opusfile

0.12-12-gb23e611

Generated by Doxygen 1.9.1

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

List of possible error codes

Many of the functions in this library return a negative error code when a function fails.

This list provides a brief explanation of the common errors. See each individual function for more details on what a specific error code means in that context.

- #define OP_FALSE (-1)

  A request did not succeed.
- #define OP_EOF (-2)

  Currently not used externally.
- #define OP_HOLE (-3)

  There was a hole in the page sequence numbers (e.g., a page was corrupt or missing).
- #define OP_EREAD (-128)

  An underlying read, seek, or tell operation failed when it should have succeeded.
- #define OP_EFAULT (-129)

  A `NULL` pointer was passed where one was unexpected, or an internal memory allocation failed, or an internal library error was encountered.
- #define OP_EIMPL (-130)

  The stream used a feature that is not implemented, such as an unsupported channel family.
- #define OP_EINVAL (-131)

  One or more parameters to a function were invalid.
- #define OP_ENOTFORMAT (-132)

  A purported Ogg Opus stream did not begin with an Ogg page, a purported header packet did not start with one of the required strings, "OpusHead" or "OpusTags", or a link in a chained file was encountered that did not contain any logical Opus streams.
- #define OP_EBADHEADER (-133)

  A required header packet was not properly formatted, contained illegal values, or was missing altogether.
- #define OP_EVERSION (-134)

  The ID header contained an unrecognized version number.
- #define OP_ENOTAUDIO (-135)

  Currently not used at all.
- #define OP_EBADPACKET (-136)

An audio packet failed to decode properly.

- #define OP_EBADLINK (-137)

  We failed to find data we had seen before, or the bitstream structure was sufficiently malformed that seeking to the target destination was impossible.
- #define OP_ENOSEEK (-138)

  An operation that requires seeking was requested on an unseekable stream.
- #define OP_EBADTIMESTAMP (-139)

  The first or last granule position of a link failed basic validity checks.

4.1.1 Detailed Description

4.1.2 Macro Definition Documentation

4.1.2.1 OP_EBADPACKET

```
#define OP_EBADPACKET (-136)
```

An audio packet failed to decode properly.

This is usually caused by a multistream Ogg packet where the durations of the individual Opus packets contained in it are not all the same.

4.2 Header Information

Data Structures

- struct OpusHead

  Ogg Opus bitstream information.
- struct OpusTags

  The metadata from an Ogg Opus stream.
- struct OpusPictureTag

  The contents of a METADATA_BLOCK_PICTURE tag.

Macros

- #define OPUS_CHANNEL_COUNT_MAX (255)

  The maximum number of channels in an Ogg Opus stream.

Functions for manipulating header data

These functions manipulate the OpusHead and OpusTags structures, which describe the audio parameters and tag-value metadata, respectively.

These can be used to query the headers returned by `libopusfile`, or to parse Opus headers from sources other than an Ogg Opus stream, provided they use the same format.

- OP_WARN_UNUSED_RESULT int opus_head_parse (OpusHead ∗_head, const unsigned char ∗_data, size_t _len) OP_ARG_NONNULL(2)

  Parses the contents of the ID header packet of an Ogg Opus stream.
- ogg_int64_t opus_granule_sample (const OpusHead ∗_head, ogg_int64_t _gp) OP_ARG_NONNULL(1)

  Converts a granule position to a sample offset for a given Ogg Opus stream.
- OP_WARN_UNUSED_RESULT int opus_tags_parse (OpusTags ∗_tags, const unsigned char ∗_data, size↩ _t _len) OP_ARG_NONNULL(2)

  Parses the contents of the 'comment' header packet of an Ogg Opus stream.
- int opus_tags_copy (OpusTags ∗_dst, const OpusTags ∗_src) OP_ARG_NONNULL(1)

  Performs a deep copy of an OpusTags structure.
- void opus_tags_init (OpusTags ∗_tags) OP_ARG_NONNULL(1)

  Initializes an OpusTags structure.
- int opus_tags_add (OpusTags ∗_tags, const char ∗_tag, const char ∗_value) OP_ARG_NONNULL(1) OP↩ _ARG_NONNULL(2) OP_ARG_NONNULL(3)

  Add a (tag, value) pair to an initialized OpusTags structure.
- int opus_tags_add_comment (OpusTags ∗_tags, const char ∗_comment) OP_ARG_NONNULL(1) OP_↩ ARG_NONNULL(2)

  Add a comment to an initialized OpusTags structure.
- int opus_tags_set_binary_suffix (OpusTags ∗_tags, const unsigned char ∗_data, int _len) OP_ARG_↩ NONNULL(1)

  Replace the binary suffix data at the end of the packet (if any).
- const char ∗ opus_tags_query (const OpusTags ∗_tags, const char ∗_tag, int _count) OP_ARG_↩ NONNULL(1) OP_ARG_NONNULL(2)

  Look up a comment value by its tag.
- int opus_tags_query_count (const OpusTags ∗_tags, const char ∗_tag) OP_ARG_NONNULL(1) OP_ARG↩ _NONNULL(2)

  Look up the number of instances of a tag.
- const unsigned char ∗ opus_tags_get_binary_suffix (const OpusTags ∗_tags, int ∗_len) OP_ARG_↩ NONNULL(1) OP_ARG_NONNULL(2)

  Retrieve the binary suffix data at the end of the packet (if any).
- int opus_tags_get_album_gain (const OpusTags ∗_tags, int ∗_gain_q8) OP_ARG_NONNULL(1) OP_ARG↩ _NONNULL(2)

  Get the album gain from an R128_ALBUM_GAIN tag, if one was specified.
- int opus_tags_get_track_gain (const OpusTags ∗_tags, int ∗_gain_q8) OP_ARG_NONNULL(1) OP_ARG↩ _NONNULL(2)

  Get the track gain from an R128_TRACK_GAIN tag, if one was specified.
- void opus_tags_clear (OpusTags ∗_tags) OP_ARG_NONNULL(1)

  Clears the OpusTags structure.
- int opus_tagcompare (const char ∗_tag_name, const char ∗_comment)

  Check if _comment is an instance of a _tag_name tag.
- int opus_tagncompare (const char ∗_tag_name, int _tag_len, const char ∗_comment)

  Check if _comment is an instance of a _tag_name tag.
- OP_WARN_UNUSED_RESULT int opus_picture_tag_parse (OpusPictureTag ∗_pic, const char ∗_tag) OP↩ _ARG_NONNULL(1) OP_ARG_NONNULL(2)

Parse a single METADATA_BLOCK_PICTURE tag.

- void opus_picture_tag_init (OpusPictureTag ∗_pic) OP_ARG_NONNULL(1)

  Initializes an OpusPictureTag structure.

- void opus_picture_tag_clear (OpusPictureTag ∗_pic) OP_ARG_NONNULL(1)

  Clears the OpusPictureTag structure.

Picture tag image formats

- #define OP_PIC_FORMAT_UNKNOWN (-1)

  The MIME type was not recognized, or the image data did not match the declared MIME type.

- #define OP_PIC_FORMAT_URL (0)

  The MIME type indicates the image data is really a URL.

- #define OP_PIC_FORMAT_JPEG (1)

  The image is a JPEG.

- #define OP_PIC_FORMAT_PNG (2)

  The image is a PNG.

- #define OP_PIC_FORMAT_GIF (3)

  The image is a GIF.

4.2.1 Detailed Description

4.2.2 Function Documentation

4.2.2.1 opus_head_parse()

```
OP_WARN_UNUSED_RESULT int opus_head_parse (
            OpusHead ∗ _head,
            const unsigned char ∗ _data,
            size_t _len )
```

Parses the contents of the ID header packet of an Ogg Opus stream.

Parameters

out	*_head*	Returns the contents of the parsed packet. The contents of this structure are untouched on error. This may be `NULL` to merely test the header for validity.
in	*_data*	The contents of the ID header packet.
	_len	The number of bytes of data in the ID header packet.

Returns

0 on success or a negative value on error.

Return values

OP_ENOTFORMAT	If the data does not start with the "OpusHead" string.
OP_EVERSION	If the version field signaled a version this library does not know how to parse.
OP_EIMPL	If the channel mapping family was 255, which general purpose players should not attempt to play.
OP_EBADHEADER	If the contents of the packet otherwise violate the Ogg Opus specification: • Insufficient data, • Too much data for the known minor versions, • An unrecognized channel mapping family, • Zero channels or too many channels, • Zero coded streams, • Too many coupled streams, or • An invalid channel mapping index.

4.2.2.2 opus_granule_sample()

```
ogg_int64_t opus_granule_sample (
            const OpusHead * _head,
            ogg_int64_t _gp )
```

Converts a granule position to a sample offset for a given Ogg Opus stream.

The sample offset is simply `_gp-_head->pre_skip`. Granule position values smaller than OpusHead::pre_skip correspond to audio that should never be played, and thus have no associated sample offset. This function returns -1 for such values. This function also correctly handles extremely large granule positions, which may have wrapped around to a negative number when stored in a signed ogg_int64_t value.

Parameters

_head	The OpusHead information from the ID header of the stream.
_gp	The granule position to convert.

Returns

The sample offset associated with the given granule position (counting at a 48 kHz sampling rate), or the special value -1 on error (i.e., the granule position was smaller than the pre-skip amount).

4.2.2.3 opus_tags_parse()

```
OP_WARN_UNUSED_RESULT int opus_tags_parse (
            OpusTags * _tags,
            const unsigned char * _data,
            size_t _len )
```

Parses the contents of the 'comment' header packet of an Ogg Opus stream.

Parameters

out	*_tags*	An uninitialized OpusTags structure. This returns the contents of the parsed packet. The contents of this structure are untouched on error. This may be `NULL` to merely test the header for validity.
in	*_data*	The contents of the 'comment' header packet.
	_len	The number of bytes of data in the 'info' header packet.

Return values

0	Success.
OP_ENOTFORMAT	If the data does not start with the "OpusTags" string.
OP_EBADHEADER	If the contents of the packet otherwise violate the Ogg Opus specification.
OP_EFAULT	If there wasn't enough memory to store the tags.

4.2.2.4 opus_tags_copy()

```
int opus_tags_copy (
            OpusTags * _dst,
            const OpusTags * _src )
```

Performs a deep copy of an OpusTags structure.

Parameters

_dst	The OpusTags structure to copy into. If this function fails, the contents of this structure remain untouched.
_src	The OpusTags structure to copy from.

