FFmpeg Formats Documentation

Table of Contents

1 Description# TOC

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

2 Format Options# TOC

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 enable detecting 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.

‘`fastseek`’

Enable fast, but inaccurate seeks for some formats.

‘`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

‘`bitexact`’

Only write platform-, build- and time-independent data. This ensures that file and data checksums are reproducible and match between platforms. Its primary use is for regression testing.

`seek2any` *`integer`* (*`input`*)

Allow seeking to non-keyframes on demuxer level when supported if set to 1. Default is 0.

`analyzeduration` *`integer`* (*`input`*)

Specify how many microseconds are analyzed to probe the input. A higher value will enable detecting 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.

`max_interleave_delta` *`integer`* (*`output`*)

Set maximum buffering duration for interleaving. The duration is expressed in microseconds, and defaults to 1000000 (1 second).

To ensure all the streams are interleaved correctly, libavformat will wait until it has at least one packet for each stream before actually writing any packets to the output file. When some streams are "sparse" (i.e. there are large gaps between successive packets), this can result in excessive buffering.

This field specifies the maximum difference between the timestamps of the first and the last packet in the muxing queue, above which libavformat will output a packet regardless of whether it has queued a packet for all the streams.

If set to 0, libavformat will continue buffering packets until it has a packet for each stream, regardless of the maximum timestamp difference between the buffered packets.

`use_wallclock_as_timestamps` *`integer`* (*`input`*)

Use wallclock as timestamps.

`avoid_negative_ts` *`integer`* (*`output`*)

Possible values:

‘make_non_negative’

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

‘make_zero’

Shift timestamps so that the first timestamp is 0.

‘auto (default)’

Enables shifting when required by the target format.

‘disabled’

Disables shifting of timestamp.

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.

`skip_initial_bytes` *integer* (*input*)

Set number of bytes to skip before reading header and frames if set to 1. Default is 0.

`correct_ts_overflow` *integer* (*input*)

Correct single timestamp overflows if set to 1. Default is 1.

`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.

`output_ts_offset` *offset* (*output*)

Set the output time offset.

offset must be a time duration specification, see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

The offset is added by the muxer to the output timestamps.

Specifying a positive offset means that the corresponding streams are delayed bt the time duration specified in *offset*. Default value is `0` (meaning that no offset is applied).

`format_whitelist` *list* (*input*)

"," separated List of allowed demuxers. By default all are allowed.

`dump_separator` *`string`* (*`input`*)

Separator used to separate the fields printed on the command line about the Stream parameters. For example to separate the fields with newlines and indention:

```
ffprobe -dump_separator "
                         "  -i ~/videos/matrixbench_mpeg2.mpg
```

2.1 Format stream specifiers# TOC

Format stream specifiers allow selection of one or more streams that match specific properties.

Possible forms of stream specifiers are:

`stream_index`

Matches the stream with this index.

`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 the 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.

The exact semantics of stream specifiers is defined by the `avformat_match_stream_specifier()` function declared in the `libavformat/avformat.h` header.

3 Demuxers# TOC

Demuxers are configured elements in FFmpeg that can 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 aa# TOC

Audible Format 2, 3, and 4 demuxer.

This demuxer is used to demux Audible Format 2, 3, and 4 (.aa) files.

3.2 applehttp# TOC

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".

3.3 apng# TOC

Animated Portable Network Graphics demuxer.

This demuxer is used to demux APNG files. All headers, but the PNG signature, up to (but not including) the first fcTL chunk are transmitted as extradata. Frames are then split as being all the chunks between two fcTL ones, or between the last fcTL and IEND chunks.

`-ignore_loop` *`bool`*

Ignore the loop variable in the file if set.

`-max_fps` *`int`*

Maximum framerate in frames per second (0 for no limit).

`-default_fps` *`int`*

Default framerate in frames per second when none is specified in the file (0 meaning as fast as possible).

3.4 asf# TOC

Advanced Systems Format demuxer.

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

`-no_resync_search` *`bool`*

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

3.5 concat# TOC

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.

3.5.1 Syntax# TOC

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 file-related 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.

`inpoint` *`timestamp`*

In point of the file. When the demuxer opens the file it instantly seeks to the specified timestamp. Seeking is done so that all streams can be presented successfully at In point.

This directive works best with intra frame codecs, because for non-intra frame ones you will usually get extra packets before the actual In point and the decoded content will most likely contain frames before In point too.

For each file, packets before the file In point will have timestamps less than the calculated start timestamp of the file (negative in case of the first file), and the duration of the files (if not specified by the `duration` directive) will be reduced based on their specified In point.

Because of potential packets before the specified In point, packet timestamps may overlap between two concatenated files.

`outpoint` *`timestamp`*

Out point of the file. When the demuxer reaches the specified decoding timestamp in any of the streams, it handles it as an end of file condition and skips the current and all the remaining packets from all streams.

Out point is exclusive, which means that the demuxer will not output packets with a decoding timestamp greater or equal to Out point.

This directive works best with intra frame codecs and formats where all streams are tightly interleaved. For non-intra frame codecs you will usually get additional packets with presentation timestamp after Out point therefore the decoded content will most likely contain frames after Out point too. If your streams are not tightly interleaved you may not get all the packets from all streams before Out point and you may only will be able to decode the earliest stream until Out point.

The duration of the files (if not specified by the `duration` directive) will be reduced based on their specified Out point.

`file_packet_metadata` *`key=value`*

Metadata of the packets of the file. The specified metadata will be set for each file packet. You can specify this directive multiple times to add multiple metadata entries.

`stream`

Introduce a stream in the virtual file. All subsequent stream-related directives apply to the last introduced stream. Some streams properties must be set in order to allow identifying the matching streams in the subfiles. If no streams are defined in the script, the streams from the first file are

copied.

`exact_stream_id` *id*

Set the id of the stream. If this directive is given, the string with the corresponding id in the subfiles will be used. This is especially useful for MPEG-PS (VOB) files, where the order of the streams is not reliable.

3.5.2 Options# TOC

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.

`auto_convert`

If set to 1, try to perform automatic conversions on packet data to make the streams concatenable. The default is 1.

Currently, the only conversion is adding the h264_mp4toannexb bitstream filter to H.264 streams in MP4 format. This is necessary in particular if there are resolution changes.

`segment_time_metadata`

If set to 1, every packet will contain the *lavf.concat.start_time* and the *lavf.concat.duration* packet metadata values which are the start_time and the duration of the respective file segments in the concatenated output expressed in microseconds. The duration metadata is only set if it is known based on the concat file. The default is 0.

3.6 flv# TOC

Adobe Flash Video Format demuxer.

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

`-flv_metadata` *bool*

Allocate the streams according to the onMetaData array content.

3.7 libgme# TOC

The Game Music Emu library is a collection of video game music file emulators.

See http://code.google.com/p/game-music-emu/ for more information.

Some files have multiple tracks. The demuxer will pick the first track by default. The `track_index` option can be used to select a different track. Track indexes start at 0. The demuxer exports the number of tracks as *tracks* meta data entry.

For very large files, the `max_size` option may have to be adjusted.

3.8 libquvi# TOC

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.

3.9 gif# TOC

Animated GIF demuxer.

It accepts the following options:

`min_delay`

> Set the minimum valid delay between frames in hundredths of seconds. Range is 0 to 6000. Default value is 2.

`max_gif_delay`

> Set the maximum valid delay between frames in hundredth of seconds. Range is 0 to 65535. Default value is 65535 (nearly eleven minutes), the maximum value allowed by the specification.

`default_delay`

> Set the default delay between frames in hundredths of seconds. Range is 0 to 6000. Default value is 10.

`ignore_loop`

> GIF files can contain information to loop a certain number of times (or infinitely). If `ignore_loop` is set to 1, then the loop setting from the input will be ignored and looping will not occur. If set to 0, then looping will occur and will cycle the number of times according to the GIF. Default value is 1.

For example, with the overlay filter, place an infinitely looping GIF over another video:

```
ffmpeg -i input.mp4 -ignore_loop 0 -i input.gif -filter_complex overlay=shortest=1 out.mkv
```

Note that in the above example the shortest option for overlay filter is used to end the output video at the length of the shortest input file, which in this case is `input.mp4` as the GIF in this example loops infinitely.

