.4 abuffer

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

This source is not intended to be part of user-supplied graph descriptions but for insertion by calling programs through the interface defined in ‘`libavfilter/buffersrc.h`’.

It accepts the following named parameters:

‘`time_base`’

Timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

‘`sample_rate`’

Audio sample rate.

‘`sample_fmt`’

Name of the sample format, as returned by `av_get_sample_fmt_name()`.

‘`channel_layout`’

Channel layout of the audio data, in the form that can be accepted by `av_get_channel_layout()`.

All the parameters need to be explicitly defined.

35.5 flite

Synthesize a voice utterance using the libflite library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libflite`.

Note that the flite library is not thread-safe.

The filter accepts the following options:

‘`list_voices`’

If set to 1, list the names of the available voices and exit immediately. Default value is 0.

‘`nb_samples, n`’

Set the maximum number of samples per frame. Default value is 512.

‘`textfile`’

Set the filename containing the text to speak.

‘`text`’

Set the text to speak.

‘`voice, v`’

Set the voice to use for the speech synthesis. Default value is `kal`. See also the *list_voices* option.

35.5.1 Examples

- Read from file ‘`speech.txt`’, and synthetize the text using the standard flite voice:

```
flite=textfile=speech.txt
```

- Read the specified text selecting the `slt` voice:

```
flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
```

- Input text to ffmpeg:

```
ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
```

- Make '`ffplay`' speak the specified text, using `flite` and the `lavfi` device:

```
ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
```

For more information about libflite, check: http://www.speech.cs.cmu.edu/flite/

35.6 sine

Generate an audio signal made of a sine wave with amplitude 1/8.

The audio signal is bit-exact.

The filter accepts the following options:

'`frequency, f`'

Set the carrier frequency. Default is 440 Hz.

'`beep_factor, b`'

Enable a periodic beep every second with frequency *beep_factor* times the carrier frequency. Default is 0, meaning the beep is disabled.

'`sample_rate, s`'

Specify the sample rate, default is 44100.

'`duration, d`'

Specify the duration of the generated audio stream.

'`samples_per_frame`'

Set the number of samples per output frame, default is 1024.

35.6.1 Examples

- Generate a simple 440 Hz sine wave:

```
sine
```

- Generate a 220 Hz sine wave with a 880 Hz beep each second, for 5 seconds:

```
sine=220:4:d=5
sine=f=220:b=4:d=5
sine=frequency=220:beep_factor=4:duration=5
```

36. Audio Sinks

Below is a description of the currently available audio sinks.

36.1 abuffersink

Buffer audio frames, and make them available to the end of filter chain.

This sink is mainly intended for programmatic use, in particular through the interface defined in ‘`libavfilter/buffersink.h`’ or the options system.

It accepts a pointer to an AVABufferSinkContext structure, which defines the incoming buffers’ formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

36.2 anullsink

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

37. Video Filters

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

Below is a description of the currently available video filters.

37.1 alphaextract

Extract the alpha component from the input as a grayscale video. This is especially useful with the *alphamerge* filter.

37.2 alphamerge

Add or replace the alpha component of the primary input with the grayscale value of a second input. This is intended for use with *alphaextract* to allow the transmission or storage of frame sequences that have alpha in a format that doesn’t support an alpha channel.

For example, to reconstruct full frames from a normal YUV-encoded video and a separate video created with *alphaextract*, you might use:

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

Since this filter is designed for reconstruction, it operates on frame sequences without considering timestamps, and terminates when either input reaches end of stream. This will cause problems if your encoding pipeline drops frames. If you're trying to apply an image as an overlay to a video stream, consider the *overlay* filter instead.

37.3 ass

Same as the subtitles filter, except that it doesn't require libavcodec and libavformat to work. On the other hand, it is limited to ASS (Advanced Substation Alpha) subtitles files.

37.4 bbox

Compute the bounding box for the non-black pixels in the input frame luminance plane.

This filter computes the bounding box containing all the pixels with a luminance value greater than the minimum allowed value. The parameters describing the bounding box are printed on the filter log.

37.5 blackdetect

Detect video intervals that are (almost) completely black. Can be useful to detect chapter transitions, commercials, or invalid recordings. Output lines contains the time for the start, end and duration of the detected black interval expressed in seconds.

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

The filter accepts the following options:

'`black_min_duration, d`'

Set the minimum detected black duration expressed in seconds. It must be a non-negative floating point number.

Default value is 2.0.

'`picture_black_ratio_th, pic_th`'

Set the threshold for considering a picture "black". Express the minimum value for the ratio:

```
nb_black_pixels / nb_pixels
```

for which a picture is considered black. Default value is 0.98.

‘pixel_black_th, pix_th’

Set the threshold for considering a pixel "black".

The threshold expresses the maximum pixel luminance value for which a pixel is considered "black". The provided value is scaled according to the following equation:

```
absolute_threshold = luminance_minimum_value + pixel_black_th * luminance_range_size
```

luminance_range_size and *luminance_minimum_value* depend on the input video format, the range is [0-255] for YUV full-range formats and [16-235] for YUV non full-range formats.

Default value is 0.10.

The following example sets the maximum pixel threshold to the minimum value, and detects only black intervals of 2 or more seconds:

```
blackdetect=d=2:pix_th=0.00
```

37.6 blackframe

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

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

The filter accepts the following options:

‘amount’

Set the percentage of the pixels that have to be below the threshold, defaults to 98.

‘threshold, thresh’

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

37.7 blend

Blend two video frames into each other.

It takes two input streams and outputs one stream, the first input is the "top" layer and second input is "bottom" layer. Output terminates when shortest input terminates.

A description of the accepted options follows.

‘c0_mode’
‘c1_mode’
‘c2_mode’
‘c3_mode’
‘all_mode’

Set blend mode for specific pixel component or all pixel components in case of *all_mode*. Default value is normal.

Available values for component modes are:

‘addition’
‘and’
‘average’
‘burn’
‘darken’
‘difference’
‘divide’
‘dodge’
‘exclusion’
‘hardlight’
‘lighten’
‘multiply’
‘negation’
‘normal’
‘or’
‘overlay’
‘phoenix’
‘pinlight’
‘reflect’
‘screen’
‘softlight’
‘subtract’
‘vividlight’
‘xor’

‘c0_opacity’
‘c1_opacity’
‘c2_opacity’
‘c3_opacity’
‘all_opacity’

Set blend opacity for specific pixel component or all pixel components in case of *all_opacity*. Only used in combination with pixel component blend modes.

‘c0_expr’
‘c1_expr’
‘c2_expr’
‘c3_expr’
‘all_expr’

Set blend expression for specific pixel component or all pixel components in case of *all_expr*. Note that related mode options will be ignored if those are set.

The expressions can use the following variables:

‘N’

The sequential number of the filtered frame, starting from 0.

‘X’
‘Y’

the coordinates of the current sample

‘W’
‘H’

the width and height of currently filtered plane

‘SW’
‘SH’

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are 1,1 for the luma plane, and 0.5,0.5 for chroma planes.

‘T’

Time of the current frame, expressed in seconds.

‘TOP, A’

Value of pixel component at current location for first video frame (top layer).

‘BOTTOM, B’

Value of pixel component at current location for second video frame (bottom layer).

37.7.1 Examples

- Apply transition from bottom layer to top layer in first 10 seconds:

```
blend=all_expr='A*(if(gte(T,10),1,T/10))+B*(1-(if(gte(T,10),1,T/10)))'
```

- Apply 1x1 checkerboard effect:

```
blend=all_expr='if(eq(mod(X,2),mod(Y,2)),A,B)'
```

37.8 boxblur

Apply boxblur algorithm to the input video.

The filter accepts the following options:

‘`luma_radius, lr`’
‘`luma_power, lp`’
‘`chroma_radius, cr`’
‘`chroma_power, cp`’
‘`alpha_radius, ar`’
‘`alpha_power, ap`’

A description of the accepted options follows.

‘`luma_radius, lr`’
‘`chroma_radius, cr`’
‘`alpha_radius, ar`’

Set an expression for the box radius in pixels used for blurring the corresponding input plane.

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

Default value for ‘`luma_radius`’ is "2". If not specified, ‘`chroma_radius`’ and ‘`alpha_radius`’ default to the corresponding value set for ‘`luma_radius`’.

The expressions can contain the following constants:

‘`w, h`’

the input width and height in pixels

‘`cw, ch`’

the input chroma image width and height in pixels

‘hsub, vsub’

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

‘luma_power, lp’
‘chroma_power, cp’
‘alpha_power, ap’

Specify how many times the boxblur filter is applied to the corresponding plane.

Default value for ‘luma_power’ is 2. If not specified, ‘chroma_power’ and ‘alpha_power’ default to the corresponding value set for ‘luma_power’.

A value of 0 will disable the effect.

37.8.1 Examples

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

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

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

```
boxblur=2:1:cr=0:ar=0
```

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

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

37.9 colorbalance

Modify intensity of primary colors (red, green and blue) of input frames.

The filter allows an input frame to be adjusted in the shadows, midtones or highlights regions for the red-cyan, green-magenta or blue-yellow balance.

A positive adjustment value shifts the balance towards the primary color, a negative value towards the complementary color.

The filter accepts the following options:

‘rs’
‘gs’
‘bs’

Adjust red, green and blue shadows (darkest pixels).

‘rm’
‘gm’
‘bm’

Adjust red, green and blue midtones (medium pixels).

‘rh’
‘gh’
‘bh’

Adjust red, green and blue highlights (brightest pixels).

Allowed ranges for options are `[-1.0, 1.0]`. Defaults are `0`.

37.9.1 Examples

- Add red color cast to shadows:

```
colorbalance=rs=.3
```

37.10 colorchannelmixer

Adjust video input frames by re-mixing color channels.

This filter modifies a color channel by adding the values associated to the other channels of the same pixels. For example if the value to modify is red, the output value will be:

```
red=red*rr + blue*rb + green*rg + alpha*ra
```

The filter accepts the following options:

‘rr’
‘rg’
‘rb’
‘ra’

Adjust contribution of input red, green, blue and alpha channels for output red channel. Default is `1` for *rr*, and `0` for *rg*, *rb* and *ra*.

‘gr’
‘gg’
‘gb’
‘ga’

Adjust contribution of input red, green, blue and alpha channels for output green channel. Default is `1` for *gg*, and `0` for *gr*, *gb* and *ga*.

‘br’
‘bg’
‘bb’
‘ba’

Adjust contribution of input red, green, blue and alpha channels for output blue channel. Default is `1` for *bb*, and `0` for *br*, *bg* and *ba*.

‘ar’
‘ag’
‘ab’
‘aa’

Adjust contribution of input red, green, blue and alpha channels for output alpha channel. Default is `1` for *aa*, and `0` for *ar*, *ag* and *ab*.

Allowed ranges for options are `[-2.0, 2.0]`.

37.10.1 Examples

- Convert source to grayscale:

```
colorchannelmixer=.3:.4:.3:0:.3:.4:.3:0:.3:.4:.3
```

37.11 colormatrix

Convert color matrix.

The filter accepts the following options:

‘src’
‘dst’

Specify the source and destination color matrix. Both values must be specified.

The accepted values are:

‘bt709’

BT.709

‘bt601’

BT.601

‘smpte240m’

SMPTE-240M

‘fcc’

FCC

For example to convert from BT.601 to SMPTE-240M, use the command:

```
colormatrix=bt601:smpte240m
```

37.12 copy

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

37.13 crop

Crop the input video to given dimensions.

The filter accepts the following options:

‘w, out_w’

Width of the output video. It defaults to iw. This expression is evaluated only once during the filter configuration.

‘h, out_h’

Height of the output video. It defaults to ih. This expression is evaluated only once during the filter configuration.

‘x’

Horizontal position, in the input video, of the left edge of the output video. It defaults to (in_w-out_w)/2. This expression is evaluated per-frame.

‘y’

Vertical position, in the input video, of the top edge of the output video. It defaults to `(in_h-out_h)/2`. This expression is evaluated per-frame.

‘`keep_aspect`’

If set to 1 will force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio. It defaults to 0.

The *out_w*, *out_h*, *x*, *y* parameters are expressions containing the following constants:

‘`x, y`’

the computed values for *x* and *y*. They are evaluated for each new frame.

‘`in_w, in_h`’

the input width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

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

‘`n`’

the number of input frame, starting from 0

‘pos’

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

‘t’

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

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

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

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

37.13.1 Examples

- Crop area with size 100x100 at position (12,34).

  ```
  crop=100:100:12:34
  ```

  Using named options, the example above becomes:

  ```
  crop=w=100:h=100:x=12:y=34
  ```

- Crop the central input area with size 100x100:

  ```
  crop=100:100
  ```

- Crop the central input area with size 2/3 of the input video:

  ```
  crop=2/3*in_w:2/3*in_h
  ```

- Crop the input video central square:

  ```
  crop=out_w=in_h
  crop=in_h
  ```

- Delimit the rectangle with the top-left corner placed at position 100:100 and the right-bottom corner corresponding to the right-bottom corner of the input image:

```
crop=in_w-100:in_h-100:100:100
```

- Crop 10 pixels from the left and right borders, and 20 pixels from the top and bottom borders

```
crop=in_w-2*10:in_h-2*20
```

- Keep only the bottom right quarter of the input image:

```
crop=in_w/2:in_h/2:in_w/2:in_h/2
```

- Crop height for getting Greek harmony:

```
crop=in_w:1/PHI*in_w
```

- Appply trembling effect:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)
```

- Apply erratic camera effect depending on timestamp:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"
```

- Set x depending on the value of y:

```
crop=in_w/2:in_h/2:y:10+10*sin(n/10)
```

37.14 cropdetect

Auto-detect crop size.

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

The filter accepts the following options:

‘`limit`’

Set higher black value threshold, which can be optionally specified from nothing (0) to everything (255). An intensity value greater to the set value is considered non-black. Default value is 24.

‘`round`’

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

‘reset_count, reset’

Set the counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Default value is 0.

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

37.15 curves

Apply color adjustments using curves.

This filter is similar to the Adobe Photoshop and GIMP curves tools. Each component (red, green and blue) has its values defined by *N* key points tied from each other using a smooth curve. The x-axis represents the pixel values from the input frame, and the y-axis the new pixel values to be set for the output frame.

By default, a component curve is defined by the two points *(0;0)* and *(1;1)*. This creates a straight line where each original pixel value is "adjusted" to its own value, which means no change to the image.

The filter allows you to redefine these two points and add some more. A new curve (using a natural cubic spline interpolation) will be define to pass smoothly through all these new coordinates. The new defined points needs to be strictly increasing over the x-axis, and their *x* and *y* values must be in the *[0;1]* interval. If the computed curves happened to go outside the vector spaces, the values will be clipped accordingly.

If there is no key point defined in `x=0`, the filter will automatically insert a *(0;0)* point. In the same way, if there is no key point defined in `x=1`, the filter will automatically insert a *(1;1)* point.

The filter accepts the following options:

‘preset’

Select one of the available color presets. This option can be used in addition to the ‘r’, ‘g’, ‘b’ parameters; in this case, the later options takes priority on the preset values. Available presets are:

‘none’
‘color_negative’
‘cross_process’
‘darker’
‘increase_contrast’
‘lighter’
‘linear_contrast’

‘medium_contrast’
‘negative’
‘strong_contrast’
‘vintage’

Default is none.

‘master, m’

Set the master key points. These points will define a second pass mapping. It is sometimes called a "luminance" or "value" mapping. It can be used with ‘r’, ‘g’, ‘b’ or ‘all’ since it acts like a post-processing LUT.

‘red, r’

Set the key points for the red component.

‘green, g’

Set the key points for the green component.

‘blue, b’

Set the key points for the blue component.

‘all’

Set the key points for all components (not including master). Can be used in addition to the other key points component options. In this case, the unset component(s) will fallback on this ‘all’ setting.

‘psfile’

Specify a Photoshop curves file (.asv) to import the settings from.

To avoid some filtergraph syntax conflicts, each key points list need to be defined using the following syntax: x0/y0 x1/y1 x2/y2 ....

37.15.1 Examples

- Increase slightly the middle level of blue:

```
curves=blue='0.5/0.58'
```

- Vintage effect:

```
curves=r='0/0.11 .42/.51 1/0.95':g='0.50/0.48':b='0/0.22 .49/.44 1/0.8'
```

Here we obtain the following coordinates for each components:

red

```
(0;0.11) (0.42;0.51) (1;0.95)
```

green

```
(0;0) (0.50;0.48) (1;1)
```

blue

```
(0;0.22) (0.49;0.44) (1;0.80)
```

- The previous example can also be achieved with the associated built-in preset:

```
curves=preset=vintage
```

- Or simply:

```
curves=vintage
```

- Use a Photoshop preset and redefine the points of the green component:

```
curves=psfile='MyCurvesPresets/purple.asv':green='0.45/0.53'
```

37.16 decimate

Drop duplicated frames at regular intervals.

The filter accepts the following options:

'`cycle`'

Set the number of frames from which one will be dropped. Setting this to *N* means one frame in every batch of *N* frames will be dropped. Default is `5`.

'`dupthresh`'

Set the threshold for duplicate detection. If the difference metric for a frame is less than or equal to this value, then it is declared as duplicate. Default is `1.1`

'`scthresh`'

Set scene change threshold. Default is `15`.

‘`blockx`’
‘`blocky`’

Set the size of the x and y-axis blocks used during metric calculations. Larger blocks give better noise suppression, but also give worse detection of small movements. Must be a power of two. Default is `32`.

‘`ppsrc`’

Mark main input as a pre-processed input and activate clean source input stream. This allows the input to be pre-processed with various filters to help the metrics calculation while keeping the frame selection lossless. When set to `1`, the first stream is for the pre-processed input, and the second stream is the clean source from where the kept frames are chosen. Default is `0`.

‘`chroma`’

Set whether or not chroma is considered in the metric calculations. Default is `1`.

37.17 delogo

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

This filter accepts the following options:

‘`x, y`’

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

‘`w, h`’

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

‘`band, t`’

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

‘`show`’

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

37.17.1 Examples

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

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

37.18 deshake

Attempt to fix small changes in horizontal and/or vertical shift. This filter helps remove camera shake from hand-holding a camera, bumping a tripod, moving on a vehicle, etc.

The filter accepts the following options:

‘x’
‘y’
‘w’
‘h’

Specify a rectangular area where to limit the search for motion vectors. If desired the search for motion vectors can be limited to a rectangular area of the frame defined by its top left corner, width and height. These parameters have the same meaning as the drawbox filter which can be used to visualise the position of the bounding box.

This is useful when simultaneous movement of subjects within the frame might be confused for camera motion by the motion vector search.

If any or all of *x*, *y*, *w* and *h* are set to -1 then the full frame is used. This allows later options to be set without specifying the bounding box for the motion vector search.

Default - search the whole frame.

‘rx’
‘ry’

Specify the maximum extent of movement in x and y directions in the range 0-64 pixels. Default 16.

‘edge’

Specify how to generate pixels to fill blanks at the edge of the frame. Available values are:

‘blank, 0’

Fill zeroes at blank locations

‘`original, 1`’

Original image at blank locations

‘`clamp, 2`’

Extruded edge value at blank locations

‘`mirror, 3`’

Mirrored edge at blank locations

Default value is ‘`mirror`’.

‘`blocksize`’

Specify the blocksize to use for motion search. Range 4-128 pixels, default 8.

‘`contrast`’

Specify the contrast threshold for blocks. Only blocks with more than the specified contrast (difference between darkest and lightest pixels) will be considered. Range 1-255, default 125.

‘`search`’

Specify the search strategy. Available values are:

‘`exhaustive, 0`’

Set exhaustive search

‘`less, 1`’

Set less exhaustive search.

Default value is ‘`exhaustive`’.

‘`filename`’

If set then a detailed log of the motion search is written to the specified file.

‘`opencl`’

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

37.19 drawbox

Draw a colored box on the input image.

This filter accepts the following options:

‘`x, y`’

Specify the top left corner coordinates of the box. Default to 0.

‘`width, w`’
‘`height, h`’

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

‘`color, c`’

Specify the color of the box to write, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence. If the special value `invert` is used, the box edge color is the same as the video with inverted luma.

‘`thickness, t`’

Set the thickness of the box edge. Default value is `4`.

37.19.1 Examples

- Draw a black box around the edge of the input image:

```
drawbox
```

- Draw a box with color red and an opacity of 50%:

```
drawbox=10:20:200:60:red@0.5
```

The previous example can be specified as:

```
drawbox=x=10:y=20:w=200:h=60:color=red@0.5
```

- Fill the box with pink color:

```
drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max
```

37.20 drawtext

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

To enable compilation of this filter you need to configure FFmpeg with `--enable-libfreetype`.

37.20.1 Syntax

The description of the accepted parameters follows.

‘`box`’

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

‘`boxcolor`’

The color to be used for drawing box around text. Either a string (e.g. "yellow") or in 0xRRGGBB[AA] format (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *boxcolor* is "white".

‘`draw`’

Set an expression which specifies if the text should be drawn. If the expression evaluates to 0, the text is not drawn. This is useful for specifying that the text should be drawn only when specific conditions are met.

Default value is "1".

See below for the list of accepted constants and functions.

‘`expansion`’

Select how the *text* is expanded. Can be either `none`, `strftime` (deprecated) or `normal` (default). See the Text expansion section below for details.

‘`fix_bounds`’

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

‘`fontcolor`’

The color to be used for drawing fonts. Either a string (e.g. "red") or in 0xRRGGBB[AA] format (e.g. "0xff000033"), possibly followed by an alpha specifier. The default value of *fontcolor* is "black".

‘`fontfile`’

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

‘`fontsize`’

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

‘`ft_load_flags`’

Flags to be used for loading the fonts.

The flags map the corresponding flags supported by libfreetype, and are a combination of the following values:

default
no_scale
no_hinting
render
no_bitmap
vertical_layout
force_autohint
crop_bitmap
pedantic
ignore_global_advance_width
no_recurse
ignore_transform
monochrome
linear_design
no_autohint

Default value is "render".

For more information consult the documentation for the FT_LOAD_* libfreetype flags.

‘`shadowcolor`’

The color to be used for drawing a shadow behind the drawn text. It can be a color name (e.g. "yellow") or a string in the 0xRRGGBB[AA] form (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *shadowcolor* is "black".

‘`shadowx, shadowy`’

The x and y offsets for the text shadow position with respect to the position of the text. They can be either positive or negative values. Default value for both is "0".

‘`tabsize`’

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

‘timecode’

Set the initial timecode representation in "hh:mm:ss[:;.]ff" format. It can be used with or without text parameter. *timecode_rate* option must be specified.

‘timecode_rate, rate, r’

Set the timecode frame rate (timecode only).

‘text’

The text string to be drawn. The text must be a sequence of UTF-8 encoded characters. This parameter is mandatory if no file is specified with the parameter *textfile*.

‘textfile’

A text file containing text to be drawn. The text must be a sequence of UTF-8 encoded characters.

This parameter is mandatory if no text string is specified with the parameter *text*.

If both *text* and *textfile* are specified, an error is thrown.

‘reload’

If set to 1, the *textfile* will be reloaded before each frame. Be sure to update it atomically, or it may be read partially, or even fail.

‘x, y’

The expressions which specify the offsets where text will be drawn within the video frame. They are relative to the top/left border of the output image.

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

See below for the list of accepted constants and functions.

The parameters for *x* and *y* are expressions containing the following constants and functions:

‘dar’

input display aspect ratio, it is the same as (*w* / *h*) * *sar*

‘hsub, vsub’

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

‘line_h, lh’

the height of each text line

‘`main_h, h, H`’

the input height

‘`main_w, w, W`’

the input width

‘`max_glyph_a, ascent`’

the maximum distance from the baseline to the highest/upper grid coordinate used to place a glyph outline point, for all the rendered glyphs. It is a positive value, due to the grid’s orientation with the Y axis upwards.

‘`max_glyph_d, descent`’

the maximum distance from the baseline to the lowest grid coordinate used to place a glyph outline point, for all the rendered glyphs. This is a negative value, due to the grid’s orientation, with the Y axis upwards.

‘`max_glyph_h`’

maximum glyph height, that is the maximum height for all the glyphs contained in the rendered text, it is equivalent to *ascent* - *descent*.

‘`max_glyph_w`’

maximum glyph width, that is the maximum width for all the glyphs contained in the rendered text

‘`n`’

the number of input frame, starting from 0

‘`rand(min, max)`’

return a random number included between *min* and *max*

‘`sar`’

input sample aspect ratio

‘`t`’

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

‘`text_h, th`’

the height of the rendered text

‘text_w, tw’

the width of the rendered text

‘x, y’

the x and y offset coordinates where the text is drawn.

These parameters allow the *x* and *y* expressions to refer each other, so you can for example specify `y=x/dar`.

If libavfilter was built with `--enable-fontconfig`, then ‘`fontfile`’ can be a fontconfig pattern or omitted.

37.20.2 Text expansion

If ‘`expansion`’ is set to `strftime`, the filter recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime(). This feature is deprecated.

If ‘`expansion`’ is set to `none`, the text is printed verbatim.

If ‘`expansion`’ is set to `normal` (which is the default), the following expansion mechanism is used.

The backslash character ’\’, followed by any character, always expands to the second character.

Sequence of the form `%{...}` are expanded. The text between the braces is a function name, possibly followed by arguments separated by ’:’. If the arguments contain special characters or delimiters (’:’ or ’}’), they should be escaped.

Note that they probably must also be escaped as the value for the ‘`text`’ option in the filter argument string and as the filter argument in the filtergraph description, and possibly also for the shell, that makes up to four levels of escaping; using a text file avoids these problems.

The following functions are available:

`expr, e`

The expression evaluation result.

It must take one argument specifying the expression to be evaluated, which accepts the same constants and functions as the *x* and *y* values. Note that not all constants should be used, for example the text size is not known when evaluating the expression, so the constants *text_w* and *text_h* will have an undefined value.

`gmtime`

The time at which the filter is running, expressed in UTC. It can accept an argument: a strftime() format string.

`localtime`

The time at which the filter is running, expressed in the local time zone. It can accept an argument: a strftime() format string.

`n, frame_num`

The frame number, starting from 0.

`pict_type`

A 1 character description of the current picture type.

`pts`

The timestamp of the current frame, in seconds, with microsecond accuracy.

37.20.3 Examples

- Draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

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

- Draw 'Test Text' with font FreeSerif of size 24 at position x=100 and y=50 (counting from the top-left corner of the screen), text is yellow with a red box around it. Both the text and the box have an opacity of 20%.

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
          x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
```

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

- Show the text at the center of the video frame:

```
drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
```

- Show a text line sliding from right to left in the last row of the video frame. The file '`LONG_LINE`' is assumed to contain a single line with no newlines.

```
drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
```

- Show the content of file '`CREDITS`' off the bottom of the frame and scroll up.

```
drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
```

- Draw a single green letter "g", at the center of the input video. The glyph baseline is placed at half screen height.

```
drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=g:x=(w-max_glyph_w)/2:y=h/2-ascent"
```

- Show text for 1 second every 3 seconds:

```
drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:draw=lt(mod(t\,3)\,1):text='blink'"
```

- Use fontconfig to set the font. Note that the colons need to be escaped.

```
drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
```

- Print the date of a real-time encoding (see strftime(3)):

```
drawtext='fontfile=FreeSans.ttf:text=%{localtime:%a %b %d %Y}'
```

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

For more information about fontconfig, check:
http://freedesktop.org/software/fontconfig/fontconfig-user.html.

37.21 edgedetect

Detect and draw edges. The filter uses the Canny Edge Detection algorithm.

The filter accepts the following options:

`'low, high'`

Set low and high threshold values used by the Canny thresholding algorithm.

The high threshold selects the "strong" edge pixels, which are then connected through 8-connectivity with the "weak" edge pixels selected by the low threshold.

low and *high* threshold values must be choosen in the range [0,1], and *low* should be lesser or equal to *high*.

Default value for *low* is `20/255`, and default value for *high* is `50/255`.

Example:

```
edgedetect=low=0.1:high=0.4
```

37.22 fade

Apply fade-in/out effect to input video.

This filter accepts the following options:

‘`type, t`’

The effect type – can be either "in" for fade-in, or "out" for a fade-out effect. Default is `in`.

‘`start_frame, s`’

Specify the number of the start frame for starting to apply the fade effect. Default is 0.

‘`nb_frames, n`’

The number of frames for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black. Default is 25.

‘`alpha`’

If set to 1, fade only alpha channel, if one exists on the input. Default value is 0.

‘`start_time, st`’

Specify the timestamp (in seconds) of the frame to start to apply the fade effect. If both start_frame and start_time are specified, the fade will start at whichever comes last. Default is 0.

‘`duration, d`’

The number of seconds for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black. If both duration and nb_frames are specified, duration is used. Default is 0.

37.22.1 Examples

- Fade in first 30 frames of video:

  ```
  fade=in:0:30
  ```

  The command above is equivalent to:

```
fade=t=in:s=0:n=30
```

- Fade out last 45 frames of a 200-frame video:

```
fade=out:155:45
fade=type=out:start_frame=155:nb_frames=45
```

- Fade in first 25 frames and fade out last 25 frames of a 1000-frame video:

```
fade=in:0:25, fade=out:975:25
```

- Make first 5 frames black, then fade in from frame 5-24:

```
fade=in:5:20
```

- Fade in alpha over first 25 frames of video:

```
fade=in:0:25:alpha=1
```

- Make first 5.5 seconds black, then fade in for 0.5 seconds:

```
fade=t=in:st=5.5:d=0.5
```

37.23 field

Extract a single field from an interlaced image using stride arithmetic to avoid wasting CPU time. The output frames are marked as non-interlaced.

The filter accepts the following options:

‘`type`’

Specify whether to extract the top (if the value is `0` or `top`) or the bottom field (if the value is `1` or `bottom`).

37.24 fieldmatch

Field matching filter for inverse telecine. It is meant to reconstruct the progressive frames from a telecined stream. The filter does not drop duplicated frames, so to achieve a complete inverse telecine `fieldmatch` needs to be followed by a decimation filter such as decimate in the filtergraph.

The separation of the field matching and the decimation is notably motivated by the possibility of inserting a de-interlacing filter fallback between the two. If the source has mixed telecined and real interlaced content, `fieldmatch` will not be able to match fields for the interlaced parts. But these remaining combed frames will be marked as interlaced, and thus can be de-interlaced by a later filter such as yadif before decimation.

In addition to the various configuration options, `fieldmatch` can take an optional second stream, activated through the ‘`ppsrc`’ option. If enabled, the frames reconstruction will be based on the fields and frames from this second stream. This allows the first input to be pre-processed in order to help the various algorithms of the filter, while keeping the output lossless (assuming the fields are matched properly). Typically, a field-aware denoiser, or brightness/contrast adjustments can help.

Note that this filter uses the same algorithms as TIVTC/TFM (AviSynth project) and VIVTC/VFM (VapourSynth project). The later is a light clone of TFM from which `fieldmatch` is based on. While the semantic and usage are very close, some behaviour and options names can differ.

The filter accepts the following options:

‘`order`’

Specify the assumed field order of the input stream. Available values are:

‘`auto`’

Auto detect parity (use FFmpeg’s internal parity value).

‘`bff`’

Assume bottom field first.

‘`tff`’

Assume top field first.

Note that it is sometimes recommended not to trust the parity announced by the stream.

Default value is *auto*.

‘`mode`’

Set the matching mode or strategy to use. ‘`pc`’ mode is the safest in the sense that it wont risk creating jerkiness due to duplicate frames when possible, but if there are bad edits or blended fields it will end up outputting combed frames when a good match might actually exist. On the other hand, ‘`pcn_ub`’ mode is the most risky in terms of creating jerkiness, but will almost always find a good frame if there is one. The other values are all somewhere in between ‘`pc`’ and ‘`pcn_ub`’ in terms of risking jerkiness and creating duplicate frames versus finding good matches in sections with bad edits, orphaned fields, blended fields, etc.

More details about p/c/n/u/b are available in p/c/n/u/b meaning section.

Available values are:

‘`pc`’

2-way matching (p/c)

‘`pc_n`’

2-way matching, and trying 3rd match if still combed (p/c + n)

‘`pc_u`’

2-way matching, and trying 3rd match (same order) if still combed (p/c + u)

‘`pc_n_ub`’

2-way matching, trying 3rd match if still combed, and trying 4th/5th matches if still combed (p/c + n + u/b)

‘`pcn`’

3-way matching (p/c/n)

‘`pcn_ub`’

3-way matching, and trying 4th/5th matches if all 3 of the original matches are detected as combed (p/c/n + u/b)

The parenthesis at the end indicate the matches that would be used for that mode assuming ‘`order`’=*tff* (and ‘`field`’ on *auto* or *top*).

In terms of speed ‘`pc`’ mode is by far the fastest and ‘`pcn_ub`’ is the slowest.

Default value is *pc_n*.

‘`ppsrc`’

Mark the main input stream as a pre-processed input, and enable the secondary input stream as the clean source to pick the fields from. See the filter introduction for more details. It is similar to the ‘`clip2`’ feature from VFM/TFM.

Default value is `0` (disabled).

‘`field`’

Set the field to match from. It is recommended to set this to the same value as ‘`order`’ unless you experience matching failures with that setting. In certain circumstances changing the field that is used to match from can have a large impact on matching performance. Available values are:

‘auto’

Automatic (same value as ‘order’).

‘bottom’

Match from the bottom field.

‘top’

Match from the top field.

Default value is *auto*.

‘mchroma’

Set whether or not chroma is included during the match comparisons. In most cases it is recommended to leave this enabled. You should set this to 0 only if your clip has bad chroma problems such as heavy rainbowing or other artifacts. Setting this to 0 could also be used to speed things up at the cost of some accuracy.

Default value is 1.

‘y0’
‘y1’

These define an exclusion band which excludes the lines between ‘y0’ and ‘y1’ from being included in the field matching decision. An exclusion band can be used to ignore subtitles, a logo, or other things that may interfere with the matching. ‘y0’ sets the starting scan line and ‘y1’ sets the ending line; all lines in between ‘y0’ and ‘y1’ (including ‘y0’ and ‘y1’) will be ignored. Setting ‘y0’ and ‘y1’ to the same value will disable the feature. ‘y0’ and ‘y1’ defaults to 0.

‘scthresh’

Set the scene change detection threshold as a percentage of maximum change on the luma plane. Good values are in the [8.0, 14.0] range. Scene change detection is only relevant in case ‘combmatch’=*sc*. The range for ‘scthresh’ is [0.0, 100.0].

Default value is 12.0.

‘combmatch’

When ‘combatch’ is not *none*, fieldmatch will take into account the combed scores of matches when deciding what match to use as the final match. Available values are:

‘none’

No final matching based on combed scores.

‘sc’

Combed scores are only used when a scene change is detected.

‘full’

Use combed scores all the time.

Default is *sc*.

‘combdbg’

Force fieldmatch to calculate the combed metrics for certain matches and print them. This setting is known as ‘micout’ in TFM/VFM vocabulary. Available values are:

‘none’

No forced calculation.

‘pcn’

Force p/c/n calculations.

‘pcnub’

Force p/c/n/u/b calculations.

Default value is *none*.

‘cthresh’

This is the area combing threshold used for combed frame detection. This essentially controls how "strong" or "visible" combing must be to be detected. Larger values mean combing must be more visible and smaller values mean combing can be less visible or strong and still be detected. Valid settings are from -1 (every pixel will be detected as combed) to 255 (no pixel will be detected as combed). This is basically a pixel difference value. A good range is [8, 12].

Default value is 9.

‘chroma’

Sets whether or not chroma is considered in the combed frame decision. Only disable this if your source has chroma problems (rainbowing, etc.) that are causing problems for the combed frame detection with chroma enabled. Actually, using ‘chroma’*=0* is usually more reliable, except for the case where there is chroma only combing in the source.

Default value is `0`.

‘`blockx`’
‘`blocky`’

Respectively set the x-axis and y-axis size of the window used during combed frame detection. This has to do with the size of the area in which ‘`combpel`’ pixels are required to be detected as combed for a frame to be declared combed. See the ‘`combpel`’ parameter description for more info. Possible values are any number that is a power of 2 starting at 4 and going up to 512.

Default value is `16`.

‘`combpel`’

The number of combed pixels inside any of the ‘`blocky`’ by ‘`blockx`’ size blocks on the frame for the frame to be detected as combed. While ‘`cthresh`’ controls how "visible" the combing must be, this setting controls "how much" combing there must be in any localized area (a window defined by the ‘`blockx`’ and ‘`blocky`’ settings) on the frame. Minimum value is `0` and maximum is `blocky x blockx` (at which point no frames will ever be detected as combed). This setting is known as ‘`MI`’ in TFM/VFM vocabulary.

Default value is `80`.

37.24.1 p/c/n/u/b meaning

37.24.1.1 p/c/n

We assume the following telecined stream:

```
Top fields:     1 2 2 3 4
Bottom fields:  1 2 3 4 4
```

The numbers correspond to the progressive frame the fields relate to. Here, the first two frames are progressive, the 3rd and 4th are combed, and so on.

When `fieldmatch` is configured to run a matching from bottom (‘`field`’=*bottom*) this is how this input stream get transformed:

```
Input stream:
                T     1 2 2 3 4
                B     1 2 3 4 4   <-- matching reference

Matches:              c c n n c

Output stream:
                T     1 2 3 4 4
                B     1 2 3 4 4
```

As a result of the field matching, we can see that some frames get duplicated. To perform a complete inverse telecine, you need to rely on a decimation filter after this operation. See for instance the decimate filter.

The same operation now matching from top fields (‘`field`’=*top*) looks like this:

```
Input stream:
                T     1 2 2 3 4   <-- matching reference
                B     1 2 3 4 4

Matches:              c c p p c

Output stream:
                T     1 2 2 3 4
                B     1 2 2 3 4
```

In these examples, we can see what *p*, *c* and *n* mean; basically, they refer to the frame and field of the opposite parity:

- *p* matches the field of the opposite parity in the previous frame
- *c* matches the field of the opposite parity in the current frame
- *n* matches the field of the opposite parity in the next frame

37.24.1.2 u/b

The *u* and *b* matching are a bit special in the sense that they match from the opposite parity flag. In the following examples, we assume that we are currently matching the 2nd frame (Top:2, bottom:2). According to the match, a ’x’ is placed above and below each matched fields.

With bottom matching (‘`field`’=*bottom*):

```
Match:           c         p           n          b          u

                 x       x               x        x          x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x         x           x        x              x

Output frames:
                 2          1          2          2          2
                 2          2          2          1          3
```

With top matching (‘`field`’=*top*):

```
Match:           c         p           n          b          u

                 x         x           x        x              x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x       x               x        x          x

Output frames:
                 2          2          2          1          2
                 2          1          3          2          2
```

37.24.2 Examples

Simple IVTC of a top field first telecined stream:

```
fieldmatch=order=tff:combmatch=none, decimate
```

Advanced IVTC, with fallback on yadif for still combed frames:

```
fieldmatch=order=tff:combmatch=full, yadif=deint=interlaced, decimate
```

37.25 fieldorder

Transform the field order of the input video.

This filter accepts the following options:

‘`order`’

Output field order. Valid values are *tff* for top field first or *bff* for bottom field first.

Default value is ‘`tff`’.

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

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

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

For example:

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

37.26 fifo

Buffer input images and send them when they are requested.

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

The filter does not take parameters.

37.27 format

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

This filter accepts the following parameters:

‘`pix_fmts`’

A ’|’-separated list of pixel format names, for example "pix_fmts=yuv420p|monow|rgb24".

37.27.1 Examples

- Convert the input video to the format *yuv420p*

  ```
  format=pix_fmts=yuv420p
  ```

  Convert the input video to any of the formats in the list

  ```
  format=pix_fmts=yuv420p|yuv444p|yuv410p
  ```

37.28 fps

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

This filter accepts the following named parameters:

‘`fps`’

Desired output frame rate. The default is `25`.

‘`round`’

Rounding method.

Possible values are:

‘zero’

zero round towards 0

‘inf’

round away from 0

‘down’

round towards -infinity

‘up’

round towards +infinity

‘near’

round to nearest

The default is `near`.

Alternatively, the options can be specified as a flat string: *fps*[:*round*].

See also the setpts filter.

37.29 framestep

Select one frame every N-th frame.

This filter accepts the following option:

‘step’

Select frame after every `step` frames. Allowed values are positive integers higher than 0. Default value is `1`.

37.30 frei0r

Apply a frei0r effect to the input video.

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

This filter accepts the following options:

‘filter_name’

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

‘filter_params’

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

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

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

37.30.1 Examples

- Apply the distort0r effect, set the first two double parameters:

```
frei0r=filter_name=distort0r:filter_params=0.5|0.01
```

- Apply the colordistance effect, take a color as first parameter:

```
frei0r=colordistance:0.2/0.3/0.4
frei0r=colordistance:violet
frei0r=colordistance:0x112233
```

- Apply the perspective effect, specify the top left and top right image positions:

```
frei0r=perspective:0.2/0.2|0.8/0.2
```

For more information see: http://frei0r.dyne.org

37.31 geq

The filter accepts the following options:

‘lum_expr’

the luminance expression

‘cb_expr’

the chrominance blue expression

‘cr_expr’

the chrominance red expression

‘alpha_expr’

the alpha expression

‘r’

the red expression

‘g’

the green expression

‘b’

the blue expression

If one of the chrominance expression is not defined, it falls back on the other one. If no alpha expression is specified it will evaluate to opaque value. If none of chrominance expressions are specified, they will evaluate the luminance expression.

The expressions can use the following variables and functions:

‘N’

The sequential number of the filtered frame, starting from 0.

‘X’
‘Y’

The coordinates of the current sample.

‘W’
‘H’

The width and height of the image.

‘SW’
‘SH’

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

‘`T`’

Time of the current frame, expressed in seconds.

‘`p(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the current plane.

‘`lum(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the luminance plane.

‘`cb(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the blue-difference chroma plane. Returns 0 if there is no such plane.

‘`cr(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the red-difference chroma plane. Returns 0 if there is no such plane.

‘`alpha(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the alpha plane. Returns 0 if there is no such plane.

For functions, if *x* and *y* are outside the area, the value will be automatically clipped to the closer edge.

37.31.1 Examples

- Flip the image horizontally:

```
geq=p(W-X\,Y)
```

- Generate a bidimensional sine wave, with angle `PI/3` and a wavelength of 100 pixels:

```
geq=128 + 100*sin(2*(PI/100)*(cos(PI/3)*(X-50*T) + sin(PI/3)*Y)):128:128
```

- Generate a fancy enigmatic moving light:

```
nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N*0.07)*W/2-W/2\,Y-sin(N*0.09)*H/2-H/2)^2*1000000*sin(N*0.02):128:128
```

- Generate a quick emboss effect:

```
format=gray,geq=lum_expr='(p(X,Y)+(256-p(X-4,Y-4)))/2'
```

37.32 gradfun

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

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

This filter accepts the following options:

`'strength'`

The maximum amount by which the filter will change any one pixel. Also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 64, default value is 1.2, out-of-range values will be clipped to the valid range.

`'radius'`

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

Alternatively, the options can be specified as a flat string: *strength*[:*radius*]

37.32.1 Examples

- Apply the filter with a `3.5` strength and radius of `8`:

```
gradfun=3.5:8
```

- Specify radius, omitting the strength (which will fall-back to the default value):

```
gradfun=radius=8
```

37.33 hflip

Flip the input video horizontally.

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

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

37.34 histeq

This filter applies a global color histogram equalization on a per-frame basis.

It can be used to correct video that has a compressed range of pixel intensities. The filter redistributes the pixel intensities to equalize their distribution across the intensity range. It may be viewed as an "automatically adjusting contrast filter". This filter is useful only for correcting degraded or poorly captured source video.

The filter accepts the following options:

‘`strength`’

Determine the amount of equalization to be applied. As the strength is reduced, the distribution of pixel intensities more-and-more approaches that of the input frame. The value must be a float number in the range [0,1] and defaults to 0.200.

‘`intensity`’

Set the maximum intensity that can generated and scale the output values appropriately. The strength should be set as desired and then the intensity can be limited if needed to avoid washing-out. The value must be a float number in the range [0,1] and defaults to 0.210.

‘`antibanding`’

Set the antibanding level. If enabled the filter will randomly vary the luminance of output pixels by a small amount to avoid banding of the histogram. Possible values are `none`, `weak` or `strong`. It defaults to `none`.

37.35 histogram

Compute and draw a color distribution histogram for the input video.

The computed histogram is a representation of distribution of color components in an image.

The filter accepts the following options:

‘`mode`’

Set histogram mode.

It accepts the following values:

‘levels’

standard histogram that display color components distribution in an image. Displays color graph for each color component. Shows distribution of the Y, U, V, A or G, B, R components, depending on input format, in current frame. Bellow each graph is color component scale meter.

‘color’

chroma values in vectorscope, if brighter more such chroma values are distributed in an image. Displays chroma values (U/V color placement) in two dimensional graph (which is called a vectorscope). It can be used to read of the hue and saturation of the current frame. At a same time it is a histogram. The whiter a pixel in the vectorscope, the more pixels of the input frame correspond to that pixel (that is the more pixels have this chroma value). The V component is displayed on the horizontal (X) axis, with the leftmost side being V = 0 and the rightmost side being V = 255. The U component is displayed on the vertical (Y) axis, with the top representing U = 0 and the bottom representing U = 255.

The position of a white pixel in the graph corresponds to the chroma value of a pixel of the input clip. So the graph can be used to read of the hue (color flavor) and the saturation (the dominance of the hue in the color). As the hue of a color changes, it moves around the square. At the center of the square, the saturation is zero, which means that the corresponding pixel has no color. If you increase the amount of a specific color, while leaving the other colors unchanged, the saturation increases, and you move towards the edge of the square.

‘color2’

chroma values in vectorscope, similar as `color` but actual chroma values are displayed.

‘waveform’

per row/column color component graph. In row mode graph in the left side represents color component value 0 and right side represents value = 255. In column mode top side represents color component value = 0 and bottom side represents value = 255.

Default value is `levels`.

‘level_height’

Set height of level in `levels`. Default value is `200`. Allowed range is [50, 2048].

‘scale_height’

Set height of color scale in `levels`. Default value is `12`. Allowed range is [0, 40].

‘step’

Set step for `waveform` mode. Smaller values are useful to find out how much of same luminance values across input rows/columns are distributed. Default value is `10`. Allowed range is [1, 255].

‘waveform_mode’

Set mode for waveform. Can be either row, or column. Default is row.

‘display_mode’

Set display mode for waveform and levels. It accepts the following values:

‘parade’

Display separate graph for the color components side by side in row waveform mode or one below other in column waveform mode for waveform histogram mode. For levels histogram mode per color component graphs are placed one bellow other.

This display mode in waveform histogram mode makes it easy to spot color casts in the highlights and shadows of an image, by comparing the contours of the top and the bottom of each waveform. Since whites, grays, and blacks are characterized by exactly equal amounts of red, green, and blue, neutral areas of the picture should display three waveforms of roughly equal width/height. If not, the correction is easy to make by making adjustments to level the three waveforms.

‘overlay’

Presents information that’s identical to that in the parade, except that the graphs representing color components are superimposed directly over one another.

This display mode in waveform histogram mode can make it easier to spot the relative differences or similarities in overlapping areas of the color components that are supposed to be identical, such as neutral whites, grays, or blacks.

Default is parade.

37.35.1 Examples

- Calculate and draw histogram:

```
ffplay -i input -vf histogram
```

37.36 hqdn3d

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

It accepts the following optional parameters:

‘luma_spatial’

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

‘chroma_spatial’

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

‘luma_tmp’

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

‘chroma_tmp’

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

37.37 hue

Modify the hue and/or the saturation of the input.

This filter accepts the following options:

‘h’

Specify the hue angle as a number of degrees. It accepts an expression, and defaults to "0".

‘s’

Specify the saturation in the [-10,10] range. It accepts an expression and defaults to "1".

‘H’

Specify the hue angle as a number of radians. It accepts an expression, and defaults to "0".

‘h’ and ‘H’ are mutually exclusive, and can’t be specified at the same time.

The ‘h’, ‘H’ and ‘s’ option values are expressions containing the following constants:

‘n’

frame count of the input frame starting from 0

‘pts’

presentation timestamp of the input frame expressed in time base units

‘r’

frame rate of the input video, NAN if the input frame rate is unknown

‘t’

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

‘tb’

time base of the input video

37.37.1 Examples

- Set the hue to 90 degrees and the saturation to 1.0:

  ```
  hue=h=90:s=1
  ```

- Same command but expressing the hue in radians:

  ```
  hue=H=PI/2:s=1
  ```

- Rotate hue and make the saturation swing between 0 and 2 over a period of 1 second:

  ```
  hue="H=2*PI*t: s=sin(2*PI*t)+1"
  ```

- Apply a 3 seconds saturation fade-in effect starting at 0:

  ```
  hue="s=min(t/3\,1)"
  ```

  The general fade-in expression can be written as:

  ```
  hue="s=min(0\, max((t-START)/DURATION\, 1))"
  ```

- Apply a 3 seconds saturation fade-out effect starting at 5 seconds:

  ```
  hue="s=max(0\, min(1\, (8-t)/3))"
  ```

  The general fade-out expression can be written as:

  ```
  hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
  ```

37.37.2 Commands

This filter supports the following commands:

‘s’
‘h’
‘H’

Modify the hue and/or the saturation of the input video. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

37.38 idet

Detect video interlacing type.

This filter tries to detect if the input is interlaced or progressive, top or bottom field first.

The filter accepts the following options:

‘intl_thres’

Set interlacing threshold.

‘prog_thres’

Set progressive threshold.

37.39 il

Deinterleave or interleave fields.

This filter allows to process interlaced images fields without deinterlacing them. Deinterleaving splits the input frame into 2 fields (so called half pictures). Odd lines are moved to the top half of the output image, even lines to the bottom half. You can process (filter) them independently and then re-interleave them.

The filter accepts the following options:

‘luma_mode, l’
‘chroma_mode, s’
‘alpha_mode, a’

Available values for *luma_mode*, *chroma_mode* and *alpha_mode* are:

‘none’

Do nothing.

‘deinterleave, d’

Deinterleave fields, placing one above the other.

‘interleave, i’

Interleave fields. Reverse the effect of deinterleaving.

Default value is none.

‘luma_swap, ls’
‘chroma_swap, cs’
‘alpha_swap, as’

Swap luma/chroma/alpha fields. Exchange even & odd lines. Default value is 0.

37.40 interlace

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

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

It accepts the following optional parameters:

‘scan’

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

‘lowpass’

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

37.41 kerndeint

Deinterlace input video by applying Donald Graft’s adaptive kernel deinterling. Work on interlaced parts of a video to produce progressive frames.

The description of the accepted parameters follows.

‘thresh’

Set the threshold which affects the filter’s tolerance when determining if a pixel line must be processed. It must be an integer in the range [0,255] and defaults to 10. A value of 0 will result in applying the process on every pixels.

‘map’

Paint pixels exceeding the threshold value to white if set to 1. Default is 0.

‘order’

Set the fields order. Swap fields if set to 1, leave fields alone if 0. Default is 0.

‘sharp’

Enable additional sharpening if set to 1. Default is 0.

‘twoway’

Enable twoway sharpening if set to 1. Default is 0.

37.41.1 Examples

- Apply default values:

```
kerndeint=thresh=10:map=0:order=0:sharp=0:twoway=0
```

- Enable additional sharpening:

```
kerndeint=sharp=1
```

- Paint processed pixels in white:

```
kerndeint=map=1
```

37.42 lut, lutrgb, lutyuv

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

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

These filters accept the following options:

‘`c0`’

set first pixel component expression

‘`c1`’

set second pixel component expression

‘`c2`’

set third pixel component expression

‘`c3`’

set fourth pixel component expression, corresponds to the alpha component

‘`r`’

set red component expression

‘`g`’

set green component expression

‘`b`’

set blue component expression

‘`a`’

alpha component expression

‘`y`’

set Y/luminance component expression

‘`u`’

set U/Cb component expression

‘`v`’

set V/Cr component expression

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

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

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

The expressions can contain the following constants and functions:

‘`w, h`’

the input width and height

‘`val`’

input value for the pixel component

‘`clipval`’

the input value clipped in the *minval-maxval* range

‘`maxval`’

maximum value for the pixel component

‘`minval`’

minimum value for the pixel component

‘`negval`’

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

‘`clip(val)`’

the computed value in *val* clipped in the *minval-maxval* range

‘`gammaval(gamma)`’

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

All expressions default to "val".

37.42.1 Examples

- Negate input video:

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

  The above is the same as:

```
lutrgb="r=negval:g=negval:b=negval"
lutyuv="y=negval:u=negval:v=negval"
```

- Negate luminance:

```
lutyuv=y=negval
```

- Remove chroma components, turns the video into a graytone image:

```
lutyuv="u=128:v=128"
```

- Apply a luma burning effect:

```
lutyuv="y=2*val"
```

- Remove green and blue components:

```
lutrgb="g=0:b=0"
```

- Set a constant alpha channel value on input:

```
format=rgba,lutrgb=a="maxval-minval/2"
```

- Correct luminance gamma by a 0.5 factor:

```
lutyuv=y=gammaval(0.5)
```

- Discard least significant bits of luma:

```
lutyuv=y='bitand(val, 128+64+32)'
```

37.43 mp

Apply an MPlayer filter to the input video.

This filter provides a wrapper around most of the filters of MPlayer/MEncoder.

This wrapper is considered experimental. Some of the wrapped filters may not work properly and we may drop support for them, as they will be implemented natively into FFmpeg. Thus you should avoid depending on them when writing portable scripts.

The filters accepts the parameters: *filter_name*[:=]*filter_params*

filter_name is the name of a supported MPlayer filter, *filter_params* is a string containing the parameters accepted by the named filter.

The list of the currently supported filters follows:

dint
eq2
eq
fil
fspp
ilpack
mcdeint
ow
perspective
phase
pp7
pullup
qp
sab
softpulldown
spp
uspp

The parameter syntax and behavior for the listed filters are the same of the corresponding MPlayer filters. For detailed instructions check the "VIDEO FILTERS" section in the MPlayer manual.

37.43.1 Examples

- Adjust gamma, brightness, contrast:

```
mp=eq2=1.0:2:0.5
```

See also mplayer(1), http://www.mplayerhq.hu/.

37.44 mpdecimate

Drop frames that do not differ greatly from the previous frame in order to reduce frame rate.

The main use of this filter is for very-low-bitrate encoding (e.g. streaming over dialup modem), but it could in theory be used for fixing movies that were inverse-telecined incorrectly.

A description of the accepted options follows.

‘`max`’

Set the maximum number of consecutive frames which can be dropped (if positive), or the minimum interval between dropped frames (if negative). If the value is 0, the frame is dropped unregarding the number of previous sequentially dropped frames.

Default value is 0.

‘`hi`’
‘`lo`’
‘`frac`’

Set the dropping threshold values.

Values for ‘`hi`’ and ‘`lo`’ are for 8x8 pixel blocks and represent actual pixel value differences, so a threshold of 64 corresponds to 1 unit of difference for each pixel, or the same spread out differently over the block.

A frame is a candidate for dropping if no 8x8 blocks differ by more than a threshold of ‘`hi`’, and if no more than ‘`frac`’ blocks (1 meaning the whole image) differ by more than a threshold of ‘`lo`’.

Default value for ‘`hi`’ is 64*12, default value for ‘`lo`’ is 64*5, and default value for ‘`frac`’ is 0.33.

37.45 negate

Negate input video.

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

37.46 noformat

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

This filter accepts the following parameters:

‘pix_fmts’

A ’|’-separated list of pixel format names, for example "pix_fmts=yuv420p|monow|rgb24".

37.46.1 Examples

- Force libavfilter to use a format different from *yuv420p* for the input to the vflip filter:

```
noformat=pix_fmts=yuv420p,vflip
```

- Convert the input video to any of the formats not contained in the list:

```
noformat=yuv420p|yuv444p|yuv410p
```

37.47 noise

Add noise on video input frame.

The filter accepts the following options:

‘all_seed’
‘c0_seed’
‘c1_seed’
‘c2_seed’
‘c3_seed’

Set noise seed for specific pixel component or all pixel components in case of *all_seed*. Default value is 123457.

‘all_strength, alls’
‘c0_strength, c0s’
‘c1_strength, c1s’
‘c2_strength, c2s’
‘c3_strength, c3s’

Set noise strength for specific pixel component or all pixel components in case *all_strength*. Default value is 0. Allowed range is [0, 100].

‘all_flags, allf’
‘c0_flags, c0f’
‘c1_flags, c1f’
‘c2_flags, c2f’

‘c3_flags, c3f’

Set pixel component flags or set flags for all components if *all_flags*. Available values for component flags are:

‘a’

averaged temporal noise (smoother)

‘p’

mix random noise with a (semi)regular pattern

‘t’

temporal noise (noise pattern changes between frames)

‘u’

uniform noise (gaussian otherwise)

37.47.1 Examples

Add temporal and uniform noise to input video:

```
noise=alls=20:allf=t+u
```

37.48 null

Pass the video source unchanged to the output.

37.49 ocv

Apply video transform using libopencv.

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

This filter accepts the following parameters:

‘filter_name’

The name of the libopencv filter to apply.

‘filter_params’

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

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

Follows the list of supported libopencv filters.

37.49.1 dilate

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

It accepts the parameters: *struct_el*|*nb_iterations*.

struct_el represents a structuring element, and has the syntax: *cols*x*rows*+*anchor_x*x*anchor_y*/*shape*

cols and *rows* represent the number of columns and rows of the structuring element, *anchor_x* and *anchor_y* the anchor point, and *shape* the shape for the structuring element, and can be one of the values "rect", "cross", "ellipse", "custom".

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

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

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

Follow some example:

```
# use the default values
ocv=dilate

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

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

37.49.2 erode

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

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

37.49.3 smooth

Smooth the input video.

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

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

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

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

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

37.50 overlay

Overlay one video on top of another.

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

This filter accepts the following parameters:

A description of the accepted options follows.

‘`x`’
‘`y`’

Set the expression for the x and y coordinates of the overlayed video on the main video. Default value is "0" for both expressions. In case the expression is invalid, it is set to a huge value (meaning that the overlay will not be displayed within the output visible area).

‘`eval`’

Set when the expressions for ‘`x`’, and ‘`y`’ are evaluated.

It accepts the following values:

‘`init`’

only evaluate expressions once during the filter initialization or when a command is processed

‘`frame`’

evaluate expressions for each incoming frame

Default value is ‘`frame`’.

‘`shortest`’

If set to 1, force the output to terminate when the shortest input terminates. Default value is 0.

‘`format`’

Set the format for the output video.

It accepts the following values:

‘`yuv420`’

force YUV420 output

‘`yuv444`’

force YUV444 output

‘`rgb`’

force RGB output

Default value is ‘`yuv420`’.

‘`rgb` *(deprecated)*’

If set to 1, force the filter to accept inputs in the RGB color space. Default value is 0. This option is deprecated, use ‘`format`’ instead.

‘`repeatlast`’

If set to 1, force the filter to draw the last overlay frame over the main input until the end of the stream. A value of 0 disables this behavior, which is enabled by default.

The ‘`x`’, and ‘`y`’ expressions can contain the following parameters.

‘main_w, W’
‘main_h, H’

main input width and height

‘overlay_w, w’
‘overlay_h, h’

overlay input width and height

‘x’
‘y’

the computed values for *x* and *y*. They are evaluated for each new frame.

‘hsub’
‘vsub’

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

‘n’

the number of input frame, starting from 0

‘pos’

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

‘t’

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

Note that the *n*, *pos*, *t* variables are available only when evaluation is done *per frame*, and will evaluate to NAN when ‘eval’ is set to ‘init’.

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

You can chain together more overlays but you should test the efficiency of such approach.