Return values

0	Success.
OP_EFAULT	If there wasn't enough memory to copy the tags.

4.2.2.5 opus_tags_init()

```
void opus_tags_init (
            OpusTags * _tags )
```

Initializes an OpusTags structure.

This should be called on a freshly allocated OpusTags structure before attempting to use it.

Parameters

_tags	The OpusTags structure to initialize.

4.2.2.6 opus_tags_add()

```
int opus_tags_add (
            OpusTags * _tags,
            const char * _tag,
            const char * _value )
```

Add a (tag, value) pair to an initialized OpusTags structure.

Note

Neither opus_tags_add() nor opus_tags_add_comment() support values containing embedded NULs, although the bitstream format does support them. To add such tags, you will need to manipulate the OpusTags structure directly.

Parameters

_tags	The OpusTags structure to add the (tag, value) pair to.
_tag	A NUL-terminated, case-insensitive, ASCII string containing the tag to add (without an '=' character).
_value	A NUL-terminated UTF-8 containing the corresponding value.

Returns

0 on success, or a negative value on failure.

Return values

OP_EFAULT	An internal memory allocation failed.

4.2.2.7 opus_tags_add_comment()

```
int opus_tags_add_comment (
            OpusTags * _tags,
            const char * _comment )
```

Add a comment to an initialized OpusTags structure.

Note

Neither opus_tags_add_comment() nor opus_tags_add() support comments containing embedded NULs, although the bitstream format does support them. To add such tags, you will need to manipulate the OpusTags structure directly.

Parameters

_tags	The OpusTags structure to add the comment to.
_comment	A NUL-terminated UTF-8 string containing the comment in "TAG=value" form.

Returns

0 on success, or a negative value on failure.

Return values

OP_EFAULT	An internal memory allocation failed.

4.2.2.8 opus_tags_set_binary_suffix()

```
int opus_tags_set_binary_suffix (
             OpusTags * _tags,
             const unsigned char * _data,
             int _len )
```

Replace the binary suffix data at the end of the packet (if any).

Parameters

_tags	An initialized OpusTags structure.
_data	A buffer of binary data to append after the encoded user comments. The least significant bit of the first byte of this data must be set (to ensure the data is preserved by other editors).
_len	The number of bytes of binary data to append. This may be zero to remove any existing binary suffix data.

Returns

0 on success, or a negative value on error.

Return values

OP_EINVAL	*_len* was negative, or *_len* was positive but *_data* was `NULL` or the least significant bit of the first byte was not set.
OP_EFAULT	An internal memory allocation failed.

4.2.2.9 opus_tags_query()

```
const char* opus_tags_query (
             const OpusTags * _tags,
             const char * _tag,
             int _count )
```

Look up a comment value by its tag.

Parameters

_tags	An initialized OpusTags structure.
_tag	The tag to look up.
_count	The instance of the tag. The same tag can appear multiple times, each with a distinct value, so an index is required to retrieve them all. The order in which these values appear is significant and should be preserved. Use opus_tags_query_count() to get the legal range for the *_count* parameter.

Returns

A pointer to the queried tag's value. This points directly to data in the OpusTags structure. It should not be modified or freed by the application, and modifications to the structure may invalidate the pointer.

Return values

NULL	If no matching tag is found.

4.2.2.10 opus_tags_query_count()

```
int opus_tags_query_count (
             const OpusTags * _tags,
             const char * _tag )
```

Look up the number of instances of a tag.

Call this first when querying for a specific tag and then iterate over the number of instances with separate calls to opus_tags_query() to retrieve all the values for that tag in order.

Parameters

_tags	An initialized OpusTags structure.
_tag	The tag to look up.

Returns

The number of instances of this particular tag.

4.2.2.11 opus_tags_get_binary_suffix()

```
const unsigned char* opus_tags_get_binary_suffix (
             const OpusTags * _tags,
             int * _len )
```

Retrieve the binary suffix data at the end of the packet (if any).

Parameters

	_tags	An initialized OpusTags structure.
out	*_len*	Returns the number of bytes of binary suffix data returned.

Returns

A pointer to the binary suffix data, or `NULL` if none was present.

4.2.2.12 opus_tags_get_album_gain()

```
int opus_tags_get_album_gain (
            const OpusTags * _tags,
            int * _gain_q8 )
```

Get the album gain from an R128_ALBUM_GAIN tag, if one was specified.

This searches for the first R128_ALBUM_GAIN tag with a valid signed, 16-bit decimal integer value and returns the value. This routine is exposed merely for convenience for applications which wish to do something special with the album gain (i.e., display it). If you simply wish to apply the album gain instead of the header gain, you can use op_set_gain_offset() with an OP_ALBUM_GAIN type and no offset.

Parameters

	_tags	An initialized OpusTags structure.
out	*_gain_q8*	The album gain, in 1/256ths of a dB. This will lie in the range [-32768,32767], and should be applied in *addition* to the header gain. On error, no value is returned, and the previous contents remain unchanged.

Returns

0 on success, or a negative value on error.

Return values

OP_FALSE	There was no album gain available in the given tags.

4.2.2.13 opus_tags_get_track_gain()

```
int opus_tags_get_track_gain (
            const OpusTags * _tags,
            int * _gain_q8 )
```

Get the track gain from an R128_TRACK_GAIN tag, if one was specified.

This searches for the first R128_TRACK_GAIN tag with a valid signed, 16-bit decimal integer value and returns the value. This routine is exposed merely for convenience for applications which wish to do something special with the track gain (i.e., display it). If you simply wish to apply the track gain instead of the header gain, you can use op_set_gain_offset() with an OP_TRACK_GAIN type and no offset.

Parameters

	_tags	An initialized OpusTags structure.
out	*_gain_q8*	The track gain, in 1/256ths of a dB. This will lie in the range [-32768,32767], and should be applied in *addition* to the header gain. On error, no value is returned, and the previous contents remain unchanged.

Returns

0 on success, or a negative value on error.

Return values

OP_FALSE	There was no track gain available in the given tags.

4.2.2.14 opus_tags_clear()

```
void opus_tags_clear (
             OpusTags * _tags )
```

Clears the OpusTags structure.

This should be called on an OpusTags structure after it is no longer needed. It will free all memory used by the structure members.

Parameters

_tags	The OpusTags structure to clear.

4.2.2.15 opus_tagcompare()

```
int opus_tagcompare (
             const char * _tag_name,
             const char * _comment )
```

Check if *_comment* is an instance of a *_tag_name* tag.

See also

opus_tagncompare

Parameters

_tag_name	A NUL-terminated, case-insensitive, ASCII string containing the name of the tag to check for (without the terminating '=' character).
_comment	The comment string to check.

Returns

An integer less than, equal to, or greater than zero if *_comment* is found respectively, to be less than, to match, or be greater than a "tag=value" string whose tag matches *_tag_name*.

4.2.2.16 opus_tagncompare()

```
int opus_tagncompare (
            const char * _tag_name,
            int _tag_len,
            const char * _comment )
```

Check if *_comment* is an instance of a *_tag_name* tag.

This version is slightly more efficient than opus_tagcompare() if the length of the tag name is already known (e.g., because it is a constant).

See also

opus_tagcompare

Parameters

_tag_name	A case-insensitive ASCII string containing the name of the tag to check for (without the terminating '=' character).
_tag_len	The number of characters in the tag name. This must be non-negative.
_comment	The comment string to check.

Returns

An integer less than, equal to, or greater than zero if *_comment* is found respectively, to be less than, to match, or be greater than a "tag=value" string whose tag matches the first *_tag_len* characters of *_tag_name*.

4.2.2.17 opus_picture_tag_parse()

```
OP_WARN_UNUSED_RESULT int opus_picture_tag_parse (
            OpusPictureTag * _pic,
            const char * _tag )
```

Parse a single METADATA_BLOCK_PICTURE tag.

This decodes the BASE64-encoded content of the tag and returns a structure with the MIME type, description, image parameters (if known), and the compressed image data. If the MIME type indicates the presence of an image format we recognize (JPEG, PNG, or GIF) and the actual image data contains the magic signature associated with that format, then the OpusPictureTag::format field will be set to the corresponding format. This is provided as a convenience to avoid requiring applications to parse the MIME type and/or do their own format detection for the commonly used formats. In this case, we also attempt to extract the image parameters directly from the image data (overriding any that were present in the tag, which the specification says applications are not meant to rely on). The application must still provide its own support for actually decoding the image data and, if applicable, retrieving that data from URLs.

Parameters

`out`	*_pic*	Returns the parsed picture data. No sanitation is done on the type, MIME type, or description fields, so these might return invalid values. The contents of this structure are left unmodified on failure.
	_tag	The METADATA_BLOCK_PICTURE tag contents. The leading "METADATA_BLOCK_PICTURE=" portion is optional, to allow the function to be used on either directly on the values in OpusTags::user_comments or on the return value of opus_tags_query().

Returns

0 on success or a negative value on error.

Return values

OP_ENOTFORMAT	The METADATA_BLOCK_PICTURE contents were not valid.
OP_EFAULT	There was not enough memory to store the picture tag contents.

4.2.2.18 opus_picture_tag_init()

```
void opus_picture_tag_init (
            OpusPictureTag * _pic )
```

Initializes an OpusPictureTag structure.

This should be called on a freshly allocated OpusPictureTag structure before attempting to use it.

Parameters

_pic	The OpusPictureTag structure to initialize.

4.2.2.19 opus_picture_tag_clear()

```
void opus_picture_tag_clear (
            OpusPictureTag * _pic )
```

Clears the OpusPictureTag structure.

This should be called on an OpusPictureTag structure after it is no longer needed. It will free all memory used by the structure members.

Parameters

_pic	The OpusPictureTag structure to clear.

4.3 URL Reading Options

Data Structures

- struct OpusServerInfo

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

URL reading options

Options for op_url_stream_create() and associated functions.

These allow you to provide proxy configuration parameters, skip SSL certificate checks, etc. Options are processed in order, and if the same option is passed multiple times, only the value specified by the last occurrence has an effect (unless otherwise specified). They may be expanded in the future.