3.10 image2# TOC

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.

`none`

Disable pattern matching, therefore the video will only contain the specified image. You should use this option if you do not want to create sequences from multiple images and your filenames may contain special pattern characters.

`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.

`ts_from_file`

If set to 1, will set frame timestamp to modification time of image file. Note that monotonity of timestamps is not provided: images go in the same order as without this option. Default value is 0. If set to 2, will set frame timestamp to the modification time of the image file in nanosecond precision.

`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.

3.10.1 Examples# TOC

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

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

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

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

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

3.11 mpegts# TOC

MPEG-2 transport stream demuxer.

This demuxer accepts the following options:

`resync_size`

Set size limit for looking up a new synchronization. Default value is 65536.

`fix_teletext_pts`

Override teletext packet PTS and DTS values with the timestamps calculated from the PCR of the first program which the teletext stream is part of and is not discarded. Default value is 1, set this option to 0 if you want your teletext packet PTS and DTS values untouched.

`ts_packetsize`

Output option carrying the raw packet size in bytes. Show the detected raw packet size, cannot be set by the user.

`scan_all_pmts`

Scan and combine all PMTs. The value is an integer with value from -1 to 1 (-1 means automatic setting, 1 means enabled, 0 means disabled). Default value is -1.

3.12 mpjpeg# TOC

MJPEG encapsulated in multi-part MIME demuxer.

This demuxer allows reading of MJPEG, where each frame is represented as a part of multipart/x-mixed-replace stream.

`strict_mime_boundary`

Default implementation applies a relaxed standard to multi-part MIME boundary detection, to prevent regression with numerous existing endpoints not generating a proper MIME MJPEG stream. Turning this option on by setting it to 1 will result in a stricter check of the boundary value.

3.13 rawvideo# TOC

Raw video demuxer.

This demuxer allows one 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
```

3.14 sbg# TOC

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.

3.15 tedcaptions# TOC

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# TOC

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 aiff# TOC

Audio Interchange File Format muxer.

4.1.1 Options# TOC

It accepts the following options:

`write_id3v2`

> Enable ID3v2 tags writing when set to 1. Default is 0 (disabled).

`id3v2_version`

> Select ID3v2 version to write. Currently only version 3 and 4 (aka. ID3v2.3 and ID3v2.4) are supported. The default is version 4.

4.2 chromaprint# TOC

Chromaprint fingerprinter

This muxer feeds audio data to the Chromaprint library, which generates a fingerprint for the provided audio data. It takes a single signed native-endian 16-bit raw audio stream.

4.2.1 Options# TOC

`silence_threshold`

> Threshold for detecting silence, ranges from 0 to 32767. -1 for default (required for use with the AcoustID service).

`algorithm`

Algorithm index to fingerprint with.

`fp_format`

Format to output the fingerprint as. Accepts the following options:

‘`raw`’

Binary raw fingerprint

‘`compressed`’

Binary compressed fingerprint

‘`base64`’

Base64 compressed fingerprint

4.3 crc# TOC

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.

See also the framecrc muxer.

4.3.1 Examples# TOC

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 -
```

4.4 framecrc# TOC

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*, 0x*CRC*

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

4.4.1 Examples# TOC

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.

4.5 framemd5# TOC

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.

4.5.1 Examples# TOC

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.

4.6 gif# TOC

Animated GIF muxer.

It accepts the following options:

`loop`

Set the number of times to loop the output. Use `-1` for no loop, `0` for looping indefinitely (default).

`final_delay`

Force the delay (expressed in centiseconds) after the last frame. Each frame ends with a delay until the next frame. The default is `-1`, which is a special value to tell the muxer to re-use the previous delay. In case of a loop, you might want to customize this value to mark a pause for instance.

For example, to encode a gif looping 10 times, with a 5 seconds delay between the loops:

```
ffmpeg -i INPUT -loop 10 -final_delay 500 out.gif
```

Note 1: if you wish to extract the frames in separate GIF files, you need to force the image2 muxer:

```
ffmpeg -i INPUT -c:v gif -f image2 "out%d.gif"
```

Note 2: the GIF format has a very small time base: the delay between two frames can not be smaller than one centi second.

4.7 hls# TOC

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

It creates a playlist file, and one or more segment files. The output filename specifies the playlist filename.

By default, the muxer creates a file for each segment produced. These files have the same name as the playlist, followed by a sequential number and a .ts extension.

For example, to convert an input file with `ffmpeg`:

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

This example will produce the playlist, `out.m3u8`, and segment files: `out0.ts`, `out1.ts`, `out2.ts`, etc.

See also the segment muxer, which provides a more generic and flexible implementation of a segmenter, and can be used to perform HLS segmentation.

4.7.1 Options# TOC

This muxer supports the following options:

`hls_time` *`seconds`*

Set the segment length in seconds. Default value is 2.

`hls_list_size` *`size`*

Set the maximum number of playlist entries. If set to 0 the list file will contain all the segments. Default value is 5.

`hls_ts_options` *`options_list`*

Set output format options using a :-separated list of key=value parameters. Values containing `:` special characters must be escaped.

`hls_wrap` *`wrap`*

Set the number after which the segment filename number (the number specified in each segment file) wraps. If set to 0 the number will be never wrapped. Default value is 0.

This option is useful to avoid to fill the disk with many segment files, and limits the maximum number of segment files written to disk to *wrap*.

`start_number` *`number`*

Start the playlist sequence number from *number*. Default value is 0.

`hls_allow_cache` *`allowcache`*

Explicitly set whether the client MAY (1) or MUST NOT (0) cache media segments.

`hls_base_url` *`baseurl`*

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

Note that the playlist sequence number must be unique for each segment and it is not to be confused with the segment filename sequence number which can be cyclic, for example if the `wrap` option is specified.

`hls_segment_filename` *`filename`*

Set the segment filename. Unless hls_flags single_file is set *filename* is used as a string format with the segment number:

```
ffmpeg in.nut -hls_segment_filename 'file%03d.ts' out.m3u8
```

This example will produce the playlist, `out.m3u8`, and segment files: `file000.ts`, `file001.ts`, `file002.ts`, etc.

`hls_key_info_file` *`key_info_file`*

Use the information in *key_info_file* for segment encryption. The first line of *key_info_file* specifies the key URI written to the playlist. The key URL is used to access the encryption key during playback. The second line specifies the path to the key file used to obtain the key during the encryption process. The key file is read as a single packed array of 16 octets in binary format. The optional third line specifies the initialization vector (IV) as a hexadecimal string to be used instead of the segment sequence number (default) for encryption. Changes to *key_info_file* will result in segment encryption with the new key/IV and an entry in the playlist for the new key URI/IV.

Key info file format:

```
key URI
key file path
IV (optional)
```

Example key URIs:

```
http://server/file.key
/path/to/file.key
file.key
```

Example key file paths:

```
file.key
/path/to/file.key
```

Example IV:

```
0123456789ABCDEF0123456789ABCDEF
```

Key info file example:

```
http://server/file.key
/path/to/file.key
0123456789ABCDEF0123456789ABCDEF
```

Example shell script:

```
#!/bin/sh
BASE_URL=${1:-'.'}
openssl rand 16 > file.key
echo $BASE_URL/file.key > file.keyinfo
echo file.key >> file.keyinfo
echo $(openssl rand -hex 16) >> file.keyinfo
ffmpeg -f lavfi -re -i testsrc -c:v h264 -hls_flags delete_segments \
  -hls_key_info_file file.keyinfo out.m3u8
```

`hls_flags single_file`

If this flag is set, the muxer will store all segments in a single MPEG-TS file, and will use byte ranges in the playlist. HLS playlists generated with this way will have the version number 4. For example:

```
ffmpeg -i in.nut -hls_flags single_file out.m3u8
```

Will produce the playlist, `out.m3u8`, and a single segment file, `out.ts`.

`hls_flags delete_segments`

Segment files removed from the playlist are deleted after a period of time equal to the duration of the segment plus the duration of the playlist.

