libtheora Reference Manual

1.0alpha4

Generated by Doxygen 1.3.9.1

Wed Dec 15 00:13:21 2004

Contents

Chapter 1

libtheora Main Page

1.1 Introduction

This is the documentation for the libtheora C API. libtheora is the reference implementation for `Theora`, a free video codec. Theora is derived from On2's VP3 codec with improved integration for Ogg multimedia formats by `Xiph.Org`.

Chapter 2

libtheora Data Structure Index

2.1 libtheora Data Structures

Here are the data structures with brief descriptions:

Chapter 3

libtheora File Index

3.1 libtheora File List

Here is a list of all documented files with brief descriptions:

Chapter 4

libtheora Data Structure Documentation

4.1 theora_comment Struct Reference

Comment header metadata.

`#include <theora.h>`

Data Fields

- char ** **user_comments**
  an array of comment string vectors
- int * **comment_lengths**
  an array of corresponding string vector lengths in bytes
- int **comments**
  the total number of comment string vectors
- char * **vendor**
  the vendor string identifying the encoder, null terminated

4.1.1 Detailed Description

Comment header metadata.

This structure holds the in-stream metadata corresponding to the 'comment' header packet.

Meta data is stored as a series of (tag, value) pairs, in length-encoded string vectors. The first occurence of the '=' character delimits the tag and value. A particular tag may occur more than once. The character set encoding for the strings is always utf-8, but the tag names are limited to case-insensitive ascii. See the spec for details.

In filling in this structure, **theora_decode_header()**(p. 18) will null-terminate the user_comment strings for safety. However, the bitstream format itself treats them as 8-bit clean, and so the length array should be treated as authoritative for their length.

The documentation for this struct was generated from the following file:

- **theora.h**

4.2 theora_info Struct Reference

Theora bitstream info.

`#include <theora.h>`

Data Fields

- ogg_uint32_t **width**
- ogg_uint32_t **height**
- ogg_uint32_t **frame_width**
- ogg_uint32_t **frame_height**
- ogg_uint32_t **offset_x**
- ogg_uint32_t **offset_y**
- ogg_uint32_t **fps_numerator**
- ogg_uint32_t **fps_denominator**
- ogg_uint32_t **aspect_numerator**
- ogg_uint32_t **aspect_denominator**
- **theora_colorspace colorspace**
- int **target_bitrate**
- int **quality**

  nominal quality setting, 0-63

- int **quick_p**

  quick encode/decode

- unsigned char **version_major**
- unsigned char **version_minor**
- unsigned char **version_subminor**
- void * **codec_setup**
- int **dropframes_p**
- int **keyframe_auto_p**
- ogg_uint32_t **keyframe_frequency**
- ogg_uint32_t **keyframe_frequency_force**
- ogg_uint32_t **keyframe_data_target_bitrate**
- ogg_int32_t **keyframe_auto_threshold**
- ogg_uint32_t **keyframe_mindistance**
- ogg_int32_t **noise_sensitivity**
- ogg_int32_t **sharpness**

4.2.1 Detailed Description

Theora bitstream info.

Contains the basic playback parameters for a stream, corresponds to the initial 'info' header packet.

Encoded theora frames must be a multiple of 16 is size; this is what the width and height members represent. To handle other sizes, a crop rectangle is specified in frame_height and frame_width, offset_x and offset_y. The offset and size should still be a power of 2 to avoid chroma sampling shifts.

Frame rate, in frames per second is stored as a rational fraction. So is the aspect ratio. Note that this refers to the aspect ratio of the frame pixels, not of the overall frame itself.

see the example code for use of the other parameters and good default settings for the encoder parameters.

The documentation for this struct was generated from the following file:

- **theora.h**

4.3 theora_state Struct Reference

Codec internal state and context.

`#include <theora.h>`

Data Fields

- **theora_info** * **i**
- ogg_int64_t **granulepos**
- void * **internal_encode**
- void * **internal_decode**

4.3.1 Detailed Description

Codec internal state and context.

The documentation for this struct was generated from the following file:

- **theora.h**

4.4 yuv_buffer Struct Reference

A YUV buffer for passing uncompressed frames to and from the codec.