- void opus_server_info_init (OpusServerInfo *_info) OP_ARG_NONNULL(1)

  Initializes an OpusServerInfo structure.
- void opus_server_info_clear (OpusServerInfo *_info) OP_ARG_NONNULL(1)

  Clears the OpusServerInfo structure.
- #define OP_SSL_SKIP_CERTIFICATE_CHECK(_b)

  Skip the certificate check when connecting via TLS/SSL (https).
- #define OP_HTTP_PROXY_HOST(_host)

  Proxy connections through the given host.
- #define OP_HTTP_PROXY_PORT(_port)

  Use the given port when proxying connections.
- #define OP_HTTP_PROXY_USER(_user)

  Use the given user name for authentication when proxying connections.
- #define OP_HTTP_PROXY_PASS(_pass)

  Use the given password for authentication when proxying connections.
- #define OP_GET_SERVER_INFO(_info)

  Parse information about the streaming server (if any) and return it.

4.3.1 Detailed Description

4.3.2 Macro Definition Documentation

4.3.2.1 OP_SSL_SKIP_CERTIFICATE_CHECK

```
#define OP_SSL_SKIP_CERTIFICATE_CHECK(
              _b )
```

Skip the certificate check when connecting via TLS/SSL (https).

Parameters

↩ _↩ *b*	`opus_int32`: Whether or not to skip the certificate check. The check will be skipped if *_b* is non-zero, and will not be skipped if *_b* is zero.

4.3.2.2 OP_HTTP_PROXY_HOST

```
#define OP_HTTP_PROXY_HOST(
              _host )
```

Proxy connections through the given host.

If no port is specified via OP_HTTP_PROXY_PORT, the port number defaults to 8080 (http-alt). All proxy parameters are ignored for non-http and non-https URLs.

Parameters

_host	`const char *`: The proxy server hostname. This may be `NULL` to disable the use of a proxy server.

4.3.2.3 OP_HTTP_PROXY_PORT

```
#define OP_HTTP_PROXY_PORT(
              _port )
```

Use the given port when proxying connections.

This option only has an effect if OP_HTTP_PROXY_HOST is specified with a non-`NULL` *_host*. If this option is not provided, the proxy port number defaults to 8080 (http-alt). All proxy parameters are ignored for non-http and non-https URLs.

Parameters

_port	`opus_int32`: The proxy server port. This must be in the range 0...65535 (inclusive), or the URL function this is passed to will fail.

4.3.2.4 OP_HTTP_PROXY_USER

```
#define OP_HTTP_PROXY_USER(
              _user )
```

Use the given user name for authentication when proxying connections.

All proxy parameters are ignored for non-http and non-https URLs.

Parameters

_user	const char ∗: The proxy server user name. This may be `NULL` to disable proxy authentication. A non-`NULL` value only has an effect if OP_HTTP_PROXY_HOST and OP_HTTP_PROXY_PASS are also specified with non-`NULL` arguments.

4.3.2.5 OP_HTTP_PROXY_PASS

```
#define OP_HTTP_PROXY_PASS(
              _pass )
```

Use the given password for authentication when proxying connections.

All proxy parameters are ignored for non-http and non-https URLs.

Parameters

_pass	const char ∗: The proxy server password. This may be `NULL` to disable proxy authentication. A non-`NULL` value only has an effect if OP_HTTP_PROXY_HOST and OP_HTTP_PROXY_USER are also specified with non-`NULL` arguments.

4.3.2.6 OP_GET_SERVER_INFO

```
#define OP_GET_SERVER_INFO(
              _info )
```

Parse information about the streaming server (if any) and return it.

Very little validation is done. In particular, OpusServerInfo::url may not be a valid URL, OpusServerInfo::bitrate_kbps may not really be in kbps, and OpusServerInfo::content_type may not be a valid MIME type. The character set of the string fields is not specified anywhere, and should not be assumed to be valid UTF-8.

Parameters

_info	OpusServerInfo ∗: Returns information about the server. If there is any error opening the stream, the contents of this structure remain unmodified. On success, fills in the structure with the server information that was available, if any. After a successful return, the contents of this structure should be freed by calling opus_server_info_clear().

4.3.3 Function Documentation

4.3.3.1 opus_server_info_init()

```
void opus_server_info_init (
            OpusServerInfo * _info )
```

Initializes an OpusServerInfo structure.

All fields are set as if the corresponding header was not available.

Parameters

_info	The OpusServerInfo structure to initialize.

Note

If you use this function, you must link against `libopusurl`.

4.3.3.2 opus_server_info_clear()

```
void opus_server_info_clear (
            OpusServerInfo * _info )
```

Clears the OpusServerInfo structure.

This should be called on an OpusServerInfo structure after it is no longer needed. It will free all memory used by the structure members.

Parameters

_info	The OpusServerInfo structure to clear.

Note

If you use this function, you must link against `libopusurl`.

4.4 Abstract Stream Reading Interface

Data Structures

- struct OpusFileCallbacks

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

Functions for reading from streams

These functions define the interface used to read from and seek in a stream of data.

A stream does not need to implement seeking, but the decoder will not be able to seek if it does not do so. These functions also include some convenience routines for working with standard `FILE` pointers, complete streams stored in a single block of memory, or URLs.

- typedef int(∗ op_read_func) (void ∗_stream, unsigned char ∗_ptr, int _nbytes)

  Reads up to _nbytes bytes of data from _stream.
- typedef int(∗ op_seek_func) (void ∗_stream, opus_int64 _offset, int _whence)

  Sets the position indicator for _stream.
- typedef opus_int64(∗ op_tell_func) (void ∗_stream)

  Obtains the current value of the position indicator for _stream.
- typedef int(∗ op_close_func) (void ∗_stream)

  Closes the underlying stream.
- OP_WARN_UNUSED_RESULT void ∗ op_fopen (OpusFileCallbacks ∗_cb, const char ∗_path, const char ∗_mode) OP_ARG_NONNULL(1) OP_ARG_NONNULL(2) OP_ARG_NONNULL(3)

  Opens a stream with `fopen()` and fills in a set of callbacks that can be used to access it.
- OP_WARN_UNUSED_RESULT void ∗ op_fdopen (OpusFileCallbacks ∗_cb, int _fd, const char ∗_mode) OP_ARG_NONNULL(1) OP_ARG_NONNULL(3)

  Opens a stream with `fdopen()` and fills in a set of callbacks that can be used to access it.
- OP_WARN_UNUSED_RESULT void ∗ op_freopen (OpusFileCallbacks ∗_cb, const char ∗_path, const char ∗_mode, void ∗_stream) OP_ARG_NONNULL(1) OP_ARG_NONNULL(2) OP_ARG_NONNULL(3) OP_↩ ARG_NONNULL(4)

  Opens a stream with `freopen()` and fills in a set of callbacks that can be used to access it.
- OP_WARN_UNUSED_RESULT void ∗ op_mem_stream_create (OpusFileCallbacks ∗_cb, const unsigned char ∗_data, size_t _size) OP_ARG_NONNULL(1)

  Creates a stream that reads from the given block of memory.
- OP_WARN_UNUSED_RESULT void ∗ op_url_stream_vcreate (OpusFileCallbacks ∗_cb, const char ∗_url, va_list _ap) OP_ARG_NONNULL(1) OP_ARG_NONNULL(2)

  Creates a stream that reads from the given URL.
- OP_WARN_UNUSED_RESULT void ∗ op_url_stream_create (OpusFileCallbacks ∗_cb, const char ∗_url,...) OP_ARG_NONNULL(1) OP_ARG_NONNULL(2)

  Creates a stream that reads from the given URL.

4.4.1 Detailed Description

4.4.2 Typedef Documentation

4.4.2.1 op_read_func

```
typedef int(∗ op_read_func) (void ∗_stream, unsigned char ∗_ptr, int _nbytes)
```

Reads up to *_nbytes* bytes of data from *_stream*.

Parameters

	_stream	The stream to read from.
out	*_ptr*	The buffer to store the data in.
	_nbytes	The maximum number of bytes to read. This function may return fewer, though it will not return zero unless it reaches end-of-file.

Returns

The number of bytes successfully read, or a negative value on error.

4.4.2.2 op_seek_func

```
typedef int(* op_seek_func) (void *_stream, opus_int64 _offset, int _whence)
```

Sets the position indicator for *_stream*.

The new position, measured in bytes, is obtained by adding *_offset* bytes to the position specified by *_whence*. If *_whence* is set to `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of the stream, the current position indicator, or end-of-file, respectively.

Return values

0	Success.
-1	Seeking is not supported or an error occurred. `errno` need not be set.

4.4.2.3 op_tell_func

```
typedef opus_int64(* op_tell_func) (void *_stream)
```

Obtains the current value of the position indicator for *_stream*.

Returns

The current position indicator.

4.4.2.4 op_close_func

```
typedef int(* op_close_func) (void *_stream)
```

Closes the underlying stream.

Return values

0	Success.
EOF	An error occurred. `errno` need not be set.

4.4.3 Function Documentation

4.4.3.1 op_fopen()

```
OP_WARN_UNUSED_RESULT void* op_fopen (
             OpusFileCallbacks * _cb,
             const char * _path,
             const char * _mode )
```

Opens a stream with `fopen()` and fills in a set of callbacks that can be used to access it.

This is useful to avoid writing your own portable 64-bit seeking wrappers, and also avoids cross-module linking issues on Windows, where a `FILE *` must be accessed by routines defined in the same module that opened it.

Parameters

`out`	*_cb*	The callbacks to use for this file. If there is an error opening the file, nothing will be filled in here.
	_path	The path to the file to open. On Windows, this string must be UTF-8 (to allow access to files whose names cannot be represented in the current MBCS code page). All other systems use the native character encoding.
	_mode	The mode to open the file in.

Returns

A stream handle to use with the callbacks, or `NULL` on error.

4.4.3.2 op_fdopen()

```
OP_WARN_UNUSED_RESULT void* op_fdopen (
             OpusFileCallbacks * _cb,
             int _fd,
             const char * _mode )
```

Opens a stream with `fdopen()` and fills in a set of callbacks that can be used to access it.

This is useful to avoid writing your own portable 64-bit seeking wrappers, and also avoids cross-module linking issues on Windows, where a `FILE *` must be accessed by routines defined in the same module that opened it.

Parameters

out	*_cb*	The callbacks to use for this file. If there is an error opening the file, nothing will be filled in here.
	_fd	The file descriptor to open.
	_mode	The mode to open the file in.

Returns

A stream handle to use with the callbacks, or `NULL` on error.

4.4.3.3 op_freopen()

```
OP_WARN_UNUSED_RESULT void* op_freopen (
            OpusFileCallbacks * _cb,
            const char * _path,
            const char * _mode,
            void * _stream )
```

Opens a stream with `freopen()` and fills in a set of callbacks that can be used to access it.