4.8 ico# TOC

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

4.9 image2# TOC

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.

4.9.1 Examples# TOC

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
```

The `strftime` option allows you to expand the filename with date and time information. Check the documentation of the `strftime()` function for the syntax.

For example to generate image files from the `strftime()` "%Y-%m-%d_%H-%M-%S" pattern, the following `ffmpeg` command can be used:

```
ffmpeg -f v4l2 -r 1 -i /dev/video0 -f image2 -strftime 1 "%Y-%m-%d_%H-%M-%S.jpg"
```

4.9.2 Options# TOC

`start_number`

Start the sequence from the specified number. Default value is 0.

`update`

If set to 1, the filename will always be interpreted as just a filename, not a pattern, and the corresponding file will be continuously overwritten with new images. Default value is 0.

`strftime`

If set to 1, expand the filename with date and time information from `strftime()`. Default value is 0.

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.

4.10 matroska# TOC

Matroska container muxer.

This muxer implements the matroska and webm container specs.

4.10.1 Metadata# TOC

The recognized metadata settings in this muxer are:

`title`

Set title name provided to a single track.

`language`

Specify the language of the track in the Matroska languages form.

The language can be either the 3 letters bibliographic ISO-639-2 (ISO 639-2/B) form (like "fre" for French), or a language code mixed with a country code for specialities in languages (like "fre-ca" for Canadian French).

`stereo_mode`

Set stereo 3D video layout of two views in a single video track.

The following values are recognized:

‘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
```

4.10.2 Options# TOC

This muxer supports the following options:

`reserve_index_space`

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

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

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

4.11 md5# TOC

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. Timestamps are ignored.

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.

4.12 mov, mp4, ismv# TOC

MOV/MP4/ISMV (Smooth Streaming) muxer.

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.

4.12.1 Options# TOC

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.

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

`-movflags separate_moof`

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

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

`-movflags faststart`

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

`-movflags rtphint`

Add RTP hinting tracks to the output file.

`-movflags disable_chpl`

Disable Nero chapter markers (chpl atom). Normally, both Nero chapters and a QuickTime chapter track are written to the file. With this option set, only the QuickTime chapter track will be written. Nero chapters can cause failures when the file is reprocessed with certain tagging programs, like mp3Tag 2.61a and iTunes 11.3, most likely other versions are affected as well.

`-movflags omit_tfhd_offset`

Do not write any absolute base_data_offset in tfhd atoms. This avoids tying fragments to absolute byte positions in the file/streams.

`-movflags default_base_moof`

Similarly to the omit_tfhd_offset, this flag avoids writing the absolute base_data_offset field in tfhd atoms, but does so by using the new default-base-is-moof flag instead. This flag is new from 14496-12:2012. This may make the fragments easier to parse in certain circumstances (avoiding basing track fragment location calculations on the implicit end of the previous track fragment).

4.12.2 Example# TOC

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)
```

4.12.3 Audible AAX# TOC

Audible AAX files are encrypted M4B files, and they can be decrypted by specifying a 4 byte activation secret.

```
ffmpeg -activation_bytes 1CEB00DA -i test.aax -vn -c:a copy output.mp4
```

4.13 mp3# TOC

The MP3 muxer writes a raw MP3 stream with the following optional features:

- An ID3v2 metadata header at the beginning (enabled by default). Versions 2.3 and 2.4 are supported, the `id3v2_version` private option controls which one is used (3 or 4). Setting `id3v2_version` to 0 disables the ID3v2 header completely.

  The muxer supports writing attached pictures (APIC frames) to the ID3v2 header. 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.

- A Xing/LAME frame right after the ID3v2 header (if present). It is enabled by default, but will be written only if the output is seekable. The `write_xing` private option can be used to disable it. The frame contains various information that may be useful to the decoder, like the audio duration or encoder delay.
- A legacy ID3v1 tag at the end of the file (disabled by default). It may be enabled with the `write_id3v1` private option, but as its capabilities are very limited, its usage is not recommended.

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
```

Write a "clean" MP3 without any extra features:

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

4.14 mpegts# TOC

MPEG transport stream muxer.

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

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".

4.14.1 Options# TOC

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_service_type` *`number`*

Set the program service_type (default *digital_tv*), see below a list of pre defined values.

`-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).

`-mpegts_m2ts_mode` *number*

Enable m2ts mode if set to 1. Default value is -1 which disables m2ts mode.

`-muxrate` *number*

Set a constant muxrate (default VBR).

`-pcr_period` *numer*

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

`pat_period` *number*

Maximal time in seconds between PAT/PMT tables.

`sdt_period` *number*

Maximal time in seconds between SDT tables.

`-pes_payload_size` *number*

Set minimum PES packet payload in bytes.

`-mpegts_flags` *flags*

Set flags (see below).

`-mpegts_copyts` *number*

Preserve original timestamps, if value is set to 1. Default value is -1, which results in shifting timestamps so that they start from 0.

`-tables_version` *number*

Set PAT, PMT and SDT version (default 0, valid values are from 0 to 31, inclusively). This option allows updating stream structure so that standard consumer may detect the change. To do so, reopen output AVFormatContext (in case of API usage) or restart ffmpeg instance, cyclically changing tables_version value:

```
ffmpeg -i source1.ts -codec copy -f mpegts -tables_version 0 udp://1.1.1.1:1111
ffmpeg -i source2.ts -codec copy -f mpegts -tables_version 1 udp://1.1.1.1:1111
...
ffmpeg -i source3.ts -codec copy -f mpegts -tables_version 31 udp://1.1.1.1:1111
ffmpeg -i source1.ts -codec copy -f mpegts -tables_version 0 udp://1.1.1.1:1111
ffmpeg -i source2.ts -codec copy -f mpegts -tables_version 1 udp://1.1.1.1:1111
...
```

Option mpegts_service_type accepts the following values:

`hex_value`

Any hexdecimal value between 0x01 to 0xff as defined in ETSI 300 468.

`digital_tv`

Digital TV service.

`digital_radio`

Digital Radio service.

`teletext`

Teletext service.

`advanced_codec_digital_radio`

Advanced Codec Digital Radio service.

`mpeg2_digital_hdtv`

MPEG2 Digital HDTV service.

`advanced_codec_digital_sdtv`

Advanced Codec Digital SDTV service.

`advanced_codec_digital_hdtv`

Advanced Codec Digital HDTV service.

Option mpegts_flags may take a set of such flags:

`resend_headers`

Reemit PAT/PMT before writing the next packet.

`latm`

Use LATM packetization for AAC.

`pat_pmt_at_frames`

Reemit PAT and PMT at each video frame.

`system_b`

Conform to System B (DVB) instead of System A (ATSC).

4.14.2 Example# TOC

```
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
```

4.15 mxf, mxf_d10# TOC

MXF muxer.

4.15.1 Options# TOC

The muxer options are:

`store_user_comments` *`bool`*

Set if user comments should be stored if available or never. IRT D-10 does not allow user comments. The default is thus to write them for mxf but not for mxf_d10

