opusfile

0.12

Generated by Doxygen 1.8.17

Chapter 1

Main Page

1.1 Introduction

This is the documentation for the `libopusfile` C API.

The `libopusfile` package provides a convenient high-level API for decoding and basic manipulation of all Ogg Opus audio streams. `libopusfile` is implemented as a layer on top of Xiph.Org's reference `libogg` and `libopus` libraries.

`libopusfile` provides several sets of built-in routines for file/stream access, and may also use custom stream I/O routines provided by the embedded environment. There are built-in I/O routines provided for ANSI-compliant `stdio` (`FILE *`), memory buffers, and URLs (including <file:> URLs, plus optionally <http:> and <https:> URLs).

1.2 Organization

The main API is divided into several sections:

- Opening and Closing
- Stream Information
- Decoding
- Seeking

Several additional sections are not tied to the main API.

- Abstract Stream Reading Interface
- Header Information
- Error Codes

1.3 Overview

The libopusfile API always decodes files to 48 kHz. The original sample rate is not preserved by the lossy compression, though it is stored in the header to allow you to resample to it after decoding (the libopusfile API does not currently provide a resampler, but the the Speex resampler is a good choice if you need one). In general, if you are playing back the audio, you should leave it at 48 kHz, provided your audio hardware supports it. When decoding to a file, it may be worth resampling back to the original sample rate, so as not to surprise users who might not expect the sample rate to change after encoding to Opus and decoding.

Opus files can contain anywhere from 1 to 255 channels of audio. The channel mappings for up to 8 channels are the same as the Vorbis mappings. A special stereo API can convert everything to 2 channels, making it simple to support multichannel files in an application which only has stereo output. Although the libopusfile ABI provides support for the theoretical maximum number of channels, the current implementation does not support files with more than 8 channels, as they do not have well-defined channel mappings.

Like all Ogg files, Opus files may be "chained". That is, multiple Opus files may be combined into a single, longer file just by concatenating the original files. This is commonly done in internet radio streaming, as it allows the title and artist to be updated each time the song changes, since each link in the chain includes its own set of metadata.

libopusfile fully supports chained files. It will decode the first Opus stream found in each link of a chained file (ignoring any other streams that might be concurrently multiplexed with it, such as a video stream).

The channel count can also change between links. If your application is not prepared to deal with this, it can use the stereo API to ensure the audio from all links will always get decoded into a common format. Since libopusfile always decodes to 48 kHz, you do not have to worry about the sample rate changing between links (as was possible with Vorbis). This makes application support for chained files with libopusfile very easy.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 Error Codes

4.2 Header Information

4.3 URL Reading Options

4.4 Abstract Stream Reading Interface

4.5 Opening and Closing

4.6 Stream Information

4.7 Seeking

4.8 Decoding

Chapter 5

Data Structure Documentation

5.1 OpusFileCallbacks Struct Reference

The callbacks used to access non-`FILE` stream resources.

`#include <opusfile.h>`

Data Fields

- op_read_func read
  Used to read data from the stream.
- op_seek_func seek
  Used to seek in the stream.
- op_tell_func tell
  Used to return the current read position in the stream.
- op_close_func close
  Used to close the stream when the decoder is freed.

5.1.1 Detailed Description

The callbacks used to access non-`FILE` stream resources.

The function prototypes are basically the same as for the stdio functions `fread()`, `fseek()`, `ftell()`, and `fclose()`. The differences are that the `FILE *` arguments have been replaced with a `void *`, which is to be used as a pointer to whatever internal data these functions might need, that seek and tell take and return 64-bit offsets, and that seek *must* return -1 if the stream is unseekable.

5.1.2 Field Documentation

5.1.2.1 read

`op_read_func OpusFileCallbacks::read`

Used to read data from the stream.

This must not be `NULL`.

5.1.2.2 seek

`op_seek_func OpusFileCallbacks::seek`

Used to seek in the stream.

This may be `NULL` if seeking is not implemented.

5.1.2.3 tell

`op_tell_func OpusFileCallbacks::tell`

Used to return the current read position in the stream.

This may be `NULL` if seeking is not implemented.

5.1.2.4 close

`op_close_func OpusFileCallbacks::close`

Used to close the stream when the decoder is freed.

This may be `NULL` to leave the stream open.

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

- opusfile.h

5.2 OpusHead Struct Reference

Ogg Opus bitstream information.

`#include <opusfile.h>`

Data Fields

- int version
  The Ogg Opus format version, in the range 0...255.
- int channel_count
  The number of channels, in the range 1...255.
- unsigned pre_skip
  The number of samples that should be discarded from the beginning of the stream.
- opus_uint32 input_sample_rate
  The sampling rate of the original input.
- int output_gain
  The gain to apply to the decoded output, in dB, as a Q8 value in the range -32768...32767.
- int mapping_family
  The channel mapping family, in the range 0...255.
- int stream_count
  The number of Opus streams in each Ogg packet, in the range 1...255.
- int coupled_count
  The number of coupled Opus streams in each Ogg packet, in the range 0...127.
- unsigned char mapping [OPUS_CHANNEL_COUNT_MAX]
  The mapping from coded stream channels to output channels.