This is useful to avoid writing your own portable 64-bit seeking wrappers, and also avoids cross-module linking issues on Windows, where a `FILE *` must be accessed by routines defined in the same module that opened it.

Parameters

out	*_cb*	The callbacks to use for this file. If there is an error opening the file, nothing will be filled in here.
	_path	The path to the file to open. On Windows, this string must be UTF-8 (to allow access to files whose names cannot be represented in the current MBCS code page). All other systems use the native character encoding.
	_mode	The mode to open the file in.
	_stream	A stream previously returned by op_fopen(), op_fdopen(), or op_freopen().

Returns

A stream handle to use with the callbacks, or `NULL` on error.

4.4.3.4 op_mem_stream_create()

```
OP_WARN_UNUSED_RESULT void* op_mem_stream_create (
            OpusFileCallbacks * _cb,
            const unsigned char * _data,
            size_t _size )
```

Creates a stream that reads from the given block of memory.

This block of memory must contain the complete stream to decode. This is useful for caching small streams (e.g., sound effects) in RAM.

Parameters

`out`	*_cb*	The callbacks to use for this stream. If there is an error creating the stream, nothing will be filled in here.
	_data	The block of memory to read from.
	_size	The size of the block of memory.

Returns

A stream handle to use with the callbacks, or `NULL` on error.

4.4.3.5 op_url_stream_vcreate()

```
OP_WARN_UNUSED_RESULT void* op_url_stream_vcreate (
            OpusFileCallbacks * _cb,
            const char * _url,
            va_list _ap )
```

Creates a stream that reads from the given URL.

This function behaves identically to op_url_stream_create(), except that it takes a va_list instead of a variable number of arguments. It does not call the `va_end` macro, and because it invokes the `va_arg` macro, the value of *_ap* is undefined after the call.

Note

If you use this function, you must link against `libopusurl`.

Parameters

`out`	*_cb*	The callbacks to use for this stream. If there is an error creating the stream, nothing will be filled in here.
	_url	The URL to read from. Currently only the <file:>, <http:>, and <https:> schemes are supported. Both <http:> and <https:> may be disabled at compile time, in which case opening such URLs will always fail. Currently this only supports URIs. IRIs should be converted to UTF-8 and URL-escaped, with internationalized domain names encoded in punycode, before passing them to this function.
`in,out`	*_ap*	A list of the optional flags to use. This is a variable-length list of options terminated with `NULL`.

Returns

A stream handle to use with the callbacks, or `NULL` on error.

4.4.3.6 op_url_stream_create()

```
OP_WARN_UNUSED_RESULT void* op_url_stream_create (
            OpusFileCallbacks * _cb,
            const char * _url,
             ... )
```

Creates a stream that reads from the given URL.

Note

If you use this function, you must link against `libopusurl`.

Parameters

out	*_cb*	The callbacks to use for this stream. If there is an error creating the stream, nothing will be filled in here.
	_url	The URL to read from. Currently only the <file:>, <http:>, and <https:> schemes are supported. Both <http:> and <https:> may be disabled at compile time, in which case opening such URLs will always fail. Currently this only supports URIs. IRIs should be converted to UTF-8 and URL-escaped, with internationalized domain names encoded in punycode, before passing them to this function.
	...	The optional flags to use. This is a variable-length list of options terminated with `NULL`.

Returns

A stream handle to use with the callbacks, or `NULL` on error.

4.5 Opening and Closing

Functions for opening and closing streams

These functions allow you to test a stream to see if it is Opus, open it, and close it.

Several flavors are provided for each of the built-in stream types, plus a more general version which takes a set of application-provided callbacks.

- int op_test (OpusHead ∗_head, const unsigned char ∗_initial_data, size_t _initial_bytes)

  Test to see if this is an Opus stream.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_open_file (const char ∗_path, int ∗_error) OP_ARG_↩ NONNULL(1)

  Open a stream from the given file path.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_open_memory (const unsigned char ∗_data, size_t _↩ size, int ∗_error)

  Open a stream from a memory buffer.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_vopen_url (const char ∗_url, int ∗_error, va_list _ap) OP_ARG_NONNULL(1)

  Open a stream from a URL.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_open_url (const char ∗_url, int ∗_error,...) OP_ARG_↩ NONNULL(1)

Open a stream from a URL.

- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_open_callbacks (void ∗_stream, const OpusFileCallbacks ∗_cb, const unsigned char ∗_initial_data, size_t _initial_bytes, int ∗_error) OP_ARG_NONNULL(2)

    Open a stream using the given set of callbacks to access it.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_test_file (const char ∗_path, int ∗_error) OP_ARG_←NONNULL(1)

    Partially open a stream from the given file path.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_test_memory (const unsigned char ∗_data, size_t _size, int ∗_error)

    Partially open a stream from a memory buffer.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_vtest_url (const char ∗_url, int ∗_error, va_list _ap) OP←_ARG_NONNULL(1)

    Partially open a stream from a URL.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_test_url (const char ∗_url, int ∗_error,...) OP_ARG_←NONNULL(1)

    Partially open a stream from a URL.
- OP_WARN_UNUSED_RESULT OggOpusFile ∗ op_test_callbacks (void ∗_stream, const OpusFileCallbacks ∗_cb, const unsigned char ∗_initial_data, size_t _initial_bytes, int ∗_error) OP_ARG_NONNULL(2)

    Partially open a stream using the given set of callbacks to access it.
- int op_test_open (OggOpusFile ∗_of) OP_ARG_NONNULL(1)

    Finish opening a stream partially opened with op_test_callbacks() or one of the associated convenience functions.
- void op_free (OggOpusFile ∗_of)

    Release all memory used by an `OggOpusFile`.

4.5.1 Detailed Description

4.5.2 Function Documentation

4.5.2.1 op_test()

```
int op_test (
            OpusHead * _head,
            const unsigned char * _initial_data,
            size_t _initial_bytes )
```

Test to see if this is an Opus stream.

For good results, you will need at least 57 bytes (for a pure Opus-only stream). Something like 512 bytes will give more reliable results for multiplexed streams. This function is meant to be a quick-rejection filter. Its purpose is not to guarantee that a stream is a valid Opus stream, but to ensure that it looks enough like Opus that it isn't going to be recognized as some other format (except possibly an Opus stream that is also multiplexed with other codecs, such as video).

Parameters

out	*_head*	The parsed ID header contents. You may pass `NULL` if you do not need this information. If the function fails, the contents of this structure remain untouched.
	_initial_data	An initial buffer of data from the start of the stream.
	_initial_bytes	The number of bytes in *_initial_data*.

Returns

0 if the data appears to be Opus, or a negative value on error.

Return values

OP_FALSE	There was not enough data to tell if this was an Opus stream or not.
OP_EFAULT	An internal memory allocation failed.
OP_EIMPL	The stream used a feature that is not implemented, such as an unsupported channel family.
OP_ENOTFORMAT	If the data did not contain a recognizable ID header for an Opus stream.
OP_EVERSION	If the version field signaled a version this library does not know how to parse.
OP_EBADHEADER	The ID header was not properly formatted or contained illegal values.

4.5.2.2 op_open_file()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_open_file (
             const char * _path,
             int * _error )
```

Open a stream from the given file path.

Parameters

	_path	The path to the file to open.
out	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. The failure code will be OP_EFAULT if the file could not be opened, or one of the other failure codes from op_open_callbacks() otherwise.

Returns

A freshly opened `OggOpusFile`, or `NULL` on error.

4.5.2.3 op_open_memory()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_open_memory (
             const unsigned char * _data,
             size_t _size,
             int * _error )
```

Open a stream from a memory buffer.

Parameters

	_data	The memory buffer to open.
	_size	The number of bytes in the buffer.
out	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.

Returns

A freshly opened `OggOpusFile`, or `NULL` on error.

4.5.2.4 op_vopen_url()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_vopen_url (
             const char * _url,
             int * _error,
             va_list _ap )
```

Open a stream from a URL.

This function behaves identically to op_open_url(), except that it takes a va_list instead of a variable number of arguments. It does not call the `va_end` macro, and because it invokes the `va_arg` macro, the value of *_ap* is undefined after the call.

Note

If you use this function, you must link against `libopusurl`.

Parameters

	_url	The URL to open. Currently only the <file:>, <http:>, and <https:> schemes are supported. Both <http:> and <https:> may be disabled at compile time, in which case opening such URLs will always fail. Currently this only supports URIs. IRIs should be converted to UTF-8 and URL-escaped, with internationalized domain names encoded in punycode, before passing them to this function.
`out`	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.
`in,out`	*_ap*	A list of the optional flags to use. This is a variable-length list of options terminated with `NULL`.

Returns

A freshly opened `OggOpusFile`, or `NULL` on error.

4.5.2.5 op_open_url()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_open_url (
             const char * _url,
             int * _error,
              ... )
```

Open a stream from a URL.

Note

If you use this function, you must link against `libopusurl`.

Parameters

	_url	The URL to open. Currently only the <file:>, <http:>, and <https:> schemes are supported. Both <http:> and <https:> may be disabled at compile time, in which case opening such URLs will always fail. Currently this only supports URIs. IRIs should be converted to UTF-8 and URL-escaped, with internationalized domain names encoded in punycode, before passing them to this function.
out	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.
	...	The optional flags to use. This is a variable-length list of options terminated with `NULL`.

Returns

A freshly opened `OggOpusFile`, or `NULL` on error.