4.16 null# TOC

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 -
```

4.17 nut# TOC

`-syncpoints` *`flags`*

Change the syncpoint usage in nut:

`default` use the normal low-overhead seeking aids.
`none` do not use the syncpoints at all, reducing the overhead but making the stream non-seekable;

Use of this option is not recommended, as the resulting files are very damage sensitive and seeking is not possible. Also in general the overhead from syncpoints is negligible. Note, -`write_index` 0 can be used to disable all growing data tables, allowing to mux endless streams with limited memory and without these disadvantages.

`timestamped` extend the syncpoint with a wallclock field.

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

`-write_index` *`bool`*

Write index at the end, the default is to write an index.

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

4.18 ogg# TOC

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.

`-serial_offset` *`value`*

Serial value from which to set the streams serial number. Setting it to different and sufficiently large values ensures that the produced ogg files can be safely chained.

4.19 segment, stream_segment, ssegment# TOC

Basic stream segmenter.

This 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, or by using a `strftime` template if the `strftime` option is enabled.

`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 entry filenames in the segment list are set by default to the basename of the corresponding segment files.

See also the hls muxer, which provides a more specific implementation for HLS segmentation.

4.19.1 Options# TOC

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 chosen 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_format_options` *`options_list`*

> Set output format options using a :-separated list of key=value parameters. Values containing the `:` special character must be escaped.

`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.

`segment_list_size` *`size`*

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

`segment_list_entry_prefix` *`prefix`*

Prepend *prefix* to each entry. Useful to generate absolute paths. By default no prefix is applied.

`segment_list_type` *`type`*

Select the listing format.

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_atclocktime *1|0*

If set to "1" split at regular clock time intervals starting from 00:00 o'clock. The *time* value specified in segment_time is used for setting the length of the splitting interval.

For example with segment_time set to "900" this makes it possible to create files at 12:00 o'clock, 12:15, 12:30, etc.

Default value is "0".

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`.

`strftime` *1|0*

Use the `strftime` function to define the name of the new segments to write. If this is selected, the output segment name must contain a `strftime` function template. Default value is `0`.

`break_non_keyframes` *1|0*

If enabled, allow segments to start on frames other than keyframes. This improves behavior on some players when the time between keyframes is inconsistent, but may make things worse on others, and can cause some oddities during seeking. 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.

`initial_offset` *offset*

Specify timestamp offset to apply to the output packet timestamps. The argument must be a time duration specification, and defaults to 0.

4.19.2 Examples# TOC

- 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
  ```

- Segment input and set output format options for the output segments:

  ```
  ffmpeg -i in.mkv -f segment -segment_time 10 -segment_format_options movflags=+faststart out%03d.mp4
  ```

- 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
  ```

- 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
  ```

- 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
  ```

4.20 smoothstreaming# TOC

Smooth Streaming muxer generates a set of files (Manifest, chunks) suitable for serving with conventional web server.

`window_size`

Specify the number of fragments kept in the manifest. Default 0 (keep all).

`extra_window_size`

Specify the number of fragments kept outside of the manifest before removing from disk. Default 5.

`lookahead_count`

Specify the number of lookahead fragments. Default 2.

`min_frag_duration`

Specify the minimum fragment duration (in microseconds). Default 5000000.

`remove_at_exit`

Specify whether to remove all fragments when finished. Default 0 (do not remove).

4.21 tee# TOC

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 (ffmpeg-utils)the "Quoting and escaping" section in the ffmpeg-utils(1) manual).

Muxer 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.

The following special options are also recognized:

`f`

Specify the format name. Useful if it cannot be guessed from the output name suffix.

`bsfs[/spec]`

Specify a list of bitstream filters to apply to the specified output.

It is possible to specify to which streams a given bitstream filter applies, by appending a stream specifier to the option separated by `/`. *spec* must be a stream specifier (see Format stream specifiers). If the stream specifier is not specified, the bitstream filters will be applied to all streams in the output.

Several bitstream filters can be specified, separated by ",".

`select`

Select the streams that should be mapped to the slave output, specified by a stream specifier. If not specified, this defaults to all the input streams. You may use multiple stream specifiers separated by commas (`,`) e.g.: `a:0,v`

4.21.1 Examples# TOC

- 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/"
```

- Use `ffmpeg` to encode the input, and send the output to three different destinations. The `dump_extra` bitstream filter is used to add extradata information to all the output video keyframes packets, as requested by the MPEG-TS format. The select option is applied to `out.aac` in order to make it contain only audio packets.

```
ffmpeg -i ... -map 0 -flags +global_header -c:v libx264 -c:a aac -strict experimental
       -f tee "[bsfs/v=dump_extra]out.ts|[movflags=+faststart]out.mp4|[select=a]out.aac"
```

- As below, but select only stream `a:1` for the audio output. Note that a second level escaping must be performed, as ":" is a special character used to separate options.

```
ffmpeg -i ... -map 0 -flags +global_header -c:v libx264 -c:a aac -strict experimental
       -f tee "[bsfs/v=dump_extra]out.ts|[movflags=+faststart]out.mp4|[select=\'a:1\']out.aac"
```

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.

4.22 webm_dash_manifest# TOC

WebM DASH Manifest muxer.

This muxer implements the WebM DASH Manifest specification to generate the DASH manifest XML. It also supports manifest generation for DASH live streams.

For more information see:

- WebM DASH Specification: https://sites.google.com/a/webmproject.org/wiki/adaptive-streaming/webm-dash-specification
- ISO DASH Specification: http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

4.22.1 Options# TOC

This muxer supports the following options:

`adaptation_sets`

This option has the following syntax: "id=x,streams=a,b,c id=y,streams=d,e" where x and y are the unique identifiers of the adaptation sets and a,b,c,d and e are the indices of the corresponding audio and video streams. Any number of adaptation sets can be added using this option.

`live`

Set this to 1 to create a live stream DASH Manifest. Default: 0.

`chunk_start_index`

Start index of the first chunk. This will go in the '`startNumber`' attribute of the '`SegmentTemplate`' element in the manifest. Default: 0.

`chunk_duration_ms`

Duration of each chunk in milliseconds. This will go in the '`duration`' attribute of the '`SegmentTemplate`' element in the manifest. Default: 1000.

`utc_timing_url`

URL of the page that will return the UTC timestamp in ISO format. This will go in the '`value`' attribute of the '`UTCTiming`' element in the manifest. Default: None.

`time_shift_buffer_depth`

Smallest time (in seconds) shifting buffer for which any Representation is guaranteed to be available. This will go in the '`timeShiftBufferDepth`' attribute of the '`MPD`' element. Default: 60.

`minimum_update_period`

Minimum update period (in seconds) of the manifest. This will go in the '`minimumUpdatePeriod`' attribute of the '`MPD`' element. Default: 0.

4.22.2 Example# TOC

```
ffmpeg -f webm_dash_manifest -i video1.webm \
       -f webm_dash_manifest -i video2.webm \
       -f webm_dash_manifest -i audio1.webm \
       -f webm_dash_manifest -i audio2.webm \
       -map 0 -map 1 -map 2 -map 3 \
       -c copy \
       -f webm_dash_manifest \
       -adaptation_sets "id=0,streams=0,1 id=1,streams=2,3" \
       manifest.xml
```

4.23 webm_chunk# TOC

WebM Live Chunk Muxer.

This muxer writes out WebM headers and chunks as separate files which can be consumed by clients that support WebM Live streams via DASH.

4.23.1 Options# TOC

This muxer supports the following options:

`chunk_start_index`

Index of the first chunk (defaults to 0).

`header`

Filename of the header where the initialization data will be written.

`audio_chunk_duration`

Duration of each audio chunk in milliseconds (defaults to 5000).

4.23.2 Example# TOC

```
ffmpeg -f v4l2 -i /dev/video0 \
       -f alsa -i hw:0 \
       -map 0:0 \
       -c:v libvpx-vp9 \
       -s 640x360 -keyint_min 30 -g 30 \
       -f webm_chunk \
       -header webm_live_video_360.hdr \
       -chunk_start_index 1 \
       webm_live_video_360_%d.chk \
       -map 1:0 \
       -c:a libvorbis \
       -b:a 128k \
       -f webm_chunk \
       -header webm_live_audio_128.hdr \
       -chunk_start_index 1 \
       -audio_chunk_duration 1000 \
       webm_live_audio_128_%d.chk
```

5 Metadata# TOC

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
```

By using the ffmetadata muxer and demuxer it is possible to extract metadata from an input file to an ffmetadata file, and then transcode the file into an output file with the edited ffmetadata file.

Extracting an ffmetadata file with `ffmpeg` goes as follows:

```
ffmpeg -i INPUT -f ffmetadata FFMETADATAFILE
```

Reinserting edited metadata information from the FFMETADATAFILE file can be done as:

```
ffmpeg -i INPUT -i FFMETADATAFILE -map_metadata 1 -codec copy OUTPUT
```

6 See Also# TOC

ffmpeg, ffplay, ffprobe, ffserver, libavformat

7 Authors# TOC

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 using *makeinfo*.

FFmpeg Formats Documentation

Table of Contents

1 Description# TOC

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

2 Format Options# TOC

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 enable detecting 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.

‘`fastseek`’

Enable fast, but inaccurate seeks for some formats.

‘`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

‘`bitexact`’

Only write platform-, build- and time-independent data. This ensures that file and data checksums are reproducible and match between platforms. Its primary use is for regression testing.

`seek2any` *`integer`* (*`input`*)

Allow seeking to non-keyframes on demuxer level when supported if set to 1. Default is 0.

`analyzeduration` *`integer`* (*`input`*)

Specify how many microseconds are analyzed to probe the input. A higher value will enable detecting 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.

`max_interleave_delta` *`integer`* (*`output`*)

Set maximum buffering duration for interleaving. The duration is expressed in microseconds, and defaults to 1000000 (1 second).

To ensure all the streams are interleaved correctly, libavformat will wait until it has at least one packet for each stream before actually writing any packets to the output file. When some streams are "sparse" (i.e. there are large gaps between successive packets), this can result in excessive buffering.

This field specifies the maximum difference between the timestamps of the first and the last packet in the muxing queue, above which libavformat will output a packet regardless of whether it has queued a packet for all the streams.

If set to 0, libavformat will continue buffering packets until it has a packet for each stream, regardless of the maximum timestamp difference between the buffered packets.

`use_wallclock_as_timestamps` *`integer`* (*`input`*)

Use wallclock as timestamps.

`avoid_negative_ts` *`integer`* (*`output`*)

Possible values:

‘make_non_negative’

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

‘make_zero’

Shift timestamps so that the first timestamp is 0.

‘auto (default)’

Enables shifting when required by the target format.

‘disabled’

Disables shifting of timestamp.

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.

`skip_initial_bytes` *integer* (*input*)

Set number of bytes to skip before reading header and frames if set to 1. Default is 0.

`correct_ts_overflow` *integer* (*input*)

Correct single timestamp overflows if set to 1. Default is 1.

`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.

`output_ts_offset` *offset* (*output*)

Set the output time offset.

offset must be a time duration specification, see (ffmpeg-utils)the Time duration section in the ffmpeg-utils(1) manual.

The offset is added by the muxer to the output timestamps.

Specifying a positive offset means that the corresponding streams are delayed bt the time duration specified in *offset*. Default value is `0` (meaning that no offset is applied).

`format_whitelist` *list* (*input*)

"," separated List of allowed demuxers. By default all are allowed.

`dump_separator` *`string`* (*`input`*)

Separator used to separate the fields printed on the command line about the Stream parameters. For example to separate the fields with newlines and indention:

```
ffprobe -dump_separator "
                         "  -i ~/videos/matrixbench_mpeg2.mpg
```

2.1 Format stream specifiers# TOC

Format stream specifiers allow selection of one or more streams that match specific properties.

Possible forms of stream specifiers are:

`stream_index`

Matches the stream with this index.

`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 the 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.

The exact semantics of stream specifiers is defined by the `avformat_match_stream_specifier()` function declared in the `libavformat/avformat.h` header.

3 Demuxers# TOC

Demuxers are configured elements in FFmpeg that can 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 aa# TOC

Audible Format 2, 3, and 4 demuxer.

This demuxer is used to demux Audible Format 2, 3, and 4 (.aa) files.

3.2 applehttp# TOC

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".

3.3 apng# TOC

Animated Portable Network Graphics demuxer.

This demuxer is used to demux APNG files. All headers, but the PNG signature, up to (but not including) the first fcTL chunk are transmitted as extradata. Frames are then split as being all the chunks between two fcTL ones, or between the last fcTL and IEND chunks.

`-ignore_loop` *`bool`*

Ignore the loop variable in the file if set.

`-max_fps` *`int`*

Maximum framerate in frames per second (0 for no limit).

`-default_fps` *`int`*

Default framerate in frames per second when none is specified in the file (0 meaning as fast as possible).

3.4 asf# TOC

Advanced Systems Format demuxer.

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

`-no_resync_search` *`bool`*

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

3.5 concat# TOC

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.

3.5.1 Syntax# TOC

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 file-related 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.

`inpoint` *`timestamp`*

In point of the file. When the demuxer opens the file it instantly seeks to the specified timestamp. Seeking is done so that all streams can be presented successfully at In point.

This directive works best with intra frame codecs, because for non-intra frame ones you will usually get extra packets before the actual In point and the decoded content will most likely contain frames before In point too.

For each file, packets before the file In point will have timestamps less than the calculated start timestamp of the file (negative in case of the first file), and the duration of the files (if not specified by the `duration` directive) will be reduced based on their specified In point.

Because of potential packets before the specified In point, packet timestamps may overlap between two concatenated files.

`outpoint` *`timestamp`*

Out point of the file. When the demuxer reaches the specified decoding timestamp in any of the streams, it handles it as an end of file condition and skips the current and all the remaining packets from all streams.

Out point is exclusive, which means that the demuxer will not output packets with a decoding timestamp greater or equal to Out point.

This directive works best with intra frame codecs and formats where all streams are tightly interleaved. For non-intra frame codecs you will usually get additional packets with presentation timestamp after Out point therefore the decoded content will most likely contain frames after Out point too. If your streams are not tightly interleaved you may not get all the packets from all streams before Out point and you may only will be able to decode the earliest stream until Out point.

The duration of the files (if not specified by the `duration` directive) will be reduced based on their specified Out point.

`file_packet_metadata` *`key=value`*

Metadata of the packets of the file. The specified metadata will be set for each file packet. You can specify this directive multiple times to add multiple metadata entries.

`stream`

Introduce a stream in the virtual file. All subsequent stream-related directives apply to the last introduced stream. Some streams properties must be set in order to allow identifying the matching streams in the subfiles. If no streams are defined in the script, the streams from the first file are

copied.

`exact_stream_id` *id*

Set the id of the stream. If this directive is given, the string with the corresponding id in the subfiles will be used. This is especially useful for MPEG-PS (VOB) files, where the order of the streams is not reliable.

3.5.2 Options# TOC

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.

`auto_convert`

If set to 1, try to perform automatic conversions on packet data to make the streams concatenable. The default is 1.

Currently, the only conversion is adding the h264_mp4toannexb bitstream filter to H.264 streams in MP4 format. This is necessary in particular if there are resolution changes.

`segment_time_metadata`

If set to 1, every packet will contain the *lavf.concat.start_time* and the *lavf.concat.duration* packet metadata values which are the start_time and the duration of the respective file segments in the concatenated output expressed in microseconds. The duration metadata is only set if it is known based on the concat file. The default is 0.

3.6 flv# TOC

Adobe Flash Video Format demuxer.

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

`-flv_metadata` *bool*

Allocate the streams according to the onMetaData array content.

3.7 libgme# TOC

The Game Music Emu library is a collection of video game music file emulators.

See http://code.google.com/p/game-music-emu/ for more information.

Some files have multiple tracks. The demuxer will pick the first track by default. The `track_index` option can be used to select a different track. Track indexes start at 0. The demuxer exports the number of tracks as *tracks* meta data entry.

For very large files, the `max_size` option may have to be adjusted.

3.8 libquvi# TOC

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.

3.9 gif# TOC

Animated GIF demuxer.

It accepts the following options:

`min_delay`

> Set the minimum valid delay between frames in hundredths of seconds. Range is 0 to 6000. Default value is 2.

`max_gif_delay`

> Set the maximum valid delay between frames in hundredth of seconds. Range is 0 to 65535. Default value is 65535 (nearly eleven minutes), the maximum value allowed by the specification.

`default_delay`

> Set the default delay between frames in hundredths of seconds. Range is 0 to 6000. Default value is 10.

`ignore_loop`

> GIF files can contain information to loop a certain number of times (or infinitely). If `ignore_loop` is set to 1, then the loop setting from the input will be ignored and looping will not occur. If set to 0, then looping will occur and will cycle the number of times according to the GIF. Default value is 1.

For example, with the overlay filter, place an infinitely looping GIF over another video:

```
ffmpeg -i input.mp4 -ignore_loop 0 -i input.gif -filter_complex overlay=shortest=1 out.mkv
```

Note that in the above example the shortest option for overlay filter is used to end the output video at the length of the shortest input file, which in this case is `input.mp4` as the GIF in this example loops infinitely.

3.10 image2# TOC

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.

`none`

Disable pattern matching, therefore the video will only contain the specified image. You should use this option if you do not want to create sequences from multiple images and your filenames may contain special pattern characters.

`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.

`ts_from_file`

If set to 1, will set frame timestamp to modification time of image file. Note that monotonity of timestamps is not provided: images go in the same order as without this option. Default value is 0. If set to 2, will set frame timestamp to the modification time of the image file in nanosecond precision.

`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.

3.10.1 Examples# TOC

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

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

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

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

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

3.11 mpegts# TOC

MPEG-2 transport stream demuxer.

This demuxer accepts the following options:

`resync_size`

Set size limit for looking up a new synchronization. Default value is 65536.

`fix_teletext_pts`

Override teletext packet PTS and DTS values with the timestamps calculated from the PCR of the first program which the teletext stream is part of and is not discarded. Default value is 1, set this option to 0 if you want your teletext packet PTS and DTS values untouched.

`ts_packetsize`

Output option carrying the raw packet size in bytes. Show the detected raw packet size, cannot be set by the user.

`scan_all_pmts`

Scan and combine all PMTs. The value is an integer with value from -1 to 1 (-1 means automatic setting, 1 means enabled, 0 means disabled). Default value is -1.

3.12 mpjpeg# TOC

MJPEG encapsulated in multi-part MIME demuxer.

This demuxer allows reading of MJPEG, where each frame is represented as a part of multipart/x-mixed-replace stream.

`strict_mime_boundary`

Default implementation applies a relaxed standard to multi-part MIME boundary detection, to prevent regression with numerous existing endpoints not generating a proper MIME MJPEG stream. Turning this option on by setting it to 1 will result in a stricter check of the boundary value.

3.13 rawvideo# TOC

Raw video demuxer.

This demuxer allows one 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
```

3.14 sbg# TOC

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.

3.15 tedcaptions# TOC

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# TOC

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 aiff# TOC

Audio Interchange File Format muxer.

4.1.1 Options# TOC

It accepts the following options:

`write_id3v2`

> Enable ID3v2 tags writing when set to 1. Default is 0 (disabled).

`id3v2_version`

> Select ID3v2 version to write. Currently only version 3 and 4 (aka. ID3v2.3 and ID3v2.4) are supported. The default is version 4.

4.2 chromaprint# TOC

Chromaprint fingerprinter

This muxer feeds audio data to the Chromaprint library, which generates a fingerprint for the provided audio data. It takes a single signed native-endian 16-bit raw audio stream.

4.2.1 Options# TOC

`silence_threshold`

> Threshold for detecting silence, ranges from 0 to 32767. -1 for default (required for use with the AcoustID service).

`algorithm`

Algorithm index to fingerprint with.

`fp_format`

Format to output the fingerprint as. Accepts the following options:

‘`raw`’

Binary raw fingerprint

‘`compressed`’

Binary compressed fingerprint

‘`base64`’

Base64 compressed fingerprint

4.3 crc# TOC

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.

See also the framecrc muxer.

4.3.1 Examples# TOC

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 -
```

4.4 framecrc# TOC

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*, 0x*CRC*

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

4.4.1 Examples# TOC

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.

4.5 framemd5# TOC

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.

4.5.1 Examples# TOC

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.

4.6 gif# TOC

Animated GIF muxer.

It accepts the following options:

`loop`

Set the number of times to loop the output. Use `-1` for no loop, `0` for looping indefinitely (default).

`final_delay`

Force the delay (expressed in centiseconds) after the last frame. Each frame ends with a delay until the next frame. The default is `-1`, which is a special value to tell the muxer to re-use the previous delay. In case of a loop, you might want to customize this value to mark a pause for instance.

For example, to encode a gif looping 10 times, with a 5 seconds delay between the loops:

```
ffmpeg -i INPUT -loop 10 -final_delay 500 out.gif
```

Note 1: if you wish to extract the frames in separate GIF files, you need to force the image2 muxer:

```
ffmpeg -i INPUT -c:v gif -f image2 "out%d.gif"
```

Note 2: the GIF format has a very small time base: the delay between two frames can not be smaller than one centi second.

4.7 hls# TOC

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

It creates a playlist file, and one or more segment files. The output filename specifies the playlist filename.

By default, the muxer creates a file for each segment produced. These files have the same name as the playlist, followed by a sequential number and a .ts extension.

For example, to convert an input file with `ffmpeg`:

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

This example will produce the playlist, `out.m3u8`, and segment files: `out0.ts`, `out1.ts`, `out2.ts`, etc.

See also the segment muxer, which provides a more generic and flexible implementation of a segmenter, and can be used to perform HLS segmentation.

4.7.1 Options# TOC

This muxer supports the following options:

`hls_time` *`seconds`*

Set the segment length in seconds. Default value is 2.

`hls_list_size` *`size`*

Set the maximum number of playlist entries. If set to 0 the list file will contain all the segments. Default value is 5.

`hls_ts_options` *`options_list`*

Set output format options using a :-separated list of key=value parameters. Values containing `:` special characters must be escaped.

`hls_wrap` *`wrap`*

Set the number after which the segment filename number (the number specified in each segment file) wraps. If set to 0 the number will be never wrapped. Default value is 0.

This option is useful to avoid to fill the disk with many segment files, and limits the maximum number of segment files written to disk to *wrap*.

`start_number` *`number`*

Start the playlist sequence number from *number*. Default value is 0.

`hls_allow_cache` *`allowcache`*

Explicitly set whether the client MAY (1) or MUST NOT (0) cache media segments.

`hls_base_url` *`baseurl`*

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

Note that the playlist sequence number must be unique for each segment and it is not to be confused with the segment filename sequence number which can be cyclic, for example if the `wrap` option is specified.

`hls_segment_filename` *`filename`*

Set the segment filename. Unless hls_flags single_file is set *filename* is used as a string format with the segment number:

```
ffmpeg in.nut -hls_segment_filename 'file%03d.ts' out.m3u8
```

This example will produce the playlist, `out.m3u8`, and segment files: `file000.ts`, `file001.ts`, `file002.ts`, etc.

`hls_key_info_file` *`key_info_file`*

Use the information in *key_info_file* for segment encryption. The first line of *key_info_file* specifies the key URI written to the playlist. The key URL is used to access the encryption key during playback. The second line specifies the path to the key file used to obtain the key during the encryption process. The key file is read as a single packed array of 16 octets in binary format. The optional third line specifies the initialization vector (IV) as a hexadecimal string to be used instead of the segment sequence number (default) for encryption. Changes to *key_info_file* will result in segment encryption with the new key/IV and an entry in the playlist for the new key URI/IV.

Key info file format:

```
key URI
key file path
IV (optional)
```

Example key URIs:

```
http://server/file.key
/path/to/file.key
file.key
```

Example key file paths:

```
file.key
/path/to/file.key
```

Example IV:

```
0123456789ABCDEF0123456789ABCDEF
```

Key info file example:

```
http://server/file.key
/path/to/file.key
0123456789ABCDEF0123456789ABCDEF
```

Example shell script:

```
#!/bin/sh
BASE_URL=${1:-'.'}
openssl rand 16 > file.key
echo $BASE_URL/file.key > file.keyinfo
echo file.key >> file.keyinfo
echo $(openssl rand -hex 16) >> file.keyinfo
ffmpeg -f lavfi -re -i testsrc -c:v h264 -hls_flags delete_segments \
  -hls_key_info_file file.keyinfo out.m3u8
```

`hls_flags single_file`

If this flag is set, the muxer will store all segments in a single MPEG-TS file, and will use byte ranges in the playlist. HLS playlists generated with this way will have the version number 4. For example:

```
ffmpeg -i in.nut -hls_flags single_file out.m3u8
```

Will produce the playlist, `out.m3u8`, and a single segment file, `out.ts`.

`hls_flags delete_segments`

Segment files removed from the playlist are deleted after a period of time equal to the duration of the segment plus the duration of the playlist.

4.8 ico# TOC

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

4.9 image2# TOC

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.

4.9.1 Examples# TOC

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
```

The `strftime` option allows you to expand the filename with date and time information. Check the documentation of the `strftime()` function for the syntax.

