

原 FFMPEG结构体分析：AVCodecContext

2013年11月08日 00:49:27 阅读数：51202

注：写了一系列的结构体的分析的文章，在这里列一个列表：

[FFMPEG结构体分析：AVFrame](#)

[FFMPEG结构体分析：AVFormatContext](#)

[FFMPEG结构体分析：AVCodecContext](#)

[FFMPEG结构体分析：AVIOContext](#)

[FFMPEG结构体分析：AVCodec](#)

[FFMPEG结构体分析：AVStream](#)

[FFMPEG结构体分析：AVPacket](#)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](#)

在此不再详述，其中AVCodecContext是包含变量较多的结构体（感觉差不多是变量最多的结构体）。本文将会大概分析一下该结构体里每个变量的含义和作用。因为如果每个变量都分析的话，工作量太大，实在来不及。

首先看一下结构体的定义（位于avcodec.h）：

```
[cpp]
1.  /*
2.   *雷霄骅
3.   *leixiaohua1020@126.com
4.   *中国传媒大学/数字电视技术
5.   */
6.  /**
7.   * main external API structure.
8.   * New fields can be added to the end with minor version bumps.
9.   * Removal, reordering and changes to existing fields require a major
10.  * version bump.
11.  * Please use AVOptions (av_opt* / av_set/get*()) to access these fields from user
12.  * applications.
13.  * sizeof(AVCodecContext) must not be used outside libav*.
14.  */
15.  typedef struct AVCodecContext {
16.      /**
17.       * information on struct for av_log
18.       * - set by avcodec_alloc_context3
19.       */
20.      const AVClass *av_class;
21.      int log_level_offset;
22.
23.      enum AVMediaType codec_type; /* see AVMEDIA_TYPE_xxx */
24.      const struct AVCodec *codec;
25.      char codec_name[32];
26.      enum AVCodecID codec_id; /* see AV_CODEC_ID_xxx */
27.
28.      /**
29.       * fourcc (LSB first, so "ABCD" -> ('D'<<24) + ('C'<<16) + ('B'<<8) + 'A').
30.       * This is used to work around some encoder bugs.
31.       * A demuxer should set this to what is stored in the field used to identify the codec.
32.       * If there are multiple such fields in a container then the demuxer should choose the one
33.       * which maximizes the information about the used codec.
34.       * If the codec tag field in a container is larger than 32 bits then the demuxer should
35.       * remap the longer ID to 32 bits with a table or other structure. Alternatively a new
36.       * extra_codec_tag + size could be added but for this a clear advantage must be demonstrated
37.       * first.
38.       * - encoding: Set by user, if not then the default based on codec_id will be used.
39.       * - decoding: Set by user, will be converted to uppercase by libavcodec during init.
40.       */
41.      unsigned int codec_tag;
42.
43.      /**
44.       * fourcc from the AVI stream header (LSB first, so "ABCD" -> ('D'<<24) + ('C'<<16) + ('B'<<8) + 'A').
45.       * This is used to work around some encoder bugs.
46.       * - encoding: unused
47.       * - decoding: Set by user, will be converted to uppercase by libavcodec during init.
48.       */
49.      unsigned int stream_codec_tag;
50.
51. #if FF_API_SUB_ID
52.      /**
```

```

53.     * @deprecated this field is unused
54.     */
55.     attribute_deprecated int sub_id;
56. #endif
57.
58.     void *priv_data;
59.
60.     /**
61.      * Private context used for internal data.
62.      *
63.      * Unlike priv_data, this is not codec-specific. It is used in general
64.      * libavcodec functions.
65.      */
66.     struct AVCodecInternal *internal;
67.
68.     /**
69.      * Private data of the user, can be used to carry app specific stuff.
70.      * - encoding: Set by user.
71.      * - decoding: Set by user.
72.      */
73.     void *opaque;
74.
75.     /**
76.      * the average bitrate
77.      * - encoding: Set by user; unused for constant quantizer encoding.
78.      * - decoding: Set by libavcodec. 0 or some bitrate if this info is available in the stream.
79.      */
80.     int bit_rate;
81.
82.     /**
83.      * number of bits the bitstream is allowed to diverge from the reference.
84.      *     the reference can be CBR (for CBR pass1) or VBR (for pass2)
85.      * - encoding: Set by user; unused for constant quantizer encoding.
86.      * - decoding: unused
87.      */
88.     int bit_rate_tolerance;
89.
90.     /**
91.      * Global quality for codecs which cannot change it per frame.
92.      * This should be proportional to MPEG-1/2/4 qscale.
93.      * - encoding: Set by user.
94.      * - decoding: unused
95.      */
96.     int global_quality;
97.
98.     /**
99.      * - encoding: Set by user.
100.     * - decoding: unused
101.     */
102.     int compression_level;
103. #define FF_COMPRESSION_DEFAULT -1
104.
105.     /**
106.      * CODEC_FLAG_*.
107.      * - encoding: Set by user.
108.      * - decoding: Set by user.
109.      */
110.     int flags;
111.
112.     /**
113.      * CODEC_FLAG2 *
114.      * - encoding: Set by user.
115.      * - decoding: Set by user.
116.      */
117.     int flags2;
118.
119.     /**
120.      * some codecs need / can use extradata like Huffman tables.
121.      * mjpeg: Huffman tables
122.      * rv10: additional flags
123.      * mpeg4: global headers (they can be in the bitstream or here)
124.      * The allocated memory should be FF_INPUT_BUFFER_PADDING_SIZE bytes larger
125.      * than extradata_size to avoid problems if it is read with the bitstream reader.
126.      * The bitwise contents of extradata must not depend on the architecture or CPU endianness.
127.      * - encoding: Set/allocated/freed by libavcodec.
128.      * - decoding: Set/allocated/freed by user.
129.      */
130.     uint8_t *extradata;
131.     int extradata_size;
132.
133.     /**
134.      * This is the fundamental unit of time (in seconds) in terms
135.      * of which frame timestamps are represented. For fixed-fps content,
136.      * timebase should be 1/framerate and timestamp increments should be
137.      * identically 1.
138.      * - encoding: MUST be set by user.
139.      * - decoding: Set by libavcodec.
140.      */
141.     AVRational time_base;
142.
143.     /**

```

```

144.     * For some codecs, the time base is closer to the field rate than the frame rate.
145.     * Most notably, H.264 and MPEG-2 specify time_base as half of frame duration
146.     * if no telecine is used ...
147.     *
148.     * Set to time_base ticks per frame. Default 1, e.g., H.264/MPEG-2 set it to 2.
149.     */
150.     int ticks_per_frame;
151.
152.     /**
153.     * Encoding: Number of frames delay there will be from the encoder input to
154.     *           the decoder output. (we assume the decoder matches the spec)
155.     * Decoding: Number of frames delay in addition to what a standard decoder
156.     *           as specified in the spec would produce.
157.     *
158.     * Video:
159.     *   Number of frames the decoded output will be delayed relative to the
160.     *   encoded input.
161.     *
162.     * Audio:
163.     *   For encoding, this is the number of "priming" samples added to the
164.     *   beginning of the stream. The decoded output will be delayed by this
165.     *   many samples relative to the input to the encoder. Note that this
166.     *   field is purely informational and does not directly affect the pts
167.     *   output by the encoder, which should always be based on the actual
168.     *   presentation time, including any delay.
169.     *   For decoding, this is the number of samples the decoder needs to
170.     *   output before the decoder's output is valid. When seeking, you should
171.     *   start decoding this many samples prior to your desired seek point.
172.     *
173.     * - encoding: Set by libavcodec.
174.     * - decoding: Set by libavcodec.
175.     */
176.     int delay;
177.
178.
179.     /* video only */
180.     /**
181.     * picture width / height.
182.     * - encoding: MUST be set by user.
183.     * - decoding: Set by libavcodec.
184.     * Note: For compatibility it is possible to set this instead of
185.     * coded_width/height before decoding.
186.     */
187.     int width, height;
188.
189.     /**
190.     * Bitstream width / height, may be different from width/height if lowres enabled.
191.     * - encoding: unused
192.     * - decoding: Set by user before init if known. Codec should override / dynamically change if needed.
193.     */
194.     int coded_width, coded_height;
195.
196. #define FF_ASPECT_EXTENDED 15
197.
198.     /**
199.     * the number of pictures in a group of pictures, or 0 for intra_only
200.     * - encoding: Set by user.
201.     * - decoding: unused
202.     */
203.     int gop_size;
204.
205.     /**
206.     * Pixel format, see AV_PIX_FMT_*.
207.     * May be set by the demuxer if known from headers.
208.     * May be overridden by the decoder if it knows better.
209.     * - encoding: Set by user.
210.     * - decoding: Set by user if known, overridden by libavcodec if known
211.     */
212.     enum AVPixelFormat pix_fmt;
213.
214.     /**
215.     * Motion estimation algorithm used for video coding.
216.     * 1 (zero), 2 (full), 3 (log), 4 (phods), 5 (epzs), 6 (x1), 7 (hex),
217.     * 8 (umh), 9 (iter), 10 (tesa) [7, 8, 10 are x264 specific, 9 is snow specific]
218.     * - encoding: MUST be set by user.
219.     * - decoding: unused
220.     */
221.     int me_method;
222.
223.     /**
224.     * If non NULL, 'draw_horiz_band' is called by the libavcodec
225.     * decoder to draw a horizontal band. It improves cache usage. Not
226.     * all codecs can do that. You must check the codec capabilities
227.     * beforehand.
228.     * When multithreading is used, it may be called from multiple threads
229.     * at the same time; threads might draw different parts of the same AVFrame,
230.     * or multiple AVFrames, and there is no guarantee that slices will be drawn
231.     * in order.
232.     * The function is also used by hardware acceleration APIs.
233.     * It is called at least once during frame decoding to pass
234.     * the data needed for hardware render.