4.5.2.6 op_open_callbacks()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_open_callbacks (
            void * _stream,
            const OpusFileCallbacks * _cb,
            const unsigned char * _initial_data,
            size_t _initial_bytes,
            int * _error )
```

Open a stream using the given set of callbacks to access it.

Parameters

	_stream	The stream to read from (e.g., a `FILE *`). This value will be passed verbatim as the first argument to all of the callbacks.
	_cb	The callbacks with which to access the stream. read() must be implemented. seek() and tell() may be `NULL`, or may always return -1 to indicate a stream is unseekable, but if seek() is implemented and succeeds on a particular stream, then tell() must also. close() may be `NULL`, but if it is not, it will be called when the `OggOpusFile` is destroyed by op_free(). It will not be called if op_open_callbacks() fails with an error.
	_initial_data	An initial buffer of data from the start of the stream. Applications can read some number of bytes from the start of the stream to help identify this as an Opus stream, and then provide them here to allow the stream to be opened, even if it is unseekable.
	_initial_bytes	The number of bytes in *_initial_data*. If the stream is seekable, its current position (as reported by tell() at the start of this function) must be equal to *_initial_bytes*. Otherwise, seeking to absolute positions will generate inconsistent results.

Parameters

out	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. The failure code will be one of **OP_EREAD** An underlying read, seek, or tell operation failed when it should have succeeded, or we failed to find data in the stream we had seen before. **OP_EFAULT** There was a memory allocation failure, or an internal library error. **OP_EIMPL** The stream used a feature that is not implemented, such as an unsupported channel family. **OP_EINVAL** seek() was implemented and succeeded on this source, but tell() did not, or the starting position indicator was not equal to *_initial_bytes*. **OP_ENOTFORMAT** The stream contained a link that did not have any logical Opus streams in it. **OP_EBADHEADER** A required header packet was not properly formatted, contained illegal values, or was missing altogether. **OP_EVERSION** An ID header contained an unrecognized version number. **OP_EBADLINK** We failed to find data we had seen before after seeking. **OP_EBADTIMESTAMP** The first or last timestamp in a link failed basic validity checks.

Returns

A freshly opened `OggOpusFile`, or `NULL` on error. `libopusfile` does *not* take ownership of the stream if the call fails. The calling application is responsible for closing the stream if this call returns an error.

4.5.2.7 op_test_file()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_test_file (
              const char * _path,
              int * _error )
```

Partially open a stream from the given file path.

See also

op_test_callbacks

Parameters

	_path	The path to the file to open.
out	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. The failure code will be OP_EFAULT if the file could not be opened, or one of the other failure codes from op_open_callbacks() otherwise.

Returns

A partially opened `OggOpusFile`, or `NULL` on error.

4.5.2.8 op_test_memory()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_test_memory (
             const unsigned char * _data,
             size_t _size,
             int * _error )
```

Partially open a stream from a memory buffer.

See also

op_test_callbacks

Parameters

	_data	The memory buffer to open.
	_size	The number of bytes in the buffer.
`out`	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.

Returns

A partially opened `OggOpusFile`, or `NULL` on error.

4.5.2.9 op_vtest_url()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_vtest_url (
             const char * _url,
             int * _error,
             va_list _ap )
```

Partially open a stream from a URL.

This function behaves identically to op_test_url(), except that it takes a va_list instead of a variable number of arguments. It does not call the `va_end` macro, and because it invokes the `va_arg` macro, the value of *_ap* is undefined after the call.

Note

If you use this function, you must link against `libopusurl`.

See also

op_test_url
op_test_callbacks

Parameters

	_url	The URL to open. Currently only the <file:>, <http:>, and <https:> schemes are supported. Both <http:> and <https:> may be disabled at compile time, in which case opening such URLs will always fail. Currently this only supports URIs. IRIs should be converted to UTF-8 and URL-escaped, with internationalized domain names encoded in punycode, before passing them to this function.
`out`	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.
`in,out`	*_ap*	A list of the optional flags to use. This is a variable-length list of options terminated with `NULL`.

Returns

A partially opened `OggOpusFile`, or `NULL` on error.

4.5.2.10 op_test_url()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_test_url (
            const char * _url,
            int * _error,
             ...  )
```

Partially open a stream from a URL.

Note

If you use this function, you must link against `libopusurl`.

See also

op_test_callbacks

Parameters

	_url	The URL to open. Currently only the <file:>, <http:>, and <https:> schemes are supported. Both <http:> and <https:> may be disabled at compile time, in which case opening such URLs will always fail. Currently this only supports URIs. IRIs should be converted to UTF-8 and URL-escaped, with internationalized domain names encoded in punycode, before passing them to this function.
`out`	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.
	...	The optional flags to use. This is a variable-length list of options terminated with `NULL`.

Returns

A partially opened `OggOpusFile`, or `NULL` on error.

4.5.2.11 op_test_callbacks()

```
OP_WARN_UNUSED_RESULT OggOpusFile* op_test_callbacks (
             void * _stream,
             const OpusFileCallbacks * _cb,
             const unsigned char * _initial_data,
             size_t _initial_bytes,
             int * _error )
```

Partially open a stream using the given set of callbacks to access it.

This tests for Opusness and loads the headers for the first link. It does not seek (although it tests for seekability). You can query a partially open stream for the few pieces of basic information returned by op_serialno(), op_channel_count(), op_head(), and op_tags() (but only for the first link). You may also determine if it is seekable via a call to op_seekable(). You cannot read audio from the stream, seek, get the size or duration, get information from links other than the first one, or even get the total number of links until you finish opening the stream with op_test_open(). If you do not need to do any of these things, you can dispose of it with op_free() instead.

This function is provided mostly to simplify porting existing code that used `libvorbisfile`. For new code, you are likely better off using op_test() instead, which is less resource-intensive, requires less data to succeed, and imposes a hard limit on the amount of data it examines (important for unseekable streams, where all such data must be buffered until you are sure of the stream type).

Parameters

	_stream	The stream to read from (e.g., a `FILE *`). This value will be passed verbatim as the first argument to all of the callbacks.
	_cb	The callbacks with which to access the stream. read() must be implemented. seek() and tell() may be `NULL`, or may always return -1 to indicate a stream is unseekable, but if seek() is implemented and succeeds on a particular stream, then tell() must also. close() may be `NULL`, but if it is not, it will be called when the `OggOpusFile` is destroyed by op_free(). It will not be called if op_open_callbacks() fails with an error.
	_initial_data	An initial buffer of data from the start of the stream. Applications can read some number of bytes from the start of the stream to help identify this as an Opus stream, and then provide them here to allow the stream to be tested more thoroughly, even if it is unseekable.
	_initial_bytes	The number of bytes in *_initial_data*. If the stream is seekable, its current position (as reported by tell() at the start of this function) must be equal to *_initial_bytes*. Otherwise, seeking to absolute positions will generate inconsistent results.
`out`	*_error*	Returns 0 on success, or a failure code on error. You may pass in `NULL` if you don't want the failure code. See op_open_callbacks() for a full list of failure codes.

Returns

A partially opened `OggOpusFile`, or `NULL` on error. `libopusfile` does *not* take ownership of the stream if the call fails. The calling application is responsible for closing the stream if this call returns an error.

4.5.2.12 op_test_open()

```
int op_test_open (
             OggOpusFile * _of )
```

Finish opening a stream partially opened with op_test_callbacks() or one of the associated convenience functions.

If this function fails, you are still responsible for freeing the `OggOpusFile` with op_free().

Parameters

↩ _↩ *of*	The `OggOpusFile` to finish opening.

Returns

0 on success, or a negative value on error.

Return values

OP_EREAD	An underlying read, seek, or tell operation failed when it should have succeeded.
OP_EFAULT	There was a memory allocation failure, or an internal library error.
OP_EIMPL	The stream used a feature that is not implemented, such as an unsupported channel family.
OP_EINVAL	The stream was not partially opened with op_test_callbacks() or one of the associated convenience functions.
OP_ENOTFORMAT	The stream contained a link that did not have any logical Opus streams in it.
OP_EBADHEADER	A required header packet was not properly formatted, contained illegal values, or was missing altogether.
OP_EVERSION	An ID header contained an unrecognized version number.
OP_EBADLINK	We failed to find data we had seen before after seeking.
OP_EBADTIMESTAMP	The first or last timestamp in a link failed basic validity checks.

4.5.2.13 op_free()

```
void op_free (
            OggOpusFile * _of )
```

Release all memory used by an `OggOpusFile`.

Parameters

↩ _↩ *of*	The `OggOpusFile` to free.

4.6 Stream Information

Functions for obtaining information about streams

These functions allow you to get basic information about a stream, including seekability, the number of links (for chained streams), plus the size, duration, bitrate, header parameters, and meta information for each link (or, where available, the stream as a whole).

Some of these (size, duration) are only available for seekable streams. You can also query the current stream position, link, and playback time, and instantaneous bitrate during playback.

Some of these functions may be used successfully on the partially open streams returned by op_test_callbacks() or one of the associated convenience functions. Their documention will indicate so explicitly.

- int op_seekable (const OggOpusFile ∗_of) OP_ARG_NONNULL(1)

  Returns whether or not the stream being read is seekable.
- int op_link_count (const OggOpusFile ∗_of) OP_ARG_NONNULL(1)

  Returns the number of links in this chained stream.
- opus_uint32 op_serialno (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Get the serial number of the given link in a (possibly-chained) Ogg Opus stream.
- int op_channel_count (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Get the channel count of the given link in a (possibly-chained) Ogg Opus stream.
- opus_int64 op_raw_total (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Get the total (compressed) size of the stream, or of an individual link in a (possibly-chained) Ogg Opus stream, including all headers and Ogg muxing overhead.
- ogg_int64_t op_pcm_total (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Get the total PCM length (number of samples at 48 kHz) of the stream, or of an individual link in a (possibly-chained) Ogg Opus stream.
- const OpusHead ∗ op_head (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Get the ID header information for the given link in a (possibly chained) Ogg Opus stream.
- const OpusTags ∗ op_tags (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Get the comment header information for the given link in a (possibly chained) Ogg Opus stream.
- int op_current_link (const OggOpusFile ∗_of) OP_ARG_NONNULL(1)

  Retrieve the index of the current link.
- opus_int32 op_bitrate (const OggOpusFile ∗_of, int _li) OP_ARG_NONNULL(1)

  Computes the bitrate of the stream, or of an individual link in a (possibly-chained) Ogg Opus stream.
- opus_int32 op_bitrate_instant (OggOpusFile ∗_of) OP_ARG_NONNULL(1)

  Compute the instantaneous bitrate, measured as the ratio of bits to playable samples decoded since a) the last call to op_bitrate_instant(), b) the last seek, or c) the start of playback, whichever was most recent.
- opus_int64 op_raw_tell (const OggOpusFile ∗_of) OP_ARG_NONNULL(1)

  Obtain the current value of the position indicator for _of.
- ogg_int64_t op_pcm_tell (const OggOpusFile ∗_of) OP_ARG_NONNULL(1)

  Obtain the PCM offset of the next sample to be read.

4.6.1 Detailed Description

4.6.2 Function Documentation

4.6.2.1 op_seekable()