37.50.1 Commands

This filter supports the following commands:

‘x’

‘y’

Modify the x and y of the overlay input. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

37.50.2 Examples

- Draw the overlay at 10 pixels from the bottom right corner of the main video:

```
overlay=main_w-overlay_w-10:main_h-overlay_h-10
```

Using named options the example above becomes:

```
overlay=x=main_w-overlay_w-10:y=main_h-overlay_h-10
```

- Insert a transparent PNG logo in the bottom left corner of the input, using the `ffmpeg` tool with the `-filter_complex` option:

```
ffmpeg -i input -i logo -filter_complex 'overlay=10:main_h-overlay_h-10' output
```

- Insert 2 different transparent PNG logos (second logo on bottom right corner) using the `ffmpeg` tool:

```
ffmpeg -i input -i logo1 -i logo2 -filter_complex 'overlay=x=10:y=H-h-10,overlay=x=W-w-10:y=H-h-10' output
```

- Add a transparent color layer on top of the main video, `WxH` must specify the size of the main input to the overlay filter:

```
color=color=red@.3:size=WxH [over]; [in][over] overlay [out]
```

- Play an original video and a filtered version (here with the deshake filter) side by side using the `ffplay` tool:

```
ffplay input.avi -vf 'split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w'
```

The above command is the same as:

```
ffplay input.avi -vf 'split[b], pad=iw*2[src], [b]deshake, [src]overlay=w'
```

- Make a sliding overlay appearing from the left to the right top part of the screen starting since time 2:

```
overlay=x='if(gte(t,2), -w+(t-2)*20, NAN)':y=0
```

- Compose output by putting two input videos side to side:

```
ffmpeg -i left.avi -i right.avi -filter_complex "
nullsrc=size=200x100 [background];
[0:v] setpts=PTS-STARTPTS, scale=100x100 [left];
[1:v] setpts=PTS-STARTPTS, scale=100x100 [right];
[background][left]       overlay=shortest=1       [background+left];
[background+left][right] overlay=shortest=1:x=100 [left+right]
"
```

- Chain several overlays in cascade:

```
nullsrc=s=200x200 [bg];
testsrc=s=100x100, split=4 [in0][in1][in2][in3];
[in0] lutrgb=r=0, [bg]   overlay=0:0     [mid0];
[in1] lutrgb=g=0, [mid0] overlay=100:0   [mid1];
[in2] lutrgb=b=0, [mid1] overlay=0:100   [mid2];
[in3] null,       [mid2] overlay=100:100 [out0]
```

37.51 pad

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

This filter accepts the following parameters:

‘`width, w`’
‘`height, h`’

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

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

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

‘`x`’
‘`y`’

Specify an expression for the offsets where to place the input image in the padded area with respect to the top/left border of the output image.

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

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

‘color’

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

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

The value for the *width*, *height*, *x*, and *y* options are expressions containing the following constants:

‘in_w, in_h’

the input video width and height

‘iw, ih’

same as *in_w* and *in_h*

‘out_w, out_h’

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

‘ow, oh’

same as *out_w* and *out_h*

‘x, y’

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

‘a’

same as *iw* / *ih*

‘sar’

input sample aspect ratio

‘dar’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘hsub, vsub’

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

37.51.1 Examples

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

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

  The example above is equivalent to the following command:

  ```
  pad=width=640:height=480:x=0:y=40:color=violet
  ```

- Pad the input to get an output with dimensions increased by 3/2, and put the input video at the center of the padded area:

  ```
  pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a squared output with size equal to the maximum value between the input width and height, and put the input video at the center of the padded area:

  ```
  pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a final w/h ratio of 16:9:

  ```
  pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- In case of anamorphic video, in order to set the output display aspect correctly, it is necessary to use *sar* in the expression, according to the relation:

  ```
  (ih * X / ih) * sar = output_dar
  X = output_dar / sar
  ```

  Thus the previous example needs to be modified to:

  ```
  pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Double output size and put the input video in the bottom-right corner of the output padded area:

  ```
  pad="2*iw:2*ih:ow-iw:oh-ih"
  ```

37.52 pixdesctest

Pixel format descriptor test filter, mainly useful for internal testing. The output video should be equal to the input video.

For example:

```
format=monow, pixdesctest
```

can be used to test the monowhite pixel format descriptor definition.

37.53 pp

Enable the specified chain of postprocessing subfilters using libpostproc. This library should be automatically selected with a GPL build (`--enable-gpl`). Subfilters must be separated by '/' and can be disabled by prepending a '-'. Each subfilter and some options have a short and a long name that can be used interchangeably, i.e. dr/dering are the same.

The filters accept the following options:

'`subfilters`'

Set postprocessing subfilters string.

All subfilters share common options to determine their scope:

'`a/autoq`'

Honor the quality commands for this subfilter.

'`c/chrom`'

Do chrominance filtering, too (default).

'`y/nochrom`'

Do luminance filtering only (no chrominance).

'`n/noluma`'

Do chrominance filtering only (no luminance).

These options can be appended after the subfilter name, separated by a '|'.

Available subfilters are:

‘hb/hdeblock[|difference[|flatness]]’

Horizontal deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

‘vb/vdeblock[|difference[|flatness]]’

Vertical deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

‘ha/hadeblock[|difference[|flatness]]’

Accurate horizontal deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

‘va/vadeblock[|difference[|flatness]]’

Accurate vertical deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

The horizontal and vertical deblocking filters share the difference and flatness values so you cannot set different horizontal and vertical thresholds.

‘h1/x1hdeblock’

Experimental horizontal deblocking filter

‘v1/x1vdeblock’

Experimental vertical deblocking filter

‘dr/dering’

Deringing filter

‘tn/tmpnoise[|threshold1[|threshold2[|threshold3]]], temporal noise reducer’

‘threshold1’

larger -> stronger filtering

‘threshold2’

larger -> stronger filtering

‘threshold3’

larger -> stronger filtering

‘al/autolevels[:f/fullyrange], automatic brightness / contrast correction’

‘f/fullyrange’

Stretch luminance to 0-255.

‘lb/linblenddeint’

Linear blend deinterlacing filter that deinterlaces the given block by filtering all lines with a (1 2 1) filter.

‘li/linipoldeint’

Linear interpolating deinterlacing filter that deinterlaces the given block by linearly interpolating every second line.

‘ci/cubicipoldeint’

Cubic interpolating deinterlacing filter deinterlaces the given block by cubically interpolating every second line.

‘md/mediandeint’

Median deinterlacing filter that deinterlaces the given block by applying a median filter to every second line.

‘fd/ffmpegdeint’

FFmpeg deinterlacing filter that deinterlaces the given block by filtering every second line with a (-1 4 2 4 -1) filter.

‘l5/lowpass5’

Vertically applied FIR lowpass deinterlacing filter that deinterlaces the given block by filtering all lines with a (-1 2 6 2 -1) filter.

‘fq/forceQuant[|quantizer]’

Overrides the quantizer table from the input with the constant quantizer you specify.

‘quantizer’

Quantizer to use

‘de/default’

Default pp filter combination (hb|a,vb|a,dr|a)

‘fa/fast’

Fast pp filter combination (h1|a,v1|a,dr|a)

‘ac’

High quality pp filter combination (ha|a|128|7,va|a,dr|a)

37.53.1 Examples

- Apply horizontal and vertical deblocking, deringing and automatic brightness/contrast:

```
pp=hb/vb/dr/al
```

- Apply default filters without brightness/contrast correction:

```
pp=de/-al
```

- Apply default filters and temporal denoiser:

```
pp=default/tmpnoise|1|2|3
```

- Apply deblocking on luminance only, and switch vertical deblocking on or off automatically depending on available CPU time:

```
pp=hb|y/vb|a
```

37.54 removelogo

Suppress a TV station logo, using an image file to determine which pixels comprise the logo. It works by filling in the pixels that comprise the logo with neighboring pixels.

The filter accepts the following options:

‘`filename, f`’

Set the filter bitmap file, which can be any image format supported by libavformat. The width and height of the image file must match those of the video stream being processed.

Pixels in the provided bitmap image with a value of zero are not considered part of the logo, non-zero pixels are considered part of the logo. If you use white (255) for the logo and black (0) for the rest, you will be safe. For making the filter bitmap, it is recommended to take a screen capture of a black frame with the logo visible, and then using a threshold filter followed by the erode filter once or twice.

If needed, little splotches can be fixed manually. Remember that if logo pixels are not covered, the filter quality will be much reduced. Marking too many pixels as part of the logo does not hurt as much, but it will increase the amount of blurring needed to cover over the image and will destroy more information than necessary, and extra pixels will slow things down on a large logo.

37.55 scale

Scale (resize) the input video, using the libswscale library.

The scale filter forces the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

The filter accepts the following options:

‘`width, w`’

Set the output video width expression. Default value is `iw`. See below for the list of accepted constants.

‘`height, h`’

Set the output video height expression. Default value is `ih`. See below for the list of accepted constants.

‘`interl`’

Set the interlacing. It accepts the following values:

‘`1`’

force interlaced aware scaling

‘`0`’

do not apply interlaced scaling

‘`-1`’

select interlaced aware scaling depending on whether the source frames are flagged as interlaced or not

Default value is `0`.

‘`flags`’

Set libswscale scaling flags. If not explictly specified the filter applies a bilinear scaling algorithm.

‘`size, s`’

Set the video size, the value must be a valid abbreviation or in the form *width*x*height*.

The values of the *w* and *h* options are expressions containing the following constants:

‘`in_w, in_h`’

the input width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘a’

same as *iw* / *ih*

‘sar’

input sample aspect ratio

‘dar’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘hsub, vsub’

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

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

If the value for *w* or *h* is 0, the respective input size is used for the output.

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

37.55.1 Examples

- Scale the input video to a size of 200x100:

  ```
  scale=w=200:h=100
  ```

  This is equivalent to:

  ```
  scale=200:100
  ```

  or:

  ```
  scale=200x100
  ```

- Specify a size abbreviation for the output size:

  ```
  scale=qcif
  ```

  which can also be written as:

```
scale=size=qcif
```

- Scale the input to 2x:

```
scale=w=2*iw:h=2*ih
```

- The above is the same as:

```
scale=2*in_w:2*in_h
```

- Scale the input to 2x with forced interlaced scaling:

```
scale=2*iw:2*ih:interl=1
```

- Scale the input to half size:

```
scale=w=iw/2:h=ih/2
```

- Increase the width, and set the height to the same size:

```
scale=3/2*iw:ow
```

- Seek for Greek harmony:

```
scale=iw:1/PHI*iw
scale=ih*PHI:ih
```

- Increase the height, and set the width to 3/2 of the height:

```
scale=w=3/2*oh:h=3/5*ih
```

- Increase the size, but make the size a multiple of the chroma subsample values:

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- Increase the width to a maximum of 500 pixels, keep the same input aspect ratio:

```
scale=w='min(500\, iw*3/2):h=-1'
```

37.56 separatefields

The `separatefields` takes a frame-based video input and splits each frame into its components fields, producing a new half height clip with twice the frame rate and twice the frame count.

This filter use field-dominance information in frame to decide which of each pair of fields to place first in the output. If it gets it wrong use setfield filter before `separatefields` filter.

37.57 setdar, setsar

The `setdar` filter sets the Display Aspect Ratio for the filter output video.

This is done by changing the specified Sample (aka Pixel) Aspect Ratio, according to the following equation:

```
DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR
```

Keep in mind that the `setdar` filter does not modify the pixel dimensions of the video frame. Also the display aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. in case of scaling or if another "setdar" or a "setsar" filter is applied.

The `setsar` filter sets the Sample (aka Pixel) Aspect Ratio for the filter output video.

Note that as a consequence of the application of this filter, the output display aspect ratio will change according to the equation above.

Keep in mind that the sample aspect ratio set by the `setsar` filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

The filters accept the following options:

‘`r, ratio, dar (setdar only), sar (setsar only)`’

Set the aspect ratio used by the filter.

The parameter can be a floating point number string, an expression, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0". In case the form "*num*:*den*" is used, the `:` character should be escaped.

‘`max`’

Set the maximum integer value to use for expressing numerator and denominator when reducing the expressed aspect ratio to a rational. Default value is `100`.

37.57.1 Examples

- To change the display aspect ratio to 16:9, specify one of the following:

```
setdar=dar=1.77777
setdar=dar=16/9
setdar=dar=1.77777
```

- To change the sample aspect ratio to 10:11, specify:

```
setsar=sar=10/11
```

- To set a display aspect ratio of 16:9, and specify a maximum integer value of 1000 in the aspect ratio reduction, use the command:

```
setdar=ratio=16/9:max=1000
```

37.58 setfield

Force field for the output video frame.

The `setfield` filter marks the interlace type field for the output frames. It does not change the input frame, but only sets the corresponding property, which affects how the frame is treated by following filters (e.g. `fieldorder` or `yadif`).

The filter accepts the following options:

‘`mode`’

Available values are:

‘`auto`’

Keep the same field property.

‘`bff`’

Mark the frame as bottom-field-first.

‘`tff`’

Mark the frame as top-field-first.

‘`prog`’

Mark the frame as progressive.

37.59 showinfo

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

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

A description of each shown parameter follows:

‘`n`’

sequential number of the input frame, starting from 0

‘`pts`’

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

‘`pts_time`’

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

‘`pos`’

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

‘`fmt`’

pixel format name

‘`sar`’

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

‘`s`’

size of the input frame, expressed in the form *width*x*height*

‘`i`’

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

‘`iskey`’

1 if the frame is a key frame, 0 otherwise

‘type’

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

‘checksum’

Adler-32 checksum (printed in hexadecimal) of all the planes of the input frame

‘plane_checksum’

Adler-32 checksum (printed in hexadecimal) of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]"

37.60 smartblur

Blur the input video without impacting the outlines.

The filter accepts the following options:

‘luma_radius, lr’

Set the luma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

‘luma_strength, ls’

Set the luma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

‘luma_threshold, lt’

Set the luma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

‘chroma_radius, cr’

Set the chroma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

‘chroma_strength, cs’

Set the chroma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

‘chroma_threshold, ct’

Set the chroma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

If a chroma option is not explicitly set, the corresponding luma value is set.

37.61 stereo3d

Convert between different stereoscopic image formats.

The filters accept the following options:

‘in’

Set stereoscopic image format of input.

Available values for input image formats are:

‘sbsl’

side by side parallel (left eye left, right eye right)

‘sbsr’

side by side crosseye (right eye left, left eye right)

‘sbs2l’

side by side parallel with half width resolution (left eye left, right eye right)

‘sbs2r’

side by side crosseye with half width resolution (right eye left, left eye right)

‘abl’

above-below (left eye above, right eye below)

‘abr’

above-below (right eye above, left eye below)

‘ab2l’

above-below with half height resolution (left eye above, right eye below)

‘ab2r’

above-below with half height resolution (right eye above, left eye below)

‘al’

alternating frames (left eye first, right eye second)

‘ar’

alternating frames (right eye first, left eye second)

Default value is ‘sbsl’.

‘out’

Set stereoscopic image format of output.

Available values for output image formats are all the input formats as well as:

‘arbg’

anaglyph red/blue gray (red filter on left eye, blue filter on right eye)

‘argg’

anaglyph red/green gray (red filter on left eye, green filter on right eye)

‘arcg’

anaglyph red/cyan gray (red filter on left eye, cyan filter on right eye)

‘arch’

anaglyph red/cyan half colored (red filter on left eye, cyan filter on right eye)

‘arcc’

anaglyph red/cyan color (red filter on left eye, cyan filter on right eye)

‘arcd’

anaglyph red/cyan color optimized with the least squares projection of dubois (red filter on left eye, cyan filter on right eye)

‘`agmg`’

anaglyph green/magenta gray (green filter on left eye, magenta filter on right eye)

‘`agmh`’

anaglyph green/magenta half colored (green filter on left eye, magenta filter on right eye)

‘`agmc`’

anaglyph green/magenta colored (green filter on left eye, magenta filter on right eye)

‘`agmd`’

anaglyph green/magenta color optimized with the least squares projection of dubois (green filter on left eye, magenta filter on right eye)

‘`aybg`’

anaglyph yellow/blue gray (yellow filter on left eye, blue filter on right eye)

‘`aybh`’

anaglyph yellow/blue half colored (yellow filter on left eye, blue filter on right eye)

‘`aybc`’

anaglyph yellow/blue colored (yellow filter on left eye, blue filter on right eye)

‘`aybd`’

anaglyph yellow/blue color optimized with the least squares projection of dubois (yellow filter on left eye, blue filter on right eye)

‘`irl`’

interleaved rows (left eye has top row, right eye starts on next row)

‘`irr`’

interleaved rows (right eye has top row, left eye starts on next row)

‘`ml`’

mono output (left eye only)

‘mr’

mono output (right eye only)

Default value is ‘arcd’.

37.61.1 Examples

- Convert input video from side by side parallel to anaglyph yellow/blue dubois:

```
stereo3d=sbsl:aybd
```

- Convert input video from above bellow (left eye above, right eye below) to side by side crosseye.

```
stereo3d=abl:sbsr
```

37.62 subtitles

Draw subtitles on top of input video using the libass library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libass`. This filter also requires a build with libavcodec and libavformat to convert the passed subtitles file to ASS (Advanced Substation Alpha) subtitles format.

The filter accepts the following options:

‘filename, f’

Set the filename of the subtitle file to read. It must be specified.

‘original_size’

Specify the size of the original video, the video for which the ASS file was composed. Due to a misdesign in ASS aspect ratio arithmetic, this is necessary to correctly scale the fonts if the aspect ratio has been changed.

‘charenc’

Set subtitles input character encoding. `subtitles` filter only. Only useful if not UTF-8.

If the first key is not specified, it is assumed that the first value specifies the ‘filename’.

For example, to render the file ‘sub.srt’ on top of the input video, use the command:

```
subtitles=sub.srt
```

which is equivalent to:

```
subtitles=filename=sub.srt
```

37.63 super2xsai

Scale the input by 2x and smooth using the Super2xSaI (Scale and Interpolate) pixel art scaling algorithm.

Useful for enlarging pixel art images without reducing sharpness.

37.64 swapuv

Swap U & V plane.

37.65 telecine

Apply telecine process to the video.

This filter accepts the following options:

‘first_field’
‘top, t’

top field first

‘bottom, b’

bottom field first The default value is top.

‘pattern’

A string of numbers representing the pulldown pattern you wish to apply. The default value is 23.

```
Some typical patterns:

NTSC output (30i):
27.5p: 32222
24p: 23 (classic)
24p: 2332 (preferred)
20p: 33
18p: 334
16p: 3444

PAL output (25i):
27.5p: 12222
24p: 222222222223 ("Euro pulldown")
16.67p: 33
16p: 33333334
```

37.66 thumbnail

Select the most representative frame in a given sequence of consecutive frames.

The filter accepts the following options:

‘`n`’

Set the frames batch size to analyze; in a set of *n* frames, the filter will pick one of them, and then handle the next batch of *n* frames until the end. Default is `100`.

Since the filter keeps track of the whole frames sequence, a bigger *n* value will result in a higher memory usage, so a high value is not recommended.

37.66.1 Examples

- Extract one picture each 50 frames:

```
thumbnail=50
```

- Complete example of a thumbnail creation with `ffmpeg`:

```
ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
```

37.67 tile

Tile several successive frames together.

The filter accepts the following options:

‘`layout`’

Set the grid size (i.e. the number of lines and columns) in the form "*w*x*h*".

‘`nb_frames`’

Set the maximum number of frames to render in the given area. It must be less than or equal to *w*x*h*. The default value is `0`, meaning all the area will be used.

‘`margin`’

Set the outer border margin in pixels.

‘`padding`’

Set the inner border thickness (i.e. the number of pixels between frames). For more advanced padding options (such as having different values for the edges), refer to the pad video filter.

37.67.1 Examples

- Produce 8x8 PNG tiles of all keyframes (‘`-skip_frame nokey`’) in a movie:

```
ffmpeg -skip_frame nokey -i file.avi -vf 'scale=128:72,tile=8x8' -an -vsync 0 keyframes%03d.png
```

  The ‘`-vsync 0`’ is necessary to prevent `ffmpeg` from duplicating each output frame to accomodate the originally detected frame rate.

- Display `5` pictures in an area of `3x2` frames, with `7` pixels between them, and `2` pixels of initial margin, using mixed flat and named options:

```
tile=3x2:nb_frames=5:padding=7:margin=2
```

37.68 tinterlace

Perform various types of temporal field interlacing.

Frames are counted starting from 1, so the first input frame is considered odd.

The filter accepts the following options:

‘`mode`’

Specify the mode of the interlacing. This option can also be specified as a value alone. See below for a list of values for this option.

Available values are:

‘`merge, 0`’

Move odd frames into the upper field, even into the lower field, generating a double height frame at half frame rate.

‘`drop_odd, 1`’

Only output even frames, odd frames are dropped, generating a frame with unchanged height at half frame rate.

‘`drop_even, 2`’

Only output odd frames, even frames are dropped, generating a frame with unchanged height at half frame rate.

‘pad, 3’

Expand each frame to full height, but pad alternate lines with black, generating a frame with double height at the same input frame rate.

‘interleave_top, 4’

Interleave the upper field from odd frames with the lower field from even frames, generating a frame with unchanged height at half frame rate.

‘interleave_bottom, 5’

Interleave the lower field from odd frames with the upper field from even frames, generating a frame with unchanged height at half frame rate.

‘interlacex2, 6’

Double frame rate with unchanged height. Frames are inserted each containing the second temporal field from the previous input frame and the first temporal field from the next input frame. This mode relies on the top_field_first flag. Useful for interlaced video displays with no field synchronisation.

Numeric values are deprecated but are accepted for backward compatibility reasons.

Default mode is merge.

‘flags’

Specify flags influencing the filter process.

Available value for *flags* is:

‘low_pass_filter, vlfp’

Enable vertical low-pass filtering in the filter. Vertical low-pass filtering is required when creating an interlaced destination from a progressive source which contains high-frequency vertical detail. Filtering will reduce interlace ’twitter’ and Moire patterning.

Vertical low-pass filtering can only be enabled for ‘mode’ *interleave_top* and *interleave_bottom*.

37.69 transpose

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

This filter accepts the following options:

‘dir’

Specify the transposition direction.

Can assume the following values:

‘0, 4, cclock_flip’

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

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

‘1, 5, clock’

Rotate by 90 degrees clockwise, that is:

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

‘2, 6, cclock’

Rotate by 90 degrees counterclockwise, that is:

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

‘3, 7, clock_flip’

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

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

For values between 4-7, the transposition is only done if the input video geometry is portrait and not landscape. These values are deprecated, the `passthrough` option should be used instead.

Numerical values are deprecated, and should be dropped in favor of symbolic constants.

‘passthrough’

Do not apply the transposition if the input geometry matches the one specified by the specified value. It accepts the following values:

‘none’

Always apply transposition.

‘portrait’

Preserve portrait geometry (when *height* >= *width*).

‘landscape’

Preserve landscape geometry (when *width* >= *height*).

Default value is none.

For example to rotate by 90 degrees clockwise and preserve portrait layout:

```
transpose=dir=1:passthrough=portrait
```

The command above can also be specified as:

```
transpose=1:portrait
```

37.70 trim

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

This filter accepts the following options:

‘start’

Timestamp (in seconds) of the start of the kept section. I.e. the frame with the timestamp *start* will be the first frame in the output.

‘end’

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

‘start_pts’

Same as *start*, except this option sets the start timestamp in timebase units instead of seconds.

‘end_pts’

Same as *end*, except this option sets the end timestamp in timebase units instead of seconds.

‘duration’

Maximum duration of the output in seconds.

‘start_frame’

Number of the first frame that should be passed to output.

‘end_frame’

Number of the first frame that should be dropped.

Note that the first two sets of the start/end options and the ‘duration’ option look at the frame timestamp, while the _frame variants simply count the frames that pass through the filter. Also note that this filter does not modify the timestamps. If you wish that the output timestamps start at zero, insert a setpts filter after the trim filter.

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

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

Examples:

- drop everything except the second minute of input

```
ffmpeg -i INPUT -vf trim=60:120
```

- keep only the first second

```
ffmpeg -i INPUT -vf trim=duration=1
```

37.71 unsharp

Sharpen or blur the input video.

It accepts the following parameters:

‘luma_msize_x, lx’

Set the luma matrix horizontal size. It must be an odd integer between 3 and 63, default value is 5.

‘luma_msize_y, ly’

Set the luma matrix vertical size. It must be an odd integer between 3 and 63, default value is 5.

‘`luma_amount, la`’

Set the luma effect strength. It can be a float number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 1.0.

‘`chroma_msize_x, cx`’

Set the chroma matrix horizontal size. It must be an odd integer between 3 and 63, default value is 5.

‘`chroma_msize_y, cy`’

Set the chroma matrix vertical size. It must be an odd integer between 3 and 63, default value is 5.

‘`chroma_amount, ca`’

Set the chroma effect strength. It can be a float number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 0.0.

‘`opencl`’

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

All parameters are optional and default to the equivalent of the string ’5:5:1.0:5:5:0.0’.

37.71.1 Examples

- Apply strong luma sharpen effect:

```
unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5
```

- Apply strong blur of both luma and chroma parameters:

```
unsharp=7:7:-2:7:7:-2
```

37.72 vidstabdetect

Analyze video stabilization/deshaking. Perform pass 1 of 2, see vidstabtransform for pass 2.

This filter generates a file with relative translation and rotation transform information about subsequent frames, which is then used by the vidstabtransform filter.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

This filter accepts the following options:

‘`result`’

Set the path to the file used to write the transforms information. Default value is ‘`transforms.trf`’.

‘`shakiness`’

Set how shaky the video is and how quick the camera is. It accepts an integer in the range 1-10, a value of 1 means little shakiness, a value of 10 means strong shakiness. Default value is 5.

‘`accuracy`’

Set the accuracy of the detection process. It must be a value in the range 1-15. A value of 1 means low accuracy, a value of 15 means high accuracy. Default value is 9.

‘`stepsize`’

Set stepsize of the search process. The region around minimum is scanned with 1 pixel resolution. Default value is 6.

‘`mincontrast`’

Set minimum contrast. Below this value a local measurement field is discarded. Must be a floating point value in the range 0-1. Default value is 0.3.

‘`tripod`’

Set reference frame number for tripod mode.

If enabled, the motion of the frames is compared to a reference frame in the filtered stream, identified by the specified number. The idea is to compensate all movements in a more-or-less static scene and keep the camera view absolutely still.

If set to 0, it is disabled. The frames are counted starting from 1.

‘`show`’

Show fields and transforms in the resulting frames. It accepts an integer in the range 0-2. Default value is 0, which disables any visualization.

37.72.1 Examples

- Use default values:

```
vidstabdetect
```

- Analyze strongly shaky movie and put the results in file ‘`mytransforms.trf`’:

```
vidstabdetect=shakiness=10:accuracy=15:result="mytransforms.trf"
```

- Visualize the result of internal transformations in the resulting video:

```
vidstabdetect=show=1
```

- Analyze a video with medium shakiness using `ffmpeg`:

```
ffmpeg -i input -vf vidstabdetect=shakiness=5:show=1 dummy.avi
```

37.73 vidstabtransform

Video stabilization/deshaking: pass 2 of 2, see vidstabdetect for pass 1.

Read a file with transform information for each frame and apply/compensate them. Together with the vidstabdetect filter this can be used to deshake videos. See also http://public.hronopik.de/vid.stab. It is important to also use the unsharp filter, see below.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

This filter accepts the following options:

‘`input`’

path to the file used to read the transforms (default: ‘`transforms.trf`’)

‘`smoothing`’

number of frames (value*2 + 1) used for lowpass filtering the camera movements (default: 10). For example a number of 10 means that 21 frames are used (10 in the past and 10 in the future) to smoothen the motion in the video. A larger values leads to a smoother video, but limits the acceleration of the camera (pan/tilt movements).

‘maxshift’

maximal number of pixels to translate frames (default: -1 no limit)

‘maxangle’

maximal angle in radians (degree*PI/180) to rotate frames (default: -1 no limit)

‘crop’

How to deal with borders that may be visible due to movement compensation. Available values are:

‘keep’

keep image information from previous frame (default)

‘black’

fill the border black

‘invert’

‘0’

keep transforms normal (default)

‘1’

invert transforms

‘relative’

consider transforms as

‘0’

absolute

‘1’

relative to previous frame (default)

‘zoom’

percentage to zoom (default: 0)

‘>0’

zoom in

‘<0’

zoom out

‘optzoom’

if 1 then optimal zoom value is determined (default). Optimal zoom means no (or only little) border should be visible. Note that the value given at zoom is added to the one calculated here.

‘interpol’

type of interpolation

Available values are:

‘no’

no interpolation

‘linear’

linear only horizontal

‘bilinear’

linear in both directions (default)

‘bicubic’

cubic in both directions (slow)

‘tripod’

virtual tripod mode means that the video is stabilized such that the camera stays stationary. Use also `tripod` option of vidstabdetect.

‘0’

off (default)

‘1’

virtual tripod mode: equivalent to `relative=0:smoothing=0`

37.73.1 Examples

- typical call with default default values: (note the unsharp filter which is always recommended)

```
ffmpeg -i inp.mpeg -vf vidstabtransform,unsharp=5:5:0.8:3:3:0.4 inp_stabilized.mpeg
```

- zoom in a bit more and load transform data from a given file

```
vidstabtransform=zoom=5:input="mytransforms.trf"
```

- smoothen the video even more

```
vidstabtransform=smoothing=30
```

37.74 vflip

Flip the input video vertically.

For example, to vertically flip a video with `ffmpeg`:

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

37.75 yadif

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

This filter accepts the following options:

‘`mode`’

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

‘`0, send_frame`’

output 1 frame for each frame

‘`1, send_field`’

output 1 frame for each field

‘`2, send_frame_nospatial`’

like `send_frame` but skip spatial interlacing check

‘`3, send_field_nospatial`’

like `send_field` but skip spatial interlacing check

Default value is `send_frame`.

‘parity’

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

‘0, tff’

assume top field first

‘1, bff’

assume bottom field first

‘-1, auto’

enable automatic detection

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

‘deint’

Specify which frames to deinterlace. Accept one of the following values:

‘0, all’

deinterlace all frames

‘1, interlaced’

only deinterlace frames marked as interlaced

Default value is all.

38. Video Sources

Below is a description of the currently available video sources.

38.1 buffer

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

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

This source accepts the following options:

‘video_size’

Specify the size (width and height) of the buffered video frames.

‘width’

Input video width.

‘height’

Input video height.

‘pix_fmt’

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

‘time_base’

Specify the timebase assumed by the timestamps of the buffered frames.

‘frame_rate’

Specify the frame rate expected for the video stream.

‘pixel_aspect, sar’

Specify the sample aspect ratio assumed by the video frames.

‘sws_param’

Specify the optional parameters to be used for the scale filter which is automatically inserted when an input change is detected in the input size or format.

For example:

```
buffer=width=320:height=240:pix_fmt=yuv410p:time_base=1/24:sar=1
```

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

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

Alternatively, the options can be specified as a flat string, but this syntax is deprecated:

width:*height*:*pix_fmt*:*time_base.num*:*time_base.den*:*pixel_aspect.num*:*pixel_aspect.den*[:*sws_param*]

38.2 cellauto

Create a pattern generated by an elementary cellular automaton.

The initial state of the cellular automaton can be defined through the ‘`filename`’, and ‘`pattern`’ options. If such options are not specified an initial state is created randomly.

At each new frame a new row in the video is filled with the result of the cellular automaton next generation. The behavior when the whole frame is filled is defined by the ‘`scroll`’ option.

This source accepts the following options:

‘`filename, f`’

> Read the initial cellular automaton state, i.e. the starting row, from the specified file. In the file, each non-whitespace character is considered an alive cell, a newline will terminate the row, and further characters in the file will be ignored.

‘`pattern, p`’

> Read the initial cellular automaton state, i.e. the starting row, from the specified string.
>
> Each non-whitespace character in the string is considered an alive cell, a newline will terminate the row, and further characters in the string will be ignored.

‘`rate, r`’

> Set the video rate, that is the number of frames generated per second. Default is 25.

‘`random_fill_ratio, ratio`’

> Set the random fill ratio for the initial cellular automaton row. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI.
>
> This option is ignored when a file or a pattern is specified.

‘`random_seed, seed`’

> Set the seed for filling randomly the initial row, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

‘`rule`’

> Set the cellular automaton rule, it is a number ranging from 0 to 255. Default value is 110.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ or ‘`pattern`’ is specified, the size is set by default to the width of the specified initial state row, and the height is set to *width* * PHI.

If ‘`size`’ is set, it must contain the width of the specified pattern string, and the specified pattern will be centered in the larger row.

If a filename or a pattern string is not specified, the size value defaults to "320x518" (used for a randomly generated initial state).

‘`scroll`’

If set to 1, scroll the output upward when all the rows in the output have been already filled. If set to 0, the new generated row will be written over the top row just after the bottom row is filled. Defaults to 1.

‘`start_full, full`’

If set to 1, completely fill the output with generated rows before outputting the first frame. This is the default behavior, for disabling set the value to 0.

‘`stitch`’

If set to 1, stitch the left and right row edges together. This is the default behavior, for disabling set the value to 0.

38.2.1 Examples

- Read the initial state from ‘`pattern`’, and specify an output of size 200x400.

```
cellauto=f=pattern:s=200x400
```

- Generate a random initial row with a width of 200 cells, with a fill ratio of 2/3:

```
cellauto=ratio=2/3:s=200x200
```

- Create a pattern generated by rule 18 starting by a single alive cell centered on an initial row with width 100:

```
cellauto=p=@:s=100x400:full=0:rule=18
```

- Specify a more elaborated initial pattern:

```
cellauto=p='@@ @ @@':s=100x400:full=0:rule=18
```

38.3 mandelbrot

Generate a Mandelbrot set fractal, and progressively zoom towards the point specified with *start_x* and *start_y*.

This source accepts the following options:

‘`end_pts`’

Set the terminal pts value. Default value is 400.

‘`end_scale`’

Set the terminal scale value. Must be a floating point value. Default value is 0.3.

‘`inner`’

Set the inner coloring mode, that is the algorithm used to draw the Mandelbrot fractal internal region.

It shall assume one of the following values:

‘`black`’

Set black mode.

‘`convergence`’

Show time until convergence.

‘`mincol`’

Set color based on point closest to the origin of the iterations.

‘`period`’

Set period mode.

Default value is *mincol*.

‘`bailout`’

Set the bailout value. Default value is 10.0.

‘`maxiter`’

Set the maximum of iterations performed by the rendering algorithm. Default value is 7189.

‘`outer`’

Set outer coloring mode. It shall assume one of following values:

‘`iteration_count`’

Set iteration cound mode.

‘`normalized_iteration_count`’

set normalized iteration count mode.

Default value is *normalized_iteration_count*.

‘`rate, r`’

Set frame rate, expressed as number of frames per second. Default value is "25".

‘`size, s`’

Set frame size. Default value is "640x480".

‘`start_scale`’

Set the initial scale value. Default value is 3.0.

‘`start_x`’

Set the initial x position. Must be a floating point value between -100 and 100. Default value is -0.743643887037158704752191506114774.

‘`start_y`’

Set the initial y position. Must be a floating point value between -100 and 100. Default value is -0.131825904205311970493132056385139.

38.4 mptestsrc

Generate various test patterns, as generated by the MPlayer test filter.

The size of the generated video is fixed, and is 256x256. This source is useful in particular for testing encoding features.

This source accepts the following options:

‘rate, r’

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

‘duration, d’

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

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

See also the function `av_parse_time()`.

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

‘test, t’

Set the number or the name of the test to perform. Supported tests are:

‘dc_luma’
‘dc_chroma’
‘freq_luma’
‘freq_chroma’
‘amp_luma’
‘amp_chroma’
‘cbp’
‘mv’
‘ring1’
‘ring2’
‘all’

Default value is "all", which will cycle through the list of all tests.

For example the following:

```
testsrc=t=dc_luma
```

will generate a "dc_luma" test pattern.

38.5 frei0r_src

Provide a frei0r source.

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

This source accepts the following options:

‘`size`’

The size of the video to generate, may be a string of the form *width*x*height* or a frame size abbreviation.

‘`framerate`’

Framerate of the generated video, may be a string of the form *num*/*den* or a frame rate abbreviation.

‘`filter_name`’

The name to the frei0r source to load. For more information regarding frei0r and how to set the parameters read the section frei0r in the description of the video filters.

‘`filter_params`’

A ’|’-separated list of parameters to pass to the frei0r source.

For example, to generate a frei0r partik0l source with size 200x200 and frame rate 10 which is overlayed on the overlay filter main input:

```
frei0r_src=size=200x200:framerate=10:filter_name=partik0l:filter_params=1234 [overlay]; [in][overlay] overlay
```

38.6 life

Generate a life pattern.

This source is based on a generalization of John Conway’s life game.

The sourced input represents a life grid, each pixel represents a cell which can be in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.

At each interaction the grid evolves according to the adopted rule, which specifies the number of neighbor alive cells which will make a cell stay alive or born. The ‘`rule`’ option allows to specify the rule to adopt.

This source accepts the following options:

‘`filename, f`’

Set the file from which to read the initial grid state. In the file, each non-whitespace character is considered an alive cell, and newline is used to delimit the end of each row.

If this option is not specified, the initial grid is generated randomly.

‘`rate, r`’

Set the video rate, that is the number of frames generated per second. Default is 25.

‘`random_fill_ratio, ratio`’

Set the random fill ratio for the initial random grid. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI. It is ignored when a file is specified.

‘`random_seed, seed`’

Set the seed for filling the initial random grid, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

‘`rule`’

Set the life rule.

A rule can be specified with a code of the kind "S*NS*/B*NB*", where *NS* and *NB* are sequences of numbers in the range 0-8, *NS* specifies the number of alive neighbor cells which make a live cell stay alive, and *NB* the number of alive neighbor cells which make a dead cell to become alive (i.e. to "born"). "s" and "b" can be used in place of "S" and "B", respectively.

Alternatively a rule can be specified by an 18-bits integer. The 9 high order bits are used to encode the next cell state if it is alive for each number of neighbor alive cells, the low order bits specify the rule for "borning" new cells. Higher order bits encode for an higher number of neighbor cells. For example the number 6153 = `(12<<9)+9` specifies a stay alive rule of 12 and a born rule of 9, which corresponds to "S23/B03".

Default value is "S23/B3", which is the original Conway’s game of life rule, and will keep a cell alive if it has 2 or 3 neighbor alive cells, and will born a new cell if there are three alive cells around a dead cell.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ is specified, the size is set by default to the same size of the input file. If ‘`size`’ is set, it must contain the size specified in the input file, and the initial grid defined in that file is centered in the larger resulting area.

If a filename is not specified, the size value defaults to "320x240" (used for a randomly generated initial grid).

‘stitch’

If set to 1, stitch the left and right grid edges together, and the top and bottom edges also. Defaults to 1.

‘mold’

Set cell mold speed. If set, a dead cell will go from ‘death_color’ to ‘mold_color’ with a step of ‘mold’. ‘mold’ can have a value from 0 to 255.

‘life_color’

Set the color of living (or new born) cells.

‘death_color’

Set the color of dead cells. If ‘mold’ is set, this is the first color used to represent a dead cell.

‘mold_color’

Set mold color, for definitely dead and moldy cells.

38.6.1 Examples

- Read a grid from ‘pattern’, and center it on a grid of size 300x300 pixels:

```
life=f=pattern:s=300x300
```

- Generate a random grid of size 200x200, with a fill ratio of 2/3:

```
life=ratio=2/3:s=200x200
```

- Specify a custom rule for evolving a randomly generated grid:

```
life=rule=S14/B34
```

- Full example with slow death effect (mold) using ffplay:

```
ffplay -f lavfi life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
```

38.7 color, nullsrc, rgbtestsrc, smptebars, smptehdbars, testsrc

The color source provides an uniformly colored input.

The `nullsrc` source returns unprocessed video frames. It is mainly useful to be employed in analysis / debugging tools, or as the source for filters which ignore the input data.

The `rgbtestsrc` source generates an RGB test pattern useful for detecting RGB vs BGR issues. You should see a red, green and blue stripe from top to bottom.

The `smptebars` source generates a color bars pattern, based on the SMPTE Engineering Guideline EG 1-1990.

The `smptehdbars` source generates a color bars pattern, based on the SMPTE RP 219-2002.

The `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

The sources accept the following options:

‘`color, c`’

Specify the color of the source, only used in the `color` source. It can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence, possibly followed by an alpha specifier. The default value is "black".

‘`size, s`’

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation. The default value is "320x240".

‘`rate, r`’

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

‘`sar`’

Set the sample aspect ratio of the sourced video.

‘`duration, d`’

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

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

See also the function `av_parse_time()`.

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

‘decimals, n’

Set the number of decimals to show in the timestamp, only used in the testsrc source.

The displayed timestamp value will correspond to the original timestamp value multiplied by the power of 10 of the specified value. Default value is 0.

For example the following:

```
testsrc=duration=5.3:size=qcif:rate=10
```

will generate a video with a duration of 5.3 seconds, with size 176x144 and a frame rate of 10 frames per second.

The following graph description will generate a red source with an opacity of 0.2, with size "qcif" and a frame rate of 10 frames per second.

```
color=c=red@0.2:s=qcif:r=10
```

If the input content is to be ignored, nullsrc can be used. The following command generates noise in the luminance plane by employing the geq filter:

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

39. Video Sinks

Below is a description of the currently available video sinks.

39.1 buffersink

Buffer video frames, and make them available to the end of the filter graph.

This sink is mainly intended for a programmatic use, in particular through the interface defined in ‘libavfilter/buffersink.h’ or the options system.

It accepts a pointer to an AVBufferSinkContext structure, which defines the incoming buffers’ formats, to be passed as the opaque parameter to avfilter_init_filter for initialization.

39.2 nullsink

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

40. Multimedia Filters

Below is a description of the currently available multimedia filters.

40.1 concat

Concatenate audio and video streams, joining them together one after the other.

The filter works on segments of synchronized video and audio streams. All segments must have the same number of streams of each type, and that will also be the number of streams at output.

The filter accepts the following options:

‘n’

Set the number of segments. Default is 2.

‘v’

Set the number of output video streams, that is also the number of video streams in each segment. Default is 1.

‘a’

Set the number of output audio streams, that is also the number of video streams in each segment. Default is 0.

‘unsafe’

Activate unsafe mode: do not fail if segments have a different format.

The filter has *v*+*a* outputs: first *v* video outputs, then *a* audio outputs.

There are *n*x(*v*+*a*) inputs: first the inputs for the first segment, in the same order as the outputs, then the inputs for the second segment, etc.

Related streams do not always have exactly the same duration, for various reasons including codec frame size or sloppy authoring. For that reason, related synchronized streams (e.g. a video and its audio track) should be concatenated at once. The concat filter will use the duration of the longest stream in each segment (except the last one), and if necessary pad shorter audio streams with silence.

For this filter to work correctly, all segments must start at timestamp 0.

All corresponding streams must have the same parameters in all segments; the filtering system will automatically select a common pixel format for video streams, and a common sample format, sample rate and channel layout for audio streams, but other settings, such as resolution, must be converted explicitly by the user.

Different frame rates are acceptable but will result in variable frame rate at output; be sure to configure the output file to handle it.

40.1.1 Examples

- Concatenate an opening, an episode and an ending, all in bilingual version (video in stream 0, audio in streams 1 and 2):

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
  '[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2]
   concat=n=3:v=1:a=2 [v] [a1] [a2]' \
  -map '[v]' -map '[a1]' -map '[a2]' output.mkv
```

- Concatenate two parts, handling audio and video separately, using the (a)movie sources, and adjusting the resolution:

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;
movie=part2.mp4, scale=512:288 [v2] ; amovie=part2.mp4 [a2] ;
[v1] [v2] concat [outv] ; [a1] [a2] concat=v=0:a=1 [outa]
```

  Note that a desync will happen at the stitch if the audio and video streams do not have exactly the same duration in the first file.

40.2 ebur128

EBU R128 scanner filter. This filter takes an audio stream as input and outputs it unchanged. By default, it logs a message at a frequency of 10Hz with the Momentary loudness (identified by `M`), Short-term loudness (`S`), Integrated loudness (`I`) and Loudness Range (`LRA`).

The filter also has a video output (see the *video* option) with a real time graph to observe the loudness evolution. The graphic contains the logged message mentioned above, so it is not printed anymore when this option is set, unless the verbose logging is set. The main graphing area contains the short-term loudness (3 seconds of analysis), and the gauge on the right is for the momentary loudness (400 milliseconds).

More information about the Loudness Recommendation EBU R128 on http://tech.ebu.ch/loudness.

The filter accepts the following options:

‘`video`’

  Activate the video output. The audio stream is passed unchanged whether this option is set or no. The video stream will be the first output stream if activated. Default is `0`.

‘`size`’

Set the video size. This option is for video only. Default and minimum resolution is `640x480`.

‘`meter`’

Set the EBU scale meter. Default is `9`. Common values are `9` and `18`, respectively for EBU scale meter +9 and EBU scale meter +18. Any other integer value between this range is allowed.

‘`metadata`’

Set metadata injection. If set to `1`, the audio input will be segmented into 100ms output frames, each of them containing various loudness information in metadata. All the metadata keys are prefixed with `lavfi.r128.`.

Default is `0`.

‘`framelog`’

Force the frame logging level.

Available values are:

‘`info`’

information logging level

‘`verbose`’

verbose logging level

By default, the logging level is set to *info*. If the ‘`video`’ or the ‘`metadata`’ options are set, it switches to *verbose*.

40.2.1 Examples

- Real-time graph using `ffplay`, with a EBU scale meter +18:

```
ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
```

- Run an analysis with `ffmpeg`:

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

40.3 interleave, ainterleave

Temporally interleave frames from several inputs.

`interleave` works with video inputs, `ainterleave` with audio.

These filters read frames from several inputs and send the oldest queued frame to the output.

Input streams must have a well defined, monotonically increasing frame timestamp values.

In order to submit one frame to output, these filters need to enqueue at least one frame for each input, so they cannot work in case one input is not yet terminated and will not receive incoming frames.

For example consider the case when one input is a `select` filter which always drop input frames. The `interleave` filter will keep reading from that input, but it will never be able to send new frames to output until the input will send an end-of-stream signal.

Also, depending on inputs synchronization, the filters will drop frames in case one input receives more frames than the other ones, and the queue is already filled.

These filters accept the following options:

‘`nb_inputs, n`’

Set the number of different inputs, it is 2 by default.

40.3.1 Examples

- Interleave frames belonging to different streams using `ffmpeg`:

```
ffmpeg -i bambi.avi -i pr0n.mkv -filter_complex "[0:v][1:v] interleave" out.avi
```

- Add flickering blur effect:

```
select='if(gt(random(0), 0.2), 1, 2)':n=2 [tmp], boxblur=2:2, [tmp] interleave
```

40.4 perms, aperms

Set read/write permissions for the output frames.

These filters are mainly aimed at developers to test direct path in the following filter in the filtergraph.

The filters accept the following options:

‘mode’

Select the permissions mode.

It accepts the following values:

‘none’

Do nothing. This is the default.

‘ro’

Set all the output frames read-only.

‘rw’

Set all the output frames directly writable.

‘toggle’

Make the frame read-only if writable, and writable if read-only.

‘random’

Set each output frame read-only or writable randomly.

‘seed’

Set the seed for the *random* mode, must be an integer included between `0` and `UINT32_MAX`. If not specified, or if explicitly set to `-1`, the filter will try to use a good random seed on a best effort basis.

Note: in case of auto-inserted filter between the permission filter and the following one, the permission might not be received as expected in that following filter. Inserting a format or aformat filter before the perms/aperms filter can avoid this problem.

40.5 select, aselect

Select frames to pass in output.

This filter accepts the following options:

‘expr, e’

Set expression, which is evaluated for each input frame.

If the expression is evaluated to zero, the frame is discarded.

If the evaluation result is negative or NaN, the frame is sent to the first output; otherwise it is sent to the output with index `ceil(val)-1`, assuming that the input index starts from 0.

For example a value of `1.2` corresponds to the output with index `ceil(1.2)-1 = 2-1 = 1`, that is the second output.

‘`outputs, n`’

Set the number of outputs. The output to which to send the selected frame is based on the result of the evaluation. Default value is 1.

The expression can contain the following constants:

‘`n`’

the sequential number of the filtered frame, starting from 0

‘`selected_n`’

the sequential number of the selected frame, starting from 0

‘`prev_selected_n`’

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

‘`TB`’

timebase of the input timestamps

‘`pts`’

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

‘`t`’

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

‘`prev_pts`’

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

‘`prev_selected_pts`’

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

‘`prev_selected_t`’

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

‘start_pts’

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

‘start_t’

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

‘pict_type *(video only)*’

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

‘I’
‘P’
‘B’
‘S’
‘SI’
‘SP’
‘BI’

‘interlace_type *(video only)*’

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

‘PROGRESSIVE’

the frame is progressive (not interlaced)

‘TOPFIRST’

the frame is top-field-first

‘BOTTOMFIRST’

the frame is bottom-field-first

‘consumed_sample_n *(audio only)*’

the number of selected samples before the current frame

‘samples_n *(audio only)*’

the number of samples in the current frame

‘sample_rate *(audio only)*’

the input sample rate

‘key’

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

‘pos’

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

‘scene *(video only)*’

value between 0 and 1 to indicate a new scene; a low value reflects a low probability for the current frame to introduce a new scene, while a higher value means the current frame is more likely to be one (see the example below)

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

40.5.1 Examples

- Select all frames in input:

```
select
```

  The example above is the same as:

```
select=1
```

- Skip all frames:

```
select=0
```

- Select only I-frames:

```
select='eq(pict_type\,I)'
```

- Select one frame every 100:

```
select='not(mod(n\,100))'
```

- Select only frames contained in the 10-20 time interval:

```
select='gte(t\,10)*lte(t\,20)'
```

- Select only I frames contained in the 10-20 time interval:

```
select='gte(t\,10)*lte(t\,20)*eq(pict_type\,I)'
```

- Select frames with a minimum distance of 10 seconds:

```
select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
```

- Use aselect to select only audio frames with samples number > 100:

```
aselect='gt(samples_n\,100)'
```

- Create a mosaic of the first scenes:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

  Comparing *scene* against a value between 0.3 and 0.5 is generally a sane choice.

- Send even and odd frames to separate outputs, and compose them:

```
select=n=2:e='mod(n, 2)+1' [odd][even]; [odd] pad=h=2*ih [tmp]; [tmp][even] overlay=y=h
```

40.6 sendcmd, asendcmd

Send commands to filters in the filtergraph.

These filters read commands to be sent to other filters in the filtergraph.

`sendcmd` must be inserted between two video filters, `asendcmd` must be inserted between two audio filters, but apart from that they act the same way.

The specification of commands can be provided in the filter arguments with the *commands* option, or in a file specified by the *filename* option.

These filters accept the following options:

`'commands, c'`

Set the commands to be read and sent to the other filters.

`'filename, f'`

Set the filename of the commands to be read and sent to the other filters.

40.6.1 Commands syntax

A commands description consists of a sequence of interval specifications, comprising a list of commands to be executed when a particular event related to that interval occurs. The occurring event is typically the current frame time entering or leaving a given time interval.

An interval is specified by the following syntax:

```
START[-END] COMMANDS;
```

The time interval is specified by the *START* and *END* times. *END* is optional and defaults to the maximum time.

The current frame time is considered within the specified interval if it is included in the interval [*START*, *END*), that is when the time is greater or equal to *START* and is lesser than *END*.

COMMANDS consists of a sequence of one or more command specifications, separated by ",", relating to that interval. The syntax of a command specification is given by:

```
[FLAGS] TARGET COMMAND ARG
```

FLAGS is optional and specifies the type of events relating to the time interval which enable sending the specified command, and must be a non-null sequence of identifier flags separated by "+" or "|" and enclosed between "[" and "]".

The following flags are recognized:

`‘enter’`

The command is sent when the current frame timestamp enters the specified interval. In other words, the command is sent when the previous frame timestamp was not in the given interval, and the current is.

`‘leave’`

The command is sent when the current frame timestamp leaves the specified interval. In other words, the command is sent when the previous frame timestamp was in the given interval, and the current is not.

If *FLAGS* is not specified, a default value of `[enter]` is assumed.

TARGET specifies the target of the command, usually the name of the filter class or a specific filter instance name.

COMMAND specifies the name of the command for the target filter.

ARG is optional and specifies the optional list of argument for the given *COMMAND*.

Between one interval specification and another, whitespaces, or sequences of characters starting with # until the end of line, are ignored and can be used to annotate comments.

A simplified BNF description of the commands specification syntax follows:

```
COMMAND_FLAG  ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND       ::= ["[" COMMAND_FLAGS "]"] TARGET COMMAND [ARG]
COMMANDS      ::= COMMAND [,COMMANDS]
INTERVAL      ::= START[-END] COMMANDS
INTERVALS     ::= INTERVAL[;INTERVALS]
```

40.6.2 Examples

- Specify audio tempo change at second 4:

```
asendcmd=c='4.0 atempo tempo 1.5',atempo
```

- Specify a list of drawtext and hue commands in a file.

```
# show text in the interval 5-10
5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
         [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';

# desaturate the image in the interval 15-20
15.0-20.0 [enter] hue s 0,
          [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
          [leave] hue s 1,
          [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';

# apply an exponential saturation fade-out effect, starting from time 25
25 [enter] hue s exp(25-t)
```

A filtergraph allowing to read and process the above command list stored in a file '`test.cmd`', can be specified with:

```
sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text='',hue
```

40.7 setpts, asetpts

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

`setpts` works on video frames, `asetpts` on audio frames.

This filter accepts the following options:

‘expr’

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

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

‘FRAME_RATE’

frame rate, only defined for constant frame-rate video

‘PTS’

the presentation timestamp in input

‘N’

the count of the input frame, starting from 0.

‘NB_CONSUMED_SAMPLES’

the number of consumed samples, not including the current frame (only audio)

‘NB_SAMPLES’

the number of samples in the current frame (only audio)

‘SAMPLE_RATE’

audio sample rate

‘STARTPTS’

the PTS of the first frame

‘STARTT’

the time in seconds of the first frame

‘INTERLACED’

tell if the current frame is interlaced

‘T’

the time in seconds of the current frame

‘TB’

the time base

‘POS’

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

‘PREV_INPTS’

previous input PTS

‘PREV_INT’

previous input time in seconds

‘PREV_OUTPTS’

previous output PTS

‘PREV_OUTT’

previous output time in seconds

‘RTCTIME’

wallclock (RTC) time in microseconds. This is deprecated, use time(0) instead.

‘RTCSTART’

wallclock (RTC) time at the start of the movie in microseconds

40.7.1 Examples

- Start counting PTS from zero

```
setpts=PTS-STARTPTS
```

- Apply fast motion effect:

```
setpts=0.5*PTS
```

- Apply slow motion effect:

```
setpts=2.0*PTS
```

- Set fixed rate of 25 frames per second:

```
setpts=N/(25*TB)
```

- Set fixed rate 25 fps with some jitter:

```
setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'
```

- Apply an offset of 10 seconds to the input PTS:

```
setpts=PTS+10/TB
```

- Generate timestamps from a "live source" and rebase onto the current timebase:

```
setpts='(RTCTIME - RTCSTART) / (TB * 1000000)'
```

40.8 settb, asettb

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

This filter accepts the following options:

`'expr, tb'`

The expression which is evaluated into the output timebase.

The value for '`tb`' is an arithmetic expression representing a rational. The expression can contain the constants "AVTB" (the default timebase), "intb" (the input timebase) and "sr" (the sample rate, audio only). Default value is "intb".

40.8.1 Examples

- Set the timebase to 1/25:

```
settb=expr=1/25
```

- Set the timebase to 1/10:

```
settb=expr=0.1
```

- Set the timebase to 1001/1000:

```
settb=1+0.001
```

- Set the timebase to 2*intb:

```
settb=2*intb
```

- Set the default timebase value:

```
settb=AVTB
```

40.9 showspectrum

Convert input audio to a video output, representing the audio frequency spectrum.

The filter accepts the following options:

‘`size, s`’

Specify the video size for the output. Default value is `640x512`.

‘`slide`’

Specify if the spectrum should slide along the window. Default value is `0`.

‘`mode`’

Specify display mode.

It accepts the following values:

‘`combined`’

all channels are displayed in the same row

‘`separate`’

all channels are displayed in separate rows

Default value is ‘`combined`’.

‘`color`’

Specify display color mode.

It accepts the following values:

‘`channel`’

each channel is displayed in a separate color

‘intensity’

each channel is is displayed using the same color scheme

Default value is ‘channel’.

‘scale’

Specify scale used for calculating intensity color values.

It accepts the following values:

‘lin’

linear

‘sqrt’

square root, default

‘cbrt’

cubic root

‘log’

logarithmic

Default value is ‘sqrt’.

‘saturation’

Set saturation modifier for displayed colors. Negative values provide alternative color scheme. 0 is no saturation at all. Saturation must be in [-10.0, 10.0] range. Default value is 1.

The usage is very similar to the showwaves filter; see the examples in that section.

40.9.1 Examples

- Large window with logarithmic color scaling:

```
showspectrum=s=1280x480:scale=log
```

- Complete example for a colored and sliding spectrum per channel using ffplay:

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
             [a] showspectrum=mode=separate:color=intensity:slide=1:scale=cbrt [out0]'
```

40.10 showwaves

Convert input audio to a video output, representing the samples waves.

The filter accepts the following options:

‘`size, s`’

Specify the video size for the output. Default value is "600x240".

‘`mode`’

Set display mode.

Available values are:

‘`point`’

Draw a point for each sample.

‘`line`’

Draw a vertical line for each sample.

Default value is `point`.

‘`n`’

Set the number of samples which are printed on the same column. A larger value will decrease the frame rate. Must be a positive integer. This option can be set only if the value for *rate* is not explicitly specified.

‘`rate, r`’

Set the (approximate) output frame rate. This is done by setting the option *n*. Default value is "25".

40.10.1 Examples

- Output the input file audio and the corresponding video representation at the same time:

```
amovie=a.mp3,asplit[out0],showwaves[out1]
```

- Create a synthetic signal and show it with showwaves, forcing a frame rate of 30 frames per second:

```
aevalsrc=sin(1*2*PI*t)*sin(880*2*PI*t):cos(2*PI*200*t),asplit[out0],showwaves=r=30[out1]
```

40.11 split, asplit

Split input into several identical outputs.

`asplit` works with audio input, `split` with video.

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

40.11.1 Examples

- Create two separate outputs from the same input:

```
[in] split [out0][out1]
```

- To create 3 or more outputs, you need to specify the number of outputs, like in:

```
[in] asplit=3 [out0][out1][out2]
```

- Create two separate outputs from the same input, one cropped and one padded:

```
[in] split [splitout1][splitout2];
[splitout1] crop=100:100:0:0    [cropout];
[splitout2] pad=200:200:100:100 [padout];
```

- Create 5 copies of the input audio with `ffmpeg`:

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

41. Multimedia Sources

Below is a description of the currently available multimedia sources.

41.1 amovie

This is the same as movie source, except it selects an audio stream by default.

41.2 movie

Read audio and/or video stream(s) from a movie container.

This filter accepts the following options:

‘`filename`’

The name of the resource to read (not necessarily a file but also a device or a stream accessed through some protocol).

‘`format_name, f`’

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

‘`seek_point, sp`’

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

‘`streams, s`’

Specifies the streams to read. Several streams can be specified, separated by "+". The source will then have as many outputs, in the same order. The syntax is explained in the “Stream specifiers” section in the ffmpeg manual. Two special names, "dv" and "da" specify respectively the default (best suited) video and audio stream. Default is "dv", or "da" if the filter is called as "amovie".

‘`stream_index, si`’

Specifies the index of the video stream to read. If the value is -1, the best suited video stream will be automatically selected. Default value is "-1". Deprecated. If the filter is called "amovie", it will select audio instead of video.

‘`loop`’

Specifies how many times to read the stream in sequence. If the value is less than 1, the stream will be read again and again. Default value is "1".

Note that when the movie is looped the source timestamps are not changed, so it will generate non monotonically increasing timestamps.

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

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

41.2.1 Examples

- Skip 3.2 seconds from the start of the avi file in.avi, and overlay it on top of the input labelled as "in":

```
movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [over];
[in] setpts=PTS-STARTPTS [main];
[main][over] overlay=16:16 [out]
```

- Read from a video4linux2 device, and overlay it on top of the input labelled as "in":

```
movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [over];
[in] setpts=PTS-STARTPTS [main];
[main][over] overlay=16:16 [out]
```

- Read the first video stream and the audio stream with id 0x81 from dvd.vob; the video is connected to the pad named "video" and the audio is connected to the pad named "audio":

```
movie=dvd.vob:s=v:0+#0x81 [video] [audio]
```

42. See Also

ffmpeg ffplay, ffprobe, ffserver, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

43. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file ‘`MAINTAINERS`’ in the source code tree.

This document was generated by *john* on *May 2, 2013* using *texi2html 1.82*.

ffmpeg Documentation

Table of Contents

1. Synopsis

ffmpeg [*global_options*] {[*input_file_options*] -i ‘`input_file`’} ... {[*output_file_options*] ‘`output_file`’} ...

2. Description

`ffmpeg` is a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality polyphase filter.

`ffmpeg` reads from an arbitrary number of input "files" (which can be regular files, pipes, network streams, grabbing devices, etc.), specified by the `-i` option, and writes to an arbitrary number of output "files", which are specified by a plain output filename. Anything found on the command line which cannot be interpreted as an option is considered to be an output filename.

Each input or output file can, in principle, contain any number of streams of different types (video/audio/subtitle/attachment/data). The allowed number and/or types of streams may be limited by the container format. Selecting which streams from which inputs will go into which output is either done automatically or with the `-map` option (see the Stream selection chapter).

To refer to input files in options, you must use their indices (0-based). E.g. the first input file is `0`, the second is `1`, etc. Similarly, streams within a file are referred to by their indices. E.g. `2:3` refers to the fourth stream in the third input file. Also see the Stream specifiers chapter.

As a general rule, options are applied to the next specified file. Therefore, order is important, and you can have the same option on the command line multiple times. Each occurrence is then applied to the next input or output file. Exceptions from this rule are the global options (e.g. verbosity level), which should be specified first.

Do not mix input and output files – first specify all input files, then all output files. Also do not mix options which belong to different files. All options apply ONLY to the next input or output file and are reset between files.

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

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi
```

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

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

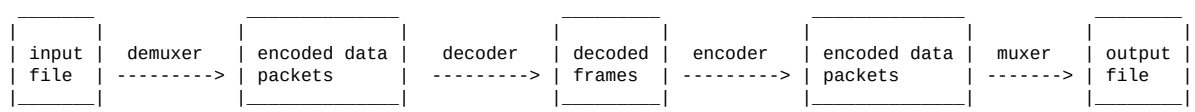
- To force the frame rate of the input file (valid for raw formats only) to 1 fps and the frame rate of the output file to 24 fps:

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

The format option may be needed for raw input files.