```

```

235.     * In that mode instead of pixel data, AVFrame points to
236.     * a structure specific to the acceleration API. The application
237.     * reads the structure and can change some fields to indicate progress
238.     * or mark state.
239.     * - encoding: unused
240.     * - decoding: Set by user.
241.     * @param height the height of the slice
242.     * @param y the y position of the slice
243.     * @param type 1->top field, 2->bottom field, 3->frame
244.     * @param offset offset into the AVFrame.data from which the slice should be read
245.     */
246. void (*draw_horiz_band)(struct AVCodecContext *s,
247.                        const AVFrame *src, int offset[AV_NUM_DATA_POINTERS],
248.                        int y, int type, int height);
249.
250. /**
251.  * callback to negotiate the pixelFormat
252.  * @param fmt is the list of formats which are supported by the codec,
253.  * it is terminated by -1 as 0 is a valid format, the formats are ordered by quality.
254.  * The first is always the native one.
255.  * @return the chosen format
256.  * - encoding: unused
257.  * - decoding: Set by user, if not set the native format will be chosen.
258.  */
259. enum AVPixelFormat (*get_format)(struct AVCodecContext *s, const enum AVPixelFormat * fmt);
260.
261. /**
262.  * maximum number of B-frames between non-B-frames
263.  * Note: The output will be delayed by max_b_frames+1 relative to the input.
264.  * - encoding: Set by user.
265.  * - decoding: unused
266.  */
267. int max_b_frames;
268.
269. /**
270.  * qscale factor between IP and B-frames
271.  * If > 0 then the last P-frame quantizer will be used (q= lastp_q*factor+offset).
272.  * If < 0 then normal ratecontrol will be done (q= -normal_q*factor+offset).
273.  * - encoding: Set by user.
274.  * - decoding: unused
275.  */
276. float b_quant_factor;
277.
278. /** obsolete FIXME remove */
279. int rc_strategy;
280. #define FF_RC_STRATEGY_XVID 1
281.
282. int b_frame_strategy;
283.
284. #if FF_API_MPV_GLOBAL_OPTS
285. /**
286.  * luma single coefficient elimination threshold
287.  * - encoding: Set by user.
288.  * - decoding: unused
289.  */
290. attribute_deprecated int luma_elim_threshold;
291.
292. /**
293.  * chroma single coeff elimination threshold
294.  * - encoding: Set by user.
295.  * - decoding: unused
296.  */
297. attribute_deprecated int chroma_elim_threshold;
298. #endif
299.
300. /**
301.  * qscale offset between IP and B-frames
302.  * - encoding: Set by user.
303.  * - decoding: unused
304.  */
305. float b_quant_offset;
306.
307. /**
308.  * Size of the frame reordering buffer in the decoder.
309.  * For MPEG-2 it is 1 IPB or 0 low delay IP.
310.  * - encoding: Set by libavcodec.
311.  * - decoding: Set by libavcodec.
312.  */
313. int has_b_frames;
314.
315. /**
316.  * 0-> h263 quant 1-> mpeg quant
317.  * - encoding: Set by user.
318.  * - decoding: unused
319.  */
320. int mpeg_quant;
321.
322. /**
323.  * qscale factor between P and I-frames
324.  * If > 0 then the last p frame quantizer will be used (q= lastp_q*factor+offset).
325.  * If < 0 then normal ratecontrol will be done (q= -normal_q*factor+offset).
326.  * - encoding: Set by user.
327.  * - decoding: unused

```

```

326.     * - encoding: Set by user.
327.     * - decoding: unused
328.     */
329.     float i_quant_factor;
330.
331.     /**
332.      * qscale offset between P and I-frames
333.      * - encoding: Set by user.
334.      * - decoding: unused
335.      */
336.     float i_quant_offset;
337.
338.     /**
339.      * luminance masking (0-> disabled)
340.      * - encoding: Set by user.
341.      * - decoding: unused
342.      */
343.     float lumi_masking;
344.
345.     /**
346.      * temporary complexity masking (0-> disabled)
347.      * - encoding: Set by user.
348.      * - decoding: unused
349.      */
350.     float temporal_cplx_masking;
351.
352.     /**
353.      * spatial complexity masking (0-> disabled)
354.      * - encoding: Set by user.
355.      * - decoding: unused
356.      */
357.     float spatial_cplx_masking;
358.
359.     /**
360.      * p block masking (0-> disabled)
361.      * - encoding: Set by user.
362.      * - decoding: unused
363.      */
364.     float p_masking;
365.
366.     /**
367.      * darkness masking (0-> disabled)
368.      * - encoding: Set by user.
369.      * - decoding: unused
370.      */
371.     float dark_masking;
372.
373.     /**
374.      * slice count
375.      * - encoding: Set by libavcodec.
376.      * - decoding: Set by user (or 0).
377.      */
378.     int slice_count;
379.     /**
380.      * prediction method (needed for huffyuv)
381.      * - encoding: Set by user.
382.      * - decoding: unused
383.      */
384.     int prediction_method;
385. #define FF_PRED_LEFT 0
386. #define FF_PRED_PLANE 1
387. #define FF_PRED_MEDIAN 2
388.
389.     /**
390.      * slice offsets in the frame in bytes
391.      * - encoding: Set/allocated by libavcodec.
392.      * - decoding: Set/allocated by user (or NULL).
393.      */
394.     int *slice_offset;
395.
396.     /**
397.      * sample aspect ratio (0 if unknown)
398.      * That is the width of a pixel divided by the height of the pixel.
399.      * Numerator and denominator must be relatively prime and smaller than 256 for some video standards.
400.      * - encoding: Set by user.
401.      * - decoding: Set by libavcodec.
402.      */
403.     AVRational sample_aspect_ratio;
404.
405.     /**
406.      * motion estimation comparison function
407.      * - encoding: Set by user.
408.      * - decoding: unused
409.      */
410.     int me_cmp;
411.     /**
412.      * subpixel motion estimation comparison function
413.      * - encoding: Set by user.
414.      * - decoding: unused
415.      */
416.     int me_sub_cmp;
417.     /**