```
int op_seekable (
            const OggOpusFile ∗ _of )
```

Returns whether or not the stream being read is seekable.

This is true if

1. The seek() and tell() callbacks are both non-NULL,
2. The seek() callback was successfully executed at least once, and
3. The tell() callback was successfully able to report the position indicator afterwards.

This function may be called on partially-opened streams.

Parameters

↩ _↩ *of*	The `OggOpusFile` whose seekable status is to be returned.

Returns

A non-zero value if seekable, and 0 if unseekable.

4.6.2.2 op_link_count()

```
int op_link_count (
            const OggOpusFile * _of )
```

Returns the number of links in this chained stream.

This function may be called on partially-opened streams, but it will always return 1. The actual number of links is not known until the stream is fully opened.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the link count.

Returns

For fully-open seekable streams, this returns the total number of links in the whole stream, which will be at least 1. For partially-open or unseekable streams, this always returns 1.

4.6.2.3 op_serialno()

```
opus_uint32 op_serialno (
            const OggOpusFile * _of,
            int _li )
```

Get the serial number of the given link in a (possibly-chained) Ogg Opus stream.

This function may be called on partially-opened streams, but it will always return the serial number of the Opus stream in the first link.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the serial number.
↩ _↩ *li*	The index of the link whose serial number should be retrieved. Use a negative number to get the serial number of the current link.

Returns

The serial number of the given link. If *_li* is greater than the total number of links, this returns the serial number of the last link. If the stream is not seekable, this always returns the serial number of the current link.

4.6.2.4 op_channel_count()

```
int op_channel_count (
            const OggOpusFile * _of,
            int _li )
```

Get the channel count of the given link in a (possibly-chained) Ogg Opus stream.

This is equivalent to `op_head(_of,_li)->channel_count`, but is provided for convenience. This function may be called on partially-opened streams, but it will always return the channel count of the Opus stream in the first link.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the channel count.
↩ _↩ *li*	The index of the link whose channel count should be retrieved. Use a negative number to get the channel count of the current link.

Returns

The channel count of the given link. If *_li* is greater than the total number of links, this returns the channel count of the last link. If the stream is not seekable, this always returns the channel count of the current link.

4.6.2.5 op_raw_total()

```
opus_int64 op_raw_total (
            const OggOpusFile * _of,
            int _li )
```

Get the total (compressed) size of the stream, or of an individual link in a (possibly-chained) Ogg Opus stream, including all headers and Ogg muxing overhead.

Warning

If the Opus stream (or link) is concurrently multiplexed with other logical streams (e.g., video), this returns the size of the entire stream (or link), not just the number of bytes in the first logical Opus stream. Returning the latter would require scanning the entire file.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the compressed size.
↩ _↩ *li*	The index of the link whose compressed size should be computed. Use a negative number to get the compressed size of the entire stream.

Returns

The compressed size of the entire stream if *_li* is negative, the compressed size of link *_li* if it is non-negative, or a negative value on error. The compressed size of the entire stream may be smaller than that of the underlying stream if trailing garbage was detected in the file.

Return values

OP_EINVAL	The stream is not seekable (so we can't know the length), *_li* wasn't less than the total number of links in the stream, or the stream was only partially open.

4.6.2.6 op_pcm_total()

```
ogg_int64_t op_pcm_total (
             const OggOpusFile * _of,
             int _li )
```

Get the total PCM length (number of samples at 48 kHz) of the stream, or of an individual link in a (possibly-chained) Ogg Opus stream.

Users looking for `op_time_total()` should use op_pcm_total() instead. Because timestamps in Opus are fixed at 48 kHz, there is no need for a separate function to convert this to seconds (and leaving it out avoids introducing floating point to the API, for those that wish to avoid it).

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the PCM offset.
↩ _↩ *li*	The index of the link whose PCM length should be computed. Use a negative number to get the PCM length of the entire stream.

Returns

The PCM length of the entire stream if *_li* is negative, the PCM length of link *_li* if it is non-negative, or a negative value on error.

Return values

OP_EINVAL	The stream is not seekable (so we can't know the length), *_li* wasn't less than the total number of links in the stream, or the stream was only partially open.

4.6.2.7 op_head()

```
const OpusHead* op_head (
            const OggOpusFile * _of,
            int _li )
```

Get the ID header information for the given link in a (possibly chained) Ogg Opus stream.

This function may be called on partially-opened streams, but it will always return the ID header information of the Opus stream in the first link.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the ID header information.
↩ _↩ *li*	The index of the link whose ID header information should be retrieved. Use a negative number to get the ID header information of the current link. For an unseekable stream, *_li* is ignored, and the ID header information for the current link is always returned, if available.

Returns

The contents of the ID header for the given link.

4.6.2.8 op_tags()

```
const OpusTags* op_tags (
            const OggOpusFile * _of,
            int _li )
```

Get the comment header information for the given link in a (possibly chained) Ogg Opus stream.

This function may be called on partially-opened streams, but it will always return the tags from the Opus stream in the first link.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the comment header information.
↩ _↩ *li*	The index of the link whose comment header information should be retrieved. Use a negative number to get the comment header information of the current link. For an unseekable stream, *_li* is ignored, and the comment header information for the current link is always returned, if available.

Returns

The contents of the comment header for the given link, or `NULL` if this is an unseekable stream that encountered an invalid link.

4.6.2.9 op_current_link()

```
int op_current_link (
             const OggOpusFile * _of )
```

Retrieve the index of the current link.

This is the link that produced the data most recently read by op_read_float() or its associated functions, or, after a seek, the link that the seek target landed in. Reading more data may advance the link index (even on the first read after a seek).

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the current link index.

Returns

The index of the current link on success, or a negative value on failure. For seekable streams, this is a number between 0 (inclusive) and the value returned by op_link_count() (exclusive). For unseekable streams, this value starts at 0 and increments by one each time a new link is encountered (even though op_link_count() always returns 1).

Return values

OP_EINVAL	The stream was only partially open.

4.6.2.10 op_bitrate()

```
opus_int32 op_bitrate (
             const OggOpusFile * _of,
             int _li )
```

Computes the bitrate of the stream, or of an individual link in a (possibly-chained) Ogg Opus stream.

The stream must be seekable to compute the bitrate. For unseekable streams, use op_bitrate_instant() to get periodic estimates.

Warning

If the Opus stream (or link) is concurrently multiplexed with other logical streams (e.g., video), this uses the size of the entire stream (or link) to compute the bitrate, not just the number of bytes in the first logical Opus stream. Returning the latter requires scanning the entire file, but this may be done by decoding the whole file and calling op_bitrate_instant() once at the end. Install a trivial decoding callback with op_set_decode_callback() if you wish to skip actual decoding during this process.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the bitrate.
↩ _↩ *li*	The index of the link whose bitrate should be computed. Use a negative number to get the bitrate of the whole stream.

Returns

The bitrate on success, or a negative value on error.

Return values

OP_EINVAL	The stream was only partially open, the stream was not seekable, or _*li* was larger than the number of links.

4.6.2.11 op_bitrate_instant()

```
opus_int32 op_bitrate_instant (
            OggOpusFile * _of )
```

Compute the instantaneous bitrate, measured as the ratio of bits to playable samples decoded since a) the last call to op_bitrate_instant(), b) the last seek, or c) the start of playback, whichever was most recent.

This will spike somewhat after a seek or at the start/end of a chain boundary, as pre-skip, pre-roll, and end-trimming causes samples to be decoded but not played.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the bitrate.

Returns

The bitrate, in bits per second, or a negative value on error.

Return values

OP_FALSE	No data has been decoded since any of the events described above.
OP_EINVAL	The stream was only partially open.

4.6.2.12 op_raw_tell()

```
opus_int64 op_raw_tell (
            const OggOpusFile * _of )
```

Obtain the current value of the position indicator for *_of*.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the position indicator.

Returns

The byte position that is currently being read from.

Return values

OP_EINVAL	The stream was only partially open.

4.6.2.13 op_pcm_tell()

```
ogg_int64_t op_pcm_tell (
            const OggOpusFile * _of )
```

Obtain the PCM offset of the next sample to be read.

If the stream is not properly timestamped, this might not increment by the proper amount between reads, or even return monotonically increasing values.

Parameters

↩ _↩ *of*	The `OggOpusFile` from which to retrieve the PCM offset.

Returns

The PCM offset of the next sample to be read.

Return values

OP_EINVAL	The stream was only partially open.

4.7 Seeking

Functions for seeking in Opus streams

These functions let you seek in Opus streams, if the underlying stream support it.

Seeking is implemented for all built-in stream I/O routines, though some individual streams may not be seekable (pipes, live HTTP streams, or HTTP streams from a server that does not support `Range` requests).

op_raw_seek() is the fastest: it is guaranteed to perform at most one physical seek, but, since the target is a byte position, makes no guarantee how close to a given time it will come. op_pcm_seek() provides sample-accurate seeking. The number of physical seeks it requires is still quite small (often 1 or 2, even in highly variable bitrate streams).

Seeking in Opus requires decoding some pre-roll amount before playback to allow the internal state to converge (as if recovering from packet loss). This is handled internally by `libopusfile`, but means there is little extra overhead for decoding up to the exact position requested (since it must decode some amount of audio anyway). It also means that decoding after seeking may not return exactly the same values as would be obtained by decoding the stream straight through. However, such differences are expected to be smaller than the loss introduced by Opus's lossy compression.

- int op_raw_seek (OggOpusFile ∗_of, opus_int64 _byte_offset) OP_ARG_NONNULL(1)

  Seek to a byte offset relative to the ***compressed*** *data.*
- int op_pcm_seek (OggOpusFile ∗_of, ogg_int64_t _pcm_offset) OP_ARG_NONNULL(1)

  Seek to the specified PCM offset, such that decoding will begin at exactly the requested position.

4.7.1 Detailed Description

4.7.2 Function Documentation

4.7.2.1 op_raw_seek()

```
int op_raw_seek (
            OggOpusFile ∗ _of,
            opus_int64 _byte_offset )
```

Seek to a byte offset relative to the **compressed** data.

This also scans packets to update the PCM cursor. It will cross a logical bitstream boundary, but only if it can't get any packets out of the tail of the link to which it seeks.

Parameters

_of	The `OggOpusFile` in which to seek.
_byte_offset	The byte position to seek to. This must be between 0 and op_raw_total(*_of*,`-1`) (inclusive).

Returns

0 on success, or a negative error code on failure.

Return values

OP_EREAD	The underlying seek operation failed.
OP_EINVAL	The stream was only partially open, or the target was outside the valid range for the stream.
OP_ENOSEEK	This stream is not seekable.
OP_EBADLINK	Failed to initialize a decoder for a stream for an unknown reason.

4.7.2.2 op_pcm_seek()