For example to generate image files from the `strftime()` "%Y-%m-%d_%H-%M-%S" pattern, the following `ffmpeg` command can be used:

```
ffmpeg -f v4l2 -r 1 -i /dev/video0 -f image2 -strftime 1 "%Y-%m-%d_%H-%M-%S.jpg"
```

4.9.2 Options# TOC

`start_number`

Start the sequence from the specified number. Default value is 0.

`update`

If set to 1, the filename will always be interpreted as just a filename, not a pattern, and the corresponding file will be continuously overwritten with new images. Default value is 0.

`strftime`

If set to 1, expand the filename with date and time information from `strftime()`. Default value is 0.

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.

4.10 matroska# TOC

Matroska container muxer.

This muxer implements the matroska and webm container specs.

4.10.1 Metadata# TOC

The recognized metadata settings in this muxer are:

`title`

Set title name provided to a single track.

`language`

Specify the language of the track in the Matroska languages form.

The language can be either the 3 letters bibliographic ISO-639-2 (ISO 639-2/B) form (like "fre" for French), or a language code mixed with a country code for specialities in languages (like "fre-ca" for Canadian French).

`stereo_mode`

Set stereo 3D video layout of two views in a single video track.

The following values are recognized:

‘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
```

4.10.2 Options# TOC

This muxer supports the following options:

`reserve_index_space`

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

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

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

4.11 md5# TOC

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. Timestamps are ignored.

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.

4.12 mov, mp4, ismv# TOC

MOV/MP4/ISMV (Smooth Streaming) muxer.

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.

4.12.1 Options# TOC

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.

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

`-movflags separate_moof`

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

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

`-movflags faststart`

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

`-movflags rtphint`

Add RTP hinting tracks to the output file.

`-movflags disable_chpl`

Disable Nero chapter markers (chpl atom). Normally, both Nero chapters and a QuickTime chapter track are written to the file. With this option set, only the QuickTime chapter track will be written. Nero chapters can cause failures when the file is reprocessed with certain tagging programs, like mp3Tag 2.61a and iTunes 11.3, most likely other versions are affected as well.

`-movflags omit_tfhd_offset`

Do not write any absolute base_data_offset in tfhd atoms. This avoids tying fragments to absolute byte positions in the file/streams.

`-movflags default_base_moof`

Similarly to the omit_tfhd_offset, this flag avoids writing the absolute base_data_offset field in tfhd atoms, but does so by using the new default-base-is-moof flag instead. This flag is new from 14496-12:2012. This may make the fragments easier to parse in certain circumstances (avoiding basing track fragment location calculations on the implicit end of the previous track fragment).

4.12.2 Example# TOC

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)
```

4.12.3 Audible AAX# TOC

Audible AAX files are encrypted M4B files, and they can be decrypted by specifying a 4 byte activation secret.

```
ffmpeg -activation_bytes 1CEB00DA -i test.aax -vn -c:a copy output.mp4
```

4.13 mp3# TOC

The MP3 muxer writes a raw MP3 stream with the following optional features:

- An ID3v2 metadata header at the beginning (enabled by default). Versions 2.3 and 2.4 are supported, the `id3v2_version` private option controls which one is used (3 or 4). Setting `id3v2_version` to 0 disables the ID3v2 header completely.

  The muxer supports writing attached pictures (APIC frames) to the ID3v2 header. 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.

- A Xing/LAME frame right after the ID3v2 header (if present). It is enabled by default, but will be written only if the output is seekable. The `write_xing` private option can be used to disable it. The frame contains various information that may be useful to the decoder, like the audio duration or encoder delay.
- A legacy ID3v1 tag at the end of the file (disabled by default). It may be enabled with the `write_id3v1` private option, but as its capabilities are very limited, its usage is not recommended.

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
```

Write a "clean" MP3 without any extra features:

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

4.14 mpegts# TOC

MPEG transport stream muxer.

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

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".

4.14.1 Options# TOC

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_service_type` *`number`*

Set the program service_type (default *digital_tv*), see below a list of pre defined values.

`-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).

`-mpegts_m2ts_mode` *number*

Enable m2ts mode if set to 1. Default value is -1 which disables m2ts mode.

`-muxrate` *number*

Set a constant muxrate (default VBR).

`-pcr_period` *numer*

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

`pat_period` *number*

Maximal time in seconds between PAT/PMT tables.

`sdt_period` *number*

Maximal time in seconds between SDT tables.

`-pes_payload_size` *number*

Set minimum PES packet payload in bytes.

`-mpegts_flags` *flags*

Set flags (see below).

`-mpegts_copyts` *number*

Preserve original timestamps, if value is set to 1. Default value is -1, which results in shifting timestamps so that they start from 0.

`-tables_version` *number*

Set PAT, PMT and SDT version (default 0, valid values are from 0 to 31, inclusively). This option allows updating stream structure so that standard consumer may detect the change. To do so, reopen output AVFormatContext (in case of API usage) or restart ffmpeg instance, cyclically changing tables_version value:

```
ffmpeg -i source1.ts -codec copy -f mpegts -tables_version 0 udp://1.1.1.1:1111
ffmpeg -i source2.ts -codec copy -f mpegts -tables_version 1 udp://1.1.1.1:1111
...
ffmpeg -i source3.ts -codec copy -f mpegts -tables_version 31 udp://1.1.1.1:1111
ffmpeg -i source1.ts -codec copy -f mpegts -tables_version 0 udp://1.1.1.1:1111
ffmpeg -i source2.ts -codec copy -f mpegts -tables_version 1 udp://1.1.1.1:1111
...
```

Option mpegts_service_type accepts the following values:

`hex_value`

Any hexdecimal value between 0x01 to 0xff as defined in ETSI 300 468.

`digital_tv`

Digital TV service.

`digital_radio`

Digital Radio service.

`teletext`

Teletext service.

`advanced_codec_digital_radio`

Advanced Codec Digital Radio service.

`mpeg2_digital_hdtv`

MPEG2 Digital HDTV service.

`advanced_codec_digital_sdtv`

Advanced Codec Digital SDTV service.

`advanced_codec_digital_hdtv`

Advanced Codec Digital HDTV service.

Option mpegts_flags may take a set of such flags:

`resend_headers`

Reemit PAT/PMT before writing the next packet.

`latm`

Use LATM packetization for AAC.

`pat_pmt_at_frames`

Reemit PAT and PMT at each video frame.

`system_b`

Conform to System B (DVB) instead of System A (ATSC).

4.14.2 Example# TOC

```
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
```

4.15 mxf, mxf_d10# TOC

MXF muxer.

4.15.1 Options# TOC

The muxer options are:

`store_user_comments` *`bool`*

Set if user comments should be stored if available or never. IRT D-10 does not allow user comments. The default is thus to write them for mxf but not for mxf_d10

4.16 null# TOC

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 -
```

4.17 nut# TOC

`-syncpoints` *`flags`*

Change the syncpoint usage in nut:

`default` use the normal low-overhead seeking aids.
`none` do not use the syncpoints at all, reducing the overhead but making the stream non-seekable;

Use of this option is not recommended, as the resulting files are very damage sensitive and seeking is not possible. Also in general the overhead from syncpoints is negligible. Note, -`write_index` 0 can be used to disable all growing data tables, allowing to mux endless streams with limited memory and without these disadvantages.

`timestamped` extend the syncpoint with a wallclock field.

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

`-write_index` *`bool`*

Write index at the end, the default is to write an index.

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

4.18 ogg# TOC

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.

`-serial_offset` *`value`*

Serial value from which to set the streams serial number. Setting it to different and sufficiently large values ensures that the produced ogg files can be safely chained.

4.19 segment, stream_segment, ssegment# TOC

Basic stream segmenter.

This 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, or by using a `strftime` template if the `strftime` option is enabled.