`#include <theora.h>`

Data Fields

- int **y_width**
  width of the Y' luminance plane
- int **y_height**
  height of the luminance plane
- int **y_stride**
  offset in bytes between successive rows
- int **uv_width**
  height of the Cb and Cr chroma planes
- int **uv_height**
  width of the chroma planes
- int **uv_stride**
  offset between successive chroma rows
- unsigned char * **y**
  pointer to start of luminance data
- unsigned char * **u**
  pointer to start of Cb data
- unsigned char * **v**
  pointer to start of Cr data

4.4.1 Detailed Description

A YUV buffer for passing uncompressed frames to and from the codec.

This holds a Y'CbCr frame in planar format. The CbCr planes can be subsampled and have their own separate dimensions and row stride offsets. Note that the strides may be negative in some configurations. For theora the width and height of the largest plane must be a multiple of 16. The actual meaningful picture size and offset are stored in the **theora_info**(p. 9) structure; frames returned by the decoder my been to be cropped for display. All samples are 8 bits.

The documentation for this struct was generated from the following file:

- **theora.h**

Chapter 5

libtheora File Documentation

5.1 theora.h File Reference

The libtheora C API.

`#include <ogg/ogg.h>`

Data Structures

- struct **yuv_buffer**
  A YUV buffer for passing uncompressed frames to and from the codec.
- struct **theora_info**
  Theora bitstream info.
- struct **theora_state**
  Codec internal state and context.
- struct **theora_comment**
  Comment header metadata.

Defines

- #define **OC_FAULT** -1
  general failure
- #define **OC_EINVAL** -10
  library encountered invalid internal data
- #define **OC_DISABLED** -11
  requested action is disabled
- #define **OC_BADHEADER** -20
  header packet was corrupt/invalid

- #define **OC_NOTFORMAT** -21
  packet is not a theora packet
- #define **OC_VERSION** -22
  bitstream version is not handled
- #define **OC_IMPL** -23
  feature or action not implemented
- #define **OC_BADPACKET** -24
  packet is corrupt
- #define **OC_NEWPACKET** -25
  packet is an (ignorable) unhandled extension

Typedefs

- typedef **theora_comment theora_comment**
  Comment header metadata.

Enumerations

- enum **theora_colorspace** { **OC_CS_UNSPECIFIED**, **OC_CS_ITU_REC_470M**, **OC_CS_ITU_REC_470BG** }
  A Colorspace.

Functions

- const char * **theora_version_string** (void)
  Retrieve a human-readable string to identify the encoder vendor and version.
- ogg_uint32_t **theora_version_number** (void)
  Retrieve a 32-bit version number.
- int **theora_encode_init** (**theora_state** *th, **theora_info** *c)
  Initialize the theora encoder.
- int **theora_encode_YUVin** (**theora_state** *t, **yuv_buffer** *yuv)
  Submit a YUV buffer to the theora encoder.
- int **theora_encode_packetout** (**theora_state** *t, int last_p, ogg_packet *op)
  Request the next packet of encoded video.
- int **theora_encode_header** (**theora_state** *t, ogg_packet *op)
  Request a packet containing the initial header.

- int **theora_encode_comment** (**theora_comment** *tc, ogg_packet *op)

  Request a comment header packet from provided metadata.

- int **theora_encode_tables** (**theora_state** *t, ogg_packet *op)

  Request a packet containing the codebook tables for the stream.

- int **theora_decode_header** (**theora_info** *ci, **theora_comment** *cc, ogg_packet *op)

  Decode an Ogg packet, with the expectation that the packet contains an initial header, comment data or codebook tables.

- int **theora_decode_init** (**theora_state** *th, **theora_info** *c)

  Initialize a **theora_state**(p. 11) *handle for decoding.*

- int **theora_decode_packetin** (**theora_state** *th, ogg_packet *op)

  Input a packet containing encoded data into the theora decoder.

- int **theora_decode_YUVout** (**theora_state** *th, **yuv_buffer** *yuv)

  Output the next available frame of decoded YUV data.

- double **theora_granule_time** (**theora_state** *th, ogg_int64_t granulepos)

  Convert a granulepos to absolute time in seconds.

- int **theora_packet_isheader** (ogg_packet *op)

  Report whether a theora packet is a header or not This function does no verification beyond checking the header flag bit so it should not be used for bitstream identification; use **theora_decode_header()**(p. 18) *for that.*

- int **theora_packet_iskeyframe** (ogg_packet *op)

  Report whether a theora packet is a keyframe or not.

