Opus

1.6

Generated on Mon Dec 15 2025 12:54:10 for Opus by Doxygen 1.9.8

Mon Dec 15 2025 12:54:10

Chapter 1

Opus

The Opus codec is designed for interactive speech and audio transmission over the Internet. It is designed by the IETF Codec Working Group and incorporates technology from Skype's SILK codec and Xiph.Org's CELT codec.

The Opus codec is designed to handle a wide range of interactive audio applications, including Voice over IP, videoconferencing, in-game chat, and even remote live music performances. It can scale from low bit-rate narrowband speech to very high quality stereo music. Its main features are:

- Sampling rates from 8 to 48 kHz
- Bit-rates from 6 kb/s to 510 kb/s
- Support for both constant bit-rate (CBR) and variable bit-rate (VBR)
- Audio bandwidth from narrowband to full-band
- Support for speech and music
- Support for mono and stereo
- Support for multichannel (up to 255 channels)
- Frame sizes from 2.5 ms to 60 ms
- Good loss robustness and packet loss concealment (PLC)
- Floating point and fixed-point implementation

Documentation sections:

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Topic Documentation

4.1 Opus Encoder

This page describes the process and functions used to encode Opus.

Typedefs

- typedef struct OpusEncoder OpusEncoder

  Opus encoder state.

Functions

- int opus_encoder_get_size (int channels)

  Gets the size of an `OpusEncoder` structure.
- OpusEncoder * opus_encoder_create (opus_int32 Fs, int channels, int application, int *error)

  Allocates and initializes an encoder state.
- int opus_encoder_init (OpusEncoder *st, opus_int32 Fs, int channels, int application)

  Initializes a previously allocated encoder state The memory pointed to by st must be at least the size returned by opus_encoder_get_size().
- opus_int32 opus_encode (OpusEncoder *st, const opus_int16 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes an Opus frame.
- opus_int32 opus_encode24 (OpusEncoder *st, const opus_int32 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes an Opus frame.
- opus_int32 opus_encode_float (OpusEncoder *st, const float *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes an Opus frame from floating point input.
- void opus_encoder_destroy (OpusEncoder *st)

  Frees an `OpusEncoder` allocated by opus_encoder_create().
- int opus_encoder_ctl (OpusEncoder *st, int request,...)

  Perform a CTL function on an Opus encoder.

4.1.1 Detailed Description

This page describes the process and functions used to encode Opus.

Since Opus is a stateful codec, the encoding process starts with creating an encoder state. This can be done with:

```
int          error;
OpusEncoder *enc;
enc = opus_encoder_create(Fs, channels, application, &error);
```

From this point, `enc` can be used for encoding an audio stream. An encoder state **must not** be used for more than one stream at the same time. Similarly, the encoder state **must not** be re-initialized for each frame.

While opus_encoder_create() allocates memory for the state, it's also possible to initialize pre-allocated memory:

```
int          size;
int          error;
OpusEncoder *enc;
size = opus_encoder_get_size(channels);
enc = malloc(size);
error = opus_encoder_init(enc, Fs, channels, application);
```

where opus_encoder_get_size() returns the required size for the encoder state. Note that future versions of this code may change the size, so no assumptions should be made about it.

The encoder state is always continuous in memory and only a shallow copy is sufficient to copy it (e.g. memcpy())

It is possible to change some of the encoder's settings using the opus_encoder_ctl() interface. All these settings already default to the recommended value, so they should only be changed when necessary. The most common settings one may want to change are:

```
opus_encoder_ctl(enc, OPUS_SET_BITRATE(bitrate));
opus_encoder_ctl(enc, OPUS_SET_COMPLEXITY(complexity));
opus_encoder_ctl(enc, OPUS_SET_SIGNAL(signal_type));
```

where

- bitrate is in bits per second (b/s)
- complexity is a value from 1 to 10, where 1 is the lowest complexity and 10 is the highest
- signal_type is either OPUS_AUTO (default), OPUS_SIGNAL_VOICE, or OPUS_SIGNAL_MUSIC

See Encoder related CTLs and Generic CTLs for a complete list of parameters that can be set or queried. Most parameters can be set or changed at any time during a stream.

To encode a frame, opus_encode() or opus_encode_float() must be called with exactly one frame (2.5, 5, 10, 20, 40 or 60 ms) of audio data:

```
len = opus_encode(enc, audio_frame, frame_size, packet, max_packet);
```

where

- audio_frame is the audio data in opus_int16 (or float for opus_encode_float())
- frame_size is the duration of the frame in samples (per channel)
- packet is the byte array to which the compressed data is written
- max_packet is the maximum number of bytes that can be written in the packet (4000 bytes is recommended). Do not use max_packet to control VBR target bitrate, instead use the OPUS_SET_BITRATE CTL.

opus_encode() and opus_encode_float() return the number of bytes actually written to the packet. The return value **can be negative**, which indicates that an error has occurred. If the return value is 2 bytes or less, then the packet does not need to be transmitted (DTX).

Once the encoder state if no longer needed, it can be destroyed with

```
opus_encoder_destroy(enc);
```

If the encoder was created with opus_encoder_init() rather than opus_encoder_create(), then no action is required aside from potentially freeing the memory that was manually allocated for it (calling free(enc) for the example above)

4.1.2 Typedef Documentation

4.1.2.1 OpusEncoder

```
typedef struct OpusEncoder OpusEncoder
```

Opus encoder state.

This contains the complete state of an Opus encoder. It is position independent and can be freely copied.

See also

opus_encoder_create,opus_encoder_init

4.1.3 Function Documentation

4.1.3.1 opus_encode()

```
opus_int32 opus_encode (
            OpusEncoder * st,
            const opus_int16 * pcm,
            int frame_size,
            unsigned char * data,
            opus_int32 max_data_bytes )
```

Encodes an Opus frame.

Parameters

in	*st*	OpusEncoder*: Encoder state
in	*pcm*	opus_int16*: Input signal (interleaved if 2 channels). length is frame_size*channels*sizeof(opus_int16)
in	*frame_size*	int: Number of samples per channel in the input signal. This must be an Opus frame size for the encoder's sampling rate. For example, at 48 kHz the permitted values are 120, 240, 480, 960, 1920, and 2880. Passing in a duration of less than 10 ms (480 samples at 48 kHz) will prevent the encoder from using the LPC or hybrid modes.
out	*data*	unsigned char*: Output payload. This must contain storage for at least *max_data_bytes*.
in	*max_data_bytes*	opus_int32: Size of the allocated memory for the output payload. This may be used to impose an upper limit on the instant bitrate, but should not be used as the only bitrate control. Use OPUS_SET_BITRATE to control the bitrate.

Returns

The length of the encoded packet (in bytes) on success or a negative error code (see Error codes) on failure.

4.1.3.2 opus_encode24()

```
opus_int32 opus_encode24 (
            OpusEncoder * st,
            const opus_int32 * pcm,
            int frame_size,
            unsigned char * data,
            opus_int32 max_data_bytes )
```

Encodes an Opus frame.

Parameters

in	*st*	OpusEncoder*: Encoder state
in	*pcm*	opus_int32*: Input signal (interleaved if 2 channels) representing (or slightly exceeding) 24-bit values. length is frame_size*channels*sizeof(opus_int32)
in	*frame_size*	int: Number of samples per channel in the input signal. This must be an Opus frame size for the encoder's sampling rate. For example, at 48 kHz the permitted values are 120, 240, 480, 960, 1920, and 2880. Passing in a duration of less than 10 ms (480 samples at 48 kHz) will prevent the encoder from using the LPC or hybrid modes.
out	*data*	unsigned char*: Output payload. This must contain storage for at least *max_data_bytes*.
in	*max_data_bytes*	opus_int32: Size of the allocated memory for the output payload. This may be used to impose an upper limit on the instant bitrate, but should not be used as the only bitrate control. Use OPUS_SET_BITRATE to control the bitrate.

Returns

The length of the encoded packet (in bytes) on success or a negative error code (see Error codes) on failure.

4.1.3.3 opus_encode_float()

```
opus_int32 opus_encode_float (
            OpusEncoder * st,
            const float * pcm,
            int frame_size,
            unsigned char * data,
            opus_int32 max_data_bytes )
```

Encodes an Opus frame from floating point input.

Parameters

in	*st*	OpusEncoder*: Encoder state
in	*pcm*	float*: Input in float format (interleaved if 2 channels), with a normal range of +/-1.0. Samples with a range beyond +/-1.0 are supported but will be clipped by decoders using the integer API and should only be used if it is known that the far end supports extended dynamic range. length is frame_size*channels*sizeof(float)

Parameters

in	*frame_size*	`int`: Number of samples per channel in the input signal. This must be an Opus frame size for the encoder's sampling rate. For example, at 48 kHz the permitted values are 120, 240, 480, 960, 1920, and 2880. Passing in a duration of less than 10 ms (480 samples at 48 kHz) will prevent the encoder from using the LPC or hybrid modes.
out	*data*	`unsigned char*`: Output payload. This must contain storage for at least *max_data_bytes*.
in	*max_data_bytes*	`opus_int32`: Size of the allocated memory for the output payload. This may be used to impose an upper limit on the instant bitrate, but should not be used as the only bitrate control. Use OPUS_SET_BITRATE to control the bitrate.

Returns

The length of the encoded packet (in bytes) on success or a negative error code (see Error codes) on failure.

4.1.3.4 opus_encoder_create()

```
OpusEncoder * opus_encoder_create (
             opus_int32 Fs,
             int channels,
             int application,
             int * error )
```

Allocates and initializes an encoder state.

There are three coding modes:

OPUS_APPLICATION_VOIP gives best quality at a given bitrate for voice signals. It enhances the input signal by high-pass filtering and emphasizing formants and harmonics. Optionally it includes in-band forward error correction to protect against packet loss. Use this mode for typical VoIP applications. Because of the enhancement, even at high bitrates the output may sound different from the input.

OPUS_APPLICATION_AUDIO gives best quality at a given bitrate for most non-voice signals like music. Use this mode for music and mixed (music/voice) content, broadcast, and applications requiring less than 15 ms of coding delay.

OPUS_APPLICATION_RESTRICTED_LOWDELAY configures low-delay mode that disables the speech-optimized mode in exchange for slightly reduced delay. This mode can only be set on an newly initialized or freshly reset encoder because it changes the codec delay.

This is useful when the caller knows that the speech-optimized modes will not be needed (use with caution).

Parameters

in	*Fs*	`opus_int32`: Sampling rate of input signal (Hz) This must be one of 8000, 12000, 16000, 24000, or 48000.
in	*channels*	`int`: Number of channels (1 or 2) in input signal
in	*application*	`int`: Coding mode (one of OPUS_APPLICATION_VOIP, OPUS_APPLICATION_AUDIO, or OPUS_APPLICATION_RESTRICTED_LOWDELAY)
out	*error*	`int*`: Error codes

Note

Regardless of the sampling rate and number channels selected, the Opus encoder can switch to a lower audio bandwidth or number of channels if the bitrate selected is too low. This also means that it is safe to always use 48 kHz stereo input and let the encoder optimize the encoding.

4.1.3.5 opus_encoder_ctl()

```
int opus_encoder_ctl (
            OpusEncoder * st,
            int request,
             ...  )
```

Perform a CTL function on an Opus encoder.

Generally the request and subsequent arguments are generated by a convenience macro.

Parameters

st	`OpusEncoder*`: Encoder state.
request	This and all remaining parameters should be replaced by one of the convenience macros in Generic CTLs or Encoder related CTLs.

See also

Generic CTLs

Encoder related CTLs

4.1.3.6 opus_encoder_destroy()

```
void opus_encoder_destroy (
            OpusEncoder * st )
```

Frees an `OpusEncoder` allocated by opus_encoder_create().

Parameters

in	*st*	`OpusEncoder*`: State to be freed.

4.1.3.7 opus_encoder_get_size()

```
int opus_encoder_get_size (
            int channels )
```

Gets the size of an `OpusEncoder` structure.

Parameters

in	*channels*	int: Number of channels. This must be 1 or 2.

Returns

The size in bytes.

Note

Since this function does not take the application as input, it will overestimate the size required for OPUS↩ _APPLICATION_RESTRICTED_SILK and OPUS_APPLICATION_RESTRICTED_CELT. That is generally not a problem, except when trying to know the size to use for a copy.

4.1.3.8 opus_encoder_init()

```
int opus_encoder_init (
            OpusEncoder * st,
            opus_int32 Fs,
            int channels,
            int application )
```

Initializes a previously allocated encoder state The memory pointed to by st must be at least the size returned by opus_encoder_get_size().

This is intended for applications which use their own allocator instead of malloc.

See also

opus_encoder_create(),opus_encoder_get_size() To reset a previously initialized state, use the OPUS_RESET_STATE CTL.

Parameters

in	*st*	OpusEncoder*: Encoder state
in	*Fs*	opus_int32: Sampling rate of input signal (Hz) This must be one of 8000, 12000, 16000, 24000, or 48000.
in	*channels*	int: Number of channels (1 or 2) in input signal
in	*application*	int: Coding mode (one of OPUS_APPLICATION_VOIP, OPUS_APPLICATION_AUDIO, or OPUS_APPLICATION_RESTRICTED_LOWDELAY)

Return values

OPUS_OK	Success or Error codes

4.2 Opus Decoder

This page describes the process and functions used to decode Opus.

Typedefs

- typedef struct OpusDecoder OpusDecoder

  Opus decoder state.
- typedef struct OpusDREDDecoder OpusDREDDecoder

  Opus DRED decoder.
- typedef struct OpusDRED OpusDRED

  Opus DRED state.

Functions

- int opus_decoder_get_size (int channels)

  Gets the size of an `OpusDecoder` structure.
- OpusDecoder * opus_decoder_create (opus_int32 Fs, int channels, int *error)

  Allocates and initializes a decoder state.
- int opus_decoder_init (OpusDecoder *st, opus_int32 Fs, int channels)

  Initializes a previously allocated decoder state.
- int opus_decode (OpusDecoder *st, const unsigned char *data, opus_int32 len, opus_int16 *pcm, int frame_size, int decode_fec)

  Decode an Opus packet.
- int opus_decode24 (OpusDecoder *st, const unsigned char *data, opus_int32 len, opus_int32 *pcm, int frame_size, int decode_fec)

  Decode an Opus packet.
- int opus_decode_float (OpusDecoder *st, const unsigned char *data, opus_int32 len, float *pcm, int frame_size, int decode_fec)

  Decode an Opus packet with floating point output.
- int opus_decoder_ctl (OpusDecoder *st, int request,...)

  Perform a CTL function on an Opus decoder.
- void opus_decoder_destroy (OpusDecoder *st)

  Frees an `OpusDecoder` allocated by opus_decoder_create().
- int opus_dred_decoder_get_size (void)

  Gets the size of an `OpusDREDDecoder` structure.
- OpusDREDDecoder * opus_dred_decoder_create (int *error)

  Allocates and initializes an OpusDREDDecoder state.
- int opus_dred_decoder_init (OpusDREDDecoder *dec)

  Initializes an `OpusDREDDecoder` state.
- void opus_dred_decoder_destroy (OpusDREDDecoder *dec)

  Frees an `OpusDREDDecoder` allocated by opus_dred_decoder_create().
- int opus_dred_decoder_ctl (OpusDREDDecoder *dred_dec, int request,...)

  Perform a CTL function on an Opus DRED decoder.
- int opus_dred_get_size (void)

  Gets the size of an `OpusDRED` structure.

- OpusDRED * opus_dred_alloc (int *error)

  Allocates and initializes a DRED state.
- void opus_dred_free (OpusDRED *dec)

  Frees an `OpusDRED` allocated by opus_dred_create().
- int opus_dred_parse (OpusDREDDecoder *dred_dec, OpusDRED *dred, const unsigned char *data, opus_int32 len, opus_int32 max_dred_samples, opus_int32 sampling_rate, int *dred_end, int defer_processing)

  Decode an Opus DRED packet.
- int opus_dred_process (OpusDREDDecoder *dred_dec, const OpusDRED *src, OpusDRED *dst)

  Finish decoding an Opus DRED packet.
- int opus_decoder_dred_decode (OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset, opus_int16 *pcm, opus_int32 frame_size)

  Decode audio from an Opus DRED packet with 16-bit output.
- int opus_decoder_dred_decode24 (OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset, opus_int32 *pcm, opus_int32 frame_size)

  Decode audio from an Opus DRED packet with 24-bit output.
- int opus_decoder_dred_decode_float (OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset, float *pcm, opus_int32 frame_size)

  Decode audio from an Opus DRED packet with floating point output.
- int opus_packet_parse (const unsigned char *data, opus_int32 len, unsigned char *out_toc, const unsigned char *frames[48], opus_int16 size[48], int *payload_offset)

  Parse an opus packet into one or more frames.
- int opus_packet_get_bandwidth (const unsigned char *data)

  Gets the bandwidth of an Opus packet.
- int opus_packet_get_samples_per_frame (const unsigned char *data, opus_int32 Fs)

  Gets the number of samples per frame from an Opus packet.
- int opus_packet_get_nb_channels (const unsigned char *data)

  Gets the number of channels from an Opus packet.
- int opus_packet_get_nb_frames (const unsigned char packet[ ], opus_int32 len)

  Gets the number of frames in an Opus packet.
- int opus_packet_get_nb_samples (const unsigned char packet[ ], opus_int32 len, opus_int32 Fs)

  Gets the number of samples of an Opus packet.
- int opus_packet_has_lbrr (const unsigned char packet[ ], opus_int32 len)

  Checks whether an Opus packet has LBRR.
- int opus_decoder_get_nb_samples (const OpusDecoder *dec, const unsigned char packet[ ], opus_int32 len)

  Gets the number of samples of an Opus packet.
- void opus_pcm_soft_clip (float *pcm, int frame_size, int channels, float *softclip_mem)

  Applies soft-clipping to bring a float signal within the [-1,1] range.

4.2.1 Detailed Description

This page describes the process and functions used to decode Opus.

The decoding process also starts with creating a decoder state. This can be done with:

```
int          error;
OpusDecoder *dec;
dec = opus_decoder_create(Fs, channels, &error);
```

where

- Fs is the sampling rate and must be 8000, 12000, 16000, 24000, or 48000
- channels is the number of channels (1 or 2)
- error will hold the error code in case of failure (or OPUS_OK on success)
- the return value is a newly created decoder state to be used for decoding

While opus_decoder_create() allocates memory for the state, it's also possible to initialize pre-allocated memory:

```
int          size;
int          error;
OpusDecoder *dec;
size = opus_decoder_get_size(channels);
dec = malloc(size);
error = opus_decoder_init(dec, Fs, channels);
```

where opus_decoder_get_size() returns the required size for the decoder state. Note that future versions of this code may change the size, so no assumptions should be made about it.

The decoder state is always continuous in memory and only a shallow copy is sufficient to copy it (e.g. memcpy())

To decode a frame, opus_decode() or opus_decode_float() must be called with a packet of compressed audio data:

```
frame_size = opus_decode(dec, packet, len, decoded, max_size, 0);
```

where

- packet is the byte array containing the compressed data
- len is the exact number of bytes contained in the packet
- decoded is the decoded audio data in opus_int16 (or float for opus_decode_float())
- max_size is the max duration of the frame in samples (per channel) that can fit into the decoded_frame array

opus_decode() and opus_decode_float() return the number of samples (per channel) decoded from the packet. If that value is negative, then an error has occurred. This can occur if the packet is corrupted or if the audio buffer is too small to hold the decoded audio.

Opus is a stateful codec with overlapping blocks and as a result Opus packets are not coded independently of each other. Packets must be passed into the decoder serially and in the correct order for a correct decode. Lost packets can be replaced with loss concealment by calling the decoder with a null pointer and zero length for the missing packet.

A single codec state may only be accessed from a single thread at a time and any required locking must be performed by the caller. Separate streams must be decoded with separate decoder states and can be decoded in parallel unless the library was compiled with NONTHREADSAFE_PSEUDOSTACK defined.

4.2.2 Typedef Documentation

4.2.2.1 OpusDecoder

```
typedef struct OpusDecoder OpusDecoder
```

Opus decoder state.

This contains the complete state of an Opus decoder. It is position independent and can be freely copied.

See also

opus_decoder_create,opus_decoder_init

4.2.2.2 OpusDRED

```
typedef struct OpusDRED OpusDRED
```

Opus DRED state.

This contains the complete state of an Opus DRED packet. It is position independent and can be freely copied.

See also

opus_dred_create,opus_dred_init

4.2.2.3 OpusDREDDecoder

```
typedef struct OpusDREDDecoder OpusDREDDecoder
```

Opus DRED decoder.

This contains the complete state of an Opus DRED decoder. It is position independent and can be freely copied.

See also

opus_dred_decoder_create,opus_dred_decoder_init

4.2.3 Function Documentation

4.2.3.1 opus_decode()

```
int opus_decode (
            OpusDecoder * st,
            const unsigned char * data,
            opus_int32 len,
            opus_int16 * pcm,
            int frame_size,
            int decode_fec )
```

Decode an Opus packet.

Parameters

in	*st*	OpusDecoder*: Decoder state
in	*data*	char*: Input payload. Use a NULL pointer to indicate packet loss
in	*len*	opus_int32: Number of bytes in payload*
out	*pcm*	opus_int16*: Output signal (interleaved if 2 channels). length is frame_size*channels*sizeof(opus_int16)
in	*frame_size*	Number of samples per channel of available space in *pcm*. If this is less than the maximum packet duration (120ms; 5760 for 48kHz), this function will not be capable of decoding some packets. In the case of PLC (data==NULL) or FEC (decode_fec=1), then frame_size needs to be exactly the duration of audio that is missing, otherwise the decoder will not be in the optimal state to decode the next incoming packet. For the PLC and FEC cases, frame_size **must** be a multiple of 2.5 ms.
in	*decode_fec*	int: Flag (0 or 1) to request that any in-band forward error correction data be decoded. If no such data is available, the frame is decoded as if it were lost.

Returns

Number of decoded samples per channel or Error codes

4.2.3.2 opus_decode24()

```
int opus_decode24 (
              OpusDecoder * st,
              const unsigned char * data,
              opus_int32 len,
              opus_int32 * pcm,
              int frame_size,
              int decode_fec )
```

Decode an Opus packet.

Parameters

in	*st*	OpusDecoder*: Decoder state
in	*data*	char*: Input payload. Use a NULL pointer to indicate packet loss
in	*len*	opus_int32: Number of bytes in payload*
out	*pcm*	opus_int32*: Output signal (interleaved if 2 channels) representing (or slightly exceeding) 24-bit values. length is frame_size*channels*sizeof(opus_int32)
in	*frame_size*	Number of samples per channel of available space in *pcm*. If this is less than the maximum packet duration (120ms; 5760 for 48kHz), this function will not be capable of decoding some packets. In the case of PLC (data==NULL) or FEC (decode_fec=1), then frame_size needs to be exactly the duration of audio that is missing, otherwise the decoder will not be in the optimal state to decode the next incoming packet. For the PLC and FEC cases, frame_size **must** be a multiple of 2.5 ms.
in	*decode_fec*	int: Flag (0 or 1) to request that any in-band forward error correction data be decoded. If no such data is available, the frame is decoded as if it were lost.

Returns

Number of decoded samples or Error codes

4.2.3.3 opus_decode_float()

```
int opus_decode_float (
              OpusDecoder * st,
              const unsigned char * data,
              opus_int32 len,
              float * pcm,
              int frame_size,
              int decode_fec )
```

Decode an Opus packet with floating point output.

Parameters

in	*st*	OpusDecoder∗: Decoder state
in	*data*	char∗: Input payload. Use a NULL pointer to indicate packet loss
in	*len*	opus_int32: Number of bytes in payload
out	*pcm*	float∗: Output signal (interleaved if 2 channels). length is frame_size∗channels∗sizeof(float)
in	*frame_size*	Number of samples per channel of available space in *pcm*. If this is less than the maximum packet duration (120ms; 5760 for 48kHz), this function will not be capable of decoding some packets. In the case of PLC (data==NULL) or FEC (decode_fec=1), then frame_size needs to be exactly the duration of audio that is missing, otherwise the decoder will not be in the optimal state to decode the next incoming packet. For the PLC and FEC cases, frame_size **must** be a multiple of 2.5 ms.
in	*decode_fec*	int: Flag (0 or 1) to request that any in-band forward error correction data be decoded. If no such data is available the frame is decoded as if it were lost.

Returns

Number of decoded samples per channel or Error codes

4.2.3.4 opus_decoder_create()

```
OpusDecoder * opus_decoder_create (
             opus_int32 Fs,
             int channels,
             int * error )
```

Allocates and initializes a decoder state.

Parameters

in	*Fs*	opus_int32: Sample rate to decode at (Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
in	*channels*	int: Number of channels (1 or 2) to decode
out	*error*	int∗: OPUS_OK Success or Error codes

Internally Opus stores data at 48000 Hz, so that should be the default value for Fs. However, the decoder can efficiently decode to buffers at 8, 12, 16, and 24 kHz so if for some reason the caller cannot use data at the full sample rate, or knows the compressed data doesn't use the full frequency range, it can request decoding at a reduced rate. Likewise, the decoder is capable of filling in either mono or interleaved stereo pcm buffers, at the caller's request.

4.2.3.5 opus_decoder_ctl()

```
int opus_decoder_ctl (
             OpusDecoder * st,
```

```
            int request,
             ...  )
```

Perform a CTL function on an Opus decoder.

Generally the request and subsequent arguments are generated by a convenience macro.

Parameters

st	`OpusDecoder*`: Decoder state.
request	This and all remaining parameters should be replaced by one of the convenience macros in Generic CTLs or Decoder related CTLs.