```

```

417.  /*
418.  * macroblock comparison function (not supported yet)
419.  * - encoding: Set by user.
420.  * - decoding: unused
421.  */
422.  int mb_cmp;
423.  /**
424.  * interlaced DCT comparison function
425.  * - encoding: Set by user.
426.  * - decoding: unused
427.  */
428.  int ildct_cmp;
429.  #define FF_CMP_SAD      0
430.  #define FF_CMP_SSE      1
431.  #define FF_CMP_SATD     2
432.  #define FF_CMP_DCT      3
433.  #define FF_CMP_PSNR     4
434.  #define FF_CMP_BIT      5
435.  #define FF_CMP_RD       6
436.  #define FF_CMP_ZERO     7
437.  #define FF_CMP_VSAD     8
438.  #define FF_CMP_VSSE     9
439.  #define FF_CMP_NSSE    10
440.  #define FF_CMP_W53     11
441.  #define FF_CMP_W97     12
442.  #define FF_CMP_DCTMAX   13
443.  #define FF_CMP_DCT264   14
444.  #define FF_CMP_CHROMA  256
445.
446.  /**
447.  * ME diamond size & shape
448.  * - encoding: Set by user.
449.  * - decoding: unused
450.  */
451.  int dia_size;
452.
453.  /**
454.  * amount of previous MV predictors (2a+1 x 2a+1 square)
455.  * - encoding: Set by user.
456.  * - decoding: unused
457.  */
458.  int last_predictor_count;
459.
460.  /**
461.  * prepass for motion estimation
462.  * - encoding: Set by user.
463.  * - decoding: unused
464.  */
465.  int pre_me;
466.
467.  /**
468.  * motion estimation prepass comparison function
469.  * - encoding: Set by user.
470.  * - decoding: unused
471.  */
472.  int me_pre_cmp;
473.
474.  /**
475.  * ME prepass diamond size & shape
476.  * - encoding: Set by user.
477.  * - decoding: unused
478.  */
479.  int pre_dia_size;
480.
481.  /**
482.  * subpel ME quality
483.  * - encoding: Set by user.
484.  * - decoding: unused
485.  */
486.  int me_subpel_quality;
487.
488.  /**
489.  * DTG active format information (additional aspect ratio
490.  * information only used in DVB MPEG-2 transport streams)
491.  * 0 if not set.
492.  *
493.  * - encoding: unused
494.  * - decoding: Set by decoder.
495.  */
496.  int dtg_active_format;
497.  #define FF_DTG_AFD_SAME      8
498.  #define FF_DTG_AFD_4_3      9
499.  #define FF_DTG_AFD_16_9     10
500.  #define FF_DTG_AFD_14_9     11
501.  #define FF_DTG_AFD_4_3_SP_14_9 13
502.  #define FF_DTG_AFD_16_9_SP_14_9 14
503.  #define FF_DTG_AFD_SP_4_3    15
504.
505.  /**
506.  * maximum motion estimation search range in subpel units
507.  * If 0 then no limit.
508.  *

```

```

509.     * - encoding: Set by user.
510.     * - decoding: unused
511.     */
512.     int me_range;
513.
514.     /**
515.     * intra quantizer bias
516.     * - encoding: Set by user.
517.     * - decoding: unused
518.     */
519.     int intra_quant_bias;
520. #define FF_DEFAULT_QUANT_BIAS 999999
521.
522.     /**
523.     * inter quantizer bias
524.     * - encoding: Set by user.
525.     * - decoding: unused
526.     */
527.     int inter_quant_bias;
528.
529. #if FF_API_COLOR_TABLE_ID
530.     /**
531.     * color table ID
532.     * - encoding: unused
533.     * - decoding: Which clrtbl should be used for 8bit RGB images.
534.     *             Tables have to be stored somewhere. FIXME
535.     */
536.     attribute_deprecated int color_table_id;
537. #endif
538.
539.     /**
540.     * slice flags
541.     * - encoding: unused
542.     * - decoding: Set by user.
543.     */
544.     int slice_flags;
545. #define SLICE_FLAG_CODED_ORDER    0x0001 ///< draw_horiz_band() is called in coded order instead of display
546. #define SLICE_FLAG_ALLOW_FIELD    0x0002 ///< allow draw_horiz_band() with field slices (MPEG2 field pics)
547. #define SLICE_FLAG_ALLOW_PLANE    0x0004 ///< allow draw_horiz_band() with 1 component at a time (SVQ1)
548.
549.     /**
550.     * XVideo Motion Acceleration
551.     * - encoding: forbidden
552.     * - decoding: set by decoder
553.     */
554.     int xvmc_acceleration;
555.
556.     /**
557.     * macroblock decision mode
558.     * - encoding: Set by user.
559.     * - decoding: unused
560.     */
561.     int mb_decision;
562. #define FF_MB_DECISION_SIMPLE 0    ///< uses mb_cmp
563. #define FF_MB_DECISION_BITS   1    ///< chooses the one which needs the fewest bits
564. #define FF_MB_DECISION_RD     2    ///< rate distortion
565.
566.     /**
567.     * custom intra quantization matrix
568.     * - encoding: Set by user, can be NULL.
569.     * - decoding: Set by libavcodec.
570.     */
571.     uint16_t *intra_matrix;
572.
573.     /**
574.     * custom inter quantization matrix
575.     * - encoding: Set by user, can be NULL.
576.     * - decoding: Set by libavcodec.
577.     */
578.     uint16_t *inter_matrix;
579.
580.     /**
581.     * scene change detection threshold
582.     * 0 is default, larger means fewer detected scene changes.
583.     * - encoding: Set by user.
584.     * - decoding: unused
585.     */
586.     int scenechange_threshold;
587.
588.     /**
589.     * noise reduction strength
590.     * - encoding: Set by user.
591.     * - decoding: unused
592.     */
593.     int noise_reduction;
594.
595. #if FF_API_INTER_THRESHOLD
596.     /**
597.     * @deprecated this field is unused
598.     */
599.     attribute_deprecated int inter_threshold;

```

```

600. #endif
601.
602. #if FF_API_MPV_GLOBAL_OPTS
603. /**
604.  * @deprecated use mpegvideo private options instead
605.  */
606. attribute_deprecated int quantizer_noise_shaping;
607. #endif
608.
609. /**
610.  * Motion estimation threshold below which no motion estimation is
611.  * performed, but instead the user specified motion vectors are used.
612.  *
613.  * - encoding: Set by user.
614.  * - decoding: unused
615.  */
616. int me_threshold;
617.
618. /**
619.  * Macroblock threshold below which the user specified macroblock types will be used.
620.  * - encoding: Set by user.
621.  * - decoding: unused
622.  */
623. int mb_threshold;
624.
625. /**
626.  * precision of the intra DC coefficient - 8
627.  * - encoding: Set by user.
628.  * - decoding: unused
629.  */
630. int intra_dc_precision;
631.
632. /**
633.  * Number of macroblock rows at the top which are skipped.
634.  * - encoding: unused
635.  * - decoding: Set by user.
636.  */
637. int skip_top;
638.
639. /**
640.  * Number of macroblock rows at the bottom which are skipped.
641.  * - encoding: unused
642.  * - decoding: Set by user.
643.  */
644. int skip_bottom;
645.
646. /**
647.  * Border processing masking, raises the quantizer for mbs on the borders
648.  * of the picture.
649.  * - encoding: Set by user.
650.  * - decoding: unused
651.  */
652. float border_masking;
653.
654. /**
655.  * minimum MB lagrange multipler
656.  * - encoding: Set by user.
657.  * - decoding: unused
658.  */
659. int mb_lmin;
660.
661. /**
662.  * maximum MB lagrange multipler
663.  * - encoding: Set by user.
664.  * - decoding: unused
665.  */
666. int mb_lmax;
667.
668. /**
669.  *
670.  * - encoding: Set by user.
671.  * - decoding: unused
672.  */
673. int me_penalty_compensation;
674.
675. /**
676.  *
677.  * - encoding: Set by user.
678.  * - decoding: unused
679.  */
680. int bidir_refine;
681.
682. /**
683.  *
684.  * - encoding: Set by user.
685.  * - decoding: unused
686.  */
687. int brd_scale;
688.
689. /**
690.  * minimum GOP size

```



```

691.     * - encoding: Set by user.
692.     * - decoding: unused
693.     */
694.     int keyint_min;
695.
696.     /**
697.      * number of reference frames
698.      * - encoding: Set by user.
699.      * - decoding: Set by lavc.
700.      */
701.     int refs;
702.
703.     /**
704.      * chroma qp offset from luma
705.      * - encoding: Set by user.
706.      * - decoding: unused
707.      */
708.     int chromaoffset;
709.
710.     /**
711.      * Multiplied by qscale for each frame and added to scene_change_score.
712.      * - encoding: Set by user.
713.      * - decoding: unused
714.      */
715.     int scenechange_factor;
716.
717.     /**
718.      *
719.      * Note: Value depends upon the compare function used for fullpel ME.
720.      * - encoding: Set by user.
721.      * - decoding: unused
722.      */
723.     int mv0_threshold;
724.
725.     /**
726.      * Adjust sensitivity of b_frame_strategy 1.
727.      * - encoding: Set by user.
728.      * - decoding: unused
729.      */
730.     int b_sensitivity;
731.
732.     /**
733.      * Chromaticity coordinates of the source primaries.
734.      * - encoding: Set by user
735.      * - decoding: Set by libavcodec
736.      */
737.     enum AVColorPrimaries color_primaries;
738.
739.     /**
740.      * Color Transfer Characteristic.
741.      * - encoding: Set by user
742.      * - decoding: Set by libavcodec
743.      */
744.     enum AVColorTransferCharacteristic color_trc;
745.
746.     /**
747.      * YUV colorspace type.
748.      * - encoding: Set by user
749.      * - decoding: Set by libavcodec
750.      */
751.     enum AVColorSpace colorspace;
752.
753.     /**
754.      * MPEG vs JPEG YUV range.
755.      * - encoding: Set by user
756.      * - decoding: Set by libavcodec
757.      */
758.     enum AVColorRange color_range;
759.
760.     /**
761.      * This defines the location of chroma samples.
762.      * - encoding: Set by user
763.      * - decoding: Set by libavcodec
764.      */
765.     enum AVChromaLocation chroma_sample_location;
766.
767.     /**
768.      * Number of slices.
769.      * Indicates number of picture subdivisions. Used for parallelized
770.      * decoding.
771.      * - encoding: Set by user
772.      * - decoding: unused
773.      */
774.     int slices;
775.
776.     /** Field order
777.      * - encoding: set by libavcodec
778.      * - decoding: Set by user.
779.      */
780.     enum AVFieldOrder field_order;
781.