5.2.1 Detailed Description

Ogg Opus bitstream information.

This contains the basic playback parameters for a stream, and corresponds to the initial ID header packet of an Ogg Opus stream.

5.2.2 Field Documentation

5.2.2.1 version

```
int OpusHead::version
```

The Ogg Opus format version, in the range 0...255.

The top 4 bits represent a "major" version, and the bottom four bits represent backwards-compatible "minor" revisions. The current specification describes version 1. This library will recognize versions up through 15 as backwards compatible with the current specification. An earlier draft of the specification described a version 0, but the only difference between version 1 and version 0 is that version 0 did not specify the semantics for handling the version field.

5.2.2.2 input_sample_rate

```
opus_uint32 OpusHead::input_sample_rate
```

The sampling rate of the original input.

All Opus audio is coded at 48 kHz, and should also be decoded at 48 kHz for playback (unless the target hardware does not support this sampling rate). However, this field may be used to resample the audio back to the original sampling rate, for example, when saving the output to a file.

5.2.2.3 output_gain

```
int OpusHead::output_gain
```

The gain to apply to the decoded output, in dB, as a Q8 value in the range -32768...32767.

The `libopusfile` API will automatically apply this gain to the decoded output before returning it, scaling it by `pow(10,output_gain/(20.0*256))`. You can adjust this behavior with op_set_gain_offset().

5.2.2.4 mapping_family

```
int OpusHead::mapping_family
```

The channel mapping family, in the range 0...255.

Channel mapping family 0 covers mono or stereo in a single stream. Channel mapping family 1 covers 1 to 8 channels in one or more streams, using the Vorbis speaker assignments. Channel mapping family 255 covers 1 to 255 channels in one or more streams, but without any defined speaker assignment.

5.2.2.5 coupled_count

```
int OpusHead::coupled_count
```

The number of coupled Opus streams in each Ogg packet, in the range 0...127.

This must satisfy `0 <= coupled_count <= stream_count` and `coupled_count + stream_count <= 255`. The coupled streams appear first, before all uncoupled streams, in an Ogg Opus packet.

5.2.2.6 mapping

```
unsigned char OpusHead::mapping[OPUS_CHANNEL_COUNT_MAX]
```

The mapping from coded stream channels to output channels.

Let `index=mapping[k]` be the value for channel `k`. If `index<2*coupled_count`, then it refers to the left channel from stream `(index/2)` if even, and the right channel from stream `(index/2)` if odd. Otherwise, it refers to the output of the uncoupled stream `(index-coupled_count)`.

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

- opusfile.h

5.3 OpusPictureTag Struct Reference

The contents of a METADATA_BLOCK_PICTURE tag.

```
#include <opusfile.h>
```

Data Fields

- opus_int32 type

  The picture type according to the ID3v2 APIC frame:
- char * mime_type

  The MIME type of the picture, in printable ASCII characters 0x20-0x7E.
- char * description

  The description of the picture, in UTF-8.
- opus_uint32 width

  The width of the picture in pixels.
- opus_uint32 height

  The height of the picture in pixels.
- opus_uint32 depth

  The color depth of the picture in bits-per-pixel (not bits-per-channel).
- opus_uint32 colors

  For indexed-color pictures (e.g., GIF), the number of colors used, or 0 for non-indexed pictures.
- opus_uint32 data_length

  The length of the picture data in bytes.
- unsigned char * data

  The binary picture data.
- int format

  The format of the picture data, if known.

5.3.1 Detailed Description

The contents of a METADATA_BLOCK_PICTURE tag.

5.3.2 Field Documentation

5.3.2.1 type

```
opus_int32 OpusPictureTag::type
```

The picture type according to the ID3v2 APIC frame:

1. Other
2. 32x32 pixels 'file icon' (PNG only)
3. Other file icon
4. Cover (front)
5. Cover (back)
6. Leaflet page
7. Media (e.g. label side of CD)
8. Lead artist/lead performer/soloist

9. Artist/performer
10. Conductor
11. Band/Orchestra
12. Composer
13. Lyricist/text writer
14. Recording Location
15. During recording
16. During performance
17. Movie/video screen capture
18. A bright colored fish
19. Illustration
20. Band/artist logotype
21. Publisher/Studio logotype

Others are reserved and should not be used. There may only be one each of picture type 1 and 2 in a file.

5.3.2.2 mime_type

```
char* OpusPictureTag::mime_type
```

The MIME type of the picture, in printable ASCII characters 0x20-0x7E.

The MIME type may also be "-->" to signify that the data part is a URL pointing to the picture instead of the picture data itself. In this case, a terminating NUL is appended to the URL string in data, but data_length is set to the length of the string excluding that terminating NUL.