- ogg_int64_t **theora_granule_frame** (**theora_state** *th, ogg_int64_t granulepos)

  Convert a granulepos to an absolute frame number.

- void **theora_info_init** (**theora_info** *c)

  Initialize a **theora_info**(p. 9) *structure.*

- void **theora_info_clear** (**theora_info** *c)

  Clear a **theora_info**(p. 9) *structure.*

- void **theora_clear** (**theora_state** *t)

  Free all internal data associated with a **theora_state**(p. 11) *handle.*

- void **theora_comment_init** (**theora_comment** *tc)

  Initialize an allocated **theora_comment**(p. 7) *structure.*

- void **theora_comment_add** (**theora_comment** *tc, char *comment)

  Add a comment to an initialized **theora_comment**(p. 7) *structure.*

- void **theora_comment_add_tag** (**theora_comment** *tc, char *tag, char *value)
  Add a comment to an initialized **theora_comment**(p. 7) *structure.*
- char * **theora_comment_query** (**theora_comment** *tc, char *tag, int count)
  look up a comment value by tag
- int **theora_comment_query_count** (**theora_comment** *tc, char *tag)
  look up the number of instances of a tag
- void **theora_comment_clear** (**theora_comment** *tc)
  clears an allocated **theora_comment**(p. 7) *struct so that it can be freed.*

5.1.1 Detailed Description

The libtheora C API.

5.1.2 Typedef Documentation

5.1.2.1 typedef struct theora_comment theora_comment

Comment header metadata.

This structure holds the in-stream metadata corresponding to the 'comment' header packet.

Meta data is stored as a series of (tag, value) pairs, in length-encoded string vectors. The first occurence of the '=' character delimits the tag and value. A particular tag may occur more than once. The character set encoding for the strings is always utf-8, but the tag names are limited to case-insensitive ascii. See the spec for details.

In filling in this structure, **theora_decode_header()**(p. 18) will null-terminate the user_comment strings for safety. However, the bitstream format itself treats them as 8-bit clean, and so the length array should be treated as authoritative for their length.

5.1.3 Enumeration Type Documentation

5.1.3.1 enum theora_colorspace

A Colorspace.

Enumeration values:

OC_CS_UNSPECIFIED the colorspace is unknown or unspecified

OC_CS_ITU_REC_470M best option for 'NTSC' content

OC_CS_ITU_REC_470BG best option for 'PAL' content

5.1.4 Function Documentation

5.1.4.1 void theora_clear (theora_state * *t*)

Free all internal data associated with a **theora_state**(p. 11) handle.

Parameters:
t A **theora_state**(p. 11) handle.

5.1.4.2 void theora_comment_add (theora_comment * *tc*, char * *comment*)

Add a comment to an initialized **theora_comment**(p. 7) structure.

Parameters:
comment must be a null-terminated string encoding the comment in "TAG=the value" form

5.1.4.3 void theora_comment_add_tag (theora_comment * *tc*, char * *tag*, char * *value*)

Add a comment to an initialized **theora_comment**(p. 7) structure.

Parameters:
tag a null-terminated string containing the tag associated with the comment.

value the corresponding value as a null-terminated string Neither **theora_comment_add()**(p. 17) nor **theora_comment_add_tag()**(p. 17) support comments containing null values, although the bitstream format supports this. To add such comments you will need to manipulate the **theora_comment**(p. 7) structure directly

5.1.4.4 char* theora_comment_query (theora_comment * *tc*, char * *tag*, int *count*)

look up a comment value by tag

Parameters:
tc an initialized **theora_comment**(p. 7) structure

tag the tag to look up

count the instance of the tag. The same tag can appear multiple times, each with a distinct and ordered value, so an index is required to retrieve them all. Use **theora_comment_query_count()**(p. 17) to get the legal range for the count parameter

Returns:
a pointer to the queried tag's value

Return values:
NULL if no matching tag is found

5.1.4.5 int theora_comment_query_count (theora_comment * *tc*, char * *tag*)

look up the number of instances of a tag

Parameters:
tc an initialized **theora_comment**(p. 7) structure

tag the tag to look up

Returns:

the number on instances of a particular tag. Call this first when querying for a specific tag and then interate over the number of instances with separate calls to **theora_comment_query()**(p. 17) to retrieve all instances in order.