```

```

782. /* audio only */
783. int sample_rate; ///< samples per second
784. int channels;    ///< number of audio channels
785.
786. /**
787.  * audio sample format
788.  * - encoding: Set by user.
789.  * - decoding: Set by libavcodec.
790.  */
791. enum AVSampleFormat sample_fmt; ///< sample format
792.
793. /* The following data should not be initialized. */
794. /**
795.  * Samples per packet, initialized when calling 'init'.
796.  */
797. int frame_size;
798.
799. /**
800.  * Frame counter, set by libavcodec.
801.  *
802.  * - decoding: total number of frames returned from the decoder so far.
803.  * - encoding: total number of frames passed to the encoder so far.
804.  *
805.  * @note the counter is not incremented if encoding/decoding resulted in
806.  *       an error.
807.  */
808. int frame_number;
809.
810. /**
811.  * number of bytes per packet if constant and known or 0
812.  * Used by some WAV based audio codecs.
813.  */
814. int block_align;
815.
816. /**
817.  * Audio cutoff bandwidth (0 means "automatic")
818.  * - encoding: Set by user.
819.  * - decoding: unused
820.  */
821. int cutoff;
822.
823. #if FF_API_REQUEST_CHANNELS
824. /**
825.  * Decoder should decode to this many channels if it can (0 for default)
826.  * - encoding: unused
827.  * - decoding: Set by user.
828.  * @deprecated Deprecated in favor of request_channel_layout.
829.  */
830. int request_channels;
831. #endif
832.
833. /**
834.  * Audio channel layout.
835.  * - encoding: set by user.
836.  * - decoding: set by user, may be overwritten by libavcodec.
837.  */
838. uint64_t channel_layout;
839.
840. /**
841.  * Request decoder to use this channel layout if it can (0 for default)
842.  * - encoding: unused
843.  * - decoding: Set by user.
844.  */
845. uint64_t request_channel_layout;
846.
847. /**
848.  * Type of service that the audio stream conveys.
849.  * - encoding: Set by user.
850.  * - decoding: Set by libavcodec.
851.  */
852. enum AVAudioServiceType audio_service_type;
853.
854. /**
855.  * desired sample format
856.  * - encoding: Not used.
857.  * - decoding: Set by user.
858.  * Decoder will decode to this format if it can.
859.  */
860. enum AVSampleFormat request_sample_fmt;
861.
862. /**
863.  * Called at the beginning of each frame to get a buffer for it.
864.  *
865.  * The function will set AVFrame.data[], AVFrame.linesize[].
866.  * AVFrame.extended_data[] must also be set, but it should be the same as
867.  * AVFrame.data[] except for planar audio with more channels than can fit
868.  * in AVFrame.data[]. In that case, AVFrame.data[] shall still contain as
869.  * many data pointers as it can hold.
870.  *
871.  * if CODEC_CAP_DR1 is not set then get_buffer() must call
872.  * avcodec_default_get_buffer() instead of providing buffers allocated by
873.  * some other means

```

```

873.     * some other means.
874.     *
875.     * AVFrame.data[] should be 32- or 16-byte-aligned unless the CPU doesn't
876.     * need it. avcodec_default_get_buffer() aligns the output buffer properly,
877.     * but if get_buffer() is overridden then alignment considerations should
878.     * be taken into account.
879.     *
880.     * @see avcodec_default_get_buffer()
881.     *
882.     * Video:
883.     *
884.     * If pic.reference is set then the frame will be read later by libavcodec.
885.     * avcodec_align_dimensions2() should be used to find the required width and
886.     * height, as they normally need to be rounded up to the next multiple of 16.
887.     *
888.     * If frame multithreading is used and thread_safe_callbacks is set,
889.     * it may be called from a different thread, but not from more than one at
890.     * once. Does not need to be reentrant.
891.     *
892.     * @see release_buffer(), reget_buffer()
893.     * @see avcodec_align_dimensions2()
894.     *
895.     * Audio:
896.     *
897.     * Decoders request a buffer of a particular size by setting
898.     * AVFrame.nb_samples prior to calling get_buffer(). The decoder may,
899.     * however, utilize only part of the buffer by setting AVFrame.nb_samples
900.     * to a smaller value in the output frame.
901.     *
902.     * Decoders cannot use the buffer after returning from
903.     * avcodec_decode_audio4(), so they will not call release_buffer(), as it
904.     * is assumed to be released immediately upon return.
905.     *
906.     * As a convenience, av_samples_get_buffer_size() and
907.     * av_samples_fill_arrays() in libavutil may be used by custom get_buffer()
908.     * functions to find the required data size and to fill data pointers and
909.     * linesize. In AVFrame.linesize, only linesize[0] may be set for audio
910.     * since all planes must be the same size.
911.     *
912.     * @see av_samples_get_buffer_size(), av_samples_fill_arrays()
913.     *
914.     * - encoding: unused
915.     * - decoding: Set by libavcodec, user can override.
916.     */
917. int (*get_buffer)(struct AVCodecContext *c, AVFrame *pic);
918.
919. /**
920.  * Called to release buffers which were allocated with get_buffer.
921.  * A released buffer can be reused in get_buffer().
922.  * pic.data[*] must be set to NULL.
923.  * May be called from a different thread if frame multithreading is used,
924.  * but not by more than one thread at once, so does not need to be reentrant.
925.  * - encoding: unused
926.  * - decoding: Set by libavcodec, user can override.
927.  */
928. void (*release_buffer)(struct AVCodecContext *c, AVFrame *pic);
929.
930. /**
931.  * Called at the beginning of a frame to get cr buffer for it.
932.  * Buffer type (size, hints) must be the same. libavcodec won't check it.
933.  * libavcodec will pass previous buffer in pic, function should return
934.  * same buffer or new buffer with old frame "painted" into it.
935.  * If pic.data[0] == NULL must behave like get_buffer().
936.  * if CODEC_CAP_DR1 is not set then reget_buffer() must call
937.  * avcodec_default_reget_buffer() instead of providing buffers allocated by
938.  * some other means.
939.  * - encoding: unused
940.  * - decoding: Set by libavcodec, user can override.
941.  */
942. int (*reget_buffer)(struct AVCodecContext *c, AVFrame *pic);
943.
944.
945. /* - encoding parameters */
946. float qcompress; ///< amount of qscale change between easy & hard scenes (0.0-1.0)
947. float qblur;     ///< amount of qscale smoothing over time (0.0-1.0)
948.
949. /**
950.  * minimum quantizer
951.  * - encoding: Set by user.
952.  * - decoding: unused
953.  */
954. int qmin;
955.
956. /**
957.  * maximum quantizer
958.  * - encoding: Set by user.
959.  * - decoding: unused
960.  */
961. int qmax;
962.
963. /**
964.  * maximum quantizer difference between frames