See also

Generic CTLs

Decoder related CTLs

4.2.3.6 opus_decoder_destroy()

```
void opus_decoder_destroy (
            OpusDecoder * st )
```

Frees an `OpusDecoder` allocated by opus_decoder_create().

Parameters

in	*st*	`OpusDecoder*`: State to be freed.

4.2.3.7 opus_decoder_dred_decode()

```
int opus_decoder_dred_decode (
            OpusDecoder * st,
            const OpusDRED * dred,
            opus_int32 dred_offset,
            opus_int16 * pcm,
            opus_int32 frame_size )
```

Decode audio from an Opus DRED packet with 16-bit output.

Parameters

in	*st*	`OpusDecoder*`: Decoder state
in	*dred*	`OpusDRED*`: DRED state
in	*dred_offset*	`opus_int32`: position of the redundancy to decode (in samples before the beginning of the real audio data in the packet).

Parameters

out	*pcm*	opus_int16*: Output signal (interleaved if 2 channels). length is frame_size*channels*sizeof(opus_int16)
in	*frame_size*	Number of samples per channel to decode in *pcm*. frame_size **must** be a multiple of 2.5 ms.

Returns

Number of decoded samples or Error codes

4.2.3.8 opus_decoder_dred_decode24()

```
int opus_decoder_dred_decode24 (
            OpusDecoder * st,
            const OpusDRED * dred,
            opus_int32 dred_offset,
            opus_int32 * pcm,
            opus_int32 frame_size )
```

Decode audio from an Opus DRED packet with 24-bit output.

Parameters

in	*st*	OpusDecoder*: Decoder state
in	*dred*	OpusDRED*: DRED state
in	*dred_offset*	opus_int32: position of the redundancy to decode (in samples before the beginning of the real audio data in the packet).
out	*pcm*	opus_int32*: Output signal (interleaved if 2 channels). length is frame_size*channels*sizeof(opus_int16)
in	*frame_size*	Number of samples per channel to decode in *pcm*. frame_size **must** be a multiple of 2.5 ms.

Returns

Number of decoded samples or Error codes

4.2.3.9 opus_decoder_dred_decode_float()

```
int opus_decoder_dred_decode_float (
            OpusDecoder * st,
            const OpusDRED * dred,
            opus_int32 dred_offset,
            float * pcm,
            opus_int32 frame_size )
```

Decode audio from an Opus DRED packet with floating point output.

Parameters

in	*st*	OpusDecoder*: Decoder state
in	*dred*	OpusDRED*: DRED state
in	*dred_offset*	opus_int32: position of the redundancy to decode (in samples before the beginning of the real audio data in the packet).
out	*pcm*	float*: Output signal (interleaved if 2 channels). length is frame_size*channels*sizeof(float)
in	*frame_size*	Number of samples per channel to decode in *pcm*. frame_size **must** be a multiple of 2.5 ms.

Returns

Number of decoded samples or Error codes

4.2.3.10 opus_decoder_get_nb_samples()

```
int opus_decoder_get_nb_samples (
            const OpusDecoder * dec,
            const unsigned char packet[],
            opus_int32 len )
```

Gets the number of samples of an Opus packet.

Parameters

in	*dec*	OpusDecoder*: Decoder state
in	*packet*	char*: Opus packet
in	*len*	opus_int32: Length of packet

Returns

Number of samples

Return values

OPUS_BAD_ARG	Insufficient data was passed to the function
OPUS_INVALID_PACKET	The compressed data passed is corrupted or of an unsupported type

4.2.3.11 opus_decoder_get_size()

```
int opus_decoder_get_size (
            int channels )
```

Gets the size of an OpusDecoder structure.

Parameters

in	*channels*	int: Number of channels. This must be 1 or 2.

Returns

The size in bytes.

4.2.3.12 opus_decoder_init()

```
int opus_decoder_init (
            OpusDecoder * st,
            opus_int32 Fs,
            int channels )
```

Initializes a previously allocated decoder state.

The state must be at least the size returned by opus_decoder_get_size(). This is intended for applications which use their own allocator instead of malloc.

See also

opus_decoder_create,opus_decoder_get_size To reset a previously initialized state, use the OPUS_RESET_STATE CTL.

Parameters

in	*st*	OpusDecoder*: Decoder state.
in	*Fs*	opus_int32: Sampling rate to decode to (Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
in	*channels*	int: Number of channels (1 or 2) to decode

Return values

OPUS_OK	Success or Error codes

4.2.3.13 opus_dred_alloc()

```
OpusDRED * opus_dred_alloc (
            int * error )
```

Allocates and initializes a DRED state.

Parameters

out	*error*	int*: OPUS_OK Success or Error codes

4.2.3.14 opus_dred_decoder_create()

```
OpusDREDDecoder * opus_dred_decoder_create (
            int * error )
```

Allocates and initializes an OpusDREDDecoder state.

Parameters

out	*error*	int*: OPUS_OK Success or Error codes

4.2.3.15 opus_dred_decoder_ctl()

```
int opus_dred_decoder_ctl (
            OpusDREDDecoder * dred_dec,
            int request,
             ... )
```

Perform a CTL function on an Opus DRED decoder.

Generally the request and subsequent arguments are generated by a convenience macro.

Parameters

dred_dec	OpusDREDDecoder*: DRED Decoder state.
request	This and all remaining parameters should be replaced by one of the convenience macros in Generic CTLs or Decoder related CTLs.

See also

Generic CTLs

Decoder related CTLs

4.2.3.16 opus_dred_decoder_destroy()

```
void opus_dred_decoder_destroy (
            OpusDREDDecoder * dec )
```

Frees an OpusDREDDecoder allocated by opus_dred_decoder_create().

Parameters

in	*dec*	OpusDREDDecoder*: State to be freed.

4.2.3.17 opus_dred_decoder_get_size()

```
int opus_dred_decoder_get_size (
            void  )
```

Gets the size of an OpusDREDDecoder structure.

Returns

The size in bytes.

4.2.3.18 opus_dred_decoder_init()

```
int opus_dred_decoder_init (
            OpusDREDDecoder * dec )
```

Initializes an OpusDREDDecoder state.

Parameters

in	*dec*	OpusDREDDecoder*: State to be initialized.

4.2.3.19 opus_dred_free()

```
void opus_dred_free (
            OpusDRED * dec )
```

Frees an OpusDRED allocated by opus_dred_create().

Parameters

in	*dec*	OpusDRED*: State to be freed.

4.2.3.20 opus_dred_get_size()

```
int opus_dred_get_size (
            void  )
```

Gets the size of an OpusDRED structure.

Returns

The size in bytes.

4.2.3.21 opus_dred_parse()

```
int opus_dred_parse (
            OpusDREDDecoder * dred_dec,
            OpusDRED * dred,
            const unsigned char * data,
            opus_int32 len,
            opus_int32 max_dred_samples,
            opus_int32 sampling_rate,
            int * dred_end,
            int defer_processing )
```

Decode an Opus DRED packet.

Parameters

in	*dred_dec*	OpusDRED*: DRED Decoder state
in	*dred*	OpusDRED*: DRED state
in	*data*	char*: Input payload
in	*len*	opus_int32: Number of bytes in payload
in	*max_dred_samples*	opus_int32: Maximum number of DRED samples that may be needed (if available in the packet).
in	*sampling_rate*	opus_int32: Sampling rate used for max_dred_samples argument. Needs not match the actual sampling rate of the decoder.
out	*dred_end*	opus_int32*: Number of non-encoded (silence) samples between the DRED timestamp and the last DRED sample.
in	*defer_processing*	int: Flag (0 or 1). If set to one, the CPU-intensive part of the DRED decoding is deferred until opus_dred_process() is called.

Returns

Offset (positive) of the first decoded DRED samples, zero if no DRED is present, or Error codes

4.2.3.22 opus_dred_process()

```
int opus_dred_process (
            OpusDREDDecoder * dred_dec,
            const OpusDRED * src,
            OpusDRED * dst )
```

Finish decoding an Opus DRED packet.

The function only needs to be called if opus_dred_parse() was called with defer_processing=1. The source and destination will often be the same DRED state.

Parameters

in	*dred_dec*	OpusDRED*: DRED Decoder state
in	*src*	OpusDRED*: Source DRED state to start the processing from.
out	*dst*	OpusDRED*: Destination DRED state to store the updated state after processing.

Returns

Error codes

4.2.3.23 opus_packet_get_bandwidth()

```
int opus_packet_get_bandwidth (
            const unsigned char * data )
```

Gets the bandwidth of an Opus packet.

Parameters

in	*data*	char*: Opus packet

Return values

OPUS_BANDWIDTH_NARROWBAND	Narrowband (4kHz bandpass)
OPUS_BANDWIDTH_MEDIUMBAND	Mediumband (6kHz bandpass)
OPUS_BANDWIDTH_WIDEBAND	Wideband (8kHz bandpass)
OPUS_BANDWIDTH_SUPERWIDEBAND	Superwideband (12kHz bandpass)
OPUS_BANDWIDTH_FULLBAND	Fullband (20kHz bandpass)
OPUS_INVALID_PACKET	The compressed data passed is corrupted or of an unsupported type

4.2.3.24 opus_packet_get_nb_channels()

```
int opus_packet_get_nb_channels (
            const unsigned char * data )
```

Gets the number of channels from an Opus packet.

Parameters

in	*data*	char*: Opus packet

Returns

Number of channels

Return values

OPUS_INVALID_PACKET	The compressed data passed is corrupted or of an unsupported type

4.2.3.25 opus_packet_get_nb_frames()

```
int opus_packet_get_nb_frames (
            const unsigned char packet[],
            opus_int32 len )
```

Gets the number of frames in an Opus packet.

Parameters

in	*packet*	`char*`: Opus packet
in	*len*	`opus_int32`: Length of packet

Returns

Number of frames

Return values

OPUS_BAD_ARG	Insufficient data was passed to the function
OPUS_INVALID_PACKET	The compressed data passed is corrupted or of an unsupported type

4.2.3.26 opus_packet_get_nb_samples()

```
int opus_packet_get_nb_samples (
            const unsigned char packet[],
            opus_int32 len,
            opus_int32 Fs )
```

Gets the number of samples of an Opus packet.

Parameters

in	*packet*	`char*`: Opus packet
in	*len*	`opus_int32`: Length of packet
in	*Fs*	`opus_int32`: Sampling rate in Hz. This must be a multiple of 400, or inaccurate results will be returned.

Returns

Number of samples

Return values

OPUS_BAD_ARG	Insufficient data was passed to the function
OPUS_INVALID_PACKET	The compressed data passed is corrupted or of an unsupported type

4.2.3.27 opus_packet_get_samples_per_frame()

```
int opus_packet_get_samples_per_frame (
            const unsigned char * data,
            opus_int32 Fs )
```

Gets the number of samples per frame from an Opus packet.

Parameters

in	*data*	`char*`: Opus packet. This must contain at least one byte of data.
in	*Fs*	`opus_int32`: Sampling rate in Hz. This must be a multiple of 400, or inaccurate results will be returned.

Returns

Number of samples per frame.

4.2.3.28 opus_packet_has_lbrr()

```
int opus_packet_has_lbrr (
            const unsigned char packet[],
            opus_int32 len )
```

Checks whether an Opus packet has LBRR.

Parameters

in	*packet*	`char*`: Opus packet
in	*len*	`opus_int32`: Length of packet

Returns

1 is LBRR is present, 0 otherwise

Return values

OPUS_INVALID_PACKET	The compressed data passed is corrupted or of an unsupported type

4.2.3.29 opus_packet_parse()

```
int opus_packet_parse (
             const unsigned char * data,
             opus_int32 len,
             unsigned char * out_toc,
             const unsigned char * frames[48],
             opus_int16 size[48],
             int * payload_offset )
```

Parse an opus packet into one or more frames.

Opus_decode will perform this operation internally so most applications do not need to use this function. This function does not copy the frames, the returned pointers are pointers into the input packet.

Parameters

in	*data*	`char*`: Opus packet to be parsed
in	*len*	`opus_int32`: size of data
out	*out_toc*	`char*`: TOC pointer
out	*frames*	`char*[48]` encapsulated frames
out	*size*	`opus_int16[48]` sizes of the encapsulated frames
out	*payload_offset*	`int*`: returns the position of the payload within the packet (in bytes)

Returns

number of frames

4.2.3.30 opus_pcm_soft_clip()

```
void opus_pcm_soft_clip (
             float * pcm,
             int frame_size,
             int channels,
             float * softclip_mem )
```

Applies soft-clipping to bring a float signal within the [-1,1] range.

If the signal is already in that range, nothing is done. If there are values outside of [-1,1], then the signal is clipped as smoothly as possible to both fit in the range and avoid creating excessive distortion in the process.

Parameters

in,out	*pcm*	float*: Input PCM and modified PCM
in	*frame_size*	int Number of samples per channel to process
in	*channels*	int: Number of channels
in,out	*softclip_mem*	float*: State memory for the soft clipping process (one float per channel, initialized to zero)

4.3 Repacketizer

The repacketizer can be used to merge multiple Opus packets into a single packet or alternatively to split Opus packets that have previously been merged.

Typedefs

- typedef struct OpusRepacketizer OpusRepacketizer

Functions

- int opus_repacketizer_get_size (void)

  Gets the size of an `OpusRepacketizer` structure.
- OpusRepacketizer * opus_repacketizer_init (OpusRepacketizer *rp)

  (Re)initializes a previously allocated repacketizer state.
- OpusRepacketizer * opus_repacketizer_create (void)

  Allocates memory and initializes the new repacketizer with opus_repacketizer_init().
- void opus_repacketizer_destroy (OpusRepacketizer *rp)

  Frees an `OpusRepacketizer` allocated by opus_repacketizer_create().
- int opus_repacketizer_cat (OpusRepacketizer *rp, const unsigned char *data, opus_int32 len)

  Add a packet to the current repacketizer state.
- opus_int32 opus_repacketizer_out_range (OpusRepacketizer *rp, int begin, int end, unsigned char *data, opus_int32 maxlen)

  Construct a new packet from data previously submitted to the repacketizer state via opus_repacketizer_cat().
- int opus_repacketizer_get_nb_frames (OpusRepacketizer *rp)

  Return the total number of frames contained in packet data submitted to the repacketizer state so far via opus_repacketizer_cat() since the last call to opus_repacketizer_init() or opus_repacketizer_create().
- opus_int32 opus_repacketizer_out (OpusRepacketizer *rp, unsigned char *data, opus_int32 maxlen)

  Construct a new packet from data previously submitted to the repacketizer state via opus_repacketizer_cat().
- int opus_packet_pad (unsigned char *data, opus_int32 len, opus_int32 new_len)

  Pads a given Opus packet to a larger size (possibly changing the TOC sequence).
- opus_int32 opus_packet_unpad (unsigned char *data, opus_int32 len)

  Remove all padding from a given Opus packet and rewrite the TOC sequence to minimize space usage.
- int opus_multistream_packet_pad (unsigned char *data, opus_int32 len, opus_int32 new_len, int nb_streams)

  Pads a given Opus multi-stream packet to a larger size (possibly changing the TOC sequence).
- opus_int32 opus_multistream_packet_unpad (unsigned char *data, opus_int32 len, int nb_streams)

  Remove all padding from a given Opus multi-stream packet and rewrite the TOC sequence to minimize space usage.

4.3.1 Detailed Description

The repacketizer can be used to merge multiple Opus packets into a single packet or alternatively to split Opus packets that have previously been merged.

Splitting valid Opus packets is always guaranteed to succeed, whereas merging valid packets only succeeds if all frames have the same mode, bandwidth, and frame size, and when the total duration of the merged packet is no more than 120 ms. The 120 ms limit comes from the specification and limits decoder memory requirements at a point where framing overhead becomes negligible.

The repacketizer currently only operates on elementary Opus streams. It will not manipulate multistream packets successfully, except in the degenerate case where they consist of data from a single stream.

The repacketizing process starts with creating a repacketizer state, either by calling opus_repacketizer_create() or by allocating the memory yourself, e.g.,

```
OpusRepacketizer *rp;
rp = (OpusRepacketizer*)malloc(opus_repacketizer_get_size());
if (rp != NULL)
    opus_repacketizer_init(rp);
```

Then the application should submit packets with opus_repacketizer_cat(), extract new packets with opus_repacketizer_out() or opus_repacketizer_out_range(), and then reset the state for the next set of input packets via opus_repacketizer_init().

For example, to split a sequence of packets into individual frames:

```
unsigned char *data;
int len;
while (get_next_packet(&data, &len))
{
  unsigned char out[1276];
  opus_int32 out_len;
  int nb_frames;
  int err;
  int i;
  err = opus_repacketizer_cat(rp, data, len);
  if (err != OPUS_OK)
  {
    release_packet(data);
    return err;
  }
  nb_frames = opus_repacketizer_get_nb_frames(rp);
  for (i = 0; i < nb_frames; i++)
  {
    out_len = opus_repacketizer_out_range(rp, i, i+1, out, sizeof(out));
    if (out_len < 0)
    {
       release_packet(data);
       return (int)out_len;
    }
    output_next_packet(out, out_len);
  }
  opus_repacketizer_init(rp);
  release_packet(data);
}
```

Alternatively, to combine a sequence of frames into packets that each contain up to `TARGET_DURATION_MS` milliseconds of data:

```
// The maximum number of packets with duration TARGET_DURATION_MS occurs
// when the frame size is 2.5 ms, for a total of (TARGET_DURATION_MS*2/5)
// packets.
unsigned char *data[(TARGET_DURATION_MS*2/5)+1];
opus_int32 len[(TARGET_DURATION_MS*2/5)+1];
int nb_packets;
unsigned char out[1277*(TARGET_DURATION_MS*2/2)];
opus_int32 out_len;
int prev_toc;
nb_packets = 0;
while (get_next_packet(data+nb_packets, len+nb_packets))
{
  int nb_frames;
```

```
  int err;
  nb_frames = opus_packet_get_nb_frames(data[nb_packets], len[nb_packets]);
  if (nb_frames < 1)
  {
    release_packets(data, nb_packets+1);
    return nb_frames;
  }
  nb_frames += opus_repacketizer_get_nb_frames(rp);
  // If adding the next packet would exceed our target, or it has an
  // incompatible TOC sequence, output the packets we already have before
  // submitting it.
  // N.B., The nb_packets > 0 check ensures we've submitted at least one
  // packet since the last call to opus_repacketizer_init(). Otherwise a
  // single packet longer than TARGET_DURATION_MS would cause us to try to
  // output an (invalid) empty packet. It also ensures that prev_toc has
  // been set to a valid value. Additionally, len[nb_packets] > 0 is
  // guaranteed by the call to opus_packet_get_nb_frames() above, so the
  // reference to data[nb_packets][0] should be valid.
  if (nb_packets > 0 && (
      ((prev_toc & 0xFC) != (data[nb_packets][0] & 0xFC)) ||
      opus_packet_get_samples_per_frame(data[nb_packets], 48000)*nb_frames >
      TARGET_DURATION_MS*48))
  {
    out_len = opus_repacketizer_out(rp, out, sizeof(out));
    if (out_len < 0)
    {
       release_packets(data, nb_packets+1);
       return (int)out_len;
    }
    output_next_packet(out, out_len);
    opus_repacketizer_init(rp);
    release_packets(data, nb_packets);
    data[0] = data[nb_packets];
    len[0] = len[nb_packets];
    nb_packets = 0;
  }
  err = opus_repacketizer_cat(rp, data[nb_packets], len[nb_packets]);
  if (err != OPUS_OK)
  {
    release_packets(data, nb_packets+1);
    return err;
  }
  prev_toc = data[nb_packets][0];
  nb_packets++;
}
// Output the final, partial packet.
if (nb_packets > 0)
{
  out_len = opus_repacketizer_out(rp, out, sizeof(out));
  release_packets(data, nb_packets);
  if (out_len < 0)
    return (int)out_len;
  output_next_packet(out, out_len);
}
```

An alternate way of merging packets is to simply call opus_repacketizer_cat() unconditionally until it fails. At that point, the merged packet can be obtained with opus_repacketizer_out() and the input packet for which opus_repacketizer_cat() needs to be re-added to a newly reinitialized repacketizer state.

4.3.2 Typedef Documentation

4.3.2.1 OpusRepacketizer

```
typedef struct OpusRepacketizer OpusRepacketizer
```

4.3.3 Function Documentation

4.3.3.1 opus_multistream_packet_pad()

```
int opus_multistream_packet_pad (
            unsigned char * data,
```

```
opus_int32 len,
opus_int32 new_len,
int nb_streams )
```

Pads a given Opus multi-stream packet to a larger size (possibly changing the TOC sequence).

Parameters

in,out		
	data	const unsigned char*: The buffer containing the packet to pad.
	len	opus_int32: The size of the packet. This must be at least 1.
	new_len	opus_int32: The desired size of the packet after padding. This must be at least 1.
	nb_streams	opus_int32: The number of streams (not channels) in the packet. This must be at least as large as len.

Returns

an error code

Return values

OPUS_OK	*on* success.
OPUS_BAD_ARG	*len* was less than 1.
OPUS_INVALID_PACKET	*data* did not contain a valid Opus packet.

4.3.3.2 opus_multistream_packet_unpad()

```
opus_int32 opus_multistream_packet_unpad (
              unsigned char * data,
              opus_int32 len,
              int nb_streams )
```

Remove all padding from a given Opus multi-stream packet and rewrite the TOC sequence to minimize space usage.

Parameters

in,out		
	data	const unsigned char*: The buffer containing the packet to strip.
	len	opus_int32: The size of the packet. This must be at least 1.
	nb_streams	opus_int32: The number of streams (not channels) in the packet. This must be at least 1.

Returns

The new size of the output packet on success, or an error code on failure.

Return values

OPUS_BAD_ARG	*len* was less than 1 or new_len was less than len.
OPUS_INVALID_PACKET	*data* did not contain a valid Opus packet.

4.3.3.3 opus_packet_pad()

```
int opus_packet_pad (
              unsigned char * data,
              opus_int32 len,
              opus_int32 new_len )
```

Pads a given Opus packet to a larger size (possibly changing the TOC sequence).

Parameters

in,out	*data*	`const unsigned char*`: The buffer containing the packet to pad.
	len	`opus_int32`: The size of the packet. This must be at least 1.
	new_len	`opus_int32`: The desired size of the packet after padding. This must be at least as large as len.

Returns

an error code

Return values

OPUS_OK	*on* success.
OPUS_BAD_ARG	*len* was less than 1 or new_len was less than len.
OPUS_INVALID_PACKET	*data* did not contain a valid Opus packet.

4.3.3.4 opus_packet_unpad()

```
opus_int32 opus_packet_unpad (
              unsigned char * data,
              opus_int32 len )
```

Remove all padding from a given Opus packet and rewrite the TOC sequence to minimize space usage.

Parameters

in,out	*data*	`const unsigned char*`: The buffer containing the packet to strip.
	len	`opus_int32`: The size of the packet. This must be at least 1.

Returns

The new size of the output packet on success, or an error code on failure.

Return values

OPUS_BAD_ARG	*len* was less than 1.
OPUS_INVALID_PACKET	*data* did not contain a valid Opus packet.

4.3.3.5 opus_repacketizer_cat()

```
int opus_repacketizer_cat (
            OpusRepacketizer * rp,
            const unsigned char * data,
            opus_int32 len )
```

Add a packet to the current repacketizer state.

This packet must match the configuration of any packets already submitted for repacketization since the last call to opus_repacketizer_init(). This means that it must have the same coding mode, audio bandwidth, frame size, and channel count. This can be checked in advance by examining the top 6 bits of the first byte of the packet, and ensuring they match the top 6 bits of the first byte of any previously submitted packet. The total duration of audio in the repacketizer state also must not exceed 120 ms, the maximum duration of a single packet, after adding this packet.

The contents of the current repacketizer state can be extracted into new packets using opus_repacketizer_out() or opus_repacketizer_out_range().

In order to add a packet with a different configuration or to add more audio beyond 120 ms, you must clear the repacketizer state by calling opus_repacketizer_init(). If a packet is too large to add to the current repacketizer state, no part of it is added, even if it contains multiple frames, some of which might fit. If you wish to be able to add parts of such packets, you should first use another repacketizer to split the packet into pieces and add them individually.

See also

opus_repacketizer_out_range

opus_repacketizer_out

opus_repacketizer_init

Parameters

	rp	`OpusRepacketizer*`: The repacketizer state to which to add the packet.
in	*data*	`const unsigned char*`: The packet data. The application must ensure this pointer remains valid until the next call to opus_repacketizer_init() or opus_repacketizer_destroy().
	len	`opus_int32`: The number of bytes in the packet data.

Returns

An error code indicating whether or not the operation succeeded.

Return values

OPUS_OK	The packet's contents have been added to the repacketizer state.
OPUS_INVALID_PACKET	The packet did not have a valid TOC sequence, the packet's TOC sequence was not compatible with previously submitted packets (because the coding mode, audio bandwidth, frame size, or channel count did not match), or adding this packet would increase the total amount of audio stored in the repacketizer state to more than 120 ms.

4.3.3.6 opus_repacketizer_create()

```
OpusRepacketizer * opus_repacketizer_create (
            void  )
```

Allocates memory and initializes the new repacketizer with opus_repacketizer_init().

4.3.3.7 opus_repacketizer_destroy()

```
void opus_repacketizer_destroy (
            OpusRepacketizer * rp )
```

Frees an `OpusRepacketizer` allocated by opus_repacketizer_create().

Parameters

in	*rp*	`OpusRepacketizer*`: State to be freed.