3. Detailed description

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



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

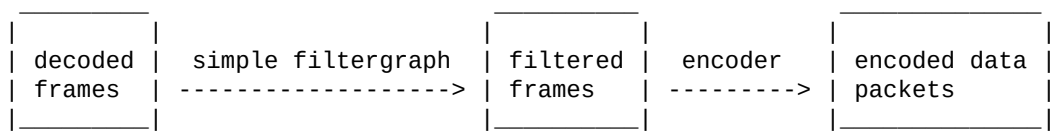
Encoded packets are then passed to the decoder (unless streamcopy is selected for the stream, see further for a description). The decoder produces uncompressed frames (raw video/PCM audio/...) which can be processed further by filtering (see next section). After filtering, the frames are passed to the encoder, which encodes them and outputs encoded packets. Finally those are passed to the muxer, which writes the encoded packets to the output file.

3.1 Filtering

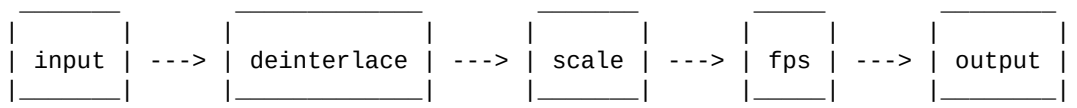
Before encoding, `ffmpeg` can process raw audio and video frames using filters from the libavfilter library. Several chained filters form a filter graph. `ffmpeg` distinguishes between two types of filtergraphs: simple and complex.

3.1.1 Simple filtergraphs

Simple filtergraphs are those that have exactly one input and output, both of the same type. In the above diagram they can be represented by simply inserting an additional step between decoding and encoding:



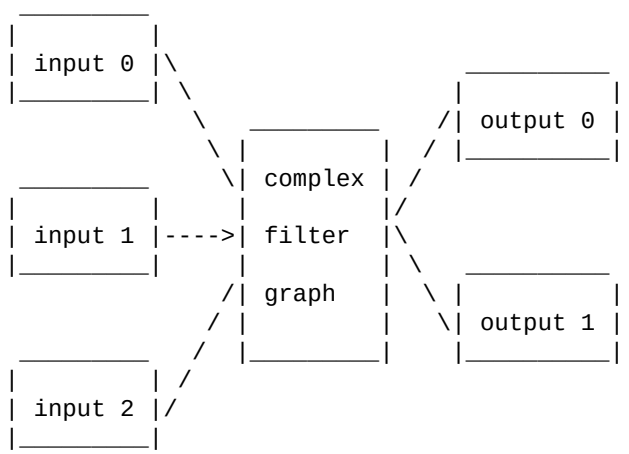
Simple filtergraphs are configured with the per-stream ‘`-filter`’ option (with ‘`-vf`’ and ‘`-af`’ aliases for video and audio respectively). A simple filtergraph for video can look for example like this:



Note that some filters change frame properties but not frame contents. E.g. the `fps` filter in the example above changes number of frames, but does not touch the frame contents. Another example is the `setpts` filter, which only sets timestamps and otherwise passes the frames unchanged.

3.1.2 Complex filtergraphs

Complex filtergraphs are those which cannot be described as simply a linear processing chain applied to one stream. This is the case, for example, when the graph has more than one input and/or output, or when output stream type is different from input. They can be represented with the following diagram:



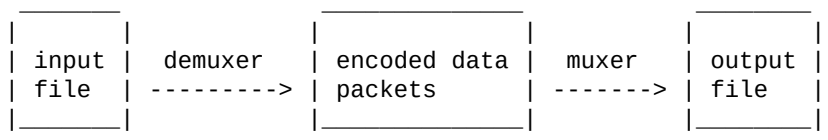
Complex filtergraphs are configured with the '`-filter_complex`' option. Note that this option is global, since a complex filtergraph, by its nature, cannot be unambiguously associated with a single stream or file.

The '`-lavfi`' option is equivalent to '`-filter_complex`'.

A trivial example of a complex filtergraph is the `overlay` filter, which has two video inputs and one video output, containing one video overlaid on top of the other. Its audio counterpart is the `amix` filter.

3.2 Stream copy

Stream copy is a mode selected by supplying the `copy` parameter to the '`-codec`' option. It makes `ffmpeg` omit the decoding and encoding step for the specified stream, so it does only demuxing and muxing. It is useful for changing the container format or modifying container-level metadata. The diagram above will, in this case, simplify to this:



Since there is no decoding or encoding, it is very fast and there is no quality loss. However, it might not work in some cases because of many factors. Applying filters is obviously also impossible, since filters work on uncompressed data.

4. Stream selection

By default, `ffmpeg` includes only one stream of each type (video, audio, subtitle) present in the input files and adds them to each output file. It picks the "best" of each based upon the following criteria: for video, it is the stream with the highest resolution, for audio, it is the stream with the most channels, for subtitles, it is the first subtitle stream. In the case where several streams of the same type rate equally, the stream with the lowest index is chosen.

You can disable some of those defaults by using the `-vn/-an/-sn` options. For full manual control, use the `-map` option, which disables the defaults just described.

5. Options

All the numerical options, if not specified otherwise, accept a string representing a number as input, which may be followed by one of the SI unit prefixes, for example: ’K’, ’M’, or ’G’.

If ’i’ is appended to the SI unit prefix, the complete prefix will be interpreted as a unit prefix for binary multiplies, which are based on powers of 1024 instead of powers of 1000. Appending ’B’ to the SI unit prefix multiplies the value by 8. This allows using, for example: ’KB’, ’MiB’, ’G’ and ’B’ as number suffixes.

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

5.1 Stream specifiers

Some options are applied per-stream, e.g. bitrate or codec. Stream specifiers are used to precisely specify which stream(s) a given option belongs 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` contains the `a:1` stream specifier, which matches the second audio stream. Therefore, it would select the ac3 codec for the second audio stream.

A stream specifier can match several streams, so that the option is 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 following: ’v’ for video, ’a’ for audio, ’s’ for subtitle, ’d’ for data, and ’t’ for attachments. If *stream_index* is given, then it matches stream number *stream_index* of this type. Otherwise, it matches all streams of this type.

‘p:*program_id*[:*stream_index*]’

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

‘#*stream_id*’

Matches the stream by a format-specific ID.

5.2 Generic options

These options are shared amongst the ff* 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.

‘`-codecs`’

Show all codecs known to libavcodec.

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

‘`-decoders`’

Show available decoders.

‘`-encoders`’

Show all available encoders.

‘`-bsfs`’

Show available bitstream filters.

‘`-protocols`’

Show available protocols.

‘`-filters`’

Show available libavfilter filters.

‘`-pix_fmts`’

Show available pixel formats.

‘`-sample_fmts`’

Show available sample formats.

‘`-layouts`’

Show channel names and standard channel layouts.

‘`-loglevel [repeat+]`*`loglevel`*` | -v [repeat+]`*`loglevel`*’

Set the logging level used by the library. Adding "repeat+" indicates that repeated log output should not be compressed to the first line and the "Last message repeated n times" line will be omitted. "repeat" can also be used alone. If "repeat" is used alone, and with no prior loglevel set, the default loglevel will be used. If multiple loglevel parameters are given, using ’repeat’ will not change the loglevel. *loglevel* is a number or a string containing one of the following values:

‘`quiet`’

Show nothing at all; be silent.

‘`panic`’

Only show fatal errors which could lead the process to crash, such as and assert failure. This is not currently used for anything.

‘`fatal`’

Only show fatal errors. These are errors after which the process absolutely cannot continue after.

‘`error`’

Show all errors, including ones which can be recovered from.

‘`warning`’

Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

‘info’

Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

‘verbose’

Same as info, except more verbose.

‘debug’

Show everything, including debugging information.

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

‘-report’

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

Setting the environment variable FFREPORT to any value has the same effect. If the value is a ’:’-separated key=value sequence, these options will affect the report; options values must be escaped if they contain special characters or the options delimiter ’:’ (see the “Quoting and escaping” section in the ffmpeg-utils manual). The following option is recognized:

‘file’

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

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

‘-cpuflags flags (*global*)’

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

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

Possible flags for this option are:

‘x86’
‘mmx’
‘mmxext’
‘sse’
‘sse2’
‘sse2slow’
‘sse3’
‘sse3slow’
‘ssse3’
‘atom’
‘sse4.1’
‘sse4.2’
‘avx’
‘xop’
‘fma4’
‘3dnow’
‘3dnowext’
‘cmov’
‘ARM’
‘armv5te’
‘armv6’
‘armv6t2’
‘vfp’
‘vfpv3’
‘neon’
‘PowerPC’
‘altivec’
‘Specific Processors’
‘pentium2’
‘pentium3’
‘pentium4’
‘k6’
‘k62’
‘athlon’
‘athlonxp’
‘k8’

‘-opencl_options options (*global*)’

Set OpenCL environment options. This option is only available when FFmpeg has been compiled with `--enable-opencl`.

options must be a list of *key=value* option pairs separated by ’:’. See the “OpenCL Options” section in the ffmpeg-utils manual for the list of supported options.

5.3 AVOptions

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

‘`generic`’

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

‘`private`’

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

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

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

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

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

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

5.4 Main options

‘`-f` *`fmt`* (*`input/output`*)’

> Force input or output file format. The format is normally auto detected for input files and guessed from the file extension for output files, so this option is not needed in most cases.

‘`-i` *`filename`* (*`input`*)’

> input file name

‘`-y` (*`global`*)’

> Overwrite output files without asking.

‘`-n` (*`global`*)’

> Do not overwrite output files, and exit immediately if a specified output file already exists.

‘-c[:*stream_specifier*] *codec* (*input/output,per-stream*)’
‘-codec[:*stream_specifier*] *codec* (*input/output,per-stream*)’

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. *codec* is the name of a decoder/encoder or a special value `copy` (output only) to indicate that the stream is not to be re-encoded.

For example

```
ffmpeg -i INPUT -map 0 -c:v libx264 -c:a copy OUTPUT
```

encodes all video streams with libx264 and copies all audio streams.

For each stream, the last matching `c` option is applied, so

```
ffmpeg -i INPUT -map 0 -c copy -c:v:1 libx264 -c:a:137 libvorbis OUTPUT
```

will copy all the streams except the second video, which will be encoded with libx264, and the 138th audio, which will be encoded with libvorbis.

‘-t *duration* (*output*)’

Stop writing the output after its duration reaches *duration*. *duration* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

‘-to *position* (*output*)’

Stop writing the output at *position*. *position* may be a number in seconds, or in `hh:mm:ss[.xxx]` form.

-to and -t are mutually exclusive and -t has priority.

‘-fs *limit_size* (*output*)’

Set the file size limit, expressed in bytes.

‘-ss *position* (*input/output*)’

When used as an input option (before `-i`), seeks in this input file to *position*. When used as an output option (before an output filename), decodes but discards input until the timestamps reach *position*. This is slower, but more accurate.

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

‘`-itsoffset` *`offset`* (*`input`*)’

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

‘`-timestamp` *`time`* (*`output`*)’

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

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

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

‘`-metadata[:metadata_specifier]` *`key=value`* (*`output,per-metadata`*)’

Set a metadata key/value pair.

An optional *metadata_specifier* may be given to set metadata on streams or chapters. See `-map_metadata` documentation for details.

This option overrides metadata set with `-map_metadata`. It is also possible to delete metadata by using an empty value.

For example, for setting the title in the output file:

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

To set the language of the first audio stream:

```
ffmpeg -i INPUT -metadata:s:a:1 language=eng OUTPUT
```

‘`-target` *`type`* (*`output`*)’

Specify target file type (`vcd`, `svcd`, `dvd`, `dv`, `dv50`). *type* may be prefixed with `pal-`, `ntsc-` or `film-` to use the corresponding standard. All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

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

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

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

‘-dframes *number* (*output*)’

Set the number of data frames to record. This is an alias for `-frames:d`.

‘-frames[:*stream_specifier*] *framecount* (*output,per-stream*)’

Stop writing to the stream after *framecount* frames.

‘-q[:*stream_specifier*] *q* (*output,per-stream*)’
‘-qscale[:*stream_specifier*] *q* (*output,per-stream*)’

Use fixed quality scale (VBR). The meaning of *q* is codec-dependent.

‘-filter[:*stream_specifier*] *filtergraph* (*output,per-stream*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

filtergraph is a description of the filtergraph to apply to the stream, and must have a single input and a single output of the same type of the stream. In the filtergraph, the input is associated to the label `in`, and the output to the label `out`. See the ffmpeg-filters manual for more information about the filtergraph syntax.

See the -filter_complex option if you want to create filtergraphs with multiple inputs and/or outputs.

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

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

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

Specify the preset for matching stream(s).

‘-stats (*global*)’

Print encoding progress/statistics. It is on by default, to explicitly disable it you need to specify `-nostats`.

‘-progress *url* (*global*)’

Send program-friendly progress information to *url*.

Progress information is written approximately every second and at the end of the encoding process. It is made of "*key=value*" lines. *key* consists of only alphanumeric characters. The last key of a sequence of progress information is always "progress".

‘`-stdin`’

Enable interaction on standard input. On by default unless standard input is used as an input. To explicitly disable interaction you need to specify `-nostdin`.

Disabling interaction on standard input is useful, for example, if ffmpeg is in the background process group. Roughly the same result can be achieved with `ffmpeg ... < /dev/null` but it requires a shell.

‘`-debug_ts (`*`global`*`)`’

Print timestamp information. It is off by default. This option is mostly useful for testing and debugging purposes, and the output format may change from one version to another, so it should not be employed by portable scripts.

See also the option `-fdebug ts`.

‘`-attach `*`filename`*` (`*`output`*`)`’

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

(assuming that the attachment stream will be third in the output file).

‘`-dump_attachment[:`*`stream_specifier`*`] `*`filename`*` (`*`input,per-stream`*`)`’

Extract the matching attachment stream into a file named *filename*. If *filename* is empty, then the value of the `filename` metadata tag will be used.

E.g. to extract the first attachment to a file named ’out.ttf’:

```
ffmpeg -dump_attachment:t:0 out.ttf -i INPUT
```

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

```
ffmpeg -dump_attachment:t "" -i INPUT
```

Technical note – attachments are implemented as codec extradata, so this option can actually be used to extract extradata from any stream, not just attachments.

5.5 Video Options

‘`-vframes` *`number`* `(`*`output`*`)`’

Set the number of video frames to record. This is an alias for `-frames:v`.

‘`-r[:`*`stream_specifier`*`]` *`fps`* `(`*`input/output,per-stream`*`)`’

Set frame rate (Hz value, fraction or abbreviation).

As an input option, ignore any timestamps stored in the file and instead generate timestamps assuming constant frame rate *fps*.

As an output option, duplicate or drop input frames to achieve constant output frame rate *fps*.

‘`-s[:`*`stream_specifier`*`]` *`size`* `(`*`input/output,per-stream`*`)`’

Set frame size.

As an input option, this is a shortcut for the ‘`video_size`’ private option, recognized by some demuxers for which the frame size is either not stored in the file or is configurable – e.g. raw video or video grabbers.

As an output option, this inserts the `scale` video filter to the *end* of the corresponding filtergraph. Please use the `scale` filter directly to insert it at the beginning or some other place.

The format is ‘`wxh`’ (default - same as source).

‘`-aspect[:`*`stream_specifier`*`]` *`aspect`* `(`*`output,per-stream`*`)`’

Set the video display aspect ratio specified by *aspect*.

aspect can be a floating point number string, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. For example "4:3", "16:9", "1.3333", and "1.7777" are valid argument values.

If used together with ‘`-vcodec copy`’, it will affect the aspect ratio stored at container level, but not the aspect ratio stored in encoded frames, if it exists.

‘`-vn (`*`output`*`)`’

Disable video recording.

‘`-vcodec` *`codec`* `(`*`output`*`)`’

Set the video codec. This is an alias for `-codec:v`.

‘-pass[:*stream_specifier*] *n* (*output,per-stream*)’

Select the pass number (1 or 2). It is used to do two-pass video encoding. The statistics of the video are recorded in the first pass into a log file (see also the option -passlogfile), and in the second pass that log file is used to generate the video at the exact requested bitrate. On pass 1, you may just deactivate audio and set output to null, examples for Windows and Unix:

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

‘-passlogfile[:*stream_specifier*] *prefix* (*output,per-stream*)’

Set two-pass log file name prefix to *prefix*, the default file name prefix is “ffmpeg2pass”. The complete file name will be ‘PREFIX-N.log’, where N is a number specific to the output stream

‘-vlang *code*’

Set the ISO 639 language code (3 letters) of the current video stream.

‘-vf *filtergraph* (*output*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for -filter:v, see the -filter option.

5.6 Advanced Video Options

‘-pix_fmt[:*stream_specifier*] *format* (*input/output,per-stream*)’

Set pixel format. Use -pix_fmts to show all the supported pixel formats. If the selected pixel format can not be selected, ffmpeg will print a warning and select the best pixel format supported by the encoder. If *pix_fmt* is prefixed by a +, ffmpeg will exit with an error if the requested pixel format can not be selected, and automatic conversions inside filtergraphs are disabled. If *pix_fmt* is a single +, ffmpeg selects the same pixel format as the input (or graph output) and automatic conversions are disabled.

‘-sws_flags *flags* (*input/output*)’

Set SwScaler flags.

‘-vdt *n*’

Discard threshold.

‘-rc_override[:*stream_specifier*] *override* (*output,per-stream*)’

Rate control override for specific intervals, formatted as "int,int,int" list separated with slashes. Two first values are the beginning and end frame numbers, last one is quantizer to use if positive, or quality factor if negative.

‘`-deinterlace`’

Deinterlace pictures. This option is deprecated since the deinterlacing is very low quality. Use the yadif filter with `-filter:v yadif`.

‘`-ilme`’

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

‘`-psnr`’

Calculate PSNR of compressed frames.

‘`-vstats`’

Dump video coding statistics to ‘`vstats_HHMMSS.log`’.

‘`-vstats_file` *`file`*’

Dump video coding statistics to *file*.

‘`-top[:`*`stream_specifier`*`]` *`n`* `(`*`output,per-stream`*`)`’

top=1/bottom=0/auto=-1 field first

‘`-dc` *`precision`*’

Intra_dc_precision.

‘`-vtag` *`fourcc/tag`* `(`*`output`*`)`’

Force video tag/fourcc. This is an alias for `-tag:v`.

‘`-qphist (`*`global`*`)`’

Show QP histogram

‘`-vbsf` *`bitstream_filter`*’

Deprecated see -bsf

‘`-force_key_frames[:`*`stream_specifier`*`]` *`time`*`[,`*`time`*`...]`
`(`*`output,per-stream`*`)`’

‘-force_key_frames[:*stream_specifier*] expr:*expr* (*output,per-stream*)’

Force key frames at the specified timestamps, more precisely at the first frames after each specified time.

If the argument is prefixed with `expr:`, the string *expr* is interpreted like an expression and is evaluated for each frame. A key frame is forced in case the evaluation is non-zero.

If one of the times is "`chapters`[*delta*]", it is expanded into the time of the beginning of all chapters in the file, shifted by *delta*, expressed as a time in seconds. This option can be useful to ensure that a seek point is present at a chapter mark or any other designated place in the output file.

For example, to insert a key frame at 5 minutes, plus key frames 0.1 second before the beginning of every chapter:

```
-force_key_frames 0:05:00,chapters-0.1
```

The expression in *expr* can contain the following constants:

‘`n`’

the number of current processed frame, starting from 0

‘`n_forced`’

the number of forced frames

‘`prev_forced_n`’

the number of the previous forced frame, it is `NAN` when no keyframe was forced yet

‘`prev_forced_t`’

the time of the previous forced frame, it is `NAN` when no keyframe was forced yet

‘`t`’

the time of the current processed frame

For example to force a key frame every 5 seconds, you can specify:

```
-force_key_frames expr:gte(t,n_forced*5)
```

To force a key frame 5 seconds after the time of the last forced one, starting from second 13:

```
-force_key_frames expr:if(isnan(prev_forced_t),gte(t,13),gte(t,prev_forced_t+5))
```

Note that forcing too many keyframes is very harmful for the lookahead algorithms of certain encoders: using fixed-GOP options or similar would be more efficient.

‘-copyinkf[:*stream_specifier*] (*output,per-stream*)’

When doing stream copy, copy also non-key frames found at the beginning.

5.7 Audio Options

‘-aframes *number* (*output*)’

Set the number of audio frames to record. This is an alias for -frames:a.

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

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

‘-aq *q* (*output*)’

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

‘-ac[:*stream_specifier*] *channels* (*input/output,per-stream*)’

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

‘-an (*output*)’

Disable audio recording.

‘-acodec *codec* (*input/output*)’

Set the audio codec. This is an alias for -codec:a.

‘-sample_fmt[:*stream_specifier*] *sample_fmt* (*output,per-stream*)’

Set the audio sample format. Use -sample_fmts to get a list of supported sample formats.

‘-af *filtergraph* (*output*)’

Create the filtergraph specified by *filtergraph* and use it to filter the stream.

This is an alias for -filter:a, see the -filter option.

5.8 Advanced Audio options:

‘-atag *fourcc/tag* (*output*)’

Force audio tag/fourcc. This is an alias for `-tag:a`.

‘-absf *bitstream_filter*’

Deprecated, see -bsf

‘-guess_layout_max *channels* (*input,per-stream*)’

If some input channel layout is not known, try to guess only if it corresponds to at most the specified number of channels. For example, 2 tells to `ffmpeg` to recognize 1 channel as mono and 2 channels as stereo but not 6 channels as 5.1. The default is to always try to guess. Use 0 to disable all guessing.

5.9 Subtitle options:

‘-slang *code*’

Set the ISO 639 language code (3 letters) of the current subtitle stream.

‘-scodec *codec* (*input/output*)’

Set the subtitle codec. This is an alias for `-codec:s`.

‘-sn (*output*)’

Disable subtitle recording.

‘-sbsf *bitstream_filter*’

Deprecated, see -bsf

5.10 Advanced Subtitle options:

‘-fix_sub_duration’

Fix subtitles durations. For each subtitle, wait for the next packet in the same stream and adjust the duration of the first to avoid overlap. This is necessary with some subtitles codecs, especially DVB subtitles, because the duration in the original packet is only a rough estimate and the end is actually marked by an empty subtitle frame. Failing to use this option when necessary can result in exaggerated durations or muxing failures due to non-monotonic timestamps.

Note that this option will delay the output of all data until the next subtitle packet is decoded: it may increase memory consumption and latency a lot.

‘-canvas_size *size*’

Set the size of the canvas used to render subtitles.

5.11 Advanced options

‘-map [-]*input_file_id*[:*stream_specifier*][,*sync_file_id*[:*stream_specifier*]] | *[linklabel]* (*output*)’

Designate one or more input streams as a source for the output file. Each input stream is identified by the input file index *input_file_id* and the input stream index *input_stream_id* within the input file. Both indices start at 0. If specified, *sync_file_id*:*stream_specifier* sets which input stream is used as a presentation sync reference.

The first `-map` option on the command line specifies the source for output stream 0, the second `-map` option specifies the source for output stream 1, etc.

A `-` character before the stream identifier creates a "negative" mapping. It disables matching streams from already created mappings.

An alternative *[linklabel]* form will map outputs from complex filter graphs (see the ‘`-filter_complex`’ option) to the output file. *linklabel* must correspond to a defined output link label in the graph.

For example, to map ALL streams from the first input file to output

```
ffmpeg -i INPUT -map 0 output
```

For example, if you have two audio streams in the first input file, these streams are identified by "0:0" and "0:1". You can use `-map` to select which streams to place in an output file. For example:

```
ffmpeg -i INPUT -map 0:1 out.wav
```

will map the input stream in ‘`INPUT`’ identified by "0:1" to the (single) output stream in ‘`out.wav`’.

For example, to select the stream with index 2 from input file ‘`a.mov`’ (specified by the identifier "0:2"), and stream with index 6 from input ‘`b.mov`’ (specified by the identifier "1:6"), and copy them to the output file ‘`out.mov`’:

```
ffmpeg -i a.mov -i b.mov -c copy -map 0:2 -map 1:6 out.mov
```

To select all video and the third audio stream from an input file:

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

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

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

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

‘`-map_channel` [*input_file_id*`.`*stream_specifier*`.`*channel_id*`|-1][:`*output_file_id*`.`*stream_specifier*`]’`

Map an audio channel from a given input to an output. If *output_file_id.stream_specifier* is not set, the audio channel will be mapped on all the audio streams.

Using "-1" instead of *input_file_id.stream_specifier.channel_id* will map a muted channel.

For example, assuming *INPUT* is a stereo audio file, you can switch the two audio channels with the following command:

```
ffmpeg -i INPUT -map_channel 0.0.1 -map_channel 0.0.0 OUTPUT
```

If you want to mute the first channel and keep the second:

```
ffmpeg -i INPUT -map_channel -1 -map_channel 0.0.1 OUTPUT
```

The order of the "-map_channel" option specifies the order of the channels in the output stream. The output channel layout is guessed from the number of channels mapped (mono if one "-map_channel", stereo if two, etc.). Using "-ac" in combination of "-map_channel" makes the channel gain levels to be updated if input and output channel layouts don’t match (for instance two "-map_channel" options and "-ac 6").

You can also extract each channel of an input to specific outputs; the following command extracts two channels of the *INPUT* audio stream (file 0, stream 0) to the respective *OUTPUT_CH0* and *OUTPUT_CH1* outputs:

```
ffmpeg -i INPUT -map_channel 0.0.0 OUTPUT_CH0 -map_channel 0.0.1 OUTPUT_CH1
```

The following example splits the channels of a stereo input into two separate streams, which are put into the same output file:

```
ffmpeg -i stereo.wav -map 0:0 -map 0:0 -map_channel 0.0.0:0.0 -map_channel 0.0.1:0.1 -y out.ogg
```

Note that currently each output stream can only contain channels from a single input stream; you can't for example use "-map_channel" to pick multiple input audio channels contained in different streams (from the same or different files) and merge them into a single output stream. It is therefore not currently possible, for example, to turn two separate mono streams into a single stereo stream. However splitting a stereo stream into two single channel mono streams is possible.

If you need this feature, a possible workaround is to use the *amerge* filter. For example, if you need to merge a media (here '`input.mkv`') with 2 mono audio streams into one single stereo channel audio stream (and keep the video stream), you can use the following command:

```
ffmpeg -i input.mkv -filter_complex "[0:1] [0:2] amerge" -c:a pcm_s16le -c:v copy output.mkv
```

'`-map_metadata[:`*`metadata_spec_out`*`] `*`infile`*`[:`*`metadata_spec_in`*`] (`*`output,per-metadata`*`)`'

Set metadata information of the next output file from *infile*. Note that those are file indices (zero-based), not filenames. Optional *metadata_spec_in/out* parameters specify, which metadata to copy. A metadata specifier can have the following forms:

'*`g`*'

global metadata, i.e. metadata that applies to the whole file

'*`s`*`[:`*`stream_spec`*`]`'

per-stream metadata. *stream_spec* is a stream specifier as described in the Stream specifiers chapter. In an input metadata specifier, the first matching stream is copied from. In an output metadata specifier, all matching streams are copied to.

'*`c`*`:`*`chapter_index`*'

per-chapter metadata. *chapter_index* is the zero-based chapter index.

'*`p`*`:`*`program_index`*'

per-program metadata. *program_index* is the zero-based program index.

If metadata specifier is omitted, it defaults to global.

By default, global metadata is copied from the first input file, per-stream and per-chapter metadata is copied along with streams/chapters. These default mappings are disabled by creating any mapping of the relevant type. A negative file index can be used to create a dummy mapping that just disables automatic copying.

For example to copy metadata from the first stream of the input file to global metadata of the output file:

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

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

```
ffmpeg -i in.mkv -map_metadata:s:a 0:g out.mkv
```

Note that simple `0` would work as well in this example, since global metadata is assumed by default.

‘`-map_chapters` *`input_file_index`* (*`output`*)’

Copy chapters from input file with index *input_file_index* to the next output file. If no chapter mapping is specified, then chapters are copied from the first input file with at least one chapter. Use a negative file index to disable any chapter copying.

‘`-benchmark` (*`global`*)’

Show benchmarking information at the end of an encode. Shows CPU time used and maximum memory consumption. Maximum memory consumption is not supported on all systems, it will usually display as 0 if not supported.

‘`-benchmark_all` (*`global`*)’

Show benchmarking information during the encode. Shows CPU time used in various steps (audio/video encode/decode).

‘`-timelimit` *`duration`* (*`global`*)’

Exit after ffmpeg has been running for *duration* seconds.

‘`-dump` (*`global`*)’

Dump each input packet to stderr.

‘`-hex` (*`global`*)’

When dumping packets, also dump the payload.

‘`-re` (*`input`*)’

Read input at native frame rate. Mainly used to simulate a grab device. By default `ffmpeg` attempts to read the input(s) as fast as possible. This option will slow down the reading of the input(s) to the native frame rate of the input(s). It is useful for real-time output (e.g. live streaming). If your input(s) is coming from some other live streaming source (through HTTP or UDP for example) the server might already be in real-time, thus the option will likely not be required. On the other hand, this is meaningful if your input(s) is a file you are trying to push in real-time.

‘`-loop_input`’

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing. This option is deprecated, use -loop 1.

‘`-loop_output` *`number_of_times`*’

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely). This option is deprecated, use -loop.

‘`-vsync` *`parameter`*’

Video sync method. For compatibility reasons old values can be specified as numbers. Newly added values will have to be specified as strings always.

‘`0, passthrough`’

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

‘`1, cfr`’

Frames will be duplicated and dropped to achieve exactly the requested constant frame rate.

‘`2, vfr`’

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

‘`drop`’

As passthrough but destroys all timestamps, making the muxer generate fresh timestamps based on frame-rate.

‘`-1, auto`’

Chooses between 1 and 2 depending on muxer capabilities. This is the default method.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option ‘`avoid_negative_ts`’ is enabled.

With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

‘`-async` *`samples_per_second`*’

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction.

Note that the timestamps may be further modified by the muxer, after this. For example, in the case that the format option ‘`avoid_negative_ts`’ is enabled.

This option has been deprecated. Use the `aresample` audio filter instead.

‘`-copyts`’

Do not process input timestamps, but keep their values without trying to sanitize them. In particular, do not remove the initial start time offset value.

Note that, depending on the ‘`vsync`’ option or on specific muxer processing (e.g. in case the format option ‘`avoid_negative_ts`’ is enabled) the output timestamps may mismatch with the input timestamps even when this option is selected.

‘`-copytb` *mode*’

Specify how to set the encoder timebase when stream copying. *mode* is an integer numeric value, and can assume one of the following values:

‘`1`’

Use the demuxer timebase.

The time base is copied to the output encoder from the corresponding input demuxer. This is sometimes required to avoid non monotonically increasing timestamps when copying video streams with variable frame rate.

‘`0`’

Use the decoder timebase.

The time base is copied to the output encoder from the corresponding input decoder.

‘`-1`’

Try to make the choice automatically, in order to generate a sane output.

Default value is -1.

‘`-shortest (`*output*`)`’

Finish encoding when the shortest input stream ends.

‘`-dts_delta_threshold`’

Timestamp discontinuity delta threshold.

‘`-muxdelay` *seconds* `(`*input*`)`’

Set the maximum demux-decode delay.

‘`-muxpreload` *`seconds`* (*`input`*)’

Set the initial demux-decode delay.

‘`-streamid` *`output-stream-index`*`:`*`new-value`* (*`output`*)’

Assign a new stream-id value to an output stream. This option should be specified prior to the output filename to which it applies. For the situation where multiple output files exist, a streamid may be reassigned to a different value.

For example, to set the stream 0 PID to 33 and the stream 1 PID to 36 for an output mpegts file:

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

‘`-bsf[:`*`stream_specifier`*`]` *`bitstream_filters`* (*`output,per-stream`*)’

Set bitstream filters for matching streams. *bitstream_filters* is a comma-separated list of bitstream filters. Use the `-bsfs` option to get the list of bitstream filters.

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

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

‘`-tag[:`*`stream_specifier`*`]` *`codec_tag`* (*`per-stream`*)’

Force a tag/fourcc for matching streams.

‘`-timecode` *`hh`*`:`*`mm`*`:`*`ss`*`SEP`*`ff`*’

Specify Timecode for writing. *SEP* is ’:’ for non drop timecode and ’;’ (or ’.’) for drop.

```
ffmpeg -i input.mpg -timecode 01:02:03.04 -r 30000/1001 -s ntsc output.mpg
```

‘`-filter_complex` *`filtergraph`* (*`global`*)’

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. For simple graphs – those with one input and one output of the same type – see the ‘`-filter`’ options. *filtergraph* is a description of the filtergraph, as described in the “Filtergraph syntax” section of the ffmpeg-filters manual.

Input link labels must refer to input streams using the `[file_index:stream_specifier]` syntax (i.e. the same as ‘`-map`’ uses). If *stream_specifier* matches multiple streams, the first one will be used. An unlabeled input will be connected to the first unused input stream of the matching type.

Output link labels are referred to with ‘`-map`’. Unlabeled outputs are added to the first output file.

Note that with this option it is possible to use only lavfi sources without normal input files.

For example, to overlay an image over video

```
ffmpeg -i video.mkv -i image.png -filter_complex '[0:v][1:v]overlay[out]' -map
'[out]' out.mkv
```

Here `[0:v]` refers to the first video stream in the first input file, which is linked to the first (main) input of the overlay filter. Similarly the first video stream in the second input is linked to the second (overlay) input of overlay.

Assuming there is only one video stream in each input file, we can omit input labels, so the above is equivalent to

```
ffmpeg -i video.mkv -i image.png -filter_complex 'overlay[out]' -map
'[out]' out.mkv
```

Furthermore we can omit the output label and the single output from the filter graph will be added to the output file automatically, so we can simply write

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

To generate 5 seconds of pure red video using lavfi `color` source:

```
ffmpeg -filter_complex 'color=c=red' -t 5 out.mkv
```

‘`-lavfi` *`filtergraph`* (*`global`*)’

Define a complex filtergraph, i.e. one with arbitrary number of inputs and/or outputs. Equivalent to ‘`-filter_complex`’.

‘`-filter_complex_script` *`filename`* (*`global`*)’

This option is similar to ‘`-filter_complex`’, the only difference is that its argument is the name of the file from which a complex filtergraph description is to be read.

As a special exception, you can use a bitmap subtitle stream as input: it will be converted into a video with the same size as the largest video in the file, or 720x576 if no video is present. Note that this is an experimental and temporary solution. It will be removed once libavfilter has proper support for subtitles.

For example, to hardcode subtitles on top of a DVB-T recording stored in MPEG-TS format, delaying the subtitles by 1 second:

```
ffmpeg -i input.ts -filter_complex \
  '[#0x2ef] setpts=PTS+1/TB [sub] ; [#0x2d0] [sub] overlay' \
  -sn -map '#0x2dc' output.mkv
```

(0x2d0, 0x2dc and 0x2ef are the MPEG-TS PIDs of respectively the video, audio and subtitles streams; 0:0, 0:3 and 0:7 would have worked too)

5.12 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which would be awkward to specify on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `vpre`, `apre`, `spre`, and `fpre` options. The `fpre` option takes the filename of the preset instead of a preset name as input and can be used for any kind of codec. For the `vpre`, `apre`, and `spre` options, the options specified in a preset file are applied to the currently selected codec of the same type as the preset option.

The argument passed to the `vpre`, `apre`, and `spre` preset options identifies the preset file to use according to the following rules:

First ffmpeg searches for a file named *arg*.ffpreset in the directories '`$FFMPEG_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the datadir defined at configuration time (usually '`PREFIX/share/ffmpeg`') or in a '`ffpresets`' folder along the executable on win32, in that order. For example, if the argument is `libvpx-1080p`, it will search for the file '`libvpx-1080p.ffpreset`'.

If no such file is found, then ffmpeg will search for a file named *codec_name-arg*.ffpreset in the above-mentioned directories, where *codec_name* is the name of the codec to which the preset file options will be applied. For example, if you select the video codec with `-vcodec libvpx` and use `-vpre 1080p`, then it will search for the file '`libvpx-1080p.ffpreset`'.

6. Tips

- For streaming at very low bitrate application, use a low frame rate and a small GOP size. This is especially true for RealVideo where the Linux player does not seem to be very fast, so it can miss frames. An example is:

  ```
  ffmpeg -g 3 -r 3 -t 10 -b:v 50k -s qcif -f rv10 /tmp/b.rm
  ```

- The parameter 'q' which is displayed while encoding is the current quantizer. The value 1 indicates that a very good quality could be achieved. The value 31 indicates the worst quality. If q=31 appears too often, it means that the encoder cannot compress enough to meet your bitrate. You must either increase the bitrate, decrease the frame rate or decrease the frame size.
- If your computer is not fast enough, you can speed up the compression at the expense of the

compression ratio. You can use '-me zero' to speed up motion estimation, and '-g 0' to disable motion estimation completely (you have only I-frames, which means it is about as good as JPEG compression).

- To have very low audio bitrates, reduce the sampling frequency (down to 22050 Hz for MPEG audio, 22050 or 11025 for AC-3).
- To have a constant quality (but a variable bitrate), use the option '-qscale n' when 'n' is between 1 (excellent quality) and 31 (worst quality).

7. Examples

7.1 Preset files

A preset file contains a sequence of *option=value* pairs, one for each line, specifying a sequence of options which can be specified also on the command line. Lines starting with the hash ('#') character are ignored and are used to provide comments. Empty lines are also ignored. Check the '`presets`' directory in the FFmpeg source tree for examples.

Preset files are specified with the `pre` option, this option takes a preset name as input. FFmpeg searches for a file named *preset_name*.avpreset in the directories '`$AVCONV_DATADIR`' (if set), and '`$HOME/.ffmpeg`', and in the data directory defined at configuration time (usually '`$PREFIX/share/ffmpeg`') in that order. For example, if the argument is `libx264-max`, it will search for the file '`libx264-max.avpreset`'.

7.2 Video and Audio grabbing

If you specify the input format and device then ffmpeg can grab video and audio directly.

```
ffmpeg -f oss -i /dev/dsp -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Or with an ALSA audio source (mono input, card id 1) instead of OSS:

```
ffmpeg -f alsa -ac 1 -i hw:1 -f video4linux2 -i /dev/video0 /tmp/out.mpg
```

Note that you must activate the right video source and channel before launching ffmpeg with any TV viewer such as xawtv by Gerd Knorr. You also have to set the audio recording levels correctly with a standard mixer.

7.3 X11 grabbing

Grab the X11 display with ffmpeg via

```
ffmpeg -f x11grab -s cif -r 25 -i :0.0 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable.

```
ffmpeg -f x11grab -s cif -r 25 -i :0.0+10,20 /tmp/out.mpg
```

0.0 is display.screen number of your X11 server, same as the DISPLAY environment variable. 10 is the x-offset and 20 the y-offset for the grabbing.

7.4 Video and Audio file format conversion

Any supported file format and protocol can serve as input to ffmpeg:

Examples:

- You can use YUV files as input:

  ```
  ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
  ```

  It will use the files:

  ```
  /tmp/test0.Y, /tmp/test0.U, /tmp/test0.V,
  /tmp/test1.Y, /tmp/test1.U, /tmp/test1.V, etc...
  ```

  The Y files use twice the resolution of the U and V files. They are raw files, without header. They can be generated by all decent video decoders. You must specify the size of the image with the '`-s`' option if ffmpeg cannot guess it.

- You can input from a raw YUV420P file:

  ```
  ffmpeg -i /tmp/test.yuv /tmp/out.avi
  ```

  test.yuv is a file containing raw YUV planar data. Each frame is composed of the Y plane followed by the U and V planes at half vertical and horizontal resolution.

- You can output to a raw YUV420P file:

  ```
  ffmpeg -i mydivx.avi hugefile.yuv
  ```

- You can set several input files and output files:

  ```
  ffmpeg -i /tmp/a.wav -s 640x480 -i /tmp/a.yuv /tmp/a.mpg
  ```

Converts the audio file a.wav and the raw YUV video file a.yuv to MPEG file a.mpg.

- You can also do audio and video conversions at the same time:

```
ffmpeg -i /tmp/a.wav -ar 22050 /tmp/a.mp2
```

Converts a.wav to MPEG audio at 22050 Hz sample rate.

- You can encode to several formats at the same time and define a mapping from input stream to output streams:

```
ffmpeg -i /tmp/a.wav -map 0:a -b:a 64k /tmp/a.mp2 -map 0:a -b:a 128k /tmp/b.mp2
```

Converts a.wav to a.mp2 at 64 kbits and to b.mp2 at 128 kbits. '-map file:index' specifies which input stream is used for each output stream, in the order of the definition of output streams.

- You can transcode decrypted VOBs:

```
ffmpeg -i snatch_1.vob -f avi -c:v mpeg4 -b:v 800k -g 300 -bf 2 -c:a libmp3lame -b:a 128k snatch.avi
```

This is a typical DVD ripping example; the input is a VOB file, the output an AVI file with MPEG-4 video and MP3 audio. Note that in this command we use B-frames so the MPEG-4 stream is DivX5 compatible, and GOP size is 300 which means one intra frame every 10 seconds for 29.97fps input video. Furthermore, the audio stream is MP3-encoded so you need to enable LAME support by passing `--enable-libmp3lame` to configure. The mapping is particularly useful for DVD transcoding to get the desired audio language.

NOTE: To see the supported input formats, use `ffmpeg -formats`.

- You can extract images from a video, or create a video from many images:

For extracting images from a video:

```
ffmpeg -i foo.avi -r 1 -s WxH -f image2 foo-%03d.jpeg
```

This will extract one video frame per second from the video and will output them in files named '`foo-001.jpeg`', '`foo-002.jpeg`', etc. Images will be rescaled to fit the new WxH values.

If you want to extract just a limited number of frames, you can use the above command in combination with the -vframes or -t option, or in combination with -ss to start extracting from a certain point in time.

For creating a video from many images:

```
ffmpeg -f image2 -i foo-%03d.jpeg -r 12 -s WxH foo.avi
```

The syntax `foo-%03d.jpeg` specifies to use a decimal number composed of three digits padded with zeroes to express the sequence number. It is the same syntax supported by the C printf function, but only formats accepting a normal integer are suitable.

When importing an image sequence, -i also supports expanding shell-like wildcard patterns (globbing) internally, by selecting the image2-specific `-pattern_type glob` option.

For example, for creating a video from filenames matching the glob pattern `foo-*.jpeg`:

```
ffmpeg -f image2 -pattern_type glob -i 'foo-*.jpeg' -r 12 -s WxH foo.avi
```

- You can put many streams of the same type in the output:

```
ffmpeg -i test1.avi -i test2.avi -map 0:3 -map 0:2 -map 0:1 -map 0:0 -c copy test12.nut
```

The resulting output file '`test12.avi`' will contain first four streams from the input file in reverse order.

- To force CBR video output:

```
ffmpeg -i myfile.avi -b 4000k -minrate 4000k -maxrate 4000k -bufsize 1835k out.m2v
```

- The four options lmin, lmax, mblmin and mblmax use 'lambda' units, but you may use the QP2LAMBDA constant to easily convert from 'q' units:

```
ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext
```

8. Syntax

This section documents the syntax and formats employed by the FFmpeg libraries and tools.

8.1 Quoting and escaping

FFmpeg adopts the following quoting and escaping mechanism, unless explicitly specified. The following rules are applied:

- `'` and \ are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.
- A special character is escaped by prefixing it with a '\'.
- All characters enclosed between '' are included literally in the parsed string. The quote character `'` itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

Note that you may need to add a second level of escaping when using the command line or a script, which depends on the syntax of the adopted shell language.

The function `av_get_token` defined in '`libavutil/avstring.h`' can be used to parse a token quoted or escaped according to the rules defined above.

The tool '`tools/ffescape`' in the FFmpeg source tree can be used to automatically quote or escape a string in a script.

8.1.1 Examples

- Escape the string `Crime d'Amour` containing the `'` special character:

```
Crime d\'Amour
```

- The string above contains a quote, so the `'` needs to be escaped when quoting it:

```
'Crime d'\''Amour'
```

- Include leading or trailing whitespaces using quoting:

```
'  this string starts and ends with whitespaces  '
```

- Escaping and quoting can be mixed together:

```
' The string '\''string\'' is a string '
```

- To include a literal `\` you can use either escaping or quoting:

```
'c:\foo' can be written as c:\\foo
```

8.2 Date

The accepted syntax is:

```
[(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...]]])|(HHMMSS[.m...]]]))[Z]
now
```

If the value is "now" it takes the current time.

Time is local time unless Z is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

8.3 Time duration

The accepted syntax is:

```
[-][HH:]MM:SS[.m...]
[-]S+[.m...]
```

HH expresses the number of hours, *MM* the number a of minutes and *SS* the number of seconds.

8.4 Video size

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation.

The following abbreviations are recognized:

‘`ntsc`’

720x480

‘`pal`’

720x576

‘`qntsc`’

352x240

‘`qpal`’

352x288

‘`sntsc`’

640x480

‘`spal`’

768x576

‘`film`’

352x240

‘`ntsc-film`’

352x240

‘sqcif’

128x96

‘qcif’

176x144

‘cif’

352x288

‘4cif’

704x576

‘16cif’

1408x1152

‘qqvga’

160x120

‘qvga’

320x240

‘vga’

640x480

‘svga’

800x600

‘xga’

1024x768

‘uxga’

1600x1200

‘qxga’

2048x1536

‘sxga’

1280x1024

‘qsxga’

2560x2048

‘hsxga’

5120x4096

‘wvga’

852x480

‘wxga’

1366x768

‘wsxga’

1600x1024

‘wuxga’

1920x1200

‘woxga’

2560x1600

‘wqsxga’

3200x2048

‘wquxga’

3840x2400

‘whsxga’

6400x4096

‘whuxga’

7680x4800

‘cga’

320x200

‘ega’

640x350

‘hd480’

852x480

‘hd720’

1280x720

‘hd1080’

1920x1080

‘2k’

2048x1080

‘2kflat’

1998x1080

‘2kscope’

2048x858

‘4k’

4096x2160

‘4kflat’

3996x2160

‘4kscope’

4096x1716

8.5 Video rate

Specify the frame rate of a video, expressed as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation.

The following abbreviations are recognized:

‘`ntsc`’

30000/1001

‘`pal`’

25/1

‘`qntsc`’

30000/1001

‘`qpal`’

25/1

‘`sntsc`’

30000/1001

‘`spal`’

25/1

‘`film`’

24/1

‘`ntsc-film`’

24000/1001

8.6 Ratio

A ratio can be expressed as an expression, or in the form *numerator*:*denominator*.

Note that a ratio with infinite (1/0) or negative value is considered valid, so you should check on the returned value if you want to exclude those values.

The undefined value can be expressed using the "0:0" string.

8.7 Color

It can be the name of a color (case insensitive match) or a [0x|#]RRGGBB[AA] sequence, possibly followed by "@" and a string representing the alpha component.

The alpha component may be a string composed by "0x" followed by an hexadecimal number or a decimal number between 0.0 and 1.0, which represents the opacity value (0x00/0.0 means completely transparent, 0xff/1.0 completely opaque). If the alpha component is not specified then 0xff is assumed.

The string "random" will result in a random color.

9. Expression Evaluation

When evaluating an arithmetic expression, FFmpeg uses an internal formula evaluator, implemented through the ‘`libavutil/eval.h`’ interface.

An expression may contain unary, binary operators, constants, and functions.

Two expressions *expr1* and *expr2* can be combined to form another expression "*expr1*;*expr2*". *expr1* and *expr2* are evaluated in turn, and the new expression evaluates to the value of *expr2*.

The following binary operators are available: `+`, `-`, `*`, `/`, `^`.

The following unary operators are available: `+`, `-`.

The following functions are available:

‘`abs(x)`’

Compute absolute value of *x*.

‘`acos(x)`’

Compute arccosine of *x*.

‘`asin(x)`’

Compute arcsine of *x*.

‘`atan(x)`’

Compute arctangent of *x*.

‘`between(x, min, max)`’

Return 1 if *x* is greater than or equal to *min* and lesser than or equal to *max*, 0 otherwise.

‘`bitand(x, y)`’
‘`bitor(x, y)`’

Compute bitwise and/or operation on *x* and *y*.

The results of the evaluation of *x* and *y* are converted to integers before executing the bitwise operation.

Note that both the conversion to integer and the conversion back to floating point can lose precision. Beware of unexpected results for large numbers (usually 2^53 and larger).

‘`ceil(expr)`’

Round the value of expression *expr* upwards to the nearest integer. For example, "ceil(1.5)" is "2.0".

‘`cos(x)`’

Compute cosine of *x*.

‘`cosh(x)`’

Compute hyperbolic cosine of *x*.

‘`eq(x, y)`’

Return 1 if *x* and *y* are equivalent, 0 otherwise.

‘`exp(x)`’

Compute exponential of *x* (with base `e`, the Euler’s number).

‘`floor(expr)`’

Round the value of expression *expr* downwards to the nearest integer. For example, "floor(-1.5)" is "-2.0".

‘`gauss(x)`’

Compute Gauss function of *x*, corresponding to `exp(-x*x/2) / sqrt(2*PI)`.

‘`gcd(x, y)`’

Return the greatest common divisor of *x* and *y*. If both *x* and *y* are 0 or either or both are less than zero then behavior is undefined.

‘`gt(x, y)`’

Return 1 if *x* is greater than *y*, 0 otherwise.

‘`gte(x, y)`’

Return 1 if *x* is greater than or equal to *y*, 0 otherwise.

‘`hypot(x, y)`’

This function is similar to the C function with the same name; it returns "sqrt(*x***x* + *y***y*)", the length of the hypotenuse of a right triangle with sides of length *x* and *y*, or the distance of the point (*x*, *y*) from the origin.

‘`if(x, y)`’

Evaluate *x*, and if the result is non-zero return the result of the evaluation of *y*, return 0 otherwise.

‘`if(x, y, z)`’

Evaluate *x*, and if the result is non-zero return the evaluation result of *y*, otherwise the evaluation result of *z*.

‘`ifnot(x, y)`’

Evaluate *x*, and if the result is zero return the result of the evaluation of *y*, return 0 otherwise.

‘`ifnot(x, y, z)`’

Evaluate *x*, and if the result is zero return the evaluation result of *y*, otherwise the evaluation result of *z*.

‘`isinf(x)`’

Return 1.0 if *x* is +/-INFINITY, 0.0 otherwise.

‘`isnan(x)`’

Return 1.0 if *x* is NAN, 0.0 otherwise.

‘`ld(var)`’

Allow to load the value of the internal variable with number *var*, which was previously stored with st(*var*, *expr*). The function returns the loaded value.

‘`log(x)`’

Compute natural logarithm of *x*.

‘`lt(x, y)`’

Return 1 if *x* is lesser than *y*, 0 otherwise.

‘`lte(x, y)`’

Return 1 if *x* is lesser than or equal to *y*, 0 otherwise.

‘`max(x, y)`’

Return the maximum between *x* and *y*.

‘`min(x, y)`’

Return the maximum between *x* and *y*.

‘`mod(x, y)`’

Compute the remainder of division of *x* by *y*.

‘`not(expr)`’

Return 1.0 if *expr* is zero, 0.0 otherwise.

‘`pow(x, y)`’

Compute the power of *x* elevated *y*, it is equivalent to "(*x*)^(*y*)".

‘`print(t)`’
‘`print(t, l)`’

Print the value of expression *t* with loglevel *l*. If *l* is not specified then a default log level is used. Returns the value of the expression printed.

Prints t with loglevel l

‘`random(x)`’

Return a pseudo random value between 0.0 and 1.0. *x* is the index of the internal variable which will be used to save the seed/state.

‘`root(expr, max)`’

Find an input value for which the function represented by *expr* with argument *ld(0)* is 0 in the interval 0..*max*.

The expression in *expr* must denote a continuous function or the result is undefined.

ld(0) is used to represent the function input value, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(0)`. When the expression evaluates to 0 then the corresponding input value will be returned.

‘`sin(x)`’

Compute sine of *x*.

‘`sinh(x)`’

Compute hyperbolic sine of *x*.

‘`sqrt(expr)`’

Compute the square root of *expr*. This is equivalent to "(*expr*)^.5".

‘`squish(x)`’

Compute expression `1/(1 + exp(4*x))`.

‘`st(var, expr)`’

Allow to store the value of the expression *expr* in an internal variable. *var* specifies the number of the variable where to store the value, and it is a value ranging from 0 to 9. The function returns the value stored in the internal variable. Note, Variables are currently not shared between expressions.

‘`tan(x)`’

Compute tangent of *x*.

‘`tanh(x)`’

Compute hyperbolic tangent of *x*.

‘`taylor(expr, x)`’
‘`taylor(expr, x, id)`’

Evaluate a Taylor series at *x*, given an expression representing the `ld(id)`-th derivative of a function at 0.

When the series does not converge the result is undefined.

ld(id) is used to represent the derivative order in *expr*, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(id)`. If *id* is not specified then 0 is assumed.

Note, when you have the derivatives at y instead of 0, `taylor(expr, x-y)` can be used.

‘`time(0)`’

Return the current (wallclock) time in seconds.

‘`trunc(expr)`’

Round the value of expression *expr* towards zero to the nearest integer. For example, "trunc(-1.5)" is "-1.0".

‘`while(cond, expr)`’

Evaluate expression *expr* while the expression *cond* is non-zero, and returns the value of the last *expr* evaluation, or NAN if *cond* was always false.

The following constants are available:

‘`PI`’

area of the unit disc, approximately 3.14

‘`E`’

exp(1) (Euler’s number), approximately 2.718

‘`PHI`’

golden ratio (1+sqrt(5))/2, approximately 1.618

Assuming that an expression is considered "true" if it has a non-zero value, note that:

`*` works like AND

`+` works like OR

For example the construct:

```
if (A AND B) then C
```

is equivalent to:

```
if(A*B, C)
```

In your C code, you can extend the list of unary and binary functions, and define recognized constants, so that they are available for your expressions.

The evaluator also recognizes the International System unit prefixes. 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.

The list of available International System prefixes follows, with indication of the corresponding powers of 10 and of 2.

‘y’

10^-24 / 2^-80

‘z’

10^-21 / 2^-70

‘a’

10^-18 / 2^-60

‘f’

10^-15 / 2^-50

‘p’

10^-12 / 2^-40

‘n’

10^-9 / 2^-30

‘u’

10^-6 / 2^-20

‘m’

10^-3 / 2^-10

‘c’

10^-2

‘d’

10^-1

‘h’

10^2

‘k’

10^3 / 2^10

‘K’

10^3 / 2^10

‘M’

10^6 / 2^20

‘G’

10^9 / 2^30

‘T’

10^12 / 2^40

‘P’

10^15 / 2^40

‘E’

10^18 / 2^50

‘Z’

10^21 / 2^60

‘Y’

10^24 / 2^70

10. OpenCL Options

When FFmpeg is configured with `--enable-opencl`, it is possible to set the options for the global OpenCL context.

The list of supported options follows:

‘`build_options`’

Set build options used to compile the registered kernels.

See reference "OpenCL Specification Version: 1.2 chapter 5.6.4".

‘platform_idx’

Select the index of the platform to run OpenCL code.

The specified index must be one of the indexes in the device list which can be obtained with `av_opencl_get_device_list()`.

‘device_idx’

Select the index of the device used to run OpenCL code.

The specifed index must be one of the indexes in the device list which can be obtained with `av_opencl_get_device_list()`.

11. Codec Options

libavcodec provides some generic global options, which can be set on all the encoders and decoders. In addition each codec may support so-called private options, which are specific for a given codec.

Sometimes, a global option may only affect a specific kind of codec, and may be unsensical or ignored by another, so you need to be aware of the meaning of the specified options. Also some options are meant only for decoding or encoding.

Options may be set by specifying *-option value* in the FFmpeg tools, or by setting the value explicitly in the `AVCodecContext` options or using the ‘`libavutil/opt.h`’ API for programmatic use.

The list of supported options follow:

‘b *integer* (*encoding,audio,video*)’

Set bitrate in bits/s. Default value is 200K.

‘ab *integer* (*encoding,audio*)’

Set audio bitrate (in bits/s). Default value is 128K.

‘bt *integer* (*encoding,video*)’

Set video bitrate tolerance (in bits/s). In 1-pass mode, bitrate tolerance specifies how far ratecontrol is willing to deviate from the target average bitrate value. This is not related to min/max bitrate. Lowering tolerance too much has an adverse effect on quality.

‘flags *flags* (*decoding/encoding,audio,video,subtitles*)’

Set generic flags.

Possible values:

‘mv4’

Use four motion vector by macroblock (mpeg4).

‘qpel’

Use 1/4 pel motion compensation.

‘loop’

Use loop filter.

‘qscale’

Use fixed qscale.

‘gmc’

Use gmc.

‘mv0’

Always try a mb with mv=<0,0>.

‘input_preserved’
‘pass1’

Use internal 2pass ratecontrol in first pass mode.

‘pass2’

Use internal 2pass ratecontrol in second pass mode.

‘gray’

Only decode/encode grayscale.

‘emu_edge’

Do not draw edges.

‘psnr’

Set error[?] variables during encoding.

‘truncated’

‘`naq`’

Normalize adaptive quantization.

‘`ildct`’

Use interlaced DCT.

‘`low_delay`’

Force low delay.

‘`global_header`’

Place global headers in extradata instead of every keyframe.

‘`bitexact`’

Use only bitexact stuff (except (I)DCT).

‘`aic`’

Apply H263 advanced intra coding / mpeg4 ac prediction.

‘`cbp`’

Deprecated, use mpegvideo private options instead.

‘`qprd`’

Deprecated, use mpegvideo private options instead.

‘`ilme`’

Apply interlaced motion estimation.

‘`cgop`’

Use closed gop.

‘`sub_id` *`integer`*’

Deprecated, currently unused.

‘`me_method` *`integer`* (*`encoding,video`*)’

Set motion estimation method.

Possible values:

‘zero’

zero motion estimation (fastest)

‘full’

full motion estimation (slowest)

‘epzs’

EPZS motion estimation (default)

‘esa’

esa motion estimation (alias for full)

‘tesa’

tesa motion estimation

‘dia’

dia motion estimation (alias for epzs)

‘log’

log motion estimation

‘phods’

phods motion estimation

‘x1’

X1 motion estimation

‘hex’

hex motion estimation

‘umh’

umh motion estimation

‘iter’

iter motion estimation

‘extradata_size *integer*’

Set extradata size.

‘time_base *rational number*’

Set codec time base.

It is the fundamental unit of time (in seconds) in terms of which frame timestamps are represented. For fixed-fps content, timebase should be `1 / frame_rate` and timestamp increments should be identically 1.

‘g *integer* (*encoding,video*)’

Set the group of picture size. Default value is 12.

‘ar *integer* (*decoding/encoding,audio*)’

Set audio sampling rate (in Hz).