```

```

964.      * maximum quantizer difference between frames
965.      * - encoding: Set by user.
966.      * - decoding: unused
967.      */
968.      int max_qdiff;
969.
970.      /**
971.       * ratecontrol qmin qmax limiting method
972.       * 0-> clipping, 1-> use a nice continuous function to limit qscale wthin qmin/qmax.
973.       * - encoding: Set by user.
974.       * - decoding: unused
975.       */
976.      float rc_qsquish;
977.
978.      float rc_qmod_amp;
979.      int rc_qmod_freq;
980.
981.      /**
982.       * decoder bitstream buffer size
983.       * - encoding: Set by user.
984.       * - decoding: unused
985.       */
986.      int rc_buffer_size;
987.
988.      /**
989.       * ratecontrol override, see RcOverride
990.       * - encoding: Allocated/set/freed by user.
991.       * - decoding: unused
992.       */
993.      int rc_override_count;
994.      RcOverride *rc_override;
995.
996.      /**
997.       * rate control equation
998.       * - encoding: Set by user
999.       * - decoding: unused
1000.      */
1001.      const char *rc_eq;
1002.
1003.      /**
1004.       * maximum bitrate
1005.       * - encoding: Set by user.
1006.       * - decoding: unused
1007.       */
1008.      int rc_max_rate;
1009.
1010.      /**
1011.       * minimum bitrate
1012.       * - encoding: Set by user.
1013.       * - decoding: unused
1014.       */
1015.      int rc_min_rate;
1016.
1017.      float rc_buffer_aggressivity;
1018.
1019.      /**
1020.       * initial complexity for pass1 ratecontrol
1021.       * - encoding: Set by user.
1022.       * - decoding: unused
1023.       */
1024.      float rc_initial_cplx;
1025.
1026.      /**
1027.       * Ratecontrol attempt to use, at maximum, <value> of what can be used without an underflow.
1028.       * - encoding: Set by user.
1029.       * - decoding: unused.
1030.       */
1031.      float rc_max_available_vbv_use;
1032.
1033.      /**
1034.       * Ratecontrol attempt to use, at least, <value> times the amount needed to prevent a vbv overflow.
1035.       * - encoding: Set by user.
1036.       * - decoding: unused.
1037.       */
1038.      float rc_min_vbv_overflow_use;
1039.
1040.      /**
1041.       * Number of bits which should be loaded into the rc buffer before decoding starts.
1042.       * - encoding: Set by user.
1043.       * - decoding: unused
1044.       */
1045.      int rc_initial_buffer_occupancy;
1046.
1047.      #define FF_CODER_TYPE_VLC      0
1048.      #define FF_CODER_TYPE_AC      1
1049.      #define FF_CODER_TYPE_RAW     2
1050.      #define FF_CODER_TYPE_RLE     3
1051.      #define FF_CODER_TYPE_DEFLATE 4
1052.      /**
1053.       * coder type
1054.       * - encoding: Set by user.
1055.       * - decoding: unused

```

```

1056.     */
1057.     int coder_type;
1058.
1059.     /**
1060.      * context model
1061.      * - encoding: Set by user.
1062.      * - decoding: unused
1063.      */
1064.     int context_model;
1065.
1066.     /**
1067.      * minimum Lagrange multiplier
1068.      * - encoding: Set by user.
1069.      * - decoding: unused
1070.      */
1071.     int lmin;
1072.
1073.     /**
1074.      * maximum Lagrange multiplier
1075.      * - encoding: Set by user.
1076.      * - decoding: unused
1077.      */
1078.     int lmax;
1079.
1080.     /**
1081.      * frame skip threshold
1082.      * - encoding: Set by user.
1083.      * - decoding: unused
1084.      */
1085.     int frame_skip_threshold;
1086.
1087.     /**
1088.      * frame skip factor
1089.      * - encoding: Set by user.
1090.      * - decoding: unused
1091.      */
1092.     int frame_skip_factor;
1093.
1094.     /**
1095.      * frame skip exponent
1096.      * - encoding: Set by user.
1097.      * - decoding: unused
1098.      */
1099.     int frame_skip_exp;
1100.
1101.     /**
1102.      * frame skip comparison function
1103.      * - encoding: Set by user.
1104.      * - decoding: unused
1105.      */
1106.     int frame_skip_cmp;
1107.
1108.     /**
1109.      * trellis RD quantization
1110.      * - encoding: Set by user.
1111.      * - decoding: unused
1112.      */
1113.     int trellis;
1114.
1115.     /**
1116.      * - encoding: Set by user.
1117.      * - decoding: unused
1118.      */
1119.     int min_prediction_order;
1120.
1121.     /**
1122.      * - encoding: Set by user.
1123.      * - decoding: unused
1124.      */
1125.     int max_prediction_order;
1126.
1127.     /**
1128.      * GOP timecode frame start number
1129.      * - encoding: Set by user, in non drop frame format
1130.      * - decoding: Set by libavcodec (timecode in the 25 bits format, -1 if unset)
1131.      */
1132.     int64_t timecode_frame_start;
1133.
1134.     /* The RTP callback: This function is called
1135.      * every time the encoder has a packet to send.
1136.      * It depends on the encoder if the data starts
1137.      * with a Start Code (it should). H.263 does.
1138.      * mb_nb contains the number of macroblocks
1139.      * encoded in the RTP payload.
1140.      void (*rtp_callback)(struct AVCodecContext *avctx, void *data, int size, int mb_nb);
1141.
1142.     int rtp_payload_size; /* The size of the RTP payload: the coder will
1143.                          * /* do its best to deliver a chunk with size
1144.                          * /* below rtp_payload_size, the chunk will start
1145.                          * /* with a start code on some codecs like H.263.
1146.                          * /* This doesn't take account of any particular

```

```

1147.         /* headers inside the transmitted RTP payload. */
1148.
1149.     /* statistics, used for 2-pass encoding */
1150.     int mv_bits;
1151.     int header_bits;
1152.     int i_tex_bits;
1153.     int p_tex_bits;
1154.     int i_count;
1155.     int p_count;
1156.     int skip_count;
1157.     int misc_bits;
1158.
1159.     /**
1160.      * number of bits used for the previously encoded frame
1161.      * - encoding: Set by libavcodec.
1162.      * - decoding: unused
1163.      */
1164.     int frame_bits;
1165.
1166.     /**
1167.      * pass1 encoding statistics output buffer
1168.      * - encoding: Set by libavcodec.
1169.      * - decoding: unused
1170.      */
1171.     char *stats_out;
1172.
1173.     /**
1174.      * pass2 encoding statistics input buffer
1175.      * Concatenated stuff from stats_out of pass1 should be placed here.
1176.      * - encoding: Allocated/set/freed by user.
1177.      * - decoding: unused
1178.      */
1179.     char *stats_in;
1180.
1181.     /**
1182.      * Work around bugs in encoders which sometimes cannot be detected automatically.
1183.      * - encoding: Set by user
1184.      * - decoding: Set by user
1185.      */
1186.     int workaround_bugs;
1187.     #define FF_BUG_AUTODETECT      1 ///< autodetection
1188.     #define FF_BUG_OLD_MSMPEG4    2
1189.     #define FF_BUG_XVID_ILACE     4
1190.     #define FF_BUG_UMP4           8
1191.     #define FF_BUG_NO_PADDING     16
1192.     #define FF_BUG_AMV            32
1193.     #define FF_BUG_AC_VLC         0 ///< Will be removed, libavcodec can now handle these non-compliant files by default.
1194.     #define FF_BUG_QPEL_CHROMA    64
1195.     #define FF_BUG_STD_QPEL       128
1196.     #define FF_BUG_QPEL_CHROMA2   256
1197.     #define FF_BUG_DIRECT_BLOCKSIZE 512
1198.     #define FF_BUG_EDGE           1024
1199.     #define FF_BUG_HPEL_CHROMA    2048
1200.     #define FF_BUG_DC_CLIP        4096
1201.     #define FF_BUG_MS             8192 ///< Work around various bugs in Microsoft's broken decoders.
1202.     #define FF_BUG_TRUNCATED      16384
1203.
1204.     /**
1205.      * strictly follow the standard (MPEG4, ...).
1206.      * - encoding: Set by user.
1207.      * - decoding: Set by user.
1208.      * Setting this to STRICT or higher means the encoder and decoder will
1209.      * generally do stupid things, whereas setting it to unofficial or lower
1210.      * will mean the encoder might produce output that is not supported by all
1211.      * spec-compliant decoders. Decoders don't differentiate between normal,
1212.      * unofficial and experimental (that is, they always try to decode things
1213.      * when they can) unless they are explicitly asked to behave stupidly
1214.      * (=strictly conform to the specs)
1215.      */
1216.     int strict_std_compliance;
1217.     #define FF_COMPLIANCE_VERY_STRICT 2 ///< Strictly conform to an older more strict version of the spec or reference software.
1218.     #define FF_COMPLIANCE_STRICT     1 ///< Strictly conform to all the things in the spec no matter what consequences.
1219.     #define FF_COMPLIANCE_NORMAL     0
1220.     #define FF_COMPLIANCE_UNOFFICIAL -1 ///< Allow unofficial extensions
1221.     #define FF_COMPLIANCE_EXPERIMENTAL -2 ///< Allow nonstandardized experimental things.
1222.
1223.     /**
1224.      * error concealment flags
1225.      * - encoding: unused
1226.      * - decoding: Set by user.
1227.      */
1228.     int error_concealment;
1229.     #define FF_EC_GUESS_MVS      1
1230.     #define FF_EC_DEBLOCK        2
1231.
1232.     /**
1233.      * debug
1234.      * - encoding: Set by user.
1235.      * - decoding: Set by user.
1236.      */
1237.     int debug;

