FFmpeg Formats Documentation

Table of Contents

1. Description

This document describes the supported formats (muxers and demuxers) provided by the libavformat library.

2. 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.

3. 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.

3.1 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".

3.2 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.

3.2.1 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.

3.2.2 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.

3.3 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.

3.4 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.

3.4.1 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
```

3.5 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
```

3.6 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.

3.7 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
```

4. 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.

4.1 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.

4.2 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.

4.3 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.

4.4 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*.

4.5 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

4.6 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.

4.7 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.

4.8 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)
```

4.9 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
```

4.10 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 -
```

4.11 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
```

4.12 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.

4.12.1 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
```

4.13 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
```

4.14 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.

4.15 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.

5. 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
```

6. See Also

ffmpeg, ffplay, ffprobe, ffserver, libavformat

7. 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 Formats Documentation

Table of Contents

1. Description

This document describes the supported formats (muxers and demuxers) provided by the libavformat library.

2. 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.

3. 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.

3.1 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".

3.2 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.

3.2.1 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.

3.2.2 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.

3.3 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.

3.4 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.

3.4.1 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
```

3.5 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
```

3.6 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.

3.7 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
```

4. 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.

4.1 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.

4.2 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.

4.3 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.

4.4 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*.

4.5 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

4.6 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.

4.7 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.

4.8 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)
```

4.9 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
```

4.10 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 -
```

4.11 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
```

4.12 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.

4.12.1 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
```

4.13 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
```

4.14 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.

4.15 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.

5. 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
```

6. See Also

ffmpeg, ffplay, ffprobe, ffserver, libavformat

7. 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*.