‘ac *integer* (*decoding/encoding,audio*)’

Set number of audio channels.

‘cutoff *integer* (*encoding,audio*)’

Set cutoff bandwidth.

‘frame_size *integer* (*encoding,audio*)’

Set audio frame size.

Each submitted frame except the last must contain exactly frame_size samples per channel. May be 0 when the codec has CODEC_CAP_VARIABLE_FRAME_SIZE set, in that case the frame size is not restricted. It is set by some decoders to indicate constant frame size.

‘frame_number *integer*’

Set the frame number.

‘delay *integer*’
‘qcomp *float* (*encoding,video*)’

Set video quantizer scale compression (VBR). It is used as a constant in the ratecontrol equation. Recommended range for default rc_eq: 0.0-1.0.

‘qblur *float* (*encoding,video*)’

Set video quantizer scale blur (VBR).

‘qmin *integer* (*encoding,video*)’

Set min video quantizer scale (VBR). Must be included between -1 and 69, default value is 2.

‘qmax *integer* (*encoding,video*)’

Set max video quantizer scale (VBR). Must be included between -1 and 1024, default value is 31.

‘qdiff *integer* (*encoding,video*)’

Set max difference between the quantizer scale (VBR).

‘bf *integer* (*encoding,video*)’

Set max number of B frames.

‘b_qfactor *float* (*encoding,video*)’

Set qp factor between P and B frames.

‘rc_strategy *integer* (*encoding,video*)’

Set ratecontrol method.

‘b_strategy *integer* (*encoding,video*)’

Set strategy to choose between I/P/B-frames.

‘ps *integer* (*encoding,video*)’

Set RTP payload size in bytes.

‘mv_bits *integer*’
‘header_bits *integer*’
‘i_tex_bits *integer*’
‘p_tex_bits *integer*’
‘i_count *integer*’
‘p_count *integer*’
‘skip_count *integer*’
‘misc_bits *integer*’
‘frame_bits *integer*’
‘codec_tag *integer*’
‘bug *flags* (*decoding,video*)’

Workaround not auto detected encoder bugs.

Possible values:

‘autodetect’
‘old_msmpeg4’

some old lavc generated msmpeg4v3 files (no autodetection)

‘xvid_ilace’

Xvid interlacing bug (autodetected if fourcc==XVIX)

‘ump4’

(autodetected if fourcc==UMP4)

‘no_padding’

padding bug (autodetected)

‘amv’
‘ac_vlc’

illegal vlc bug (autodetected per fourcc)

‘qpel_chroma’
‘std_qpel’

old standard qpel (autodetected per fourcc/version)

‘qpel_chroma2’
‘direct_blocksize’

direct-qpel-blocksize bug (autodetected per fourcc/version)

‘edge’

edge padding bug (autodetected per fourcc/version)

‘hpel_chroma’
‘dc_clip’
‘ms’

Workaround various bugs in microsoft broken decoders.

‘trunc’

trancated frames

‘lelim *integer* (*encoding,video*)’

Set single coefficient elimination threshold for luminance (negative values also consider DC coefficient).

‘celim *integer* (*encoding,video*)’

Set single coefficient elimination threshold for chrominance (negative values also consider dc coefficient)

‘strict *integer* (*decoding/encoding,audio,video*)’

Specify how strictly to follow the standards.

Possible values:

‘very’

strictly conform to a older more strict version of the spec or reference software

‘strict’

strictly conform to all the things in the spec no matter what consequences

‘normal’
‘unofficial’

allow unofficial extensions

‘experimental’

allow non standardized experimental things

‘b_qoffset *float* (*encoding,video*)’

Set QP offset between P and B frames.

‘err_detect *flags* (*decoding,audio,video*)’

Set error detection flags.

Possible values:

‘crccheck’

verify embedded CRCs

‘bitstream’

detect bitstream specification deviations

‘buffer’

detect improper bitstream length

‘explode’

abort decoding on minor error detection

‘careful’

consider things that violate the spec and have not been seen in the wild as errors

‘compliant’

consider all spec non compliancies as errors

‘aggressive’

consider things that a sane encoder should not do as an error

‘has_b_frames *integer*’
‘block_align *integer*’
‘mpeg_quant *integer* (*encoding,video*)’

Use MPEG quantizers instead of H.263.

‘qsquish *float* (*encoding,video*)’

How to keep quantizer between qmin and qmax (0 = clip, 1 = use differentiable function).

‘rc_qmod_amp *float* (*encoding,video*)’

Set experimental quantizer modulation.

‘rc_qmod_freq *integer* (*encoding,video*)’

Set experimental quantizer modulation.

‘rc_override_count *integer*’
‘rc_eq *string* (*encoding,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` *`integer`* (*`encoding,audio,video`*)’

Set max bitrate tolerance (in bits/s). Requires bufsize to be set.

‘`minrate` *`integer`* (*`encoding,audio,video`*)’

Set min bitrate tolerance (in bits/s). Most useful in setting up a CBR encode. It is of little use elsewise.

‘`bufsize` *`integer`* (*`encoding,audio,video`*)’

Set ratecontrol buffer size (in bits).

‘`rc_buf_aggressivity` *`float`* (*`encoding,video`*)’

Currently useless.

‘`i_qfactor` *`float`* (*`encoding,video`*)’

Set QP factor between P and I frames.

‘`i_qoffset` *`float`* (*`encoding,video`*)’

Set QP offset between P and I frames.

‘`rc_init_cplx` *`float`* (*`encoding,video`*)’

Set initial complexity for 1-pass encoding.

‘`dct` *`integer`* (*`encoding,video`*)’

Set 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 *float* (*encoding*,*video*)’

Compress bright areas stronger than medium ones.

‘tcplx_mask *float* (*encoding*,*video*)’

Set temporal complexity masking.

‘scplx_mask *float* (*encoding*,*video*)’

Set spatial complexity masking.

‘p_mask *float* (*encoding*,*video*)’

Set inter masking.

‘dark_mask *float* (*encoding*,*video*)’

Compress dark areas stronger than medium ones.

‘idct *integer* (*decoding/encoding*,*video*)’

Select IDCT implementation.

Possible values:

‘auto’
‘int’
‘simple’
‘simplemmx’
‘libmpeg2mmx’
‘mmi’
‘arm’
‘altivec’
‘sh4’
‘simplearm’
‘simplearmv5te’
‘simplearmv6’
‘simpleneon’
‘simplealpha’
‘h264’
‘vp3’

‘ipp’
‘xvidmmx’
‘faani’

floating point AAN IDCT

‘slice_count *integer*’
‘ec *flags* (*decoding,video*)’

Set error concealment strategy.

Possible values:

‘guess_mvs’

iterative motion vector (MV) search (slow)

‘deblock’

use strong deblock filter for damaged MBs

‘bits_per_coded_sample *integer*’
‘pred *integer* (*encoding,video*)’

Set prediction method.

Possible values:

‘left’
‘plane’
‘median’
‘aspect *rational number* (*encoding,video*)’

Set sample aspect ratio.

‘debug *flags* (*decoding/encoding,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` *`integer`* (*`decoding`*,*`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` *`integer`* (*`encoding`*`,`*`video`*`)`’

Set full pel me compare function.

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’
‘subcmp *integer* (*encoding,video*)’

Set sub pel me compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’

‘chroma’

‘mbcmp *integer* (*encoding,video*)’

Set macroblock compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’

‘`ildctcmp` *`integer`* (*`encoding,video`*)’

Set interlaced dct compare function.

Possible values:

‘`sad`’

sum of absolute differences, fast (default)

‘`sse`’

sum of squared errors

‘`satd`’

sum of absolute Hadamard transformed differences

‘`dct`’

sum of absolute DCT transformed differences

‘`psnr`’

sum of squared quantization errors (avoid, low quality)

‘`bit`’

number of bits needed for the block

‘`rd`’

rate distortion optimal, slow

‘`zero`’

0

‘`vsad`’

sum of absolute vertical differences

‘`vsse`’

sum of squared vertical differences

‘`nsse`’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’

‘dia_size *integer* (*encoding,video*)’

Set diamond type & size for motion estimation.

‘last_pred *integer* (*encoding,video*)’

Set amount of motion predictors from the previous frame.

‘preme *integer* (*encoding,video*)’

Set pre motion estimation.

‘precmp *integer* (*encoding,video*)’

Set pre motion estimation compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’

‘chroma’

‘pre_dia_size *integer* (*encoding,video*)’

Set diamond type & size for motion estimation pre-pass.

‘subq *integer* (*encoding,video*)’

Set sub pel motion estimation quality.

‘dtg_active_format *integer*’

‘me_range *integer* (*encoding,video*)’

Set limit motion vectors range (1023 for DivX player).

‘ibias *integer* (*encoding,video*)’

Set intra quant bias.

‘pbias *integer* (*encoding,video*)’

Set inter quant bias.

‘color_table_id *integer*’
‘global_quality *integer* (*encoding,audio,video*)’
‘coder *integer* (*encoding,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 *integer* (*encoding,video*)’

Set context model.

‘slice_flags *integer*’
‘xvmc_acceleration *integer*’
‘mbd *integer* (*encoding,video*)’

Set macroblock decision algorithm (high quality mode).

Possible values:

‘simple’

use mbcmp (default)

‘bits’

use fewest bits

‘rd’

use best rate distortion

‘stream_codec_tag *integer*’
‘sc_threshold *integer* (*encoding,video*)’

Set scene change threshold.

‘lmin *integer* (*encoding,video*)’

Set min lagrange factor (VBR).

‘lmax *integer* (*encoding,video*)’

Set max lagrange factor (VBR).

‘nr *integer* (*encoding,video*)’

Set noise reduction.

‘rc_init_occupancy *integer* (*encoding,video*)’

Set number of bits which should be loaded into the rc buffer before decoding starts.

‘inter_threshold *integer* (*encoding,video*)’
‘flags2 *flags* (*decoding/encoding,audio,video*)’

Possible values:

‘fast’

allow non spec compliant speedup tricks

‘sgop’

Deprecated, use mpegvideo private options instead

‘noout’

skip bitstream encoding

‘local_header’

place global headers at every keyframe instead of in extradata

‘chunks’

Frame data might be split into multiple chunks

‘showall’

Show all frames before the first keyframe

‘skiprd’

Deprecated, use mpegvideo private options instead

‘error *integer* (*encoding,video*)’
‘qns *integer* (*encoding,video*)’

Deprecated, use mpegvideo private options instead.

‘threads *integer* (*decoding/encoding,video*)’

Possible values:

‘auto’

detect a good number of threads

‘me_threshold *integer* (*encoding,video*)’

Set motion estimation threshold.

‘mb_threshold *integer* (*encoding,video*)’

Set macroblock threshold.

‘dc *integer* (*encoding,video*)’

Set intra_dc_precision.

‘nssew *integer* (*encoding,video*)’

Set nsse weight.

‘skip_top *integer* (*decoding,video*)’

Set number of macroblock rows at the top which are skipped.

‘skip_bottom *integer* (*decoding,video*)’

Set number of macroblock rows at the bottom which are skipped.

‘profile *integer* (*encoding,audio,video*)’

Possible values:

‘unknown’
‘aac_main’
‘aac_low’
‘aac_ssr’
‘aac_ltp’
‘aac_he’
‘aac_he_v2’
‘aac_ld’
‘aac_eld’
‘dts’
‘dts_es’
‘dts_96_24’
‘dts_hd_hra’
‘dts_hd_ma’

‘level *integer* (*encoding,audio,video*)’

Possible values:

‘unknown’

‘lowres *integer* (*decoding,audio,video*)’

Decode at 1= 1/2, 2=1/4, 3=1/8 resolutions.

‘skip_threshold *integer* (*encoding,video*)’

Set frame skip threshold.

‘skip_factor *integer* (*encoding,video*)’

Set frame skip factor.

‘skip_exp *integer* (*encoding,video*)’

Set frame skip exponent.

‘skipcmp *integer* (*encoding,video*)’

Set frame skip compare function.

Possible values:

‘sad’

sum of absolute differences, fast (default)

‘sse’

sum of squared errors

‘satd’

sum of absolute Hadamard transformed differences

‘dct’

sum of absolute DCT transformed differences

‘psnr’

sum of squared quantization errors (avoid, low quality)

‘bit’

number of bits needed for the block

‘rd’

rate distortion optimal, slow

‘zero’

0

‘vsad’

sum of absolute vertical differences

‘vsse’

sum of squared vertical differences

‘nsse’

noise preserving sum of squared differences

‘w53’

5/3 wavelet, only used in snow

‘w97’

9/7 wavelet, only used in snow

‘dctmax’
‘chroma’
‘border_mask *float* (*encoding,video*)’

Increase the quantizer for macroblocks close to borders.

‘mblmin *integer* (*encoding,video*)’

Set min macroblock lagrange factor (VBR).

‘mblmax *integer* (*encoding,video*)’

Set max macroblock lagrange factor (VBR).

‘mepc *integer* (*encoding,video*)’

Set motion estimation bitrate penalty compensation (1.0 = 256).

‘skip_loop_filter *integer* (*decoding,video*)’
‘skip_idct *integer* (*decoding,video*)’
‘skip_frame *integer* (*decoding,video*)’

Make decoder discard processing depending on the frame type selected by the option value.

‘skip_loop_filter’ skips frame loop filtering, ‘skip_idct’ skips frame IDCT/dequantization, ‘skip_frame’ skips decoding.

Possible values:

‘none’

Discard no frame.

‘default’

Discard useless frames like 0-sized frames.

‘noref’

Discard all non-reference frames.

‘bidir’

Discard all bidirectional frames.

‘`nokey`’

Discard all frames excepts keyframes.

‘`all`’

Discard all frames.

Default value is ‘`default`’.

‘`bidir_refine` *`integer`* (*`encoding,video`*)’

Refine the two motion vectors used in bidirectional macroblocks.

‘`brd_scale` *`integer`* (*`encoding,video`*)’

Downscale frames for dynamic B-frame decision.

‘`keyint_min` *`integer`* (*`encoding,video`*)’

Set minimum interval between IDR-frames.

‘`refs` *`integer`* (*`encoding,video`*)’

Set reference frames to consider for motion compensation.

‘`chromaoffset` *`integer`* (*`encoding,video`*)’

Set chroma qp offset from luma.

‘`trellis` *`integer`* (*`encoding,audio,video`*)’

Set rate-distortion optimal quantization.

‘`sc_factor` *`integer`* (*`encoding,video`*)’

Set value multiplied by qscale for each frame and added to scene_change_score.

‘`mv0_threshold` *`integer`* (*`encoding,video`*)’
‘`b_sensitivity` *`integer`* (*`encoding,video`*)’

Adjust sensitivity of b_frame_strategy 1.

‘`compression_level` *`integer`* (*`encoding,audio,video`*)’
‘`min_prediction_order` *`integer`* (*`encoding,audio`*)’
‘`max_prediction_order` *`integer`* (*`encoding,audio`*)’

‘timecode_frame_start *integer* (*encoding,video*)’

Set GOP timecode frame start number, in non drop frame format.

‘request_channels *integer* (*decoding,audio*)’

Set desired number of audio channels.

‘bits_per_raw_sample *integer*’
‘channel_layout *integer* (*decoding/encoding,audio*)’

Possible values:

‘request_channel_layout *integer* (*decoding,audio*)’

Possible values:

‘rc_max_vbv_use *float* (*encoding,video*)’
‘rc_min_vbv_use *float* (*encoding,video*)’
‘ticks_per_frame *integer* (*decoding/encoding,audio,video*)’
‘color_primaries *integer* (*decoding/encoding,video*)’
‘color_trc *integer* (*decoding/encoding,video*)’
‘colorspace *integer* (*decoding/encoding,video*)’
‘color_range *integer* (*decoding/encoding,video*)’
‘chroma_sample_location *integer* (*decoding/encoding,video*)’
‘log_level_offset *integer*’

Set the log level offset.

‘slices *integer* (*encoding,video*)’

Number of slices, used in parallelized encoding.

‘thread_type *flags* (*decoding/encoding,video*)’

Select multithreading type.

Possible values:

‘slice’
‘frame’
‘audio_service_type *integer* (*encoding,audio*)’

Set 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 *sample_fmt* (*decoding,audio*)’

Set sample format audio decoders should prefer. Default value is none.

‘pkt_timebase *rational number*’
‘sub_charenc *encoding* (*decoding,subtitles*)’

Set the input subtitles character encoding.

12. Decoders

Decoders are configured elements in FFmpeg which allow the decoding of multimedia streams.

When you configure your FFmpeg build, all the supported native decoders are enabled by default. Decoders requiring an external library must be enabled manually via the corresponding `--enable-lib` option. You can list all available decoders using the configure option `--list-decoders`.

You can disable all the decoders with the configure option `--disable-decoders` and selectively enable / disable single decoders with the options `--enable-decoder=`*`DECODER`* / `--disable-decoder=`*`DECODER`*.

The option `-codecs` of the ff* tools will display the list of enabled decoders.

13. Video Decoders

A description of some of the currently available video decoders follows.

13.1 rawvideo

Raw video decoder.

This decoder decodes rawvideo streams.

13.1.1 Options

‘`top` *`top_field_first`*’

Specify the assumed field type of the input video.

‘`-1`’

the video is assumed to be progressive (default)

‘`0`’

bottom-field-first is assumed

‘`1`’

top-field-first is assumed

14. Audio Decoders

14.1 ffwavesynth

Internal wave synthetizer.

This decoder generates wave patterns according to predefined sequences. Its use is purely internal and the format of the data it accepts is not publicly documented.

15. Subtitles Decoders

15.1 dvdsub

This codec decodes the bitmap subtitles used in DVDs; the same subtitles can also be found in VobSub file pairs and in some Matroska files.

15.1.1 Options

‘`palette`’

Specify the global palette used by the bitmaps. When stored in VobSub, the palette is normally specified in the index file; in Matroska, the palette is stored in the codec extra-data in the same format as in VobSub. In DVDs, the palette is stored in the IFO file, and therefore not available when reading from dumped VOB files.

The format for this option is a string containing 16 24-bits hexadecimal numbers (without 0x prefix) separated by comas, for example `0d00ee, ee450d, 101010, eaeaea, 0ce60b, ec14ed, ebff0b, 0d617a, 7b7b7b, d1d1d1, 7b2a0e, 0d950c, 0f007b, cf0dec, cfa80c, 7c127b`.

16. Encoders

Encoders are configured elements in FFmpeg which allow the encoding of multimedia streams.

When you configure your FFmpeg build, all the supported native encoders are enabled by default. Encoders requiring an external library must be enabled manually via the corresponding `--enable-lib` option. You can list all available encoders using the configure option `--list-encoders`.

You can disable all the encoders with the configure option `--disable-encoders` and selectively enable / disable single encoders with the options `--enable-encoder=`*`ENCODER`* / `--disable-encoder=`*`ENCODER`*.

The option `-codecs` of the ff* tools will display the list of enabled encoders.

17. Audio Encoders

A description of some of the currently available audio encoders follows.

17.1 ac3 and ac3_fixed

AC-3 audio encoders.

These encoders implement part of ATSC A/52:2010 and ETSI TS 102 366, as well as the undocumented RealAudio 3 (a.k.a. dnet).

The *ac3* encoder uses floating-point math, while the *ac3_fixed* encoder only uses fixed-point integer math. This does not mean that one is always faster, just that one or the other may be better suited to a particular system. The floating-point encoder will generally produce better quality audio for a given bitrate. The *ac3_fixed* encoder is not the default codec for any of the output formats, so it must be specified explicitly using the option `-acodec ac3_fixed` in order to use it.

17.1.1 AC-3 Metadata

The AC-3 metadata options are used to set parameters that describe the audio, but in most cases do not affect the audio encoding itself. Some of the options do directly affect or influence the decoding and playback of the resulting bitstream, while others are just for informational purposes. A few of the options will add bits to the output stream that could otherwise be used for audio data, and will thus affect the quality of the output. Those will be indicated accordingly with a note in the option list below.

These parameters are described in detail in several publicly-available documents.

- A/52:2010 - Digital Audio Compression (AC-3) (E-AC-3) Standard
- A/54 - Guide to the Use of the ATSC Digital Television Standard
- Dolby Metadata Guide
- Dolby Digital Professional Encoding Guidelines

17.1.1.1 Metadata Control Options

‘`-per_frame_metadata` *`boolean`*’

Allow Per-Frame Metadata. Specifies if the encoder should check for changing metadata for each frame.

‘`0`’

The metadata values set at initialization will be used for every frame in the stream. (default)

‘`1`’

Metadata values can be changed before encoding each frame.

17.1.1.2 Downmix Levels

‘`-center_mixlev` *`level`*’

Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo. This field will only be written to the bitstream if a center channel is present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6dB gain

‘`-surround_mixlev` *`level`*’

Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo. This field will only be written to the bitstream if one or more surround channels are present. The value is specified as a scale factor. There are 3 valid values:

‘`0.707`’

Apply -3dB gain

‘`0.500`’

Apply -6dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

17.1.1.3 Audio Production Information

Audio Production Information is optional information describing the mixing environment. Either none or both of the fields are written to the bitstream.

‘`-mixing_level` *`number`*’

Mixing Level. Specifies peak sound pressure level (SPL) in the production environment when the mix was mastered. Valid values are 80 to 111, or -1 for unknown or not indicated. The default value is -1, but that value cannot be used if the Audio Production Information is written to the bitstream. Therefore, if the `room_type` option is not the default value, the `mixing_level` option must not be -1.

‘`-room_type` *`type`*’

Room Type. Describes the equalization used during the final mixing session at the studio or on the dubbing stage. A large room is a dubbing stage with the industry standard X-curve equalization; a small room has flat equalization. This field will not be written to the bitstream if both the `mixing_level` option and the `room_type` option have the default values.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`large`’

Large Room

‘`2`’
‘`small`’

Small Room

17.1.1.4 Other Metadata Options

‘`-copyright` *`boolean`*’

Copyright Indicator. Specifies whether a copyright exists for this audio.

‘`0`’
‘`off`’

No Copyright Exists (default)

‘`1`’
‘`on`’

Copyright Exists

‘`-dialnorm` *`value`*’

Dialogue Normalization. Indicates how far the average dialogue level of the program is below digital 100% full scale (0 dBFS). This parameter determines a level shift during audio reproduction that sets the average volume of the dialogue to a preset level. The goal is to match volume level between program sources. A value of -31dB will result in no volume level change, relative to the source volume, during audio reproduction. Valid values are whole numbers in the range -31 to -1, with -31 being the default.

‘`-dsur_mode` *`mode`*’

Dolby Surround Mode. Specifies whether the stereo signal uses Dolby Surround (Pro Logic). This field will only be written to the bitstream if the audio stream is stereo. Using this option does **NOT** mean the encoder will actually apply Dolby Surround processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`off`’

Not Dolby Surround Encoded

‘`2`’
‘`on`’

Dolby Surround Encoded

‘`-original` *`boolean`*’

Original Bit Stream Indicator. Specifies whether this audio is from the original source and not a copy.

‘`0`’
‘`off`’

Not Original Source

‘`1`’
‘`on`’

Original Source (default)

17.1.2 Extended Bitstream Information

The extended bitstream options are part of the Alternate Bit Stream Syntax as specified in Annex D of the A/52:2010 standard. It is grouped into 2 parts. If any one parameter in a group is specified, all values in that group will be written to the bitstream. Default values are used for those that are written but have not been specified. If the mixing levels are written, the decoder will use these values instead of the ones specified in the `center_mixlev` and `surround_mixlev` options if it supports the Alternate Bit Stream Syntax.

17.1.2.1 Extended Bitstream Information - Part 1

‘`-dmix_mode` *`mode`*’

Preferred Stereo Downmix Mode. Allows the user to select either Lt/Rt (Dolby Surround) or Lo/Ro (normal stereo) as the preferred stereo downmix mode.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`ltrt`’

Lt/Rt Downmix Preferred

‘`2`’
‘`loro`’

Lo/Ro Downmix Preferred

‘`-ltrt_cmixlev` *`level`*’

Lt/Rt Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo in Lt/Rt mode.

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-ltrt_surmixlev` *`level`*’

Lt/Rt Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo in Lt/Rt mode.

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

‘`-loro_cmixlev` *`level`*’

Lo/Ro Center Mix Level. The amount of gain the decoder should apply to the center channel when downmixing to stereo in Lo/Ro mode.

‘`1.414`’

Apply +3dB gain

‘`1.189`’

Apply +1.5dB gain

‘`1.000`’

Apply 0dB gain

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain (default)

‘`0.500`’

Apply -6.0dB gain

‘`0.000`’

Silence Center Channel

‘`-loro_surmixlev` *`level`*’

Lo/Ro Surround Mix Level. The amount of gain the decoder should apply to the surround channel(s) when downmixing to stereo in Lo/Ro mode.

‘`0.841`’

Apply -1.5dB gain

‘`0.707`’

Apply -3.0dB gain

‘`0.595`’

Apply -4.5dB gain

‘`0.500`’

Apply -6.0dB gain (default)

‘`0.000`’

Silence Surround Channel(s)

17.1.2.2 Extended Bitstream Information - Part 2

‘`-dsurex_mode` *`mode`*’

Dolby Surround EX Mode. Indicates whether the stream uses Dolby Surround EX (7.1 matrixed to 5.1). Using this option does **NOT** mean the encoder will actually apply Dolby Surround EX processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`on`’

Dolby Surround EX Off

‘`2`’
‘`off`’

Dolby Surround EX On

‘`-dheadphone_mode` *`mode`*’

Dolby Headphone Mode. Indicates whether the stream uses Dolby Headphone encoding (multi-channel matrixed to 2.0 for use with headphones). Using this option does **NOT** mean the encoder will actually apply Dolby Headphone processing.

‘`0`’
‘`notindicated`’

Not Indicated (default)

‘`1`’
‘`on`’

Dolby Headphone Off

‘`2`’
‘`off`’

Dolby Headphone On

‘`-ad_conv_type` *`type`*’

A/D Converter Type. Indicates whether the audio has passed through HDCD A/D conversion.

‘0’
‘standard’

Standard A/D Converter (default)

‘1’
‘hdcd’

HDCD A/D Converter

17.1.3 Other AC-3 Encoding Options

‘-stereo_rematrixing *boolean*’

Stereo Rematrixing. Enables/Disables use of rematrixing for stereo input. This is an optional AC-3 feature that increases quality by selectively encoding the left/right channels as mid/side. This option is enabled by default, and it is highly recommended that it be left as enabled except for testing purposes.

17.1.4 Floating-Point-Only AC-3 Encoding Options

These options are only valid for the floating-point encoder and do not exist for the fixed-point encoder due to the corresponding features not being implemented in fixed-point.

‘-channel_coupling *boolean*’

Enables/Disables use of channel coupling, which is an optional AC-3 feature that increases quality by combining high frequency information from multiple channels into a single channel. The per-channel high frequency information is sent with less accuracy in both the frequency and time domains. This allows more bits to be used for lower frequencies while preserving enough information to reconstruct the high frequencies. This option is enabled by default for the floating-point encoder and should generally be left as enabled except for testing purposes or to increase encoding speed.

‘-1’
‘auto’

Selected by Encoder (default)

‘0’
‘off’

Disable Channel Coupling

‘1’
‘on’

Enable Channel Coupling

‘-cpl_start_band *number*’

Coupling Start Band. Sets the channel coupling start band, from 1 to 15. If a value higher than the bandwidth is used, it will be reduced to 1 less than the coupling end band. If *auto* is used, the start band will be determined by the encoder based on the bit rate, sample rate, and channel layout. This option has no effect if channel coupling is disabled.

‘-1’
‘auto’

Selected by Encoder (default)

18. Video Encoders

A description of some of the currently available video encoders follows.

18.1 libtheora

Theora format supported through libtheora.

Requires the presence of the libtheora headers and library during configuration. You need to explicitly configure the build with `--enable-libtheora`.

18.1.1 Options

The following global options are mapped to internal libtheora options which affect the quality and the bitrate of the encoded stream.

‘b’

Set the video bitrate, only works if the `qscale` flag in ‘`flags`’ is not enabled.

‘`flags`’

Used to enable constant quality mode encoding through the ‘`qscale`’ flag, and to enable the `pass1` and `pass2` modes.

‘g’

Set the GOP size.

‘`global_quality`’

Set the global quality in lambda units, only works if the `qscale` flag in ‘`flags`’ is enabled. The value is clipped in the [0 - 10*`FF_QP2LAMBDA`] range, and then multiplied for 6.3 to get a value in the native libtheora range [0-63]. A higher value corresponds to a higher quality.

For example, to set maximum constant quality encoding with `ffmpeg`:

```
ffmpeg -i INPUT -flags:v qscale -global_quality:v "10*QP2LAMBDA" -codec:v libtheora OUTPUT.ogg
```

18.2 libvpx

VP8 format supported through libvpx.

Requires the presence of the libvpx headers and library during configuration. You need to explicitly configure the build with `--enable-libvpx`.

18.2.1 Options

Mapping from FFmpeg to libvpx options with conversion notes in parentheses.

‘`threads`’

g_threads

‘`profile`’

g_profile

‘`vb`’

rc_target_bitrate

‘`g`’

kf_max_dist

‘`keyint_min`’

kf_min_dist

‘`qmin`’

rc_min_quantizer

‘`qmax`’

rc_max_quantizer

‘bufsize, vb’

rc_buf_sz (bufsize * 1000 / vb)

rc_buf_optimal_sz (bufsize * 1000 / vb * 5 / 6)

‘rc_init_occupancy, vb’

rc_buf_initial_sz (rc_init_occupancy * 1000 / vb)

‘rc_buffer_aggressivity’

rc_undershoot_pct

‘skip_threshold’

rc_dropframe_thresh

‘qcomp’

rc_2pass_vbr_bias_pct

‘maxrate, vb’

rc_2pass_vbr_maxsection_pct (maxrate * 100 / vb)

‘minrate, vb’

rc_2pass_vbr_minsection_pct (minrate * 100 / vb)

‘minrate, maxrate, vb’

VPX_CBR (minrate == maxrate == vb)

‘crf’

VPX_CQ, VP8E_SET_CQ_LEVEL

‘quality’

‘*best*’

VPX_DL_BEST_QUALITY

‘*good*’

VPX_DL_GOOD_QUALITY

‘*realtime*’

VPX_DL_REALTIME

‘speed’

VP8E_SET_CPUUSED

‘nr’

VP8E_SET_NOISE_SENSITIVITY

‘mb_threshold’

VP8E_SET_STATIC_THRESHOLD

‘slices’

VP8E_SET_TOKEN_PARTITIONS

‘max-intra-rate’

VP8E_SET_MAX_INTRA_BITRATE_PCT

‘force_key_frames’

VPX_EFLAG_FORCE_KF

‘Alternate reference frame related’

‘vp8flags altref’

VP8E_SET_ENABLEAUTOALTREF

‘*arnr_max_frames*’

VP8E_SET_ARNR_MAXFRAMES

‘*arnr_type*’

VP8E_SET_ARNR_TYPE

‘*arnr_strength*’

VP8E_SET_ARNR_STRENGTH

‘*rc_lookahead*’

g_lag_in_frames

‘vp8flags error_resilient’

g_error_resilient

For more information about libvpx see: http://www.webmproject.org/

18.3 libx264

x264 H.264/MPEG-4 AVC encoder wrapper

Requires the presence of the libx264 headers and library during configuration. You need to explicitly configure the build with `--enable-libx264`.

x264 supports an impressive number of features, including 8x8 and 4x4 adaptive spatial transform, adaptive B-frame placement, CAVLC/CABAC entropy coding, interlacing (MBAFF), lossless mode, psy optimizations for detail retention (adaptive quantization, psy-RD, psy-trellis).

The FFmpeg wrapper provides a mapping for most of them using global options that match those of the encoders and provides private options for the unique encoder options. Additionally an expert override is provided to directly pass a list of key=value tuples as accepted by x264_param_parse.

18.3.1 Option Mapping

The following options are supported by the x264 wrapper, the x264-equivalent options follow the FFmpeg ones.

b	bitrate FFmpeg `b` option is expressed in bits/s, x264 `bitrate` in kilobits/s.
bf	bframes Maximum number of B-frames.
g	keyint Maximum GOP size.
qmin	qpmin
qmax	qpmax
qdiff	qpstep
qblur	qblur
qcomp	qcomp
refs	ref
sc_threshold	scenecut
trellis	trellis
nr	nr Noise reduction.
me_range	merange
me_method	me
subq	subme
b_strategy	b-adapt
keyint_min	keyint-min
coder	cabac Set coder to `ac` to use CABAC.
cmp	chroma-me Set to `chroma` to use chroma motion estimation.
threads	threads
thread_type	sliced_threads Set to `slice` to use sliced threading instead of frame threading.
flags -cgop	open-gop Set `-cgop` to use recovery points to close GOPs.
rc_init_occupancy	vbv-init Initial buffer occupancy.

18.3.2 Private Options

‘`-preset` *`string`*’

Set the encoding preset (cf. x264 –fullhelp).

‘`-tune` *`string`*’

Tune the encoding params (cf. x264 –fullhelp).

‘`-profile` *`string`*’

Set profile restrictions (cf. x264 –fullhelp).

‘`-fastfirstpass` *`integer`*’

Use fast settings when encoding first pass.

‘`-crf` *`float`*’

Select the quality for constant quality mode.

‘`-crf_max` *`float`*’

In CRF mode, prevents VBV from lowering quality beyond this point.

‘`-qp` *`integer`*’

Constant quantization parameter rate control method.

‘`-aq-mode` *`integer`*’

AQ method

Possible values:

‘`none`’
‘`variance`’

Variance AQ (complexity mask).

‘`autovariance`’

Auto-variance AQ (experimental).

‘`-aq-strength` *`float`*’

AQ strength, reduces blocking and blurring in flat and textured areas.

‘`-psy` *`integer`*’

Use psychovisual optimizations.

‘`-psy-rd` *`string`*’

Strength of psychovisual optimization, in <psy-rd>:<psy-trellis> format.

‘`-rc-lookahead` *`integer`*’

Number of frames to look ahead for frametype and ratecontrol.

‘`-weightb` *`integer`*’

Weighted prediction for B-frames.

‘`-weightp` *`integer`*’

Weighted prediction analysis method.

Possible values:

‘`none`’
‘`simple`’
‘`smart`’

‘`-ssim` *`integer`*’

Calculate and print SSIM stats.

‘`-intra-refresh` *`integer`*’

Use Periodic Intra Refresh instead of IDR frames.

‘`-b-bias` *`integer`*’

Influences how often B-frames are used.

‘`-b-pyramid` *`integer`*’

Keep some B-frames as references.

Possible values:

‘`none`’
‘`strict`’

Strictly hierarchical pyramid.

‘`normal`’

Non-strict (not Blu-ray compatible).

‘`-mixed-refs` *`integer`*’

One reference per partition, as opposed to one reference per macroblock.

‘`-8x8dct` *`integer`*’

High profile 8x8 transform.

‘`-fast-pskip` *`integer`*’
‘`-aud` *`integer`*’

Use access unit delimiters.

‘`-mbtree` *`integer`*’

Use macroblock tree ratecontrol.

‘`-deblock` *`string`*’

Loop filter parameters, in <alpha:beta> form.

‘`-cplxblur` *`float`*’

Reduce fluctuations in QP (before curve compression).

‘`-partitions` *`string`*’

A comma-separated list of partitions to consider, possible values: p8x8, p4x4, b8x8, i8x8, i4x4, none, all.

‘`-direct-pred` *`integer`*’

Direct MV prediction mode

Possible values:

‘`none`’
‘`spatial`’
‘`temporal`’
‘`auto`’

‘`-slice-max-size` *`integer`*’

Limit the size of each slice in bytes.

‘`-stats` *`string`*’

Filename for 2 pass stats.

‘`-nal-hrd` *`integer`*’

Signal HRD information (requires vbv-bufsize; cbr not allowed in .mp4).

Possible values:

‘none’
‘vbr’
‘cbr’

‘x264opts *options*’

Allow to set any x264 option, see `x264 --fullhelp` for a list.

options is a list of *key=value* couples separated by ":". In *filter* and *psy-rd* options that use ":" as a separator themselves, use "," instead. They accept it as well since long ago but this is kept undocumented for some reason.

For example to specify libx264 encoding options with `ffmpeg`:

```
ffmpeg -i foo.mpg -vcodec libx264 -x264opts keyint=123:min-keyint=20 -an out.mkv
```

For more information about libx264 and the supported options see:
http://www.videolan.org/developers/x264.html

‘-x264-params *string*’

Override the x264 configuration using a :-separated list of key=value parameters.

```
-x264-params level=30:bframes=0:weightp=0:cabac=0:ref=1:vbv-maxrate=768:vbv-bufsize=2000:analyse=all:me=umh:no-fast-pskip=1:subq=6:8x8dct=0:trellis=0
```

Encoding avpresets for common usages are provided so they can be used with the general presets system (e.g. passing the `-pre` option).

18.4 ProRes

Apple ProRes encoder.

FFmpeg contains 2 ProRes encoders, the prores-aw and prores-ks encoder. The used encoder can be choosen with the `-vcodec` option.

18.4.1 Private Options for prores-ks

‘profile *integer*’

Select the ProRes profile to encode

‘proxy’

‘lt’
‘standard’
‘hq’

‘quant_mat *integer*’

Select quantization matrix.

‘auto’
‘default’
‘proxy’
‘lt’
‘standard’
‘hq’

If set to *auto*, the matrix matching the profile will be picked. If not set, the matrix providing the highest quality, *default*, will be picked.

‘bits_per_mb *integer*’

How many bits to allot for coding one macroblock. Different profiles use between 200 and 2400 bits per macroblock, the maximum is 8000.

‘mbs_per_slice *integer*’

Number of macroblocks in each slice (1-8); the default value (8) should be good in almost all situations.

‘vendor *string*’

Override the 4-byte vendor ID. A custom vendor ID like *apl0* would claim the stream was produced by the Apple encoder.

18.4.2 Speed considerations

In the default mode of operation the encoder has to honor frame constraints (i.e. not produc frames with size bigger than requested) while still making output picture as good as possible. A frame containing a lot of small details is harder to compress and the encoder would spend more time searching for appropriate quantizers for each slice.

Setting a higher ‘bits_per_mb’ limit will improve the speed.

For the fastest encoding speed set the ‘qscale’ parameter (4 is the recommended value) and do not set a size constraint.

19. Bitstream Filters

When you configure your FFmpeg build, all the supported bitstream filters are enabled by default. You can list all available ones using the configure option `--list-bsfs`.

You can disable all the bitstream filters using the configure option `--disable-bsfs`, and selectively enable any bitstream filter using the option `--enable-bsf=BSF`, or you can disable a particular bitstream filter using the option `--disable-bsf=BSF`.

The option `-bsfs` of the ff* tools will display the list of all the supported bitstream filters included in your build.

Below is a description of the currently available bitstream filters.

19.1 aac_adtstoasc

19.2 chomp

19.3 dump_extradata

19.4 h264_mp4toannexb

Convert an H.264 bitstream from length prefixed mode to start code prefixed mode (as defined in the Annex B of the ITU-T H.264 specification).

This is required by some streaming formats, typically the MPEG-2 transport stream format ("mpegts").

For example to remux an MP4 file containing an H.264 stream to mpegts format with `ffmpeg`, you can use the command:

```
ffmpeg -i INPUT.mp4 -codec copy -bsf:v h264_mp4toannexb OUTPUT.ts
```

19.5 imx_dump_header

19.6 mjpeg2jpeg

Convert MJPEG/AVI1 packets to full JPEG/JFIF packets.

MJPEG is a video codec wherein each video frame is essentially a JPEG image. The individual frames can be extracted without loss, e.g. by

```
ffmpeg -i ../some_mjpeg.avi -c:v copy frames_%d.jpg
```

Unfortunately, these chunks are incomplete JPEG images, because they lack the DHT segment required for decoding. Quoting from http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml:

Avery Lee, writing in the rec.video.desktop newsgroup in 2001, commented that "MJPEG, or at least the MJPEG in AVIs having the MJPG fourcc, is restricted JPEG with a fixed – and *omitted* – Huffman table. The JPEG must be YCbCr colorspace, it must be 4:2:2, and it must use basic Huffman encoding, not arithmetic or progressive. . . . You can indeed extract the MJPEG frames and decode them with a regular JPEG decoder, but you have to prepend the DHT segment to them, or else the decoder won’t have any idea how to decompress the data. The exact table necessary is given in the OpenDML spec."

This bitstream filter patches the header of frames extracted from an MJPEG stream (carrying the AVI1 header ID and lacking a DHT segment) to produce fully qualified JPEG images.

```
ffmpeg -i mjpeg-movie.avi -c:v copy -bsf:v mjpeg2jpeg frame_%d.jpg
exiftran -i -9 frame*.jpg
ffmpeg -i frame_%d.jpg -c:v copy rotated.avi
```

19.7 mjpega_dump_header

19.8 movsub

19.9 mp3_header_compress

19.10 mp3_header_decompress

19.11 noise

19.12 remove_extradata

20. Format Options

The libavformat library provides some generic global options, which can be set on all the muxers and demuxers. In addition each muxer or demuxer may support so-called private options, which are specific for that component.

Options may be set by specifying *-option value* in the FFmpeg tools, or by setting the value explicitly in the `AVFormatContext` options or using the ‘`libavutil/opt.h`’ API for programmatic use.

The list of supported options follows:

‘`avioflags` *`flags`* (*`input/output`*)’

Possible values:

‘`direct`’

Reduce buffering.

‘`probesize` *`integer`* (*`input`*)’

Set probing size in bytes, i.e. the size of the data to analyze to get stream information. A higher value will allow to detect more information in case it is dispersed into the stream, but will increase latency. Must be an integer not lesser than 32. It is 5000000 by default.

‘`packetsize` *`integer`* (*`output`*)’

Set packet size.

‘`fflags` *`flags`* (*`input/output`*)’

Set format flags.

Possible values:

‘`ignidx`’

Ignore index.

‘`genpts`’

Generate PTS.

‘`nofillin`’

Do not fill in missing values that can be exactly calculated.

‘`noparse`’

Disable AVParsers, this needs `+nofillin` too.

‘`igndts`’

Ignore DTS.

‘`discardcorrupt`’

Discard corrupted frames.

‘`sortdts`’

Try to interleave output packets by DTS.

‘keepside’

Do not merge side data.

‘latm’

Enable RTP MP4A-LATM payload.

‘nobuffer’

Reduce the latency introduced by optional buffering

‘analyzeduration *integer* (*input*)’

Specify how many microseconds are analyzed to probe the input. A higher value will allow to detect more accurate information, but will increase latency. It defaults to 5,000,000 microseconds = 5 seconds.

‘cryptokey *hexadecimal string* (*input*)’

Set decryption key.

‘indexmem *integer* (*input*)’

Set max memory used for timestamp index (per stream).

‘rtbufsize *integer* (*input*)’

Set max memory used for buffering real-time frames.

‘fdebug *flags* (*input/output*)’

Print specific debug info.

Possible values:

‘ts’

‘max_delay *integer* (*input/output*)’

Set maximum muxing or demuxing delay in microseconds.

‘fpsprobesize *integer* (*input*)’

Set number of frames used to probe fps.

‘audio_preload *integer* (*output*)’

Set microseconds by which audio packets should be interleaved earlier.

‘`chunk_duration` *`integer`* (*`output`*)’

Set microseconds for each chunk.

‘`chunk_size` *`integer`* (*`output`*)’

Set size in bytes for each chunk.

‘`err_detect, f_err_detect` *`flags`* (*`input`*)’

Set error detection flags. `f_err_detect` is deprecated and should be used only via the `ffmpeg` tool.

Possible values:

‘`crccheck`’

Verify embedded CRCs.

‘`bitstream`’

Detect bitstream specification deviations.

‘`buffer`’

Detect improper bitstream length.

‘`explode`’

Abort decoding on minor error detection.

‘`careful`’

Consider things that violate the spec and have not been seen in the wild as errors.

‘`compliant`’

Consider all spec non compliancies as errors.

‘`aggressive`’

Consider things that a sane encoder should not do as an error.

‘`use_wallclock_as_timestamps` *`integer`* (*`input`*)’

Use wallclock as timestamps.

‘avoid_negative_ts *integer* (*output*)’

Shift timestamps to make them positive. A value of 1 enables shifting, a value of 0 disables it, the default value of -1 enables shifting when required by the target format.

When shifting is enabled, all output timestamps are shifted by the same amount. Audio, video, and subtitles desynching and relative timestamp differences are preserved compared to how they would have been without shifting.

Also note that this affects only leading negative timestamps, and not non-monotonic negative timestamps.

‘flush_packets *integer* (*output*)’

Flush the underlying I/O stream after each packet. Default 1 enables it, and has the effect of reducing the latency; 0 disables it and may slightly increase performance in some cases.

21. Demuxers

Demuxers are configured elements in FFmpeg which allow to read the multimedia streams from a particular type of file.

When you configure your FFmpeg build, all the supported demuxers are enabled by default. You can list all available ones using the configure option `--list-demuxers`.

You can disable all the demuxers using the configure option `--disable-demuxers`, and selectively enable a single demuxer with the option `--enable-demuxer=`*`DEMUXER`*, or disable it with the option `--disable-demuxer=`*`DEMUXER`*.

The option `-formats` of the ff* tools will display the list of enabled demuxers.

The description of some of the currently available demuxers follows.

21.1 applehttp

Apple HTTP Live Streaming demuxer.

This demuxer presents all AVStreams from all variant streams. The id field is set to the bitrate variant index number. By setting the discard flags on AVStreams (by pressing ’a’ or ’v’ in ffplay), the caller can decide which variant streams to actually receive. The total bitrate of the variant that the stream belongs to is available in a metadata key named "variant_bitrate".

21.2 concat

Virtual concatenation script demuxer.

This demuxer reads a list of files and other directives from a text file and demuxes them one after the other, as if all their packet had been muxed together.

The timestamps in the files are adjusted so that the first file starts at 0 and each next file starts where the previous one finishes. Note that it is done globally and may cause gaps if all streams do not have exactly the same length.

All files must have the same streams (same codecs, same time base, etc.).

The duration of each file is used to adjust the timestamps of the next file: if the duration is incorrect (because it was computed using the bit-rate or because the file is truncated, for example), it can cause artifacts. The `duration` directive can be used to override the duration stored in each file.

21.2.1 Syntax

The script is a text file in extended-ASCII, with one directive per line. Empty lines, leading spaces and lines starting with '#' are ignored. The following directive is recognized:

'`file` *`path`*'

Path to a file to read; special characters and spaces must be escaped with backslash or single quotes.

All subsequent directives apply to that file.

'`ffconcat version 1.0`'

Identify the script type and version. It also sets the '`safe`' option to 1 if it was to its default -1.

To make FFmpeg recognize the format automatically, this directive must appears exactly as is (no extra space or byte-order-mark) on the very first line of the script.

'`duration` *`dur`*'

Duration of the file. This information can be specified from the file; specifying it here may be more efficient or help if the information from the file is not available or accurate.

If the duration is set for all files, then it is possible to seek in the whole concatenated video.

21.2.2 Options

This demuxer accepts the following option:

‘safe’

If set to 1, reject unsafe file paths. A file path is considered safe if it does not contain a protocol specification and is relative and all components only contain characters from the portable character set (letters, digits, period, underscore and hyphen) and have no period at the beginning of a component.

If set to 0, any file name is accepted.

The default is -1, it is equivalent to 1 if the format was automatically probed and 0 otherwise.

21.3 libquvi

Play media from Internet services using the quvi project.

The demuxer accepts a ‘format’ option to request a specific quality. It is by default set to *best*.

See http://quvi.sourceforge.net/ for more information.

FFmpeg needs to be built with --enable-libquvi for this demuxer to be enabled.

21.4 image2

Image file demuxer.

This demuxer reads from a list of image files specified by a pattern. The syntax and meaning of the pattern is specified by the option *pattern_type*.

The pattern may contain a suffix which is used to automatically determine the format of the images contained in the files.

The size, the pixel format, and the format of each image must be the same for all the files in the sequence.

This demuxer accepts the following options:

‘framerate’

Set the frame rate for the video stream. It defaults to 25.

‘loop’

If set to 1, loop over the input. Default value is 0.

‘pattern_type’

Select the pattern type used to interpret the provided filename.

pattern_type accepts one of the following values.

‘`sequence`’

Select a sequence pattern type, used to specify a sequence of files indexed by sequential numbers.

A sequence pattern may contain the string "%d" or "%0*N*d", which specifies the position of the characters representing a sequential number in each filename matched by the pattern. If the form "%d0*N*d" is used, the string representing the number in each filename is 0-padded and *N* is the total number of 0-padded digits representing the number. The literal character ’%’ can be specified in the pattern with the string "%%".

If the sequence pattern contains "%d" or "%0*N*d", the first filename of the file list specified by the pattern must contain a number inclusively contained between *start_number* and *start_number*+*start_number_range*-1, and all the following numbers must be sequential.

For example the pattern "img-%03d.bmp" will match a sequence of filenames of the form ‘`img-001.bmp`’, ‘`img-002.bmp`’, ..., ‘`img-010.bmp`’, etc.; the pattern "i%%m%%g-%d.jpg" will match a sequence of filenames of the form ‘`i%m%g-1.jpg`’, ‘`i%m%g-2.jpg`’, ..., ‘`i%m%g-10.jpg`’, etc.

Note that the pattern must not necessarily contain "%d" or "%0*N*d", for example to convert a single image file ‘`img.jpeg`’ you can employ the command:

```
ffmpeg -i img.jpeg img.png
```

‘`glob`’

Select a glob wildcard pattern type.

The pattern is interpreted like a `glob()` pattern. This is only selectable if libavformat was compiled with globbing support.

‘`glob_sequence` *`(deprecated, will be removed)`*’

Select a mixed glob wildcard/sequence pattern.

If your version of libavformat was compiled with globbing support, and the provided pattern contains at least one glob meta character among `%*?[]{}` that is preceded by an unescaped "%", the pattern is interpreted like a `glob()` pattern, otherwise it is interpreted like a sequence pattern.

All glob special characters `%*?[]{}` must be prefixed with "%". To escape a literal "%" you shall use "%%".

For example the pattern `foo-%*.jpeg` will match all the filenames prefixed by "foo-" and terminating with ".jpeg", and `foo-%?%?%?.jpeg` will match all the filenames prefixed with "foo-", followed by a sequence of three characters, and terminating with ".jpeg".

This pattern type is deprecated in favor of *glob* and *sequence*.

Default value is *glob_sequence*.

‘`pixel_format`’

Set the pixel format of the images to read. If not specified the pixel format is guessed from the first image file in the sequence.

‘`start_number`’

Set the index of the file matched by the image file pattern to start to read from. Default value is 0.

‘`start_number_range`’

Set the index interval range to check when looking for the first image file in the sequence, starting from *start_number*. Default value is 5.

‘`video_size`’

Set the video size of the images to read. If not specified the video size is guessed from the first image file in the sequence.

21.4.1 Examples

- Use `ffmpeg` for creating a video from the images in the file sequence ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., assuming an input frame rate of 10 frames per second:

```
ffmpeg -i 'img-%03d.jpeg' -r 10 out.mkv
```

- As above, but start by reading from a file with index 100 in the sequence:

```
ffmpeg -start_number 100 -i 'img-%03d.jpeg' -r 10 out.mkv
```

- Read images matching the "*.png" glob pattern , that is all the files terminating with the ".png" suffix:

```
ffmpeg -pattern_type glob -i "*.png" -r 10 out.mkv
```

21.5 rawvideo

Raw video demuxer.

This demuxer allows to read raw video data. Since there is no header specifying the assumed video parameters, the user must specify them in order to be able to decode the data correctly.

This demuxer accepts the following options:

‘framerate’

Set input video frame rate. Default value is 25.

‘pixel_format’

Set the input video pixel format. Default value is yuv420p.

‘video_size’

Set the input video size. This value must be specified explicitly.

For example to read a rawvideo file ‘input.raw’ with ffplay, assuming a pixel format of rgb24, a video size of 320x240, and a frame rate of 10 images per second, use the command:

```
ffplay -f rawvideo -pixel_format rgb24 -video_size 320x240 -framerate 10 input.raw
```

21.6 sbg

SBaGen script demuxer.

This demuxer reads the script language used by SBaGen http://uazu.net/sbagen/ to generate binaural beats sessions. A SBG script looks like that:

```
-SE
a: 300-2.5/3 440+4.5/0
b: 300-2.5/0 440+4.5/3
off: -
NOW      == a
+0:07:00 == b
+0:14:00 == a
+0:21:00 == b
+0:30:00    off
```

A SBG script can mix absolute and relative timestamps. If the script uses either only absolute timestamps (including the script start time) or only relative ones, then its layout is fixed, and the conversion is straightforward. On the other hand, if the script mixes both kind of timestamps, then the *NOW* reference for relative timestamps will be taken from the current time of day at the time the script is read, and the script layout will be frozen according to that reference. That means that if the script is directly played, the

actual times will match the absolute timestamps up to the sound controller's clock accuracy, but if the user somehow pauses the playback or seeks, all times will be shifted accordingly.

21.7 tedcaptions

JSON captions used for TED Talks.

TED does not provide links to the captions, but they can be guessed from the page. The file '`tools/bookmarklets.html`' from the FFmpeg source tree contains a bookmarklet to expose them.

This demuxer accepts the following option:

'`start_time`'

Set the start time of the TED talk, in milliseconds. The default is 15000 (15s). It is used to sync the captions with the downloadable videos, because they include a 15s intro.

Example: convert the captions to a format most players understand:

```
ffmpeg -i http://www.ted.com/talks/subtitles/id/1/lang/en talk1-en.srt
```

22. Muxers

Muxers are configured elements in FFmpeg which allow writing multimedia streams to a particular type of file.

When you configure your FFmpeg build, all the supported muxers are enabled by default. You can list all available muxers using the configure option `--list-muxers`.

You can disable all the muxers with the configure option `--disable-muxers` and selectively enable / disable single muxers with the options `--enable-muxer=`*`MUXER`* / `--disable-muxer=`*`MUXER`*.

The option `-formats` of the ff* tools will display the list of enabled muxers.

A description of some of the currently available muxers follows.

22.1 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`’:

```
ffmpeg -i INPUT -f crc out.crc
```

You can print the CRC to stdout with the command:

```
ffmpeg -i INPUT -f crc -
```

You can select the output format of each frame with `ffmpeg` 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:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f crc -
```

See also the framecrc muxer.

22.2 framecrc

Per-packet CRC (Cyclic Redundancy Check) testing format.

This muxer computes and prints the Adler-32 CRC for each audio and video packet. 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 packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, 0xCRC
```

CRC is a hexadecimal number 0-padded to 8 digits containing the CRC of the packet.

For example to compute the CRC of the audio and video frames in ‘`INPUT`’, converted to raw audio and video packets, and store it in the file ‘`out.crc`’:

```
ffmpeg -i INPUT -f framecrc out.crc
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framecrc -
```

With `ffmpeg`, you can select the output format to which the audio and video frames are encoded before computing the CRC for each packet by specifying the audio and video codec. 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:

```
ffmpeg -i INPUT -c:a pcm_u8 -c:v mpeg2video -f framecrc -
```

See also the crc muxer.

22.3 framemd5

Per-packet MD5 testing format.

This muxer computes and prints the MD5 hash for each audio and video packet. By default audio frames are converted to signed 16-bit raw audio and video frames to raw video before computing the hash.

The output of the muxer consists of a line for each audio and video packet of the form:

```
stream_index, packet_dts, packet_pts, packet_duration, packet_size, MD5
```

MD5 is a hexadecimal number representing the computed MD5 hash for the packet.

For example to compute the MD5 of the audio and video frames in '`INPUT`', converted to raw audio and video packets, and store it in the file '`out.md5`':

```
ffmpeg -i INPUT -f framemd5 out.md5
```

To print the information to stdout, use the command:

```
ffmpeg -i INPUT -f framemd5 -
```

See also the md5 muxer.

22.4 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.

```
ffmpeg -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*.

22.5 ico

ICO file muxer.

Microsoft’s icon file format (ICO) has some strict limitations that should be noted:

- Size cannot exceed 256 pixels in any dimension
- Only BMP and PNG images can be stored
- If a BMP image is used, it must be one of the following pixel formats:

```
BMP Bit Depth      FFmpeg Pixel Format
1bit               pal8
4bit               pal8
8bit               pal8
16bit              rgb555le
24bit              bgr24
32bit              bgra
```

- If a BMP image is used, it must use the BITMAPINFOHEADER DIB header
- If a PNG image is used, it must use the rgba pixel format

22.6 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 "%0*N*d", this string specifies the position of the characters representing a numbering in the filenames. If the form "%0*N*d" is used, the string representing the number in each filename is 0-padded to *N* digits. The literal character ’%’ can be specified in the pattern with the string "%%".

If the pattern contains "%d" or "%0*N*d", the first filename of the file list specified will contain the number 1, all the following numbers will be sequential.

The pattern may contain a suffix which is used to automatically determine the format of the image files to write.

For example the pattern "img-%03d.bmp" will specify a sequence of filenames of the form ‘`img-001.bmp`’, ‘`img-002.bmp`’, ..., ‘`img-010.bmp`’, etc. The pattern "img%%-%d.jpg" will specify a sequence of filenames of the form ‘`img%-1.jpg`’, ‘`img%-2.jpg`’, ..., ‘`img%-10.jpg`’, etc.

The following example shows how to use `ffmpeg` for creating a sequence of files ‘`img-001.jpeg`’, ‘`img-002.jpeg`’, ..., taking one image every second from the input video:

```
ffmpeg -i in.avi -vsync 1 -r 1 -f image2 'img-%03d.jpeg'
```

Note that with `ffmpeg`, 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:

```
ffmpeg -i in.avi -vsync 1 -r 1 'img-%03d.jpeg'
```

Note also that the pattern must not necessarily contain "%d" or "%0*N*d", for example to create a single image file ‘`img.jpeg`’ from the input video you can employ the command:

```
ffmpeg -i in.avi -f image2 -frames:v 1 img.jpeg
```

‘`start_number` *`number`*’

Start the sequence from *number*. Default value is 1. Must be a positive 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.

The image muxer supports the .Y.U.V image file format. This format is special in that that each image frame consists of three files, for each of the YUV420P components. To read or write this image file format, specify the name of the '.Y' file. The muxer will automatically open the '.U' and '.V' files as required.

22.7 md5

MD5 testing format.

This muxer computes and prints the MD5 hash 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 hash.

The output of the muxer consists of a single line of the form: MD5=*MD5*, where *MD5* is a hexadecimal number representing the computed MD5 hash.

For example to compute the MD5 hash of the input converted to raw audio and video, and store it in the file '`out.md5`':

```
ffmpeg -i INPUT -f md5 out.md5
```

You can print the MD5 to stdout with the command:

```
ffmpeg -i INPUT -f md5 -
```

See also the framemd5 muxer.

22.8 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 by adding *faststart* to the *movflags*, or 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:

'`-moov_size` *`bytes`*'

> Reserves space for the moov atom at the beginning of the file instead of placing the moov atom at the end. If the space reserved is insufficient, muxing will fail.

'`-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 `ffmpeg`.)

‘`-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 moov atom on top 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 rtphint`’

Add RTP hinting tracks to the output file.

Smooth Streaming content can be pushed in real time to a publishing point on IIS with this muxer. Example:

```
ffmpeg -re <normal input/transcoding options> -movflags isml+frag_keyframe -f ismv http://server/publishingpoint.isml/Streams(Encoder1)
```

22.9 mpegts

MPEG transport stream muxer.

This muxer implements ISO 13818-1 and part of ETSI EN 300 468.

The muxer options are:

‘`-mpegts_original_network_id` *number*’

Set the original_network_id (default 0x0001). This is unique identifier of a network in DVB. Its main use is in the unique identification of a service through the path Original_Network_ID, Transport_Stream_ID.

‘`-mpegts_transport_stream_id` *number*’

Set the transport_stream_id (default 0x0001). This identifies a transponder in DVB.

‘`-mpegts_service_id` *number*’

Set the service_id (default 0x0001) also known as program in DVB.

‘`-mpegts_pmt_start_pid` *number*’

Set the first PID for PMT (default 0x1000, max 0x1f00).

‘`-mpegts_start_pid` *number*’

Set the first PID for data packets (default 0x0100, max 0x0f00).

The recognized metadata settings in mpegts muxer are `service_provider` and `service_name`. If they are not set the default for `service_provider` is "FFmpeg" and the default for `service_name` is "Service01".