```

```

1238. #define FF_DEBUG_PICT_INFO 1
1239. #define FF_DEBUG_RC 2
1240. #define FF_DEBUG_BITSTREAM 4
1241. #define FF_DEBUG_MB_TYPE 8
1242. #define FF_DEBUG_QP 16
1243. #define FF_DEBUG_MV 32
1244. #define FF_DEBUG_DCT_COEFF 0x00000040
1245. #define FF_DEBUG_SKIP 0x00000080
1246. #define FF_DEBUG_STARTCODE 0x00000100
1247. #define FF_DEBUG_PTS 0x00000200
1248. #define FF_DEBUG_ER 0x00000400
1249. #define FF_DEBUG_MMCO 0x00000800
1250. #define FF_DEBUG_BUGS 0x00001000
1251. #define FF_DEBUG_VIS_QP 0x00002000
1252. #define FF_DEBUG_VIS_MB_TYPE 0x00004000
1253. #define FF_DEBUG_BUFFERS 0x00008000
1254. #define FF_DEBUG_THREADS 0x00010000
1255.
1256. /**
1257.  * debug
1258.  * - encoding: Set by user.
1259.  * - decoding: Set by user.
1260.  */
1261. int debug_mv;
1262. #define FF_DEBUG_VIS_MV_P_FOR 0x00000001 //visualize forward predicted MVs of P frames
1263. #define FF_DEBUG_VIS_MV_B_FOR 0x00000002 //visualize forward predicted MVs of B frames
1264. #define FF_DEBUG_VIS_MV_B_BACK 0x00000004 //visualize backward predicted MVs of B frames
1265.
1266. /**
1267.  * Error recognition; may misdetect some more or less valid parts as errors.
1268.  * - encoding: unused
1269.  * - decoding: Set by user.
1270.  */
1271. int err_recognition;
1272. #define AV_EF_CRCHECK (1<<0)
1273. #define AV_EF_BITSTREAM (1<<1)
1274. #define AV_EF_BUFFER (1<<2)
1275. #define AV_EF_EXPLODE (1<<3)
1276.
1277. #define AV_EF_CAREFUL (1<<16)
1278. #define AV_EF_COMPLIANT (1<<17)
1279. #define AV_EF_AGGRESSIVE (1<<18)
1280.
1281.
1282. /**
1283.  * opaque 64bit number (generally a PTS) that will be reordered and
1284.  * output in AVFrame.reordered_opaque
1285.  * @deprecated in favor of pkt_pts
1286.  * - encoding: unused
1287.  * - decoding: Set by user.
1288.  */
1289. int64_t reordered_opaque;
1290.
1291. /**
1292.  * Hardware accelerator in use
1293.  * - encoding: unused.
1294.  * - decoding: Set by libavcodec
1295.  */
1296. struct AVHWAcel *hwaccel;
1297.
1298. /**
1299.  * Hardware accelerator context.
1300.  * For some hardware accelerators, a global context needs to be
1301.  * provided by the user. In that case, this holds display-dependent
1302.  * data FFmpeg cannot instantiate itself. Please refer to the
1303.  * FFmpeg HW accelerator documentation to know how to fill this
1304.  * is. e.g. for VA API, this is a struct vaapi_context.
1305.  * - encoding: unused
1306.  * - decoding: Set by user
1307.  */
1308. void *hwaccel_context;
1309.
1310. /**
1311.  * error
1312.  * - encoding: Set by libavcodec if flags&CODEC_FLAG_PSNR.
1313.  * - decoding: unused
1314.  */
1315. uint64_t error[AV_NUM_DATA_POINTERS];
1316.
1317. /**
1318.  * DCT algorithm, see FF_DCT_* below
1319.  * - encoding: Set by user.
1320.  * - decoding: unused
1321.  */
1322. int dct_algo;
1323. #define FF_DCT_AUTO 0
1324. #define FF_DCT_FASTINT 1
1325. #define FF_DCT_INT 2
1326. #define FF_DCT_MMX 3
1327. #define FF_DCT_ALTIVEC 5
1328. #define FF_DCT_FAAN 6

```

```

1329.
1330. /**
1331.  * IDCT algorithm, see FF_IDCT_* below.
1332.  * - encoding: Set by user.
1333.  * - decoding: Set by user.
1334.  */
1335. int idct_algo;
1336. #define FF_IDCT_AUTO      0
1337. #define FF_IDCT_INT       1
1338. #define FF_IDCT_SIMPLE    2
1339. #define FF_IDCT_SIMPLEMMX 3
1340. #define FF_IDCT_LIBMPEG2MMX 4
1341. #define FF_IDCT_MMI       5
1342. #define FF_IDCT_ARM       7
1343. #define FF_IDCT_ALTIVEC   8
1344. #define FF_IDCT_SH4       9
1345. #define FF_IDCT_SIMPLEARM 10
1346. #define FF_IDCT_H264      11
1347. #define FF_IDCT_VP3       12
1348. #define FF_IDCT_IPP       13
1349. #define FF_IDCT_XVIDMMX   14
1350. #define FF_IDCT_CAVS      15
1351. #define FF_IDCT_SIMPLEARMV5TE 16
1352. #define FF_IDCT_SIMPLEARMV6 17
1353. #define FF_IDCT_SIMPLEVIS 18
1354. #define FF_IDCT_WMV2      19
1355. #define FF_IDCT_FAAN      20
1356. #define FF_IDCT_EA        21
1357. #define FF_IDCT_SIMPLENEON 22
1358. #define FF_IDCT_SIMPLEALPHA 23
1359. #define FF_IDCT_BINK      24
1360.
1361. #if FF_API_DSP_MASK
1362. /**
1363.  * Unused.
1364.  * @deprecated use av_set_cpu_flags_mask() instead.
1365.  */
1366. attribute_deprecated unsigned dsp_mask;
1367. #endif
1368.
1369. /**
1370.  * bits per sample/pixel from the demuxer (needed for huffyuv).
1371.  * - encoding: Set by libavcodec.
1372.  * - decoding: Set by user.
1373.  */
1374. int bits_per_coded_sample;
1375.
1376. /**
1377.  * Bits per sample/pixel of internal libavcodec pixel/sample format.
1378.  * - encoding: set by user.
1379.  * - decoding: set by libavcodec.
1380.  */
1381. int bits_per_raw_sample;
1382.
1383. /**
1384.  * low resolution decoding, 1-> 1/2 size, 2->1/4 size
1385.  * - encoding: unused
1386.  * - decoding: Set by user.
1387.  */
1388. int lowres;
1389.
1390. /**
1391.  * the picture in the bitstream
1392.  * - encoding: Set by libavcodec.
1393.  * - decoding: Set by libavcodec.
1394.  */
1395. AVFrame *coded_frame;
1396.
1397. /**
1398.  * thread count
1399.  * is used to decide how many independent tasks should be passed to execute()
1400.  * - encoding: Set by user.
1401.  * - decoding: Set by user.
1402.  */
1403. int thread_count;
1404.
1405. /**
1406.  * Which multithreading methods to use.
1407.  * Use of FF_THREAD_FRAME will increase decoding delay by one frame per thread,
1408.  * so clients which cannot provide future frames should not use it.
1409.  *
1410.  * - encoding: Set by user, otherwise the default is used.
1411.  * - decoding: Set by user, otherwise the default is used.
1412.  */
1413. int thread_type;
1414. #define FF_THREAD_FRAME 1 ///< Decode more than one frame at once
1415. #define FF_THREAD_SLICE 2 ///< Decode more than one part of a single frame at once
1416.
1417. /**
1418.  * Which multithreading methods are in use by the codec.
1419.  * - encoding: Set by libavcodec.
1420.  * - decoding: Set by libavcodec.