4.3.3.8 opus_repacketizer_get_nb_frames()

```
int opus_repacketizer_get_nb_frames (
            OpusRepacketizer * rp )
```

Return the total number of frames contained in packet data submitted to the repacketizer state so far via opus_repacketizer_cat() since the last call to opus_repacketizer_init() or opus_repacketizer_create().

This defines the valid range of packets that can be extracted with opus_repacketizer_out_range() or opus_repacketizer_out().

Parameters

rp	`OpusRepacketizer*`: The repacketizer state containing the frames.

Returns

The total number of frames contained in the packet data submitted to the repacketizer state.

4.3.3.9 opus_repacketizer_get_size()

```
int opus_repacketizer_get_size (
            void  )
```

Gets the size of an `OpusRepacketizer` structure.

Returns

The size in bytes.

4.3.3.10 opus_repacketizer_init()

```
OpusRepacketizer * opus_repacketizer_init (
            OpusRepacketizer * rp )
```

(Re)initializes a previously allocated repacketizer state.

The state must be at least the size returned by opus_repacketizer_get_size(). This can be used for applications which use their own allocator instead of malloc(). It must also be called to reset the queue of packets waiting to be repacketized, which is necessary if the maximum packet duration of 120 ms is reached or if you wish to submit packets with a different Opus configuration (coding mode, audio bandwidth, frame size, or channel count). Failure to do so will prevent a new packet from being added with opus_repacketizer_cat().

See also

opus_repacketizer_create

opus_repacketizer_get_size

opus_repacketizer_cat

Parameters

rp	`OpusRepacketizer*`: The repacketizer state to (re)initialize.

Returns

A pointer to the same repacketizer state that was passed in.

4.3.3.11 opus_repacketizer_out()

```
opus_int32 opus_repacketizer_out (
            OpusRepacketizer * rp,
```

```
            unsigned char * data,
            opus_int32 maxlen )
```

Construct a new packet from data previously submitted to the repacketizer state via opus_repacketizer_cat().

This is a convenience routine that returns all the data submitted so far in a single packet. It is equivalent to calling
`opus_repacketizer_out_range(rp, 0, opus_repacketizer_get_nb_frames(rp), data, maxlen)`

Parameters

	rp	`OpusRepacketizer*`: The repacketizer state from which to construct the new packet.
out	*data*	`const unsigned char*`: The buffer in which to store the output packet.
	maxlen	`opus_int32`: The maximum number of bytes to store in the output buffer. In order to guarantee success, this should be at least `1277*opus_repacketizer_get_nb_frames(rp)`. However, `1*opus_repacketizer_get_nb_frames(rp)` plus the size of all packet data submitted to the repacketizer since the last call to opus_repacketizer_init() or opus_repacketizer_create() is also sufficient, and possibly much smaller.

Returns

The total size of the output packet on success, or an error code on failure.

Return values

OPUS_BUFFER_TOO_SMALL	*maxlen* was insufficient to contain the complete output packet.

4.3.3.12 opus_repacketizer_out_range()

```
opus_int32 opus_repacketizer_out_range (
            OpusRepacketizer * rp,
            int begin,
            int end,
            unsigned char * data,
            opus_int32 maxlen )
```

Construct a new packet from data previously submitted to the repacketizer state via opus_repacketizer_cat().

Parameters

	rp	`OpusRepacketizer*`: The repacketizer state from which to construct the new packet.
	begin	`int`: The index of the first frame in the current repacketizer state to include in the output.
	end	`int`: One past the index of the last frame in the current repacketizer state to include in the output.
out	*data*	`const unsigned char*`: The buffer in which to store the output packet.
	maxlen	`opus_int32`: The maximum number of bytes to store in the output buffer. In order to guarantee success, this should be at least `1276` for a single frame, or for multiple frames, `1277*(end-begin)`. However, `1*(end-begin)` plus the size of all packet data submitted to the repacketizer since the last call to opus_repacketizer_init() or opus_repacketizer_create() is also sufficient, and possibly much smaller.

Returns

The total size of the output packet on success, or an error code on failure.

Return values

OPUS_BAD_ARG	`[begin,end)` was an invalid range of frames (begin < 0, begin >= end, or end > opus_repacketizer_get_nb_frames()).
OPUS_BUFFER_TOO_SMALL	*maxlen* was insufficient to contain the complete output packet.

4.4 Error codes

Macros

- #define OPUS_OK

  No error.
- #define OPUS_BAD_ARG

  One or more invalid/out of range arguments.
- #define OPUS_BUFFER_TOO_SMALL

  Not enough bytes allocated in the buffer.
- #define OPUS_INTERNAL_ERROR

  An internal error was detected.
- #define OPUS_INVALID_PACKET

  The compressed data passed is corrupted.
- #define OPUS_UNIMPLEMENTED

  Invalid/unsupported request number.
- #define OPUS_INVALID_STATE

  An encoder or decoder structure is invalid or already freed.
- #define OPUS_ALLOC_FAIL

  Memory allocation has failed.

4.4.1 Detailed Description

4.4.2 Macro Definition Documentation

4.4.2.1 OPUS_ALLOC_FAIL

```
#define OPUS_ALLOC_FAIL
```

Memory allocation has failed.

4.4.2.2 OPUS_BAD_ARG

```
#define OPUS_BAD_ARG
```

One or more invalid/out of range arguments.

4.4.2.3 OPUS_BUFFER_TOO_SMALL

```
#define OPUS_BUFFER_TOO_SMALL
```

Not enough bytes allocated in the buffer.

4.4.2.4 OPUS_INTERNAL_ERROR

```
#define OPUS_INTERNAL_ERROR
```

An internal error was detected.

4.4.2.5 OPUS_INVALID_PACKET

```
#define OPUS_INVALID_PACKET
```

The compressed data passed is corrupted.

4.4.2.6 OPUS_INVALID_STATE

```
#define OPUS_INVALID_STATE
```

An encoder or decoder structure is invalid or already freed.

4.4.2.7 OPUS_OK

```
#define OPUS_OK
```

No error.

4.4.2.8 OPUS_UNIMPLEMENTED

```
#define OPUS_UNIMPLEMENTED
```

Invalid/unsupported request number.

4.5 Pre-defined values for CTL interface

Macros

- #define OPUS_AUTO

  Auto/default setting.
- #define OPUS_BITRATE_MAX

  Maximum bitrate.
- #define OPUS_APPLICATION_VOIP

  Best for most VoIP/videoconference applications where listening quality and intelligibility matter most.
- #define OPUS_APPLICATION_AUDIO

  Best for broadcast/high-fidelity application where the decoded audio should be as close as possible to the input.
- #define OPUS_APPLICATION_RESTRICTED_LOWDELAY

  Only use when lowest-achievable latency is what matters most.
- #define OPUS_APPLICATION_RESTRICTED_SILK 2052

  Experts only: forces SILK encoding; don't allocate CELT state at all.
- #define OPUS_APPLICATION_RESTRICTED_CELT 2053

  Experts only: forces CELT encoding; don't allocate SILK state at all.
- #define OPUS_SIGNAL_VOICE 3001

  Signal being encoded is voice.
- #define OPUS_SIGNAL_MUSIC 3002

  Signal being encoded is music.
- #define OPUS_BANDWIDTH_NARROWBAND

  4 kHz bandpass
- #define OPUS_BANDWIDTH_MEDIUMBAND

  6 kHz bandpass
- #define OPUS_BANDWIDTH_WIDEBAND

  8 kHz bandpass
- #define OPUS_BANDWIDTH_SUPERWIDEBAND

  12 kHz bandpass
- #define OPUS_BANDWIDTH_FULLBAND

  20 kHz bandpass
- #define OPUS_FRAMESIZE_ARG 5000

  Select frame size from the argument (default)
- #define OPUS_FRAMESIZE_2_5_MS 5001

  Use 2.5 ms frames.
- #define OPUS_FRAMESIZE_5_MS 5002

  Use 5 ms frames.
- #define OPUS_FRAMESIZE_10_MS 5003

  Use 10 ms frames.
- #define OPUS_FRAMESIZE_20_MS 5004

  Use 20 ms frames.
- #define OPUS_FRAMESIZE_40_MS 5005

  Use 40 ms frames.
- #define OPUS_FRAMESIZE_60_MS 5006

  Use 60 ms frames.
- #define OPUS_FRAMESIZE_80_MS 5007

Use 80 ms frames.

- #define OPUS_FRAMESIZE_100_MS 5008

  Use 100 ms frames.

- #define OPUS_FRAMESIZE_120_MS 5009

  Use 120 ms frames.

4.5.1 Detailed Description

See also

Generic CTLs, Encoder related CTLs

4.5.2 Macro Definition Documentation

4.5.2.1 OPUS_APPLICATION_AUDIO

```
#define OPUS_APPLICATION_AUDIO
```

Best for broadcast/high-fidelity application where the decoded audio should be as close as possible to the input.

4.5.2.2 OPUS_APPLICATION_RESTRICTED_CELT

```
#define OPUS_APPLICATION_RESTRICTED_CELT 2053
```

Experts only: forces CELT encoding; don't allocate SILK state at all.

Disables OPUS_SET_APPLICATION.

4.5.2.3 OPUS_APPLICATION_RESTRICTED_LOWDELAY

```
#define OPUS_APPLICATION_RESTRICTED_LOWDELAY
```

Only use when lowest-achievable latency is what matters most.

Voice-optimized modes cannot be used.

4.5.2.4 OPUS_APPLICATION_RESTRICTED_SILK

```
#define OPUS_APPLICATION_RESTRICTED_SILK 2052
```

Experts only: forces SILK encoding; don't allocate CELT state at all.

Disables OPUS_SET_APPLICATION.

4.5.2.5 OPUS_APPLICATION_VOIP

```
#define OPUS_APPLICATION_VOIP
```

Best for most VoIP/videoconference applications where listening quality and intelligibility matter most.

4.5.2.6 OPUS_AUTO

```
#define OPUS_AUTO
```

Auto/default setting.

4.5.2.7 OPUS_BANDWIDTH_FULLBAND

```
#define OPUS_BANDWIDTH_FULLBAND
```

20 kHz bandpass

4.5.2.8 OPUS_BANDWIDTH_MEDIUMBAND

```
#define OPUS_BANDWIDTH_MEDIUMBAND
```

6 kHz bandpass

4.5.2.9 OPUS_BANDWIDTH_NARROWBAND

```
#define OPUS_BANDWIDTH_NARROWBAND
```

4 kHz bandpass

4.5.2.10 OPUS_BANDWIDTH_SUPERWIDEBAND

```
#define OPUS_BANDWIDTH_SUPERWIDEBAND
```

12 kHz bandpass

4.5.2.11 OPUS_BANDWIDTH_WIDEBAND

```
#define OPUS_BANDWIDTH_WIDEBAND
```

8 kHz bandpass

4.5.2.12 OPUS_BITRATE_MAX

```
#define OPUS_BITRATE_MAX
```

Maximum bitrate.

4.5.2.13 OPUS_FRAMESIZE_100_MS

```
#define OPUS_FRAMESIZE_100_MS 5008
```

Use 100 ms frames.

4.5.2.14 OPUS_FRAMESIZE_10_MS

```
#define OPUS_FRAMESIZE_10_MS 5003
```

Use 10 ms frames.

4.5.2.15 OPUS_FRAMESIZE_120_MS

```
#define OPUS_FRAMESIZE_120_MS 5009
```

Use 120 ms frames.

4.5.2.16 OPUS_FRAMESIZE_20_MS

```
#define OPUS_FRAMESIZE_20_MS 5004
```

Use 20 ms frames.

4.5.2.17 OPUS_FRAMESIZE_2_5_MS

```
#define OPUS_FRAMESIZE_2_5_MS 5001
```

Use 2.5 ms frames.

4.5.2.18 OPUS_FRAMESIZE_40_MS

```
#define OPUS_FRAMESIZE_40_MS 5005
```

Use 40 ms frames.

4.5.2.19 OPUS_FRAMESIZE_5_MS

```
#define OPUS_FRAMESIZE_5_MS 5002
```

Use 5 ms frames.

4.5.2.20 OPUS_FRAMESIZE_60_MS

```
#define OPUS_FRAMESIZE_60_MS 5006
```

Use 60 ms frames.

4.5.2.21 OPUS_FRAMESIZE_80_MS

```
#define OPUS_FRAMESIZE_80_MS 5007
```

Use 80 ms frames.

4.5.2.22 OPUS_FRAMESIZE_ARG

```
#define OPUS_FRAMESIZE_ARG 5000
```

Select frame size from the argument (default)

4.5.2.23 OPUS_SIGNAL_MUSIC

```
#define OPUS_SIGNAL_MUSIC 3002
```

Signal being encoded is music.

4.5.2.24 OPUS_SIGNAL_VOICE

```
#define OPUS_SIGNAL_VOICE 3001
```

Signal being encoded is voice.

4.6 Encoder related CTLs

These are convenience macros for use with the `opus_encode_ctl` interface.

Macros

- #define OPUS_SET_COMPLEXITY(x)

  Configures the encoder's computational complexity.
- #define OPUS_GET_COMPLEXITY(x)

  Gets the encoder's complexity configuration.
- #define OPUS_SET_BITRATE(x)

  Configures the bitrate in the encoder.
- #define OPUS_GET_BITRATE(x)

  Gets the encoder's bitrate configuration.
- #define OPUS_SET_VBR(x)

  Enables or disables variable bitrate (VBR) in the encoder.
- #define OPUS_GET_VBR(x)

  Determine if variable bitrate (VBR) is enabled in the encoder.
- #define OPUS_SET_VBR_CONSTRAINT(x)

  Enables or disables constrained VBR in the encoder.
- #define OPUS_GET_VBR_CONSTRAINT(x)

  Determine if constrained VBR is enabled in the encoder.
- #define OPUS_SET_FORCE_CHANNELS(x)

  Configures mono/stereo forcing in the encoder.
- #define OPUS_GET_FORCE_CHANNELS(x)

  Gets the encoder's forced channel configuration.
- #define OPUS_SET_MAX_BANDWIDTH(x)

  Configures the maximum bandpass that the encoder will select automatically.
- #define OPUS_GET_MAX_BANDWIDTH(x)

  Gets the encoder's configured maximum allowed bandpass.
- #define OPUS_SET_BANDWIDTH(x)

  Sets the encoder's bandpass to a specific value.
- #define OPUS_SET_SIGNAL(x)

  Configures the type of signal being encoded.
- #define OPUS_GET_SIGNAL(x)

  Gets the encoder's configured signal type.
- #define OPUS_SET_APPLICATION(x)

  Configures the encoder's intended application.
- #define OPUS_GET_APPLICATION(x)

  Gets the encoder's configured application.
- #define OPUS_GET_LOOKAHEAD(x)

  Gets the total samples of delay added by the entire codec.
- #define OPUS_SET_INBAND_FEC(x)

  Configures the encoder's use of inband forward error correction (FEC).
- #define OPUS_GET_INBAND_FEC(x)

  Gets encoder's configured use of inband forward error correction.
- #define OPUS_SET_PACKET_LOSS_PERC(x)

  Configures the encoder's expected packet loss percentage.
- #define OPUS_GET_PACKET_LOSS_PERC(x)

  Gets the encoder's configured packet loss percentage.
- #define OPUS_SET_DTX(x)

Configures the encoder's use of discontinuous transmission (DTX).

- #define OPUS_GET_DTX(x)

  Gets encoder's configured use of discontinuous transmission.
- #define OPUS_SET_LSB_DEPTH(x)

  Configures the depth of signal being encoded.
- #define OPUS_GET_LSB_DEPTH(x)

  Gets the encoder's configured signal depth.
- #define OPUS_SET_EXPERT_FRAME_DURATION(x)

  Configures the encoder's use of variable duration frames.
- #define OPUS_GET_EXPERT_FRAME_DURATION(x)

  Gets the encoder's configured use of variable duration frames.
- #define OPUS_SET_PREDICTION_DISABLED(x)

  If set to 1, disables almost all use of prediction, making frames almost completely independent.
- #define OPUS_GET_PREDICTION_DISABLED(x)

  Gets the encoder's configured prediction status.
- #define OPUS_SET_DRED_DURATION(x)

  If non-zero, enables Deep Redundancy (DRED) and use the specified maximum number of 10-ms redundant frames.
- #define OPUS_GET_DRED_DURATION(x)

  Gets the encoder's configured Deep Redundancy (DRED) maximum number of frames.
- #define OPUS_SET_DNN_BLOB(data, len)

  Provide external DNN weights from binary object (only when explicitly built without the weights)
- #define OPUS_SET_QEXT(x)

  If set to 1, enables quality extension (QEXT), otherwise disables it (default).
- #define OPUS_GET_QEXT(x)

  Gets the encoder's configured quality extension (QEXT).

4.6.1 Detailed Description

These are convenience macros for use with the `opus_encode_ctl` interface.

They are used to generate the appropriate series of arguments for that call, passing the correct type, size and so on as expected for each particular request.

Some usage examples:

```
int ret;
ret = opus_encoder_ctl(enc_ctx, OPUS_SET_BANDWIDTH(OPUS_AUTO));
if (ret != OPUS_OK) return ret;

opus_int32 rate;
opus_encoder_ctl(enc_ctx, OPUS_GET_BANDWIDTH(&rate));

opus_encoder_ctl(enc_ctx, OPUS_RESET_STATE);
```

See also

Generic CTLs, Opus Encoder

4.6.2 Macro Definition Documentation

4.6.2.1 OPUS_GET_APPLICATION

```
#define OPUS_GET_APPLICATION(
              x )
```

Gets the encoder's configured application.

See also

OPUS_SET_APPLICATION

Parameters

out	*x*	`opus_int32 *`: Returns one of the following values: **OPUS_APPLICATION_VOIP** Process signal for improved speech intelligibility. **OPUS_APPLICATION_AUDIO** Favor faithfulness to the original input. **OPUS_APPLICATION_RESTRICTED_LOWDELAY** Configure the minimum possible coding delay by disabling certain modes of operation.

4.6.2.2 OPUS_GET_BITRATE

```
#define OPUS_GET_BITRATE(
              x )
```

Gets the encoder's bitrate configuration.

See also

OPUS_SET_BITRATE

Parameters

out	*x*	`opus_int32 *`: Returns the bitrate in bits per second. The default is determined based on the number of channels and the input sampling rate.

4.6.2.3 OPUS_GET_COMPLEXITY

```
#define OPUS_GET_COMPLEXITY(
              x )
```

Gets the encoder's complexity configuration.

See also

OPUS_SET_COMPLEXITY

Parameters

out	*x*	opus_int32 *: Returns a value in the range 0-10, inclusive.

4.6.2.4 OPUS_GET_DRED_DURATION

```
#define OPUS_GET_DRED_DURATION(
              x )
```

Gets the encoder's configured Deep Redundancy (DRED) maximum number of frames.

4.6.2.5 OPUS_GET_DTX

```
#define OPUS_GET_DTX(
              x )
```

Gets encoder's configured use of discontinuous transmission.

See also

OPUS_SET_DTX

Parameters

out	*x*	opus_int32 *: Returns one of the following values: **0** DTX disabled (default). **1** DTX enabled.

4.6.2.6 OPUS_GET_EXPERT_FRAME_DURATION

```
#define OPUS_GET_EXPERT_FRAME_DURATION(
              x )
```

Gets the encoder's configured use of variable duration frames.

See also

OPUS_SET_EXPERT_FRAME_DURATION

Parameters

out	*x*	`opus_int32 *`: Returns one of the following values: **OPUS_FRAMESIZE_ARG** Select frame size from the argument (default). **OPUS_FRAMESIZE_2_5_MS** Use 2.5 ms frames. **OPUS_FRAMESIZE_5_MS** Use 5 ms frames. **OPUS_FRAMESIZE_10_MS** Use 10 ms frames. **OPUS_FRAMESIZE_20_MS** Use 20 ms frames. **OPUS_FRAMESIZE_40_MS** Use 40 ms frames. **OPUS_FRAMESIZE_60_MS** Use 60 ms frames. **OPUS_FRAMESIZE_80_MS** Use 80 ms frames. **OPUS_FRAMESIZE_100_MS** Use 100 ms frames. **OPUS_FRAMESIZE_120_MS** Use 120 ms frames.

4.6.2.7 OPUS_GET_FORCE_CHANNELS

```
#define OPUS_GET_FORCE_CHANNELS(
              x )
```

Gets the encoder's forced channel configuration.

See also

OPUS_SET_FORCE_CHANNELS

Parameters

out	*x*	`opus_int32 *`: **OPUS_AUTO** Not forced (default) **1** Forced mono **2** Forced stereo

4.6.2.8 OPUS_GET_INBAND_FEC

```
#define OPUS_GET_INBAND_FEC(
              x )
```

Gets encoder's configured use of inband forward error correction.

See also

OPUS_SET_INBAND_FEC

Parameters

out	*x*	`opus_int32 *`: Returns one of the following values: **0** Inband FEC disabled (default). **1** Inband FEC enabled. If the packet loss rate is sufficiently high, Opus will automatically switch to SILK even at high rates to enable use of that FEC. **2** Inband FEC enabled, but does not necessarily switch to SILK if we have music.

4.6.2.9 OPUS_GET_LOOKAHEAD

```
#define OPUS_GET_LOOKAHEAD(
              x )
```

Gets the total samples of delay added by the entire codec.

This can be queried by the encoder and then the provided number of samples can be skipped on from the start of the decoder's output to provide time aligned input and output. From the perspective of a decoding application the real data begins this many samples late.

The decoder contribution to this delay is identical for all decoders, but the encoder portion of the delay may vary from implementation to implementation, version to version, or even depend on the encoder's initial configuration. Applications needing delay compensation should call this CTL rather than hard-coding a value.

Parameters

out	*x*	`opus_int32 *`: Number of lookahead samples

4.6.2.10 OPUS_GET_LSB_DEPTH

```
#define OPUS_GET_LSB_DEPTH(
              x )
```

Gets the encoder's configured signal depth.

See also

OPUS_SET_LSB_DEPTH

Parameters

out	*x*	opus_int32 *: Input precision in bits, between 8 and 24 (default: 24).

4.6.2.11 OPUS_GET_MAX_BANDWIDTH

```
#define OPUS_GET_MAX_BANDWIDTH(
              x )
```

Gets the encoder's configured maximum allowed bandpass.

See also

OPUS_SET_MAX_BANDWIDTH

Parameters

out	*x*	opus_int32 *: Allowed values: **OPUS_BANDWIDTH_NARROWBAND** 4 kHz passband **OPUS_BANDWIDTH_MEDIUMBAND** 6 kHz passband **OPUS_BANDWIDTH_WIDEBAND** 8 kHz passband **OPUS_BANDWIDTH_SUPERWIDEBAND** 12 kHz passband **OPUS_BANDWIDTH_FULLBAND** 20 kHz passband (default)

4.6.2.12 OPUS_GET_PACKET_LOSS_PERC

```
#define OPUS_GET_PACKET_LOSS_PERC(
              x )
```

Gets the encoder's configured packet loss percentage.

See also

OPUS_SET_PACKET_LOSS_PERC

Parameters

out	*x*	opus_int32 *: Returns the configured loss percentage in the range 0-100, inclusive (default: 0).

4.6.2.13 OPUS_GET_PREDICTION_DISABLED

```
#define OPUS_GET_PREDICTION_DISABLED(
              x )
```

Gets the encoder's configured prediction status.

See also

OPUS_SET_PREDICTION_DISABLED

Parameters

out	*x*	opus_int32 *: Returns one of the following values: **0** Prediction enabled (default). **1** Prediction disabled.

4.6.2.14 OPUS_GET_QEXT

```
#define OPUS_GET_QEXT(
              x )
```

Gets the encoder's configured quality extension (QEXT).

4.6.2.15 OPUS_GET_SIGNAL

```
#define OPUS_GET_SIGNAL(
              x )
```

Gets the encoder's configured signal type.

See also

OPUS_SET_SIGNAL

Parameters

out	*x*	opus_int32 *: Returns one of the following values: **OPUS_AUTO** (default) **OPUS_SIGNAL_VOICE** Bias thresholds towards choosing LPC or Hybrid modes. **OPUS_SIGNAL_MUSIC** Bias thresholds towards choosing MDCT modes.

4.6.2.16 OPUS_GET_VBR

```
#define OPUS_GET_VBR(
              x )
```

Determine if variable bitrate (VBR) is enabled in the encoder.

See also

OPUS_SET_VBR

OPUS_GET_VBR_CONSTRAINT

Parameters

out	*x*	opus_int32 *: Returns one of the following values: **0** Hard CBR. **1** VBR (default). The exact type of VBR may be retrieved via OPUS_GET_VBR_CONSTRAINT.

4.6.2.17 OPUS_GET_VBR_CONSTRAINT

```
#define OPUS_GET_VBR_CONSTRAINT(
              x )
```

Determine if constrained VBR is enabled in the encoder.

See also

OPUS_SET_VBR_CONSTRAINT

OPUS_GET_VBR

Parameters

out	*x*	opus_int32 *: Returns one of the following values: **0** Unconstrained VBR. **1** Constrained VBR (default).

4.6.2.18 OPUS_SET_APPLICATION

```
#define OPUS_SET_APPLICATION(
              x )
```

Configures the encoder's intended application.

The initial value is a mandatory argument to the encoder_create function.

See also

OPUS_GET_APPLICATION

Parameters

in	*x*	`opus_int32`: Returns one of the following values: **OPUS_APPLICATION_VOIP** Process signal for improved speech intelligibility. **OPUS_APPLICATION_AUDIO** Favor faithfulness to the original input. **OPUS_APPLICATION_RESTRICTED_LOWDELAY** Configure the minimum possible coding delay by disabling certain modes of operation.