5.1.4.6 int theora_decode_header (theora_info * *ci*, theora_comment * *cc*, ogg_packet * *op*)

Decode an Ogg packet, with the expectation that the packet contains an initial header, comment data or codebook tables.

Parameters:

ci A **theora_info**(p. 9) structure to fill. This must have been previously initialized with **theora_info_init()**(p. 22). If *op* contains an initial header, **theora_decode_header()**(p. 18) will fill *ci* with the parsed header values. If *op* contains codebook tables, **theora_decode_header()**(p. 18) will parse these and attach an internal representation to *ci->codec_setup*.

cc A **theora_comment**(p. 7) structure to fill. If *op* contains comment data, **theora_decode_header()**(p. 18) will fill *cc* with the parsed comments.

op An ogg_packet structure which you expect contains an initial header, comment data or codebook tables.

Return values:

OC_BADHEADER *op* is NULL; OR the first byte of *op->packet* has the signature of an initial packet, but op is not a b_o_s packet; OR this packet has the signature of an initial header packet, but an initial header packet has already been seen; OR this packet has the signature of a comment packet, but the initial header has not yet been seen; OR this packet has the signature of a comment packet, but contains invalid data; OR this packet has the signature of codebook tables, but the initial header or comments have not yet been seen; OR this packet has the signature of codebook tables, but contains invalid data; OR the stream being decoded has a compatible version but this packet does not have the signature of a theora initial header, comments, or codebook packet

OC_VERSION The packet data of *op* is an initial header with a version which is incompatible with this version of libtheora.

OC_NEWPACKET the stream being decoded has an incompatible (future) version and contains an unknown signature.

0 Success

Note:

The normal usage is that **theora_decode_header()**(p. 18) be called on the first three packets of a theora logical bitstream in succession.

5.1.4.7 int theora_decode_init (theora_state * *th*, theora_info * *c*)

Initialize a **theora_state**(p. 11) handle for decoding.

Parameters:

th The **theora_state**(p. 11) handle to initialize.

c A **theora_info**(p. 9) struct filled with the desired decoding parameters. This is of course usually obtained from a previous call to **theora_decode_header()**(p. 18).

Returns:
0 Success

5.1.4.8 int theora_decode_packetin (theora_state * *th*, ogg_packet * *op*)

Input a packet containing encoded data into the theora decoder.

Parameters:
th A **theora_state**(p. 11) handle previously initialized for decoding.
op An ogg_packet containing encoded theora data.

Return values:
OC_BADPACKET *op* does not contain encoded video data

5.1.4.9 int theora_decode_YUVout (theora_state * *th*, yuv_buffer * *yuv*)

Output the next available frame of decoded YUV data.

Parameters:
th A **theora_state**(p. 11) handle previously initialized for decoding.
yuv A **yuv_buffer**(p. 12) in which libtheora should place the decoded data.

Return values:
0 Success

5.1.4.10 int theora_encode_comment (theora_comment * *tc*, ogg_packet * *op*)

Request a comment header packet from provided metadata.

A pointer to the comment data is placed in a user-provided ogg_packet structure.

Parameters:
tc A **theora_comment**(p. 7) structure filled with the desired metadata
op An ogg_packet structure to fill. libtheora will set all elements of this structure, including a pointer to the encoded comment data. The memory for the comment data is owned by libtheora.

Return values:
0 Success

5.1.4.11 int theora_encode_header (theora_state * *t*, ogg_packet * *op*)

Request a packet containing the initial header.

A pointer to the header data is placed in a user-provided ogg_packet structure.

Parameters:
t A **theora_state**(p. 11) handle previously initialized for encoding.

op An ogg_packet structure to fill. libtheora will set all elements of this structure, including a pointer to the header data. The memory for the header data is owned by libtheora.

Return values:

0 Success

5.1.4.12 int theora_encode_init (theora_state * *th*, theora_info * *c*)

Initialize the theora encoder.

Parameters:

th The **theora_state**(p. 11) handle to initialize for encoding.

ti A **theora_info**(p. 9) struct filled with the desired encoding parameters.