```



```

1420.     * - decoding: Set by libavcodec.
1421.     */
1422.     int active_thread_type;
1423.
1424.     /**
1425.      * Set by the client if its custom get_buffer() callback can be called
1426.      * synchronously from another thread, which allows faster multithreaded decoding.
1427.      * draw_horiz_band() will be called from other threads regardless of this setting.
1428.      * Ignored if the default get_buffer() is used.
1429.      * - encoding: Set by user.
1430.      * - decoding: Set by user.
1431.      */
1432.     int thread_safe_callbacks;
1433.
1434.     /**
1435.      * The codec may call this to execute several independent things.
1436.      * It will return only after finishing all tasks.
1437.      * The user may replace this with some multithreaded implementation,
1438.      * the default implementation will execute the parts serially.
1439.      * @param count the number of things to execute
1440.      * - encoding: Set by libavcodec, user can override.
1441.      * - decoding: Set by libavcodec, user can override.
1442.      */
1443.     int (*execute)(struct AVCodecContext *c, int (*func)(struct AVCodecContext *c2, void *arg), void *arg2, int *ret, int count, int
size);
1444.
1445.     /**
1446.      * The codec may call this to execute several independent things.
1447.      * It will return only after finishing all tasks.
1448.      * The user may replace this with some multithreaded implementation,
1449.      * the default implementation will execute the parts serially.
1450.      * Also see avcodec_thread_init and e.g. the --enable-pthread configure option.
1451.      * @param c context passed also to func
1452.      * @param count the number of things to execute
1453.      * @param arg2 argument passed unchanged to func
1454.      * @param ret return values of executed functions, must have space for "count" values. May be NULL.
1455.      * @param func function that will be called count times, with jobnr from 0 to count-1.
1456.      *          threadnr will be in the range 0 to c->thread_count-1 < MAX_THREADS and so that no
1457.      *          two instances of func executing at the same time will have the same threadnr.
1458.      * @return always 0 currently, but code should handle a future improvement where when any call to func
1459.      *          returns < 0 no further calls to func may be done and < 0 is returned.
1460.      * - encoding: Set by libavcodec, user can override.
1461.      * - decoding: Set by libavcodec, user can override.
1462.      */
1463.     int (*execute2)(struct AVCodecContext *c, int (*func)(struct AVCodecContext *c2, void *arg, int jobnr, int threadnr), void *arg2
, int *ret, int count);
1464.
1465.     /**
1466.      * thread opaque
1467.      * Can be used by execute() to store some per AVCodecContext stuff.
1468.      * - encoding: set by execute()
1469.      * - decoding: set by execute()
1470.      */
1471.     void *thread_opaque;
1472.
1473.     /**
1474.      * noise vs. sse weight for the nsse comparsion function
1475.      * - encoding: Set by user.
1476.      * - decoding: unused
1477.      */
1478.     int nsse_weight;
1479.
1480.     /**
1481.      * profile
1482.      * - encoding: Set by user.
1483.      * - decoding: Set by libavcodec.
1484.      */
1485.     int profile;
1486. #define FF_PROFILE_UNKNOWN -99
1487. #define FF_PROFILE_RESERVED -100
1488.
1489. #define FF_PROFILE_AAC_MAIN 0
1490. #define FF_PROFILE_AAC_LOW 1
1491. #define FF_PROFILE_AAC_SSR 2
1492. #define FF_PROFILE_AAC_LTP 3
1493. #define FF_PROFILE_AAC_HE 4
1494. #define FF_PROFILE_AAC_HE_V2 28
1495. #define FF_PROFILE_AAC_LD 22
1496. #define FF_PROFILE_AAC_ELD 38
1497.
1498. #define FF_PROFILE_DTS 20
1499. #define FF_PROFILE_DTS_ES 30
1500. #define FF_PROFILE_DTS_96_24 40
1501. #define FF_PROFILE_DTS_HD_HRA 50
1502. #define FF_PROFILE_DTS_HD_MA 60
1503.
1504. #define FF_PROFILE_MPEG2_422 0
1505. #define FF_PROFILE_MPEG2_HIGH 1
1506. #define FF_PROFILE_MPEG2_SS 2
1507. #define FF_PROFILE_MPEG2_SNR_SCALABLE 3
1508. #define FF_PROFILE_MPEG2_MAIN 4
1509. #define FF_PROFILE_MPEG2_STUDIO 5

```

```

1509. #define FF_PROFILE_MPEG2_SIMPLE 0
1510.
1511. #define FF_PROFILE_H264_CONSTRAINED (1<<9) // 8+1; constraint_set1_flag
1512. #define FF_PROFILE_H264_INTRA (1<<11) // 8+3; constraint_set3_flag
1513.
1514. #define FF_PROFILE_H264_BASELINE 66
1515. #define FF_PROFILE_H264_CONSTRAINED_BASELINE (66|FF_PROFILE_H264_CONSTRAINED)
1516. #define FF_PROFILE_H264_MAIN 77
1517. #define FF_PROFILE_H264_EXTENDED 88
1518. #define FF_PROFILE_H264_HIGH 100
1519. #define FF_PROFILE_H264_HIGH_10 110
1520. #define FF_PROFILE_H264_HIGH_10_INTRA (110|FF_PROFILE_H264_INTRA)
1521. #define FF_PROFILE_H264_HIGH_422 122
1522. #define FF_PROFILE_H264_HIGH_422_INTRA (122|FF_PROFILE_H264_INTRA)
1523. #define FF_PROFILE_H264_HIGH_444 144
1524. #define FF_PROFILE_H264_HIGH_444_PREDICTIVE 244
1525. #define FF_PROFILE_H264_HIGH_444_INTRA (244|FF_PROFILE_H264_INTRA)
1526. #define FF_PROFILE_H264_CAVLC_444 44
1527.
1528. #define FF_PROFILE_VC1_SIMPLE 0
1529. #define FF_PROFILE_VC1_MAIN 1
1530. #define FF_PROFILE_VC1_COMPLEX 2
1531. #define FF_PROFILE_VC1_ADVANCED 3
1532.
1533. #define FF_PROFILE_MPEG4_SIMPLE 0
1534. #define FF_PROFILE_MPEG4_SIMPLE_SCALABLE 1
1535. #define FF_PROFILE_MPEG4_CORE 2
1536. #define FF_PROFILE_MPEG4_MAIN 3
1537. #define FF_PROFILE_MPEG4_N_BIT 4
1538. #define FF_PROFILE_MPEG4_SCALABLE_TEXTURE 5
1539. #define FF_PROFILE_MPEG4_SIMPLE_FACE_ANIMATION 6
1540. #define FF_PROFILE_MPEG4_BASIC_ANIMATED_TEXTURE 7
1541. #define FF_PROFILE_MPEG4_HYBRID 8
1542. #define FF_PROFILE_MPEG4_ADVANCED_REAL_TIME 9
1543. #define FF_PROFILE_MPEG4_CORE_SCALABLE 10
1544. #define FF_PROFILE_MPEG4_ADVANCED_CODING 11
1545. #define FF_PROFILE_MPEG4_ADVANCED_CORE 12
1546. #define FF_PROFILE_MPEG4_ADVANCED_SCALABLE_TEXTURE 13
1547. #define FF_PROFILE_MPEG4_SIMPLE_STUDIO 14
1548. #define FF_PROFILE_MPEG4_ADVANCED_SIMPLE 15
1549.
1550. /**
1551.  * level
1552.  * - encoding: Set by user.
1553.  * - decoding: Set by libavcodec.
1554.  */
1555. int level;
1556. #define FF_LEVEL_UNKNOWN -99
1557.
1558. /**
1559.  *
1560.  * - encoding: unused
1561.  * - decoding: Set by user.
1562.  */
1563. enum AVDiscard skip_loop_filter;
1564.
1565. /**
1566.  *
1567.  * - encoding: unused
1568.  * - decoding: Set by user.
1569.  */
1570. enum AVDiscard skip_idct;
1571.
1572. /**
1573.  *
1574.  * - encoding: unused
1575.  * - decoding: Set by user.
1576.  */
1577. enum AVDiscard skip_frame;
1578.
1579. /**
1580.  * Header containing style information for text subtitles.
1581.  * For SUBTITLE_ASS subtitle type, it should contain the whole ASS
1582.  * [Script Info] and [V4+ Styles] section, plus the [Events] line and
1583.  * the Format line following. It shouldn't include any Dialogue line.
1584.  * - encoding: Set/allocated/freed by user (before avcodec_open2())
1585.  * - decoding: Set/allocated/freed by libavcodec (by avcodec_open2())
1586.  */
1587. uint8_t *subtitle_header;
1588. int subtitle_header_size;
1589.
1590. /**
1591.  * Simulates errors in the bitstream to test error concealment.
1592.  * - encoding: Set by user.
1593.  * - decoding: unused
1594.  */
1595. int error_rate;
1596.
1597. /**
1598.  * Current packet as passed into the decoder, to avoid having
1599.  * to pass the packet into every function. Currently only valid
1600.  * inside lavc and get/release buffer callbacks.