4.6.2.19 OPUS_SET_BANDWIDTH

```
#define OPUS_SET_BANDWIDTH(
              x )
```

Sets the encoder's bandpass to a specific value.

This prevents the encoder from automatically selecting the bandpass based on the available bitrate. If an application knows the bandpass of the input audio it is providing, it should normally use OPUS_SET_MAX_BANDWIDTH instead, which still gives the encoder the freedom to reduce the bandpass when the bitrate becomes too low, for better overall quality.

See also

OPUS_GET_BANDWIDTH

Parameters

in	*x*	`opus_int32`: Allowed values: **OPUS_AUTO** (default) **OPUS_BANDWIDTH_NARROWBAND** 4 kHz passband **OPUS_BANDWIDTH_MEDIUMBAND** 6 kHz passband **OPUS_BANDWIDTH_WIDEBAND** 8 kHz passband **OPUS_BANDWIDTH_SUPERWIDEBAND** 12 kHz passband **OPUS_BANDWIDTH_FULLBAND** 20 kHz passband

4.6.2.20 OPUS_SET_BITRATE

```
#define OPUS_SET_BITRATE(
              x )
```

Configures the bitrate in the encoder.

Rates from 500 to 512000 bits per second are meaningful, as well as the special values OPUS_AUTO and OPUS_BITRATE_MAX. The value OPUS_BITRATE_MAX can be used to cause the codec to use as much rate as it can, which is useful for controlling the rate by adjusting the output buffer size.

See also

OPUS_GET_BITRATE

Parameters

in	*x*	opus_int32: Bitrate in bits per second. The default is determined based on the number of channels and the input sampling rate.

4.6.2.21 OPUS_SET_COMPLEXITY

```
#define OPUS_SET_COMPLEXITY(
              x )
```

Configures the encoder's computational complexity.

The supported range is 0-10 inclusive with 10 representing the highest complexity.

See also

OPUS_GET_COMPLEXITY

Parameters

in	*x*	opus_int32: Allowed values: 0-10, inclusive.

4.6.2.22 OPUS_SET_DNN_BLOB

```
#define OPUS_SET_DNN_BLOB(
              data,
              len )
```

Provide external DNN weights from binary object (only when explicitly built without the weights)

4.6.2.23 OPUS_SET_DRED_DURATION

```
#define OPUS_SET_DRED_DURATION(
              x )
```

If non-zero, enables Deep Redundancy (DRED) and use the specified maximum number of 10-ms redundant frames.

4.6.2.24 OPUS_SET_DTX

```
#define OPUS_SET_DTX(
              x )
```

Configures the encoder's use of discontinuous transmission (DTX).

Note

This is only applicable to the LPC layer

See also

OPUS_GET_DTX

Parameters

in	*x*	opus_int32: Allowed values: **0** Disable DTX (default). **1** Enabled DTX.

4.6.2.25 OPUS_SET_EXPERT_FRAME_DURATION

```
#define OPUS_SET_EXPERT_FRAME_DURATION(
              x )
```

Configures the encoder's use of variable duration frames.

When variable duration is enabled, the encoder is free to use a shorter frame size than the one requested in the opus↩ _encode∗() call. It is then the user's responsibility to verify how much audio was encoded by checking the ToC byte of the encoded packet. The part of the audio that was not encoded needs to be resent to the encoder for the next call. Do not use this option unless you **really** know what you are doing.

See also

OPUS_GET_EXPERT_FRAME_DURATION

Parameters

in	*x*	opus_int32: Allowed values: **OPUS_FRAMESIZE_ARG** Select frame size from the argument (default). **OPUS_FRAMESIZE_2_5_MS** Use 2.5 ms frames. **OPUS_FRAMESIZE_5_MS** Use 5 ms frames. **OPUS_FRAMESIZE_10_MS** Use 10 ms frames. **OPUS_FRAMESIZE_20_MS** Use 20 ms frames. **OPUS_FRAMESIZE_40_MS** Use 40 ms frames. **OPUS_FRAMESIZE_60_MS** Use 60 ms frames. **OPUS_FRAMESIZE_80_MS** Use 80 ms frames. **OPUS_FRAMESIZE_100_MS** Use 100 ms frames. **OPUS_FRAMESIZE_120_MS** Use 120 ms frames.

4.6.2.26 OPUS_SET_FORCE_CHANNELS

```
#define OPUS_SET_FORCE_CHANNELS(
              x )
```

Configures mono/stereo forcing in the encoder.

This can force the encoder to produce packets encoded as either mono or stereo, regardless of the format of the input audio. This is useful when the caller knows that the input signal is currently a mono source embedded in a stereo stream.

See also

OPUS_GET_FORCE_CHANNELS

Parameters

in	*x*	opus_int32: Allowed values: **OPUS_AUTO** Not forced (default) **1** Forced mono **2** Forced stereo

4.6.2.27 OPUS_SET_INBAND_FEC

```
#define OPUS_SET_INBAND_FEC(
              x )
```

Configures the encoder's use of inband forward error correction (FEC).

Note

This is only applicable to the LPC layer

See also

OPUS_GET_INBAND_FEC

Parameters

in	*x*	`opus_int32`: Allowed values: **0** Disable inband FEC (default). **1** Inband FEC enabled. If the packet loss rate is sufficiently high, Opus will automatically switch to SILK even at high rates to enable use of that FEC. **2** Inband FEC enabled, but does not necessarily switch to SILK if we have music.

4.6.2.28 OPUS_SET_LSB_DEPTH

```
#define OPUS_SET_LSB_DEPTH(
              x )
```

Configures the depth of signal being encoded.

This is a hint which helps the encoder identify silence and near-silence. It represents the number of significant bits of linear intensity below which the signal contains ignorable quantization or other noise.

For example, OPUS_SET_LSB_DEPTH(14) would be an appropriate setting for G.711 u-law input. OPUS_SET_LSB_DEPTH(16) would be appropriate for 16-bit linear pcm input with opus_encode_float().

When using opus_encode() instead of opus_encode_float(), or when libopus is compiled for fixed-point, the encoder uses the minimum of the value set here and the value 16.

See also

OPUS_GET_LSB_DEPTH

Parameters

in	*x*	`opus_int32`: Input precision in bits, between 8 and 24 (default: 24).

4.6.2.29 OPUS_SET_MAX_BANDWIDTH

```
#define OPUS_SET_MAX_BANDWIDTH(
              x )
```

Configures the maximum bandpass that the encoder will select automatically.

Applications should normally use this instead of OPUS_SET_BANDWIDTH (leaving that set to the default, OPUS_AUTO). This allows the application to set an upper bound based on the type of input it is providing, but still gives the encoder the freedom to reduce the bandpass when the bitrate becomes too low, for better overall quality.

See also

OPUS_GET_MAX_BANDWIDTH

Parameters

in	*x*	`opus_int32`: Allowed values: **OPUS_BANDWIDTH_NARROWBAND** 4 kHz passband **OPUS_BANDWIDTH_MEDIUMBAND** 6 kHz passband **OPUS_BANDWIDTH_WIDEBAND** 8 kHz passband **OPUS_BANDWIDTH_SUPERWIDEBAND** 12 kHz passband **OPUS_BANDWIDTH_FULLBAND** 20 kHz passband (default)

4.6.2.30 OPUS_SET_PACKET_LOSS_PERC

```
#define OPUS_SET_PACKET_LOSS_PERC(
              x )
```

Configures the encoder's expected packet loss percentage.

Higher values trigger progressively more loss resistant behavior in the encoder at the expense of quality at a given bitrate in the absence of packet loss, but greater quality under loss.

See also

OPUS_GET_PACKET_LOSS_PERC

Parameters

in	*x*	opus_int32: Loss percentage in the range 0-100, inclusive (default: 0).

4.6.2.31 OPUS_SET_PREDICTION_DISABLED

```
#define OPUS_SET_PREDICTION_DISABLED(
              x )
```

If set to 1, disables almost all use of prediction, making frames almost completely independent.

This reduces quality.

See also

OPUS_GET_PREDICTION_DISABLED

Parameters

in	*x*	opus_int32: Allowed values: **0** Enable prediction (default). **1** Disable prediction.

4.6.2.32 OPUS_SET_QEXT

```
#define OPUS_SET_QEXT(
              x )
```

If set to 1, enables quality extension (QEXT), otherwise disables it (default).

Warning: This will *hurt* audio quality unless operating at a very high bitrate.

4.6.2.33 OPUS_SET_SIGNAL

```
#define OPUS_SET_SIGNAL(
              x )
```

Configures the type of signal being encoded.

This is a hint which helps the encoder's mode selection.

See also

OPUS_GET_SIGNAL

Parameters

in	*x*	opus_int32: Allowed values: **OPUS_AUTO** (default) **OPUS_SIGNAL_VOICE** Bias thresholds towards choosing LPC or Hybrid modes. **OPUS_SIGNAL_MUSIC** Bias thresholds towards choosing MDCT modes.

4.6.2.34 OPUS_SET_VBR

```
#define OPUS_SET_VBR(
              x )
```

Enables or disables variable bitrate (VBR) in the encoder.

The configured bitrate may not be met exactly because frames must be an integer number of bytes in length.

See also

OPUS_GET_VBR
OPUS_SET_VBR_CONSTRAINT

Parameters

in	*x*	opus_int32: Allowed values: **0** Hard CBR. For LPC/hybrid modes at very low bit-rate, this can cause noticeable quality degradation. **1** VBR (default). The exact type of VBR is controlled by OPUS_SET_VBR_CONSTRAINT.

4.6.2.35 OPUS_SET_VBR_CONSTRAINT

```
#define OPUS_SET_VBR_CONSTRAINT(
              x )
```

Enables or disables constrained VBR in the encoder.

This setting is ignored when the encoder is in CBR mode.

Warning

Only the MDCT mode of Opus currently heeds the constraint. Speech mode ignores it completely, hybrid mode may fail to obey it if the LPC layer uses more bitrate than the constraint would have permitted.

See also

OPUS_GET_VBR_CONSTRAINT

OPUS_SET_VBR

Parameters

in	*x*	opus_int32: Allowed values: **0** Unconstrained VBR. **1** Constrained VBR (default). This creates a maximum of one frame of buffering delay assuming a transport with a serialization speed of the nominal bitrate.

4.7 Generic CTLs

These macros are used with the opus_decoder_ctl and opus_encoder_ctl calls to generate a particular request.

Macros

- #define OPUS_RESET_STATE

  Resets the codec state to be equivalent to a freshly initialized state.
- #define OPUS_GET_FINAL_RANGE(x)

  Gets the final state of the codec's entropy coder.
- #define OPUS_GET_BANDWIDTH(x)

  Gets the encoder's configured bandpass or the decoder's last bandpass.
- #define OPUS_GET_SAMPLE_RATE(x)

  Gets the sampling rate the encoder or decoder was initialized with.
- #define OPUS_SET_PHASE_INVERSION_DISABLED(x)

  If set to 1, disables the use of phase inversion for intensity stereo, improving the quality of mono downmixes, but slightly reducing normal stereo quality.
- #define OPUS_GET_PHASE_INVERSION_DISABLED(x)

  Gets the encoder's configured phase inversion status.
- #define OPUS_GET_IN_DTX(x)

  Gets the DTX state of the encoder.

4.7.1 Detailed Description

These macros are used with the opus_decoder_ctl and opus_encoder_ctl calls to generate a particular request.

When called on an OpusDecoder they apply to that particular decoder instance. When called on an OpusEncoder they apply to the corresponding setting on that encoder instance, if present.

Some usage examples:

```
int ret;
opus_int32 pitch;
ret = opus_decoder_ctl(dec_ctx, OPUS_GET_PITCH(&pitch));
if (ret == OPUS_OK) return ret;

opus_encoder_ctl(enc_ctx, OPUS_RESET_STATE);
opus_decoder_ctl(dec_ctx, OPUS_RESET_STATE);

opus_int32 enc_bw, dec_bw;
opus_encoder_ctl(enc_ctx, OPUS_GET_BANDWIDTH(&enc_bw));
opus_decoder_ctl(dec_ctx, OPUS_GET_BANDWIDTH(&dec_bw));
if (enc_bw != dec_bw) {
  printf("packet bandwidth mismatch!\n");
}
```

See also

Opus Encoder, opus_decoder_ctl, opus_encoder_ctl, Decoder related CTLs, Encoder related CTLs

4.7.2 Macro Definition Documentation

4.7.2.1 OPUS_GET_BANDWIDTH

```
#define OPUS_GET_BANDWIDTH(
              x )
```

Gets the encoder's configured bandpass or the decoder's last bandpass.

See also

OPUS_SET_BANDWIDTH

Parameters

out	*x*	opus_int32 *: Returns one of the following values: **OPUS_AUTO** (default) **OPUS_BANDWIDTH_NARROWBAND** 4 kHz passband **OPUS_BANDWIDTH_MEDIUMBAND** 6 kHz passband **OPUS_BANDWIDTH_WIDEBAND** 8 kHz passband **OPUS_BANDWIDTH_SUPERWIDEBAND** 12 kHz passband **OPUS_BANDWIDTH_FULLBAND** 20 kHz passband

4.7.2.2 OPUS_GET_FINAL_RANGE

```
#define OPUS_GET_FINAL_RANGE(
              x )
```

Gets the final state of the codec's entropy coder.

This is used for testing purposes, The encoder and decoder state should be identical after coding a payload (assuming no data corruption or software bugs)

Parameters

out	*x*	opus_uint32 *: Entropy coder state

4.7.2.3 OPUS_GET_IN_DTX

```
#define OPUS_GET_IN_DTX(
              x )
```

Gets the DTX state of the encoder.

Returns whether the last encoded frame was either a comfort noise update during DTX or not encoded because of DTX.

Parameters

out	*x*	`opus_int32 *`: Returns one of the following values: **0** The encoder is not in DTX. **1** The encoder is in DTX.

4.7.2.4 OPUS_GET_PHASE_INVERSION_DISABLED

```
#define OPUS_GET_PHASE_INVERSION_DISABLED(
              x )
```

Gets the encoder's configured phase inversion status.

See also

OPUS_SET_PHASE_INVERSION_DISABLED

Parameters

out	*x*	`opus_int32 *`: Returns one of the following values: **0** Stereo phase inversion enabled (default). **1** Stereo phase inversion disabled.

4.7.2.5 OPUS_GET_SAMPLE_RATE

```
#define OPUS_GET_SAMPLE_RATE(
              x )
```

Gets the sampling rate the encoder or decoder was initialized with.

This simply returns the `Fs` value passed to opus_encoder_init() or opus_decoder_init().

Parameters

out	*x*	opus_int32 *: Sampling rate of encoder or decoder.

4.7.2.6 OPUS_RESET_STATE

```
#define OPUS_RESET_STATE
```

Resets the codec state to be equivalent to a freshly initialized state.

This should be called when switching streams in order to prevent the back to back decoding from giving different results from one at a time decoding.

4.7.2.7 OPUS_SET_PHASE_INVERSION_DISABLED

```
#define OPUS_SET_PHASE_INVERSION_DISABLED(
              x )
```

If set to 1, disables the use of phase inversion for intensity stereo, improving the quality of mono downmixes, but slightly reducing normal stereo quality.

Disabling phase inversion in the decoder does not comply with RFC 6716, although it does not cause any interoperability issue and is expected to become part of the Opus standard once RFC 6716 is updated by draft-ietf-codec-opus-update.

See also

OPUS_GET_PHASE_INVERSION_DISABLED

Parameters

in	*x*	opus_int32: Allowed values: **0** Enable phase inversion (default). **1** Disable phase inversion.

4.8 Decoder related CTLs

Macros

- #define OPUS_SET_GAIN(x)

  Configures decoder gain adjustment.
- #define OPUS_GET_GAIN(x)

Gets the decoder's configured gain adjustment.

- #define OPUS_GET_LAST_PACKET_DURATION(x)

  Gets the duration (in samples) of the last packet successfully decoded or concealed.
- #define OPUS_GET_PITCH(x)

  Gets the pitch of the last decoded frame, if available.
- #define OPUS_SET_OSCE_BWE(x)

  Enables blind bandwidth extension for wideband signals if decoding sampling rate is 48 kHz.
- #define OPUS_GET_OSCE_BWE(x)

  Gets blind bandwidth extension flag for wideband signals if decoding sampling rate is 48 kHz.
- #define OPUS_SET_IGNORE_EXTENSIONS(x)

  If set to 1, the decoder will ignore all extensions found in the padding area (does not affect DRED, which is decoded separately).
- #define OPUS_GET_IGNORE_EXTENSIONS(x)

  Gets whether the decoder is ignoring extensions.

4.8.1 Detailed Description

See also

Generic CTLs, Encoder related CTLs, Opus Decoder

4.8.2 Macro Definition Documentation

4.8.2.1 OPUS_GET_GAIN

```
#define OPUS_GET_GAIN(
              x )
```

Gets the decoder's configured gain adjustment.

See also

OPUS_SET_GAIN

Parameters

out	*x*	opus_int32 *: Amount to scale PCM signal by in Q8 dB units.

4.8.2.2 OPUS_GET_IGNORE_EXTENSIONS

```
#define OPUS_GET_IGNORE_EXTENSIONS(
              x )
```

Gets whether the decoder is ignoring extensions.

4.8.2.3 OPUS_GET_LAST_PACKET_DURATION

```
#define OPUS_GET_LAST_PACKET_DURATION(
              x )
```

Gets the duration (in samples) of the last packet successfully decoded or concealed.

Parameters

out	*x*	opus_int32 *: Number of samples (at current sampling rate).

4.8.2.4 OPUS_GET_OSCE_BWE

```
#define OPUS_GET_OSCE_BWE(
              x )
```

Gets blind bandwidth extension flag for wideband signals if decoding sampling rate is 48 kHz.

Parameters

out	*x*	opus_int32 *: 1 if bwe enabled, 0 if disabled.

4.8.2.5 OPUS_GET_PITCH

```
#define OPUS_GET_PITCH(
              x )
```

Gets the pitch of the last decoded frame, if available.

This can be used for any post-processing algorithm requiring the use of pitch, e.g. time stretching/shortening. If the last frame was not voiced, or if the pitch was not coded in the frame, then zero is returned.

This CTL is only implemented for decoder instances.

Parameters

out	*x*	opus_int32 *: pitch period at 48 kHz (or 0 if not available)

4.8.2.6 OPUS_SET_GAIN

```
#define OPUS_SET_GAIN(
              x )
```

Configures decoder gain adjustment.

Scales the decoded output by a factor specified in Q8 dB units. This has a maximum range of -32768 to 32767 inclusive, and returns OPUS_BAD_ARG otherwise. The default is zero indicating no adjustment. This setting survives decoder reset.

gain = pow(10, x/(20.0∗256))

Parameters

in	*x*	opus_int32: Amount to scale PCM signal by in Q8 dB units.

4.8.2.7 OPUS_SET_IGNORE_EXTENSIONS

```
#define OPUS_SET_IGNORE_EXTENSIONS(
              x )
```

If set to 1, the decoder will ignore all extensions found in the padding area (does not affect DRED, which is decoded separately).

4.8.2.8 OPUS_SET_OSCE_BWE

```
#define OPUS_SET_OSCE_BWE(
              x )
```

Enables blind bandwidth extension for wideband signals if decoding sampling rate is 48 kHz.

Parameters

in	*x*	opus_int32 : 1 enables bandwidth extension, 0 disables it. The default is 0.

4.9 Opus library information functions

Functions

- const char ∗ opus_strerror (int error)

  Converts an opus error code into a human readable string.
- const char ∗ opus_get_version_string (void)

  Gets the libopus version string.

4.9.1 Detailed Description

4.9.2 Function Documentation

4.9.2.1 opus_get_version_string()

```
const char * opus_get_version_string (
            void  )
```

Gets the libopus version string.

Applications may look for the substring "-fixed" in the version string to determine whether they have a fixed-point or floating-point build at runtime.

Returns

Version string

4.9.2.2 opus_strerror()

```
const char * opus_strerror (
            int error )
```

Converts an opus error code into a human readable string.

Parameters

in	*error*	int: Error number

Returns

Error string

4.10 Multistream specific encoder and decoder CTLs

These are convenience macros that are specific to the opus_multistream_encoder_ctl() and opus_multistream_decoder_ctl() interface.

Macros

- #define OPUS_MULTISTREAM_GET_ENCODER_STATE(x, y)

  Gets the encoder state for an individual stream of a multistream encoder.
- #define OPUS_MULTISTREAM_GET_DECODER_STATE(x, y)

  Gets the decoder state for an individual stream of a multistream decoder.

4.10.1 Detailed Description

These are convenience macros that are specific to the opus_multistream_encoder_ctl() and opus_multistream_decoder_ctl() interface.

The CTLs from Generic CTLs, Encoder related CTLs, and Decoder related CTLs may be applied to a multistream encoder or decoder as well. In addition, you may retrieve the encoder or decoder state for an specific stream via OPUS_MULTISTREAM_GET_ENCODER_STATE or OPUS_MULTISTREAM_GET_DECODER_STATE and apply CTLs to it individually.

4.10.2 Macro Definition Documentation

4.10.2.1 OPUS_MULTISTREAM_GET_DECODER_STATE

```
#define OPUS_MULTISTREAM_GET_DECODER_STATE(
              x,
              y )
```

Gets the decoder state for an individual stream of a multistream decoder.

Parameters

in	*x*	`opus_int32`: The index of the stream whose decoder you wish to retrieve. This must be non-negative and less than the `streams` parameter used to initialize the decoder.
out	*y*	`OpusDecoder**`: Returns a pointer to the given decoder state.

Return values

OPUS_BAD_ARG	The index of the requested stream was out of range.

4.10.2.2 OPUS_MULTISTREAM_GET_ENCODER_STATE

```
#define OPUS_MULTISTREAM_GET_ENCODER_STATE(
              x,
              y )
```

Gets the encoder state for an individual stream of a multistream encoder.

Parameters

in	*x*	`opus_int32`: The index of the stream whose encoder you wish to retrieve. This must be non-negative and less than the `streams` parameter used to initialize the encoder.
out	*y*	`OpusEncoder**`: Returns a pointer to the given encoder state.

Return values

OPUS_BAD_ARG	The index of the requested stream was out of range.

4.11 Opus Multistream API

The multistream API allows individual Opus streams to be combined into a single packet, enabling support for up to 255 channels.

Typedefs

- typedef struct OpusMSEncoder OpusMSEncoder

  Opus multistream encoder state.
- typedef struct OpusMSDecoder OpusMSDecoder

  Opus multistream decoder state.

Multistream encoder functions

- opus_int32 opus_multistream_encoder_get_size (int streams, int coupled_streams)

  Gets the size of an OpusMSEncoder structure.
- opus_int32 opus_multistream_surround_encoder_get_size (int channels, int mapping_family)
- OpusMSEncoder * opus_multistream_encoder_create (opus_int32 Fs, int channels, int streams, int coupled_↩ streams, const unsigned char *mapping, int application, int *error)

  Allocates and initializes a multistream encoder state.
- OpusMSEncoder * opus_multistream_surround_encoder_create (opus_int32 Fs, int channels, int mapping_↩ family, int *streams, int *coupled_streams, unsigned char *mapping, int application, int *error)
- int opus_multistream_encoder_init (OpusMSEncoder *st, opus_int32 Fs, int channels, int streams, int coupled↩ _streams, const unsigned char *mapping, int application)

  Initialize a previously allocated multistream encoder state.
- int opus_multistream_surround_encoder_init (OpusMSEncoder *st, opus_int32 Fs, int channels, int mapping_↩ family, int *streams, int *coupled_streams, unsigned char *mapping, int application)
- int opus_multistream_encode (OpusMSEncoder *st, const opus_int16 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes a multistream Opus frame.
- int opus_multistream_encode24 (OpusMSEncoder *st, const opus_int32 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes a multistream Opus frame.
- int opus_multistream_encode_float (OpusMSEncoder *st, const float *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes a multistream Opus frame from floating point input.
- void opus_multistream_encoder_destroy (OpusMSEncoder *st)

  Frees an `OpusMSEncoder` allocated by opus_multistream_encoder_create().
- int opus_multistream_encoder_ctl (OpusMSEncoder *st, int request,...)

  Perform a CTL function on a multistream Opus encoder.

Multistream decoder functions

- opus_int32 opus_multistream_decoder_get_size (int streams, int coupled_streams)

  Gets the size of an `OpusMSDecoder` structure.
- OpusMSDecoder * opus_multistream_decoder_create (opus_int32 Fs, int channels, int streams, int coupled_↩ streams, const unsigned char *mapping, int *error)

  Allocates and initializes a multistream decoder state.
- int opus_multistream_decoder_init (OpusMSDecoder *st, opus_int32 Fs, int channels, int streams, int coupled↩ _streams, const unsigned char *mapping)

  Initialize a previously allocated decoder state object.
- int opus_multistream_decode (OpusMSDecoder *st, const unsigned char *data, opus_int32 len, opus_int16 *pcm, int frame_size, int decode_fec)

Decode a multistream Opus packet.

- int opus_multistream_decode24 (OpusMSDecoder *st, const unsigned char *data, opus_int32 len, opus_int32 *pcm, int frame_size, int decode_fec)

  Decode a multistream Opus packet.
- int opus_multistream_decode_float (OpusMSDecoder *st, const unsigned char *data, opus_int32 len, float *pcm, int frame_size, int decode_fec)

  Decode a multistream Opus packet with floating point output.
- int opus_multistream_decoder_ctl (OpusMSDecoder *st, int request,...)

  Perform a CTL function on a multistream Opus decoder.
- void opus_multistream_decoder_destroy (OpusMSDecoder *st)

  Frees an `OpusMSDecoder` allocated by opus_multistream_decoder_create().