Returns:

0 Success

5.1.4.13 int theora_encode_packetout (theora_state * *t*, int *last_p*, ogg_packet * *op*)

Request the next packet of encoded video.

The encoded data is placed in a user-provided ogg_packet structure.

Parameters:

t A **theora_state**(p. 11) handle previously initialized for encoding.

last_p whether this is the last packet the encoder should produce.

op An ogg_packet structure to fill. libtheora will set all elements of this structure, including a pointer to encoded data. The memory for the encoded data is owned by libtheora.

Return values:

0 No internal storage exists OR no packet is ready

-1 The encoding process has completed

1 Success

5.1.4.14 int theora_encode_tables (theora_state * *t*, ogg_packet * *op*)

Request a packet containing the codebook tables for the stream.

A pointer to the codebook data is placed in a user-provided ogg_packet structure.

Parameters:

t A **theora_state**(p. 11) handle previously initialized for encoding.

op An ogg_packet structure to fill. libtheora will set all elements of this structure, including a pointer to the codebook data. The memory for the header data is owned by libtheora.

Return values:

0 Success

5.1.4.15 int theora_encode_YUVin (theora_state * *t*, yuv_buffer * *yuv*)

Submit a YUV buffer to the theora encoder.

Parameters:

t A **theora_state**(p. 11) handle previously initialized for encoding.

yuv A buffer of YUV data to encode.

Return values:

OC_EINVAL Encoder is not ready, or is finished.

-1 The size of the given frame differs from those previously input

0 Success

5.1.4.16 ogg_int64_t theora_granule_frame (theora_state * *th*, ogg_int64_t *granulepos*)

Convert a granulepos to an absolute frame number.

The granulepos is interpreted in the context of a given **theora_state**(p. 11) handle.

Parameters:

th A previously initialized **theora_state**(p. 11) handle (encode or decode)

granulepos The granulepos to convert.

Returns:

The frame number corresponding to *granulepos*.

Return values:

-1 The given granulepos is invalid (ie. negative)

5.1.4.17 double theora_granule_time (theora_state * *th*, ogg_int64_t *granulepos*)

Convert a granulepos to absolute time in seconds.

The granulepos is interpreted in the context of a given **theora_state**(p. 11) handle.

Parameters:

th A previously initialized **theora_state**(p. 11) handle (encode or decode)

granulepos The granulepos to convert.

Returns:

The absolute time in seconds corresponding to *granulepos*.

Return values:

-1 The given granulepos is invalid (ie. negative)

5.1.4.18 void theora_info_clear (theora_info * c)

Clear a **theora_info**(p. 9) structure.

All values within the given **theora_info**(p. 9) structure are cleared, and associated internal codec setup data is freed.

Parameters:

c A **theora_info**(p. 9) struct to initialize.

5.1.4.19 void theora_info_init (theora_info * c)

Initialize a **theora_info**(p. 9) structure.

All values within the given **theora_info**(p. 9) structure are initialized, and space is allocated within libtheora for internal codec setup data.

Parameters:

c A **theora_info**(p. 9) struct to initialize.

5.1.4.20 int theora_packet_isheader (ogg_packet * op)

Report whether a theora packet is a header or not This function does no verification beyond checking the header flag bit so it should not be used for bitstream identification; use **theora_decode_header()**(p. 18) for that.

Parameters:

op An ogg_packet containing encoded theora data.

Return values:

1 The packet is a header packet

0 The packet is not a header packet (and so contains frame data)

5.1.4.21 int theora_packet_iskeyframe (ogg_packet * op)

Report whether a theora packet is a keyframe or not.

Parameters:

op An ogg_packet containing encoded theora data.

Return values:

1 The packet contains a keyframe image

0 The packet is contains an interframe delta

-1 the packet is not an image data packet at all

5.1.4.22 ogg_uint32_t theora_version_number (void)

Retrieve a 32-bit version number.

This number is composed of a 16-bit major version, 8-bit minor version and 8 bit sub-version, composed as follows:

```
(VERSION_MAJOR$<$$<$16) + (VERSION_MINOR$<$$<$8) + (VERSION_SUB)
```

Returns:
the version number.

5.1.4.23 const char* theora_version_string (void)

Retrieve a human-readable string to identify the encoder vendor and version.

Returns:
a version string.

Index