```
int op_pcm_seek (
            OggOpusFile * _of,
            ogg_int64_t _pcm_offset )
```

Seek to the specified PCM offset, such that decoding will begin at exactly the requested position.

Parameters

_of	The `OggOpusFile` in which to seek.
_pcm_offset	The PCM offset to seek to. This is in samples at 48 kHz relative to the start of the stream.

Returns

0 on success, or a negative value on error.

Return values

OP_EREAD	An underlying read or seek operation failed.
OP_EINVAL	The stream was only partially open, or the target was outside the valid range for the stream.
OP_ENOSEEK	This stream is not seekable.
OP_EBADLINK	We failed to find data we had seen before, or the bitstream structure was sufficiently malformed that seeking to the target destination was impossible.

4.8 Decoding

Functions for decoding audio data

These functions retrieve actual decoded audio data from the stream.

The general functions, op_read() and op_read_float() return 16-bit or floating-point output, both using native endian ordering. The number of channels returned can change from link to link in a chained stream. There are special functions, op_read_stereo() and op_read_float_stereo(), which always output two channels, to simplify applications which do not wish to handle multichannel audio. These downmix multichannel files to two channels, so they can always return samples in the same format for every link in a chained file.

If the rest of your audio processing chain can handle floating point, the floating-point routines should be preferred, as they prevent clipping and other issues which might be avoided entirely if, e.g., you scale down the volume at some other stage. However, if you intend to consume 16-bit samples directly, the conversion in `libopusfile` provides noise-shaping dithering and, if compiled against `libopus` 1.1 or later, soft-clipping prevention.

`libopusfile` can also be configured at compile time to use the fixed-point `libopus` API. If so, `libopusfile`'s floating-point API may also be disabled. In that configuration, nothing in `libopusfile` will use any floating-point operations, to simplify support on devices without an adequate FPU.

Warning

HTTPS streams may be be vulnerable to truncation attacks if you do not check the error return code from op_read_float() or its associated functions. If the remote peer does not close the connection gracefully (with a TLS "close notify" message), these functions will return OP_EREAD instead of 0 when they reach the end of the file. If you are reading from an <https:> URL (particularly if seeking is not supported), you should make sure to check for this error and warn the user appropriately.

- typedef int(∗ op_decode_cb_func) (void ∗_ctx, OpusMSDecoder ∗_decoder, void ∗_pcm, const ogg_packet ∗_op, int _nsamples, int _nchannels, int _format, int _li)

  Called to decode an Opus packet.
- void op_set_decode_callback (OggOpusFile ∗_of, op_decode_cb_func _decode_cb, void ∗_ctx) OP_ARG↩ _NONNULL(1)

  Sets the packet decode callback function.
- int op_set_gain_offset (OggOpusFile ∗_of, int _gain_type, opus_int32 _gain_offset_q8) OP_ARG_↩ NONNULL(1)

  Sets the gain to be used for decoded output.
- void op_set_dither_enabled (OggOpusFile ∗_of, int _enabled) OP_ARG_NONNULL(1)

  Sets whether or not dithering is enabled for 16-bit decoding.
- OP_WARN_UNUSED_RESULT int op_read (OggOpusFile ∗_of, opus_int16 ∗_pcm, int _buf_size, int ∗_li) OP_ARG_NONNULL(1)

  Reads more samples from the stream.
- OP_WARN_UNUSED_RESULT int op_read_float (OggOpusFile ∗_of, float ∗_pcm, int _buf_size, int ∗_li) OP_ARG_NONNULL(1)

  Reads more samples from the stream.
- OP_WARN_UNUSED_RESULT int op_read_stereo (OggOpusFile ∗_of, opus_int16 ∗_pcm, int _buf_size) OP_ARG_NONNULL(1)

  Reads more samples from the stream and downmixes to stereo, if necessary.
- OP_WARN_UNUSED_RESULT int op_read_float_stereo (OggOpusFile ∗_of, float ∗_pcm, int _buf_size) OP_ARG_NONNULL(1)

  Reads more samples from the stream and downmixes to stereo, if necessary.
- #define OP_DEC_FORMAT_SHORT (7008)

Indicates that the decoding callback should produce signed 16-bit native-endian output samples.

- #define OP_DEC_FORMAT_FLOAT (7040)

  Indicates that the decoding callback should produce 32-bit native-endian float samples.
- #define OP_DEC_USE_DEFAULT (6720)

  Indicates that the decoding callback did not decode anything, and that `libopusfile` should decode normally instead.
- #define OP_HEADER_GAIN (0)

  Gain offset type that indicates that the provided offset is relative to the header gain.
- #define OP_ALBUM_GAIN (3007)

  Gain offset type that indicates that the provided offset is relative to the R128_ALBUM_GAIN value (if any), in addition to the header gain.
- #define OP_TRACK_GAIN (3008)

  Gain offset type that indicates that the provided offset is relative to the R128_TRACK_GAIN value (if any), in addition to the header gain.
- #define OP_ABSOLUTE_GAIN (3009)

  Gain offset type that indicates that the provided offset should be used as the gain directly, without applying any the header or track gains.

4.8.1 Detailed Description

4.8.2 Macro Definition Documentation

4.8.2.1 OP_HEADER_GAIN

```
#define OP_HEADER_GAIN (0)
```

Gain offset type that indicates that the provided offset is relative to the header gain.

This is the default.

4.8.3 Typedef Documentation

4.8.3.1 op_decode_cb_func

```
typedef int(* op_decode_cb_func) (void *_ctx, OpusMSDecoder *_decoder, void *_pcm, const ogg_↩
packet *_op, int _nsamples, int _nchannels, int _format, int _li)
```

Called to decode an Opus packet.

This should invoke the functional equivalent of opus_multistream_decode() or opus_multistream_decode_float(), except that it returns 0 on success instead of the number of decoded samples (which is known a priori).

Parameters

	_ctx	The application-provided callback context.
	_decoder	The decoder to use to decode the packet.
out	*_pcm*	The buffer to decode into. This will always have enough room for *_nchannels* of *_nsamples* samples, which should be placed into this buffer interleaved.
	_op	The packet to decode. This will always have its granule position set to a valid value.
	_nsamples	The number of samples expected from the packet.
	_nchannels	The number of channels expected from the packet.
	_format	The desired sample output format. This is either OP_DEC_FORMAT_SHORT or OP_DEC_FORMAT_FLOAT.
	_li	The index of the link from which this packet was decoded.

Returns

A non-negative value on success, or a negative value on error. Any error codes should be the same as those returned by opus_multistream_decode() or opus_multistream_decode_float(). Success codes are as follows:

Return values

0	Decoding was successful. The application has filled the buffer with exactly `_nsamples*_nchannels samples` in the requested format.
OP_DEC_USE_DEFAULT	No decoding was done. `libopusfile` should do the decoding by itself instead.

4.8.4 Function Documentation

4.8.4.1 op_set_decode_callback()

```
void op_set_decode_callback (
            OggOpusFile * _of,
            op_decode_cb_func _decode_cb,
            void * _ctx )
```

Sets the packet decode callback function.

If set, this is called once for each packet that needs to be decoded. This can be used by advanced applications to do additional processing on the compressed or uncompressed data. For example, an application might save the final entropy coder state for debugging and testing purposes, or it might apply additional filters before the downmixing, dithering, or soft-clipping performed by `libopusfile`, so long as these filters do not introduce any latency.

A call to this function is no guarantee that the audio will eventually be delivered to the application. `libopusfile` may discard some or all of the decoded audio data (i.e., at the beginning or end of a link, or after a seek), however the callback is still required to provide all of it.

Parameters

_of	The `OggOpusFile` on which to set the decode callback.
_decode_cb	The callback function to call. This may be `NULL` to disable calling the callback.
_ctx	The application-provided context pointer to pass to the callback on each call.

4.8.4.2 op_set_gain_offset()

```
int op_set_gain_offset (
            OggOpusFile * _of,
            int _gain_type,
            opus_int32 _gain_offset_q8 )
```

Sets the gain to be used for decoded output.

By default, the gain in the header is applied with no additional offset. The total gain (including header gain and/or track gain, if applicable, and this offset), will be clamped to [-32768,32767]/256 dB. This is more than enough to saturate or underflow 16-bit PCM.

Note

The new gain will not be applied to any already buffered, decoded output. This means you cannot change it sample-by-sample, as at best it will be updated packet-by-packet. It is meant for setting a target volume level, rather than applying smooth fades, etc.

Parameters

_of	The `OggOpusFile` on which to set the gain offset.
_gain_type	One of OP_HEADER_GAIN, OP_ALBUM_GAIN, OP_TRACK_GAIN, or OP_ABSOLUTE_GAIN.
_gain_offset_q8	The gain offset to apply, in 1/256ths of a dB.

Returns

0 on success or a negative value on error.

Return values

OP_EINVAL	The *_gain_type* was unrecognized.

4.8.4.3 op_set_dither_enabled()

```
void op_set_dither_enabled (
            OggOpusFile * _of,
            int _enabled )
```

Sets whether or not dithering is enabled for 16-bit decoding.

By default, when `libopusfile` is compiled to use floating-point internally, calling op_read() or op_read_stereo() will first decode to float, and then convert to fixed-point using noise-shaping dithering. This flag can be used to disable that dithering. When the application uses op_read_float() or op_read_float_stereo(), or when the library has been compiled to decode directly to fixed point, this flag has no effect.

Parameters

_of	The `OggOpusFile` on which to enable or disable dithering.
_enabled	A non-zero value to enable dithering, or 0 to disable it.

4.8.4.4 op_read()