4.11.1 Detailed Description

The multistream API allows individual Opus streams to be combined into a single packet, enabling support for up to 255 channels.

Unlike an elementary Opus stream, the encoder and decoder must negotiate the channel configuration before the decoder can successfully interpret the data in the packets produced by the encoder. Some basic information, such as packet duration, can be computed without any special negotiation.

The format for multistream Opus packets is defined in RFC 7845 and is based on the self-delimited Opus framing described in Appendix B of RFC 6716. Normal Opus packets are just a degenerate case of multistream Opus packets, and can be encoded or decoded with the multistream API by setting `streams` to `1` when initializing the encoder or decoder.

Multistream Opus streams can contain up to 255 elementary Opus streams. These may be either "uncoupled" or "coupled", indicating that the decoder is configured to decode them to either 1 or 2 channels, respectively. The streams are ordered so that all coupled streams appear at the beginning.

A `mapping` table defines which decoded channel `i` should be used for each input/output (I/O) channel `j`. This table is typically provided as an unsigned char array. Let `i = mapping[j]` be the index for I/O channel `j`. If `i < 2*coupled_streams`, then I/O channel `j` is encoded as the left channel of stream `(i/2)` if `i` is even, or as the right channel of stream `(i/2)` if `i` is odd. Otherwise, I/O channel `j` is encoded as mono in stream `(i - coupled_streams)`, unless it has the special value 255, in which case it is omitted from the encoding entirely (the decoder will reproduce it as silence). Each value `i` must either be the special value 255 or be less than `streams + coupled_streams`.

The output channels specified by the encoder should use the Vorbis channel ordering. A decoder may wish to apply an additional permutation to the mapping the encoder used to achieve a different output channel order (e.g. for outputting in WAV order).

Each multistream packet contains an Opus packet for each stream, and all of the Opus packets in a single multistream packet must have the same duration. Therefore the duration of a multistream packet can be extracted from the TOC sequence of the first stream, which is located at the beginning of the packet, just like an elementary Opus stream:

```
int nb_samples;
int nb_frames;
nb_frames = opus_packet_get_nb_frames(data, len);
if (nb_frames < 1)
  return nb_frames;
nb_samples = opus_packet_get_samples_per_frame(data, 48000) * nb_frames;
```

The general encoding and decoding process proceeds exactly the same as in the normal Opus Encoder and Opus Decoder APIs. See their documentation for an overview of how to use the corresponding multistream functions.

4.11.2 Typedef Documentation

4.11.2.1 OpusMSDecoder

```
typedef struct OpusMSDecoder OpusMSDecoder
```

Opus multistream decoder state.

This contains the complete state of a multistream Opus decoder. It is position independent and can be freely copied.

See also

opus_multistream_decoder_create

opus_multistream_decoder_init

4.11.2.2 OpusMSEncoder

```
typedef struct OpusMSEncoder OpusMSEncoder
```

Opus multistream encoder state.

This contains the complete state of a multistream Opus encoder. It is position independent and can be freely copied.

See also

opus_multistream_encoder_create

opus_multistream_encoder_init

4.11.3 Function Documentation

4.11.3.1 opus_multistream_decode()

```
int opus_multistream_decode (
            OpusMSDecoder * st,
            const unsigned char * data,
            opus_int32 len,
            opus_int16 * pcm,
            int frame_size,
            int decode_fec )
```

Decode a multistream Opus packet.

Parameters

	st	`OpusMSDecoder*`: Multistream decoder state.
in	*data*	`const unsigned char*`: Input payload. Use a `NULL` pointer to indicate packet loss.
	len	`opus_int32`: Number of bytes in payload.
out	*pcm*	`opus_int16*`: Output signal, with interleaved samples. This must contain room for `frame_size*channels` samples.
	frame_size	`int`: The number of samples per channel of available space in *pcm*. If this is less than the maximum packet duration (120 ms; 5760 for 48kHz), this function will not be capable of

Returns

Number of samples decoded on success or a negative error code (see Error codes) on failure.

4.11.3.2 opus_multistream_decode24()

```
int opus_multistream_decode24 (
              OpusMSDecoder * st,
              const unsigned char * data,
              opus_int32 len,
              opus_int32 * pcm,
              int frame_size,
              int decode_fec )
```

Decode a multistream Opus packet.

Parameters

	st	`OpusMSDecoder*`: Multistream decoder state.
in	*data*	`const unsigned char*`: Input payload. Use a `NULL` pointer to indicate packet loss.
	len	`opus_int32`: Number of bytes in payload.
out	*pcm*	`opus_int32*`: Output signal, with interleaved samples representing (or slightly exceeding) 24-bit values. This must contain room for `frame_size*channels` samples.
	frame_size	`int`: The number of samples per channel of available space in *pcm*. If this is less than the maximum packet duration (120 ms; 5760 for 48kHz), this function will not be capable of decoding some packets. In the case of PLC (data==NULL) or FEC (decode_fec=1), then frame_size needs to be exactly the duration of audio that is missing, otherwise the decoder will not be in the optimal state to decode the next incoming packet. For the PLC and FEC cases, frame_size **must** be a multiple of 2.5 ms.
	decode_fec	`int`: Flag (0 or 1) to request that any in-band forward error correction data be decoded. If no such data is available, the frame is decoded as if it were lost.

Returns

Number of samples decoded on success or a negative error code (see Error codes) on failure.

4.11.3.3 opus_multistream_decode_float()

```
int opus_multistream_decode_float (
              OpusMSDecoder * st,
              const unsigned char * data,
              opus_int32 len,
              float * pcm,
              int frame_size,
              int decode_fec )
```

Decode a multistream Opus packet with floating point output.

Parameters

	st	`OpusMSDecoder*`: Multistream decoder state.
in	*data*	`const unsigned char*`: Input payload. Use a `NULL` pointer to indicate packet loss.
	len	`opus_int32`: Number of bytes in payload.
out	*pcm*	`opus_int16*`: Output signal, with interleaved samples. This must contain room for `frame_size*channels` samples.
	frame_size	`int`: The number of samples per channel of available space in *pcm*. If this is less than the maximum packet duration (120 ms; 5760 for 48kHz), this function will not be capable of decoding some packets. In the case of PLC (data==NULL) or FEC (decode_fec=1), then frame_size needs to be exactly the duration of audio that is missing, otherwise the decoder will not be in the optimal state to decode the next incoming packet. For the PLC and FEC cases, frame_size **must** be a multiple of 2.5 ms.
	decode_fec	`int`: Flag (0 or 1) to request that any in-band forward error correction data be decoded. If no such data is available, the frame is decoded as if it were lost.

Returns

Number of samples decoded on success or a negative error code (see Error codes) on failure.

4.11.3.4 opus_multistream_decoder_create()

```
OpusMSDecoder * opus_multistream_decoder_create (
             opus_int32 Fs,
             int channels,
             int streams,
             int coupled_streams,
             const unsigned char * mapping,
             int * error )
```

Allocates and initializes a multistream decoder state.

Call opus_multistream_decoder_destroy() to release this object when finished.

Parameters

	Fs	`opus_int32`: Sampling rate to decode at (in Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
	channels	`int`: Number of channels to output. This must be at most 255. It may be different from the number of coded channels (`streams + coupled_streams`).
	streams	`int`: The total number of streams coded in the input. This must be no more than 255.
	coupled_streams	`int`: Number of streams to decode as coupled (2 channel) streams. This must be no larger than the total number of streams. Additionally, The total number of coded channels (`streams + coupled_streams`) must be no more than 255.
in	*mapping*	`const unsigned char[channels]`: Mapping from coded channels to output channels, as described in Opus Multistream API.
out	*error*	`int *`: Returns OPUS_OK on success, or an error code (see Error codes) on failure.

4.11.3.5 opus_multistream_decoder_ctl()

```
int opus_multistream_decoder_ctl (
              OpusMSDecoder * st,
              int request,
               ...  )
```

Perform a CTL function on a multistream Opus decoder.

Generally the request and subsequent arguments are generated by a convenience macro.

Parameters

st	`OpusMSDecoder*`: Multistream decoder state.
request	This and all remaining parameters should be replaced by one of the convenience macros in Generic CTLs, Decoder related CTLs, or Multistream specific encoder and decoder CTLs.

See also

Generic CTLs

Decoder related CTLs

Multistream specific encoder and decoder CTLs

4.11.3.6 opus_multistream_decoder_destroy()

```
void opus_multistream_decoder_destroy (
              OpusMSDecoder * st )
```

Frees an `OpusMSDecoder` allocated by opus_multistream_decoder_create().

Parameters

st	`OpusMSDecoder`: Multistream decoder state to be freed.

4.11.3.7 opus_multistream_decoder_get_size()

```
opus_int32 opus_multistream_decoder_get_size (
              int streams,
              int coupled_streams )
```

Gets the size of an `OpusMSDecoder` structure.

Parameters

streams	`int`: The total number of streams coded in the input. This must be no more than 255.

Parameters

coupled_streams	`int`: Number streams to decode as coupled (2 channel) streams. This must be no larger than the total number of streams. Additionally, The total number of coded channels (`streams + coupled_streams`) must be no more than 255.

Returns

The size in bytes on success, or a negative error code (see Error codes) on error.

4.11.3.8 opus_multistream_decoder_init()

```
int opus_multistream_decoder_init (
            OpusMSDecoder * st,
            opus_int32 Fs,
            int channels,
            int streams,
            int coupled_streams,
            const unsigned char * mapping )
```

Initialize a previously allocated decoder state object.

The memory pointed to by *st* must be at least the size returned by opus_multistream_encoder_get_size(). This is intended for applications which use their own allocator instead of malloc. To reset a previously initialized state, use the OPUS_RESET_STATE CTL.

See also

opus_multistream_decoder_create

opus_multistream_deocder_get_size

Parameters

	st	`OpusMSEncoder*`: Multistream encoder state to initialize.
	Fs	`opus_int32`: Sampling rate to decode at (in Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
	channels	`int`: Number of channels to output. This must be at most 255. It may be different from the number of coded channels (`streams + coupled_streams`).
	streams	`int`: The total number of streams coded in the input. This must be no more than 255.
	coupled_streams	`int`: Number of streams to decode as coupled (2 channel) streams. This must be no larger than the total number of streams. Additionally, The total number of coded channels (`streams + coupled_streams`) must be no more than 255.
in	*mapping*	`const unsigned char[channels]`: Mapping from coded channels to output channels, as described in Opus Multistream API.

Returns

OPUS_OK on success, or an error code (see Error codes) on failure.

4.11.3.9 opus_multistream_encode()

```
int opus_multistream_encode (
            OpusMSEncoder * st,
            const opus_int16 * pcm,
            int frame_size,
            unsigned char * data,
            opus_int32 max_data_bytes )
```

Encodes a multistream Opus frame.

Parameters

	st	`OpusMSEncoder*`: Multistream encoder state.
in	*pcm*	`const opus_int16*`: The input signal as interleaved samples. This must contain `frame_size*channels` samples.
	frame_size	`int`: Number of samples per channel in the input signal. This must be an Opus frame size for the encoder's sampling rate. For example, at 48 kHz the permitted values are 120, 240, 480, 960, 1920, and 2880. Passing in a duration of less than 10 ms (480 samples at 48 kHz) will prevent the encoder from using the LPC or hybrid modes.
out	*data*	`unsigned char*`: Output payload. This must contain storage for at least *max_data_bytes*.
in	*max_data_bytes*	`opus_int32`: Size of the allocated memory for the output payload. This may be used to impose an upper limit on the instant bitrate, but should not be used as the only bitrate control. Use OPUS_SET_BITRATE to control the bitrate.

Returns

The length of the encoded packet (in bytes) on success or a negative error code (see Error codes) on failure.

4.11.3.10 opus_multistream_encode24()

```
int opus_multistream_encode24 (
            OpusMSEncoder * st,
            const opus_int32 * pcm,
            int frame_size,
            unsigned char * data,
            opus_int32 max_data_bytes )
```

Encodes a multistream Opus frame.

Parameters

	st	`OpusMSEncoder*`: Multistream encoder state.

Parameters

in	*pcm*	`const opus_int32*`: The input signal as interleaved samples representing (or slightly exceeding) 24-bit values. This must contain `frame_size*channels` samples.
	frame_size	`int`: Number of samples per channel in the input signal. This must be an Opus frame size for the encoder's sampling rate. For example, at 48 kHz the permitted values are 120, 240, 480, 960, 1920, and 2880. Passing in a duration of less than 10 ms (480 samples at 48 kHz) will prevent the encoder from using the LPC or hybrid modes.
out	*data*	`unsigned char*`: Output payload. This must contain storage for at least *max_data_bytes*.
in	*max_data_bytes*	`opus_int32`: Size of the allocated memory for the output payload. This may be used to impose an upper limit on the instant bitrate, but should not be used as the only bitrate control. Use OPUS_SET_BITRATE to control the bitrate.

Returns

The length of the encoded packet (in bytes) on success or a negative error code (see Error codes) on failure.

4.11.3.11 opus_multistream_encode_float()

```
int opus_multistream_encode_float (
            OpusMSEncoder * st,
            const float * pcm,
            int frame_size,
            unsigned char * data,
            opus_int32 max_data_bytes )
```

Encodes a multistream Opus frame from floating point input.

Parameters

	st	`OpusMSEncoder*`: Multistream encoder state.
in	*pcm*	`const float*`: The input signal as interleaved samples with a normal range of +/-1.0. Samples with a range beyond +/-1.0 are supported but will be clipped by decoders using the integer API and should only be used if it is known that the far end supports extended dynamic range. This must contain `frame_size*channels` samples.
	frame_size	`int`: Number of samples per channel in the input signal. This must be an Opus frame size for the encoder's sampling rate. For example, at 48 kHz the permitted values are 120, 240, 480, 960, 1920, and 2880. Passing in a duration of less than 10 ms (480 samples at 48 kHz) will prevent the encoder from using the LPC or hybrid modes.
out	*data*	`unsigned char*`: Output payload. This must contain storage for at least *max_data_bytes*.
in	*max_data_bytes*	`opus_int32`: Size of the allocated memory for the output payload. This may be used to impose an upper limit on the instant bitrate, but should not be used as the only bitrate control. Use OPUS_SET_BITRATE to control the bitrate.

Returns

The length of the encoded packet (in bytes) on success or a negative error code (see Error codes) on failure.

4.11.3.12 opus_multistream_encoder_create()

```
OpusMSEncoder * opus_multistream_encoder_create (
             opus_int32 Fs,
             int channels,
             int streams,
             int coupled_streams,
             const unsigned char * mapping,
             int application,
             int * error )
```

Allocates and initializes a multistream encoder state.

Call opus_multistream_encoder_destroy() to release this object when finished.

Parameters

	Fs	`opus_int32`: Sampling rate of the input signal (in Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
	channels	`int`: Number of channels in the input signal. This must be at most 255. It may be greater than the number of coded channels (`streams + coupled_streams`).
	streams	`int`: The total number of streams to encode from the input. This must be no more than the number of channels.
	coupled_streams	`int`: Number of coupled (2 channel) streams to encode. This must be no larger than the total number of streams. Additionally, The total number of encoded channels (`streams + coupled_streams`) must be no more than the number of input channels.
in	*mapping*	`const unsigned char[channels]`: Mapping from encoded channels to input channels, as described in Opus Multistream API. As an extra constraint, the multistream encoder does not allow encoding coupled streams for which one channel is unused since this is never a good idea.
	application	`int`: The target encoder application. This must be one of the following: **OPUS_APPLICATION_VOIP** Process signal for improved speech intelligibility. **OPUS_APPLICATION_AUDIO** Favor faithfulness to the original input. **OPUS_APPLICATION_RESTRICTED_LOWDELAY** Configure the minimum possible coding delay by disabling certain modes of operation.
out	*error*	`int *`: Returns OPUS_OK on success, or an error code (see Error codes) on failure.

4.11.3.13 opus_multistream_encoder_ctl()

```
int opus_multistream_encoder_ctl (
             OpusMSEncoder * st,
```

```
            int request,
             ...  )
```

Perform a CTL function on a multistream Opus encoder.

Generally the request and subsequent arguments are generated by a convenience macro.

Parameters

st	`OpusMSEncoder*`: Multistream encoder state.
request	This and all remaining parameters should be replaced by one of the convenience macros in Generic CTLs, Encoder related CTLs, or Multistream specific encoder and decoder CTLs.

See also

Generic CTLs

Encoder related CTLs

Multistream specific encoder and decoder CTLs

4.11.3.14 opus_multistream_encoder_destroy()

```
void opus_multistream_encoder_destroy (
            OpusMSEncoder * st )
```

Frees an `OpusMSEncoder` allocated by opus_multistream_encoder_create().

Parameters

st	`OpusMSEncoder*`: Multistream encoder state to be freed.

4.11.3.15 opus_multistream_encoder_get_size()

```
opus_int32 opus_multistream_encoder_get_size (
            int streams,
            int coupled_streams )
```

Gets the size of an OpusMSEncoder structure.

Parameters

streams	`int`: The total number of streams to encode from the input. This must be no more than 255.
coupled_streams	`int`: Number of coupled (2 channel) streams to encode. This must be no larger than the total number of streams. Additionally, The total number of encoded channels (`streams + coupled_streams`) must be no more than 255.

Returns

The size in bytes on success, or a negative error code (see Error codes) on error.

4.11.3.16 opus_multistream_encoder_init()

```
int opus_multistream_encoder_init (
              OpusMSEncoder * st,
              opus_int32 Fs,
              int channels,
              int streams,
              int coupled_streams,
              const unsigned char * mapping,
              int application )
```

Initialize a previously allocated multistream encoder state.

The memory pointed to by *st* must be at least the size returned by opus_multistream_encoder_get_size(). This is intended for applications which use their own allocator instead of malloc. To reset a previously initialized state, use the OPUS_RESET_STATE CTL.

See also

opus_multistream_encoder_create

opus_multistream_encoder_get_size

Parameters

	st	`OpusMSEncoder*`: Multistream encoder state to initialize.
	Fs	`opus_int32`: Sampling rate of the input signal (in Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
	channels	`int`: Number of channels in the input signal. This must be at most 255. It may be greater than the number of coded channels (`streams + coupled_streams`).
	streams	`int`: The total number of streams to encode from the input. This must be no more than the number of channels.
	coupled_streams	`int`: Number of coupled (2 channel) streams to encode. This must be no larger than the total number of streams. Additionally, The total number of encoded channels (`streams + coupled_streams`) must be no more than the number of input channels.
in	*mapping*	`const unsigned char[channels]`: Mapping from encoded channels to input channels, as described in Opus Multistream API. As an extra constraint, the multistream encoder does not allow encoding coupled streams for which one channel is unused since this is never a good idea.
	application	`int`: The target encoder application. This must be one of the following: **OPUS_APPLICATION_VOIP** Process signal for improved speech intelligibility. **OPUS_APPLICATION_AUDIO** Favor faithfulness to the original input. **OPUS_APPLICATION_RESTRICTED_LOWDELAY** Configure the minimum possible coding delay by disabling certain modes of operation.

Returns

OPUS_OK on success, or an error code (see Error codes) on failure.

4.11.3.17 opus_multistream_surround_encoder_create()

```
OpusMSEncoder * opus_multistream_surround_encoder_create (
             opus_int32 Fs,
             int channels,
             int mapping_family,
             int * streams,
             int * coupled_streams,
             unsigned char * mapping,
             int application,
             int * error )
```

4.11.3.18 opus_multistream_surround_encoder_get_size()

```
opus_int32 opus_multistream_surround_encoder_get_size (
             int channels,
             int mapping_family )
```

4.11.3.19 opus_multistream_surround_encoder_init()

```
int opus_multistream_surround_encoder_init (
             OpusMSEncoder * st,
             opus_int32 Fs,
             int channels,
             int mapping_family,
             int * streams,
             int * coupled_streams,
             unsigned char * mapping,
             int application )
```

4.12 Opus Custom

Opus Custom is an optional part of the Opus specification and reference implementation which uses a distinct API from the regular API and supports frame sizes that are not normally supported. Use of Opus Custom is discouraged for all but very special applications for which a frame size different from 2.5, 5, 10, or 20 ms is needed (for either complexity or latency reasons) and where interoperability is less important.

Typedefs

- typedef struct OpusCustomEncoder OpusCustomEncoder

  Contains the state of an encoder.
- typedef struct OpusCustomDecoder OpusCustomDecoder

  State of the decoder.
- typedef struct OpusCustomMode OpusCustomMode

  The mode contains all the information necessary to create an encoder.

Functions

- OpusCustomMode * opus_custom_mode_create (opus_int32 Fs, int frame_size, int *error)

  Creates a new mode struct.
- void opus_custom_mode_destroy (OpusCustomMode *mode)

  Destroys a mode struct.
- int opus_custom_encoder_get_size (const OpusCustomMode *mode, int channels)

  Gets the size of an OpusCustomEncoder structure.
- OpusCustomEncoder * opus_custom_encoder_create (const OpusCustomMode *mode, int channels, int *error)

  Creates a new encoder state.
- void opus_custom_encoder_destroy (OpusCustomEncoder *st)

  Destroys an encoder state.
- int opus_custom_encode_float (OpusCustomEncoder *st, const float *pcm, int frame_size, unsigned char *compressed, int maxCompressedBytes)

  Encodes a frame of audio.
- int opus_custom_encode (OpusCustomEncoder *st, const opus_int16 *pcm, int frame_size, unsigned char *compressed, int maxCompressedBytes)

  Encodes a frame of audio.
- int opus_custom_encode24 (OpusCustomEncoder *st, const opus_int32 *pcm, int frame_size, unsigned char *compressed, int maxCompressedBytes)

  Encodes a frame of audio.
- int opus_custom_encoder_ctl (OpusCustomEncoder *OPUS_RESTRICT st, int request,...)

  Perform a CTL function on an Opus custom encoder.
- int opus_custom_decoder_get_size (const OpusCustomMode *mode, int channels)

  Gets the size of an OpusCustomDecoder structure.
- int opus_custom_decoder_init (OpusCustomDecoder *st, const OpusCustomMode *mode, int channels)

  Initializes a previously allocated decoder state The memory pointed to by st must be the size returned by opus_custom↩ _decoder_get_size.
- OpusCustomDecoder * opus_custom_decoder_create (const OpusCustomMode *mode, int channels, int *error)

  Creates a new decoder state.
- void opus_custom_decoder_destroy (OpusCustomDecoder *st)

  Destroys a decoder state.
- int opus_custom_decode_float (OpusCustomDecoder *st, const unsigned char *data, int len, float *pcm, int frame_size)

  Decode an opus custom frame with floating point output.
- int opus_custom_decode (OpusCustomDecoder *st, const unsigned char *data, int len, opus_int16 *pcm, int frame_size)

  Decode an opus custom frame.
- int opus_custom_decode24 (OpusCustomDecoder *st, const unsigned char *data, int len, opus_int32 *pcm, int frame_size)

  Decode an opus custom frame.
- int opus_custom_decoder_ctl (OpusCustomDecoder *OPUS_RESTRICT st, int request,...)

  Perform a CTL function on an Opus custom decoder.

4.12.1 Detailed Description

Opus Custom is an optional part of the Opus specification and reference implementation which uses a distinct API from the regular API and supports frame sizes that are not normally supported. Use of Opus Custom is discouraged for all but very special applications for which a frame size different from 2.5, 5, 10, or 20 ms is needed (for either complexity or latency reasons) and where interoperability is less important.

In addition to the interoperability limitations the use of Opus custom disables a substantial chunk of the codec and generally lowers the quality available at a given bitrate. Normally when an application needs a different frame size from the codec it should buffer to match the sizes but this adds a small amount of delay which may be important in some very low latency applications. Some transports (especially constant rate RF transports) may also work best with frames of particular durations.

Libopus only supports custom modes if they are enabled at compile time.

The Opus Custom API is similar to the regular API but the opus_encoder_create and opus_decoder_create calls take an additional mode parameter which is a structure produced by a call to opus_custom_mode_create. Both the encoder and decoder must create a mode using the same sample rate (fs) and frame size (frame size) so these parameters must either be signaled out of band or fixed in a particular implementation.

Similar to regular Opus the custom modes support on the fly frame size switching, but the sizes available depend on the particular frame size in use. For some initial frame sizes on a single on the fly size is available.

4.12.2 Typedef Documentation

4.12.2.1 OpusCustomDecoder

```
typedef struct OpusCustomDecoder OpusCustomDecoder
```

State of the decoder.

One decoder state is needed for each stream. It is initialized once at the beginning of the stream. Do *not* re-initialize the state for every frame.

Decoder state

4.12.2.2 OpusCustomEncoder

```
typedef struct OpusCustomEncoder OpusCustomEncoder
```

Contains the state of an encoder.

One encoder state is needed for each stream. It is initialized once at the beginning of the stream. Do *not* re-initialize the state for every frame.

Encoder state

4.12.2.3 OpusCustomMode

```
typedef struct OpusCustomMode OpusCustomMode
```

The mode contains all the information necessary to create an encoder.

Both the encoder and decoder need to be initialized with exactly the same mode, otherwise the output will be corrupted. The mode MUST NOT BE DESTROYED until the encoders and decoders that use it are destroyed as well.

Mode configuration

4.12.3 Function Documentation

4.12.3.1 opus_custom_decode()

```
int opus_custom_decode (
            OpusCustomDecoder * st,
            const unsigned char * data,
            int len,
            opus_int16 * pcm,
            int frame_size )
```

Decode an opus custom frame.

Parameters

in	*st*	OpusCustomDecoder*: Decoder state
in	*data*	char*: Input payload. Use a NULL pointer to indicate packet loss
in	*len*	int: Number of bytes in payload
out	*pcm*	opus_int16*: Output signal (interleaved if 2 channels). length is frame_size*channels*sizeof(opus_int16)
in	*frame_size*	Number of samples per channel of available space in *pcm.

