opusfile

0.2

Generated by Doxygen 1.7.6.1

Tue Apr 30 2013 08:56:10

Contents

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:> UR-Ls).

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

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)
- #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)
- #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 Define Documentation

4.1.1.1 #define OP_FALSE (-1)

A request did not succeed.

4.1.1.2 #define OP_HOLE (-3)

There was a hole in the page sequence numbers (e.g., a page was corrupt or missing).

4.1.1.3 #define OP_EREAD (-128)

An underlying read, seek, or tell operation failed when it should have succeeded.

4.1.1.4 #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.

4.1.1.5 #define OP_EIMPL (-130)

The stream used a feature that is not implemented, such as an unsupported channel family.

4.1.1.6 #define OP_EINVAL (-131)

One or more parameters to a function were invalid.

4.1.1.7 #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.

4.1.1.8 #define OP_EBADHEADER (-133)

A required header packet was not properly formatted, contained illegal values, or was missing altogether.

4.1.1.9 #define OP_EVERSION (-134)

The ID header contained an unrecognized version number.

4.1.1.10 #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.

4.1.1.11 #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.

4.1.1.12 #define OP_ENOSEEK (-138)

An operation that requires seeking was requested on an unseekable stream.

4.1.1.13 #define OP_EBADTIMESTAMP (-139)

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

4.2 Header Information

Data Structures

- struct OpusHead

  Ogg Opus bitstream information.
- struct OpusTags

  The metadata from an Ogg Opus stream.

Defines

- #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.
- 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) O-P_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_A-RG_NONNULL(1) OP_ARG_NONNULL(2)

  Add a comment to an initialized OpusTags structure.
- 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_AR-G_NONNULL(1) OP_ARG_NONNULL(2)

Look up the number of instances of a tag.

- void opus_tags_clear (OpusTags ∗_tags) OP_ARG_NONNULL(1)

  Clears the OpusTags structure.

4.2.1 Define Documentation

4.2.1.1 #define **OPUS_CHANNEL_COUNT_MAX** (255)

The maximum number of channels in an Ogg Opus stream.

4.2.2 Function Documentation

4.2.2.1 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 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 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 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.

4.2.2.5 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.

4.2.2.6 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.

4.2.2.7 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.

4.2.2.8 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.

4.2.2.9 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.

4.3 URL Reading Options

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.

- #define **OP_SSL_SKIP_CERTIFICATE_CHECK_REQUEST** (6464)
- #define **OP_HTTP_PROXY_HOST_REQUEST** (6528)
- #define **OP_HTTP_PROXY_PORT_REQUEST** (6592)
- #define **OP_HTTP_PROXY_USER_REQUEST** (6656)
- #define **OP_HTTP_PROXY_PASS_REQUEST** (6720)
- #define **OP_URL_OPT**(_request) ((_request)+(char ∗)0)
- #define **OP_CHECK_INT**(_x) ((void)((_x)==(opus_int32)0),(opus_int32)(_x))
- #define **OP_CHECK_CONST_CHAR_PTR**(_x) ((_x)+((_x)-(const char ∗)(_x)))
- #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.

4.3.1 Define Documentation

4.3.1.1 #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.

4.3.1.2 #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.

4.3.1.3 #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.

4.3.1.4 #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.

4.3.1.5 #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.

4.4 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 struct OpusFileCallbacks **OpusFileCallbacks**
- 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 using the specified proxy.

4.4.1 Typedef Documentation

4.4.1.1 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.

4.4.1.2 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.

4.4.1.3 typedef opus_int64(∗ **op_tell_func**)(void ∗_stream)

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

Returns

The current position indicator.

4.4.1.4 typedef int(∗ **op_close_func**)(void ∗_stream)

Closes the underlying stream.

Return values

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

4.4.2 Function Documentation

4.4.2.1 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.
	_mode	The mode to open the file in.

Returns

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

4.4.2.2 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.

4.4.2.3 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.
	_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.

4.4.2.4 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.

4.4.2.5 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.

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.
`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.

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

Creates a stream that reads from the given URL using the specified proxy.

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.
	...	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 un-signed 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 ∗_-source, 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 un-signed 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 ∗_-source, 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 Function Documentation

4.5.1.1 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.

4.5.1.2 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.

4.5.1.3 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.

4.5.1.4 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.

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.
`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.

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

Open a stream from a URL.

However, this approach will not work for live streams or HTTP/1.0 servers (which do not support Range requets).

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.
`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.