```

```

1601.     * - decoding: set by avcodec_decode_*, read by get_buffer() for setting pkt_pts
1602.     * - encoding: unused
1603.     */
1604.     AVPacket *pkt;
1605.
1606.     /**
1607.      * VBV delay coded in the last frame (in periods of a 27 MHz clock).
1608.      * Used for compliant TS muxing.
1609.      * - encoding: Set by libavcodec.
1610.      * - decoding: unused.
1611.      */
1612.     uint64_t vbv_delay;
1613.
1614.     /**
1615.      * Timebase in which pkt_dts/pts and AVPacket.dts/pts are.
1616.      * Code outside libavcodec should access this field using:
1617.      * avcodec_set_pkt_timebase(avctx)
1618.      * - encoding unused.
1619.      * - decoding set by user
1620.      */
1621.     AVRational pkt_timebase;
1622.
1623.     /**
1624.      * AVCodecDescriptor
1625.      * Code outside libavcodec should access this field using:
1626.      * avcodec_get_codec_descriptor(avctx)
1627.      * - encoding: unused.
1628.      * - decoding: set by libavcodec.
1629.      */
1630.     const AVCodecDescriptor *codec_descriptor;
1631.
1632.     /**
1633.      * Current statistics for PTS correction.
1634.      * - decoding: maintained and used by libavcodec, not intended to be used by user apps
1635.      * - encoding: unused
1636.      */
1637.     int64_t pts_correction_num_faulty_pts; /// Number of incorrect PTS values so far
1638.     int64_t pts_correction_num_faulty_dts; /// Number of incorrect DTS values so far
1639.     int64_t pts_correction_last_pts;      /// PTS of the last frame
1640.     int64_t pts_correction_last_dts;      /// DTS of the last frame
1641. } AVCodecContext;

```

光定义就真是够多的。下面挑一些关键的变量来看看（这里只考虑解码）。

enum AVMediaType codec_type：编解码器的类型（视频，音频...）

struct AVCodec *codec：采用的解码器AVCodec（H.264,MPEG2...）

int bit_rate：平均比特率

uint8_t *extradata; int extradata_size：针对特定编码器包含的附加信息（例如对于H.264解码器来说，存储SPS，PPS等）

AVRational time_base：根据该参数，可以把PTS转化为实际的时间（单位为秒s）

int width, height：如果是视频的话，代表宽和高

int refs：运动估计参考帧的个数（H.264的话会有多帧，MPEG2这类的一般就没有了）

int sample_rate：采样率（音频）

int channels：声道数（音频）

enum AVSampleFormat sample_fmt：采样格式

int profile：型（H.264里面就有，其他编码标准应该也有）

int level：级（和profile差不多）

在这里需要注意：AVCodecContext中很多的参数是编码的时候使用的，而不是解码的时候使用的。

其实这些参数都比较容易理解。就不多费篇幅了。在这里看一下以下几个参数：

1.codec_type

编解码器类型有以下几种：

```
[cpp]
1. enum AVMediaType {
2.     AVMEDIA_TYPE_UNKNOWN = -1, ///< Usually treated as AVMEDIA_TYPE_DATA
3.     AVMEDIA_TYPE_VIDEO,
4.     AVMEDIA_TYPE_AUDIO,
5.     AVMEDIA_TYPE_DATA,          ///< Opaque data information usually continuous
6.     AVMEDIA_TYPE_SUBTITLE,
7.     AVMEDIA_TYPE_ATTACHMENT,    ///< Opaque data information usually sparse
8.     AVMEDIA_TYPE_NB
9. };
```

2.sample_fmt

在FFMPEG中音频采样格式有以下几种：

```
[cpp]
1. enum AVSampleFormat {
2.     AV_SAMPLE_FMT_NONE = -1,
3.     AV_SAMPLE_FMT_U8,          ///< unsigned 8 bits
4.     AV_SAMPLE_FMT_S16,         ///< signed 16 bits
5.     AV_SAMPLE_FMT_S32,         ///< signed 32 bits
6.     AV_SAMPLE_FMT_FLT,         ///< float
7.     AV_SAMPLE_FMT_DBL,         ///< double
8.
9.     AV_SAMPLE_FMT_U8P,          ///< unsigned 8 bits, planar
10.    AV_SAMPLE_FMT_S16P,         ///< signed 16 bits, planar
11.    AV_SAMPLE_FMT_S32P,         ///< signed 32 bits, planar
12.    AV_SAMPLE_FMT_FLTP,         ///< float, planar
13.    AV_SAMPLE_FMT_DBLP,         ///< double, planar
14.
15.    AV_SAMPLE_FMT_NB            ///< Number of sample formats. DO NOT USE if linking dynamically
16. };
```

3.profile

在FFMPEG中型有以下几种，可以看出AAC，MPEG2，H.264，VC-1，MPEG4都有型的概念。

```
[cpp]
1. #define FF_PROFILE_UNKNOWN -99
2. #define FF_PROFILE_RESERVED -100
3.
4. #define FF_PROFILE_AAC_MAIN 0
5. #define FF_PROFILE_AAC_LOW 1
6. #define FF_PROFILE_AAC_SSR 2
7. #define FF_PROFILE_AAC_LTP 3
8. #define FF_PROFILE_AAC_HE 4
9. #define FF_PROFILE_AAC_HE_V2 28
10. #define FF_PROFILE_AAC_LD 22
11. #define FF_PROFILE_AAC_ELD 38
12.
13. #define FF_PROFILE_DTS 20
14. #define FF_PROFILE_DTS_ES 30
15. #define FF_PROFILE_DTS_96_24 40
16. #define FF_PROFILE_DTS_HD_HRA 50
17. #define FF_PROFILE_DTS_HD_MA 60
18.
19. #define FF_PROFILE_MPEG2_422 0
20. #define FF_PROFILE_MPEG2_HIGH 1
21. #define FF_PROFILE_MPEG2_SS 2
22. #define FF_PROFILE_MPEG2_SNR_SCALABLE 3
23. #define FF_PROFILE_MPEG2_MAIN 4
24. #define FF_PROFILE_MPEG2_SIMPLE 5
25.
26. #define FF_PROFILE_H264_CONSTRAINED (1<<9) // 8+1; constraint_set1_flag
27. #define FF_PROFILE_H264_INTRA (1<<11) // 8+3; constraint_set3_flag
28.
29. #define FF_PROFILE_H264_BASELINE 66
30. #define FF_PROFILE_H264_CONSTRAINED_BASELINE (66|FF_PROFILE_H264_CONSTRAINED)
31. #define FF_PROFILE_H264_MAIN 77
32. #define FF_PROFILE_H264_EXTENDED 88
33. #define FF_PROFILE_H264_HIGH 100
34. #define FF_PROFILE_H264_HIGH_10 110
35. #define FF_PROFILE_H264_HIGH_10_INTRA (110|FF_PROFILE_H264_INTRA)
36. #define FF_PROFILE_H264_HIGH_422 122
37. #define FF_PROFILE_H264_HIGH_422_INTRA (122|FF_PROFILE_H264_INTRA)
38. #define FF_PROFILE_H264_HIGH_444 144
39. #define FF_PROFILE_H264_HIGH_444_PREDICTIVE 244
40. #define FF_PROFILE_H264_HIGH_444_INTRA (244|FF_PROFILE_H264_INTRA)
41. #define FF_PROFILE_H264_CAVLC_444 44
42.
43. #define FF_PROFILE_VC1_SIMPLE 0
44. #define FF_PROFILE_VC1_MAIN 1
45. #define FF_PROFILE_VC1_COMPLEX 2
46. #define FF_PROFILE_VC1_ADVANCED 3
47.
48. #define FF_PROFILE_MPEG4_SIMPLE 0
49. #define FF_PROFILE_MPEG4_SIMPLE_SCALABLE 1
50. #define FF_PROFILE_MPEG4_CORE 2
51. #define FF_PROFILE_MPEG4_MAIN 3
52. #define FF_PROFILE_MPEG4_N_BIT 4
53. #define FF_PROFILE_MPEG4_SCALABLE_TEXTURE 5
54. #define FF_PROFILE_MPEG4_SIMPLE_FACE_ANIMATION 6
55. #define FF_PROFILE_MPEG4_BASIC_ANIMATED_TEXTURE 7
56. #define FF_PROFILE_MPEG4_HYBRID 8
57. #define FF_PROFILE_MPEG4_ADVANCED_REAL_TIME 9
58. #define FF_PROFILE_MPEG4_CORE_SCALABLE 10
59. #define FF_PROFILE_MPEG4_ADVANCED_CODING 11
60. #define FF_PROFILE_MPEG4_ADVANCED_CORE 12
61. #define FF_PROFILE_MPEG4_ADVANCED_SCALABLE_TEXTURE 13
62. #define FF_PROFILE_MPEG4_SIMPLE_STUDIO 14
63. #define FF_PROFILE_MPEG4_ADVANCED_SIMPLE 15
```

版权声明：本文为博主原创文章，未经博主允许不得转载。<https://blog.csdn.net/leixiaohua1020/article/details/14214859>

文章标签： [ffmpeg](#) [AVCodecContext](#) [源代码](#) [解码](#) [视频](#)

个人分类：[FFMPEG](#)

所属专栏：[FFmpeg](#)

此PDF由spyyg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com