Returns

Number of decoded samples or Error codes

4.12.3.2 opus_custom_decode24()

```
int opus_custom_decode24 (
            OpusCustomDecoder * st,
            const unsigned char * data,
            int len,
            opus_int32 * pcm,
            int frame_size )
```

Decode an opus custom frame.

Parameters

in	*st*	OpusCustomDecoder*: Decoder state
in	*data*	char*: Input payload. Use a NULL pointer to indicate packet loss
in	*len*	int: Number of bytes in payload
out	*pcm*	opus_int32*: Output signal (interleaved if 2 channels) representing (or slightly exceeding) 24-bit values. length is frame_size*channels*sizeof(opus_int32)
in	*frame_size*	Number of samples per channel of available space in *pcm.

Returns

Number of decoded samples or Error codes

4.12.3.3 opus_custom_decode_float()

```
int opus_custom_decode_float (
            OpusCustomDecoder * st,
            const unsigned char * data,
            int len,
            float * pcm,
            int frame_size )
```

Decode an opus custom frame with floating point output.

Parameters

in	*st*	OpusCustomDecoder*: Decoder state
in	*data*	char*: Input payload. Use a NULL pointer to indicate packet loss
in	*len*	int: Number of bytes in payload
out	*pcm*	float*: Output signal (interleaved if 2 channels). length is frame_size*channels*sizeof(float)
in	*frame_size*	Number of samples per channel of available space in *pcm.

Returns

Number of decoded samples or Error codes

4.12.3.4 opus_custom_decoder_create()

```
OpusCustomDecoder * opus_custom_decoder_create (
            const OpusCustomMode * mode,
            int channels,
            int * error )
```

Creates a new decoder state.

Each stream needs its own decoder state (can't be shared across simultaneous streams).

Parameters

in	*mode*	OpusCustomMode: Contains all the information about the characteristics of the stream (must be the same characteristics as used for the encoder)
in	*channels*	int: Number of channels
out	*error*	int*: Returns an error code

Returns

Newly created decoder state.

4.12.3.5 opus_custom_decoder_ctl()

```
int opus_custom_decoder_ctl (
            OpusCustomDecoder *OPUS_RESTRICT st,
            int request,
             ...  )
```

Perform a CTL function on an Opus custom decoder.

Generally the request and subsequent arguments are generated by a convenience macro.

See also

Generic CTLs

4.12.3.6 opus_custom_decoder_destroy()

```
void opus_custom_decoder_destroy (
            OpusCustomDecoder * st )
```

Destroys a decoder state.

Parameters

in	*st*	OpusCustomDecoder*: State to be freed.

4.12.3.7 opus_custom_decoder_get_size()

```
int opus_custom_decoder_get_size (
            const OpusCustomMode * mode,
            int channels )
```

Gets the size of an OpusCustomDecoder structure.

Parameters

in	*mode*	`OpusCustomMode *`: Mode configuration
in	*channels*	`int`: Number of channels

Returns

size

4.12.3.8 opus_custom_decoder_init()

```
int opus_custom_decoder_init (
            OpusCustomDecoder * st,
            const OpusCustomMode * mode,
            int channels )
```

Initializes a previously allocated decoder state The memory pointed to by st must be the size returned by opus_custom↩ _decoder_get_size.

This is intended for applications which use their own allocator instead of malloc.

See also

opus_custom_decoder_create(),opus_custom_decoder_get_size() To reset a previously initialized state use the OPUS_RESET_STATE CTL.

Parameters

in	*st*	`OpusCustomDecoder*`: Decoder state
in	*mode*	`OpusCustomMode *`: Contains all the information about the characteristics of the stream (must be the same characteristics as used for the encoder)
in	*channels*	`int`: Number of channels

Returns

OPUS_OK Success or Error codes

4.12.3.9 opus_custom_encode()

```
int opus_custom_encode (
            OpusCustomEncoder * st,
            const opus_int16 * pcm,
            int frame_size,
            unsigned char * compressed,
            int maxCompressedBytes )
```

Encodes a frame of audio.

Parameters

in	*st*	OpusCustomEncoder*: Encoder state
in	*pcm*	opus_int16*: PCM audio in signed 16-bit format (native endian). There must be exactly frame_size samples per channel.
in	*frame_size*	int: Number of samples per frame of input signal
out	*compressed*	char *: The compressed data is written here. This may not alias pcm and must be at least maxCompressedBytes long.
in	*maxCompressedBytes*	int: Maximum number of bytes to use for compressing the frame (can change from one frame to another)

Returns

Number of bytes written to "compressed". If negative, an error has occurred (see error codes). It is IMPORTANT that the length returned be somehow transmitted to the decoder. Otherwise, no decoding is possible.

4.12.3.10 opus_custom_encode24()

```
int opus_custom_encode24 (
            OpusCustomEncoder * st,
            const opus_int32 * pcm,
            int frame_size,
            unsigned char * compressed,
            int maxCompressedBytes )
```

Encodes a frame of audio.

Parameters

in	*st*	OpusCustomEncoder*: Encoder state
in	*pcm*	opus_int32*: PCM audio in signed 32-bit format (native endian) representing (or slightly exceeding) 24-bit values. There must be exactly frame_size samples per channel.
in	*frame_size*	int: Number of samples per frame of input signal
out	*compressed*	char *: The compressed data is written here. This may not alias pcm and must be at least maxCompressedBytes long.
in	*maxCompressedBytes*	int: Maximum number of bytes to use for compressing the frame (can change from one frame to another)

Returns

Number of bytes written to "compressed". If negative, an error has occurred (see error codes). It is IMPORTANT that the length returned be somehow transmitted to the decoder. Otherwise, no decoding is possible.

4.12.3.11 opus_custom_encode_float()

```
int opus_custom_encode_float (
```

```
OpusCustomEncoder * st,
const float * pcm,
int frame_size,
unsigned char * compressed,
int maxCompressedBytes )
```

Encodes a frame of audio.

Parameters

in	*st*	OpusCustomEncoder*: Encoder state
in	*pcm*	float*: PCM audio in float format, with a normal range of +/-1.0. Samples with a range beyond +/-1.0 are supported but will be clipped by decoders using the integer API and should only be used if it is known that the far end supports extended dynamic range. There must be exactly frame_size samples per channel.
in	*frame_size*	int: Number of samples per frame of input signal
out	*compressed*	char *: The compressed data is written here. This may not alias pcm and must be at least maxCompressedBytes long.
in	*maxCompressedBytes*	int: Maximum number of bytes to use for compressing the frame (can change from one frame to another)

Returns

Number of bytes written to "compressed". If negative, an error has occurred (see error codes). It is IMPORTANT that the length returned be somehow transmitted to the decoder. Otherwise, no decoding is possible.

4.12.3.12 opus_custom_encoder_create()

```
OpusCustomEncoder * opus_custom_encoder_create (
const OpusCustomMode * mode,
int channels,
int * error )
```

Creates a new encoder state.

Each stream needs its own encoder state (can't be shared across simultaneous streams).

Parameters

in	*mode*	OpusCustomMode*: Contains all the information about the characteristics of the stream (must be the same characteristics as used for the decoder)
in	*channels*	int: Number of channels
out	*error*	int*: Returns an error code

Returns

Newly created encoder state.

4.12.3.13 opus_custom_encoder_ctl()

```
int opus_custom_encoder_ctl (
              OpusCustomEncoder *OPUS_RESTRICT st,
              int request,
               ...  )
```

Perform a CTL function on an Opus custom encoder.

Generally the request and subsequent arguments are generated by a convenience macro.

See also

Encoder related CTLs

4.12.3.14 opus_custom_encoder_destroy()

```
void opus_custom_encoder_destroy (
              OpusCustomEncoder * st )
```

Destroys an encoder state.

Parameters

in	*st*	OpusCustomEncoder*: State to be freed.

4.12.3.15 opus_custom_encoder_get_size()

```
int opus_custom_encoder_get_size (
              const OpusCustomMode * mode,
              int channels )
```

Gets the size of an OpusCustomEncoder structure.

Parameters

in	*mode*	OpusCustomMode *: Mode configuration
in	*channels*	int: Number of channels

Returns

size

4.12.3.16 opus_custom_mode_create()

```
OpusCustomMode * opus_custom_mode_create (
```

```
            opus_int32 Fs,
            int frame_size,
            int * error )
```

Creates a new mode struct.

This will be passed to an encoder or decoder. The mode MUST NOT BE DESTROYED until the encoders and decoders that use it are destroyed as well.

Parameters

in	*Fs*	int: Sampling rate (8000 to 96000 Hz)
in	*frame_size*	int: Number of samples (per channel) to encode in each packet (64 - 1024, prime factorization must contain zero or more 2s, 3s, or 5s and no other primes)
out	*error*	int*: Returned error code (if NULL, no error will be returned)

Returns

A newly created mode

4.12.3.17 opus_custom_mode_destroy()

```
void opus_custom_mode_destroy (
            OpusCustomMode * mode )
```

Destroys a mode struct.

Only call this after all encoders and decoders using this mode are destroyed as well.

Parameters

in	*mode*	OpusCustomMode*: Mode to be freed.

Chapter 5

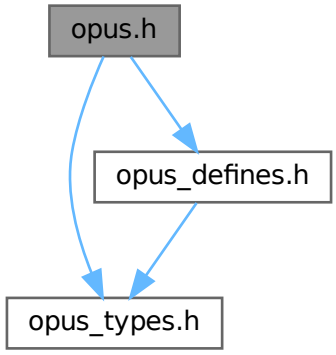
File Documentation

5.1 opus.h File Reference

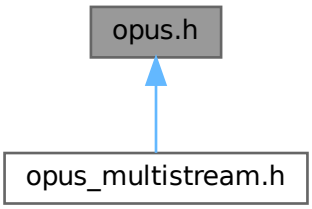
Opus reference implementation API.

```
#include "opus_types.h"
#include "opus_defines.h"
```

Include dependency graph for opus.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct OpusEncoder OpusEncoder

  Opus encoder state.
- typedef struct OpusDecoder OpusDecoder

  Opus decoder state.
- typedef struct OpusDREDDecoder OpusDREDDecoder

  Opus DRED decoder.
- typedef struct OpusDRED OpusDRED

  Opus DRED state.
- typedef struct OpusRepacketizer OpusRepacketizer

Functions

- int opus_encoder_get_size (int channels)

  Gets the size of an `OpusEncoder` structure.
- OpusEncoder * opus_encoder_create (opus_int32 Fs, int channels, int application, int *error)

  Allocates and initializes an encoder state.
- int opus_encoder_init (OpusEncoder *st, opus_int32 Fs, int channels, int application)

  Initializes a previously allocated encoder state The memory pointed to by st must be at least the size returned by opus_encoder_get_size().
- opus_int32 opus_encode (OpusEncoder *st, const opus_int16 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes an Opus frame.
- opus_int32 opus_encode24 (OpusEncoder *st, const opus_int32 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes an Opus frame.
- opus_int32 opus_encode_float (OpusEncoder *st, const float *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)

  Encodes an Opus frame from floating point input.
- void opus_encoder_destroy (OpusEncoder *st)

  Frees an `OpusEncoder` allocated by opus_encoder_create().

- int opus_encoder_ctl (OpusEncoder *st, int request,...)

  Perform a CTL function on an Opus encoder.
- int opus_decoder_get_size (int channels)

  Gets the size of an `OpusDecoder` structure.
- OpusDecoder * opus_decoder_create (opus_int32 Fs, int channels, int *error)

  Allocates and initializes a decoder state.
- int opus_decoder_init (OpusDecoder *st, opus_int32 Fs, int channels)

  Initializes a previously allocated decoder state.
- int opus_decode (OpusDecoder *st, const unsigned char *data, opus_int32 len, opus_int16 *pcm, int frame_size, int decode_fec)

  Decode an Opus packet.
- int opus_decode24 (OpusDecoder *st, const unsigned char *data, opus_int32 len, opus_int32 *pcm, int frame↩ _size, int decode_fec)

  Decode an Opus packet.
- int opus_decode_float (OpusDecoder *st, const unsigned char *data, opus_int32 len, float *pcm, int frame_size, int decode_fec)

  Decode an Opus packet with floating point output.
- int opus_decoder_ctl (OpusDecoder *st, int request,...)

  Perform a CTL function on an Opus decoder.
- void opus_decoder_destroy (OpusDecoder *st)

  Frees an `OpusDecoder` allocated by opus_decoder_create().
- int opus_dred_decoder_get_size (void)

  Gets the size of an `OpusDREDDecoder` structure.
- OpusDREDDecoder * opus_dred_decoder_create (int *error)

  Allocates and initializes an OpusDREDDecoder state.
- int opus_dred_decoder_init (OpusDREDDecoder *dec)

  Initializes an `OpusDREDDecoder` state.
- void opus_dred_decoder_destroy (OpusDREDDecoder *dec)

  Frees an `OpusDREDDecoder` allocated by opus_dred_decoder_create().
- int opus_dred_decoder_ctl (OpusDREDDecoder *dred_dec, int request,...)

  Perform a CTL function on an Opus DRED decoder.
- int opus_dred_get_size (void)

  Gets the size of an `OpusDRED` structure.
- OpusDRED * opus_dred_alloc (int *error)

  Allocates and initializes a DRED state.
- void opus_dred_free (OpusDRED *dec)

  Frees an `OpusDRED` allocated by opus_dred_create().
- int opus_dred_parse (OpusDREDDecoder *dred_dec, OpusDRED *dred, const unsigned char *data, opus_int32 len, opus_int32 max_dred_samples, opus_int32 sampling_rate, int *dred_end, int defer_processing)

  Decode an Opus DRED packet.
- int opus_dred_process (OpusDREDDecoder *dred_dec, const OpusDRED *src, OpusDRED *dst)

  Finish decoding an Opus DRED packet.
- int opus_decoder_dred_decode (OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset, opus_int16 *pcm, opus_int32 frame_size)

  Decode audio from an Opus DRED packet with 16-bit output.
- int opus_decoder_dred_decode24 (OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset, opus_int32 *pcm, opus_int32 frame_size)

  Decode audio from an Opus DRED packet with 24-bit output.

- int opus_decoder_dred_decode_float (OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset, float *pcm, opus_int32 frame_size)

  Decode audio from an Opus DRED packet with floating point output.
- int opus_packet_parse (const unsigned char *data, opus_int32 len, unsigned char *out_toc, const unsigned char *frames[48], opus_int16 size[48], int *payload_offset)

  Parse an opus packet into one or more frames.
- int opus_packet_get_bandwidth (const unsigned char *data)

  Gets the bandwidth of an Opus packet.
- int opus_packet_get_samples_per_frame (const unsigned char *data, opus_int32 Fs)

  Gets the number of samples per frame from an Opus packet.
- int opus_packet_get_nb_channels (const unsigned char *data)

  Gets the number of channels from an Opus packet.
- int opus_packet_get_nb_frames (const unsigned char packet[ ], opus_int32 len)

  Gets the number of frames in an Opus packet.
- int opus_packet_get_nb_samples (const unsigned char packet[ ], opus_int32 len, opus_int32 Fs)

  Gets the number of samples of an Opus packet.
- int opus_packet_has_lbrr (const unsigned char packet[ ], opus_int32 len)

  Checks whether an Opus packet has LBRR.
- int opus_decoder_get_nb_samples (const OpusDecoder *dec, const unsigned char packet[ ], opus_int32 len)

  Gets the number of samples of an Opus packet.
- void opus_pcm_soft_clip (float *pcm, int frame_size, int channels, float *softclip_mem)

  Applies soft-clipping to bring a float signal within the [-1,1] range.
- int opus_repacketizer_get_size (void)

  Gets the size of an `OpusRepacketizer` structure.
- OpusRepacketizer * opus_repacketizer_init (OpusRepacketizer *rp)

  (Re)initializes a previously allocated repacketizer state.
- OpusRepacketizer * opus_repacketizer_create (void)

  Allocates memory and initializes the new repacketizer with opus_repacketizer_init().
- void opus_repacketizer_destroy (OpusRepacketizer *rp)

  Frees an `OpusRepacketizer` allocated by opus_repacketizer_create().
- int opus_repacketizer_cat (OpusRepacketizer *rp, const unsigned char *data, opus_int32 len)

  Add a packet to the current repacketizer state.
- opus_int32 opus_repacketizer_out_range (OpusRepacketizer *rp, int begin, int end, unsigned char *data, opus_int32 maxlen)

  Construct a new packet from data previously submitted to the repacketizer state via opus_repacketizer_cat().
- int opus_repacketizer_get_nb_frames (OpusRepacketizer *rp)

  Return the total number of frames contained in packet data submitted to the repacketizer state so far via opus_repacketizer_cat() since the last call to opus_repacketizer_init() or opus_repacketizer_create().
- opus_int32 opus_repacketizer_out (OpusRepacketizer *rp, unsigned char *data, opus_int32 maxlen)

  Construct a new packet from data previously submitted to the repacketizer state via opus_repacketizer_cat().
- int opus_packet_pad (unsigned char *data, opus_int32 len, opus_int32 new_len)

  Pads a given Opus packet to a larger size (possibly changing the TOC sequence).
- opus_int32 opus_packet_unpad (unsigned char *data, opus_int32 len)

  Remove all padding from a given Opus packet and rewrite the TOC sequence to minimize space usage.
- int opus_multistream_packet_pad (unsigned char *data, opus_int32 len, opus_int32 new_len, int nb_streams)

  Pads a given Opus multi-stream packet to a larger size (possibly changing the TOC sequence).
- opus_int32 opus_multistream_packet_unpad (unsigned char *data, opus_int32 len, int nb_streams)

  Remove all padding from a given Opus multi-stream packet and rewrite the TOC sequence to minimize space usage.

5.1.1 Detailed Description

Opus reference implementation API.

5.2 opus.h

Go to the documentation of this file.

```
00001 /* Copyright (c) 2010-2011 Xiph.Org Foundation, Skype Limited
00002    Written by Jean-Marc Valin and Koen Vos */
00003 /*
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions
00006    are met:
00007
00008    - Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    - Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in the
00013    documentation and/or other materials provided with the distribution.
00014
00015    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00016    ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00017    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00018    A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
00019    OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00020    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00021    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00024    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00025    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026 */
00027
00033 #ifndef OPUS_H
00034 #define OPUS_H
00035
00036 #include "opus_types.h"
00037 #include "opus_defines.h"
00038
00039 #ifdef __cplusplus
00040 extern "C" {
00041 #endif
00042
00164 typedef struct OpusEncoder OpusEncoder;
00165
00174 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_encoder_get_size(int channels);
00175
00211 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusEncoder *opus_encoder_create(
00212     opus_int32 Fs,
00213     int channels,
00214     int application,
00215     int *error
00216 );
00217
00231 OPUS_EXPORT int opus_encoder_init(
00232     OpusEncoder *st,
00233     opus_int32 Fs,
00234     int channels,
00235     int application
00236 ) OPUS_ARG_NONNULL(1);
00237
00266 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_encode(
00267     OpusEncoder *st,
00268     const opus_int16 *pcm,
00269     int frame_size,
00270     unsigned char *data,
00271     opus_int32 max_data_bytes
00272 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00273
00302 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_encode24(
00303     OpusEncoder *st,
00304     const opus_int32 *pcm,
```

```
00305     int frame_size,
00306     unsigned char *data,
00307     opus_int32 max_data_bytes
00308 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00309
00343 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_encode_float(
00344     OpusEncoder *st,
00345     const float *pcm,
00346     int frame_size,
00347     unsigned char *data,
00348     opus_int32 max_data_bytes
00349 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00350
00354 OPUS_EXPORT void opus_encoder_destroy(OpusEncoder *st);
00355
00367 OPUS_EXPORT int opus_encoder_ctl(OpusEncoder *st, int request, ...) OPUS_ARG_NONNULL(1);
00438 typedef struct OpusDecoder OpusDecoder;
00439
00445 typedef struct OpusDREDDecoder OpusDREDDecoder;
00446
00447
00453 typedef struct OpusDRED OpusDRED;
00454
00460 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decoder_get_size(int channels);
00461
00477 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusDecoder *opus_decoder_create(
00478     opus_int32 Fs,
00479     int channels,
00480     int *error
00481 );
00482
00494 OPUS_EXPORT int opus_decoder_init(
00495     OpusDecoder *st,
00496     opus_int32 Fs,
00497     int channels
00498 ) OPUS_ARG_NONNULL(1);
00499
00516 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decode(
00517     OpusDecoder *st,
00518     const unsigned char *data,
00519     opus_int32 len,
00520     opus_int16 *pcm,
00521     int frame_size,
00522     int decode_fec
00523 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00524
00541 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decode24(
00542     OpusDecoder *st,
00543     const unsigned char *data,
00544     opus_int32 len,
00545     opus_int32 *pcm,
00546     int frame_size,
00547     int decode_fec
00548 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00549
00566 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decode_float(
00567     OpusDecoder *st,
00568     const unsigned char *data,
00569     opus_int32 len,
00570     float *pcm,
00571     int frame_size,
00572     int decode_fec
00573 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00574
00586 OPUS_EXPORT int opus_decoder_ctl(OpusDecoder *st, int request, ...) OPUS_ARG_NONNULL(1);
00587
00591 OPUS_EXPORT void opus_decoder_destroy(OpusDecoder *st);
00592
00596 OPUS_EXPORT int opus_dred_decoder_get_size(void);
00597
00601 OPUS_EXPORT OpusDREDDecoder *opus_dred_decoder_create(int *error);
00602
00606 OPUS_EXPORT int opus_dred_decoder_init(OpusDREDDecoder *dec);
00607
00611 OPUS_EXPORT void opus_dred_decoder_destroy(OpusDREDDecoder *dec);
00612
00624 OPUS_EXPORT int opus_dred_decoder_ctl(OpusDREDDecoder *dred_dec, int request, ...);
00625
00629 OPUS_EXPORT int opus_dred_get_size(void);
00630
00634 OPUS_EXPORT OpusDRED *opus_dred_alloc(int *error);
```

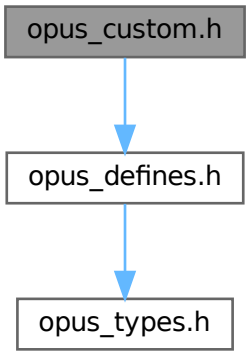
```
00635
00639 OPUS_EXPORT void opus_dred_free(OpusDRED *dec);
00640
00652 OPUS_EXPORT int opus_dred_parse(OpusDREDDecoder *dred_dec, OpusDRED *dred, const unsigned char *data,
      opus_int32 len, opus_int32 max_dred_samples, opus_int32 sampling_rate, int *dred_end, int
      defer_processing) OPUS_ARG_NONNULL(1);
00653
00661 OPUS_EXPORT int opus_dred_process(OpusDREDDecoder *dred_dec, const OpusDRED *src, OpusDRED *dst);
00662
00673 OPUS_EXPORT int opus_decoder_dred_decode(OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset,
      opus_int16 *pcm, opus_int32 frame_size);
00674
00685 OPUS_EXPORT int opus_decoder_dred_decode24(OpusDecoder *st, const OpusDRED *dred, opus_int32 dred_offset,
      opus_int32 *pcm, opus_int32 frame_size);
00686
00697 OPUS_EXPORT int opus_decoder_dred_decode_float(OpusDecoder *st, const OpusDRED *dred, opus_int32
      dred_offset, float *pcm, opus_int32 frame_size);
00698
00699
00713 OPUS_EXPORT int opus_packet_parse(
00714    const unsigned char *data,
00715    opus_int32 len,
00716    unsigned char *out_toc,
00717    const unsigned char *frames[48],
00718    opus_int16 size[48],
00719    int *payload_offset
00720 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(5);
00721
00731 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_packet_get_bandwidth(const unsigned char *data)
      OPUS_ARG_NONNULL(1);
00732
00742 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_packet_get_samples_per_frame(const unsigned char *data,
      opus_int32 Fs) OPUS_ARG_NONNULL(1);
00743
00749 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_packet_get_nb_channels(const unsigned char *data)
      OPUS_ARG_NONNULL(1);
00750
00758 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_packet_get_nb_frames(const unsigned char packet[], opus_int32
      len) OPUS_ARG_NONNULL(1);
00759
00770 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_packet_get_nb_samples(const unsigned char packet[],
      opus_int32 len, opus_int32 Fs) OPUS_ARG_NONNULL(1);
00771
00778 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_packet_has_lbrr(const unsigned char packet[], opus_int32
      len);
00779
00788 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decoder_get_nb_samples(const OpusDecoder *dec, const unsigned
      char packet[], opus_int32 len) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2);
00789
00800 OPUS_EXPORT void opus_pcm_soft_clip(float *pcm, int frame_size, int channels, float *softclip_mem);
00801
00802
00948 typedef struct OpusRepacketizer OpusRepacketizer;
00949
00953 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_repacketizer_get_size(void);
00954
00972 OPUS_EXPORT OpusRepacketizer *opus_repacketizer_init(OpusRepacketizer *rp) OPUS_ARG_NONNULL(1);
00973
00977 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusRepacketizer *opus_repacketizer_create(void);
00978
00983 OPUS_EXPORT void opus_repacketizer_destroy(OpusRepacketizer *rp);
00984
01032 OPUS_EXPORT int opus_repacketizer_cat(OpusRepacketizer *rp, const unsigned char *data, opus_int32 len)
      OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2);
01033
01034
01066 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_repacketizer_out_range(OpusRepacketizer *rp, int
      begin, int end, unsigned char *data, opus_int32 maxlen) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
01067
01078 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_repacketizer_get_nb_frames(OpusRepacketizer *rp)
      OPUS_ARG_NONNULL(1);
01079
01109 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_repacketizer_out(OpusRepacketizer *rp, unsigned char
      *data, opus_int32 maxlen) OPUS_ARG_NONNULL(1);
01110
01123 OPUS_EXPORT int opus_packet_pad(unsigned char *data, opus_int32 len, opus_int32 new_len);
01124
01136 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_packet_unpad(unsigned char *data, opus_int32 len);
01137
01152 OPUS_EXPORT int opus_multistream_packet_pad(unsigned char *data, opus_int32 len, opus_int32 new_len, int
      nb_streams);
```

```
01153
01167 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_multistream_packet_unpad(unsigned char *data,
      opus_int32 len, int nb_streams);
01168
01171 #ifdef __cplusplus
01172 }
01173 #endif
01174
01175 #endif /* OPUS_H */
```

5.3 opus_custom.h File Reference

Opus-Custom reference implementation API.