```
ffmpeg -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
```

22.10 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 `ffmpeg` you can use the command:

```
ffmpeg -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 `ffmpeg` syntax.

Alternatively you can write the command as:

```
ffmpeg -benchmark -i INPUT -f null -
```

22.11 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:

```
ffmpeg -i sample_left_right_clip.mpg -an -c:v libvpx -metadata stereo_mode=left_right -y stereo_clip.webm
```

22.12 segment, stream_segment, ssegment

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.

`stream_segment` is a variant of the muxer used to write to streaming output formats, i.e. which do not require global headers, and is recommended for outputting e.g. to MPEG transport stream segments. `ssegment` is a shorter alias for `stream_segment`.

Every segment starts with a keyframe of the selected reference stream, which is set through the '`reference_stream`' option.

Note that if you want accurate splitting for a video file, you need to make the input key frames correspond to the exact splitting times expected by the segmenter, or the segment muxer will start the new segment with the key frame found next after the specified start time.

The segment muxer works best with a single constant frame rate video.

Optionally it can generate a list of the created segments, by setting the option *segment_list*. The list type is specified by the *segment_list_type* option.

The segment muxer supports the following options:

'`reference_stream` *`specifier`*'

Set the reference stream, as specified by the string *specifier*. If *specifier* is set to `auto`, the reference is choosen automatically. Otherwise it must be a stream specifier (see the "Stream specifiers" chapter in the ffmpeg manual) which specifies the reference stream. The default value is "auto".

'`segment_format` *`format`*'

Override the inner container format, by default it is guessed by the filename extension.

'`segment_list` *`name`*'

Generate also a listfile named *name*. If not specified no listfile is generated.

'`segment_list_flags` *`flags`*'

Set flags affecting the segment list generation.

It currently supports the following flags:

cache

Allow caching (only affects M3U8 list files).

live

Allow live-friendly file generation.

Default value is `cache`.

‘`segment_list_size` *`size`*’

Update the list file so that it contains at most the last *size* segments. If 0 the list file will contain all the segments. Default value is 0.

‘`segment_list type` *`type`*’

Specify the format for the segment list file.

The following values are recognized:

‘`flat`’

Generate a flat list for the created segments, one segment per line.

‘`csv, ext`’

Generate a list for the created segments, one segment per line, each line matching the format (comma-separated values):

```
segment_filename,segment_start_time,segment_end_time
```

segment_filename is the name of the output file generated by the muxer according to the provided pattern. CSV escaping (according to RFC4180) is applied if required.

segment_start_time and *segment_end_time* specify the segment start and end time expressed in seconds.

A list file with the suffix `".csv"` or `".ext"` will auto-select this format.

`ext` is deprecated in favor or `csv`.

‘`ffconcat`’

Generate an ffconcat file for the created segments. The resulting file can be read using the FFmpeg concat demuxer.

A list file with the suffix `".ffcat"` or `".ffconcat"` will auto-select this format.

‘`m3u8`’

Generate an extended M3U8 file, version 3, compliant with http://tools.ietf.org/id/draft-pantos-http-live-streaming.

A list file with the suffix `".m3u8"` will auto-select this format.

If not specified the type is guessed from the list file name suffix.

‘`segment_time` *`time`*’

Set segment duration to *time*, the value must be a duration specification. Default value is "2". See also the ‘`segment_times`’ option.

Note that splitting may not be accurate, unless you force the reference stream key-frames at the given time. See the introductory notice and the examples below.

‘`segment_time_delta` *`delta`*’

Specify the accuracy time when selecting the start time for a segment, expressed as a duration specification. Default value is "0".

When delta is specified a key-frame will start a new segment if its PTS satisfies the relation:

```
PTS >= start_time - time_delta
```

This option is useful when splitting video content, which is always split at GOP boundaries, in case a key frame is found just before the specified split time.

In particular may be used in combination with the ‘`ffmpeg`’ option *force_key_frames*. The key frame times specified by *force_key_frames* may not be set accurately because of rounding issues, with the consequence that a key frame time may result set just before the specified time. For constant frame rate videos a value of 1/2**frame_rate* should address the worst case mismatch between the specified time and the time set by *force_key_frames*.

‘`segment_times` *`times`*’

Specify a list of split points. *times* contains a list of comma separated duration specifications, in increasing order. See also the ‘`segment_time`’ option.

‘`segment_frames` *`frames`*’

Specify a list of split video frame numbers. *frames* contains a list of comma separated integer numbers, in increasing order.

This option specifies to start a new segment whenever a reference stream key frame is found and the sequential number (starting from 0) of the frame is greater or equal to the next value in the list.

‘`segment_wrap` *`limit`*’

Wrap around segment index once it reaches *limit*.

‘`segment_start_number` *`number`*’

Set the sequence number of the first segment. Defaults to `0`.

‘`reset_timestamps` *`1|0`*’

Reset timestamps at the begin of each segment, so that each segment will start with near-zero timestamps. It is meant to ease the playback of the generated segments. May not work with some combinations of muxers/codecs. It is set to `0` by default.

22.12.1 Examples

- To remux the content of file ‘`in.mkv`’ to a list of segments ‘`out-000.nut`’, ‘`out-001.nut`’, etc., and write the list of generated segments to ‘`out.list`’:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.list out%03d.nut
```

- As the example above, but segment the input file according to the split points specified by the *segment_times* option:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 out%03d.nut
```

- As the example above, but use the `ffmpeg` *force_key_frames* option to force key frames in the input at the specified location, together with the segment option *segment_time_delta* to account for possible roundings operated when setting key frame times.

```
ffmpeg -i in.mkv -force_key_frames 1,2,3,5,8,13,21 -codec:v mpeg4 -codec:a pcm_s16le -map 0 \
-f segment -segment_list out.csv -segment_times 1,2,3,5,8,13,21 -segment_time_delta 0.05 out%03d.nut
```

In order to force key frames on the input file, transcoding is required.

- Segment the input file by splitting the input file according to the frame numbers sequence specified with the *segment_frames* option:

```
ffmpeg -i in.mkv -codec copy -map 0 -f segment -segment_list out.csv -segment_frames 100,200,300,500,800 out%03d.nut
```

- To convert the ‘`in.mkv`’ to TS segments using the `libx264` and `libfaac` encoders:

```
ffmpeg -i in.mkv -map 0 -codec:v libx264 -codec:a libfaac -f ssegment -segment_list out.list out%03d.ts
```

- Segment the input file, and create an M3U8 live playlist (can be used as live HLS source):

```
ffmpeg -re -i in.mkv -codec copy -map 0 -f segment -segment_list playlist.m3u8 \
-segment_list_flags +live -segment_time 10 out%03d.mkv
```

22.13 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. The legacy ID3v1 tag is not written by default, but may be enabled with the `write_id3v1` option.

For seekable output the muxer also writes 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 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:

```
ffmpeg -i INPUT -id3v2_version 3 -write_id3v1 1 out.mp3
```

To attach a picture to an mp3 file select both the audio and the picture stream with `map`:

```
ffmpeg -i input.mp3 -i cover.png -c copy -map 0 -map 1
-metadata:s:v title="Album cover" -metadata:s:v comment="Cover (Front)" out.mp3
```

22.14 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.

22.15 tee

The tee muxer can be used to write the same data to several files or any other kind of muxer. It can be used, for example, to both stream a video to the network and save it to disk at the same time.

It is different from specifying several outputs to the `ffmpeg` command-line tool because the audio and video data will be encoded only once with the tee muxer; encoding can be a very expensive process. It is not useful when using the libavformat API directly because it is then possible to feed the same packets to several muxers directly.

The slave outputs are specified in the file name given to the muxer, separated by '|'. If any of the slave name contains the '|' separator, leading or trailing spaces or any special character, it must be escaped (see the "Quoting and escaping" section in the ffmpeg-utils manual).

Options can be specified for each slave by prepending them as a list of *key=value* pairs separated by ':', between square brackets. If the options values contain a special character or the ':' separator, they must be escaped; note that this is a second level escaping.

Example: encode something and both archive it in a WebM file and stream it as MPEG-TS over UDP (the streams need to be explicitly mapped):

```
ffmpeg -i ... -c:v libx264 -c:a mp2 -f tee -map 0:v -map 0:a
  "archive-20121107.mkv|[f=mpegts]udp://10.0.1.255:1234/"
```

Note: some codecs may need different options depending on the output format; the auto-detection of this can not work with the tee muxer. The main example is the '`global_header`' flag.

23. Metadata

FFmpeg is able to dump metadata from media files into a simple UTF-8-encoded INI-like text file and then load it back using the metadata muxer/demuxer.

The file format is as follows:

1. A file consists of a header and a number of metadata tags divided into sections, each on its own line.
2. The header is a ';FFMETADATA' string, followed by a version number (now 1).
3. Metadata tags are of the form 'key=value'
4. Immediately after header follows global metadata
5. After global metadata there may be sections with per-stream/per-chapter metadata.
6. A section starts with the section name in uppercase (i.e. STREAM or CHAPTER) in brackets ('[', ']')

and ends with next section or end of file.

7. At the beginning of a chapter section there may be an optional timebase to be used for start/end values. It must be in form ’TIMEBASE=num/den’, where num and den are integers. If the timebase is missing then start/end times are assumed to be in milliseconds. Next a chapter section must contain chapter start and end times in form ’START=num’, ’END=num’, where num is a positive integer.
8. Empty lines and lines starting with ’;’ or ’#’ are ignored.
9. Metadata keys or values containing special characters (’=’, ’;’, ’#’, ’\’ and a newline) must be escaped with a backslash ’\’.
10. Note that whitespace in metadata (e.g. foo = bar) is considered to be a part of the tag (in the example above key is ’foo ’, value is ’ bar’).

A ffmetadata file might look like this:

```
;FFMETADATA1
title=bike\\shed
;this is a comment
artist=FFmpeg troll team

[CHAPTER]
TIMEBASE=1/1000
START=0
#chapter ends at 0:01:00
END=60000
title=chapter \#1
[STREAM]
title=multi\
line
```

24. Protocols

Protocols are configured elements in FFmpeg which allow to access resources which require the use of a particular protocol.

When you configure your FFmpeg build, all the supported protocols are enabled by default. You can list all available ones using the configure option "–list-protocols".

You can disable all the protocols using the configure option "–disable-protocols", and selectively enable a protocol using the option "–enable-protocol=*PROTOCOL*", or you can disable a particular protocol using the option "–disable-protocol=*PROTOCOL*".

The option "-protocols" of the ff* tools will display the list of supported protocols.

A description of the currently available protocols follows.

24.1 bluray

Read BluRay playlist.

The accepted options are:

‘`angle`’

BluRay angle

‘`chapter`’

Start chapter (1...N)

‘`playlist`’

Playlist to read (BDMV/PLAYLIST/?????.mpls)

Examples:

Read longest playlist from BluRay mounted to /mnt/bluray:

```
bluray:/mnt/bluray
```

Read angle 2 of playlist 4 from BluRay mounted to /mnt/bluray, start from chapter 2:

```
-playlist 4 -angle 2 -chapter 2 bluray:/mnt/bluray
```

24.2 concat

Physical concatenation protocol.

Allow to read and seek from many resource in sequence as if they were a unique resource.

A URL accepted by this protocol has the syntax:

```
concat:URL1|URL2|...|URLN
```

where *URL1*, *URL2*, ..., *URLN* are the urls of the resource to be concatenated, each one possibly specifying a distinct protocol.

For example to read a sequence of files ‘`split1.mpeg`’, ‘`split2.mpeg`’, ‘`split3.mpeg`’ with `ffplay` use the command:

```
ffplay concat:split1.mpeg\|split2.mpeg\|split3.mpeg
```

Note that you may need to escape the character "|" which is special for many shells.

24.3 data

Data in-line in the URI. See http://en.wikipedia.org/wiki/Data_URI_scheme.

For example, to convert a GIF file given inline with `ffmpeg`:

```
ffmpeg -i "" smiley.png
```

24.4 file

File access protocol.

Allow to read from or read to a file.

For example to read from a file ‘`input.mpeg`’ with `ffmpeg` use the command:

```
ffmpeg -i file:input.mpeg output.mpeg
```

The ff* tools default to the file protocol, that is a resource specified with the name "FILE.mpeg" is interpreted as the URL "file:FILE.mpeg".

24.5 gopher

Gopher protocol.

24.6 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.

24.7 http

HTTP (Hyper Text Transfer Protocol).

This protocol accepts the following options.

‘`seekable`’

Control seekability of connection. If set to 1 the resource is supposed to be seekable, if set to 0 it is assumed not to be seekable, if set to -1 it will try to autodetect if it is seekable. Default value is -1.

‘`chunked_post`’

If set to 1 use chunked transfer-encoding for posts, default is 1.

‘`headers`’

Set custom HTTP headers, can override built in default headers. The value must be a string encoding the headers.

‘`content_type`’

Force a content type.

‘`user-agent`’

Override User-Agent header. If not specified the protocol will use a string describing the libavformat build.

‘`multiple_requests`’

Use persistent connections if set to 1. By default it is 0.

‘`post_data`’

Set custom HTTP post data.

‘`timeout`’

Set timeout of socket I/O operations used by the underlying low level operation. By default it is set to -1, which means that the timeout is not specified.

‘`mime_type`’

Set MIME type.

‘`cookies`’

Set the cookies to be sent in future requests. The format of each cookie is the same as the value of a Set-Cookie HTTP response field. Multiple cookies can be delimited by a newline character.

24.7.1 HTTP Cookies

Some HTTP requests will be denied unless cookie values are passed in with the request. The ‘`cookies`’ option allows these cookies to be specified. At the very least, each cookie must specify a value along with a path and domain. HTTP requests that match both the domain and path will automatically include the cookie value in the HTTP Cookie header field. Multiple cookies can be delimited by a newline.

The required syntax to play a stream specifying a cookie is:

```
ffplay -cookies "nlqptid=nltid=tsn; path=/; domain=somedomain.com;" http://somedomain.com/somestream.m3u8
```

24.8 mmst

MMS (Microsoft Media Server) protocol over TCP.

24.9 mmsh

MMS (Microsoft Media Server) protocol over HTTP.

The required syntax is:

```
mmsh://server[:port][/app][/playpath]
```

24.10 md5

MD5 output protocol.

Computes the MD5 hash of the data to be written, and on close writes this to the designated output or stdout if none is specified. It can be used to test muxers without writing an actual file.

Some examples follow.

```
# Write the MD5 hash of the encoded AVI file to the file output.avi.md5.
ffmpeg -i input.flv -f avi -y md5:output.avi.md5

# Write the MD5 hash of the encoded AVI file to stdout.
ffmpeg -i input.flv -f avi -y md5:
```

Note that some formats (typically MOV) require the output protocol to be seekable, so they will fail with the MD5 output protocol.

24.11 pipe

UNIX pipe access protocol.

Allow to read and write from UNIX pipes.

The accepted syntax is:

```
pipe:[number]
```

number is the number corresponding to the file descriptor of the pipe (e.g. 0 for stdin, 1 for stdout, 2 for stderr). If *number* is not specified, by default the stdout file descriptor will be used for writing, stdin for reading.

For example to read from stdin with `ffmpeg`:

```
cat test.wav | ffmpeg -i pipe:0
# ...this is the same as...
cat test.wav | ffmpeg -i pipe:
```

For writing to stdout with `ffmpeg`:

```
ffmpeg -i test.wav -f avi pipe:1 | cat > test.avi
# ...this is the same as...
ffmpeg -i test.wav -f avi pipe: | cat > test.avi
```

Note that some formats (typically MOV), require the output protocol to be seekable, so they will fail with the pipe output protocol.

24.12 rtmp

Real-Time Messaging Protocol.

The Real-Time Messaging Protocol (RTMP) is used for streaming multimedia content across a TCP/IP network.

The required syntax is:

```
rtmp://server[:port][/app][/instance][/playpath]
```

The accepted parameters are:

‘`server`’

The address of the RTMP server.

‘`port`’

The number of the TCP port to use (by default is 1935).

‘`app`’

It is the name of the application to access. It usually corresponds to the path where the application is installed on the RTMP server (e.g. ‘`/ondemand/`’, ‘`/flash/live/`’, etc.). You can override the value parsed from the URI through the `rtmp_app` option, too.

‘`playpath`’

It is the path or name of the resource to play with reference to the application specified in *app*, may be prefixed by "mp4:". You can override the value parsed from the URI through the `rtmp_playpath` option, too.

‘`listen`’

Act as a server, listening for an incoming connection.

‘`timeout`’

Maximum time to wait for the incoming connection. Implies listen.

Additionally, the following parameters can be set via command line options (or in code via `AVOption`s):

‘`rtmp_app`’

Name of application to connect on the RTMP server. This option overrides the parameter specified in the URI.

‘`rtmp_buffer`’

Set the client buffer time in milliseconds. The default is 3000.

‘`rtmp_conn`’

Extra arbitrary AMF connection parameters, parsed from a string, e.g. like `B:1 S:authMe O:1 NN:code:1.23 NS:flag:ok O:0`. Each value is prefixed by a single character denoting the type, B for Boolean, N for number, S for string, O for object, or Z for null, followed by a colon. For Booleans the data must be either 0 or 1 for FALSE or TRUE, respectively. Likewise for Objects the data must be 0 or 1 to end or begin an object, respectively. Data items in subobjects may be named, by prefixing the type with ’N’ and specifying the name before the value (i.e. `NB:myFlag:1`). This option may be used multiple times to construct arbitrary AMF sequences.

‘`rtmp_flashver`’

Version of the Flash plugin used to run the SWF player. The default is LNX 9,0,124,2.

‘rtmp_flush_interval’

Number of packets flushed in the same request (RTMPT only). The default is 10.

‘rtmp_live’

Specify that the media is a live stream. No resuming or seeking in live streams is possible. The default value is any, which means the subscriber first tries to play the live stream specified in the playpath. If a live stream of that name is not found, it plays the recorded stream. The other possible values are live and recorded.

‘rtmp_pageurl’

URL of the web page in which the media was embedded. By default no value will be sent.

‘rtmp_playpath’

Stream identifier to play or to publish. This option overrides the parameter specified in the URI.

‘rtmp_subscribe’

Name of live stream to subscribe to. By default no value will be sent. It is only sent if the option is specified or if rtmp_live is set to live.

‘rtmp_swfhash’

SHA256 hash of the decompressed SWF file (32 bytes).

‘rtmp_swfsize’

Size of the decompressed SWF file, required for SWFVerification.

‘rtmp_swfurl’

URL of the SWF player for the media. By default no value will be sent.

‘rtmp_swfverify’

URL to player swf file, compute hash/size automatically.

‘rtmp_tcurl’

URL of the target stream. Defaults to proto://host[:port]/app.

For example to read with ffplay a multimedia resource named "sample" from the application "vod" from an RTMP server "myserver":

```
ffplay rtmp://myserver/vod/sample
```

24.13 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.

24.14 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.

24.15 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.

24.16 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.

24.17 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.

24.18 rtmp, rtmpe, rtmps, rtmpt, rtmpte

Real-Time Messaging Protocol and its variants supported through librtmp.

Requires the presence of the librtmp headers and library during configuration. You need to explicitly configure the build with "–enable-librtmp". If enabled this will replace the native RTMP protocol.

This protocol provides most client functions and a few server functions needed to support RTMP, RTMP tunneled in HTTP (RTMPT), encrypted RTMP (RTMPE), RTMP over SSL/TLS (RTMPS) and tunneled variants of these encrypted types (RTMPTE, RTMPTS).

The required syntax is:

```
rtmp_proto://server[:port][/app][/playpath] options
```

where *rtmp_proto* is one of the strings "rtmp", "rtmpt", "rtmpe", "rtmps", "rtmpte", "rtmpts" corresponding to each RTMP variant, and *server*, *port*, *app* and *playpath* have the same meaning as specified for the RTMP native protocol. *options* contains a list of space-separated options of the form *key=val*.

See the librtmp manual page (man 3 librtmp) for more information.

For example, to stream a file in real-time to an RTMP server using `ffmpeg`:

```
ffmpeg -re -i myfile -f flv rtmp://myserver/live/mystream
```

To play the same stream using `ffplay`:

```
ffplay "rtmp://myserver/live/mystream live=1"
```

24.19 rtp

Real-Time Protocol.

24.20 rtsp

RTSP is not technically a protocol handler in libavformat, it is a demuxer and muxer. The demuxer supports both normal RTSP (with data transferred over RTP; this is used by e.g. Apple and Microsoft) and Real-RTSP (with data transferred over RDT).

The muxer can be used to send a stream using RTSP ANNOUNCE to a server supporting it (currently Darwin Streaming Server and Mischa Spiegelmock’s RTSP server).

The required syntax for a RTSP url is:

```
rtsp://hostname[:port]/path
```

The following options (set on the `ffmpeg`/`ffplay` command line, or set in code via `AVOption`s or in `avformat_open_input`), are supported:

Flags for `rtsp_transport`:

‘`udp`’

Use UDP as lower transport protocol.

‘`tcp`’

Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

‘`udp_multicast`’

Use UDP multicast as lower transport protocol.

‘`http`’

Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

Multiple lower transport protocols may be specified, in that case they are tried one at a time (if the setup of one fails, the next one is tried). For the muxer, only the `tcp` and `udp` options are supported.

Flags for `rtsp_flags`:

‘`filter_src`’

Accept packets only from negotiated peer address and port.

‘`listen`’

Act as a server, listening for an incoming connection.

When receiving data over UDP, the demuxer tries to reorder received packets (since they may arrive out of order, or packets may get lost totally). This can be disabled by setting the maximum demuxing delay to zero (via the `max_delay` field of AVFormatContext).

When watching multi-bitrate Real-RTSP streams with `ffplay`, the streams to display can be chosen with `-vst` *n* and `-ast` *n* for video and audio respectively, and can be switched on the fly by pressing `v` and `a`.

Example command lines:

To watch a stream over UDP, with a max reordering delay of 0.5 seconds:

```
ffplay -max_delay 500000 -rtsp_transport udp rtsp://server/video.mp4
```

To watch a stream tunneled over HTTP:

```
ffplay -rtsp_transport http rtsp://server/video.mp4
```

To send a stream in realtime to a RTSP server, for others to watch:

```
ffmpeg -re -i input -f rtsp -muxdelay 0.1 rtsp://server/live.sdp
```

To receive a stream in realtime:

```
ffmpeg -rtsp_flags listen -i rtsp://ownaddress/live.sdp output
```

‘`stimeout`’

Socket IO timeout in micro seconds.

24.21 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.

24.21.1 Muxer

The syntax for a SAP url given to the muxer is:

```
sap://destination[:port][?options]
```

The RTP packets are sent to *destination* on port *port*, or to port 5004 if no port is specified. *options* is a &-separated list. The following options are supported:

‘`announce_addr=address`’

Specify the destination IP address for sending the announcements to. If omitted, the announcements are sent to the commonly used SAP announcement multicast address 224.2.127.254 (sap.mcast.net), or ff0e::2:7ffe if *destination* is an IPv6 address.

‘`announce_port=port`’

Specify the port to send the announcements on, defaults to 9875 if not specified.

‘`ttl=ttl`’

Specify the time to live value for the announcements and RTP packets, defaults to 255.

‘`same_port=`*`0|1`*’

If set to 1, send all RTP streams on the same port pair. If zero (the default), all streams are sent on unique ports, with each stream on a port 2 numbers higher than the previous. VLC/Live555 requires this to be set to 1, to be able to receive the stream. The RTP stack in libavformat for receiving requires all streams to be sent on unique ports.

Example command lines follow.

To broadcast a stream on the local subnet, for watching in VLC:

```
ffmpeg -re -i input -f sap sap://224.0.0.255?same_port=1
```

Similarly, for watching in `ffplay`:

```
ffmpeg -re -i input -f sap sap://224.0.0.255
```

And for watching in `ffplay`, over IPv6:

```
ffmpeg -re -i input -f sap sap://[ff0e::1:2:3:4]
```

24.21.2 Demuxer

The syntax for a SAP url given to the demuxer is:

```
sap://[address][:port]
```

address is the multicast address to listen for announcements on, if omitted, the default 224.2.127.254 (sap.mcast.net) is used. *port* is the port that is listened on, 9875 if omitted.

The demuxers listens for announcements on the given address and port. Once an announcement is received, it tries to receive that particular stream.

Example command lines follow.

To play back the first stream announced on the normal SAP multicast address:

```
ffplay sap://
```

To play back the first stream announced on one the default IPv6 SAP multicast address:

```
ffplay sap://[ff0e::2:7ffe]
```

24.22 tcp

Trasmission Control Protocol.

The required syntax for a TCP url is:

```
tcp://hostname:port[?options]
```

‘`listen`’

Listen for an incoming connection

‘`timeout=microseconds`’

In read mode: if no data arrived in more than this time interval, raise error. In write mode: if socket cannot be written in more than this time interval, raise error. This also sets timeout on TCP connection establishing.

```
ffmpeg -i input -f format tcp://hostname:port?listen
ffplay tcp://hostname:port
```

24.23 tls

Transport Layer Security/Secure Sockets Layer

The required syntax for a TLS/SSL url is:

```
tls://hostname:port[?options]
```

‘`listen`’

Act as a server, listening for an incoming connection.

‘`cafile=filename`’

Certificate authority file. The file must be in OpenSSL PEM format.

‘`cert=filename`’

Certificate file. The file must be in OpenSSL PEM format.

‘`key=filename`’

Private key file.

‘verify=*0|1*’

Verify the peer’s certificate.

Example command lines:

To create a TLS/SSL server that serves an input stream.

```
ffmpeg -i input -f format tls://hostname:port?listen&cert=server.crt&key=server.key
```

To play back a stream from the TLS/SSL server using `ffplay`:

```
ffplay tls://hostname:port
```

24.24 udp

User Datagram Protocol.

The required syntax for a UDP url is:

```
udp://hostname:port[?options]
```

options contains a list of &-separated options of the form *key=val*.

In case threading is enabled on the system, a circular buffer is used to store the incoming data, which allows to reduce loss of data due to UDP socket buffer overruns. The *fifo_size* and *overrun_nonfatal* options are related to this buffer.

The list of supported options follows.

‘buffer_size=*size*’

Set the UDP socket buffer size in bytes. This is used both for the receiving and the sending buffer size.

‘localport=*port*’

Override the local UDP port to bind with.

‘localaddr=*addr*’

Choose the local IP address. This is useful e.g. if sending multicast and the host has multiple interfaces, where the user can choose which interface to send on by specifying the IP address of that interface.

‘pkt_size=*size*’

Set the size in bytes of UDP packets.

‘reuse=*1|0*’

Explicitly allow or disallow reusing UDP sockets.

‘ttl=*ttl*’

Set the time to live value (for multicast only).

‘connect=*1|0*’

Initialize the UDP socket with `connect()`. In this case, the destination address can’t be changed with ff_udp_set_remote_url later. If the destination address isn’t known at the start, this option can be specified in ff_udp_set_remote_url, too. This allows finding out the source address for the packets with getsockname, and makes writes return with AVERROR(ECONNREFUSED) if "destination unreachable" is received. For receiving, this gives the benefit of only receiving packets from the specified peer address/port.

‘sources=*address*[,*address*]’

Only receive packets sent to the multicast group from one of the specified sender IP addresses.

‘block=*address*[,*address*]’

Ignore packets sent to the multicast group from the specified sender IP addresses.

‘fifo_size=*units*’

Set the UDP receiving circular buffer size, expressed as a number of packets with size of 188 bytes. If not specified defaults to 7*4096.

‘overrun_nonfatal=*1|0*’

Survive in case of UDP receiving circular buffer overrun. Default value is 0.

‘timeout=*microseconds*’

In read mode: if no data arrived in more than this time interval, raise error.

Some usage examples of the UDP protocol with `ffmpeg` follow.

To stream over UDP to a remote endpoint:

```
ffmpeg -i input -f format udp://hostname:port
```

To stream in mpegts format over UDP using 188 sized UDP packets, using a large input buffer:

```
ffmpeg -i input -f mpegts udp://hostname:port?pkt_size=188&buffer_size=65535
```

To receive over UDP from a remote endpoint:

```
ffmpeg -i udp://[multicast-address]:port
```

25. Device Options

The libavdevice library provides the same interface as libavformat. Namely, an input device is considered like a demuxer, and an output device like a muxer, and the interface and generic device options are the same provided by libavformat (see the ffmpeg-formats manual).

In addition each input or output device may support so-called private options, which are specific for that component.

Options may be set by specifying *-option value* in the FFmpeg tools, or by setting the value explicitly in the device `AVFormatContext` options or using the ‘`libavutil/opt.h`’ API for programmatic use.

26. Input Devices

Input devices are configured elements in FFmpeg which allow to access the data coming from a multimedia device attached to your system.

When you configure your FFmpeg build, all the supported input devices are enabled by default. You can list all available ones using the configure option "–list-indevs".

You can disable all the input devices using the configure option "–disable-indevs", and selectively enable an input device using the option "–enable-indev=*INDEV*", or you can disable a particular input device using the option "–disable-indev=*INDEV*".

The option "-formats" of the ff* tools will display the list of supported input devices (amongst the demuxers).

A description of the currently available input devices follows.

26.1 alsa

ALSA (Advanced Linux Sound Architecture) input device.

To enable this input device during configuration you need libasound installed on your system.

This device allows capturing from an ALSA device. The name of the device to capture has to be an ALSA card identifier.

An ALSA identifier has the syntax:

```
hw:CARD[,DEV[,SUBDEV]]
```

where the *DEV* and *SUBDEV* components are optional.

The three arguments (in order: *CARD*,*DEV*,*SUBDEV*) specify card number or identifier, device number and subdevice number (-1 means any).

To see the list of cards currently recognized by your system check the files ‘`/proc/asound/cards`’ and ‘`/proc/asound/devices`’.

For example to capture with `ffmpeg` from an ALSA device with card id 0, you may run the command:

```
ffmpeg -f alsa -i hw:0 alsaout.wav
```

For more information see: http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html

26.2 bktr

BSD video input device.

26.3 dshow

Windows DirectShow input device.

DirectShow support is enabled when FFmpeg is built with the mingw-w64 project. Currently only audio and video devices are supported.

Multiple devices may be opened as separate inputs, but they may also be opened on the same input, which should improve synchronism between them.

The input name should be in the format:

```
TYPE=NAME[:TYPE=NAME]
```

where *TYPE* can be either *audio* or *video*, and *NAME* is the device’s name.

26.3.1 Options

If no options are specified, the device’s defaults are used. If the device does not support the requested options, it will fail to open.

‘`video_size`’

Set the video size in the captured video.

‘`framerate`’

Set the frame rate in the captured video.

‘`sample_rate`’

Set the sample rate (in Hz) of the captured audio.

‘`sample_size`’

Set the sample size (in bits) of the captured audio.

‘`channels`’

Set the number of channels in the captured audio.

‘`list_devices`’

If set to ‘`true`’, print a list of devices and exit.

‘`list_options`’

If set to ‘`true`’, print a list of selected device’s options and exit.

‘`video_device_number`’

Set video device number for devices with same name (starts at 0, defaults to 0).

‘`audio_device_number`’

Set audio device number for devices with same name (starts at 0, defaults to 0).

‘`pixel_format`’

Select pixel format to be used by DirectShow. This may only be set when the video codec is not set or set to rawvideo.

‘`audio_buffer_size`’

Set audio device buffer size in milliseconds (which can directly impact latency, depending on the device). Defaults to using the audio device's default buffer size (typically some multiple of 500ms). Setting this value too low can degrade performance. See also http://msdn.microsoft.com/en-us/library/windows/desktop/dd377582(v=vs.85).aspx

26.3.2 Examples

- Print the list of DirectShow supported devices and exit:

```
$ ffmpeg -list_devices true -f dshow -i dummy
```

- Open video device *Camera*:

```
$ ffmpeg -f dshow -i video="Camera"
```

- Open second video device with name *Camera*:

```
$ ffmpeg -f dshow -video_device_number 1 -i video="Camera"
```

- Open video device *Camera* and audio device *Microphone*:

```
$ ffmpeg -f dshow -i video="Camera":audio="Microphone"
```

- Print the list of supported options in selected device and exit:

```
$ ffmpeg -list_options true -f dshow -i video="Camera"
```

26.4 dv1394

Linux DV 1394 input device.

26.5 fbdev

Linux framebuffer input device.

The Linux framebuffer is a graphic hardware-independent abstraction layer to show graphics on a computer monitor, typically on the console. It is accessed through a file device node, usually '`/dev/fb0`'.

For more detailed information read the file Documentation/fb/framebuffer.txt included in the Linux source tree.

To record from the framebuffer device ‘/dev/fb0’ with ffmpeg:

```
ffmpeg -f fbdev -r 10 -i /dev/fb0 out.avi
```

You can take a single screenshot image with the command:

```
ffmpeg -f fbdev -frames:v 1 -r 1 -i /dev/fb0 screenshot.jpeg
```

See also http://linux-fbdev.sourceforge.net/, and fbset(1).

26.6 iec61883

FireWire DV/HDV input device using libiec61883.

To enable this input device, you need libiec61883, libraw1394 and libavc1394 installed on your system. Use the configure option `--enable-libiec61883` to compile with the device enabled.

The iec61883 capture device supports capturing from a video device connected via IEEE1394 (FireWire), using libiec61883 and the new Linux FireWire stack (juju). This is the default DV/HDV input method in Linux Kernel 2.6.37 and later, since the old FireWire stack was removed.

Specify the FireWire port to be used as input file, or "auto" to choose the first port connected.

26.6.1 Options

‘dvtype’

Override autodetection of DV/HDV. This should only be used if auto detection does not work, or if usage of a different device type should be prohibited. Treating a DV device as HDV (or vice versa) will not work and result in undefined behavior. The values ‘auto’, ‘dv’ and ‘hdv’ are supported.

‘dvbuffer’

Set maxiumum size of buffer for incoming data, in frames. For DV, this is an exact value. For HDV, it is not frame exact, since HDV does not have a fixed frame size.

‘dvguid’

Select the capture device by specifying it’s GUID. Capturing will only be performed from the specified device and fails if no device with the given GUID is found. This is useful to select the input if multiple devices are connected at the same time. Look at /sys/bus/firewire/devices to find out the GUIDs.

26.6.2 Examples

- Grab and show the input of a FireWire DV/HDV device.

```
ffplay -f iec61883 -i auto
```

- Grab and record the input of a FireWire DV/HDV device, using a packet buffer of 100000 packets if the source is HDV.

```
ffmpeg -f iec61883 -i auto -hdvbuffer 100000 out.mpg
```

26.7 jack

JACK input device.

To enable this input device during configuration you need libjack installed on your system.

A JACK input device creates one or more JACK writable clients, one for each audio channel, with name *client_name*:input_*N*, where *client_name* is the name provided by the application, and *N* is a number which identifies the channel. Each writable client will send the acquired data to the FFmpeg input device.

Once you have created one or more JACK readable clients, you need to connect them to one or more JACK writable clients.

To connect or disconnect JACK clients you can use the `jack_connect` and `jack_disconnect` programs, or do it through a graphical interface, for example with `qjackctl`.

To list the JACK clients and their properties you can invoke the command `jack_lsp`.

Follows an example which shows how to capture a JACK readable client with `ffmpeg`.

```
# Create a JACK writable client with name "ffmpeg".
$ ffmpeg -f jack -i ffmpeg -y out.wav

# Start the sample jack_metro readable client.
$ jack_metro -b 120 -d 0.2 -f 4000

# List the current JACK clients.
$ jack_lsp -c
system:capture_1
system:capture_2
system:playback_1
system:playback_2
ffmpeg:input_1
metro:120_bpm

# Connect metro to the ffmpeg writable client.
$ jack_connect metro:120_bpm ffmpeg:input_1
```

For more information read: http://jackaudio.org/

26.8 lavfi

Libavfilter input virtual device.

This input device reads data from the open output pads of a libavfilter filtergraph.

For each filtergraph open output, the input device will create a corresponding stream which is mapped to the generated output. Currently only video data is supported. The filtergraph is specified through the option ‘`graph`’.

26.8.1 Options

‘`graph`’

Specify the filtergraph to use as input. Each video open output must be labelled by a unique string of the form "out*N*", where *N* is a number starting from 0 corresponding to the mapped input stream generated by the device. The first unlabelled output is automatically assigned to the "out0" label, but all the others need to be specified explicitly.

If not specified defaults to the filename specified for the input device.

‘`graph_file`’

Set the filename of the filtergraph to be read and sent to the other filters. Syntax of the filtergraph is the same as the one specified by the option *graph*.

26.8.2 Examples

- Create a color video stream and play it back with `ffplay`:

```
ffplay -f lavfi -graph "color=c=pink [out0]" dummy
```

- As the previous example, but use filename for specifying the graph description, and omit the "out0" label:

```
ffplay -f lavfi color=c=pink
```

- Create three different video test filtered sources and play them:

```
ffplay -f lavfi -graph "testsrc [out0]; testsrc,hflip [out1]; testsrc,negate [out2]" test3
```

- Read an audio stream from a file using the amovie source and play it back with `ffplay`:

```
ffplay -f lavfi "amovie=test.wav"
```

- Read an audio stream and a video stream and play it back with `ffplay`:

```
ffplay -f lavfi "movie=test.avi[out0];amovie=test.wav[out1]"
```

26.9 libdc1394

IIDC1394 input device, based on libdc1394 and libraw1394.

26.10 openal

The OpenAL input device provides audio capture on all systems with a working OpenAL 1.1 implementation.

To enable this input device during configuration, you need OpenAL headers and libraries installed on your system, and need to configure FFmpeg with `--enable-openal`.

OpenAL headers and libraries should be provided as part of your OpenAL implementation, or as an additional download (an SDK). Depending on your installation you may need to specify additional flags via the `--extra-cflags` and `--extra-ldflags` for allowing the build system to locate the OpenAL headers and libraries.

An incomplete list of OpenAL implementations follows:

Creative

The official Windows implementation, providing hardware acceleration with supported devices and software fallback. See http://openal.org/.

OpenAL Soft

Portable, open source (LGPL) software implementation. Includes backends for the most common sound APIs on the Windows, Linux, Solaris, and BSD operating systems. See http://kcat.strangesoft.net/openal.html.

Apple

OpenAL is part of Core Audio, the official Mac OS X Audio interface. See http://developer.apple.com/technologies/mac/audio-and-video.html

This device allows to capture from an audio input device handled through OpenAL.

You need to specify the name of the device to capture in the provided filename. If the empty string is provided, the device will automatically select the default device. You can get the list of the supported devices by using the option *list_devices*.

26.10.1 Options

‘`channels`’

Set the number of channels in the captured audio. Only the values ‘`1`’ (monaural) and ‘`2`’ (stereo) are currently supported. Defaults to ‘`2`’.

‘`sample_size`’

Set the sample size (in bits) of the captured audio. Only the values ‘`8`’ and ‘`16`’ are currently supported. Defaults to ‘`16`’.

‘`sample_rate`’

Set the sample rate (in Hz) of the captured audio. Defaults to ‘`44.1k`’.

‘`list_devices`’

If set to ‘`true`’, print a list of devices and exit. Defaults to ‘`false`’.

26.10.2 Examples

Print the list of OpenAL supported devices and exit:

```
$ ffmpeg -list_devices true -f openal -i dummy out.ogg
```

Capture from the OpenAL device ‘`DR-BT101 via PulseAudio`’:

```
$ ffmpeg -f openal -i 'DR-BT101 via PulseAudio' out.ogg
```

Capture from the default device (note the empty string ” as filename):

```
$ ffmpeg -f openal -i '' out.ogg
```

Capture from two devices simultaneously, writing to two different files, within the same `ffmpeg` command:

```
$ ffmpeg -f openal -i 'DR-BT101 via PulseAudio' out1.ogg -f openal -i 'ALSA Default' out2.ogg
```

Note: not all OpenAL implementations support multiple simultaneous capture - try the latest OpenAL Soft if the above does not work.

26.11 oss

Open Sound System input device.

The filename to provide to the input device is the device node representing the OSS input device, and is usually set to '`/dev/dsp`'.

For example to grab from '`/dev/dsp`' using `ffmpeg` use the command:

```
ffmpeg -f oss -i /dev/dsp /tmp/oss.wav
```

For more information about OSS see: http://manuals.opensound.com/usersguide/dsp.html

26.12 pulse

pulseaudio input device.

To enable this input device during configuration you need libpulse-simple installed in your system.

The filename to provide to the input device is a source device or the string "default"

To list the pulse source devices and their properties you can invoke the command `pactl list sources`.

```
ffmpeg -f pulse -i default /tmp/pulse.wav
```

26.12.1 *server* AVOption

The syntax is:

```
-server server name
```

Connects to a specific server.

26.12.2 *name* AVOption

The syntax is:

```
-name application name
```

Specify the application name pulse will use when showing active clients, by default it is the LIBAVFORMAT_IDENT string

26.12.3 *stream_name* AVOption

The syntax is:

```
-stream_name stream name
```

Specify the stream name pulse will use when showing active streams, by default it is "record"

26.12.4 *sample_rate* AVOption

The syntax is:

```
-sample_rate samplerate
```

Specify the samplerate in Hz, by default 48kHz is used.

26.12.5 *channels* AVOption

The syntax is:

```
-channels N
```

Specify the channels in use, by default 2 (stereo) is set.

26.12.6 *frame_size* AVOption

The syntax is:

```
-frame_size bytes
```

Specify the number of byte per frame, by default it is set to 1024.

26.12.7 *fragment_size* AVOption

The syntax is:

```
-fragment_size bytes
```

Specify the minimal buffering fragment in pulseaudio, it will affect the audio latency. By default it is unset.

26.13 sndio

sndio input device.

To enable this input device during configuration you need libsndio installed on your system.

The filename to provide to the input device is the device node representing the sndio input device, and is usually set to '`/dev/audio0`'.

For example to grab from '`/dev/audio0`' using `ffmpeg` use the command:

```
ffmpeg -f sndio -i /dev/audio0 /tmp/oss.wav
```

26.14 video4linux2, v4l2

Video4Linux2 input video device.

"v4l2" can be used as alias for "video4linux2".

If FFmpeg is built with v4l-utils support (by using the `--enable-libv4l2` configure option), the device will always rely on libv4l2.

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 frame rates. You can check which are supported using `-list_formats all` for Video4Linux2 devices. Some devices, like TV cards, support one or more standards. It is possible to list all the supported standards using `-list_standards all`.

The time base for the timestamps is 1 microsecond. Depending on the kernel version and configuration, the timestamps may be derived from the real time clock (origin at the Unix Epoch) or the monotonic clock (origin usually at boot time, unaffected by NTP or manual changes to the clock). The '`-timestamps abs`' or '`-ts abs`' option can be used to force conversion into the real time clock.

Some usage examples of the video4linux2 device with `ffmpeg` and `ffplay`:

- Grab and show the input of a video4linux2 device:

  ```
  ffplay -f video4linux2 -framerate 30 -video_size hd720 /dev/video0
  ```

- Grab and record the input of a video4linux2 device, leave the frame rate and size as previously set:

  ```
  ffmpeg -f video4linux2 -input_format mjpeg -i /dev/video0 out.mpeg
  ```

For more information about Video4Linux, check http://linuxtv.org/.

26.14.1 Options

‘`standard`’

Set the standard. Must be the name of a supported standard. To get a list of the supported standards, use the ‘`list_standards`’ option.

‘`channel`’

Set the input channel number. Default to -1, which means using the previously selected channel.

‘`video_size`’

Set the video frame size. The argument must be a string in the form *WIDTH*x*HEIGHT* or a valid size abbreviation.

‘`pixel_format`’

Select the pixel format (only valid for raw video input).

‘`input_format`’

Set the preferred pixel format (for raw video) or a codec name. This option allows to select the input format, when several are available.

‘`framerate`’

Set the preferred video frame rate.

‘`list_formats`’

List available formats (supported pixel formats, codecs, and frame sizes) and exit.

Available values are:

‘`all`’

Show all available (compressed and non-compressed) formats.

‘`raw`’

Show only raw video (non-compressed) formats.

‘`compressed`’

Show only compressed formats.

‘list_standards’

List supported standards and exit.

Available values are:

‘all’

Show all supported standards.

‘timestamps, ts’

Set type of timestamps for grabbed frames.

Available values are:

‘default’

Use timestamps from the kernel.

‘abs’

Use absolute timestamps (wall clock).

‘mono2abs’

Force conversion from monotonic to absolute timestamps.

Default value is default.

26.15 vfwcap

VfW (Video for Windows) capture input device.

The filename passed as input is the capture driver number, ranging from 0 to 9. You may use "list" as filename to print a list of drivers. Any other filename will be interpreted as device number 0.

26.16 x11grab

X11 video input device.

This device allows to capture a region of an X11 display.

The filename passed as input has the syntax:

```
[hostname]:display_number.screen_number[+x_offset,y_offset]
```

hostname:*display_number*.*screen_number* specifies the X11 display name of the screen to grab from. *hostname* can be omitted, and defaults to "localhost". The environment variable `DISPLAY` contains the default display name.

x_offset and *y_offset* specify the offsets of the grabbed area with respect to the top-left border of the X11 screen. They default to 0.

Check the X11 documentation (e.g. man X) for more detailed information.

Use the `dpyinfo` program for getting basic information about the properties of your X11 display (e.g. grep for "name" or "dimensions").

For example to grab from ‘`:0.0`’ using `ffmpeg`:

```
ffmpeg -f x11grab -r 25 -s cif -i :0.0 out.mpg
```

Grab at position `10,20`:

```
ffmpeg -f x11grab -r 25 -s cif -i :0.0+10,20 out.mpg
```

26.16.1 Options

‘`draw_mouse`’

Specify whether to draw the mouse pointer. A value of `0` specify not to draw the pointer. Default value is `1`.

‘`follow_mouse`’

Make the grabbed area follow the mouse. The argument can be `centered` or a number of pixels *PIXELS*.

When it is specified with "centered", the grabbing region follows the mouse pointer and keeps the pointer at the center of region; otherwise, the region follows only when the mouse pointer reaches within *PIXELS* (greater than zero) to the edge of region.

For example:

```
ffmpeg -f x11grab -follow_mouse centered -r 25 -s cif -i :0.0 out.mpg
```

To follow only when the mouse pointer reaches within 100 pixels to edge:

```
ffmpeg -f x11grab -follow_mouse 100 -r 25 -s cif -i :0.0 out.mpg
```

‘framerate’

Set the grabbing frame rate. Default value is ntsc, corresponding to a frame rate of 30000/1001.

‘show_region’

Show grabbed region on screen.

If *show_region* is specified with 1, then the grabbing region will be indicated on screen. With this option, it is easy to know what is being grabbed if only a portion of the screen is grabbed.

For example:

```
ffmpeg -f x11grab -show_region 1 -r 25 -s cif -i :0.0+10,20 out.mpg
```

With *follow_mouse*:

```
ffmpeg -f x11grab -follow_mouse centered -show_region 1 -r 25 -s cif -i :0.0 out.mpg
```

‘video_size’

Set the video frame size. Default value is vga.

27. Output Devices

Output devices are configured elements in FFmpeg which allow to write multimedia data to an output device attached to your system.

When you configure your FFmpeg build, all the supported output devices are enabled by default. You can list all available ones using the configure option "–list-outdevs".

You can disable all the output devices using the configure option "–disable-outdevs", and selectively enable an output device using the option "–enable-outdev=*OUTDEV*", or you can disable a particular input device using the option "–disable-outdev=*OUTDEV*".

The option "-formats" of the ff* tools will display the list of enabled output devices (amongst the muxers).

A description of the currently available output devices follows.

27.1 alsa

ALSA (Advanced Linux Sound Architecture) output device.

27.2 caca

CACA output device.

This output devices allows to show a video stream in CACA window. Only one CACA window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need to configure FFmpeg with `--enable-libcaca`. libcaca is a graphics library that outputs text instead of pixels.

For more information about libcaca, check: http://caca.zoy.org/wiki/libcaca

27.2.1 Options

‘`window_title`’

Set the CACA window title, if not specified default to the filename specified for the output device.

‘`window_size`’

Set the CACA window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video.

‘`driver`’

Set display driver.

‘`algorithm`’

Set dithering algorithm. Dithering is necessary because the picture being rendered has usually far more colours than the available palette. The accepted values are listed with `-list_dither algorithms`.

‘`antialias`’

Set antialias method. Antialiasing smoothens the rendered image and avoids the commonly seen staircase effect. The accepted values are listed with `-list_dither antialiases`.

‘`charset`’

Set which characters are going to be used when rendering text. The accepted values are listed with `-list_dither charsets`.

‘color’

Set color to be used when rendering text. The accepted values are listed with `-list_dither colors`.

‘list_drivers’

If set to ‘true’, print a list of available drivers and exit.

‘list_dither’

List available dither options related to the argument. The argument must be one of `algorithms`, `antialiases`, `charsets`, `colors`.

27.2.2 Examples

- The following command shows the `ffmpeg` output is an CACA window, forcing its size to 80x25:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt rgb24 -window_size 80x25 -f caca -
```

- Show the list of available drivers and exit:

```
ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_drivers true -
```

- Show the list of available dither colors and exit:

```
ffmpeg -i INPUT -pix_fmt rgb24 -f caca -list_dither colors -
```

27.3 oss

OSS (Open Sound System) output device.

27.4 sdl

SDL (Simple DirectMedia Layer) output device.

This output devices allows to show a video stream in an SDL window. Only one SDL window is allowed per application, so you can have only one instance of this output device in an application.

To enable this output device you need libsdl installed on your system when configuring your build.

For more information about SDL, check: http://www.libsdl.org/

27.4.1 Options

‘window_title’

Set the SDL window title, if not specified default to the filename specified for the output device.

‘icon_title’

Set the name of the iconified SDL window, if not specified it is set to the same value of *window_title*.

‘window_size’

Set the SDL window size, can be a string of the form *width*x*height* or a video size abbreviation. If not specified it defaults to the size of the input video, downscaled according to the aspect ratio.

27.4.2 Examples

The following command shows the ffmpeg output is an SDL window, forcing its size to the qcif format:

```
ffmpeg -i INPUT -vcodec rawvideo -pix_fmt yuv420p -window_size qcif -f sdl "SDL output"
```

27.5 sndio

sndio audio output device.

28. Resampler Options

The audio resampler supports the following named options.

Options may be set by specifying *-option value* in the FFmpeg tools, *option=value* for the aresample filter, by setting the value explicitly in the SwrContext options or using the ‘libavutil/opt.h’ API for programmatic use.

‘ich, in_channel_count’

Set the number of input channels. Default value is 0. Setting this value is not mandatory if the corresponding channel layout ‘in_channel_layout’ is set.

‘och, out_channel_count’

Set the number of output channels. Default value is 0. Setting this value is not mandatory if the corresponding channel layout ‘out_channel_layout’ is set.

‘uch, used_channel_count’

Set the number of used input channels. Default value is 0. This option is only used for special remapping.

‘`isr, in_sample_rate`’

Set the input sample rate. Default value is 0.

‘`osr, out_sample_rate`’

Set the output sample rate. Default value is 0.

‘`isf, in_sample_fmt`’

Specify the input sample format. It is set by default to `none`.

‘`osf, out_sample_fmt`’

Specify the output sample format. It is set by default to `none`.

‘`tsf, internal_sample_fmt`’

Set the internal sample format. Default value is `none`. This will automatically be chosen when it is not explicitly set.

‘`icl, in_channel_layout`’

Set the input channel layout.

‘`ocl, out_channel_layout`’

Set the output channel layout.

‘`clev, center_mix_level`’

Set the center mix level. It is a value expressed in deciBel, and must be in the interval [-32,32].

‘`slev, surround_mix_level`’

Set the surround mix level. It is a value expressed in deciBel, and must be in the interval [-32,32].

‘`lfe_mix_level`’

Set LFE mix into non LFE level. It is used when there is a LFE input but no LFE output. It is a value expressed in deciBel, and must be in the interval [-32,32].

‘`rmvol, rematrix_volume`’

Set rematrix volume. Default value is 1.0.

‘flags, swr_flags’

Set flags used by the converter. Default value is 0.

It supports the following individual flags:

‘res’

force resampling, this flag forces resampling to be used even when the input and output sample rates match.

‘dither_scale’

Set the dither scale. Default value is 1.

‘dither_method’

Set dither method. Default value is 0.

Supported values:

‘rectangular’

select rectangular dither

‘triangular’

select triangular dither

‘triangular_hp’

select triangular dither with high pass

‘lipshitz’

select lipshitz noise shaping dither

‘shibata’

select shibata noise shaping dither

‘low_shibata’

select low shibata noise shaping dither

‘high_shibata’

select high shibata noise shaping dither

‘f_weighted’

select f-weighted noise shaping dither

‘modified_e_weighted’

select modified-e-weighted noise shaping dither

‘improved_e_weighted’

select improved-e-weighted noise shaping dither

‘resampler’

Set resampling engine. Default value is swr.

Supported values:

‘swr’

select the native SW Resampler; filter options precision and cheby are not applicable in this case.

‘soxr’

select the SoX Resampler (where available); compensation, and filter options filter_size, phase_shift, filter_type & kaiser_beta, are not applicable in this case.

‘filter_size’

For swr only, set resampling filter size, default value is 32.

‘phase_shift’

For swr only, set resampling phase shift, default value is 10, and must be in the interval [0,30].

‘linear_interp’

Use Linear Interpolation if set to 1, default value is 0.

‘cutoff’

Set cutoff frequency (swr: 6dB point; soxr: 0dB point) ratio; must be a float value between 0 and 1. Default value is 0.97 with swr, and 0.91 with soxr (which, with a sample-rate of 44100, preserves the entire audio band to 20kHz).

‘precision’

For soxr only, the precision in bits to which the resampled signal will be calculated. The default value of 20 (which, with suitable dithering, is appropriate for a destination bit-depth of 16) gives SoX's 'High Quality'; a value of 28 gives SoX's 'Very High Quality'.

'cheby'

For soxr only, selects passband rolloff none (Chebyshev) & higher-precision approximation for 'irrational' ratios. Default value is 0.

'async'

For swr only, simple 1 parameter audio sync to timestamps using stretching, squeezing, filling and trimming. Setting this to 1 will enable filling and trimming, larger values represent the maximum amount in samples that the data may be stretched or squeezed for each second. Default value is 0, thus no compensation is applied to make the samples match the audio timestamps.

'first_pts'

For swr only, assume the first pts should be this value. The time unit is 1 / sample rate. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame's expected pts, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream or to trim any samples with a negative pts due to encoder delay.

'min_comp'

For swr only, set the minimum difference between timestamps and audio data (in seconds) to trigger stretching/squeezing/filling or trimming of the data to make it match the timestamps. The default is that stretching/squeezing/filling and trimming is disabled ('min_comp' = FLT_MAX).

'min_hard_comp'

For swr only, set the minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples to make it match the timestamps. This option effectively is a threshold to select between hard (trim/fill) and soft (squeeze/stretch) compensation. Note that all compensation is by default disabled through 'min_comp'. The default is 0.1.

'comp_duration'

For swr only, set duration (in seconds) over which data is stretched/squeezed to make it match the timestamps. Must be a non-negative double float value, default value is 1.0.

'max_soft_comp'

For swr only, set maximum factor by which data is stretched/squeezed to make it match the timestamps. Must be a non-negative double float value, default value is 0.

‘matrix_encoding’

Select matrixed stereo encoding.

It accepts the following values:

‘none’

select none

‘dolby’

select Dolby

‘dplii’

select Dolby Pro Logic II

Default value is none.

‘filter_type’

For swr only, select resampling filter type. This only affects resampling operations.

It accepts the following values:

‘cubic’

select cubic

‘blackman_nuttall’

select Blackman Nuttall Windowed Sinc

‘kaiser’

select Kaiser Windowed Sinc

‘kaiser_beta’

For swr only, set Kaiser Window Beta value. Must be an integer in the interval [2,16], default value is 9.

29. Scaler Options

The video scaler supports the following named options.

Options may be set by specifying *-option value* in the FFmpeg tools. For programmatic use, they can be set explicitly in the `SwsContext` options or through the ‘`libavutil/opt.h`’ API.

‘`sws_flags`’

Set the scaler flags. This is also used to set the scaling algorithm. Only a single algorithm should be selected.

It accepts the following values:

‘`fast_bilinear`’

Select fast bilinear scaling algorithm.

‘`bilinear`’

Select bilinear scaling algorithm.

‘`bicubic`’

Select bicubic scaling algorithm.

‘`experimental`’

Select experimental scaling algorithm.

‘`neighbor`’

Select nearest neighbor rescaling algorithm.

‘`area`’

Select averaging area rescaling algorithm.

‘`bicubiclin`’

Select bicubic scaling algorithm for the luma component, bilinear for chroma components.

‘`gauss`’

Select Gaussian rescaling algorithm.

‘`sinc`’

Select sinc rescaling algorithm.

‘`lanczos`’

Select lanczos rescaling algorithm.

‘spline’

Select natural bicubic spline rescaling algorithm.

‘print_info’

Enable printing/debug logging.

‘accurate_rnd’

Enable accurate rounding.

‘full_chroma_int’

Enable full chroma interpolation.

‘full_chroma_inp’

Select full chroma input.

‘bitexact’

Enable bitexact output.

‘srcw’

Set source width.

‘srch’

Set source height.

‘dstw’

Set destination width.

‘dsth’

Set destination height.

‘src_format’

Set source pixel format (must be expressed as an integer).

‘dst_format’

Set destination pixel format (must be expressed as an integer).

‘src_range’

Select source range.

‘dst_range’

Select destination range.

‘param0, param1’

Set scaling algorithm parameters. The specified values are specific of some scaling algorithms and ignored by others. The specified values are floating point number values.

30. Filtering Introduction

Filtering in FFmpeg is enabled through the libavfilter library.

In libavfilter, a filter can have multiple inputs and multiple outputs. To illustrate the sorts of things that are possible, we consider the following filtergraph.