5.3.2.3 format

```
int OpusPictureTag::format
```

The format of the picture data, if known.

One of

- #OP_PIC_FORMAT_UNKNOWN,
- #OP_PIC_FORMAT_URL,
- #OP_PIC_FORMAT_JPEG,
- #OP_PIC_FORMAT_PNG, or
- #OP_PIC_FORMAT_GIF.

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

- opusfile.h

5.4 OpusServerInfo Struct Reference

HTTP/Shoutcast/Icecast server information associated with a URL.

```
#include <opusfile.h>
```

Data Fields

- char * name

  The name of the server (icy-name/ice-name).
- char * description

  A short description of the server (icy-description/ice-description).
- char * genre

  The genre the server falls under (icy-genre/ice-genre).
- char * url

  The homepage for the server (icy-url/ice-url).
- char * server

  The software used by the origin server (Server).
- char * content_type

  The media type of the entity sent to the recepient (Content-Type).
- opus_int32 bitrate_kbps

  The nominal stream bitrate in kbps (icy-br/ice-bitrate).
- int is_public

  Flag indicating whether the server is public (1) or not (0) (icy-pub/ice-public).
- int is_ssl

  Flag indicating whether the server is using HTTPS instead of HTTP.

5.4.1 Detailed Description

HTTP/Shoutcast/Icecast server information associated with a URL.

5.4.2 Field Documentation

5.4.2.1 name

```
char* OpusServerInfo::name
```

The name of the server (icy-name/ice-name).

This is NULL if there was no icy-name or ice-name header.

5.4.2.2 description

```
char* OpusServerInfo::description
```

A short description of the server (icy-description/ice-description).

This is `NULL` if there was no `icy-description` or `ice-description` header.

5.4.2.3 genre

```
char* OpusServerInfo::genre
```

The genre the server falls under (icy-genre/ice-genre).

This is `NULL` if there was no `icy-genre` or `ice-genre` header.

5.4.2.4 url

```
char* OpusServerInfo::url
```

The homepage for the server (icy-url/ice-url).

This is `NULL` if there was no `icy-url` or `ice-url` header.

5.4.2.5 server

```
char* OpusServerInfo::server
```

The software used by the origin server (Server).

This is `NULL` if there was no `Server` header.

5.4.2.6 content_type

```
char* OpusServerInfo::content_type
```

The media type of the entity sent to the recepient (Content-Type).

This is `NULL` if there was no `Content-Type` header.

5.4.2.7 bitrate_kbps

```
opus_int32 OpusServerInfo::bitrate_kbps
```

The nominal stream bitrate in kbps (icy-br/ice-bitrate).

This is `-1` if there was no `icy-br` or `ice-bitrate` header.

5.4.2.8 is_public

```
int OpusServerInfo::is_public
```

Flag indicating whether the server is public (`1`) or not (`0`) (icy-pub/ice-public).

This is `-1` if there was no `icy-pub` or `ice-public` header.

5.4.2.9 is_ssl

```
int OpusServerInfo::is_ssl
```

Flag indicating whether the server is using HTTPS instead of HTTP.

This is `0` unless HTTPS is being used. This may not match the protocol used in the original URL if there were redirections.

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

- opusfile.h

5.5 OpusTags Struct Reference

The metadata from an Ogg Opus stream.

`#include <opusfile.h>`

Data Fields

- char ** user_comments
  The array of comment string vectors.
- int * comment_lengths
  An array of the corresponding length of each vector, in bytes.
- int comments
  The total number of comment streams.
- char * vendor
  The null-terminated vendor string.

5.5.1 Detailed Description

The metadata from an Ogg Opus stream.

This structure holds the in-stream metadata corresponding to the 'comment' header packet of an Ogg Opus stream. The comment header is meant to be used much like someone jotting a quick note on the label of a CD. It should be a short, to the point text note that can be more than a couple words, but not more than a short paragraph.

The metadata is stored as a series of (tag, value) pairs, in length-encoded string vectors, using the same format as Vorbis (without the final "framing bit"), Theora, and Speex, except for the packet header. The first occurrence of the '=' character delimits the tag and value. A particular tag may occur more than once, and order is significant. The character set encoding for the strings is always UTF-8, but the tag names are limited to ASCII, and treated as case-insensitive. See `the Vorbis comment header specification` for details.

In filling in this structure, `libopusfile` will null-terminate the user_comments strings for safety. However, the bit-stream format itself treats them as 8-bit clean vectors, possibly containing NUL characters, so the comment_lengths array should be treated as their authoritative length.

This structure is binary and source-compatible with a `vorbis_comment`, and pointers to it may be freely cast to `vorbis_comment` pointers, and vice versa. It is provided as a separate type to avoid introducing a compile-time dependency on the libvorbis headers.

5.5.2 Field Documentation

5.5.2.1 vendor

```
char* OpusTags::vendor
```

The null-terminated vendor string.

This identifies the software used to encode the stream.

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

- opusfile.h

Index