`#include "opus_defines.h"`
Include dependency graph for opus_custom.h:



Macros

- #define OPUS_CUSTOM_EXPORT
- #define OPUS_CUSTOM_EXPORT_STATIC

Typedefs

- typedef struct OpusCustomEncoder OpusCustomEncoder

  Contains the state of an encoder.
- typedef struct OpusCustomDecoder OpusCustomDecoder

  State of the decoder.
- typedef struct OpusCustomMode OpusCustomMode

  The mode contains all the information necessary to create an encoder.

Functions

- OpusCustomMode * opus_custom_mode_create (opus_int32 Fs, int frame_size, int *error)

  Creates a new mode struct.
- void opus_custom_mode_destroy (OpusCustomMode *mode)

  Destroys a mode struct.
- int opus_custom_encoder_get_size (const OpusCustomMode *mode, int channels)

  Gets the size of an OpusCustomEncoder structure.
- OpusCustomEncoder * opus_custom_encoder_create (const OpusCustomMode *mode, int channels, int *error)

  Creates a new encoder state.
- void opus_custom_encoder_destroy (OpusCustomEncoder *st)

  Destroys an encoder state.
- int opus_custom_encode_float (OpusCustomEncoder *st, const float *pcm, int frame_size, unsigned char *compressed, int maxCompressedBytes)

  Encodes a frame of audio.
- int opus_custom_encode (OpusCustomEncoder *st, const opus_int16 *pcm, int frame_size, unsigned char *compressed, int maxCompressedBytes)

  Encodes a frame of audio.
- int opus_custom_encode24 (OpusCustomEncoder *st, const opus_int32 *pcm, int frame_size, unsigned char *compressed, int maxCompressedBytes)

  Encodes a frame of audio.
- int opus_custom_encoder_ctl (OpusCustomEncoder *OPUS_RESTRICT st, int request,...)

  Perform a CTL function on an Opus custom encoder.
- int opus_custom_decoder_get_size (const OpusCustomMode *mode, int channels)

  Gets the size of an OpusCustomDecoder structure.
- int opus_custom_decoder_init (OpusCustomDecoder *st, const OpusCustomMode *mode, int channels)

  Initializes a previously allocated decoder state The memory pointed to by st must be the size returned by opus_custom↩ _decoder_get_size.
- OpusCustomDecoder * opus_custom_decoder_create (const OpusCustomMode *mode, int channels, int *error)

  Creates a new decoder state.
- void opus_custom_decoder_destroy (OpusCustomDecoder *st)

  Destroys a decoder state.
- int opus_custom_decode_float (OpusCustomDecoder *st, const unsigned char *data, int len, float *pcm, int frame_size)

  Decode an opus custom frame with floating point output.
- int opus_custom_decode (OpusCustomDecoder *st, const unsigned char *data, int len, opus_int16 *pcm, int frame_size)

  Decode an opus custom frame.
- int opus_custom_decode24 (OpusCustomDecoder *st, const unsigned char *data, int len, opus_int32 *pcm, int frame_size)

  Decode an opus custom frame.
- int opus_custom_decoder_ctl (OpusCustomDecoder *OPUS_RESTRICT st, int request,...)

  Perform a CTL function on an Opus custom decoder.

5.3.1 Detailed Description

Opus-Custom reference implementation API.

5.3.2 Macro Definition Documentation

5.3.2.1 OPUS_CUSTOM_EXPORT

```
#define OPUS_CUSTOM_EXPORT
```

5.3.2.2 OPUS_CUSTOM_EXPORT_STATIC

```
#define OPUS_CUSTOM_EXPORT_STATIC
```

5.4 opus_custom.h

Go to the documentation of this file.

```
00001 /* Copyright (c) 2007-2008 CSIRO
00002    Copyright (c) 2007-2009 Xiph.Org Foundation
00003    Copyright (c) 2008-2012 Gregory Maxwell
00004    Written by Jean-Marc Valin and Gregory Maxwell */
00005 /*
00006    Redistribution and use in source and binary forms, with or without
00007    modification, are permitted provided that the following conditions
00008    are met:
00009
00010    - Redistributions of source code must retain the above copyright
00011    notice, this list of conditions and the following disclaimer.
00012
00013    - Redistributions in binary form must reproduce the above copyright
00014    notice, this list of conditions and the following disclaimer in the
00015    documentation and/or other materials provided with the distribution.
00016
00017    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018    ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020    A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
00021    OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00022    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00024    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00025    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00026    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00027    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 */
00029
00035 #ifndef OPUS_CUSTOM_H
00036 #define OPUS_CUSTOM_H
00037
00038 #include "opus_defines.h"
00039
00040 #ifdef __cplusplus
00041 extern "C" {
00042 #endif
00043
00044 #if defined(CUSTOM_MODES) || defined(ENABLE_OPUS_CUSTOM_API)
00045 # define OPUS_CUSTOM_EXPORT OPUS_EXPORT
00046 # define OPUS_CUSTOM_EXPORT_STATIC OPUS_EXPORT
00047 #else
00048 # define OPUS_CUSTOM_EXPORT
00049 # ifdef OPUS_BUILD
00050 #  define OPUS_CUSTOM_EXPORT_STATIC static OPUS_INLINE
00051 # else
00052 #  define OPUS_CUSTOM_EXPORT_STATIC
00053 # endif
00054 #endif
00055
00095 typedef struct OpusCustomEncoder OpusCustomEncoder;
00096
00102 typedef struct OpusCustomDecoder OpusCustomDecoder;
00103
```

```
00111 typedef struct OpusCustomMode OpusCustomMode;
00112
00122 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT OpusCustomMode *opus_custom_mode_create(opus_int32 Fs, int
      frame_size, int *error);
00123
00128 OPUS_CUSTOM_EXPORT void opus_custom_mode_destroy(OpusCustomMode *mode);
00129
00130
00131 #if !defined(OPUS_BUILD) || defined(CELT_ENCODER_C)
00132
00133 /* Encoder */
00139 OPUS_CUSTOM_EXPORT_STATIC OPUS_WARN_UNUSED_RESULT int opus_custom_encoder_get_size(
00140     const OpusCustomMode *mode,
00141     int channels
00142 ) OPUS_ARG_NONNULL(1);
00143
00144 #if defined(CUSTOM_MODES) || defined(ENABLE_OPUS_CUSTOM_API)
00157 OPUS_CUSTOM_EXPORT int opus_custom_encoder_init(
00158     OpusCustomEncoder *st,
00159     const OpusCustomMode *mode,
00160     int channels
00161 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2);
00162 # endif
00163 #endif
00164
00165
00175 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT OpusCustomEncoder *opus_custom_encoder_create(
00176     const OpusCustomMode *mode,
00177     int channels,
00178     int *error
00179 ) OPUS_ARG_NONNULL(1);
00180
00181
00185 OPUS_CUSTOM_EXPORT void opus_custom_encoder_destroy(OpusCustomEncoder *st);
00186
00204 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT int opus_custom_encode_float(
00205     OpusCustomEncoder *st,
00206     const float *pcm,
00207     int frame_size,
00208     unsigned char *compressed,
00209     int maxCompressedBytes
00210 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00211
00225 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT int opus_custom_encode(
00226     OpusCustomEncoder *st,
00227     const opus_int16 *pcm,
00228     int frame_size,
00229     unsigned char *compressed,
00230     int maxCompressedBytes
00231 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00232
00246 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT int opus_custom_encode24(
00247     OpusCustomEncoder *st,
00248     const opus_int32 *pcm,
00249     int frame_size,
00250     unsigned char *compressed,
00251     int maxCompressedBytes
00252 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00253
00260 OPUS_CUSTOM_EXPORT int opus_custom_encoder_ctl(OpusCustomEncoder * OPUS_RESTRICT st, int request, ...)
      OPUS_ARG_NONNULL(1);
00261
00262
00263 #if !defined(OPUS_BUILD) || defined(CELT_DECODER_C)
00264 /* Decoder */
00265
00271 OPUS_CUSTOM_EXPORT_STATIC OPUS_WARN_UNUSED_RESULT int opus_custom_decoder_get_size(
00272     const OpusCustomMode *mode,
00273     int channels
00274 ) OPUS_ARG_NONNULL(1);
00275
00288 OPUS_CUSTOM_EXPORT_STATIC int opus_custom_decoder_init(
00289     OpusCustomDecoder *st,
00290     const OpusCustomMode *mode,
00291     int channels
00292 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2);
00293
00294 #endif
00295
00296
00305 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT OpusCustomDecoder *opus_custom_decoder_create(
```

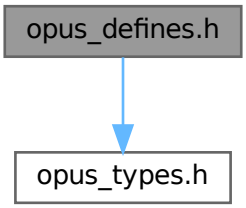
```
00306     const OpusCustomMode *mode,
00307     int channels,
00308     int *error
00309 ) OPUS_ARG_NONNULL(1);
00310
00314 OPUS_CUSTOM_EXPORT void opus_custom_decoder_destroy(OpusCustomDecoder *st);
00315
00325 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT int opus_custom_decode_float(
00326     OpusCustomDecoder *st,
00327     const unsigned char *data,
00328     int len,
00329     float *pcm,
00330     int frame_size
00331 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00332
00342 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT int opus_custom_decode(
00343     OpusCustomDecoder *st,
00344     const unsigned char *data,
00345     int len,
00346     opus_int16 *pcm,
00347     int frame_size
00348 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00349
00359 OPUS_CUSTOM_EXPORT OPUS_WARN_UNUSED_RESULT int opus_custom_decode24(
00360     OpusCustomDecoder *st,
00361     const unsigned char *data,
00362     int len,
00363     opus_int32 *pcm,
00364     int frame_size
00365 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00366
00373 OPUS_CUSTOM_EXPORT int opus_custom_decoder_ctl(OpusCustomDecoder * OPUS_RESTRICT st, int request, ...)
      OPUS_ARG_NONNULL(1);
00374
00377 #ifdef __cplusplus
00378 }
00379 #endif
00380
00381 #endif /* OPUS_CUSTOM_H */
```

5.5 opus_defines.h File Reference

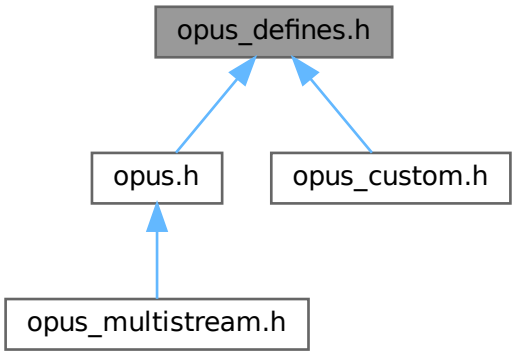
Opus reference implementation constants.

`#include "opus_types.h"`

Include dependency graph for opus_defines.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define OPUS_OK

  No error.
- #define OPUS_BAD_ARG

  One or more invalid/out of range arguments.
- #define OPUS_BUFFER_TOO_SMALL

  Not enough bytes allocated in the buffer.
- #define OPUS_INTERNAL_ERROR

  An internal error was detected.
- #define OPUS_INVALID_PACKET

  The compressed data passed is corrupted.
- #define OPUS_UNIMPLEMENTED

  Invalid/unsupported request number.
- #define OPUS_INVALID_STATE

  An encoder or decoder structure is invalid or already freed.
- #define OPUS_ALLOC_FAIL

  Memory allocation has failed.
- #define OPUS_AUTO

  Auto/default setting.
- #define OPUS_BITRATE_MAX

  Maximum bitrate.
- #define OPUS_APPLICATION_VOIP

  Best for most VoIP/videoconference applications where listening quality and intelligibility matter most.
- #define OPUS_APPLICATION_AUDIO

  Best for broadcast/high-fidelity application where the decoded audio should be as close as possible to the input.
- #define OPUS_APPLICATION_RESTRICTED_LOWDELAY

  Only use when lowest-achievable latency is what matters most.

- #define OPUS_APPLICATION_RESTRICTED_SILK 2052

  Experts only: forces SILK encoding; don't allocate CELT state at all.
- #define OPUS_APPLICATION_RESTRICTED_CELT 2053

  Experts only: forces CELT encoding; don't allocate SILK state at all.
- #define OPUS_SIGNAL_VOICE 3001

  Signal being encoded is voice.
- #define OPUS_SIGNAL_MUSIC 3002

  Signal being encoded is music.
- #define OPUS_BANDWIDTH_NARROWBAND

  4 kHz bandpass
- #define OPUS_BANDWIDTH_MEDIUMBAND

  6 kHz bandpass
- #define OPUS_BANDWIDTH_WIDEBAND

  8 kHz bandpass
- #define OPUS_BANDWIDTH_SUPERWIDEBAND

  12 kHz bandpass
- #define OPUS_BANDWIDTH_FULLBAND

  20 kHz bandpass
- #define OPUS_FRAMESIZE_ARG 5000

  Select frame size from the argument (default)
- #define OPUS_FRAMESIZE_2_5_MS 5001

  Use 2.5 ms frames.
- #define OPUS_FRAMESIZE_5_MS 5002

  Use 5 ms frames.
- #define OPUS_FRAMESIZE_10_MS 5003

  Use 10 ms frames.
- #define OPUS_FRAMESIZE_20_MS 5004

  Use 20 ms frames.
- #define OPUS_FRAMESIZE_40_MS 5005

  Use 40 ms frames.
- #define OPUS_FRAMESIZE_60_MS 5006

  Use 60 ms frames.
- #define OPUS_FRAMESIZE_80_MS 5007

  Use 80 ms frames.
- #define OPUS_FRAMESIZE_100_MS 5008

  Use 100 ms frames.
- #define OPUS_FRAMESIZE_120_MS 5009

  Use 120 ms frames.
- #define OPUS_SET_COMPLEXITY(x)

  Configures the encoder's computational complexity.
- #define OPUS_GET_COMPLEXITY(x)

  Gets the encoder's complexity configuration.
- #define OPUS_SET_BITRATE(x)

  Configures the bitrate in the encoder.
- #define OPUS_GET_BITRATE(x)

  Gets the encoder's bitrate configuration.
- #define OPUS_SET_VBR(x)

Enables or disables variable bitrate (VBR) in the encoder.

- #define OPUS_GET_VBR(x)

  Determine if variable bitrate (VBR) is enabled in the encoder.

- #define OPUS_SET_VBR_CONSTRAINT(x)

  Enables or disables constrained VBR in the encoder.

- #define OPUS_GET_VBR_CONSTRAINT(x)

  Determine if constrained VBR is enabled in the encoder.

- #define OPUS_SET_FORCE_CHANNELS(x)

  Configures mono/stereo forcing in the encoder.

- #define OPUS_GET_FORCE_CHANNELS(x)

  Gets the encoder's forced channel configuration.

- #define OPUS_SET_MAX_BANDWIDTH(x)

  Configures the maximum bandpass that the encoder will select automatically.

- #define OPUS_GET_MAX_BANDWIDTH(x)

  Gets the encoder's configured maximum allowed bandpass.

- #define OPUS_SET_BANDWIDTH(x)

  Sets the encoder's bandpass to a specific value.

- #define OPUS_SET_SIGNAL(x)

  Configures the type of signal being encoded.

- #define OPUS_GET_SIGNAL(x)

  Gets the encoder's configured signal type.

- #define OPUS_SET_APPLICATION(x)

  Configures the encoder's intended application.

- #define OPUS_GET_APPLICATION(x)

  Gets the encoder's configured application.

- #define OPUS_GET_LOOKAHEAD(x)

  Gets the total samples of delay added by the entire codec.

- #define OPUS_SET_INBAND_FEC(x)

  Configures the encoder's use of inband forward error correction (FEC).

- #define OPUS_GET_INBAND_FEC(x)

  Gets encoder's configured use of inband forward error correction.

- #define OPUS_SET_PACKET_LOSS_PERC(x)

  Configures the encoder's expected packet loss percentage.

- #define OPUS_GET_PACKET_LOSS_PERC(x)

  Gets the encoder's configured packet loss percentage.

- #define OPUS_SET_DTX(x)

  Configures the encoder's use of discontinuous transmission (DTX).

- #define OPUS_GET_DTX(x)

  Gets encoder's configured use of discontinuous transmission.

- #define OPUS_SET_LSB_DEPTH(x)

  Configures the depth of signal being encoded.

- #define OPUS_GET_LSB_DEPTH(x)

  Gets the encoder's configured signal depth.

- #define OPUS_SET_EXPERT_FRAME_DURATION(x)

  Configures the encoder's use of variable duration frames.

- #define OPUS_GET_EXPERT_FRAME_DURATION(x)

  Gets the encoder's configured use of variable duration frames.

- #define OPUS_SET_PREDICTION_DISABLED(x)

  If set to 1, disables almost all use of prediction, making frames almost completely independent.
- #define OPUS_GET_PREDICTION_DISABLED(x)

  Gets the encoder's configured prediction status.
- #define OPUS_SET_DRED_DURATION(x)

  If non-zero, enables Deep Redundancy (DRED) and use the specified maximum number of 10-ms redundant frames.
- #define OPUS_GET_DRED_DURATION(x)

  Gets the encoder's configured Deep Redundancy (DRED) maximum number of frames.
- #define OPUS_SET_DNN_BLOB(data, len)

  Provide external DNN weights from binary object (only when explicitly built without the weights)
- #define OPUS_SET_QEXT(x)

  If set to 1, enables quality extension (QEXT), otherwise disables it (default).
- #define OPUS_GET_QEXT(x)

  Gets the encoder's configured quality extension (QEXT).
- #define OPUS_RESET_STATE

  Resets the codec state to be equivalent to a freshly initialized state.
- #define OPUS_GET_FINAL_RANGE(x)

  Gets the final state of the codec's entropy coder.
- #define OPUS_GET_BANDWIDTH(x)

  Gets the encoder's configured bandpass or the decoder's last bandpass.
- #define OPUS_GET_SAMPLE_RATE(x)

  Gets the sampling rate the encoder or decoder was initialized with.
- #define OPUS_SET_PHASE_INVERSION_DISABLED(x)

  If set to 1, disables the use of phase inversion for intensity stereo, improving the quality of mono downmixes, but slightly reducing normal stereo quality.
- #define OPUS_GET_PHASE_INVERSION_DISABLED(x)

  Gets the encoder's configured phase inversion status.
- #define OPUS_GET_IN_DTX(x)

  Gets the DTX state of the encoder.
- #define OPUS_SET_GAIN(x)

  Configures decoder gain adjustment.
- #define OPUS_GET_GAIN(x)

  Gets the decoder's configured gain adjustment.
- #define OPUS_GET_LAST_PACKET_DURATION(x)

  Gets the duration (in samples) of the last packet successfully decoded or concealed.
- #define OPUS_GET_PITCH(x)

  Gets the pitch of the last decoded frame, if available.
- #define OPUS_SET_OSCE_BWE(x)

  Enables blind bandwidth extension for wideband signals if decoding sampling rate is 48 kHz.
- #define OPUS_GET_OSCE_BWE(x)

  Gets blind bandwidth extension flag for wideband signals if decoding sampling rate is 48 kHz.
- #define OPUS_SET_IGNORE_EXTENSIONS(x)

  If set to 1, the decoder will ignore all extensions found in the padding area (does not affect DRED, which is decoded separately).
- #define OPUS_GET_IGNORE_EXTENSIONS(x)

  Gets whether the decoder is ignoring extensions.

Functions

- const char * opus_strerror (int error)

  Converts an opus error code into a human readable string.
- const char * opus_get_version_string (void)

  Gets the libopus version string.

5.5.1 Detailed Description

Opus reference implementation constants.