```
OP_WARN_UNUSED_RESULT int op_read (
            OggOpusFile * _of,
            opus_int16 * _pcm,
            int _buf_size,
            int * _li )
```

Reads more samples from the stream.

Note

Although *_buf_size* must indicate the total number of values that can be stored in *_pcm*, the return value is the number of samples *per channel*. This is done because

1. The channel count cannot be known a priori (reading more samples might advance us into the next link, with a different channel count), so *_buf_size* cannot also be in units of samples per channel,
2. Returning the samples per channel matches the `libopus` API as closely as we're able,
3. Returning the total number of values instead of samples per channel would mean the caller would need a division to compute the samples per channel, and might worry about the possibility of getting back samples for some channels and not others, and
4. This approach is relatively fool-proof: if an application passes too small a value to *_buf_size*, they will simply get fewer samples back, and if they assume the return value is the total number of values, then they will simply read too few (rather than reading too many and going off the end of the buffer).

Parameters

	_of	The `OggOpusFile` from which to read.
out	*_pcm*	A buffer in which to store the output PCM samples, as signed native-endian 16-bit values at 48 kHz with a nominal range of `[-32768,32767)`. Multiple channels are interleaved using the `Vorbis channel ordering`. This must have room for at least *_buf_size* values.
	_buf_size	The number of values that can be stored in *_pcm*. It is recommended that this be large enough for at least 120 ms of data at 48 kHz per channel (5760 values per channel). Smaller buffers will simply return less data, possibly consuming more memory to buffer the data internally. `libopusfile` may return less data than requested. If so, there is no guarantee that the remaining data in *_pcm* will be unmodified.
out	*_li*	The index of the link this data was decoded from. You may pass `NULL` if you do not need this information. If this function fails (returning a negative value), this parameter is left unset.

Returns

The number of samples read per channel on success, or a negative value on failure. The channel count can be retrieved on success by calling `op_head(_of,*_li)`. The number of samples returned may be 0 if the buffer was too small to store even a single sample for all channels, or if end-of-file was reached. The list of possible failure codes follows. Most of them can only be returned by unseekable, chained streams that encounter a new link.

Return values

OP_HOLE	There was a hole in the data, and some samples may have been skipped. Call this function again to continue decoding past the hole.
OP_EREAD	An underlying read operation failed. This may signal a truncation attack from an <https:> source.
OP_EFAULT	An internal memory allocation failed.
OP_EIMPL	An unseekable stream encountered a new link that used a feature that is not implemented, such as an unsupported channel family.
OP_EINVAL	The stream was only partially open.
OP_ENOTFORMAT	An unseekable stream encountered a new link that did not have any logical Opus streams in it.
OP_EBADHEADER	An unseekable stream encountered a new link with a required header packet that was not properly formatted, contained illegal values, or was missing altogether.
OP_EVERSION	An unseekable stream encountered a new link with an ID header that contained an unrecognized version number.
OP_EBADPACKET	Failed to properly decode the next packet.
OP_EBADLINK	We failed to find data we had seen before.
OP_EBADTIMESTAMP	An unseekable stream encountered a new link with a starting timestamp that failed basic validity checks.

4.8.4.5 op_read_float()

```
OP_WARN_UNUSED_RESULT int op_read_float (
            OggOpusFile * _of,
            float * _pcm,
            int _buf_size,
            int * _li )
```

Reads more samples from the stream.

Note

Although *_buf_size* must indicate the total number of values that can be stored in *_pcm*, the return value is the number of samples *per channel*.

1. The channel count cannot be known a priori (reading more samples might advance us into the next link, with a different channel count), so *_buf_size* cannot also be in units of samples per channel,
2. Returning the samples per channel matches the `libopus` API as closely as we're able,
3. Returning the total number of values instead of samples per channel would mean the caller would need a division to compute the samples per channel, and might worry about the possibility of getting back samples for some channels and not others, and
4. This approach is relatively fool-proof: if an application passes too small a value to *_buf_size*, they will simply get fewer samples back, and if they assume the return value is the total number of values, then they will simply read too few (rather than reading too many and going off the end of the buffer).

Parameters

	_of	The `OggOpusFile` from which to read.
`out`	*_pcm*	A buffer in which to store the output PCM samples as signed floats at 48 kHz with a nominal range of `[-1.0,1.0]`. Multiple channels are interleaved using the `Vorbis channel ordering`. This must have room for at least *_buf_size* floats.
	_buf_size	The number of floats that can be stored in *_pcm*. It is recommended that this be large enough for at least 120 ms of data at 48 kHz per channel (5760 samples per channel). Smaller buffers will simply return less data, possibly consuming more memory to buffer the data internally. If less than *_buf_size* values are returned, `libopusfile` makes no guarantee that the remaining data in *_pcm* will be unmodified.
`out`	*_li*	The index of the link this data was decoded from. You may pass `NULL` if you do not need this information. If this function fails (returning a negative value), this parameter is left unset.

Returns

The number of samples read per channel on success, or a negative value on failure. The channel count can be retrieved on success by calling `op_head(_of,*_li)`. The number of samples returned may be 0 if the buffer was too small to store even a single sample for all channels, or if end-of-file was reached. The list of possible failure codes follows. Most of them can only be returned by unseekable, chained streams that encounter a new link.

Return values

OP_HOLE	There was a hole in the data, and some samples may have been skipped. Call this function again to continue decoding past the hole.
OP_EREAD	An underlying read operation failed. This may signal a truncation attack from an <https:> source.
OP_EFAULT	An internal memory allocation failed.
OP_EIMPL	An unseekable stream encountered a new link that used a feature that is not implemented, such as an unsupported channel family.
OP_EINVAL	The stream was only partially open.
OP_ENOTFORMAT	An unseekable stream encountered a new link that did not have any logical Opus streams in it.
OP_EBADHEADER	An unseekable stream encountered a new link with a required header packet that was not properly formatted, contained illegal values, or was missing altogether.
OP_EVERSION	An unseekable stream encountered a new link with an ID header that contained an unrecognized version number.
OP_EBADPACKET	Failed to properly decode the next packet.
OP_EBADLINK	We failed to find data we had seen before.
OP_EBADTIMESTAMP	An unseekable stream encountered a new link with a starting timestamp that failed basic validity checks.

4.8.4.6 op_read_stereo()

```
OP_WARN_UNUSED_RESULT int op_read_stereo (
            OggOpusFile * _of,
            opus_int16 * _pcm,
            int _buf_size )
```

Reads more samples from the stream and downmixes to stereo, if necessary.

This function is intended for simple players that want a uniform output format, even if the channel count changes between links in a chained stream.

Note

_buf_size indicates the total number of values that can be stored in *_pcm*, while the return value is the number of samples *per channel*, even though the channel count is known, for consistency with op_read().

Parameters

	_of	The `OggOpusFile` from which to read.
`out`	*_pcm*	A buffer in which to store the output PCM samples, as signed native-endian 16-bit values at 48 kHz with a nominal range of `[-32768,32767)`. The left and right channels are interleaved in the buffer. This must have room for at least *_buf_size* values.
	_buf_size	The number of values that can be stored in *_pcm*. It is recommended that this be large enough for at least 120 ms of data at 48 kHz per channel (11520 values total). Smaller buffers will simply return less data, possibly consuming more memory to buffer the data internally. If less than *_buf_size* values are returned, `libopusfile` makes no guarantee that the remaining data in *_pcm* will be unmodified.

Returns

The number of samples read per channel on success, or a negative value on failure. The number of samples returned may be 0 if the buffer was too small to store even a single sample for both channels, or if end-of-file was reached. The list of possible failure codes follows. Most of them can only be returned by unseekable, chained streams that encounter a new link.

Return values

OP_HOLE	There was a hole in the data, and some samples may have been skipped. Call this function again to continue decoding past the hole.
OP_EREAD	An underlying read operation failed. This may signal a truncation attack from an <https:> source.
OP_EFAULT	An internal memory allocation failed.
OP_EIMPL	An unseekable stream encountered a new link that used a feature that is not implemented, such as an unsupported channel family.
OP_EINVAL	The stream was only partially open.
OP_ENOTFORMAT	An unseekable stream encountered a new link that did not have any logical Opus streams in it.
OP_EBADHEADER	An unseekable stream encountered a new link with a required header packet that was not properly formatted, contained illegal values, or was missing altogether.
OP_EVERSION	An unseekable stream encountered a new link with an ID header that contained an unrecognized version number.
OP_EBADPACKET	Failed to properly decode the next packet.
OP_EBADLINK	We failed to find data we had seen before.
OP_EBADTIMESTAMP	An unseekable stream encountered a new link with a starting timestamp that failed basic validity checks.

4.8.4.7 op_read_float_stereo()

```
OP_WARN_UNUSED_RESULT int op_read_float_stereo (
            OggOpusFile * _of,
            float * _pcm,
            int _buf_size )
```

Reads more samples from the stream and downmixes to stereo, if necessary.

This function is intended for simple players that want a uniform output format, even if the channel count changes between links in a chained stream.

Note

_buf_size indicates the total number of values that can be stored in *_pcm*, while the return value is the number of samples *per channel*, even though the channel count is known, for consistency with op_read_float().

Parameters

	_of	The `OggOpusFile` from which to read.
out	*_pcm*	A buffer in which to store the output PCM samples, as signed floats at 48 kHz with a nominal range of `[-1.0,1.0]`. The left and right channels are interleaved in the buffer. This must have room for at least *_buf_size* values.
	_buf_size	The number of values that can be stored in *_pcm*. It is recommended that this be large enough for at least 120 ms of data at 48 kHz per channel (11520 values total). Smaller buffers will simply return less data, possibly consuming more memory to buffer the data internally. If less than *_buf_size* values are returned, `libopusfile` makes no guarantee that the remaining data in *_pcm* will be unmodified.

Returns

The number of samples read per channel on success, or a negative value on failure. The number of samples returned may be 0 if the buffer was too small to store even a single sample for both channels, or if end-of-file was reached. The list of possible failure codes follows. Most of them can only be returned by unseekable, chained streams that encounter a new link.

Return values

OP_HOLE	There was a hole in the data, and some samples may have been skipped. Call this function again to continue decoding past the hole.
OP_EREAD	An underlying read operation failed. This may signal a truncation attack from an <https:> source.
OP_EFAULT	An internal memory allocation failed.
OP_EIMPL	An unseekable stream encountered a new link that used a feature that is not implemented, such as an unsupported channel family.
OP_EINVAL	The stream was only partially open.
OP_ENOTFORMAT	An unseekable stream encountered a new link that that did not have any logical Opus streams in it.
OP_EBADHEADER	An unseekable stream encountered a new link with a required header packet that was not properly formatted, contained illegal values, or was missing altogether.

Return values

OP_EVERSION	An unseekable stream encountered a new link with an ID header that contained an unrecognized version number.
OP_EBADPACKET	Failed to properly decode the next packet.
OP_EBADLINK	We failed to find data we had seen before.
OP_EBADTIMESTAMP	An unseekable stream encountered a new link with a starting timestamp that failed basic validity checks.

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