```
input --> split ----------------------> overlay --> output
            |                             ^
            |                             |
            +-----> crop --> vflip -------+
```

This filtergraph splits the input stream in two streams, sends one stream through the crop filter and the vflip filter before merging it back with the other stream by overlaying it on top. You can use the following command to achieve this:

```
ffmpeg -i INPUT -vf "split [main][tmp]; [tmp] crop=iw:ih/2:0:0, vflip [flip]; [main][flip] overlay=0:H/2" OUTPUT
```

The result will be that in output the top half of the video is mirrored onto the bottom half.

Filters in the same linear chain are separated by commas, and distinct linear chains of filters are separated by semicolons. In our example, *crop,vflip* are in one linear chain, *split* and *overlay* are separately in another. The points where the linear chains join are labelled by names enclosed in square brackets. In the example, the split filter generates two outputs that are associated to the labels *[main]* and *[tmp]*.

The stream sent to the second output of *split*, labelled as *[tmp]*, is processed through the *crop* filter, which crops away the lower half part of the video, and then vertically flipped. The *overlay* filter takes in input the first unchanged output of the split filter (which was labelled as *[main]*), and overlay on its lower half the output generated by the *crop,vflip* filterchain.

Some filters take in input a list of parameters: they are specified after the filter name and an equal sign, and are separated from each other by a colon.

There exist so-called *source filters* that do not have an audio/video input, and *sink filters* that will not have audio/video output.

31. graph2dot

The ‘`graph2dot`’ program included in the FFmpeg ‘`tools`’ directory can be used to parse a filtergraph description and issue a corresponding textual representation in the dot language.

Invoke the command:

```
graph2dot -h
```

to see how to use ‘`graph2dot`’.

You can then pass the dot description to the ‘`dot`’ program (from the graphviz suite of programs) and obtain a graphical representation of the filtergraph.

For example the sequence of commands:

```
echo GRAPH_DESCRIPTION | \
tools/graph2dot -o graph.tmp && \
dot -Tpng graph.tmp -o graph.png && \
display graph.png
```

can be used to create and display an image representing the graph described by the *GRAPH_DESCRIPTION* string. Note that this string must be a complete self-contained graph, with its inputs and outputs explicitly defined. For example if your command line is of the form:

```
ffmpeg -i infile -vf scale=640:360 outfile
```

your *GRAPH_DESCRIPTION* string will need to be of the form:

```
nullsrc,scale=640:360,nullsink
```

you may also need to set the *nullsrc* parameters and add a *format* filter in order to simulate a specific input file.

32. Filtergraph description

A filtergraph is a directed graph of connected filters. It can contain cycles, and there can be multiple links between a pair of filters. Each link has one input pad on one side connecting it to one filter from which it takes its input, and one output pad on the other side connecting it to the one filter accepting its output.

Each filter in a filtergraph is an instance of a filter class registered in the application, which defines the features and the number of input and output pads of the filter.

A filter with no input pads is called a "source", a filter with no output pads is called a "sink".

32.1 Filtergraph syntax

A filtergraph can be represented using a textual representation, which is recognized by the ‘`-filter`’/‘`-vf`’ and ‘`-filter_complex`’ options in `ffmpeg` and ‘`-vf`’ in `ffplay`, and by the `avfilter_graph_parse()`/`avfilter_graph_parse2()` function defined in ‘`libavfilter/avfilter.h`’.

A filterchain consists of a sequence of connected filters, each one connected to the previous one in the sequence. A filterchain is represented by a list of ","-separated filter descriptions.

A filtergraph consists of a sequence of filterchains. A sequence of filterchains is represented by a list of ";"-separated filterchain descriptions.

A filter is represented by a string of the form:
[*in_link_1*]...[*in_link_N*]*filter_name=arguments*[*out_link_1*]...[*out_link_M*]

filter_name is the name of the filter class of which the described filter is an instance of, and has to be the name of one of the filter classes registered in the program. The name of the filter class is optionally followed by a string "=*arguments*".

arguments is a string which contains the parameters used to initialize the filter instance. It may have one of the following forms:

- A ’:’-separated list of *key=value* pairs.
- A ’:’-separated list of *value*. In this case, the keys are assumed to be the option names in the order they are declared. E.g. the `fade` filter declares three options in this order – ‘`type`’, ‘`start_frame`’ and ‘`nb_frames`’. Then the parameter list *in:0:30* means that the value *in* is assigned to the option ‘`type`’, *0* to ‘`start_frame`’ and *30* to ‘`nb_frames`’.
- A ’:’-separated list of mixed direct *value* and long *key=value* pairs. The direct *value* must precede the *key=value* pairs, and follow the same constraints order of the previous point. The following *key=value* pairs can be set in any preferred order.

If the option value itself is a list of items (e.g. the `format` filter takes a list of pixel formats), the items in the list are usually separated by ’|’.

The list of arguments can be quoted using the character "’" as initial and ending mark, and the character ’\’ for escaping the characters within the quoted text; otherwise the argument string is considered terminated when the next special character (belonging to the set "[]=;,") is encountered.

The name and arguments of the filter are optionally preceded and followed by a list of link labels. A link label allows to name a link and associate it to a filter output or input pad. The preceding labels *in_link_1* ... *in_link_N*, are associated to the filter input pads, the following labels *out_link_1* ... *out_link_M*, are associated to the output pads.

When two link labels with the same name are found in the filtergraph, a link between the corresponding input and output pad is created.

If an output pad is not labelled, it is linked by default to the first unlabelled input pad of the next filter in the filterchain. For example in the filterchain:

```
nullsrc, split[L1], [L2]overlay, nullsink
```

the split filter instance has two output pads, and the overlay filter instance two input pads. The first output pad of split is labelled "L1", the first input pad of overlay is labelled "L2", and the second output pad of split is linked to the second input pad of overlay, which are both unlabelled.

In a complete filterchain all the unlabelled filter input and output pads must be connected. A filtergraph is considered valid if all the filter input and output pads of all the filterchains are connected.

Libavfilter will automatically insert scale filters where format conversion is required. It is possible to specify swscale flags for those automatically inserted scalers by prepending `sws_flags=`*`flags`*`;` to the filtergraph description.

Follows a BNF description for the filtergraph syntax:

```
NAME             ::= sequence of alphanumeric characters and '_'
LINKLABEL        ::= "[" NAME "]"
LINKLABELS       ::= LINKLABEL [LINKLABELS]
FILTER_ARGUMENTS ::= sequence of chars (eventually quoted)
FILTER           ::= [LINKLABELS] NAME ["=" FILTER_ARGUMENTS] [LINKLABELS]
FILTERCHAIN      ::= FILTER [,FILTERCHAIN]
FILTERGRAPH      ::= [sws_flags=flags;] FILTERCHAIN [;FILTERGRAPH]
```

32.2 Notes on filtergraph escaping

Some filter arguments require the use of special characters, typically `:` to separate key=value pairs in a named options list. In this case the user should perform a first level escaping when specifying the filter arguments. For example, consider the following literal string to be embedded in the drawtext filter arguments:

```
this is a 'string': may contain one, or more, special characters
```

Since : is special for the filter arguments syntax, it needs to be escaped, so you get:

```
text=this is a \'string\'\: may contain one, or more, special characters
```

A second level of escaping is required when embedding the filter arguments in a filtergraph description, in order to escape all the filtergraph special characters. Thus the example above becomes:

```
drawtext=text=this is a \\\'string\\\'\\: may contain one\, or more\, special characters
```

Finally an additional level of escaping may be needed when writing the filtergraph description in a shell command, which depends on the escaping rules of the adopted shell. For example, assuming that \ is special and needs to be escaped with another \, the previous string will finally result in:

```
-vf "drawtext=text=this is a \\\\\\'string\\\\\\'\\\\: may contain one\\, or more\\, special characters"
```

Sometimes, it might be more convenient to employ quoting in place of escaping. For example the string:

```
Caesar: tu quoque, Brute, fili mi
```

Can be quoted in the filter arguments as:

```
text='Caesar: tu quoque, Brute, fili mi'
```

And finally inserted in a filtergraph like:

```
drawtext=text=\'Caesar: tu quoque\, Brute\, fili mi\'
```

See the “Quoting and escaping” section in the ffmpeg-utils manual for more information about the escaping and quoting rules adopted by FFmpeg.

33. Timeline editing

Some filters support a generic ‘`enable`’ option. For the filters supporting timeline editing, this option can be set to an expression which is evaluated before sending a frame to the filter. If the evaluation is non-zero, the filter will be enabled, otherwise the frame will be sent unchanged to the next filter in the filtergraph.

The expression accepts the following values:

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘n’

sequential number of the input frame, starting from 0

‘pos’

the position in the file of the input frame, NAN if unknown

Additionally, these filters support an ‘enable’ command that can be used to re-define the expression.

Like any other filtering option, the ‘enable’ option follows the same rules.

For example, to enable a blur filter (smartblur) from 10 seconds to 3 minutes, and a curves filter starting at 3 seconds:

```
smartblur = enable='between(t,10,3*60)',
curves    = enable='gte(t,3)' : preset=cross_process
```

34. Audio Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the audio filters included in your build.

Below is a description of the currently available audio filters.

34.1 aconvert

Convert the input audio format to the specified formats.

This filter is deprecated. Use aformat instead.

The filter accepts a string of the form: "*sample_format*:*channel_layout*".

sample_format specifies the sample format, and can be a string or the corresponding numeric value defined in ‘libavutil/samplefmt.h’. Use ’p’ suffix for a planar sample format.

channel_layout specifies the channel layout, and can be a string or the corresponding number value defined in ‘libavutil/channel_layout.h’.

The special parameter "auto", signifies that the filter will automatically select the output format depending on the output filter.

34.1.1 Examples

- Convert input to float, planar, stereo:

```
aconvert=fltp:stereo
```

- Convert input to unsigned 8-bit, automatically select out channel layout:

```
aconvert=u8:auto
```

34.2 allpass

Apply a two-pole all-pass filter with central frequency (in Hz) *frequency*, and filter-width *width*. An all-pass filter changes the audio's frequency to phase relationship without changing its frequency to amplitude relationship.

The filter accepts the following options:

'`frequency, f`'

Set frequency in Hz.

'`width_type`'

Set method to specify band-width of filter.

'`h`'

Hz

'`q`'

Q-Factor

'`o`'

octave

'`s`'

slope

'`width, w`'

Specify the band-width of a filter in width_type units.

34.3 highpass

Apply a high-pass filter with 3dB point frequency. The filter can be either single-pole, or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

‘`frequency, f`’

Set frequency in Hz. Default is 3000.

‘`poles, p`’

Set number of poles. Default is 2.

‘`width_type`’

Set method to specify band-width of filter.

‘`h`’

Hz

‘`q`’

Q-Factor

‘`o`’

octave

‘`s`’

slope

‘`width, w`’

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

34.4 lowpass

Apply a low-pass filter with 3dB point frequency. The filter can be either single-pole or double-pole (the default). The filter roll off at 6dB per pole per octave (20dB per pole per decade).

The filter accepts the following options:

‘frequency, f’

Set frequency in Hz. Default is 500.

‘poles, p’

Set number of poles. Default is 2.

‘width_type’

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Specify the band-width of a filter in width_type units. Applies only to double-pole filter. The default is 0.707q and gives a Butterworth response.

34.5 bass

Boost or cut the bass (lower) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi’s tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

‘gain, g’

Give the gain at 0 Hz. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

‘frequency, f’

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `100` Hz.

'`width_type`'

Set method to specify band-width of filter.

'`h`'

Hz

'`q`'

Q-Factor

'`o`'

octave

'`s`'

slope

'`width, w`'

Determine how steep is the filter's shelf transition.

34.6 treble

Boost or cut treble (upper) frequencies of the audio using a two-pole shelving filter with a response similar to that of a standard hi-fi's tone-controls. This is also known as shelving equalisation (EQ).

The filter accepts the following options:

'`gain, g`'

Give the gain at whichever is the lower of ~22 kHz and the Nyquist frequency. Its useful range is about -20 (for a large cut) to +20 (for a large boost). Beware of clipping when using a positive gain.

'`frequency, f`'

Set the filter's central frequency and so can be used to extend or reduce the frequency range to be boosted or cut. The default value is `3000` Hz.

'`width_type`'

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Determine how steep is the filter’s shelf transition.

34.7 bandpass

Apply a two-pole Butterworth band-pass filter with central frequency *frequency*, and (3dB-point) band-width width. The *csg* option selects a constant skirt gain (peak gain = Q) instead of the default: constant 0dB peak gain. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

‘frequency, f’

Set the filter’s central frequency. Default is 3000.

‘csg’

Constant skirt gain if set to 1. Defaults to 0.

‘width_type’

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Specify the band-width of a filter in width_type units.

34.8 bandreject

Apply a two-pole Butterworth band-reject filter with central frequency *frequency*, and (3dB-point) band-width *width*. The filter roll off at 6dB per octave (20dB per decade).

The filter accepts the following options:

‘frequency, f’

Set the filter’s central frequency. Default is `3000`.

‘width_type’

Set method to specify band-width of filter.

‘h’

Hz

‘q’

Q-Factor

‘o’

octave

‘s’

slope

‘width, w’

Specify the band-width of a filter in width_type units.

34.9 biquad

Apply a biquad IIR filter with the given coefficients. Where *b0*, *b1*, *b2* and *a0*, *a1*, *a2* are the numerator and denominator coefficients respectively.

34.10 equalizer

Apply a two-pole peaking equalisation (EQ) filter. With this filter, the signal-level at and around a selected frequency can be increased or decreased, whilst (unlike bandpass and bandreject filters) that at all other frequencies is unchanged.

In order to produce complex equalisation curves, this filter can be given several times, each with a different central frequency.

The filter accepts the following options:

‘`frequency, f`’

Set the filter’s central frequency in Hz.

‘`width_type`’

Set method to specify band-width of filter.

‘`h`’

Hz

‘`q`’

Q-Factor

‘`o`’

octave

‘`s`’

slope

‘`width, w`’

Specify the band-width of a filter in width_type units.

‘`gain, g`’

Set the required gain or attenuation in dB. Beware of clipping when using a positive gain.

34.11 afade

Apply fade-in/out effect to input audio.

A description of the accepted parameters follows.

‘type, t’

Specify the effect type, can be either in for fade-in, or out for a fade-out effect. Default is in.

‘start_sample, ss’

Specify the number of the start sample for starting to apply the fade effect. Default is 0.

‘nb_samples, ns’

Specify the number of samples for which the fade effect has to last. At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. Default is 44100.

‘start_time, st’

Specify time for starting to apply the fade effect. Default is 0. The accepted syntax is:

```
[-]HH[:MM[:SS[.m...]]]
[-]S+[.m...]
```

See also the function av_parse_time(). If set this option is used instead of *start_sample* one.

‘duration, d’

Specify the duration for which the fade effect has to last. Default is 0. The accepted syntax is:

```
[-]HH[:MM[:SS[.m...]]]
[-]S+[.m...]
```

See also the function av_parse_time(). At the end of the fade-in effect the output audio will have the same volume as the input audio, at the end of the fade-out transition the output audio will be silence. If set this option is used instead of *nb_samples* one.

‘curve’

Set curve for fade transition.

It accepts the following values:

‘`tri`’

select triangular, linear slope (default)

‘`qsin`’

select quarter of sine wave

‘`hsin`’

select half of sine wave

‘`esin`’

select exponential sine wave

‘`log`’

select logarithmic

‘`par`’

select inverted parabola

‘`qua`’

select quadratic

‘`cub`’

select cubic

‘`squ`’

select square root

‘`cbr`’

select cubic root

34.11.1 Examples

- Fade in first 15 seconds of audio:

```
afade=t=in:ss=0:d=15
```

- Fade out last 25 seconds of a 900 seconds audio:

```
afade=t=out:st=875:d=25
```

34.12 aformat

Set output format constraints for the input audio. The framework will negotiate the most appropriate format to minimize conversions.

The filter accepts the following named parameters:

‘`sample_fmts`’

A ’|’-separated list of requested sample formats.

‘`sample_rates`’

A ’|’-separated list of requested sample rates.

‘`channel_layouts`’

A ’|’-separated list of requested channel layouts.

If a parameter is omitted, all values are allowed.

For example to force the output to either unsigned 8-bit or signed 16-bit stereo:

```
aformat=sample_fmts=u8|s16:channel_layouts=stereo
```

34.13 amerge

Merge two or more audio streams into a single multi-channel stream.

The filter accepts the following options:

‘`inputs`’

Set the number of inputs. Default is 2.

If the channel layouts of the inputs are disjoint, and therefore compatible, the channel layout of the output will be set accordingly and the channels will be reordered as necessary. If the channel layouts of the inputs are not disjoint, the output will have all the channels of the first input then all the channels of the second input, in that order, and the channel layout of the output will be the default value corresponding to the total number of channels.

For example, if the first input is in 2.1 (FL+FR+LF) and the second input is FC+BL+BR, then the output will be in 5.1, with the channels in the following order: a1, a2, b1, a3, b2, b3 (a1 is the first channel of the first input, b1 is the first channel of the second input).

On the other hand, if both input are in stereo, the output channels will be in the default order: a1, a2, b1, b2, and the channel layout will be arbitrarily set to 4.0, which may or may not be the expected value.

All inputs must have the same sample rate, and format.

If inputs do not have the same duration, the output will stop with the shortest.

34.13.1 Examples

- Merge two mono files into a stereo stream:

```
amovie=left.wav [l] ; amovie=right.mp3 [r] ; [l] [r] amerge
```

- Multiple merges assuming 1 video stream and 6 audio streams in ‘`input.mkv`’:

```
ffmpeg -i input.mkv -filter_complex "[0:1][0:2][0:3][0:4][0:5][0:6] amerge=inputs=6" -c:a pcm_s16le output.mkv
```

34.14 amix

Mixes multiple audio inputs into a single output.

For example

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex amix=inputs=3:duration=first:dropout_transition=3 OUTPUT
```

will mix 3 input audio streams to a single output with the same duration as the first input and a dropout transition time of 3 seconds.

The filter accepts the following named parameters:

‘`inputs`’

Number of inputs. If unspecified, it defaults to 2.

‘`duration`’

How to determine the end-of-stream.

‘`longest`’

Duration of longest input. (default)

‘shortest’

Duration of shortest input.

‘first’

Duration of first input.

‘dropout_transition’

Transition time, in seconds, for volume renormalization when an input stream ends. The default value is 2 seconds.

34.15 anull

Pass the audio source unchanged to the output.

34.16 apad

Pad the end of a audio stream with silence, this can be used together with -shortest to extend audio streams to the same length as the video stream.

34.17 aphaser

Add a phasing effect to the input audio.

A phaser filter creates series of peaks and troughs in the frequency spectrum. The position of the peaks and troughs are modulated so that they vary over time, creating a sweeping effect.

A description of the accepted parameters follows.

‘in_gain’

Set input gain. Default is 0.4.

‘out_gain’

Set output gain. Default is 0.74

‘delay’

Set delay in milliseconds. Default is 3.0.

‘decay’

Set decay. Default is 0.4.

‘speed’

Set modulation speed in Hz. Default is 0.5.

‘type’

Set modulation type. Default is triangular.

It accepts the following values:

‘triangular, t’
‘sinusoidal, s’

34.18 aresample

Resample the input audio to the specified parameters, using the libswresample library. If none are specified then the filter will automatically convert between its input and output.

This filter is also able to stretch/squeeze the audio data to make it match the timestamps or to inject silence / cut out audio to make it match the timestamps, do a combination of both or do neither.

The filter accepts the syntax [*sample_rate*:]*resampler_options*, where *sample_rate* expresses a sample rate and *resampler_options* is a list of *key*=*value* pairs, separated by ":". See the ffmpeg-resampler manual for the complete list of supported options.

34.18.1 Examples

- Resample the input audio to 44100Hz:

```
aresample=44100
```

- Stretch/squeeze samples to the given timestamps, with a maximum of 1000 samples per second compensation:

```
aresample=async=1000
```

34.19 asetnsamples

Set the number of samples per each output audio frame.

The last output packet may contain a different number of samples, as the filter will flush all the remaining samples when the input audio signal its end.

The filter accepts the following options:

‘nb_out_samples, n’

Set the number of frames per each output audio frame. The number is intended as the number of samples *per each channel*. Default value is 1024.

‘pad, p’

If set to 1, the filter will pad the last audio frame with zeroes, so that the last frame will contain the same number of samples as the previous ones. Default value is 1.

For example, to set the number of per-frame samples to 1234 and disable padding for the last frame, use:

```
asetnsamples=n=1234:p=0
```

34.20 asetpts

Change the PTS (presentation timestamp) of the input audio frames.

This filter accepts the following options:

‘expr’

The expression which is evaluated for each frame to construct its timestamp.

The expression is evaluated through the eval API and can contain the following constants:

‘PTS’

the presentation timestamp in input

‘PI’

Greek PI

‘PHI’

golden ratio

‘E’

Euler number

‘N’

Number of the audio samples pass through the filter so far, starting at 0.

‘S’

Number of the audio samples in the current frame.

‘SR’

Audio sample rate.

‘STARTPTS’

the PTS of the first frame

‘PREV_INPTS’

previous input PTS

‘PREV_OUTPTS’

previous output PTS

‘RTCTIME’

wallclock (RTC) time in microseconds

‘RTCSTART’

wallclock (RTC) time at the start of the movie in microseconds

Some examples follow:

```
# start counting PTS from zero
asetpts=expr=PTS-STARTPTS

#generate timestamps by counting samples
asetpts=expr=N/SR/TB

# generate timestamps from a "live source" and rebase onto the current timebase
asetpts='(RTCTIME - RTCSTART) / (TB * 1000000)"
```

34.21 asetrate

Set the sample rate without altering the PCM data. This will result in a change of speed and pitch.

The filter accepts the following options:

‘sample_rate, r’

Set the output sample rate. Default is 44100 Hz.

34.22 ashowinfo

Show a line containing various information for each input audio frame. The input audio is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

A description of each shown parameter follows:

‘n’

sequential number of the input frame, starting from 0

‘pts’

Presentation timestamp of the input frame, in time base units; the time base depends on the filter input pad, and is usually 1/*sample_rate*.

‘pts_time’

presentation timestamp of the input frame in seconds

‘pos’

position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic audio)

‘fmt’

sample format

‘chlayout’

channel layout

‘rate’

sample rate for the audio frame

‘nb_samples’

number of samples (per channel) in the frame

‘checksum’

Adler-32 checksum (printed in hexadecimal) of the audio data. For planar audio the data is treated as if all the planes were concatenated.

‘plane_checksums’

A list of Adler-32 checksums for each data plane.

34.23 astats

Display time domain statistical information about the audio channels. Statistics are calculated and displayed for each audio channel and, where applicable, an overall figure is also given.

The filter accepts the following option:

‘length’

Short window length in seconds, used for peak and trough RMS measurement. Default is `0.05` (50 miliseconds). Allowed range is `[0.1 - 10]`.

A description of each shown parameter follows:

‘DC offset’

Mean amplitude displacement from zero.

‘Min level’

Minimal sample level.

‘Max level’

Maximal sample level.

‘Peak level dB’
‘RMS level dB’

Standard peak and RMS level measured in dBFS.

‘RMS peak dB’
‘RMS trough dB’

Peak and trough values for RMS level measured over a short window.

‘Crest factor’

Standard ratio of peak to RMS level (note: not in dB).

‘Flat factor’

Flatness (i.e. consecutive samples with the same value) of the signal at its peak levels (i.e. either *Min level* or *Max level*).

‘Peak count’

Number of occasions (not the number of samples) that the signal attained either *Min level* or *Max level*.

34.24 astreamsync

Forward two audio streams and control the order the buffers are forwarded.

The filter accepts the following options:

‘expr, e’

Set the expression deciding which stream should be forwarded next: if the result is negative, the first stream is forwarded; if the result is positive or zero, the second stream is forwarded. It can use the following variables:

b1 b2

number of buffers forwarded so far on each stream

s1 s2

number of samples forwarded so far on each stream

t1 t2

current timestamp of each stream

The default value is `t1-t2`, which means to always forward the stream that has a smaller timestamp.

34.24.1 Examples

Stress-test `amerge` by randomly sending buffers on the wrong input, while avoiding too much of a desynchronization:

```
amovie=file.ogg [a] ; amovie=file.mp3 [b] ;
[a] [b] astreamsync=(2*random(1))-1+tanh(5*(t1-t2)) [a2] [b2] ;
[a2] [b2] amerge
```

34.25 atempo

Adjust audio tempo.

The filter accepts exactly one parameter, the audio tempo. If not specified then the filter will assume nominal 1.0 tempo. Tempo must be in the [0.5, 2.0] range.

34.25.1 Examples

- Slow down audio to 80% tempo:

```
atempo=0.8
```

- To speed up audio to 125% tempo:

```
atempo=1.25
```

34.26 earwax

Make audio easier to listen to on headphones.

This filter adds ‘cues’ to 44.1kHz stereo (i.e. audio CD format) audio so that when listened to on headphones the stereo image is moved from inside your head (standard for headphones) to outside and in front of the listener (standard for speakers).

Ported from SoX.

34.27 pan

Mix channels with specific gain levels. The filter accepts the output channel layout followed by a set of channels definitions.

This filter is also designed to remap efficiently the channels of an audio stream.

The filter accepts parameters of the form: "*l*:*outdef*:*outdef*:..."

‘`l`’

output channel layout or number of channels

‘`outdef`’

output channel specification, of the form: "*out_name*=[*gain**]*in_name*[+[*gain**]*in_name*...]"

‘out_name’

output channel to define, either a channel name (FL, FR, etc.) or a channel number (c0, c1, etc.)

‘gain’

multiplicative coefficient for the channel, 1 leaving the volume unchanged

‘in_name’

input channel to use, see out_name for details; it is not possible to mix named and numbered input channels

If the ‘=’ in a channel specification is replaced by ‘<’, then the gains for that specification will be renormalized so that the total is 1, thus avoiding clipping noise.

34.27.1 Mixing examples

For example, if you want to down-mix from stereo to mono, but with a bigger factor for the left channel:

```
pan=1:c0=0.9*c0+0.1*c1
```

A customized down-mix to stereo that works automatically for 3-, 4-, 5- and 7-channels surround:

```
pan=stereo: FL < FL + 0.5*FC + 0.6*BL + 0.6*SL : FR < FR + 0.5*FC + 0.6*BR + 0.6*SR
```

Note that `ffmpeg` integrates a default down-mix (and up-mix) system that should be preferred (see "-ac" option) unless you have very specific needs.

34.27.2 Remapping examples

The channel remapping will be effective if, and only if:

- gain coefficients are zeroes or ones,
- only one input per channel output,

If all these conditions are satisfied, the filter will notify the user ("Pure channel mapping detected"), and use an optimized and lossless method to do the remapping.

For example, if you have a 5.1 source and want a stereo audio stream by dropping the extra channels:

```
pan="stereo: c0=FL : c1=FR"
```

Given the same source, you can also switch front left and front right channels and keep the input channel layout:

```
pan="5.1: c0=c1 : c1=c0 : c2=c2 : c3=c3 : c4=c4 : c5=c5"
```

If the input is a stereo audio stream, you can mute the front left channel (and still keep the stereo channel layout) with:

```
pan="stereo:c1=c1"
```

Still with a stereo audio stream input, you can copy the right channel in both front left and right:

```
pan="stereo: c0=FR : c1=FR"
```

34.28 silencedetect

Detect silence in an audio stream.

This filter logs a message when it detects that the input audio volume is less or equal to a noise tolerance value for a duration greater or equal to the minimum detected noise duration.

The printed times and duration are expressed in seconds.

The filter accepts the following options:

‘`duration, d`’

Set silence duration until notification (default is 2 seconds).

‘`noise, n`’

Set noise tolerance. Can be specified in dB (in case "dB" is appended to the specified value) or amplitude ratio. Default is -60dB, or 0.001.

34.28.1 Examples

- Detect 5 seconds of silence with -50dB noise tolerance:

  ```
  silencedetect=n=-50dB:d=5
  ```

- Complete example with `ffmpeg` to detect silence with 0.0001 noise tolerance in ‘`silence.mp3`’:

  ```
  ffmpeg -i silence.mp3 -af silencedetect=noise=0.0001 -f null -
  ```

34.29 asyncts

Synchronize audio data with timestamps by squeezing/stretching it and/or dropping samples/adding silence when needed.

This filter is not built by default, please use aresample to do squeezing/stretching.

The filter accepts the following named parameters:

‘`compensate`’

Enable stretching/squeezing the data to make it match the timestamps. Disabled by default. When disabled, time gaps are covered with silence.

‘`min_delta`’

Minimum difference between timestamps and audio data (in seconds) to trigger adding/dropping samples. Default value is 0.1. If you get non-perfect sync with this filter, try setting this parameter to 0.

‘`max_comp`’

Maximum compensation in samples per second. Relevant only with compensate=1. Default value 500.

‘`first_pts`’

Assume the first pts should be this value. The time base is 1 / sample rate. This allows for padding/trimming at the start of stream. By default, no assumption is made about the first frame’s expected pts, so no padding or trimming is done. For example, this could be set to 0 to pad the beginning with silence if an audio stream starts after the video stream or to trim any samples with a negative pts due to encoder delay.

34.30 atrim

Trim the input so that the output contains one continuous subpart of the input.

This filter accepts the following options:

‘`start`’

Timestamp (in seconds) of the start of the kept section. I.e. the audio sample with the timestamp *start* will be the first sample in the output.

‘`end`’

Timestamp (in seconds) of the first audio sample that will be dropped. I.e. the audio sample immediately preceding the one with the timestamp *end* will be the last sample in the output.

‘`start_pts`’

Same as *start*, except this option sets the start timestamp in samples instead of seconds.

‘`end_pts`’

Same as *end*, except this option sets the end timestamp in samples instead of seconds.

‘`duration`’

Maximum duration of the output in seconds.

‘`start_sample`’

Number of the first sample that should be passed to output.

‘`end_sample`’

Number of the first sample that should be dropped.

Note that the first two sets of the start/end options and the ‘`duration`’ option look at the frame timestamp, while the _sample options simply count the samples that pass through the filter. So start/end_pts and start/end_sample will give different results when the timestamps are wrong, inexact or do not start at zero. Also note that this filter does not modify the timestamps. If you wish that the output timestamps start at zero, insert the asetpts filter after the atrim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all samples that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple atrim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- drop everything except the second minute of input

```
ffmpeg -i INPUT -af atrim=60:120
```

- keep only the first 1000 samples

```
ffmpeg -i INPUT -af atrim=end_sample=1000
```

34.31 channelsplit

Split each channel in input audio stream into a separate output stream.

This filter accepts the following named parameters:

‘channel_layout’

Channel layout of the input stream. Default is "stereo".

For example, assuming a stereo input MP3 file

```
ffmpeg -i in.mp3 -filter_complex channelsplit out.mkv
```

will create an output Matroska file with two audio streams, one containing only the left channel and the other the right channel.

To split a 5.1 WAV file into per-channel files

```
ffmpeg -i in.wav -filter_complex
'channelsplit=channel_layout=5.1[FL][FR][FC][LFE][SL][SR]'
-map '[FL]' front_left.wav -map '[FR]' front_right.wav -map '[FC]'
front_center.wav -map '[LFE]' lfe.wav -map '[SL]' side_left.wav -map '[SR]'
side_right.wav
```

34.32 channelmap

Remap input channels to new locations.

This filter accepts the following named parameters:

‘channel_layout’

Channel layout of the output stream.

‘map’

Map channels from input to output. The argument is a '|'-separated list of mappings, each in the *in_channel*-*out_channel* or *in_channel* form. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the input channel layout. *out_channel* is the name of the output channel or its index in the output channel layout. If *out_channel* is not given then it is implicitly an index, starting with zero and increasing by one for each mapping.

If no mapping is present, the filter will implicitly map input channels to output channels preserving index.

For example, assuming a 5.1+downmix input MOV file

```
ffmpeg -i in.mov -filter 'channelmap=map=DL-FL|DR-FR' out.wav
```

will create an output WAV file tagged as stereo from the downmix channels of the input.

To fix a 5.1 WAV improperly encoded in AAC's native channel order

```
ffmpeg -i in.wav -filter 'channelmap=1|2|0|5|3|4:channel_layout=5.1' out.wav
```

34.33 join

Join multiple input streams into one multi-channel stream.

The filter accepts the following named parameters:

'`inputs`'

Number of input streams. Defaults to 2.

'`channel_layout`'

Desired output channel layout. Defaults to stereo.

'`map`'

Map channels from inputs to output. The argument is a '|'-separated list of mappings, each in the *`input_idx.in_channel-out_channel`* form. *input_idx* is the 0-based index of the input stream. *in_channel* can be either the name of the input channel (e.g. FL for front left) or its index in the specified input stream. *out_channel* is the name of the output channel.

The filter will attempt to guess the mappings when those are not specified explicitly. It does so by first trying to find an unused matching input channel and if that fails it picks the first unused input channel.

E.g. to join 3 inputs (with properly set channel layouts)

```
ffmpeg -i INPUT1 -i INPUT2 -i INPUT3 -filter_complex join=inputs=3 OUTPUT
```

To build a 5.1 output from 6 single-channel streams:

```
ffmpeg -i fl -i fr -i fc -i sl -i sr -i lfe -filter_complex
'join=inputs=6:channel_layout=5.1:map=0.0-FL|1.0-FR|2.0-FC|3.0-SL|4.0-SR|5.0-LFE'
out
```

34.34 resample

Convert the audio sample format, sample rate and channel layout. This filter is not meant to be used directly.

34.35 volume

Adjust the input audio volume.

The filter accepts the following options:

‘`volume`’

Expresses how the audio volume will be increased or decreased.

Output values are clipped to the maximum value.

The output audio volume is given by the relation:

```
output_volume = volume * input_volume
```

Default value for *volume* is 1.0.

‘`precision`’

Set the mathematical precision.

This determines which input sample formats will be allowed, which affects the precision of the volume scaling.

‘`fixed`’

8-bit fixed-point; limits input sample format to U8, S16, and S32.

‘`float`’

32-bit floating-point; limits input sample format to FLT. (default)

‘`double`’

64-bit floating-point; limits input sample format to DBL.

34.35.1 Examples

- Halve the input audio volume:

```
volume=volume=0.5
volume=volume=1/2
volume=volume=-6.0206dB
```

In all the above example the named key for ‘`volume`’ can be omitted, for example like in:

```
volume=0.5
```

- Increase input audio power by 6 decibels using fixed-point precision:

```
volume=volume=6dB:precision=fixed
```

34.36 volumedetect

Detect the volume of the input video.

The filter has no parameters. The input is not modified. Statistics about the volume will be printed in the log when the input stream end is reached.

In particular it will show the mean volume (root mean square), maximum volume (on a per-sample basis), and the beginning of an histogram of the registered volume values (from the maximum value to a cumulated 1/1000 of the samples).

All volumes are in decibels relative to the maximum PCM value.

34.36.1 Examples

Here is an excerpt of the output:

```
[Parsed_volumedetect_0  0xa23120] mean_volume: -27 dB
[Parsed_volumedetect_0  0xa23120] max_volume: -4 dB
[Parsed_volumedetect_0  0xa23120] histogram_4db: 6
[Parsed_volumedetect_0  0xa23120] histogram_5db: 62
[Parsed_volumedetect_0  0xa23120] histogram_6db: 286
[Parsed_volumedetect_0  0xa23120] histogram_7db: 1042
[Parsed_volumedetect_0  0xa23120] histogram_8db: 2551
[Parsed_volumedetect_0  0xa23120] histogram_9db: 4609
[Parsed_volumedetect_0  0xa23120] histogram_10db: 8409
```

It means that:

- The mean square energy is approximately -27 dB, or 10^-2.7.
- The largest sample is at -4 dB, or more precisely between -4 dB and -5 dB.
- There are 6 samples at -4 dB, 62 at -5 dB, 286 at -6 dB, etc.

In other words, raising the volume by +4 dB does not cause any clipping, raising it by +5 dB causes clipping for 6 samples, etc.

35. Audio Sources

Below is a description of the currently available audio sources.

35.1 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in ‘`libavfilter/asrc_abuffer.h`’.

It accepts the following named parameters:

‘`time_base`’

Timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

‘`sample_rate`’

The sample rate of the incoming audio buffers.

‘`sample_fmt`’

The sample format of the incoming audio buffers. Either a sample format name or its corresponging integer representation from the enum AVSampleFormat in ‘`libavutil/samplefmt.h`’

‘`channel_layout`’

The channel layout of the incoming audio buffers. Either a channel layout name from channel_layout_map in ‘`libavutil/channel_layout.c`’ or its corresponding integer representation from the AV_CH_LAYOUT_* macros in ‘`libavutil/channel_layout.h`’

‘`channels`’

The number of channels of the incoming audio buffers. If both *channels* and *channel_layout* are specified, then they must be consistent.

35.1.1 Examples

```
abuffer=sample_rate=44100:sample_fmt=s16p:channel_layout=stereo
```

will instruct the source to accept planar 16bit signed stereo at 44100Hz. Since the sample format with name "s16p" corresponds to the number 6 and the "stereo" channel layout corresponds to the value 0x3, this is equivalent to:

```
abuffer=sample_rate=44100:sample_fmt=6:channel_layout=0x3
```

35.2 aevalsrc

Generate an audio signal specified by an expression.

This source accepts in input one or more expressions (one for each channel), which are evaluated and used to generate a corresponding audio signal.

This source accepts the following options:

‘`exprs`’

Set the ’|’-separated expressions list for each separate channel. In case the ‘`channel_layout`’ option is not specified, the selected channel layout depends on the number of provided expressions.

‘`channel_layout, c`’

Set the channel layout. The number of channels in the specified layout must be equal to the number of specified expressions.

‘`duration, d`’

Set the minimum duration of the sourced audio. See the function `av_parse_time()` for the accepted format. Note that the resulting duration may be greater than the specified duration, as the generated audio is always cut at the end of a complete frame.

If not specified, or the expressed duration is negative, the audio is supposed to be generated forever.

‘`nb_samples, n`’

Set the number of samples per channel per each output frame, default to 1024.

‘`sample_rate, s`’

Specify the sample rate, default to 44100.

Each expression in *exprs* can contain the following constants:

‘`n`’

number of the evaluated sample, starting from 0

‘t’

time of the evaluated sample expressed in seconds, starting from 0

‘s’

sample rate

35.2.1 Examples

- Generate silence:

```
aevalsrc=0
```

- Generate a sin signal with frequency of 440 Hz, set sample rate to 8000 Hz:

```
aevalsrc="sin(440*2*PI*t):s=8000"
```

- Generate a two channels signal, specify the channel layout (Front Center + Back Center) explicitly:

```
aevalsrc="sin(420*2*PI*t)|cos(430*2*PI*t):c=FC|BC"
```

- Generate white noise:

```
aevalsrc="-2+random(0)"
```

- Generate an amplitude modulated signal:

```
aevalsrc="sin(10*2*PI*t)*sin(880*2*PI*t)"
```

- Generate 2.5 Hz binaural beats on a 360 Hz carrier:

```
aevalsrc="0.1*sin(2*PI*(360-2.5/2)*t) | 0.1*sin(2*PI*(360+2.5/2)*t)"
```

35.3 anullsrc

Null audio source, return unprocessed audio frames. It is mainly useful as a template and to be employed in analysis / debugging tools, or as the source for filters which ignore the input data (for example the sox synth filter).

This source accepts the following options:

‘`channel_layout, cl`’

Specify the channel layout, and can be either an integer or a string representing a channel layout. The default value of *channel_layout* is "stereo".

Check the channel_layout_map definition in ‘`libavutil/channel_layout.c`’ for the mapping between strings and channel layout values.

‘`sample_rate, r`’

Specify the sample rate, and defaults to 44100.

‘`nb_samples, n`’

Set the number of samples per requested frames.

35.3.1 Examples

- Set the sample rate to 48000 Hz and the channel layout to AV_CH_LAYOUT_MONO.

```
anullsrc=r=48000:cl=4
```

- Do the same operation with a more obvious syntax:

```
anullsrc=r=48000:cl=mono
```

35.4 abuffer

Buffer audio frames, and make them available to the filter chain.

This source is not intended to be part of user-supplied graph descriptions but for insertion by calling programs through the interface defined in ‘`libavfilter/buffersrc.h`’.

It accepts the following named parameters:

‘`time_base`’

Timebase which will be used for timestamps of submitted frames. It must be either a floating-point number or in *numerator*/*denominator* form.

‘`sample_rate`’

Audio sample rate.

‘`sample_fmt`’

Name of the sample format, as returned by `av_get_sample_fmt_name()`.

‘`channel_layout`’

Channel layout of the audio data, in the form that can be accepted by `av_get_channel_layout()`.

All the parameters need to be explicitly defined.

35.5 flite

Synthesize a voice utterance using the libflite library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libflite`.

Note that the flite library is not thread-safe.

The filter accepts the following options:

‘`list_voices`’

If set to 1, list the names of the available voices and exit immediately. Default value is 0.

‘`nb_samples, n`’

Set the maximum number of samples per frame. Default value is 512.

‘`textfile`’

Set the filename containing the text to speak.

‘`text`’

Set the text to speak.

‘`voice, v`’

Set the voice to use for the speech synthesis. Default value is `kal`. See also the *list_voices* option.

35.5.1 Examples

- Read from file ‘`speech.txt`’, and synthetize the text using the standard flite voice:

```
flite=textfile=speech.txt
```

- Read the specified text selecting the `slt` voice:

```
flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
```

- Input text to ffmpeg:

```
ffmpeg -f lavfi -i flite=text='So fare thee well, poor devil of a Sub-Sub, whose commentator I am':voice=slt
```

- Make '`ffplay`' speak the specified text, using `flite` and the `lavfi` device:

```
ffplay -f lavfi flite=text='No more be grieved for which that thou hast done.'
```

For more information about libflite, check: http://www.speech.cs.cmu.edu/flite/

35.6 sine

Generate an audio signal made of a sine wave with amplitude 1/8.

The audio signal is bit-exact.

The filter accepts the following options:

'`frequency, f`'

Set the carrier frequency. Default is 440 Hz.

'`beep_factor, b`'

Enable a periodic beep every second with frequency *beep_factor* times the carrier frequency. Default is 0, meaning the beep is disabled.

'`sample_rate, s`'

Specify the sample rate, default is 44100.

'`duration, d`'

Specify the duration of the generated audio stream.

'`samples_per_frame`'

Set the number of samples per output frame, default is 1024.

35.6.1 Examples

- Generate a simple 440 Hz sine wave:

```
sine
```

- Generate a 220 Hz sine wave with a 880 Hz beep each second, for 5 seconds:

```
sine=220:4:d=5
sine=f=220:b=4:d=5
sine=frequency=220:beep_factor=4:duration=5
```

36. Audio Sinks

Below is a description of the currently available audio sinks.

36.1 abuffersink

Buffer audio frames, and make them available to the end of filter chain.

This sink is mainly intended for programmatic use, in particular through the interface defined in ‘`libavfilter/buffersink.h`’ or the options system.

It accepts a pointer to an AVABufferSinkContext structure, which defines the incoming buffers’ formats, to be passed as the opaque parameter to `avfilter_init_filter` for initialization.

36.2 anullsink

Null audio sink, do absolutely nothing with the input audio. It is mainly useful as a template and to be employed in analysis / debugging tools.

37. Video Filters

When you configure your FFmpeg build, you can disable any of the existing filters using `--disable-filters`. The configure output will show the video filters included in your build.

Below is a description of the currently available video filters.

37.1 alphaextract

Extract the alpha component from the input as a grayscale video. This is especially useful with the *alphamerge* filter.

37.2 alphamerge

Add or replace the alpha component of the primary input with the grayscale value of a second input. This is intended for use with *alphaextract* to allow the transmission or storage of frame sequences that have alpha in a format that doesn’t support an alpha channel.

For example, to reconstruct full frames from a normal YUV-encoded video and a separate video created with *alphaextract*, you might use:

```
movie=in_alpha.mkv [alpha]; [in][alpha] alphamerge [out]
```

Since this filter is designed for reconstruction, it operates on frame sequences without considering timestamps, and terminates when either input reaches end of stream. This will cause problems if your encoding pipeline drops frames. If you're trying to apply an image as an overlay to a video stream, consider the *overlay* filter instead.

37.3 ass

Same as the subtitles filter, except that it doesn't require libavcodec and libavformat to work. On the other hand, it is limited to ASS (Advanced Substation Alpha) subtitles files.

37.4 bbox

Compute the bounding box for the non-black pixels in the input frame luminance plane.

This filter computes the bounding box containing all the pixels with a luminance value greater than the minimum allowed value. The parameters describing the bounding box are printed on the filter log.

37.5 blackdetect

Detect video intervals that are (almost) completely black. Can be useful to detect chapter transitions, commercials, or invalid recordings. Output lines contains the time for the start, end and duration of the detected black interval expressed in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the following options:

'`black_min_duration, d`'

Set the minimum detected black duration expressed in seconds. It must be a non-negative floating point number.

Default value is 2.0.

'`picture_black_ratio_th, pic_th`'

Set the threshold for considering a picture "black". Express the minimum value for the ratio:

```
nb_black_pixels / nb_pixels
```

for which a picture is considered black. Default value is 0.98.

‘pixel_black_th, pix_th’

Set the threshold for considering a pixel "black".

The threshold expresses the maximum pixel luminance value for which a pixel is considered "black". The provided value is scaled according to the following equation:

```
absolute_threshold = luminance_minimum_value + pixel_black_th * luminance_range_size
```

luminance_range_size and *luminance_minimum_value* depend on the input video format, the range is [0-255] for YUV full-range formats and [16-235] for YUV non full-range formats.

Default value is 0.10.

The following example sets the maximum pixel threshold to the minimum value, and detects only black intervals of 2 or more seconds:

```
blackdetect=d=2:pix_th=0.00
```

37.6 blackframe

Detect frames that are (almost) completely black. Can be useful to detect chapter transitions or commercials. Output lines consist of the frame number of the detected frame, the percentage of blackness, the position in the file if known or -1 and the timestamp in seconds.

In order to display the output lines, you need to set the loglevel at least to the AV_LOG_INFO value.

The filter accepts the following options:

‘amount’

Set the percentage of the pixels that have to be below the threshold, defaults to 98.

‘threshold, thresh’

Set the threshold below which a pixel value is considered black, defaults to 32.

37.7 blend

Blend two video frames into each other.

It takes two input streams and outputs one stream, the first input is the "top" layer and second input is "bottom" layer. Output terminates when shortest input terminates.

A description of the accepted options follows.

‘c0_mode’
‘c1_mode’
‘c2_mode’
‘c3_mode’
‘all_mode’

Set blend mode for specific pixel component or all pixel components in case of *all_mode*. Default value is normal.

Available values for component modes are:

‘addition’
‘and’
‘average’
‘burn’
‘darken’
‘difference’
‘divide’
‘dodge’
‘exclusion’
‘hardlight’
‘lighten’
‘multiply’
‘negation’
‘normal’
‘or’
‘overlay’
‘phoenix’
‘pinlight’
‘reflect’
‘screen’
‘softlight’
‘subtract’
‘vividlight’
‘xor’

‘c0_opacity’
‘c1_opacity’
‘c2_opacity’
‘c3_opacity’
‘all_opacity’

Set blend opacity for specific pixel component or all pixel components in case of *all_opacity*. Only used in combination with pixel component blend modes.

‘c0_expr’
‘c1_expr’
‘c2_expr’
‘c3_expr’
‘all_expr’

Set blend expression for specific pixel component or all pixel components in case of *all_expr*. Note that related mode options will be ignored if those are set.

The expressions can use the following variables:

‘N’

The sequential number of the filtered frame, starting from 0.

‘X’
‘Y’

the coordinates of the current sample

‘W’
‘H’

the width and height of currently filtered plane

‘SW’
‘SH’

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are 1,1 for the luma plane, and 0.5,0.5 for chroma planes.

‘T’

Time of the current frame, expressed in seconds.

‘TOP, A’

Value of pixel component at current location for first video frame (top layer).

‘BOTTOM, B’

Value of pixel component at current location for second video frame (bottom layer).

37.7.1 Examples

- Apply transition from bottom layer to top layer in first 10 seconds:

```
blend=all_expr='A*(if(gte(T,10),1,T/10))+B*(1-(if(gte(T,10),1,T/10)))'
```

- Apply 1x1 checkerboard effect:

```
blend=all_expr='if(eq(mod(X,2),mod(Y,2)),A,B)'
```

37.8 boxblur

Apply boxblur algorithm to the input video.

The filter accepts the following options:

‘`luma_radius, lr`’
‘`luma_power, lp`’
‘`chroma_radius, cr`’
‘`chroma_power, cp`’
‘`alpha_radius, ar`’
‘`alpha_power, ap`’

A description of the accepted options follows.

‘`luma_radius, lr`’
‘`chroma_radius, cr`’
‘`alpha_radius, ar`’

Set an expression for the box radius in pixels used for blurring the corresponding input plane.

The radius value must be a non-negative number, and must not be greater than the value of the expression `min(w,h)/2` for the luma and alpha planes, and of `min(cw,ch)/2` for the chroma planes.

Default value for ‘`luma_radius`’ is "2". If not specified, ‘`chroma_radius`’ and ‘`alpha_radius`’ default to the corresponding value set for ‘`luma_radius`’.

The expressions can contain the following constants:

‘`w, h`’

the input width and height in pixels

‘`cw, ch`’

the input chroma image width and height in pixels

‘hsub, vsub’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘luma_power, lp’
‘chroma_power, cp’
‘alpha_power, ap’

Specify how many times the boxblur filter is applied to the corresponding plane.

Default value for ‘luma_power’ is 2. If not specified, ‘chroma_power’ and ‘alpha_power’ default to the corresponding value set for ‘luma_power’.

A value of 0 will disable the effect.

37.8.1 Examples

- Apply a boxblur filter with luma, chroma, and alpha radius set to 2:

```
boxblur=luma_radius=2:luma_power=1
boxblur=2:1
```

- Set luma radius to 2, alpha and chroma radius to 0:

```
boxblur=2:1:cr=0:ar=0
```

- Set luma and chroma radius to a fraction of the video dimension:

```
boxblur=luma_radius=min(h\,w)/10:luma_power=1:chroma_radius=min(cw\,ch)/10:chroma_power=1
```

37.9 colorbalance

Modify intensity of primary colors (red, green and blue) of input frames.

The filter allows an input frame to be adjusted in the shadows, midtones or highlights regions for the red-cyan, green-magenta or blue-yellow balance.

A positive adjustment value shifts the balance towards the primary color, a negative value towards the complementary color.

The filter accepts the following options:

‘rs’
‘gs’
‘bs’

Adjust red, green and blue shadows (darkest pixels).

‘rm’
‘gm’
‘bm’

Adjust red, green and blue midtones (medium pixels).

‘rh’
‘gh’
‘bh’

Adjust red, green and blue highlights (brightest pixels).

Allowed ranges for options are `[-1.0, 1.0]`. Defaults are `0`.

37.9.1 Examples

- Add red color cast to shadows:

```
colorbalance=rs=.3
```

37.10 colorchannelmixer

Adjust video input frames by re-mixing color channels.

This filter modifies a color channel by adding the values associated to the other channels of the same pixels. For example if the value to modify is red, the output value will be:

```
red=red*rr + blue*rb + green*rg + alpha*ra
```

The filter accepts the following options:

‘rr’
‘rg’
‘rb’
‘ra’

Adjust contribution of input red, green, blue and alpha channels for output red channel. Default is `1` for *rr*, and `0` for *rg*, *rb* and *ra*.

‘gr’
‘gg’
‘gb’
‘ga’

Adjust contribution of input red, green, blue and alpha channels for output green channel. Default is `1` for *gg*, and `0` for *gr*, *gb* and *ga*.

‘br’
‘bg’
‘bb’
‘ba’

Adjust contribution of input red, green, blue and alpha channels for output blue channel. Default is `1` for *bb*, and `0` for *br*, *bg* and *ba*.

‘ar’
‘ag’
‘ab’
‘aa’

Adjust contribution of input red, green, blue and alpha channels for output alpha channel. Default is `1` for *aa*, and `0` for *ar*, *ag* and *ab*.

Allowed ranges for options are `[-2.0, 2.0]`.

37.10.1 Examples

- Convert source to grayscale:

```
colorchannelmixer=.3:.4:.3:0:.3:.4:.3:0:.3:.4:.3
```

37.11 colormatrix

Convert color matrix.

The filter accepts the following options:

‘src’
‘dst’

Specify the source and destination color matrix. Both values must be specified.

The accepted values are:

‘bt709’

BT.709

‘bt601’

BT.601

‘smpte240m’

SMPTE-240M

‘fcc’

FCC

For example to convert from BT.601 to SMPTE-240M, use the command:

```
colormatrix=bt601:smpte240m
```

37.12 copy

Copy the input source unchanged to the output. Mainly useful for testing purposes.

37.13 crop

Crop the input video to given dimensions.

The filter accepts the following options:

‘w, out_w’

Width of the output video. It defaults to iw. This expression is evaluated only once during the filter configuration.

‘h, out_h’

Height of the output video. It defaults to ih. This expression is evaluated only once during the filter configuration.

‘x’

Horizontal position, in the input video, of the left edge of the output video. It defaults to (in_w-out_w)/2. This expression is evaluated per-frame.

‘y’

Vertical position, in the input video, of the top edge of the output video. It defaults to `(in_h-out_h)/2`. This expression is evaluated per-frame.

‘`keep_aspect`’

If set to 1 will force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio. It defaults to 0.

The *out_w*, *out_h*, *x*, *y* parameters are expressions containing the following constants:

‘`x, y`’

the computed values for *x* and *y*. They are evaluated for each new frame.

‘`in_w, in_h`’

the input width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘`a`’

same as *iw* / *ih*

‘`sar`’

input sample aspect ratio

‘`dar`’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘`hsub, vsub`’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘`n`’

the number of input frame, starting from 0

‘pos’

the position in the file of the input frame, NAN if unknown

‘t’

timestamp expressed in seconds, NAN if the input timestamp is unknown

The expression for *out_w* may depend on the value of *out_h*, and the expression for *out_h* may depend on *out_w*, but they cannot depend on *x* and *y*, as *x* and *y* are evaluated after *out_w* and *out_h*.

The *x* and *y* parameters specify the expressions for the position of the top-left corner of the output (non-cropped) area. They are evaluated for each frame. If the evaluated value is not valid, it is approximated to the nearest valid value.

The expression for *x* may depend on *y*, and the expression for *y* may depend on *x*.

37.13.1 Examples

- Crop area with size 100x100 at position (12,34).

  ```
  crop=100:100:12:34
  ```

  Using named options, the example above becomes:

  ```
  crop=w=100:h=100:x=12:y=34
  ```

- Crop the central input area with size 100x100:

  ```
  crop=100:100
  ```

- Crop the central input area with size 2/3 of the input video:

  ```
  crop=2/3*in_w:2/3*in_h
  ```

- Crop the input video central square:

  ```
  crop=out_w=in_h
  crop=in_h
  ```

- Delimit the rectangle with the top-left corner placed at position 100:100 and the right-bottom corner corresponding to the right-bottom corner of the input image:

```
crop=in_w-100:in_h-100:100:100
```

- Crop 10 pixels from the left and right borders, and 20 pixels from the top and bottom borders

```
crop=in_w-2*10:in_h-2*20
```

- Keep only the bottom right quarter of the input image:

```
crop=in_w/2:in_h/2:in_w/2:in_h/2
```

- Crop height for getting Greek harmony:

```
crop=in_w:1/PHI*in_w
```

- Appply trembling effect:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(n/10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(n/7)
```

- Apply erratic camera effect depending on timestamp:

```
crop=in_w/2:in_h/2:(in_w-out_w)/2+((in_w-out_w)/2)*sin(t*10):(in_h-out_h)/2 +((in_h-out_h)/2)*sin(t*13)"
```

- Set x depending on the value of y:

```
crop=in_w/2:in_h/2:y:10+10*sin(n/10)
```

37.14 cropdetect

Auto-detect crop size.

Calculate necessary cropping parameters and prints the recommended parameters through the logging system. The detected dimensions correspond to the non-black area of the input video.

The filter accepts the following options:

‘`limit`’

Set higher black value threshold, which can be optionally specified from nothing (0) to everything (255). An intensity value greater to the set value is considered non-black. Default value is 24.

‘`round`’

Set the value for which the width/height should be divisible by. The offset is automatically adjusted to center the video. Use 2 to get only even dimensions (needed for 4:2:2 video). 16 is best when encoding to most video codecs. Default value is 16.

‘reset_count, reset’

Set the counter that determines after how many frames cropdetect will reset the previously detected largest video area and start over to detect the current optimal crop area. Default value is 0.

This can be useful when channel logos distort the video area. 0 indicates never reset and return the largest area encountered during playback.

37.15 curves

Apply color adjustments using curves.

This filter is similar to the Adobe Photoshop and GIMP curves tools. Each component (red, green and blue) has its values defined by *N* key points tied from each other using a smooth curve. The x-axis represents the pixel values from the input frame, and the y-axis the new pixel values to be set for the output frame.

By default, a component curve is defined by the two points *(0;0)* and *(1;1)*. This creates a straight line where each original pixel value is "adjusted" to its own value, which means no change to the image.

The filter allows you to redefine these two points and add some more. A new curve (using a natural cubic spline interpolation) will be define to pass smoothly through all these new coordinates. The new defined points needs to be strictly increasing over the x-axis, and their *x* and *y* values must be in the *[0;1]* interval. If the computed curves happened to go outside the vector spaces, the values will be clipped accordingly.

If there is no key point defined in `x=0`, the filter will automatically insert a *(0;0)* point. In the same way, if there is no key point defined in `x=1`, the filter will automatically insert a *(1;1)* point.

The filter accepts the following options:

‘preset’

Select one of the available color presets. This option can be used in addition to the ‘r’, ‘g’, ‘b’ parameters; in this case, the later options takes priority on the preset values. Available presets are:

‘none’
‘color_negative’
‘cross_process’
‘darker’
‘increase_contrast’
‘lighter’
‘linear_contrast’

‘medium_contrast’
‘negative’
‘strong_contrast’
‘vintage’

Default is none.

‘master, m’

Set the master key points. These points will define a second pass mapping. It is sometimes called a "luminance" or "value" mapping. It can be used with ‘r’, ‘g’, ‘b’ or ‘all’ since it acts like a post-processing LUT.

‘red, r’

Set the key points for the red component.

‘green, g’

Set the key points for the green component.

‘blue, b’

Set the key points for the blue component.

‘all’

Set the key points for all components (not including master). Can be used in addition to the other key points component options. In this case, the unset component(s) will fallback on this ‘all’ setting.

‘psfile’

Specify a Photoshop curves file (.asv) to import the settings from.

To avoid some filtergraph syntax conflicts, each key points list need to be defined using the following syntax: x0/y0 x1/y1 x2/y2 ....

37.15.1 Examples

- Increase slightly the middle level of blue:

```
curves=blue='0.5/0.58'
```

- Vintage effect:

```
curves=r='0/0.11 .42/.51 1/0.95':g='0.50/0.48':b='0/0.22 .49/.44 1/0.8'
```

Here we obtain the following coordinates for each components:

red

```
(0;0.11) (0.42;0.51) (1;0.95)
```

green

```
(0;0) (0.50;0.48) (1;1)
```

blue

```
(0;0.22) (0.49;0.44) (1;0.80)
```

- The previous example can also be achieved with the associated built-in preset:

```
curves=preset=vintage
```

- Or simply:

```
curves=vintage
```

- Use a Photoshop preset and redefine the points of the green component:

```
curves=psfile='MyCurvesPresets/purple.asv':green='0.45/0.53'
```

37.16 decimate

Drop duplicated frames at regular intervals.

The filter accepts the following options:

'`cycle`'

Set the number of frames from which one will be dropped. Setting this to *N* means one frame in every batch of *N* frames will be dropped. Default is `5`.

'`dupthresh`'

Set the threshold for duplicate detection. If the difference metric for a frame is less than or equal to this value, then it is declared as duplicate. Default is `1.1`

'`scthresh`'

Set scene change threshold. Default is `15`.

‘`blockx`’
‘`blocky`’

Set the size of the x and y-axis blocks used during metric calculations. Larger blocks give better noise suppression, but also give worse detection of small movements. Must be a power of two. Default is `32`.

‘`ppsrc`’

Mark main input as a pre-processed input and activate clean source input stream. This allows the input to be pre-processed with various filters to help the metrics calculation while keeping the frame selection lossless. When set to `1`, the first stream is for the pre-processed input, and the second stream is the clean source from where the kept frames are chosen. Default is `0`.

‘`chroma`’

Set whether or not chroma is considered in the metric calculations. Default is `1`.

37.17 delogo

Suppress a TV station logo by a simple interpolation of the surrounding pixels. Just set a rectangle covering the logo and watch it disappear (and sometimes something even uglier appear - your mileage may vary).

This filter accepts the following options:

‘`x, y`’

Specify the top left corner coordinates of the logo. They must be specified.

‘`w, h`’

Specify the width and height of the logo to clear. They must be specified.

‘`band, t`’

Specify the thickness of the fuzzy edge of the rectangle (added to *w* and *h*). The default value is 4.

‘`show`’

When set to 1, a green rectangle is drawn on the screen to simplify finding the right *x*, *y*, *w*, *h* parameters, and *band* is set to 4. The default value is 0.

37.17.1 Examples

- Set a rectangle covering the area with top left corner coordinates 0,0 and size 100x77, setting a band of size 10:

```
delogo=x=0:y=0:w=100:h=77:band=10
```

37.18 deshake

Attempt to fix small changes in horizontal and/or vertical shift. This filter helps remove camera shake from hand-holding a camera, bumping a tripod, moving on a vehicle, etc.

The filter accepts the following options:

‘x’
‘y’
‘w’
‘h’

Specify a rectangular area where to limit the search for motion vectors. If desired the search for motion vectors can be limited to a rectangular area of the frame defined by its top left corner, width and height. These parameters have the same meaning as the drawbox filter which can be used to visualise the position of the bounding box.

This is useful when simultaneous movement of subjects within the frame might be confused for camera motion by the motion vector search.

If any or all of *x*, *y*, *w* and *h* are set to -1 then the full frame is used. This allows later options to be set without specifying the bounding box for the motion vector search.

Default - search the whole frame.

‘rx’
‘ry’

Specify the maximum extent of movement in x and y directions in the range 0-64 pixels. Default 16.

‘edge’

Specify how to generate pixels to fill blanks at the edge of the frame. Available values are:

‘blank, 0’

Fill zeroes at blank locations

‘`original, 1`’

Original image at blank locations

‘`clamp, 2`’

Extruded edge value at blank locations

‘`mirror, 3`’

Mirrored edge at blank locations

Default value is ‘`mirror`’.

‘`blocksize`’

Specify the blocksize to use for motion search. Range 4-128 pixels, default 8.

‘`contrast`’

Specify the contrast threshold for blocks. Only blocks with more than the specified contrast (difference between darkest and lightest pixels) will be considered. Range 1-255, default 125.

‘`search`’

Specify the search strategy. Available values are:

‘`exhaustive, 0`’

Set exhaustive search

‘`less, 1`’

Set less exhaustive search.

Default value is ‘`exhaustive`’.

‘`filename`’

If set then a detailed log of the motion search is written to the specified file.

‘`opencl`’

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

37.19 drawbox

Draw a colored box on the input image.

This filter accepts the following options:

‘`x, y`’

Specify the top left corner coordinates of the box. Default to 0.

‘`width, w`’
‘`height, h`’

Specify the width and height of the box, if 0 they are interpreted as the input width and height. Default to 0.

‘`color, c`’

Specify the color of the box to write, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence. If the special value `invert` is used, the box edge color is the same as the video with inverted luma.

‘`thickness, t`’

Set the thickness of the box edge. Default value is `4`.

37.19.1 Examples

- Draw a black box around the edge of the input image:

```
drawbox
```

- Draw a box with color red and an opacity of 50%:

```
drawbox=10:20:200:60:red@0.5
```

The previous example can be specified as:

```
drawbox=x=10:y=20:w=200:h=60:color=red@0.5
```

- Fill the box with pink color:

```
drawbox=x=10:y=10:w=100:h=100:color=pink@0.5:t=max
```

37.20 drawtext

Draw text string or text from specified file on top of video using the libfreetype library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libfreetype`.

37.20.1 Syntax

The description of the accepted parameters follows.

‘`box`’

Used to draw a box around text using background color. Value should be either 1 (enable) or 0 (disable). The default value of *box* is 0.

‘`boxcolor`’

The color to be used for drawing box around text. Either a string (e.g. "yellow") or in 0xRRGGBB[AA] format (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *boxcolor* is "white".

‘`draw`’

Set an expression which specifies if the text should be drawn. If the expression evaluates to 0, the text is not drawn. This is useful for specifying that the text should be drawn only when specific conditions are met.

Default value is "1".

See below for the list of accepted constants and functions.

‘`expansion`’

Select how the *text* is expanded. Can be either `none`, `strftime` (deprecated) or `normal` (default). See the Text expansion section below for details.

‘`fix_bounds`’

If true, check and fix text coords to avoid clipping.

‘`fontcolor`’

The color to be used for drawing fonts. Either a string (e.g. "red") or in 0xRRGGBB[AA] format (e.g. "0xff000033"), possibly followed by an alpha specifier. The default value of *fontcolor* is "black".

‘`fontfile`’

The font file to be used for drawing text. Path must be included. This parameter is mandatory.

‘`fontsize`’

The font size to be used for drawing text. The default value of *fontsize* is 16.

‘`ft_load_flags`’

Flags to be used for loading the fonts.

The flags map the corresponding flags supported by libfreetype, and are a combination of the following values:

default
no_scale
no_hinting
render
no_bitmap
vertical_layout
force_autohint
crop_bitmap
pedantic
ignore_global_advance_width
no_recurse
ignore_transform
monochrome
linear_design
no_autohint

Default value is "render".

For more information consult the documentation for the FT_LOAD_* libfreetype flags.

‘`shadowcolor`’

The color to be used for drawing a shadow behind the drawn text. It can be a color name (e.g. "yellow") or a string in the 0xRRGGBB[AA] form (e.g. "0xff00ff"), possibly followed by an alpha specifier. The default value of *shadowcolor* is "black".

‘`shadowx, shadowy`’

The x and y offsets for the text shadow position with respect to the position of the text. They can be either positive or negative values. Default value for both is "0".

‘`tabsize`’

The size in number of spaces to use for rendering the tab. Default value is 4.

‘timecode’

Set the initial timecode representation in "hh:mm:ss[:;.]ff" format. It can be used with or without text parameter. *timecode_rate* option must be specified.

‘timecode_rate, rate, r’

Set the timecode frame rate (timecode only).

‘text’

The text string to be drawn. The text must be a sequence of UTF-8 encoded characters. This parameter is mandatory if no file is specified with the parameter *textfile*.

‘textfile’

A text file containing text to be drawn. The text must be a sequence of UTF-8 encoded characters.

This parameter is mandatory if no text string is specified with the parameter *text*.

If both *text* and *textfile* are specified, an error is thrown.

‘reload’

If set to 1, the *textfile* will be reloaded before each frame. Be sure to update it atomically, or it may be read partially, or even fail.

‘x, y’

The expressions which specify the offsets where text will be drawn within the video frame. They are relative to the top/left border of the output image.

The default value of *x* and *y* is "0".

See below for the list of accepted constants and functions.

The parameters for *x* and *y* are expressions containing the following constants and functions:

‘dar’

input display aspect ratio, it is the same as (*w* / *h*) * *sar*

‘hsub, vsub’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘line_h, lh’

the height of each text line

‘`main_h, h, H`’

the input height

‘`main_w, w, W`’

the input width

‘`max_glyph_a, ascent`’

the maximum distance from the baseline to the highest/upper grid coordinate used to place a glyph outline point, for all the rendered glyphs. It is a positive value, due to the grid’s orientation with the Y axis upwards.

‘`max_glyph_d, descent`’

the maximum distance from the baseline to the lowest grid coordinate used to place a glyph outline point, for all the rendered glyphs. This is a negative value, due to the grid’s orientation, with the Y axis upwards.

‘`max_glyph_h`’

maximum glyph height, that is the maximum height for all the glyphs contained in the rendered text, it is equivalent to *ascent* - *descent*.

‘`max_glyph_w`’

maximum glyph width, that is the maximum width for all the glyphs contained in the rendered text

‘`n`’

the number of input frame, starting from 0

‘`rand(min, max)`’

return a random number included between *min* and *max*

‘`sar`’

input sample aspect ratio

‘`t`’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘`text_h, th`’

the height of the rendered text

‘text_w, tw’

the width of the rendered text

‘x, y’

the x and y offset coordinates where the text is drawn.

These parameters allow the *x* and *y* expressions to refer each other, so you can for example specify `y=x/dar`.

If libavfilter was built with `--enable-fontconfig`, then ‘`fontfile`’ can be a fontconfig pattern or omitted.

37.20.2 Text expansion

If ‘`expansion`’ is set to `strftime`, the filter recognizes strftime() sequences in the provided text and expands them accordingly. Check the documentation of strftime(). This feature is deprecated.

If ‘`expansion`’ is set to `none`, the text is printed verbatim.

If ‘`expansion`’ is set to `normal` (which is the default), the following expansion mechanism is used.

The backslash character ’\’, followed by any character, always expands to the second character.

Sequence of the form `%{...}` are expanded. The text between the braces is a function name, possibly followed by arguments separated by ’:’. If the arguments contain special characters or delimiters (’:’ or ’}’), they should be escaped.

Note that they probably must also be escaped as the value for the ‘`text`’ option in the filter argument string and as the filter argument in the filtergraph description, and possibly also for the shell, that makes up to four levels of escaping; using a text file avoids these problems.

The following functions are available:

`expr, e`

The expression evaluation result.

It must take one argument specifying the expression to be evaluated, which accepts the same constants and functions as the *x* and *y* values. Note that not all constants should be used, for example the text size is not known when evaluating the expression, so the constants *text_w* and *text_h* will have an undefined value.

`gmtime`

The time at which the filter is running, expressed in UTC. It can accept an argument: a strftime() format string.

`localtime`

The time at which the filter is running, expressed in the local time zone. It can accept an argument: a strftime() format string.

`n, frame_num`

The frame number, starting from 0.

`pict_type`

A 1 character description of the current picture type.

`pts`

The timestamp of the current frame, in seconds, with microsecond accuracy.

37.20.3 Examples

- Draw "Test Text" with font FreeSerif, using the default values for the optional parameters.

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text'"
```

- Draw 'Test Text' with font FreeSerif of size 24 at position x=100 and y=50 (counting from the top-left corner of the screen), text is yellow with a red box around it. Both the text and the box have an opacity of 20%.

```
drawtext="fontfile=/usr/share/fonts/truetype/freefont/FreeSerif.ttf: text='Test Text':\
          x=100: y=50: fontsize=24: fontcolor=yellow@0.2: box=1: boxcolor=red@0.2"
```

  Note that the double quotes are not necessary if spaces are not used within the parameter list.