`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 entry filenames in the segment list are set by default to the basename of the corresponding segment files.

See also the hls muxer, which provides a more specific implementation for HLS segmentation.

4.19.1 Options# TOC

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 chosen 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_format_options` *`options_list`*

> Set output format options using a :-separated list of key=value parameters. Values containing the `:` special character must be escaped.

`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.

`segment_list_size` *`size`*

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

`segment_list_entry_prefix` *`prefix`*

Prepend *prefix* to each entry. Useful to generate absolute paths. By default no prefix is applied.

`segment_list_type` *`type`*

Select the listing format.

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_atclocktime *1|0*

If set to "1" split at regular clock time intervals starting from 00:00 o'clock. The *time* value specified in segment_time is used for setting the length of the splitting interval.

For example with segment_time set to "900" this makes it possible to create files at 12:00 o'clock, 12:15, 12:30, etc.

Default value is "0".

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`.

`strftime` *1|0*

Use the `strftime` function to define the name of the new segments to write. If this is selected, the output segment name must contain a `strftime` function template. Default value is `0`.

`break_non_keyframes` *1|0*

If enabled, allow segments to start on frames other than keyframes. This improves behavior on some players when the time between keyframes is inconsistent, but may make things worse on others, and can cause some oddities during seeking. 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.

`initial_offset` *offset*

Specify timestamp offset to apply to the output packet timestamps. The argument must be a time duration specification, and defaults to 0.

4.19.2 Examples# TOC

- 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
  ```

- Segment input and set output format options for the output segments:

  ```
  ffmpeg -i in.mkv -f segment -segment_time 10 -segment_format_options movflags=+faststart out%03d.mp4
  ```

- 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
  ```

- 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
  ```

- 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
  ```

4.20 smoothstreaming# TOC

Smooth Streaming muxer generates a set of files (Manifest, chunks) suitable for serving with conventional web server.

`window_size`

Specify the number of fragments kept in the manifest. Default 0 (keep all).

`extra_window_size`

Specify the number of fragments kept outside of the manifest before removing from disk. Default 5.

`lookahead_count`

Specify the number of lookahead fragments. Default 2.

`min_frag_duration`

Specify the minimum fragment duration (in microseconds). Default 5000000.

`remove_at_exit`

Specify whether to remove all fragments when finished. Default 0 (do not remove).

4.21 tee# TOC

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 (ffmpeg-utils)the "Quoting and escaping" section in the ffmpeg-utils(1) manual).

Muxer 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.

The following special options are also recognized:

`f`

Specify the format name. Useful if it cannot be guessed from the output name suffix.

`bsfs[/spec]`

Specify a list of bitstream filters to apply to the specified output.

It is possible to specify to which streams a given bitstream filter applies, by appending a stream specifier to the option separated by `/`. *spec* must be a stream specifier (see Format stream specifiers). If the stream specifier is not specified, the bitstream filters will be applied to all streams in the output.

Several bitstream filters can be specified, separated by ",".

`select`

Select the streams that should be mapped to the slave output, specified by a stream specifier. If not specified, this defaults to all the input streams. You may use multiple stream specifiers separated by commas (`,`) e.g.: `a:0,v`

4.21.1 Examples# TOC

- 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/"
```

- Use `ffmpeg` to encode the input, and send the output to three different destinations. The `dump_extra` bitstream filter is used to add extradata information to all the output video keyframes packets, as requested by the MPEG-TS format. The select option is applied to `out.aac` in order to make it contain only audio packets.

```
ffmpeg -i ... -map 0 -flags +global_header -c:v libx264 -c:a aac -strict experimental
       -f tee "[bsfs/v=dump_extra]out.ts|[movflags=+faststart]out.mp4|[select=a]out.aac"
```

- As below, but select only stream `a:1` for the audio output. Note that a second level escaping must be performed, as ":" is a special character used to separate options.

```
ffmpeg -i ... -map 0 -flags +global_header -c:v libx264 -c:a aac -strict experimental
       -f tee "[bsfs/v=dump_extra]out.ts|[movflags=+faststart]out.mp4|[select=\'a:1\']out.aac"
```

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.

4.22 webm_dash_manifest# TOC

WebM DASH Manifest muxer.

This muxer implements the WebM DASH Manifest specification to generate the DASH manifest XML. It also supports manifest generation for DASH live streams.

For more information see:

- WebM DASH Specification: https://sites.google.com/a/webmproject.org/wiki/adaptive-streaming/webm-dash-specification
- ISO DASH Specification: http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

4.22.1 Options# TOC

This muxer supports the following options:

`adaptation_sets`

This option has the following syntax: "id=x,streams=a,b,c id=y,streams=d,e" where x and y are the unique identifiers of the adaptation sets and a,b,c,d and e are the indices of the corresponding audio and video streams. Any number of adaptation sets can be added using this option.

`live`

Set this to 1 to create a live stream DASH Manifest. Default: 0.

`chunk_start_index`

Start index of the first chunk. This will go in the '`startNumber`' attribute of the '`SegmentTemplate`' element in the manifest. Default: 0.

`chunk_duration_ms`

Duration of each chunk in milliseconds. This will go in the '`duration`' attribute of the '`SegmentTemplate`' element in the manifest. Default: 1000.

`utc_timing_url`

URL of the page that will return the UTC timestamp in ISO format. This will go in the '`value`' attribute of the '`UTCTiming`' element in the manifest. Default: None.

`time_shift_buffer_depth`

Smallest time (in seconds) shifting buffer for which any Representation is guaranteed to be available. This will go in the '`timeShiftBufferDepth`' attribute of the '`MPD`' element. Default: 60.

`minimum_update_period`

Minimum update period (in seconds) of the manifest. This will go in the '`minimumUpdatePeriod`' attribute of the '`MPD`' element. Default: 0.

4.22.2 Example# TOC

```
ffmpeg -f webm_dash_manifest -i video1.webm \
       -f webm_dash_manifest -i video2.webm \
       -f webm_dash_manifest -i audio1.webm \
       -f webm_dash_manifest -i audio2.webm \
       -map 0 -map 1 -map 2 -map 3 \
       -c copy \
       -f webm_dash_manifest \
       -adaptation_sets "id=0,streams=0,1 id=1,streams=2,3" \
       manifest.xml
```

4.23 webm_chunk# TOC

WebM Live Chunk Muxer.

This muxer writes out WebM headers and chunks as separate files which can be consumed by clients that support WebM Live streams via DASH.

4.23.1 Options# TOC

This muxer supports the following options:

`chunk_start_index`

Index of the first chunk (defaults to 0).

`header`

Filename of the header where the initialization data will be written.

`audio_chunk_duration`

Duration of each audio chunk in milliseconds (defaults to 5000).

4.23.2 Example# TOC

```
ffmpeg -f v4l2 -i /dev/video0 \
       -f alsa -i hw:0 \
       -map 0:0 \
       -c:v libvpx-vp9 \
       -s 640x360 -keyint_min 30 -g 30 \
       -f webm_chunk \
       -header webm_live_video_360.hdr \
       -chunk_start_index 1 \
       webm_live_video_360_%d.chk \
       -map 1:0 \
       -c:a libvorbis \
       -b:a 128k \
       -f webm_chunk \
       -header webm_live_audio_128.hdr \
       -chunk_start_index 1 \
       -audio_chunk_duration 1000 \
       webm_live_audio_128_%d.chk
```

5 Metadata# TOC

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
```

By using the ffmetadata muxer and demuxer it is possible to extract metadata from an input file to an ffmetadata file, and then transcode the file into an output file with the edited ffmetadata file.

Extracting an ffmetadata file with `ffmpeg` goes as follows:

```
ffmpeg -i INPUT -f ffmetadata FFMETADATAFILE
```

Reinserting edited metadata information from the FFMETADATAFILE file can be done as:

```
ffmpeg -i INPUT -i FFMETADATAFILE -map_metadata 1 -codec copy OUTPUT
```

6 See Also# TOC

ffmpeg, ffplay, ffprobe, ffserver, libavformat

7 Authors# TOC

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 using *makeinfo*.