4.5.1.6 OP_WARN_UNUSED_RESULT OggOpusFile∗ **op_open_callbacks** (void ∗ *_source,* 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

	_source	The stream to read from (e.g., a `FILE *`).
	_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 source is unseekable, but if `seek()` is implemented and succeeds on a particular source, 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.
`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.

4.5.1.7 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.

4.5.1.8 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.

4.5.1.9 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.

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.
`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.

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

Partially open a stream from a URL.

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.
`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.

4.5.1.11 OP_WARN_UNUSED_RESULT OggOpusFile∗ op_test_callbacks (void ∗ *_source*, 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 sources, where all such data must be buffered until you are sure of the stream type).

Parameters

	_source	The stream to read from (e.g., a `FILE *`).
	_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 source is unseekable, but if `seek()` is implemented and succeeds on a particular source, 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.

4.5.1.12 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.

4.5.1.13 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 (OggOpusFile ∗_of) OP_ARG_NONNULL(1)

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

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

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

  Computes the bitrate for a given 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 (OggOpusFile ∗_of) OP_ARG_NONNULL(1)

Obtain the current value of the position indicator for _of.

- ogg_int64_t op_pcm_tell (OggOpusFile ∗_of) OP_ARG_NONNULL(1)

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

4.6.1 Function Documentation

4.6.1.1 int op_seekable (OggOpusFile ∗ *_of*)

Returns whether or not the data source 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.

4.6.1.2 int op_link_count (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 sources, this returns the total number of links in the whole stream. For partially-open or unseekable sources, this always returns 1.

4.6.1.3 opus_uint32 op_serialno (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 source is not seekable, this always returns the serial number of the current link.

4.6.1.4 int **op_channel_count** (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 source is not seekable, this always returns the channel count of the current link.

4.6.1.5 opus_int64 **op_raw_total** (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.

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 source if trailing garbage was detected in the file.

Return values

OP_EINVAL	The source 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.

4.6.1.6 ogg_int64_t **op_pcm_total** (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 source 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.

4.6.1.7 const **OpusHead**∗ **op_head** (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.

4.6.1.8 const **OpusTags**∗ **op_tags** (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.

4.6.1.9 int **op_current_link** (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 and the value returned by op_link_count(). 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.

4.6.1.10 opus_int32 op_bitrate (OggOpusFile ∗ *_of,* int *_li*)

Computes the bitrate for a given 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.

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.

4.6.1.11 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.

4.6.1.12 opus_int64 op_raw_tell (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.

4.6.1.13 ogg_int64_t op_pcm_tell (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 source support it.

Seeking is implemented for all built-in stream I/O routines, though some individual sources 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 Function Documentation

4.7.1.1 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.

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.

4.7.1.2 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, those routines should be preferred, as floating point output avoids introducing 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 direct consume 16-bit samples, the conversion in `libopusfile` provides noise-shaping dithering API.

`libopusfile` can also be configured at compile time to use the fixed-point `libopus` API. If so, the 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.

- 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.

4.8.1 Function Documentation

4.8.1.1 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 prior (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 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 reccommended 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.

4.8.1.2 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 prior (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 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 reccommended 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.

4.8.1.3 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 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 reccommended 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.

4.8.1.4 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 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 reccommended 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.
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 op_read_func OpusFileCallbacks::read

Used to read data from the stream.

This must not be `NULL`.

5.1.2.2 op_seek_func OpusFileCallbacks::seek

Used to seek in the stream.

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

5.1.2.3 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 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:

- /home/derf/tmp/opusfile-0.2/include/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.

- 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 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 int **OpusHead::channel_count**

The number of channels, in the range 1...255.

5.2.2.3 unsigned **OpusHead::pre_skip**

The number of samples that should be discarded from the beginning of the stream.

5.2.2.4 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.

5.2.2.5 int OpusHead::output_gain

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

-32768...32767. The decoder will automatically scale the output by pow(10,output_-gain/(20.0∗256)).

5.2.2.6 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.

5.2.2.7 int OpusHead::stream_count

The number of Opus streams in each Ogg packet, in the range 1...255.

5.2.2.8 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.

5.2.2.9 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:

- /home/derf/tmp/opusfile-0.2/include/opusfile.h

5.3 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.

5.3.1 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 bitstream 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.

5.3.2 Field Documentation

5.3.2.1 char∗∗ OpusTags::user_comments

The array of comment string vectors.

5.3.2.2 int∗ OpusTags::comment_lengths

An array of the corresponding length of each vector, in bytes.

5.3.2.3 int OpusTags::comments

The total number of comment streams.

5.3.2.4 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:

- /home/derf/tmp/opusfile-0.2/include/opusfile.h