- Show the text at the center of the video frame:

```
drawtext="fontsize=30:fontfile=FreeSerif.ttf:text='hello world':x=(w-text_w)/2:y=(h-text_h-line_h)/2"
```

- Show a text line sliding from right to left in the last row of the video frame. The file '`LONG_LINE`' is assumed to contain a single line with no newlines.

```
drawtext="fontsize=15:fontfile=FreeSerif.ttf:text=LONG_LINE:y=h-line_h:x=-50*t"
```

- Show the content of file '`CREDITS`' off the bottom of the frame and scroll up.

```
drawtext="fontsize=20:fontfile=FreeSerif.ttf:textfile=CREDITS:y=h-20*t"
```

- Draw a single green letter "g", at the center of the input video. The glyph baseline is placed at half screen height.

```
drawtext="fontsize=60:fontfile=FreeSerif.ttf:fontcolor=green:text=g:x=(w-max_glyph_w)/2:y=h/2-ascent"
```

- Show text for 1 second every 3 seconds:

```
drawtext="fontfile=FreeSerif.ttf:fontcolor=white:x=100:y=x/dar:draw=lt(mod(t\,3)\,1):text='blink'"
```

- Use fontconfig to set the font. Note that the colons need to be escaped.

```
drawtext='fontfile=Linux Libertine O-40\:style=Semibold:text=FFmpeg'
```

- Print the date of a real-time encoding (see strftime(3)):

```
drawtext='fontfile=FreeSans.ttf:text=%{localtime:%a %b %d %Y}'
```

For more information about libfreetype, check: http://www.freetype.org/.

For more information about fontconfig, check:
http://freedesktop.org/software/fontconfig/fontconfig-user.html.

37.21 edgedetect

Detect and draw edges. The filter uses the Canny Edge Detection algorithm.

The filter accepts the following options:

`'low, high'`

Set low and high threshold values used by the Canny thresholding algorithm.

The high threshold selects the "strong" edge pixels, which are then connected through 8-connectivity with the "weak" edge pixels selected by the low threshold.

low and *high* threshold values must be choosen in the range [0,1], and *low* should be lesser or equal to *high*.

Default value for *low* is `20/255`, and default value for *high* is `50/255`.

Example:

```
edgedetect=low=0.1:high=0.4
```

37.22 fade

Apply fade-in/out effect to input video.

This filter accepts the following options:

‘`type, t`’

The effect type – can be either "in" for fade-in, or "out" for a fade-out effect. Default is `in`.

‘`start_frame, s`’

Specify the number of the start frame for starting to apply the fade effect. Default is 0.

‘`nb_frames, n`’

The number of frames for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black. Default is 25.

‘`alpha`’

If set to 1, fade only alpha channel, if one exists on the input. Default value is 0.

‘`start_time, st`’

Specify the timestamp (in seconds) of the frame to start to apply the fade effect. If both start_frame and start_time are specified, the fade will start at whichever comes last. Default is 0.

‘`duration, d`’

The number of seconds for which the fade effect has to last. At the end of the fade-in effect the output video will have the same intensity as the input video, at the end of the fade-out transition the output video will be completely black. If both duration and nb_frames are specified, duration is used. Default is 0.

37.22.1 Examples

- Fade in first 30 frames of video:

  ```
  fade=in:0:30
  ```

  The command above is equivalent to:

```
fade=t=in:s=0:n=30
```

- Fade out last 45 frames of a 200-frame video:

```
fade=out:155:45
fade=type=out:start_frame=155:nb_frames=45
```

- Fade in first 25 frames and fade out last 25 frames of a 1000-frame video:

```
fade=in:0:25, fade=out:975:25
```

- Make first 5 frames black, then fade in from frame 5-24:

```
fade=in:5:20
```

- Fade in alpha over first 25 frames of video:

```
fade=in:0:25:alpha=1
```

- Make first 5.5 seconds black, then fade in for 0.5 seconds:

```
fade=t=in:st=5.5:d=0.5
```

37.23 field

Extract a single field from an interlaced image using stride arithmetic to avoid wasting CPU time. The output frames are marked as non-interlaced.

The filter accepts the following options:

‘`type`’

Specify whether to extract the top (if the value is `0` or `top`) or the bottom field (if the value is `1` or `bottom`).

37.24 fieldmatch

Field matching filter for inverse telecine. It is meant to reconstruct the progressive frames from a telecined stream. The filter does not drop duplicated frames, so to achieve a complete inverse telecine `fieldmatch` needs to be followed by a decimation filter such as decimate in the filtergraph.

The separation of the field matching and the decimation is notably motivated by the possibility of inserting a de-interlacing filter fallback between the two. If the source has mixed telecined and real interlaced content, `fieldmatch` will not be able to match fields for the interlaced parts. But these remaining combed frames will be marked as interlaced, and thus can be de-interlaced by a later filter such as yadif before decimation.

In addition to the various configuration options, `fieldmatch` can take an optional second stream, activated through the ‘`ppsrc`’ option. If enabled, the frames reconstruction will be based on the fields and frames from this second stream. This allows the first input to be pre-processed in order to help the various algorithms of the filter, while keeping the output lossless (assuming the fields are matched properly). Typically, a field-aware denoiser, or brightness/contrast adjustments can help.

Note that this filter uses the same algorithms as TIVTC/TFM (AviSynth project) and VIVTC/VFM (VapourSynth project). The later is a light clone of TFM from which `fieldmatch` is based on. While the semantic and usage are very close, some behaviour and options names can differ.

The filter accepts the following options:

‘`order`’

Specify the assumed field order of the input stream. Available values are:

‘`auto`’

Auto detect parity (use FFmpeg’s internal parity value).

‘`bff`’

Assume bottom field first.

‘`tff`’

Assume top field first.

Note that it is sometimes recommended not to trust the parity announced by the stream.

Default value is *auto*.

‘`mode`’

Set the matching mode or strategy to use. ‘`pc`’ mode is the safest in the sense that it wont risk creating jerkiness due to duplicate frames when possible, but if there are bad edits or blended fields it will end up outputting combed frames when a good match might actually exist. On the other hand, ‘`pcn_ub`’ mode is the most risky in terms of creating jerkiness, but will almost always find a good frame if there is one. The other values are all somewhere in between ‘`pc`’ and ‘`pcn_ub`’ in terms of risking jerkiness and creating duplicate frames versus finding good matches in sections with bad edits, orphaned fields, blended fields, etc.

More details about p/c/n/u/b are available in p/c/n/u/b meaning section.

Available values are:

‘`pc`’

2-way matching (p/c)

‘`pc_n`’

2-way matching, and trying 3rd match if still combed (p/c + n)

‘`pc_u`’

2-way matching, and trying 3rd match (same order) if still combed (p/c + u)

‘`pc_n_ub`’

2-way matching, trying 3rd match if still combed, and trying 4th/5th matches if still combed (p/c + n + u/b)

‘`pcn`’

3-way matching (p/c/n)

‘`pcn_ub`’

3-way matching, and trying 4th/5th matches if all 3 of the original matches are detected as combed (p/c/n + u/b)

The parenthesis at the end indicate the matches that would be used for that mode assuming ‘`order`’=*tff* (and ‘`field`’ on *auto* or *top*).

In terms of speed ‘`pc`’ mode is by far the fastest and ‘`pcn_ub`’ is the slowest.

Default value is *pc_n*.

‘`ppsrc`’

Mark the main input stream as a pre-processed input, and enable the secondary input stream as the clean source to pick the fields from. See the filter introduction for more details. It is similar to the ‘`clip2`’ feature from VFM/TFM.

Default value is `0` (disabled).

‘`field`’

Set the field to match from. It is recommended to set this to the same value as ‘`order`’ unless you experience matching failures with that setting. In certain circumstances changing the field that is used to match from can have a large impact on matching performance. Available values are:

‘auto’

Automatic (same value as ‘order’).

‘bottom’

Match from the bottom field.

‘top’

Match from the top field.

Default value is *auto*.

‘mchroma’

Set whether or not chroma is included during the match comparisons. In most cases it is recommended to leave this enabled. You should set this to 0 only if your clip has bad chroma problems such as heavy rainbowing or other artifacts. Setting this to 0 could also be used to speed things up at the cost of some accuracy.

Default value is 1.

‘y0’
‘y1’

These define an exclusion band which excludes the lines between ‘y0’ and ‘y1’ from being included in the field matching decision. An exclusion band can be used to ignore subtitles, a logo, or other things that may interfere with the matching. ‘y0’ sets the starting scan line and ‘y1’ sets the ending line; all lines in between ‘y0’ and ‘y1’ (including ‘y0’ and ‘y1’) will be ignored. Setting ‘y0’ and ‘y1’ to the same value will disable the feature. ‘y0’ and ‘y1’ defaults to 0.

‘scthresh’

Set the scene change detection threshold as a percentage of maximum change on the luma plane. Good values are in the [8.0, 14.0] range. Scene change detection is only relevant in case ‘combmatch’=*sc*. The range for ‘scthresh’ is [0.0, 100.0].

Default value is 12.0.

‘combmatch’

When ‘combatch’ is not *none*, fieldmatch will take into account the combed scores of matches when deciding what match to use as the final match. Available values are:

‘none’

No final matching based on combed scores.

‘sc’

Combed scores are only used when a scene change is detected.

‘full’

Use combed scores all the time.

Default is *sc*.

‘combdbg’

Force fieldmatch to calculate the combed metrics for certain matches and print them. This setting is known as ‘micout’ in TFM/VFM vocabulary. Available values are:

‘none’

No forced calculation.

‘pcn’

Force p/c/n calculations.

‘pcnub’

Force p/c/n/u/b calculations.

Default value is *none*.

‘cthresh’

This is the area combing threshold used for combed frame detection. This essentially controls how "strong" or "visible" combing must be to be detected. Larger values mean combing must be more visible and smaller values mean combing can be less visible or strong and still be detected. Valid settings are from -1 (every pixel will be detected as combed) to 255 (no pixel will be detected as combed). This is basically a pixel difference value. A good range is [8, 12].

Default value is 9.

‘chroma’

Sets whether or not chroma is considered in the combed frame decision. Only disable this if your source has chroma problems (rainbowing, etc.) that are causing problems for the combed frame detection with chroma enabled. Actually, using ‘chroma’*=0* is usually more reliable, except for the case where there is chroma only combing in the source.

Default value is `0`.

‘`blockx`’
‘`blocky`’

Respectively set the x-axis and y-axis size of the window used during combed frame detection. This has to do with the size of the area in which ‘`combpel`’ pixels are required to be detected as combed for a frame to be declared combed. See the ‘`combpel`’ parameter description for more info. Possible values are any number that is a power of 2 starting at 4 and going up to 512.

Default value is `16`.

‘`combpel`’

The number of combed pixels inside any of the ‘`blocky`’ by ‘`blockx`’ size blocks on the frame for the frame to be detected as combed. While ‘`cthresh`’ controls how "visible" the combing must be, this setting controls "how much" combing there must be in any localized area (a window defined by the ‘`blockx`’ and ‘`blocky`’ settings) on the frame. Minimum value is `0` and maximum is `blocky x blockx` (at which point no frames will ever be detected as combed). This setting is known as ‘`MI`’ in TFM/VFM vocabulary.

Default value is `80`.

37.24.1 p/c/n/u/b meaning

37.24.1.1 p/c/n

We assume the following telecined stream:

```
Top fields:     1 2 2 3 4
Bottom fields:  1 2 3 4 4
```

The numbers correspond to the progressive frame the fields relate to. Here, the first two frames are progressive, the 3rd and 4th are combed, and so on.

When `fieldmatch` is configured to run a matching from bottom (‘`field`’=*bottom*) this is how this input stream get transformed:

```
Input stream:
                T     1 2 2 3 4
                B     1 2 3 4 4   <-- matching reference

Matches:              c c n n c

Output stream:
                T     1 2 3 4 4
                B     1 2 3 4 4
```

As a result of the field matching, we can see that some frames get duplicated. To perform a complete inverse telecine, you need to rely on a decimation filter after this operation. See for instance the decimate filter.

The same operation now matching from top fields (‘`field`’=*top*) looks like this:

```
Input stream:
                T     1 2 2 3 4   <-- matching reference
                B     1 2 3 4 4

Matches:              c c p p c

Output stream:
                T     1 2 2 3 4
                B     1 2 2 3 4
```

In these examples, we can see what *p*, *c* and *n* mean; basically, they refer to the frame and field of the opposite parity:

- *p* matches the field of the opposite parity in the previous frame
- *c* matches the field of the opposite parity in the current frame
- *n* matches the field of the opposite parity in the next frame

37.24.1.2 u/b

The *u* and *b* matching are a bit special in the sense that they match from the opposite parity flag. In the following examples, we assume that we are currently matching the 2nd frame (Top:2, bottom:2). According to the match, a ’x’ is placed above and below each matched fields.

With bottom matching (‘`field`’=*bottom*):

```
Match:           c         p           n          b          u

                 x       x               x        x          x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x         x           x        x              x

Output frames:
                 2          1          2          2          2
                 2          2          2          1          3
```

With top matching (‘`field`’=*top*):

```
Match:           c         p           n          b          u

                 x         x           x        x              x
  Top          1 2 2     1 2 2       1 2 2      1 2 2      1 2 2
  Bottom       1 2 3     1 2 3       1 2 3      1 2 3      1 2 3
                 x       x               x        x          x

Output frames:
                 2          2          2          1          2
                 2          1          3          2          2
```

37.24.2 Examples

Simple IVTC of a top field first telecined stream:

```
fieldmatch=order=tff:combmatch=none, decimate
```

Advanced IVTC, with fallback on yadif for still combed frames:

```
fieldmatch=order=tff:combmatch=full, yadif=deint=interlaced, decimate
```

37.25 fieldorder

Transform the field order of the input video.

This filter accepts the following options:

‘`order`’

Output field order. Valid values are *tff* for top field first or *bff* for bottom field first.

Default value is ‘`tff`’.

Transformation is achieved by shifting the picture content up or down by one line, and filling the remaining line with appropriate picture content. This method is consistent with most broadcast field order converters.

If the input video is not flagged as being interlaced, or it is already flagged as being of the required output field order then this filter does not alter the incoming video.

This filter is very useful when converting to or from PAL DV material, which is bottom field first.

For example:

```
ffmpeg -i in.vob -vf "fieldorder=bff" out.dv
```

37.26 fifo

Buffer input images and send them when they are requested.

This filter is mainly useful when auto-inserted by the libavfilter framework.

The filter does not take parameters.

37.27 format

Convert the input video to one of the specified pixel formats. Libavfilter will try to pick one that is supported for the input to the next filter.

This filter accepts the following parameters:

‘`pix_fmts`’

A ’|’-separated list of pixel format names, for example "pix_fmts=yuv420p|monow|rgb24".

37.27.1 Examples

- Convert the input video to the format *yuv420p*

  ```
  format=pix_fmts=yuv420p
  ```

  Convert the input video to any of the formats in the list

  ```
  format=pix_fmts=yuv420p|yuv444p|yuv410p
  ```

37.28 fps

Convert the video to specified constant frame rate by duplicating or dropping frames as necessary.

This filter accepts the following named parameters:

‘`fps`’

Desired output frame rate. The default is `25`.

‘`round`’

Rounding method.

Possible values are:

‘zero’

zero round towards 0

‘inf’

round away from 0

‘down’

round towards -infinity

‘up’

round towards +infinity

‘near’

round to nearest

The default is `near`.

Alternatively, the options can be specified as a flat string: *fps*[:*round*].

See also the setpts filter.

37.29 framestep

Select one frame every N-th frame.

This filter accepts the following option:

‘step’

Select frame after every `step` frames. Allowed values are positive integers higher than 0. Default value is `1`.

37.30 frei0r

Apply a frei0r effect to the input video.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

This filter accepts the following options:

‘filter_name’

The name to the frei0r effect to load. If the environment variable FREI0R_PATH is defined, the frei0r effect is searched in each one of the directories specified by the colon separated list in FREIOR_PATH, otherwise in the standard frei0r paths, which are in this order: ‘HOME/.frei0r-1/lib/’, ‘/usr/local/lib/frei0r-1/’, ‘/usr/lib/frei0r-1/’.

‘filter_params’

A ’|’-separated list of parameters to pass to the frei0r effect.

A frei0r effect parameter can be a boolean (whose values are specified with "y" and "n"), a double, a color (specified by the syntax *R*/*G*/*B*, *R*, *G*, and *B* being float numbers from 0.0 to 1.0) or by an av_parse_color() color description), a position (specified by the syntax *X*/*Y*, *X* and *Y* being float numbers) and a string.

The number and kind of parameters depend on the loaded effect. If an effect parameter is not specified the default value is set.

37.30.1 Examples

- Apply the distort0r effect, set the first two double parameters:

```
frei0r=filter_name=distort0r:filter_params=0.5|0.01
```

- Apply the colordistance effect, take a color as first parameter:

```
frei0r=colordistance:0.2/0.3/0.4
frei0r=colordistance:violet
frei0r=colordistance:0x112233
```

- Apply the perspective effect, specify the top left and top right image positions:

```
frei0r=perspective:0.2/0.2|0.8/0.2
```

For more information see: http://frei0r.dyne.org

37.31 geq

The filter accepts the following options:

‘lum_expr’

the luminance expression

‘cb_expr’

the chrominance blue expression

‘cr_expr’

the chrominance red expression

‘alpha_expr’

the alpha expression

‘r’

the red expression

‘g’

the green expression

‘b’

the blue expression

If one of the chrominance expression is not defined, it falls back on the other one. If no alpha expression is specified it will evaluate to opaque value. If none of chrominance expressions are specified, they will evaluate the luminance expression.

The expressions can use the following variables and functions:

‘N’

The sequential number of the filtered frame, starting from 0.

‘X’
‘Y’

The coordinates of the current sample.

‘W’
‘H’

The width and height of the image.

‘SW’
‘SH’

Width and height scale depending on the currently filtered plane. It is the ratio between the corresponding luma plane number of pixels and the current plane ones. E.g. for YUV4:2:0 the values are `1,1` for the luma plane, and `0.5,0.5` for chroma planes.

‘`T`’

Time of the current frame, expressed in seconds.

‘`p(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the current plane.

‘`lum(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the luminance plane.

‘`cb(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the blue-difference chroma plane. Returns 0 if there is no such plane.

‘`cr(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the red-difference chroma plane. Returns 0 if there is no such plane.

‘`alpha(x, y)`’

Return the value of the pixel at location (*x*,*y*) of the alpha plane. Returns 0 if there is no such plane.

For functions, if *x* and *y* are outside the area, the value will be automatically clipped to the closer edge.

37.31.1 Examples

- Flip the image horizontally:

```
geq=p(W-X\,Y)
```

- Generate a bidimensional sine wave, with angle `PI/3` and a wavelength of 100 pixels:

```
geq=128 + 100*sin(2*(PI/100)*(cos(PI/3)*(X-50*T) + sin(PI/3)*Y)):128:128
```

- Generate a fancy enigmatic moving light:

```
nullsrc=s=256x256,geq=random(1)/hypot(X-cos(N*0.07)*W/2-W/2\,Y-sin(N*0.09)*H/2-H/2)^2*1000000*sin(N*0.02):128:128
```

- Generate a quick emboss effect:

```
format=gray,geq=lum_expr='(p(X,Y)+(256-p(X-4,Y-4)))/2'
```

37.32 gradfun

Fix the banding artifacts that are sometimes introduced into nearly flat regions by truncation to 8bit color depth. Interpolate the gradients that should go where the bands are, and dither them.

This filter is designed for playback only. Do not use it prior to lossy compression, because compression tends to lose the dither and bring back the bands.

This filter accepts the following options:

`'strength'`

The maximum amount by which the filter will change any one pixel. Also the threshold for detecting nearly flat regions. Acceptable values range from .51 to 64, default value is 1.2, out-of-range values will be clipped to the valid range.

`'radius'`

The neighborhood to fit the gradient to. A larger radius makes for smoother gradients, but also prevents the filter from modifying the pixels near detailed regions. Acceptable values are 8-32, default value is 16, out-of-range values will be clipped to the valid range.

Alternatively, the options can be specified as a flat string: *strength*[:*radius*]

37.32.1 Examples

- Apply the filter with a `3.5` strength and radius of `8`:

```
gradfun=3.5:8
```

- Specify radius, omitting the strength (which will fall-back to the default value):

```
gradfun=radius=8
```

37.33 hflip

Flip the input video horizontally.

For example to horizontally flip the input video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "hflip" out.avi
```

37.34 histeq

This filter applies a global color histogram equalization on a per-frame basis.

It can be used to correct video that has a compressed range of pixel intensities. The filter redistributes the pixel intensities to equalize their distribution across the intensity range. It may be viewed as an "automatically adjusting contrast filter". This filter is useful only for correcting degraded or poorly captured source video.

The filter accepts the following options:

‘`strength`’

Determine the amount of equalization to be applied. As the strength is reduced, the distribution of pixel intensities more-and-more approaches that of the input frame. The value must be a float number in the range [0,1] and defaults to 0.200.

‘`intensity`’

Set the maximum intensity that can generated and scale the output values appropriately. The strength should be set as desired and then the intensity can be limited if needed to avoid washing-out. The value must be a float number in the range [0,1] and defaults to 0.210.

‘`antibanding`’

Set the antibanding level. If enabled the filter will randomly vary the luminance of output pixels by a small amount to avoid banding of the histogram. Possible values are `none`, `weak` or `strong`. It defaults to `none`.

37.35 histogram

Compute and draw a color distribution histogram for the input video.

The computed histogram is a representation of distribution of color components in an image.

The filter accepts the following options:

‘`mode`’

Set histogram mode.

It accepts the following values:

‘levels’

standard histogram that display color components distribution in an image. Displays color graph for each color component. Shows distribution of the Y, U, V, A or G, B, R components, depending on input format, in current frame. Bellow each graph is color component scale meter.

‘color’

chroma values in vectorscope, if brighter more such chroma values are distributed in an image. Displays chroma values (U/V color placement) in two dimensional graph (which is called a vectorscope). It can be used to read of the hue and saturation of the current frame. At a same time it is a histogram. The whiter a pixel in the vectorscope, the more pixels of the input frame correspond to that pixel (that is the more pixels have this chroma value). The V component is displayed on the horizontal (X) axis, with the leftmost side being V = 0 and the rightmost side being V = 255. The U component is displayed on the vertical (Y) axis, with the top representing U = 0 and the bottom representing U = 255.

The position of a white pixel in the graph corresponds to the chroma value of a pixel of the input clip. So the graph can be used to read of the hue (color flavor) and the saturation (the dominance of the hue in the color). As the hue of a color changes, it moves around the square. At the center of the square, the saturation is zero, which means that the corresponding pixel has no color. If you increase the amount of a specific color, while leaving the other colors unchanged, the saturation increases, and you move towards the edge of the square.

‘color2’

chroma values in vectorscope, similar as `color` but actual chroma values are displayed.

‘waveform’

per row/column color component graph. In row mode graph in the left side represents color component value 0 and right side represents value = 255. In column mode top side represents color component value = 0 and bottom side represents value = 255.

Default value is `levels`.

‘level_height’

Set height of level in `levels`. Default value is `200`. Allowed range is [50, 2048].

‘scale_height’

Set height of color scale in `levels`. Default value is `12`. Allowed range is [0, 40].

‘step’

Set step for `waveform` mode. Smaller values are useful to find out how much of same luminance values across input rows/columns are distributed. Default value is `10`. Allowed range is [1, 255].

‘waveform_mode’

Set mode for waveform. Can be either row, or column. Default is row.

‘display_mode’

Set display mode for waveform and levels. It accepts the following values:

‘parade’

Display separate graph for the color components side by side in row waveform mode or one below other in column waveform mode for waveform histogram mode. For levels histogram mode per color component graphs are placed one bellow other.

This display mode in waveform histogram mode makes it easy to spot color casts in the highlights and shadows of an image, by comparing the contours of the top and the bottom of each waveform. Since whites, grays, and blacks are characterized by exactly equal amounts of red, green, and blue, neutral areas of the picture should display three waveforms of roughly equal width/height. If not, the correction is easy to make by making adjustments to level the three waveforms.

‘overlay’

Presents information that’s identical to that in the parade, except that the graphs representing color components are superimposed directly over one another.

This display mode in waveform histogram mode can make it easier to spot the relative differences or similarities in overlapping areas of the color components that are supposed to be identical, such as neutral whites, grays, or blacks.

Default is parade.

37.35.1 Examples

- Calculate and draw histogram:

```
ffplay -i input -vf histogram
```

37.36 hqdn3d

High precision/quality 3d denoise filter. This filter aims to reduce image noise producing smooth images and making still images really still. It should enhance compressibility.

It accepts the following optional parameters:

‘luma_spatial’

a non-negative float number which specifies spatial luma strength, defaults to 4.0

‘chroma_spatial’

a non-negative float number which specifies spatial chroma strength, defaults to 3.0**luma_spatial*/4.0

‘luma_tmp’

a float number which specifies luma temporal strength, defaults to 6.0**luma_spatial*/4.0

‘chroma_tmp’

a float number which specifies chroma temporal strength, defaults to *luma_tmp***chroma_spatial*/*luma_spatial*

37.37 hue

Modify the hue and/or the saturation of the input.

This filter accepts the following options:

‘h’

Specify the hue angle as a number of degrees. It accepts an expression, and defaults to "0".

‘s’

Specify the saturation in the [-10,10] range. It accepts an expression and defaults to "1".

‘H’

Specify the hue angle as a number of radians. It accepts an expression, and defaults to "0".

‘h’ and ‘H’ are mutually exclusive, and can’t be specified at the same time.

The ‘h’, ‘H’ and ‘s’ option values are expressions containing the following constants:

‘n’

frame count of the input frame starting from 0

‘pts’

presentation timestamp of the input frame expressed in time base units

‘r’

frame rate of the input video, NAN if the input frame rate is unknown

‘t’

timestamp expressed in seconds, NAN if the input timestamp is unknown

‘tb’

time base of the input video

37.37.1 Examples

- Set the hue to 90 degrees and the saturation to 1.0:

  ```
  hue=h=90:s=1
  ```

- Same command but expressing the hue in radians:

  ```
  hue=H=PI/2:s=1
  ```

- Rotate hue and make the saturation swing between 0 and 2 over a period of 1 second:

  ```
  hue="H=2*PI*t: s=sin(2*PI*t)+1"
  ```

- Apply a 3 seconds saturation fade-in effect starting at 0:

  ```
  hue="s=min(t/3\,1)"
  ```

  The general fade-in expression can be written as:

  ```
  hue="s=min(0\, max((t-START)/DURATION\, 1))"
  ```

- Apply a 3 seconds saturation fade-out effect starting at 5 seconds:

  ```
  hue="s=max(0\, min(1\, (8-t)/3))"
  ```

  The general fade-out expression can be written as:

  ```
  hue="s=max(0\, min(1\, (START+DURATION-t)/DURATION))"
  ```

37.37.2 Commands

This filter supports the following commands:

‘s’
‘h’
‘H’

Modify the hue and/or the saturation of the input video. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

37.38 idet

Detect video interlacing type.

This filter tries to detect if the input is interlaced or progressive, top or bottom field first.

The filter accepts the following options:

‘intl_thres’

Set interlacing threshold.

‘prog_thres’

Set progressive threshold.

37.39 il

Deinterleave or interleave fields.

This filter allows to process interlaced images fields without deinterlacing them. Deinterleaving splits the input frame into 2 fields (so called half pictures). Odd lines are moved to the top half of the output image, even lines to the bottom half. You can process (filter) them independently and then re-interleave them.

The filter accepts the following options:

‘luma_mode, l’
‘chroma_mode, s’
‘alpha_mode, a’

Available values for *luma_mode*, *chroma_mode* and *alpha_mode* are:

‘none’

Do nothing.

‘deinterleave, d’

Deinterleave fields, placing one above the other.

‘interleave, i’

Interleave fields. Reverse the effect of deinterleaving.

Default value is none.

‘luma_swap, ls’
‘chroma_swap, cs’
‘alpha_swap, as’

Swap luma/chroma/alpha fields. Exchange even & odd lines. Default value is 0.

37.40 interlace

Simple interlacing filter from progressive contents. This interleaves upper (or lower) lines from odd frames with lower (or upper) lines from even frames, halving the frame rate and preserving image height.

```
   Original        Original             New Frame
   Frame 'j'      Frame 'j+1'             (tff)
  ==========      ===========       ==================
    Line 0  --------------------->    Frame 'j' Line 0
    Line 1          Line 1  ---->   Frame 'j+1' Line 1
    Line 2 --------------------->    Frame 'j' Line 2
    Line 3          Line 3  ---->   Frame 'j+1' Line 3
     ...             ...                 ...
New Frame + 1 will be generated by Frame 'j+2' and Frame 'j+3' and so on
```

It accepts the following optional parameters:

‘scan’

determines whether the interlaced frame is taken from the even (tff - default) or odd (bff) lines of the progressive frame.

‘lowpass’

Enable (default) or disable the vertical lowpass filter to avoid twitter interlacing and reduce moire patterns.

37.41 kerndeint

Deinterlace input video by applying Donald Graft’s adaptive kernel deinterling. Work on interlaced parts of a video to produce progressive frames.

The description of the accepted parameters follows.

‘thresh’

Set the threshold which affects the filter’s tolerance when determining if a pixel line must be processed. It must be an integer in the range [0,255] and defaults to 10. A value of 0 will result in applying the process on every pixels.

‘map’

Paint pixels exceeding the threshold value to white if set to 1. Default is 0.

‘order’

Set the fields order. Swap fields if set to 1, leave fields alone if 0. Default is 0.

‘sharp’

Enable additional sharpening if set to 1. Default is 0.

‘twoway’

Enable twoway sharpening if set to 1. Default is 0.

37.41.1 Examples

- Apply default values:

```
kerndeint=thresh=10:map=0:order=0:sharp=0:twoway=0
```

- Enable additional sharpening:

```
kerndeint=sharp=1
```

- Paint processed pixels in white:

```
kerndeint=map=1
```

37.42 lut, lutrgb, lutyuv

Compute a look-up table for binding each pixel component input value to an output value, and apply it to input video.

lutyuv applies a lookup table to a YUV input video, *lutrgb* to an RGB input video.

These filters accept the following options:

‘`c0`’

set first pixel component expression

‘`c1`’

set second pixel component expression

‘`c2`’

set third pixel component expression

‘`c3`’

set fourth pixel component expression, corresponds to the alpha component

‘`r`’

set red component expression

‘`g`’

set green component expression

‘`b`’

set blue component expression

‘`a`’

alpha component expression

‘`y`’

set Y/luminance component expression

‘`u`’

set U/Cb component expression

‘`v`’

set V/Cr component expression

Each of them specifies the expression to use for computing the lookup table for the corresponding pixel component values.

The exact component associated to each of the *c** options depends on the format in input.

The *lut* filter requires either YUV or RGB pixel formats in input, *lutrgb* requires RGB pixel formats in input, and *lutyuv* requires YUV.

The expressions can contain the following constants and functions:

‘`w, h`’

the input width and height

‘`val`’

input value for the pixel component

‘`clipval`’

the input value clipped in the *minval-maxval* range

‘`maxval`’

maximum value for the pixel component

‘`minval`’

minimum value for the pixel component

‘`negval`’

the negated value for the pixel component value clipped in the *minval-maxval* range , it corresponds to the expression "maxval-clipval+minval"

‘`clip(val)`’

the computed value in *val* clipped in the *minval-maxval* range

‘`gammaval(gamma)`’

the computed gamma correction value of the pixel component value clipped in the *minval-maxval* range, corresponds to the expression "pow((clipval-minval)/(maxval-minval)\,*gamma*)*(maxval-minval)+minval"

All expressions default to "val".

37.42.1 Examples

- Negate input video:

```
lutrgb="r=maxval+minval-val:g=maxval+minval-val:b=maxval+minval-val"
lutyuv="y=maxval+minval-val:u=maxval+minval-val:v=maxval+minval-val"
```

  The above is the same as:

```
lutrgb="r=negval:g=negval:b=negval"
lutyuv="y=negval:u=negval:v=negval"
```

- Negate luminance:

```
lutyuv=y=negval
```

- Remove chroma components, turns the video into a graytone image:

```
lutyuv="u=128:v=128"
```

- Apply a luma burning effect:

```
lutyuv="y=2*val"
```

- Remove green and blue components:

```
lutrgb="g=0:b=0"
```

- Set a constant alpha channel value on input:

```
format=rgba,lutrgb=a="maxval-minval/2"
```

- Correct luminance gamma by a 0.5 factor:

```
lutyuv=y=gammaval(0.5)
```

- Discard least significant bits of luma:

```
lutyuv=y='bitand(val, 128+64+32)'
```

37.43 mp

Apply an MPlayer filter to the input video.

This filter provides a wrapper around most of the filters of MPlayer/MEncoder.

This wrapper is considered experimental. Some of the wrapped filters may not work properly and we may drop support for them, as they will be implemented natively into FFmpeg. Thus you should avoid depending on them when writing portable scripts.

The filters accepts the parameters: *filter_name*[:=]*filter_params*

filter_name is the name of a supported MPlayer filter, *filter_params* is a string containing the parameters accepted by the named filter.

The list of the currently supported filters follows:

dint
eq2
eq
fil
fspp
ilpack
mcdeint
ow
perspective
phase
pp7
pullup
qp
sab
softpulldown
spp
uspp

The parameter syntax and behavior for the listed filters are the same of the corresponding MPlayer filters. For detailed instructions check the "VIDEO FILTERS" section in the MPlayer manual.

37.43.1 Examples

- Adjust gamma, brightness, contrast:

```
mp=eq2=1.0:2:0.5
```

See also mplayer(1), http://www.mplayerhq.hu/.

37.44 mpdecimate

Drop frames that do not differ greatly from the previous frame in order to reduce frame rate.

The main use of this filter is for very-low-bitrate encoding (e.g. streaming over dialup modem), but it could in theory be used for fixing movies that were inverse-telecined incorrectly.

A description of the accepted options follows.

‘`max`’

Set the maximum number of consecutive frames which can be dropped (if positive), or the minimum interval between dropped frames (if negative). If the value is 0, the frame is dropped unregarding the number of previous sequentially dropped frames.

Default value is 0.

‘`hi`’
‘`lo`’
‘`frac`’

Set the dropping threshold values.

Values for ‘`hi`’ and ‘`lo`’ are for 8x8 pixel blocks and represent actual pixel value differences, so a threshold of 64 corresponds to 1 unit of difference for each pixel, or the same spread out differently over the block.

A frame is a candidate for dropping if no 8x8 blocks differ by more than a threshold of ‘`hi`’, and if no more than ‘`frac`’ blocks (1 meaning the whole image) differ by more than a threshold of ‘`lo`’.

Default value for ‘`hi`’ is 64*12, default value for ‘`lo`’ is 64*5, and default value for ‘`frac`’ is 0.33.

37.45 negate

Negate input video.

This filter accepts an integer in input, if non-zero it negates the alpha component (if available). The default value in input is 0.

37.46 noformat

Force libavfilter not to use any of the specified pixel formats for the input to the next filter.

This filter accepts the following parameters:

‘pix_fmts’

A ’|’-separated list of pixel format names, for example "pix_fmts=yuv420p|monow|rgb24".

37.46.1 Examples

- Force libavfilter to use a format different from *yuv420p* for the input to the vflip filter:

```
noformat=pix_fmts=yuv420p,vflip
```

- Convert the input video to any of the formats not contained in the list:

```
noformat=yuv420p|yuv444p|yuv410p
```

37.47 noise

Add noise on video input frame.

The filter accepts the following options:

‘all_seed’
‘c0_seed’
‘c1_seed’
‘c2_seed’
‘c3_seed’

Set noise seed for specific pixel component or all pixel components in case of *all_seed*. Default value is 123457.

‘all_strength, alls’
‘c0_strength, c0s’
‘c1_strength, c1s’
‘c2_strength, c2s’
‘c3_strength, c3s’

Set noise strength for specific pixel component or all pixel components in case *all_strength*. Default value is 0. Allowed range is [0, 100].

‘all_flags, allf’
‘c0_flags, c0f’
‘c1_flags, c1f’
‘c2_flags, c2f’

‘c3_flags, c3f’

Set pixel component flags or set flags for all components if *all_flags*. Available values for component flags are:

‘a’

averaged temporal noise (smoother)

‘p’

mix random noise with a (semi)regular pattern

‘t’

temporal noise (noise pattern changes between frames)

‘u’

uniform noise (gaussian otherwise)

37.47.1 Examples

Add temporal and uniform noise to input video:

```
noise=alls=20:allf=t+u
```

37.48 null

Pass the video source unchanged to the output.

37.49 ocv

Apply video transform using libopencv.

To enable this filter install libopencv library and headers and configure FFmpeg with `--enable-libopencv`.

This filter accepts the following parameters:

‘filter_name’

The name of the libopencv filter to apply.

‘filter_params’

The parameters to pass to the libopencv filter. If not specified the default values are assumed.

Refer to the official libopencv documentation for more precise information:
http://opencv.willowgarage.com/documentation/c/image_filtering.html

Follows the list of supported libopencv filters.

37.49.1 dilate

Dilate an image by using a specific structuring element. This filter corresponds to the libopencv function `cvDilate`.

It accepts the parameters: *struct_el*|*nb_iterations*.

struct_el represents a structuring element, and has the syntax: *cols*x*rows*+*anchor_x*x*anchor_y*/*shape*

cols and *rows* represent the number of columns and rows of the structuring element, *anchor_x* and *anchor_y* the anchor point, and *shape* the shape for the structuring element, and can be one of the values "rect", "cross", "ellipse", "custom".

If the value for *shape* is "custom", it must be followed by a string of the form "=*filename*". The file with name *filename* is assumed to represent a binary image, with each printable character corresponding to a bright pixel. When a custom *shape* is used, *cols* and *rows* are ignored, the number or columns and rows of the read file are assumed instead.

The default value for *struct_el* is "3x3+0x0/rect".

nb_iterations specifies the number of times the transform is applied to the image, and defaults to 1.

Follow some example:

```
# use the default values
ocv=dilate

# dilate using a structuring element with a 5x5 cross, iterate two times
ocv=filter_name=dilate:filter_params=5x5+2x2/cross|2

# read the shape from the file diamond.shape, iterate two times
# the file diamond.shape may contain a pattern of characters like this:
#   *
#  ***
# *****
#  ***
#   *
# the specified cols and rows are ignored (but not the anchor point coordinates)
ocv=dilate:0x0+2x2/custom=diamond.shape|2
```

37.49.2 erode

Erode an image by using a specific structuring element. This filter corresponds to the libopencv function `cvErode`.

The filter accepts the parameters: *struct_el*:*nb_iterations*, with the same syntax and semantics as the dilate filter.

37.49.3 smooth

Smooth the input video.

The filter takes the following parameters: *type*|*param1*|*param2*|*param3*|*param4*.

type is the type of smooth filter to apply, and can be one of the following values: "blur", "blur_no_scale", "median", "gaussian", "bilateral". The default value is "gaussian".

param1, *param2*, *param3*, and *param4* are parameters whose meanings depend on smooth type. *param1* and *param2* accept integer positive values or 0, *param3* and *param4* accept float values.

The default value for *param1* is 3, the default value for the other parameters is 0.

These parameters correspond to the parameters assigned to the libopencv function `cvSmooth`.

37.50 overlay

Overlay one video on top of another.

It takes two inputs and one output, the first input is the "main" video on which the second input is overlayed.

This filter accepts the following parameters:

A description of the accepted options follows.

‘`x`’
‘`y`’

Set the expression for the x and y coordinates of the overlayed video on the main video. Default value is "0" for both expressions. In case the expression is invalid, it is set to a huge value (meaning that the overlay will not be displayed within the output visible area).

‘`eval`’

Set when the expressions for ‘`x`’, and ‘`y`’ are evaluated.

It accepts the following values:

‘`init`’

only evaluate expressions once during the filter initialization or when a command is processed

‘`frame`’

evaluate expressions for each incoming frame

Default value is ‘`frame`’.

‘`shortest`’

If set to 1, force the output to terminate when the shortest input terminates. Default value is 0.

‘`format`’

Set the format for the output video.

It accepts the following values:

‘`yuv420`’

force YUV420 output

‘`yuv444`’

force YUV444 output

‘`rgb`’

force RGB output

Default value is ‘`yuv420`’.

‘`rgb` *(deprecated)*’

If set to 1, force the filter to accept inputs in the RGB color space. Default value is 0. This option is deprecated, use ‘`format`’ instead.

‘`repeatlast`’

If set to 1, force the filter to draw the last overlay frame over the main input until the end of the stream. A value of 0 disables this behavior, which is enabled by default.

The ‘`x`’, and ‘`y`’ expressions can contain the following parameters.

‘main_w, W’
‘main_h, H’

main input width and height

‘overlay_w, w’
‘overlay_h, h’

overlay input width and height

‘x’
‘y’

the computed values for *x* and *y*. They are evaluated for each new frame.

‘hsub’
‘vsub’

horizontal and vertical chroma subsample values of the output format. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

‘n’

the number of input frame, starting from 0

‘pos’

the position in the file of the input frame, NAN if unknown

‘t’

timestamp expressed in seconds, NAN if the input timestamp is unknown

Note that the *n*, *pos*, *t* variables are available only when evaluation is done *per frame*, and will evaluate to NAN when ‘eval’ is set to ‘init’.

Be aware that frames are taken from each input video in timestamp order, hence, if their initial timestamps differ, it is a a good idea to pass the two inputs through a *setpts=PTS-STARTPTS* filter to have them begin in the same zero timestamp, as it does the example for the *movie* filter.

You can chain together more overlays but you should test the efficiency of such approach.

37.50.1 Commands

This filter supports the following commands:

‘x’

‘y’

Modify the x and y of the overlay input. The command accepts the same syntax of the corresponding option.

If the specified expression is not valid, it is kept at its current value.

37.50.2 Examples

- Draw the overlay at 10 pixels from the bottom right corner of the main video:

```
overlay=main_w-overlay_w-10:main_h-overlay_h-10
```

Using named options the example above becomes:

```
overlay=x=main_w-overlay_w-10:y=main_h-overlay_h-10
```

- Insert a transparent PNG logo in the bottom left corner of the input, using the `ffmpeg` tool with the `-filter_complex` option:

```
ffmpeg -i input -i logo -filter_complex 'overlay=10:main_h-overlay_h-10' output
```

- Insert 2 different transparent PNG logos (second logo on bottom right corner) using the `ffmpeg` tool:

```
ffmpeg -i input -i logo1 -i logo2 -filter_complex 'overlay=x=10:y=H-h-10,overlay=x=W-w-10:y=H-h-10' output
```

- Add a transparent color layer on top of the main video, `WxH` must specify the size of the main input to the overlay filter:

```
color=color=red@.3:size=WxH [over]; [in][over] overlay [out]
```

- Play an original video and a filtered version (here with the deshake filter) side by side using the `ffplay` tool:

```
ffplay input.avi -vf 'split[a][b]; [a]pad=iw*2:ih[src]; [b]deshake[filt]; [src][filt]overlay=w'
```

The above command is the same as:

```
ffplay input.avi -vf 'split[b], pad=iw*2[src], [b]deshake, [src]overlay=w'
```

- Make a sliding overlay appearing from the left to the right top part of the screen starting since time 2:

```
overlay=x='if(gte(t,2), -w+(t-2)*20, NAN)':y=0
```

- Compose output by putting two input videos side to side:

```
ffmpeg -i left.avi -i right.avi -filter_complex "
nullsrc=size=200x100 [background];
[0:v] setpts=PTS-STARTPTS, scale=100x100 [left];
[1:v] setpts=PTS-STARTPTS, scale=100x100 [right];
[background][left]       overlay=shortest=1       [background+left];
[background+left][right] overlay=shortest=1:x=100 [left+right]
"
```

- Chain several overlays in cascade:

```
nullsrc=s=200x200 [bg];
testsrc=s=100x100, split=4 [in0][in1][in2][in3];
[in0] lutrgb=r=0, [bg]   overlay=0:0     [mid0];
[in1] lutrgb=g=0, [mid0] overlay=100:0   [mid1];
[in2] lutrgb=b=0, [mid1] overlay=0:100   [mid2];
[in3] null,       [mid2] overlay=100:100 [out0]
```

37.51 pad

Add paddings to the input image, and place the original input at the given coordinates *x*, *y*.

This filter accepts the following parameters:

‘`width, w`’
‘`height, h`’

Specify an expression for the size of the output image with the paddings added. If the value for *width* or *height* is 0, the corresponding input size is used for the output.

The *width* expression can reference the value set by the *height* expression, and vice versa.

The default value of *width* and *height* is 0.

‘`x`’
‘`y`’

Specify an expression for the offsets where to place the input image in the padded area with respect to the top/left border of the output image.

The *x* expression can reference the value set by the *y* expression, and vice versa.

The default value of *x* and *y* is 0.

‘color’

Specify the color of the padded area, it can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence.

The default value of *color* is "black".

The value for the *width*, *height*, *x*, and *y* options are expressions containing the following constants:

‘in_w, in_h’

the input video width and height

‘iw, ih’

same as *in_w* and *in_h*

‘out_w, out_h’

the output width and height, that is the size of the padded area as specified by the *width* and *height* expressions

‘ow, oh’

same as *out_w* and *out_h*

‘x, y’

x and y offsets as specified by the *x* and *y* expressions, or NAN if not yet specified

‘a’

same as *iw* / *ih*

‘sar’

input sample aspect ratio

‘dar’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘hsub, vsub’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

37.51.1 Examples

- Add paddings with color "violet" to the input video. Output video size is 640x480, the top-left corner of the input video is placed at column 0, row 40:

  ```
  pad=640:480:0:40:violet
  ```

  The example above is equivalent to the following command:

  ```
  pad=width=640:height=480:x=0:y=40:color=violet
  ```

- Pad the input to get an output with dimensions increased by 3/2, and put the input video at the center of the padded area:

  ```
  pad="3/2*iw:3/2*ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a squared output with size equal to the maximum value between the input width and height, and put the input video at the center of the padded area:

  ```
  pad="max(iw\,ih):ow:(ow-iw)/2:(oh-ih)/2"
  ```

- Pad the input to get a final w/h ratio of 16:9:

  ```
  pad="ih*16/9:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- In case of anamorphic video, in order to set the output display aspect correctly, it is necessary to use *sar* in the expression, according to the relation:

  ```
  (ih * X / ih) * sar = output_dar
  X = output_dar / sar
  ```

  Thus the previous example needs to be modified to:

  ```
  pad="ih*16/9/sar:ih:(ow-iw)/2:(oh-ih)/2"
  ```

- Double output size and put the input video in the bottom-right corner of the output padded area:

  ```
  pad="2*iw:2*ih:ow-iw:oh-ih"
  ```

37.52 pixdesctest

Pixel format descriptor test filter, mainly useful for internal testing. The output video should be equal to the input video.

For example:

```
format=monow, pixdesctest
```

can be used to test the monowhite pixel format descriptor definition.

37.53 pp

Enable the specified chain of postprocessing subfilters using libpostproc. This library should be automatically selected with a GPL build (`--enable-gpl`). Subfilters must be separated by '/' and can be disabled by prepending a '-'. Each subfilter and some options have a short and a long name that can be used interchangeably, i.e. dr/dering are the same.

The filters accept the following options:

'`subfilters`'

Set postprocessing subfilters string.

All subfilters share common options to determine their scope:

'`a/autoq`'

Honor the quality commands for this subfilter.

'`c/chrom`'

Do chrominance filtering, too (default).

'`y/nochrom`'

Do luminance filtering only (no chrominance).

'`n/noluma`'

Do chrominance filtering only (no luminance).

These options can be appended after the subfilter name, separated by a '|'.

Available subfilters are:

‘hb/hdeblock[|difference[|flatness]]’

Horizontal deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

‘vb/vdeblock[|difference[|flatness]]’

Vertical deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

‘ha/hadeblock[|difference[|flatness]]’

Accurate horizontal deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

‘va/vadeblock[|difference[|flatness]]’

Accurate vertical deblocking filter

‘difference’

Difference factor where higher values mean more deblocking (default: 32).

‘flatness’

Flatness threshold where lower values mean more deblocking (default: 39).

The horizontal and vertical deblocking filters share the difference and flatness values so you cannot set different horizontal and vertical thresholds.

‘h1/x1hdeblock’

Experimental horizontal deblocking filter

‘v1/x1vdeblock’

Experimental vertical deblocking filter

‘dr/dering’

Deringing filter

‘tn/tmpnoise[|threshold1[|threshold2[|threshold3]]], temporal noise reducer’

‘threshold1’

larger -> stronger filtering

‘threshold2’

larger -> stronger filtering

‘threshold3’

larger -> stronger filtering

‘al/autolevels[:f/fullyrange], automatic brightness / contrast correction’

‘f/fullyrange’

Stretch luminance to 0-255.

‘lb/linblenddeint’

Linear blend deinterlacing filter that deinterlaces the given block by filtering all lines with a (1 2 1) filter.

‘li/linipoldeint’

Linear interpolating deinterlacing filter that deinterlaces the given block by linearly interpolating every second line.

‘ci/cubicipoldeint’

Cubic interpolating deinterlacing filter deinterlaces the given block by cubically interpolating every second line.

‘md/mediandeint’

Median deinterlacing filter that deinterlaces the given block by applying a median filter to every second line.

‘fd/ffmpegdeint’

FFmpeg deinterlacing filter that deinterlaces the given block by filtering every second line with a (-1 4 2 4 -1) filter.

‘l5/lowpass5’

Vertically applied FIR lowpass deinterlacing filter that deinterlaces the given block by filtering all lines with a (-1 2 6 2 -1) filter.

‘fq/forceQuant[|quantizer]’

Overrides the quantizer table from the input with the constant quantizer you specify.

‘quantizer’

Quantizer to use

‘de/default’

Default pp filter combination (hb|a,vb|a,dr|a)

‘fa/fast’

Fast pp filter combination (h1|a,v1|a,dr|a)

‘ac’

High quality pp filter combination (ha|a|128|7,va|a,dr|a)

37.53.1 Examples

- Apply horizontal and vertical deblocking, deringing and automatic brightness/contrast:

```
pp=hb/vb/dr/al
```

- Apply default filters without brightness/contrast correction:

```
pp=de/-al
```

- Apply default filters and temporal denoiser:

```
pp=default/tmpnoise|1|2|3
```

- Apply deblocking on luminance only, and switch vertical deblocking on or off automatically depending on available CPU time:

```
pp=hb|y/vb|a
```

37.54 removelogo

Suppress a TV station logo, using an image file to determine which pixels comprise the logo. It works by filling in the pixels that comprise the logo with neighboring pixels.

The filter accepts the following options:

‘`filename, f`’

Set the filter bitmap file, which can be any image format supported by libavformat. The width and height of the image file must match those of the video stream being processed.

Pixels in the provided bitmap image with a value of zero are not considered part of the logo, non-zero pixels are considered part of the logo. If you use white (255) for the logo and black (0) for the rest, you will be safe. For making the filter bitmap, it is recommended to take a screen capture of a black frame with the logo visible, and then using a threshold filter followed by the erode filter once or twice.

If needed, little splotches can be fixed manually. Remember that if logo pixels are not covered, the filter quality will be much reduced. Marking too many pixels as part of the logo does not hurt as much, but it will increase the amount of blurring needed to cover over the image and will destroy more information than necessary, and extra pixels will slow things down on a large logo.

37.55 scale

Scale (resize) the input video, using the libswscale library.

The scale filter forces the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio.

The filter accepts the following options:

‘`width, w`’

Set the output video width expression. Default value is `iw`. See below for the list of accepted constants.

‘`height, h`’

Set the output video height expression. Default value is `ih`. See below for the list of accepted constants.

‘`interl`’

Set the interlacing. It accepts the following values:

‘`1`’

force interlaced aware scaling

‘`0`’

do not apply interlaced scaling

‘`-1`’

select interlaced aware scaling depending on whether the source frames are flagged as interlaced or not

Default value is `0`.

‘`flags`’

Set libswscale scaling flags. If not explictly specified the filter applies a bilinear scaling algorithm.

‘`size, s`’

Set the video size, the value must be a valid abbreviation or in the form *width*x*height*.

The values of the *w* and *h* options are expressions containing the following constants:

‘`in_w, in_h`’

the input width and height

‘`iw, ih`’

same as *in_w* and *in_h*

‘`out_w, out_h`’

the output (cropped) width and height

‘`ow, oh`’

same as *out_w* and *out_h*

‘a’

same as *iw* / *ih*

‘sar’

input sample aspect ratio

‘dar’

input display aspect ratio, it is the same as (*iw* / *ih*) * *sar*

‘hsub, vsub’

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" *hsub* is 2 and *vsub* is 1.

If the input image format is different from the format requested by the next filter, the scale filter will convert the input to the requested format.

If the value for *w* or *h* is 0, the respective input size is used for the output.

If the value for *w* or *h* is -1, the scale filter will use, for the respective output size, a value that maintains the aspect ratio of the input image.

37.55.1 Examples

- Scale the input video to a size of 200x100:

  ```
  scale=w=200:h=100
  ```

  This is equivalent to:

  ```
  scale=200:100
  ```

  or:

  ```
  scale=200x100
  ```

- Specify a size abbreviation for the output size:

  ```
  scale=qcif
  ```

  which can also be written as:

```
scale=size=qcif
```

- Scale the input to 2x:

```
scale=w=2*iw:h=2*ih
```

- The above is the same as:

```
scale=2*in_w:2*in_h
```

- Scale the input to 2x with forced interlaced scaling:

```
scale=2*iw:2*ih:interl=1
```

- Scale the input to half size:

```
scale=w=iw/2:h=ih/2
```

- Increase the width, and set the height to the same size:

```
scale=3/2*iw:ow
```

- Seek for Greek harmony:

```
scale=iw:1/PHI*iw
scale=ih*PHI:ih
```

- Increase the height, and set the width to 3/2 of the height:

```
scale=w=3/2*oh:h=3/5*ih
```

- Increase the size, but make the size a multiple of the chroma subsample values:

```
scale="trunc(3/2*iw/hsub)*hsub:trunc(3/2*ih/vsub)*vsub"
```

- Increase the width to a maximum of 500 pixels, keep the same input aspect ratio:

```
scale=w='min(500\, iw*3/2):h=-1'
```

37.56 separatefields

The `separatefields` takes a frame-based video input and splits each frame into its components fields, producing a new half height clip with twice the frame rate and twice the frame count.

This filter use field-dominance information in frame to decide which of each pair of fields to place first in the output. If it gets it wrong use setfield filter before `separatefields` filter.

37.57 setdar, setsar

The `setdar` filter sets the Display Aspect Ratio for the filter output video.

This is done by changing the specified Sample (aka Pixel) Aspect Ratio, according to the following equation:

```
DAR = HORIZONTAL_RESOLUTION / VERTICAL_RESOLUTION * SAR
```

Keep in mind that the `setdar` filter does not modify the pixel dimensions of the video frame. Also the display aspect ratio set by this filter may be changed by later filters in the filterchain, e.g. in case of scaling or if another "setdar" or a "setsar" filter is applied.

The `setsar` filter sets the Sample (aka Pixel) Aspect Ratio for the filter output video.

Note that as a consequence of the application of this filter, the output display aspect ratio will change according to the equation above.

Keep in mind that the sample aspect ratio set by the `setsar` filter may be changed by later filters in the filterchain, e.g. if another "setsar" or a "setdar" filter is applied.

The filters accept the following options:

‘`r, ratio, dar (setdar only), sar (setsar only)`’

Set the aspect ratio used by the filter.

The parameter can be a floating point number string, an expression, or a string of the form *num*:*den*, where *num* and *den* are the numerator and denominator of the aspect ratio. If the parameter is not specified, it is assumed the value "0". In case the form "*num*:*den*" is used, the `:` character should be escaped.

‘`max`’

Set the maximum integer value to use for expressing numerator and denominator when reducing the expressed aspect ratio to a rational. Default value is `100`.

37.57.1 Examples

- To change the display aspect ratio to 16:9, specify one of the following:

```
setdar=dar=1.77777
setdar=dar=16/9
setdar=dar=1.77777
```

- To change the sample aspect ratio to 10:11, specify:

```
setsar=sar=10/11
```

- To set a display aspect ratio of 16:9, and specify a maximum integer value of 1000 in the aspect ratio reduction, use the command:

```
setdar=ratio=16/9:max=1000
```

37.58 setfield

Force field for the output video frame.

The `setfield` filter marks the interlace type field for the output frames. It does not change the input frame, but only sets the corresponding property, which affects how the frame is treated by following filters (e.g. `fieldorder` or `yadif`).

The filter accepts the following options:

‘`mode`’

Available values are:

‘`auto`’

Keep the same field property.

‘`bff`’

Mark the frame as bottom-field-first.

‘`tff`’

Mark the frame as top-field-first.

‘`prog`’

Mark the frame as progressive.

37.59 showinfo

Show a line containing various information for each input video frame. The input video is not modified.

The shown line contains a sequence of key/value pairs of the form *key*:*value*.

A description of each shown parameter follows:

‘`n`’

sequential number of the input frame, starting from 0

‘`pts`’

Presentation TimeStamp of the input frame, expressed as a number of time base units. The time base unit depends on the filter input pad.

‘`pts_time`’

Presentation TimeStamp of the input frame, expressed as a number of seconds

‘`pos`’

position of the frame in the input stream, -1 if this information in unavailable and/or meaningless (for example in case of synthetic video)

‘`fmt`’

pixel format name

‘`sar`’

sample aspect ratio of the input frame, expressed in the form *num*/*den*

‘`s`’

size of the input frame, expressed in the form *width*x*height*

‘`i`’

interlaced mode ("P" for "progressive", "T" for top field first, "B" for bottom field first)

‘`iskey`’

1 if the frame is a key frame, 0 otherwise

‘type’

picture type of the input frame ("I" for an I-frame, "P" for a P-frame, "B" for a B-frame, "?" for unknown type). Check also the documentation of the AVPictureType enum and of the av_get_picture_type_char function defined in ‘libavutil/avutil.h’.

‘checksum’

Adler-32 checksum (printed in hexadecimal) of all the planes of the input frame

‘plane_checksum’

Adler-32 checksum (printed in hexadecimal) of each plane of the input frame, expressed in the form "[*c0 c1 c2 c3*]"

37.60 smartblur

Blur the input video without impacting the outlines.

The filter accepts the following options:

‘luma_radius, lr’

Set the luma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

‘luma_strength, ls’

Set the luma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

‘luma_threshold, lt’

Set the luma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

‘chroma_radius, cr’

Set the chroma radius. The option value must be a float number in the range [0.1,5.0] that specifies the variance of the gaussian filter used to blur the image (slower if larger). Default value is 1.0.

‘chroma_strength, cs’

Set the chroma strength. The option value must be a float number in the range [-1.0,1.0] that configures the blurring. A value included in [0.0,1.0] will blur the image whereas a value included in [-1.0,0.0] will sharpen the image. Default value is 1.0.

‘chroma_threshold, ct’

Set the chroma threshold used as a coefficient to determine whether a pixel should be blurred or not. The option value must be an integer in the range [-30,30]. A value of 0 will filter all the image, a value included in [0,30] will filter flat areas and a value included in [-30,0] will filter edges. Default value is 0.

If a chroma option is not explicitly set, the corresponding luma value is set.

37.61 stereo3d

Convert between different stereoscopic image formats.

The filters accept the following options:

‘in’

Set stereoscopic image format of input.

Available values for input image formats are:

‘sbsl’

side by side parallel (left eye left, right eye right)

‘sbsr’

side by side crosseye (right eye left, left eye right)

‘sbs2l’

side by side parallel with half width resolution (left eye left, right eye right)

‘sbs2r’

side by side crosseye with half width resolution (right eye left, left eye right)

‘abl’

above-below (left eye above, right eye below)

‘abr’

above-below (right eye above, left eye below)

‘ab2l’

above-below with half height resolution (left eye above, right eye below)

‘ab2r’

above-below with half height resolution (right eye above, left eye below)

‘al’

alternating frames (left eye first, right eye second)

‘ar’

alternating frames (right eye first, left eye second)

Default value is ‘sbsl’.

‘out’

Set stereoscopic image format of output.

Available values for output image formats are all the input formats as well as:

‘arbg’

anaglyph red/blue gray (red filter on left eye, blue filter on right eye)

‘argg’

anaglyph red/green gray (red filter on left eye, green filter on right eye)

‘arcg’

anaglyph red/cyan gray (red filter on left eye, cyan filter on right eye)

‘arch’

anaglyph red/cyan half colored (red filter on left eye, cyan filter on right eye)

‘arcc’

anaglyph red/cyan color (red filter on left eye, cyan filter on right eye)

‘arcd’

anaglyph red/cyan color optimized with the least squares projection of dubois (red filter on left eye, cyan filter on right eye)

‘`agmg`’

anaglyph green/magenta gray (green filter on left eye, magenta filter on right eye)

‘`agmh`’

anaglyph green/magenta half colored (green filter on left eye, magenta filter on right eye)

‘`agmc`’

anaglyph green/magenta colored (green filter on left eye, magenta filter on right eye)

‘`agmd`’

anaglyph green/magenta color optimized with the least squares projection of dubois (green filter on left eye, magenta filter on right eye)

‘`aybg`’

anaglyph yellow/blue gray (yellow filter on left eye, blue filter on right eye)

‘`aybh`’

anaglyph yellow/blue half colored (yellow filter on left eye, blue filter on right eye)

‘`aybc`’

anaglyph yellow/blue colored (yellow filter on left eye, blue filter on right eye)

‘`aybd`’

anaglyph yellow/blue color optimized with the least squares projection of dubois (yellow filter on left eye, blue filter on right eye)

‘`irl`’

interleaved rows (left eye has top row, right eye starts on next row)

‘`irr`’

interleaved rows (right eye has top row, left eye starts on next row)

‘`ml`’

mono output (left eye only)

‘mr’

mono output (right eye only)

Default value is ‘arcd’.

37.61.1 Examples

- Convert input video from side by side parallel to anaglyph yellow/blue dubois:

```
stereo3d=sbsl:aybd
```

- Convert input video from above bellow (left eye above, right eye below) to side by side crosseye.

```
stereo3d=abl:sbsr
```

37.62 subtitles

Draw subtitles on top of input video using the libass library.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libass`. This filter also requires a build with libavcodec and libavformat to convert the passed subtitles file to ASS (Advanced Substation Alpha) subtitles format.

The filter accepts the following options:

‘filename, f’

Set the filename of the subtitle file to read. It must be specified.

‘original_size’

Specify the size of the original video, the video for which the ASS file was composed. Due to a misdesign in ASS aspect ratio arithmetic, this is necessary to correctly scale the fonts if the aspect ratio has been changed.

‘charenc’

Set subtitles input character encoding. `subtitles` filter only. Only useful if not UTF-8.

If the first key is not specified, it is assumed that the first value specifies the ‘filename’.

For example, to render the file ‘sub.srt’ on top of the input video, use the command:

```
subtitles=sub.srt
```

which is equivalent to:

```
subtitles=filename=sub.srt
```

37.63 super2xsai

Scale the input by 2x and smooth using the Super2xSaI (Scale and Interpolate) pixel art scaling algorithm.

Useful for enlarging pixel art images without reducing sharpness.

37.64 swapuv

Swap U & V plane.

37.65 telecine

Apply telecine process to the video.

This filter accepts the following options:

‘first_field’
‘top, t’

top field first

‘bottom, b’

bottom field first The default value is top.

‘pattern’

A string of numbers representing the pulldown pattern you wish to apply. The default value is 23.

```
Some typical patterns:

NTSC output (30i):
27.5p: 32222
24p: 23 (classic)
24p: 2332 (preferred)
20p: 33
18p: 334
16p: 3444

PAL output (25i):
27.5p: 12222
24p: 222222222223 ("Euro pulldown")
16.67p: 33
16p: 33333334
```

37.66 thumbnail

Select the most representative frame in a given sequence of consecutive frames.

The filter accepts the following options:

‘`n`’

Set the frames batch size to analyze; in a set of *n* frames, the filter will pick one of them, and then handle the next batch of *n* frames until the end. Default is `100`.

Since the filter keeps track of the whole frames sequence, a bigger *n* value will result in a higher memory usage, so a high value is not recommended.

37.66.1 Examples

- Extract one picture each 50 frames:

```
thumbnail=50
```

- Complete example of a thumbnail creation with `ffmpeg`:

```
ffmpeg -i in.avi -vf thumbnail,scale=300:200 -frames:v 1 out.png
```

37.67 tile

Tile several successive frames together.

The filter accepts the following options:

‘`layout`’

Set the grid size (i.e. the number of lines and columns) in the form "*w*x*h*".

‘`nb_frames`’

Set the maximum number of frames to render in the given area. It must be less than or equal to *w*x*h*. The default value is `0`, meaning all the area will be used.

‘`margin`’

Set the outer border margin in pixels.

‘`padding`’

Set the inner border thickness (i.e. the number of pixels between frames). For more advanced padding options (such as having different values for the edges), refer to the pad video filter.

37.67.1 Examples

- Produce 8x8 PNG tiles of all keyframes (‘`-skip_frame nokey`’) in a movie:

```
ffmpeg -skip_frame nokey -i file.avi -vf 'scale=128:72,tile=8x8' -an -vsync 0 keyframes%03d.png
```

  The ‘`-vsync 0`’ is necessary to prevent `ffmpeg` from duplicating each output frame to accomodate the originally detected frame rate.

- Display `5` pictures in an area of `3x2` frames, with `7` pixels between them, and `2` pixels of initial margin, using mixed flat and named options:

```
tile=3x2:nb_frames=5:padding=7:margin=2
```

37.68 tinterlace

Perform various types of temporal field interlacing.

Frames are counted starting from 1, so the first input frame is considered odd.

The filter accepts the following options:

‘`mode`’

Specify the mode of the interlacing. This option can also be specified as a value alone. See below for a list of values for this option.

Available values are:

‘`merge, 0`’

Move odd frames into the upper field, even into the lower field, generating a double height frame at half frame rate.

‘`drop_odd, 1`’

Only output even frames, odd frames are dropped, generating a frame with unchanged height at half frame rate.

‘`drop_even, 2`’

Only output odd frames, even frames are dropped, generating a frame with unchanged height at half frame rate.

‘pad, 3’

Expand each frame to full height, but pad alternate lines with black, generating a frame with double height at the same input frame rate.

‘interleave_top, 4’

Interleave the upper field from odd frames with the lower field from even frames, generating a frame with unchanged height at half frame rate.

‘interleave_bottom, 5’

Interleave the lower field from odd frames with the upper field from even frames, generating a frame with unchanged height at half frame rate.

‘interlacex2, 6’

Double frame rate with unchanged height. Frames are inserted each containing the second temporal field from the previous input frame and the first temporal field from the next input frame. This mode relies on the top_field_first flag. Useful for interlaced video displays with no field synchronisation.

Numeric values are deprecated but are accepted for backward compatibility reasons.

Default mode is merge.

‘flags’

Specify flags influencing the filter process.

Available value for *flags* is:

‘low_pass_filter, vlfp’

Enable vertical low-pass filtering in the filter. Vertical low-pass filtering is required when creating an interlaced destination from a progressive source which contains high-frequency vertical detail. Filtering will reduce interlace ’twitter’ and Moire patterning.

Vertical low-pass filtering can only be enabled for ‘mode’ *interleave_top* and *interleave_bottom*.

37.69 transpose

Transpose rows with columns in the input video and optionally flip it.

This filter accepts the following options:

‘dir’

Specify the transposition direction.

Can assume the following values:

‘0, 4, cclock_flip’

Rotate by 90 degrees counterclockwise and vertically flip (default), that is:

```
L.R     L.l
. . ->  . .
l.r     R.r
```

‘1, 5, clock’

Rotate by 90 degrees clockwise, that is:

```
L.R     l.L
. . ->  . .
l.r     r.R
```

‘2, 6, cclock’

Rotate by 90 degrees counterclockwise, that is:

```
L.R     R.r
. . ->  . .
l.r     L.l
```

‘3, 7, clock_flip’

Rotate by 90 degrees clockwise and vertically flip, that is:

```
L.R     r.R
. . ->  . .
l.r     l.L
```

For values between 4-7, the transposition is only done if the input video geometry is portrait and not landscape. These values are deprecated, the `passthrough` option should be used instead.

Numerical values are deprecated, and should be dropped in favor of symbolic constants.

‘passthrough’

Do not apply the transposition if the input geometry matches the one specified by the specified value. It accepts the following values:

‘none’

Always apply transposition.

‘portrait’

Preserve portrait geometry (when *height* >= *width*).

‘landscape’

Preserve landscape geometry (when *width* >= *height*).

Default value is none.

For example to rotate by 90 degrees clockwise and preserve portrait layout:

```
transpose=dir=1:passthrough=portrait
```

The command above can also be specified as:

```
transpose=1:portrait
```

37.70 trim

Trim the input so that the output contains one continuous subpart of the input.

This filter accepts the following options:

‘start’

Timestamp (in seconds) of the start of the kept section. I.e. the frame with the timestamp *start* will be the first frame in the output.

‘end’

Timestamp (in seconds) of the first frame that will be dropped. I.e. the frame immediately preceding the one with the timestamp *end* will be the last frame in the output.

‘start_pts’

Same as *start*, except this option sets the start timestamp in timebase units instead of seconds.

‘end_pts’

Same as *end*, except this option sets the end timestamp in timebase units instead of seconds.

‘duration’

Maximum duration of the output in seconds.

‘start_frame’

Number of the first frame that should be passed to output.

‘end_frame’

Number of the first frame that should be dropped.

Note that the first two sets of the start/end options and the ‘duration’ option look at the frame timestamp, while the _frame variants simply count the frames that pass through the filter. Also note that this filter does not modify the timestamps. If you wish that the output timestamps start at zero, insert a setpts filter after the trim filter.

If multiple start or end options are set, this filter tries to be greedy and keep all the frames that match at least one of the specified constraints. To keep only the part that matches all the constraints at once, chain multiple trim filters.

The defaults are such that all the input is kept. So it is possible to set e.g. just the end values to keep everything before the specified time.

Examples:

- drop everything except the second minute of input

```
ffmpeg -i INPUT -vf trim=60:120
```

- keep only the first second

```
ffmpeg -i INPUT -vf trim=duration=1
```

37.71 unsharp

Sharpen or blur the input video.

It accepts the following parameters:

‘luma_msize_x, lx’

Set the luma matrix horizontal size. It must be an odd integer between 3 and 63, default value is 5.

‘luma_msize_y, ly’

Set the luma matrix vertical size. It must be an odd integer between 3 and 63, default value is 5.

‘`luma_amount, la`’

Set the luma effect strength. It can be a float number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 1.0.

‘`chroma_msize_x, cx`’

Set the chroma matrix horizontal size. It must be an odd integer between 3 and 63, default value is 5.

‘`chroma_msize_y, cy`’

Set the chroma matrix vertical size. It must be an odd integer between 3 and 63, default value is 5.

‘`chroma_amount, ca`’

Set the chroma effect strength. It can be a float number, reasonable values lay between -1.5 and 1.5.

Negative values will blur the input video, while positive values will sharpen it, a value of zero will disable the effect.

Default value is 0.0.

‘`opencl`’

If set to 1, specify using OpenCL capabilities, only available if FFmpeg was configured with `--enable-opencl`. Default value is 0.

All parameters are optional and default to the equivalent of the string ’5:5:1.0:5:5:0.0’.

37.71.1 Examples

- Apply strong luma sharpen effect:

```
unsharp=luma_msize_x=7:luma_msize_y=7:luma_amount=2.5
```

- Apply strong blur of both luma and chroma parameters:

```
unsharp=7:7:-2:7:7:-2
```

37.72 vidstabdetect

Analyze video stabilization/deshaking. Perform pass 1 of 2, see vidstabtransform for pass 2.

This filter generates a file with relative translation and rotation transform information about subsequent frames, which is then used by the vidstabtransform filter.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

This filter accepts the following options:

‘`result`’

Set the path to the file used to write the transforms information. Default value is ‘`transforms.trf`’.

‘`shakiness`’

Set how shaky the video is and how quick the camera is. It accepts an integer in the range 1-10, a value of 1 means little shakiness, a value of 10 means strong shakiness. Default value is 5.

‘`accuracy`’

Set the accuracy of the detection process. It must be a value in the range 1-15. A value of 1 means low accuracy, a value of 15 means high accuracy. Default value is 9.

‘`stepsize`’

Set stepsize of the search process. The region around minimum is scanned with 1 pixel resolution. Default value is 6.

‘`mincontrast`’

Set minimum contrast. Below this value a local measurement field is discarded. Must be a floating point value in the range 0-1. Default value is 0.3.

‘`tripod`’

Set reference frame number for tripod mode.

If enabled, the motion of the frames is compared to a reference frame in the filtered stream, identified by the specified number. The idea is to compensate all movements in a more-or-less static scene and keep the camera view absolutely still.

If set to 0, it is disabled. The frames are counted starting from 1.

‘`show`’

Show fields and transforms in the resulting frames. It accepts an integer in the range 0-2. Default value is 0, which disables any visualization.

37.72.1 Examples

- Use default values:

```
vidstabdetect
```

- Analyze strongly shaky movie and put the results in file ‘`mytransforms.trf`’:

```
vidstabdetect=shakiness=10:accuracy=15:result="mytransforms.trf"
```

- Visualize the result of internal transformations in the resulting video:

```
vidstabdetect=show=1
```

- Analyze a video with medium shakiness using `ffmpeg`:

```
ffmpeg -i input -vf vidstabdetect=shakiness=5:show=1 dummy.avi
```

37.73 vidstabtransform

Video stabilization/deshaking: pass 2 of 2, see vidstabdetect for pass 1.

Read a file with transform information for each frame and apply/compensate them. Together with the vidstabdetect filter this can be used to deshake videos. See also http://public.hronopik.de/vid.stab. It is important to also use the unsharp filter, see below.

To enable compilation of this filter you need to configure FFmpeg with `--enable-libvidstab`.

This filter accepts the following options:

‘`input`’

path to the file used to read the transforms (default: ‘`transforms.trf`’)

‘`smoothing`’

number of frames (value*2 + 1) used for lowpass filtering the camera movements (default: 10). For example a number of 10 means that 21 frames are used (10 in the past and 10 in the future) to smoothen the motion in the video. A larger values leads to a smoother video, but limits the acceleration of the camera (pan/tilt movements).

‘maxshift’

maximal number of pixels to translate frames (default: -1 no limit)

‘maxangle’

maximal angle in radians (degree*PI/180) to rotate frames (default: -1 no limit)

‘crop’

How to deal with borders that may be visible due to movement compensation. Available values are:

‘keep’

keep image information from previous frame (default)

‘black’

fill the border black

‘invert’

‘0’

keep transforms normal (default)

‘1’

invert transforms

‘relative’

consider transforms as

‘0’

absolute

‘1’

relative to previous frame (default)

‘zoom’

percentage to zoom (default: 0)

‘>0’

zoom in

‘<0’

zoom out

‘optzoom’

if 1 then optimal zoom value is determined (default). Optimal zoom means no (or only little) border should be visible. Note that the value given at zoom is added to the one calculated here.

‘interpol’

type of interpolation

Available values are:

‘no’

no interpolation

‘linear’

linear only horizontal

‘bilinear’

linear in both directions (default)

‘bicubic’

cubic in both directions (slow)

‘tripod’

virtual tripod mode means that the video is stabilized such that the camera stays stationary. Use also `tripod` option of vidstabdetect.

‘0’

off (default)

‘1’

virtual tripod mode: equivalent to `relative=0:smoothing=0`

37.73.1 Examples

- typical call with default default values: (note the unsharp filter which is always recommended)

```
ffmpeg -i inp.mpeg -vf vidstabtransform,unsharp=5:5:0.8:3:3:0.4 inp_stabilized.mpeg
```

- zoom in a bit more and load transform data from a given file

```
vidstabtransform=zoom=5:input="mytransforms.trf"
```

- smoothen the video even more

```
vidstabtransform=smoothing=30
```

37.74 vflip

Flip the input video vertically.

For example, to vertically flip a video with `ffmpeg`:

```
ffmpeg -i in.avi -vf "vflip" out.avi
```

37.75 yadif

Deinterlace the input video ("yadif" means "yet another deinterlacing filter").

This filter accepts the following options:

‘`mode`’

The interlacing mode to adopt, accepts one of the following values:

‘`0, send_frame`’

output 1 frame for each frame

‘`1, send_field`’

output 1 frame for each field

‘`2, send_frame_nospatial`’

like `send_frame` but skip spatial interlacing check

‘`3, send_field_nospatial`’

like `send_field` but skip spatial interlacing check

Default value is `send_frame`.

‘parity’

The picture field parity assumed for the input interlaced video, accepts one of the following values:

‘0, tff’

assume top field first

‘1, bff’

assume bottom field first

‘-1, auto’

enable automatic detection

Default value is auto. If interlacing is unknown or decoder does not export this information, top field first will be assumed.

‘deint’

Specify which frames to deinterlace. Accept one of the following values:

‘0, all’

deinterlace all frames

‘1, interlaced’

only deinterlace frames marked as interlaced

Default value is all.

38. Video Sources

Below is a description of the currently available video sources.

38.1 buffer

Buffer video frames, and make them available to the filter chain.

This source is mainly intended for a programmatic use, in particular through the interface defined in ‘libavfilter/vsrc_buffer.h’.

This source accepts the following options:

‘video_size’

Specify the size (width and height) of the buffered video frames.

‘width’

Input video width.

‘height’

Input video height.

‘pix_fmt’

A string representing the pixel format of the buffered video frames. It may be a number corresponding to a pixel format, or a pixel format name.

‘time_base’

Specify the timebase assumed by the timestamps of the buffered frames.

‘frame_rate’

Specify the frame rate expected for the video stream.

‘pixel_aspect, sar’

Specify the sample aspect ratio assumed by the video frames.

‘sws_param’

Specify the optional parameters to be used for the scale filter which is automatically inserted when an input change is detected in the input size or format.

For example:

```
buffer=width=320:height=240:pix_fmt=yuv410p:time_base=1/24:sar=1
```

will instruct the source to accept video frames with size 320x240 and with format "yuv410p", assuming 1/24 as the timestamps timebase and square pixels (1:1 sample aspect ratio). Since the pixel format with name "yuv410p" corresponds to the number 6 (check the enum AVPixelFormat definition in ‘libavutil/pixfmt.h’), this example corresponds to:

```
buffer=size=320x240:pixfmt=6:time_base=1/24:pixel_aspect=1/1
```

Alternatively, the options can be specified as a flat string, but this syntax is deprecated:

width:*height*:*pix_fmt*:*time_base.num*:*time_base.den*:*pixel_aspect.num*:*pixel_aspect.den*[:*sws_param*]

38.2 cellauto

Create a pattern generated by an elementary cellular automaton.

The initial state of the cellular automaton can be defined through the ‘`filename`’, and ‘`pattern`’ options. If such options are not specified an initial state is created randomly.

At each new frame a new row in the video is filled with the result of the cellular automaton next generation. The behavior when the whole frame is filled is defined by the ‘`scroll`’ option.

This source accepts the following options:

‘`filename, f`’

> Read the initial cellular automaton state, i.e. the starting row, from the specified file. In the file, each non-whitespace character is considered an alive cell, a newline will terminate the row, and further characters in the file will be ignored.

‘`pattern, p`’

> Read the initial cellular automaton state, i.e. the starting row, from the specified string.
>
> Each non-whitespace character in the string is considered an alive cell, a newline will terminate the row, and further characters in the string will be ignored.

‘`rate, r`’

> Set the video rate, that is the number of frames generated per second. Default is 25.

‘`random_fill_ratio, ratio`’

> Set the random fill ratio for the initial cellular automaton row. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI.
>
> This option is ignored when a file or a pattern is specified.

‘`random_seed, seed`’

> Set the seed for filling randomly the initial row, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

‘`rule`’

> Set the cellular automaton rule, it is a number ranging from 0 to 255. Default value is 110.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ or ‘`pattern`’ is specified, the size is set by default to the width of the specified initial state row, and the height is set to *width* * PHI.

If ‘`size`’ is set, it must contain the width of the specified pattern string, and the specified pattern will be centered in the larger row.

If a filename or a pattern string is not specified, the size value defaults to "320x518" (used for a randomly generated initial state).

‘`scroll`’

If set to 1, scroll the output upward when all the rows in the output have been already filled. If set to 0, the new generated row will be written over the top row just after the bottom row is filled. Defaults to 1.

‘`start_full, full`’

If set to 1, completely fill the output with generated rows before outputting the first frame. This is the default behavior, for disabling set the value to 0.

‘`stitch`’

If set to 1, stitch the left and right row edges together. This is the default behavior, for disabling set the value to 0.

38.2.1 Examples

- Read the initial state from ‘`pattern`’, and specify an output of size 200x400.

```
cellauto=f=pattern:s=200x400
```

- Generate a random initial row with a width of 200 cells, with a fill ratio of 2/3:

```
cellauto=ratio=2/3:s=200x200
```

- Create a pattern generated by rule 18 starting by a single alive cell centered on an initial row with width 100:

```
cellauto=p=@:s=100x400:full=0:rule=18
```

- Specify a more elaborated initial pattern:

```
cellauto=p='@@ @ @@':s=100x400:full=0:rule=18
```

38.3 mandelbrot

Generate a Mandelbrot set fractal, and progressively zoom towards the point specified with *start_x* and *start_y*.

This source accepts the following options:

‘`end_pts`’

Set the terminal pts value. Default value is 400.

‘`end_scale`’

Set the terminal scale value. Must be a floating point value. Default value is 0.3.

‘`inner`’

Set the inner coloring mode, that is the algorithm used to draw the Mandelbrot fractal internal region.

It shall assume one of the following values:

‘`black`’

Set black mode.

‘`convergence`’

Show time until convergence.

‘`mincol`’

Set color based on point closest to the origin of the iterations.

‘`period`’

Set period mode.

Default value is *mincol*.

‘`bailout`’

Set the bailout value. Default value is 10.0.

‘`maxiter`’

Set the maximum of iterations performed by the rendering algorithm. Default value is 7189.

‘`outer`’

Set outer coloring mode. It shall assume one of following values:

‘`iteration_count`’

Set iteration cound mode.

‘`normalized_iteration_count`’

set normalized iteration count mode.

Default value is *normalized_iteration_count*.

‘`rate, r`’

Set frame rate, expressed as number of frames per second. Default value is "25".

‘`size, s`’

Set frame size. Default value is "640x480".

‘`start_scale`’

Set the initial scale value. Default value is 3.0.

‘`start_x`’

Set the initial x position. Must be a floating point value between -100 and 100. Default value is -0.743643887037158704752191506114774.

‘`start_y`’

Set the initial y position. Must be a floating point value between -100 and 100. Default value is -0.131825904205311970493132056385139.

38.4 mptestsrc

Generate various test patterns, as generated by the MPlayer test filter.

The size of the generated video is fixed, and is 256x256. This source is useful in particular for testing encoding features.

This source accepts the following options:

‘rate, r’

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

‘duration, d’

Set the video duration of the sourced video. The accepted syntax is:

```
[-]HH:MM:SS[.m...]
[-]S+[.m...]
```

See also the function `av_parse_time()`.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

‘test, t’

Set the number or the name of the test to perform. Supported tests are:

‘dc_luma’
‘dc_chroma’
‘freq_luma’
‘freq_chroma’
‘amp_luma’
‘amp_chroma’
‘cbp’
‘mv’
‘ring1’
‘ring2’
‘all’

Default value is "all", which will cycle through the list of all tests.

For example the following:

```
testsrc=t=dc_luma
```

will generate a "dc_luma" test pattern.

38.5 frei0r_src

Provide a frei0r source.

To enable compilation of this filter you need to install the frei0r header and configure FFmpeg with `--enable-frei0r`.

This source accepts the following options:

‘`size`’

The size of the video to generate, may be a string of the form *width*x*height* or a frame size abbreviation.

‘`framerate`’

Framerate of the generated video, may be a string of the form *num*/*den* or a frame rate abbreviation.

‘`filter_name`’

The name to the frei0r source to load. For more information regarding frei0r and how to set the parameters read the section frei0r in the description of the video filters.

‘`filter_params`’

A ’|’-separated list of parameters to pass to the frei0r source.

For example, to generate a frei0r partik0l source with size 200x200 and frame rate 10 which is overlayed on the overlay filter main input:

```
frei0r_src=size=200x200:framerate=10:filter_name=partik0l:filter_params=1234 [overlay]; [in][overlay] overlay
```

38.6 life

Generate a life pattern.

This source is based on a generalization of John Conway’s life game.

The sourced input represents a life grid, each pixel represents a cell which can be in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.

At each interaction the grid evolves according to the adopted rule, which specifies the number of neighbor alive cells which will make a cell stay alive or born. The ‘`rule`’ option allows to specify the rule to adopt.

This source accepts the following options:

‘`filename, f`’

Set the file from which to read the initial grid state. In the file, each non-whitespace character is considered an alive cell, and newline is used to delimit the end of each row.

If this option is not specified, the initial grid is generated randomly.

‘`rate, r`’

Set the video rate, that is the number of frames generated per second. Default is 25.

‘`random_fill_ratio, ratio`’

Set the random fill ratio for the initial random grid. It is a floating point number value ranging from 0 to 1, defaults to 1/PHI. It is ignored when a file is specified.

‘`random_seed, seed`’

Set the seed for filling the initial random grid, must be an integer included between 0 and UINT32_MAX. If not specified, or if explicitly set to -1, the filter will try to use a good random seed on a best effort basis.

‘`rule`’

Set the life rule.

A rule can be specified with a code of the kind "S*NS*/B*NB*", where *NS* and *NB* are sequences of numbers in the range 0-8, *NS* specifies the number of alive neighbor cells which make a live cell stay alive, and *NB* the number of alive neighbor cells which make a dead cell to become alive (i.e. to "born"). "s" and "b" can be used in place of "S" and "B", respectively.

Alternatively a rule can be specified by an 18-bits integer. The 9 high order bits are used to encode the next cell state if it is alive for each number of neighbor alive cells, the low order bits specify the rule for "borning" new cells. Higher order bits encode for an higher number of neighbor cells. For example the number 6153 = `(12<<9)+9` specifies a stay alive rule of 12 and a born rule of 9, which corresponds to "S23/B03".

Default value is "S23/B3", which is the original Conway’s game of life rule, and will keep a cell alive if it has 2 or 3 neighbor alive cells, and will born a new cell if there are three alive cells around a dead cell.

‘`size, s`’

Set the size of the output video.

If ‘`filename`’ is specified, the size is set by default to the same size of the input file. If ‘`size`’ is set, it must contain the size specified in the input file, and the initial grid defined in that file is centered in the larger resulting area.

If a filename is not specified, the size value defaults to "320x240" (used for a randomly generated initial grid).

‘stitch’

If set to 1, stitch the left and right grid edges together, and the top and bottom edges also. Defaults to 1.

‘mold’

Set cell mold speed. If set, a dead cell will go from ‘death_color’ to ‘mold_color’ with a step of ‘mold’. ‘mold’ can have a value from 0 to 255.

‘life_color’

Set the color of living (or new born) cells.

‘death_color’

Set the color of dead cells. If ‘mold’ is set, this is the first color used to represent a dead cell.

‘mold_color’

Set mold color, for definitely dead and moldy cells.

38.6.1 Examples

- Read a grid from ‘pattern’, and center it on a grid of size 300x300 pixels:

```
life=f=pattern:s=300x300
```

- Generate a random grid of size 200x200, with a fill ratio of 2/3:

```
life=ratio=2/3:s=200x200
```

- Specify a custom rule for evolving a randomly generated grid:

```
life=rule=S14/B34
```

- Full example with slow death effect (mold) using ffplay:

```
ffplay -f lavfi life=s=300x200:mold=10:r=60:ratio=0.1:death_color=#C83232:life_color=#00ff00,scale=1200:800:flags=16
```

38.7 color, nullsrc, rgbtestsrc, smptebars, smptehdbars, testsrc

The color source provides an uniformly colored input.

The `nullsrc` source returns unprocessed video frames. It is mainly useful to be employed in analysis / debugging tools, or as the source for filters which ignore the input data.

The `rgbtestsrc` source generates an RGB test pattern useful for detecting RGB vs BGR issues. You should see a red, green and blue stripe from top to bottom.

The `smptebars` source generates a color bars pattern, based on the SMPTE Engineering Guideline EG 1-1990.

The `smptehdbars` source generates a color bars pattern, based on the SMPTE RP 219-2002.

The `testsrc` source generates a test video pattern, showing a color pattern, a scrolling gradient and a timestamp. This is mainly intended for testing purposes.

The sources accept the following options:

‘`color, c`’

Specify the color of the source, only used in the `color` source. It can be the name of a color (case insensitive match) or a 0xRRGGBB[AA] sequence, possibly followed by an alpha specifier. The default value is "black".

‘`size, s`’

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation. The default value is "320x240".

‘`rate, r`’

Specify the frame rate of the sourced video, as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation. The default value is "25".

‘`sar`’

Set the sample aspect ratio of the sourced video.

‘`duration, d`’

Set the video duration of the sourced video. The accepted syntax is:

```
[-]HH[:MM[:SS[.m...]]]
[-]S+[.m...]
```

See also the function `av_parse_time()`.

If not specified, or the expressed duration is negative, the video is supposed to be generated forever.

‘decimals, n’

Set the number of decimals to show in the timestamp, only used in the testsrc source.

The displayed timestamp value will correspond to the original timestamp value multiplied by the power of 10 of the specified value. Default value is 0.

For example the following:

```
testsrc=duration=5.3:size=qcif:rate=10
```

will generate a video with a duration of 5.3 seconds, with size 176x144 and a frame rate of 10 frames per second.

The following graph description will generate a red source with an opacity of 0.2, with size "qcif" and a frame rate of 10 frames per second.

```
color=c=red@0.2:s=qcif:r=10
```

If the input content is to be ignored, nullsrc can be used. The following command generates noise in the luminance plane by employing the geq filter:

```
nullsrc=s=256x256, geq=random(1)*255:128:128
```

39. Video Sinks

Below is a description of the currently available video sinks.

39.1 buffersink

Buffer video frames, and make them available to the end of the filter graph.

This sink is mainly intended for a programmatic use, in particular through the interface defined in ‘libavfilter/buffersink.h’ or the options system.

It accepts a pointer to an AVBufferSinkContext structure, which defines the incoming buffers’ formats, to be passed as the opaque parameter to avfilter_init_filter for initialization.

39.2 nullsink

Null video sink, do absolutely nothing with the input video. It is mainly useful as a template and to be employed in analysis / debugging tools.

40. Multimedia Filters

Below is a description of the currently available multimedia filters.

40.1 concat

Concatenate audio and video streams, joining them together one after the other.

The filter works on segments of synchronized video and audio streams. All segments must have the same number of streams of each type, and that will also be the number of streams at output.

The filter accepts the following options:

‘n’

Set the number of segments. Default is 2.

‘v’

Set the number of output video streams, that is also the number of video streams in each segment. Default is 1.

‘a’

Set the number of output audio streams, that is also the number of video streams in each segment. Default is 0.

‘unsafe’

Activate unsafe mode: do not fail if segments have a different format.

The filter has *v*+*a* outputs: first *v* video outputs, then *a* audio outputs.

There are *n*x(*v*+*a*) inputs: first the inputs for the first segment, in the same order as the outputs, then the inputs for the second segment, etc.

Related streams do not always have exactly the same duration, for various reasons including codec frame size or sloppy authoring. For that reason, related synchronized streams (e.g. a video and its audio track) should be concatenated at once. The concat filter will use the duration of the longest stream in each segment (except the last one), and if necessary pad shorter audio streams with silence.

For this filter to work correctly, all segments must start at timestamp 0.

All corresponding streams must have the same parameters in all segments; the filtering system will automatically select a common pixel format for video streams, and a common sample format, sample rate and channel layout for audio streams, but other settings, such as resolution, must be converted explicitly by the user.

Different frame rates are acceptable but will result in variable frame rate at output; be sure to configure the output file to handle it.

40.1.1 Examples

- Concatenate an opening, an episode and an ending, all in bilingual version (video in stream 0, audio in streams 1 and 2):

```
ffmpeg -i opening.mkv -i episode.mkv -i ending.mkv -filter_complex \
  '[0:0] [0:1] [0:2] [1:0] [1:1] [1:2] [2:0] [2:1] [2:2]
   concat=n=3:v=1:a=2 [v] [a1] [a2]' \
  -map '[v]' -map '[a1]' -map '[a2]' output.mkv
```

- Concatenate two parts, handling audio and video separately, using the (a)movie sources, and adjusting the resolution:

```
movie=part1.mp4, scale=512:288 [v1] ; amovie=part1.mp4 [a1] ;
movie=part2.mp4, scale=512:288 [v2] ; amovie=part2.mp4 [a2] ;
[v1] [v2] concat [outv] ; [a1] [a2] concat=v=0:a=1 [outa]
```

  Note that a desync will happen at the stitch if the audio and video streams do not have exactly the same duration in the first file.

40.2 ebur128

EBU R128 scanner filter. This filter takes an audio stream as input and outputs it unchanged. By default, it logs a message at a frequency of 10Hz with the Momentary loudness (identified by `M`), Short-term loudness (`S`), Integrated loudness (`I`) and Loudness Range (`LRA`).

The filter also has a video output (see the *video* option) with a real time graph to observe the loudness evolution. The graphic contains the logged message mentioned above, so it is not printed anymore when this option is set, unless the verbose logging is set. The main graphing area contains the short-term loudness (3 seconds of analysis), and the gauge on the right is for the momentary loudness (400 milliseconds).

More information about the Loudness Recommendation EBU R128 on http://tech.ebu.ch/loudness.

The filter accepts the following options:

‘`video`’

  Activate the video output. The audio stream is passed unchanged whether this option is set or no. The video stream will be the first output stream if activated. Default is `0`.

‘`size`’

Set the video size. This option is for video only. Default and minimum resolution is `640x480`.

‘`meter`’

Set the EBU scale meter. Default is `9`. Common values are `9` and `18`, respectively for EBU scale meter +9 and EBU scale meter +18. Any other integer value between this range is allowed.

‘`metadata`’

Set metadata injection. If set to `1`, the audio input will be segmented into 100ms output frames, each of them containing various loudness information in metadata. All the metadata keys are prefixed with `lavfi.r128.`.

Default is `0`.

‘`framelog`’

Force the frame logging level.

Available values are:

‘`info`’

information logging level

‘`verbose`’

verbose logging level

By default, the logging level is set to *info*. If the ‘`video`’ or the ‘`metadata`’ options are set, it switches to *verbose*.

40.2.1 Examples

- Real-time graph using `ffplay`, with a EBU scale meter +18:

```
ffplay -f lavfi -i "amovie=input.mp3,ebur128=video=1:meter=18 [out0][out1]"
```

- Run an analysis with `ffmpeg`:

```
ffmpeg -nostats -i input.mp3 -filter_complex ebur128 -f null -
```

40.3 interleave, ainterleave

Temporally interleave frames from several inputs.

`interleave` works with video inputs, `ainterleave` with audio.

These filters read frames from several inputs and send the oldest queued frame to the output.

Input streams must have a well defined, monotonically increasing frame timestamp values.

In order to submit one frame to output, these filters need to enqueue at least one frame for each input, so they cannot work in case one input is not yet terminated and will not receive incoming frames.

For example consider the case when one input is a `select` filter which always drop input frames. The `interleave` filter will keep reading from that input, but it will never be able to send new frames to output until the input will send an end-of-stream signal.

Also, depending on inputs synchronization, the filters will drop frames in case one input receives more frames than the other ones, and the queue is already filled.

These filters accept the following options:

‘`nb_inputs, n`’

Set the number of different inputs, it is 2 by default.

40.3.1 Examples

- Interleave frames belonging to different streams using `ffmpeg`:

```
ffmpeg -i bambi.avi -i pr0n.mkv -filter_complex "[0:v][1:v] interleave" out.avi
```

- Add flickering blur effect:

```
select='if(gt(random(0), 0.2), 1, 2)':n=2 [tmp], boxblur=2:2, [tmp] interleave
```

40.4 perms, aperms

Set read/write permissions for the output frames.

These filters are mainly aimed at developers to test direct path in the following filter in the filtergraph.

The filters accept the following options:

‘mode’

Select the permissions mode.

It accepts the following values:

‘none’

Do nothing. This is the default.

‘ro’

Set all the output frames read-only.

‘rw’

Set all the output frames directly writable.

‘toggle’

Make the frame read-only if writable, and writable if read-only.

‘random’

Set each output frame read-only or writable randomly.

‘seed’

Set the seed for the *random* mode, must be an integer included between `0` and `UINT32_MAX`. If not specified, or if explicitly set to `-1`, the filter will try to use a good random seed on a best effort basis.

Note: in case of auto-inserted filter between the permission filter and the following one, the permission might not be received as expected in that following filter. Inserting a format or aformat filter before the perms/aperms filter can avoid this problem.

40.5 select, aselect

Select frames to pass in output.

This filter accepts the following options:

‘expr, e’

Set expression, which is evaluated for each input frame.

If the expression is evaluated to zero, the frame is discarded.

If the evaluation result is negative or NaN, the frame is sent to the first output; otherwise it is sent to the output with index `ceil(val)-1`, assuming that the input index starts from 0.

For example a value of `1.2` corresponds to the output with index `ceil(1.2)-1 = 2-1 = 1`, that is the second output.

‘`outputs, n`’

Set the number of outputs. The output to which to send the selected frame is based on the result of the evaluation. Default value is 1.

The expression can contain the following constants:

‘`n`’

the sequential number of the filtered frame, starting from 0

‘`selected_n`’

the sequential number of the selected frame, starting from 0

‘`prev_selected_n`’

the sequential number of the last selected frame, NAN if undefined

‘`TB`’

timebase of the input timestamps

‘`pts`’

the PTS (Presentation TimeStamp) of the filtered video frame, expressed in *TB* units, NAN if undefined

‘`t`’

the PTS (Presentation TimeStamp) of the filtered video frame, expressed in seconds, NAN if undefined

‘`prev_pts`’

the PTS of the previously filtered video frame, NAN if undefined

‘`prev_selected_pts`’

the PTS of the last previously filtered video frame, NAN if undefined

‘`prev_selected_t`’

the PTS of the last previously selected video frame, NAN if undefined

‘start_pts’

the PTS of the first video frame in the video, NAN if undefined

‘start_t’

the time of the first video frame in the video, NAN if undefined

‘pict_type *(video only)*’

the type of the filtered frame, can assume one of the following values:

‘I’
‘P’
‘B’
‘S’
‘SI’
‘SP’
‘BI’

‘interlace_type *(video only)*’

the frame interlace type, can assume one of the following values:

‘PROGRESSIVE’

the frame is progressive (not interlaced)

‘TOPFIRST’

the frame is top-field-first

‘BOTTOMFIRST’

the frame is bottom-field-first

‘consumed_sample_n *(audio only)*’

the number of selected samples before the current frame

‘samples_n *(audio only)*’

the number of samples in the current frame

‘sample_rate *(audio only)*’

the input sample rate

‘key’

1 if the filtered frame is a key-frame, 0 otherwise

‘pos’

the position in the file of the filtered frame, -1 if the information is not available (e.g. for synthetic video)

‘scene *(video only)*’

value between 0 and 1 to indicate a new scene; a low value reflects a low probability for the current frame to introduce a new scene, while a higher value means the current frame is more likely to be one (see the example below)

The default value of the select expression is "1".

40.5.1 Examples

- Select all frames in input:

```
select
```

  The example above is the same as:

```
select=1
```

- Skip all frames:

```
select=0
```

- Select only I-frames:

```
select='eq(pict_type\,I)'
```

- Select one frame every 100:

```
select='not(mod(n\,100))'
```

- Select only frames contained in the 10-20 time interval:

```
select='gte(t\,10)*lte(t\,20)'
```

- Select only I frames contained in the 10-20 time interval:

```
select='gte(t\,10)*lte(t\,20)*eq(pict_type\,I)'
```

- Select frames with a minimum distance of 10 seconds:

```
select='isnan(prev_selected_t)+gte(t-prev_selected_t\,10)'
```

- Use aselect to select only audio frames with samples number > 100:

```
aselect='gt(samples_n\,100)'
```

- Create a mosaic of the first scenes:

```
ffmpeg -i video.avi -vf select='gt(scene\,0.4)',scale=160:120,tile -frames:v 1 preview.png
```

  Comparing *scene* against a value between 0.3 and 0.5 is generally a sane choice.

- Send even and odd frames to separate outputs, and compose them:

```
select=n=2:e='mod(n, 2)+1' [odd][even]; [odd] pad=h=2*ih [tmp]; [tmp][even] overlay=y=h
```

40.6 sendcmd, asendcmd

Send commands to filters in the filtergraph.

These filters read commands to be sent to other filters in the filtergraph.

`sendcmd` must be inserted between two video filters, `asendcmd` must be inserted between two audio filters, but apart from that they act the same way.

The specification of commands can be provided in the filter arguments with the *commands* option, or in a file specified by the *filename* option.

These filters accept the following options:

`'commands, c'`

Set the commands to be read and sent to the other filters.

`'filename, f'`

Set the filename of the commands to be read and sent to the other filters.

40.6.1 Commands syntax

A commands description consists of a sequence of interval specifications, comprising a list of commands to be executed when a particular event related to that interval occurs. The occurring event is typically the current frame time entering or leaving a given time interval.

An interval is specified by the following syntax:

```
START[-END] COMMANDS;
```

The time interval is specified by the *START* and *END* times. *END* is optional and defaults to the maximum time.

The current frame time is considered within the specified interval if it is included in the interval [*START*, *END*), that is when the time is greater or equal to *START* and is lesser than *END*.

COMMANDS consists of a sequence of one or more command specifications, separated by ",", relating to that interval. The syntax of a command specification is given by:

```
[FLAGS] TARGET COMMAND ARG
```

FLAGS is optional and specifies the type of events relating to the time interval which enable sending the specified command, and must be a non-null sequence of identifier flags separated by "+" or "|" and enclosed between "[" and "]".

The following flags are recognized:

`‘enter’`

The command is sent when the current frame timestamp enters the specified interval. In other words, the command is sent when the previous frame timestamp was not in the given interval, and the current is.

`‘leave’`

The command is sent when the current frame timestamp leaves the specified interval. In other words, the command is sent when the previous frame timestamp was in the given interval, and the current is not.

If *FLAGS* is not specified, a default value of `[enter]` is assumed.

TARGET specifies the target of the command, usually the name of the filter class or a specific filter instance name.

COMMAND specifies the name of the command for the target filter.

ARG is optional and specifies the optional list of argument for the given *COMMAND*.

Between one interval specification and another, whitespaces, or sequences of characters starting with # until the end of line, are ignored and can be used to annotate comments.

A simplified BNF description of the commands specification syntax follows:

```
COMMAND_FLAG  ::= "enter" | "leave"
COMMAND_FLAGS ::= COMMAND_FLAG [(+|"|")COMMAND_FLAG]
COMMAND       ::= ["[" COMMAND_FLAGS "]"] TARGET COMMAND [ARG]
COMMANDS      ::= COMMAND [,COMMANDS]
INTERVAL      ::= START[-END] COMMANDS
INTERVALS     ::= INTERVAL[;INTERVALS]
```

40.6.2 Examples

- Specify audio tempo change at second 4:

```
asendcmd=c='4.0 atempo tempo 1.5',atempo
```

- Specify a list of drawtext and hue commands in a file.

```
# show text in the interval 5-10
5.0-10.0 [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=hello world',
         [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=';

# desaturate the image in the interval 15-20
15.0-20.0 [enter] hue s 0,
          [enter] drawtext reinit 'fontfile=FreeSerif.ttf:text=nocolor',
          [leave] hue s 1,
          [leave] drawtext reinit 'fontfile=FreeSerif.ttf:text=color';

# apply an exponential saturation fade-out effect, starting from time 25
25 [enter] hue s exp(25-t)
```

A filtergraph allowing to read and process the above command list stored in a file '`test.cmd`', can be specified with:

```
sendcmd=f=test.cmd,drawtext=fontfile=FreeSerif.ttf:text='',hue
```

40.7 setpts, asetpts

Change the PTS (presentation timestamp) of the input frames.

`setpts` works on video frames, `asetpts` on audio frames.

This filter accepts the following options:

‘expr’

The expression which is evaluated for each frame to construct its timestamp.

The expression is evaluated through the eval API and can contain the following constants:

‘FRAME_RATE’

frame rate, only defined for constant frame-rate video

‘PTS’

the presentation timestamp in input

‘N’

the count of the input frame, starting from 0.

‘NB_CONSUMED_SAMPLES’

the number of consumed samples, not including the current frame (only audio)

‘NB_SAMPLES’

the number of samples in the current frame (only audio)

‘SAMPLE_RATE’

audio sample rate

‘STARTPTS’

the PTS of the first frame

‘STARTT’

the time in seconds of the first frame

‘INTERLACED’

tell if the current frame is interlaced

‘T’

the time in seconds of the current frame

‘TB’

the time base

‘POS’

original position in the file of the frame, or undefined if undefined for the current frame

‘PREV_INPTS’

previous input PTS

‘PREV_INT’

previous input time in seconds

‘PREV_OUTPTS’

previous output PTS

‘PREV_OUTT’

previous output time in seconds

‘RTCTIME’

wallclock (RTC) time in microseconds. This is deprecated, use time(0) instead.

‘RTCSTART’

wallclock (RTC) time at the start of the movie in microseconds

40.7.1 Examples

- Start counting PTS from zero

```
setpts=PTS-STARTPTS
```

- Apply fast motion effect:

```
setpts=0.5*PTS
```

- Apply slow motion effect:

```
setpts=2.0*PTS
```

- Set fixed rate of 25 frames per second:

```
setpts=N/(25*TB)
```

- Set fixed rate 25 fps with some jitter:

```
setpts='1/(25*TB) * (N + 0.05 * sin(N*2*PI/25))'
```

- Apply an offset of 10 seconds to the input PTS:

```
setpts=PTS+10/TB
```

- Generate timestamps from a "live source" and rebase onto the current timebase:

```
setpts='(RTCTIME - RTCSTART) / (TB * 1000000)'
```

40.8 settb, asettb

Set the timebase to use for the output frames timestamps. It is mainly useful for testing timebase configuration.

This filter accepts the following options:

`'expr, tb'`

The expression which is evaluated into the output timebase.

The value for '`tb`' is an arithmetic expression representing a rational. The expression can contain the constants "AVTB" (the default timebase), "intb" (the input timebase) and "sr" (the sample rate, audio only). Default value is "intb".

40.8.1 Examples

- Set the timebase to 1/25:

```
settb=expr=1/25
```

- Set the timebase to 1/10:

```
settb=expr=0.1
```

- Set the timebase to 1001/1000:

```
settb=1+0.001
```

- Set the timebase to 2*intb:

```
settb=2*intb
```

- Set the default timebase value:

```
settb=AVTB
```

40.9 showspectrum

Convert input audio to a video output, representing the audio frequency spectrum.

The filter accepts the following options:

‘`size, s`’

Specify the video size for the output. Default value is `640x512`.

‘`slide`’

Specify if the spectrum should slide along the window. Default value is `0`.

‘`mode`’

Specify display mode.

It accepts the following values:

‘`combined`’

all channels are displayed in the same row

‘`separate`’

all channels are displayed in separate rows

Default value is ‘`combined`’.

‘`color`’

Specify display color mode.

It accepts the following values:

‘`channel`’

each channel is displayed in a separate color

‘intensity’

each channel is is displayed using the same color scheme

Default value is ‘channel’.

‘scale’

Specify scale used for calculating intensity color values.

It accepts the following values:

‘lin’

linear

‘sqrt’

square root, default

‘cbrt’

cubic root

‘log’

logarithmic

Default value is ‘sqrt’.

‘saturation’

Set saturation modifier for displayed colors. Negative values provide alternative color scheme. 0 is no saturation at all. Saturation must be in [-10.0, 10.0] range. Default value is 1.

The usage is very similar to the showwaves filter; see the examples in that section.

40.9.1 Examples

- Large window with logarithmic color scaling:

```
showspectrum=s=1280x480:scale=log
```

- Complete example for a colored and sliding spectrum per channel using ffplay:

```
ffplay -f lavfi 'amovie=input.mp3, asplit [a][out1];
             [a] showspectrum=mode=separate:color=intensity:slide=1:scale=cbrt [out0]'
```

40.10 showwaves

Convert input audio to a video output, representing the samples waves.

The filter accepts the following options:

‘`size, s`’

Specify the video size for the output. Default value is "600x240".

‘`mode`’

Set display mode.

Available values are:

‘`point`’

Draw a point for each sample.

‘`line`’

Draw a vertical line for each sample.

Default value is `point`.

‘`n`’

Set the number of samples which are printed on the same column. A larger value will decrease the frame rate. Must be a positive integer. This option can be set only if the value for *rate* is not explicitly specified.

‘`rate, r`’

Set the (approximate) output frame rate. This is done by setting the option *n*. Default value is "25".

40.10.1 Examples

- Output the input file audio and the corresponding video representation at the same time:

```
amovie=a.mp3,asplit[out0],showwaves[out1]
```

- Create a synthetic signal and show it with showwaves, forcing a frame rate of 30 frames per second:

```
aevalsrc=sin(1*2*PI*t)*sin(880*2*PI*t):cos(2*PI*200*t),asplit[out0],showwaves=r=30[out1]
```

40.11 split, asplit

Split input into several identical outputs.

`asplit` works with audio input, `split` with video.

The filter accepts a single parameter which specifies the number of outputs. If unspecified, it defaults to 2.

40.11.1 Examples

- Create two separate outputs from the same input:

```
[in] split [out0][out1]
```

- To create 3 or more outputs, you need to specify the number of outputs, like in:

```
[in] asplit=3 [out0][out1][out2]
```

- Create two separate outputs from the same input, one cropped and one padded:

```
[in] split [splitout1][splitout2];
[splitout1] crop=100:100:0:0    [cropout];
[splitout2] pad=200:200:100:100 [padout];
```

- Create 5 copies of the input audio with `ffmpeg`:

```
ffmpeg -i INPUT -filter_complex asplit=5 OUTPUT
```

41. Multimedia Sources

Below is a description of the currently available multimedia sources.

41.1 amovie

This is the same as movie source, except it selects an audio stream by default.

41.2 movie

Read audio and/or video stream(s) from a movie container.

This filter accepts the following options:

‘`filename`’

The name of the resource to read (not necessarily a file but also a device or a stream accessed through some protocol).

‘`format_name, f`’

Specifies the format assumed for the movie to read, and can be either the name of a container or an input device. If not specified the format is guessed from *movie_name* or by probing.

‘`seek_point, sp`’

Specifies the seek point in seconds, the frames will be output starting from this seek point, the parameter is evaluated with `av_strtod` so the numerical value may be suffixed by an IS postfix. Default value is "0".

‘`streams, s`’

Specifies the streams to read. Several streams can be specified, separated by "+". The source will then have as many outputs, in the same order. The syntax is explained in the “Stream specifiers” section in the ffmpeg manual. Two special names, "dv" and "da" specify respectively the default (best suited) video and audio stream. Default is "dv", or "da" if the filter is called as "amovie".

‘`stream_index, si`’

Specifies the index of the video stream to read. If the value is -1, the best suited video stream will be automatically selected. Default value is "-1". Deprecated. If the filter is called "amovie", it will select audio instead of video.

‘`loop`’

Specifies how many times to read the stream in sequence. If the value is less than 1, the stream will be read again and again. Default value is "1".

Note that when the movie is looped the source timestamps are not changed, so it will generate non monotonically increasing timestamps.

This filter allows to overlay a second video on top of main input of a filtergraph as shown in this graph:

```
input -----------> deltapts0 --> overlay --> output
                                      ^
                                      |
movie --> scale--> deltapts1 -------+
```

41.2.1 Examples

- Skip 3.2 seconds from the start of the avi file in.avi, and overlay it on top of the input labelled as "in":

```
movie=in.avi:seek_point=3.2, scale=180:-1, setpts=PTS-STARTPTS [over];
[in] setpts=PTS-STARTPTS [main];
[main][over] overlay=16:16 [out]
```

- Read from a video4linux2 device, and overlay it on top of the input labelled as "in":

```
movie=/dev/video0:f=video4linux2, scale=180:-1, setpts=PTS-STARTPTS [over];
[in] setpts=PTS-STARTPTS [main];
[main][over] overlay=16:16 [out]
```

- Read the first video stream and the audio stream with id 0x81 from dvd.vob; the video is connected to the pad named "video" and the audio is connected to the pad named "audio":

```
movie=dvd.vob:s=v:0+#0x81 [video] [audio]
```

42. See Also

ffmpeg ffplay, ffprobe, ffserver, ffmpeg-utils, ffmpeg-scaler, ffmpeg-resampler, ffmpeg-codecs, ffmpeg-bitstream-filters, ffmpeg-formats, ffmpeg-devices, ffmpeg-protocols, ffmpeg-filters

43. Authors

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file ‘`MAINTAINERS`’ in the source code tree.

This document was generated by *john* on *May 2, 2013* using *texi2html 1.82*.