5.6 opus_defines.h

Go to the documentation of this file.

```
00001 /* Copyright (c) 2010-2011 Xiph.Org Foundation, Skype Limited
00002    Written by Jean-Marc Valin and Koen Vos */
00003 /*
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions
00006    are met:
00007
00008    - Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    - Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in the
00013    documentation and/or other materials provided with the distribution.
00014
00015    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00016    ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00017    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00018    A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
00019    OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00020    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00021    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00024    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00025    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026 */
00027
00033 #ifndef OPUS_DEFINES_H
00034 #define OPUS_DEFINES_H
00035
00036 #include "opus_types.h"
00037
00038 #ifdef __cplusplus
00039 extern "C" {
00040 #endif
00041
00046 #define OPUS_OK                0
00048 #define OPUS_BAD_ARG          -1
00050 #define OPUS_BUFFER_TOO_SMALL -2
00052 #define OPUS_INTERNAL_ERROR   -3
00054 #define OPUS_INVALID_PACKET   -4
00056 #define OPUS_UNIMPLEMENTED    -5
00058 #define OPUS_INVALID_STATE    -6
00060 #define OPUS_ALLOC_FAIL       -7
00066 #ifndef OPUS_EXPORT
00067 # if defined(_WIN32)
00068 #  if defined(OPUS_BUILD) && defined(DLL_EXPORT)
00069 #   define OPUS_EXPORT __declspec(dllexport)
00070 #  else
00071 #   define OPUS_EXPORT
00072 #  endif
00073 # elif defined(__GNUC__) && defined(OPUS_BUILD)
00074 #  define OPUS_EXPORT __attribute__ ((visibility ("default")))
```

```
00075 # else
00076 #  define OPUS_EXPORT
00077 # endif
00078 #endif
00079
00080 # if !defined(OPUS_GNUC_PREREQ)
00081 #  if defined(__GNUC__)&&defined(__GNUC_MINOR__)
00082 #   define OPUS_GNUC_PREREQ(_maj,_min) \
00083  ((__GNUC__«16)+__GNUC_MINOR__>=((_maj)«16)+(_min))
00084 #  else
00085 #   define OPUS_GNUC_PREREQ(_maj,_min) 0
00086 #  endif
00087 # endif
00088
00089 #if (!defined(__STDC_VERSION__) || (__STDC_VERSION__ < 199901L) )
00090 # if OPUS_GNUC_PREREQ(3,0)
00091 #  define OPUS_RESTRICT __restrict__
00092 # elif (defined(_MSC_VER) && _MSC_VER >= 1400)
00093 #  define OPUS_RESTRICT __restrict
00094 # else
00095 #  define OPUS_RESTRICT
00096 # endif
00097 #else
00098 # define OPUS_RESTRICT restrict
00099 #endif
00100
00101 #if (!defined(__STDC_VERSION__) || (__STDC_VERSION__ < 199901L) )
00102 # if OPUS_GNUC_PREREQ(2,7)
00103 #  define OPUS_INLINE __inline__
00104 # elif (defined(_MSC_VER))
00105 #  define OPUS_INLINE __inline
00106 # else
00107 #  define OPUS_INLINE
00108 # endif
00109 #else
00110 # define OPUS_INLINE inline
00111 #endif
00112
00116 #if defined(__GNUC__) && OPUS_GNUC_PREREQ(3, 4)
00117 # define OPUS_WARN_UNUSED_RESULT __attribute__ ((__warn_unused_result__))
00118 #else
00119 # define OPUS_WARN_UNUSED_RESULT
00120 #endif
00121 #if !defined(OPUS_BUILD) && defined(__GNUC__) && OPUS_GNUC_PREREQ(3, 4)
00122 # define OPUS_ARG_NONNULL(_x)  __attribute__ ((__nonnull__(_x)))
00123 #else
00124 # define OPUS_ARG_NONNULL(_x)
00125 #endif
00126
00130 #define OPUS_SET_APPLICATION_REQUEST         4000
00131 #define OPUS_GET_APPLICATION_REQUEST         4001
00132 #define OPUS_SET_BITRATE_REQUEST             4002
00133 #define OPUS_GET_BITRATE_REQUEST             4003
00134 #define OPUS_SET_MAX_BANDWIDTH_REQUEST       4004
00135 #define OPUS_GET_MAX_BANDWIDTH_REQUEST       4005
00136 #define OPUS_SET_VBR_REQUEST                 4006
00137 #define OPUS_GET_VBR_REQUEST                 4007
00138 #define OPUS_SET_BANDWIDTH_REQUEST           4008
00139 #define OPUS_GET_BANDWIDTH_REQUEST           4009
00140 #define OPUS_SET_COMPLEXITY_REQUEST          4010
00141 #define OPUS_GET_COMPLEXITY_REQUEST          4011
00142 #define OPUS_SET_INBAND_FEC_REQUEST          4012
00143 #define OPUS_GET_INBAND_FEC_REQUEST          4013
00144 #define OPUS_SET_PACKET_LOSS_PERC_REQUEST    4014
00145 #define OPUS_GET_PACKET_LOSS_PERC_REQUEST    4015
00146 #define OPUS_SET_DTX_REQUEST                 4016
00147 #define OPUS_GET_DTX_REQUEST                 4017
00148 #define OPUS_SET_VBR_CONSTRAINT_REQUEST      4020
00149 #define OPUS_GET_VBR_CONSTRAINT_REQUEST      4021
00150 #define OPUS_SET_FORCE_CHANNELS_REQUEST      4022
00151 #define OPUS_GET_FORCE_CHANNELS_REQUEST      4023
00152 #define OPUS_SET_SIGNAL_REQUEST              4024
00153 #define OPUS_GET_SIGNAL_REQUEST              4025
00154 #define OPUS_GET_LOOKAHEAD_REQUEST           4027
00155 /* #define OPUS_RESET_STATE 4028 */
00156 #define OPUS_GET_SAMPLE_RATE_REQUEST         4029
00157 #define OPUS_GET_FINAL_RANGE_REQUEST         4031
00158 #define OPUS_GET_PITCH_REQUEST               4033
00159 #define OPUS_SET_GAIN_REQUEST                4034
00160 #define OPUS_GET_GAIN_REQUEST                4045 /* Should have been 4035 */
00161 #define OPUS_SET_LSB_DEPTH_REQUEST           4036
```

```
00162 #define OPUS_GET_LSB_DEPTH_REQUEST          4037
00163 #define OPUS_GET_LAST_PACKET_DURATION_REQUEST 4039
00164 #define OPUS_SET_EXPERT_FRAME_DURATION_REQUEST 4040
00165 #define OPUS_GET_EXPERT_FRAME_DURATION_REQUEST 4041
00166 #define OPUS_SET_PREDICTION_DISABLED_REQUEST 4042
00167 #define OPUS_GET_PREDICTION_DISABLED_REQUEST 4043
00168 /* Don't use 4045, it's already taken by OPUS_GET_GAIN_REQUEST */
00169 #define OPUS_SET_PHASE_INVERSION_DISABLED_REQUEST 4046
00170 #define OPUS_GET_PHASE_INVERSION_DISABLED_REQUEST 4047
00171 #define OPUS_GET_IN_DTX_REQUEST             4049
00172 #define OPUS_SET_DRED_DURATION_REQUEST 4050
00173 #define OPUS_GET_DRED_DURATION_REQUEST 4051
00174 #define OPUS_SET_DNN_BLOB_REQUEST 4052
00175 /*#define OPUS_GET_DNN_BLOB_REQUEST 4053 */
00176 #define OPUS_SET_OSCE_BWE_REQUEST 4054
00177 #define OPUS_GET_OSCE_BWE_REQUEST 4055
00178 #define OPUS_SET_QEXT_REQUEST 4056
00179 #define OPUS_GET_QEXT_REQUEST 4057
00180 #define OPUS_SET_IGNORE_EXTENSIONS_REQUEST 4058
00181 #define OPUS_GET_IGNORE_EXTENSIONS_REQUEST 4059
00182
00184 #define OPUS_HAVE_OPUS_PROJECTION_H
00185
00186 /* Macros to trigger compilation errors when the wrong types are provided to a CTL */
00187 #define opus_check_int(x) (((void)((x) == (opus_int32)0)), (opus_int32)(x))
00188
00189 #ifdef DISABLE_PTR_CHECK
00190 /* Disable checks to prevent ubsan from complaining about NULL checks
00191    in test_opus_api. */
00192 #define opus_check_int_ptr(ptr) (ptr)
00193 #define opus_check_uint_ptr(ptr) (ptr)
00194 #define opus_check_uint8_ptr(ptr) (ptr)
00195 #define opus_check_val16_ptr(ptr) (ptr)
00196 #define opus_check_void_ptr(ptr) (ptr)
00197 #else
00198 #define opus_check_int_ptr(ptr) ((ptr) + ((ptr) - (opus_int32*)(ptr)))
00199 #define opus_check_uint_ptr(ptr) ((ptr) + ((ptr) - (opus_uint32*)(ptr)))
00200 #define opus_check_uint8_ptr(ptr) ((ptr) + ((ptr) - (opus_uint8*)(ptr)))
00201 #define opus_check_val16_ptr(ptr) ((ptr) + ((ptr) - (opus_val16*)(ptr)))
00202 #define opus_check_void_ptr(x) ((void)((void *)0 == (x)), (x))
00203 #endif
00210 /* Values for the various encoder CTLs */
00211 #define OPUS_AUTO                           -1000
00212 #define OPUS_BITRATE_MAX                      -1
00216 #define OPUS_APPLICATION_VOIP                2048
00219 #define OPUS_APPLICATION_AUDIO               2049
00222 #define OPUS_APPLICATION_RESTRICTED_LOWDELAY 2051
00224 #define OPUS_APPLICATION_RESTRICTED_SILK     2052
00226 #define OPUS_APPLICATION_RESTRICTED_CELT     2053
00227
00228 #define OPUS_SIGNAL_VOICE                    3001
00229 #define OPUS_SIGNAL_MUSIC                    3002
00230 #define OPUS_BANDWIDTH_NARROWBAND            1101
00231 #define OPUS_BANDWIDTH_MEDIUMBAND            1102
00232 #define OPUS_BANDWIDTH_WIDEBAND              1103
00233 #define OPUS_BANDWIDTH_SUPERWIDEBAND         1104
00234 #define OPUS_BANDWIDTH_FULLBAND              1105
00236 #define OPUS_FRAMESIZE_ARG                   5000
00237 #define OPUS_FRAMESIZE_2_5_MS                5001
00238 #define OPUS_FRAMESIZE_5_MS                  5002
00239 #define OPUS_FRAMESIZE_10_MS                 5003
00240 #define OPUS_FRAMESIZE_20_MS                 5004
00241 #define OPUS_FRAMESIZE_40_MS                 5005
00242 #define OPUS_FRAMESIZE_60_MS                 5006
00243 #define OPUS_FRAMESIZE_80_MS                 5007
00244 #define OPUS_FRAMESIZE_100_MS                5008
00245 #define OPUS_FRAMESIZE_120_MS                5009
00280 #define OPUS_SET_COMPLEXITY(x) OPUS_SET_COMPLEXITY_REQUEST, opus_check_int(x)
00286 #define OPUS_GET_COMPLEXITY(x) OPUS_GET_COMPLEXITY_REQUEST, opus_check_int_ptr(x)
00287
00299 #define OPUS_SET_BITRATE(x) OPUS_SET_BITRATE_REQUEST, opus_check_int(x)
00307 #define OPUS_GET_BITRATE(x) OPUS_GET_BITRATE_REQUEST, opus_check_int_ptr(x)
00308
00322 #define OPUS_SET_VBR(x) OPUS_SET_VBR_REQUEST, opus_check_int(x)
00333 #define OPUS_GET_VBR(x) OPUS_GET_VBR_REQUEST, opus_check_int_ptr(x)
00334
00351 #define OPUS_SET_VBR_CONSTRAINT(x) OPUS_SET_VBR_CONSTRAINT_REQUEST, opus_check_int(x)
00361 #define OPUS_GET_VBR_CONSTRAINT(x) OPUS_GET_VBR_CONSTRAINT_REQUEST, opus_check_int_ptr(x)
00362
00376 #define OPUS_SET_FORCE_CHANNELS(x) OPUS_SET_FORCE_CHANNELS_REQUEST, opus_check_int(x)
00386 #define OPUS_GET_FORCE_CHANNELS(x) OPUS_GET_FORCE_CHANNELS_REQUEST, opus_check_int_ptr(x)
```

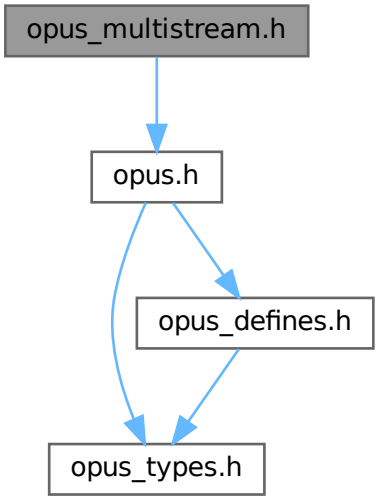
```
00387
00404 #define OPUS_SET_MAX_BANDWIDTH(x) OPUS_SET_MAX_BANDWIDTH_REQUEST, opus_check_int(x)
00405
00417 #define OPUS_GET_MAX_BANDWIDTH(x) OPUS_GET_MAX_BANDWIDTH_REQUEST, opus_check_int_ptr(x)
00418
00436 #define OPUS_SET_BANDWIDTH(x) OPUS_SET_BANDWIDTH_REQUEST, opus_check_int(x)
00437
00448 #define OPUS_SET_SIGNAL(x) OPUS_SET_SIGNAL_REQUEST, opus_check_int(x)
00458 #define OPUS_GET_SIGNAL(x) OPUS_GET_SIGNAL_REQUEST, opus_check_int_ptr(x)
00459
00460
00475 #define OPUS_SET_APPLICATION(x) OPUS_SET_APPLICATION_REQUEST, opus_check_int(x)
00489 #define OPUS_GET_APPLICATION(x) OPUS_GET_APPLICATION_REQUEST, opus_check_int_ptr(x)
00490
00504 #define OPUS_GET_LOOKAHEAD(x) OPUS_GET_LOOKAHEAD_REQUEST, opus_check_int_ptr(x)
00505
00516 #define OPUS_SET_INBAND_FEC(x) OPUS_SET_INBAND_FEC_REQUEST, opus_check_int(x)
00526 #define OPUS_GET_INBAND_FEC(x) OPUS_GET_INBAND_FEC_REQUEST, opus_check_int_ptr(x)
00527
00535 #define OPUS_SET_PACKET_LOSS_PERC(x) OPUS_SET_PACKET_LOSS_PERC_REQUEST, opus_check_int(x)
00541 #define OPUS_GET_PACKET_LOSS_PERC(x) OPUS_GET_PACKET_LOSS_PERC_REQUEST, opus_check_int_ptr(x)
00542
00552 #define OPUS_SET_DTX(x) OPUS_SET_DTX_REQUEST, opus_check_int(x)
00561 #define OPUS_GET_DTX(x) OPUS_GET_DTX_REQUEST, opus_check_int_ptr(x)
00580 #define OPUS_SET_LSB_DEPTH(x) OPUS_SET_LSB_DEPTH_REQUEST, opus_check_int(x)
00586 #define OPUS_GET_LSB_DEPTH(x) OPUS_GET_LSB_DEPTH_REQUEST, opus_check_int_ptr(x)
00587
00611 #define OPUS_SET_EXPERT_FRAME_DURATION(x) OPUS_SET_EXPERT_FRAME_DURATION_REQUEST, opus_check_int(x)
00628 #define OPUS_GET_EXPERT_FRAME_DURATION(x) OPUS_GET_EXPERT_FRAME_DURATION_REQUEST, opus_check_int_ptr(x)
00629
00639 #define OPUS_SET_PREDICTION_DISABLED(x) OPUS_SET_PREDICTION_DISABLED_REQUEST, opus_check_int(x)
00648 #define OPUS_GET_PREDICTION_DISABLED(x) OPUS_GET_PREDICTION_DISABLED_REQUEST, opus_check_int_ptr(x)
00649
00652 #define OPUS_SET_DRED_DURATION(x) OPUS_SET_DRED_DURATION_REQUEST, opus_check_int(x)
00655 #define OPUS_GET_DRED_DURATION(x) OPUS_GET_DRED_DURATION_REQUEST, opus_check_int_ptr(x)
00656
00659 #define OPUS_SET_DNN_BLOB(data, len) OPUS_SET_DNN_BLOB_REQUEST, opus_check_void_ptr(data),
      opus_check_int(len)
00660
00664 #define OPUS_SET_QEXT(x) OPUS_SET_QEXT_REQUEST, opus_check_int(x)
00667 #define OPUS_GET_QEXT(x) OPUS_GET_QEXT_REQUEST, opus_check_int_ptr(x)
00668
00710 #define OPUS_RESET_STATE 4028
00711
00720 #define OPUS_GET_FINAL_RANGE(x) OPUS_GET_FINAL_RANGE_REQUEST, opus_check_uint_ptr(x)
00721
00734 #define OPUS_GET_BANDWIDTH(x) OPUS_GET_BANDWIDTH_REQUEST, opus_check_int_ptr(x)
00735
00742 #define OPUS_GET_SAMPLE_RATE(x) OPUS_GET_SAMPLE_RATE_REQUEST, opus_check_int_ptr(x)
00743
00757 #define OPUS_SET_PHASE_INVERSION_DISABLED(x) OPUS_SET_PHASE_INVERSION_DISABLED_REQUEST, opus_check_int(x)
00766 #define OPUS_GET_PHASE_INVERSION_DISABLED(x) OPUS_GET_PHASE_INVERSION_DISABLED_REQUEST,
      opus_check_int_ptr(x)
00776 #define OPUS_GET_IN_DTX(x) OPUS_GET_IN_DTX_REQUEST, opus_check_int_ptr(x)
00777
00795 #define OPUS_SET_GAIN(x) OPUS_SET_GAIN_REQUEST, opus_check_int(x)
00800 #define OPUS_GET_GAIN(x) OPUS_GET_GAIN_REQUEST, opus_check_int_ptr(x)
00801
00805 #define OPUS_GET_LAST_PACKET_DURATION(x) OPUS_GET_LAST_PACKET_DURATION_REQUEST, opus_check_int_ptr(x)
00806
00817 #define OPUS_GET_PITCH(x) OPUS_GET_PITCH_REQUEST, opus_check_int_ptr(x)
00818
00824  #define OPUS_SET_OSCE_BWE(x) OPUS_SET_OSCE_BWE_REQUEST, opus_check_int(x)
00829  #define OPUS_GET_OSCE_BWE(x) OPUS_GET_OSCE_BWE_REQUEST, opus_check_int_ptr(x)
00830
00834 #define OPUS_SET_IGNORE_EXTENSIONS(x) OPUS_SET_IGNORE_EXTENSIONS_REQUEST, opus_check_int(x)
00837 #define OPUS_GET_IGNORE_EXTENSIONS(x) OPUS_GET_IGNORE_EXTENSIONS_REQUEST, opus_check_int_ptr(x)
00838
00850 OPUS_EXPORT const char *opus_strerror(int error);
00851
00860 OPUS_EXPORT const char *opus_get_version_string(void);
00863 #ifdef __cplusplus
00864 }
00865 #endif
00866
00867 #endif /* OPUS_DEFINES_H */
```

5.7 opus_multistream.h File Reference

Opus reference implementation multistream API.

```
#include "opus.h"
```

Include dependency graph for opus_multistream.h:



Macros

- #define OPUS_MULTISTREAM_GET_ENCODER_STATE(x, y)

  Gets the encoder state for an individual stream of a multistream encoder.
- #define OPUS_MULTISTREAM_GET_DECODER_STATE(x, y)

  Gets the decoder state for an individual stream of a multistream decoder.

Typedefs

- typedef struct OpusMSEncoder OpusMSEncoder

  Opus multistream encoder state.
- typedef struct OpusMSDecoder OpusMSDecoder

  Opus multistream decoder state.

Functions

Multistream encoder functions

- opus_int32 opus_multistream_encoder_get_size (int streams, int coupled_streams)
  Gets the size of an OpusMSEncoder structure.
- opus_int32 opus_multistream_surround_encoder_get_size (int channels, int mapping_family)
- OpusMSEncoder * opus_multistream_encoder_create (opus_int32 Fs, int channels, int streams, int coupled_streams, const unsigned char *mapping, int application, int *error)
  Allocates and initializes a multistream encoder state.
- OpusMSEncoder * opus_multistream_surround_encoder_create (opus_int32 Fs, int channels, int mapping_family, int *streams, int *coupled_streams, unsigned char *mapping, int application, int *error)
- int opus_multistream_encoder_init (OpusMSEncoder *st, opus_int32 Fs, int channels, int streams, int coupled_streams, const unsigned char *mapping, int application)
  Initialize a previously allocated multistream encoder state.
- int opus_multistream_surround_encoder_init (OpusMSEncoder *st, opus_int32 Fs, int channels, int mapping_family, int *streams, int *coupled_streams, unsigned char *mapping, int application)
- int opus_multistream_encode (OpusMSEncoder *st, const opus_int16 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)
  Encodes a multistream Opus frame.
- int opus_multistream_encode24 (OpusMSEncoder *st, const opus_int32 *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)
  Encodes a multistream Opus frame.
- int opus_multistream_encode_float (OpusMSEncoder *st, const float *pcm, int frame_size, unsigned char *data, opus_int32 max_data_bytes)
  Encodes a multistream Opus frame from floating point input.
- void opus_multistream_encoder_destroy (OpusMSEncoder *st)
  Frees an `OpusMSEncoder` allocated by opus_multistream_encoder_create().
- int opus_multistream_encoder_ctl (OpusMSEncoder *st, int request,...)
  Perform a CTL function on a multistream Opus encoder.

Multistream decoder functions

- opus_int32 opus_multistream_decoder_get_size (int streams, int coupled_streams)
  Gets the size of an `OpusMSDecoder` structure.
- OpusMSDecoder * opus_multistream_decoder_create (opus_int32 Fs, int channels, int streams, int coupled_streams, const unsigned char *mapping, int *error)
  Allocates and initializes a multistream decoder state.
- int opus_multistream_decoder_init (OpusMSDecoder *st, opus_int32 Fs, int channels, int streams, int coupled_streams, const unsigned char *mapping)
  Initialize a previously allocated decoder state object.
- int opus_multistream_decode (OpusMSDecoder *st, const unsigned char *data, opus_int32 len, opus_int16 *pcm, int frame_size, int decode_fec)
  Decode a multistream Opus packet.
- int opus_multistream_decode24 (OpusMSDecoder *st, const unsigned char *data, opus_int32 len, opus_int32 *pcm, int frame_size, int decode_fec)
  Decode a multistream Opus packet.
- int opus_multistream_decode_float (OpusMSDecoder *st, const unsigned char *data, opus_int32 len, float *pcm, int frame_size, int decode_fec)
  Decode a multistream Opus packet with floating point output.
- int opus_multistream_decoder_ctl (OpusMSDecoder *st, int request,...)
  Perform a CTL function on a multistream Opus decoder.
- void opus_multistream_decoder_destroy (OpusMSDecoder *st)
  Frees an `OpusMSDecoder` allocated by opus_multistream_decoder_create().

5.7.1 Detailed Description

Opus reference implementation multistream API.

5.8 opus_multistream.h

Go to the documentation of this file.

```
00001 /* Copyright (c) 2011 Xiph.Org Foundation
00002    Written by Jean-Marc Valin */
00003 /*
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions
00006    are met:
00007
00008    - Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    - Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in the
00013    documentation and/or other materials provided with the distribution.
00014
00015    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00016    ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00017    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00018    A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
00019    OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00020    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00021    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00024    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00025    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026 */
00027
00033 #ifndef OPUS_MULTISTREAM_H
00034 #define OPUS_MULTISTREAM_H
00035
00036 #include "opus.h"
00037
00038 #ifdef __cplusplus
00039 extern "C" {
00040 #endif
00041
00047 #define opus_check_encstate_ptr(ptr) ((ptr) + ((ptr) - (OpusEncoder**)(ptr)))
00048 #define opus_check_decstate_ptr(ptr) ((ptr) + ((ptr) - (OpusDecoder**)(ptr)))
00055 #define OPUS_MULTISTREAM_GET_ENCODER_STATE_REQUEST 5120
00056 #define OPUS_MULTISTREAM_GET_DECODER_STATE_REQUEST 5122
00086 #define OPUS_MULTISTREAM_GET_ENCODER_STATE(x,y) OPUS_MULTISTREAM_GET_ENCODER_STATE_REQUEST,
      opus_check_int(x), opus_check_encstate_ptr(y)
00087
00099 #define OPUS_MULTISTREAM_GET_DECODER_STATE(x,y) OPUS_MULTISTREAM_GET_DECODER_STATE_REQUEST,
      opus_check_int(x), opus_check_decstate_ptr(y)
00100
00175 typedef struct OpusMSEncoder OpusMSEncoder;
00176
00183 typedef struct OpusMSDecoder OpusMSDecoder;
00184
00203 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_multistream_encoder_get_size(
00204       int streams,
00205       int coupled_streams
00206 );
00207
00208 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_multistream_surround_encoder_get_size(
00209       int channels,
00210       int mapping_family
00211 );
00212
00213
00257 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusMSEncoder *opus_multistream_encoder_create(
00258       opus_int32 Fs,
00259       int channels,
00260       int streams,
00261       int coupled_streams,
```

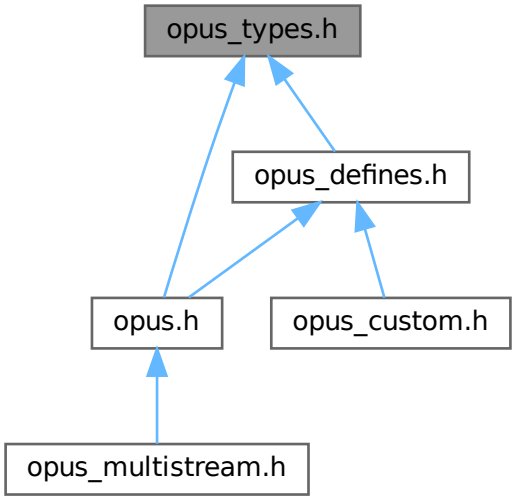
```
00262       const unsigned char *mapping,
00263       int application,
00264       int *error
00265 ) OPUS_ARG_NONNULL(5);
00266
00267 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusMSEncoder *opus_multistream_surround_encoder_create(
00268       opus_int32 Fs,
00269       int channels,
00270       int mapping_family,
00271       int *streams,
00272       int *coupled_streams,
00273       unsigned char *mapping,
00274       int application,
00275       int *error
00276 ) OPUS_ARG_NONNULL(4) OPUS_ARG_NONNULL(5) OPUS_ARG_NONNULL(6);
00277
00326 OPUS_EXPORT int opus_multistream_encoder_init(
00327       OpusMSEncoder *st,
00328       opus_int32 Fs,
00329       int channels,
00330       int streams,
00331       int coupled_streams,
00332       const unsigned char *mapping,
00333       int application
00334 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(6);
00335
00336 OPUS_EXPORT int opus_multistream_surround_encoder_init(
00337       OpusMSEncoder *st,
00338       opus_int32 Fs,
00339       int channels,
00340       int mapping_family,
00341       int *streams,
00342       int *coupled_streams,
00343       unsigned char *mapping,
00344       int application
00345 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(5) OPUS_ARG_NONNULL(6) OPUS_ARG_NONNULL(7);
00346
00377 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_multistream_encode(
00378     OpusMSEncoder *st,
00379     const opus_int16 *pcm,
00380     int frame_size,
00381     unsigned char *data,
00382     opus_int32 max_data_bytes
00383 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00384
00415 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_multistream_encode24(
00416     OpusMSEncoder *st,
00417     const opus_int32 *pcm,
00418     int frame_size,
00419     unsigned char *data,
00420     opus_int32 max_data_bytes
00421 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00422
00460 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_multistream_encode_float(
00461       OpusMSEncoder *st,
00462       const float *pcm,
00463       int frame_size,
00464       unsigned char *data,
00465       opus_int32 max_data_bytes
00466 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(2) OPUS_ARG_NONNULL(4);
00467
00472 OPUS_EXPORT void opus_multistream_encoder_destroy(OpusMSEncoder *st);
00473
00486 OPUS_EXPORT int opus_multistream_encoder_ctl(OpusMSEncoder *st, int request, ...) OPUS_ARG_NONNULL(1);
00487
00508 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT opus_int32 opus_multistream_decoder_get_size(
00509       int streams,
00510       int coupled_streams
00511 );
00512
00542 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusMSDecoder *opus_multistream_decoder_create(
00543       opus_int32 Fs,
00544       int channels,
00545       int streams,
00546       int coupled_streams,
00547       const unsigned char *mapping,
00548       int *error
00549 ) OPUS_ARG_NONNULL(5);
00550
00585 OPUS_EXPORT int opus_multistream_decoder_init(
00586       OpusMSDecoder *st,
```

```
00587       opus_int32 Fs,
00588       int channels,
00589       int streams,
00590       int coupled_streams,
00591       const unsigned char *mapping
00592 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(6);
00593
00623 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_multistream_decode(
00624     OpusMSDecoder *st,
00625     const unsigned char *data,
00626     opus_int32 len,
00627     opus_int16 *pcm,
00628     int frame_size,
00629     int decode_fec
00630 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00631
00661 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_multistream_decode24(
00662     OpusMSDecoder *st,
00663     const unsigned char *data,
00664     opus_int32 len,
00665     opus_int32 *pcm,
00666     int frame_size,
00667     int decode_fec
00668 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00669
00699 OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_multistream_decode_float(
00700     OpusMSDecoder *st,
00701     const unsigned char *data,
00702     opus_int32 len,
00703     float *pcm,
00704     int frame_size,
00705     int decode_fec
00706 ) OPUS_ARG_NONNULL(1) OPUS_ARG_NONNULL(4);
00707
00720 OPUS_EXPORT int opus_multistream_decoder_ctl(OpusMSDecoder *st, int request, ...) OPUS_ARG_NONNULL(1);
00721
00726 OPUS_EXPORT void opus_multistream_decoder_destroy(OpusMSDecoder *st);
00727
00732 #ifdef __cplusplus
00733 }
00734 #endif
00735
00736 #endif /* OPUS_MULTISTREAM_H */
```

5.9 opus_types.h File Reference

Opus reference implementation types.

This graph shows which files directly or indirectly include this file:



Macros

- #define opus_int int /∗ used for counters etc; at least 16 bits ∗/
- #define opus_int64 long long
- #define opus_int8 signed char
- #define opus_uint unsigned int /∗ used for counters etc; at least 16 bits ∗/
- #define opus_uint64 unsigned long long
- #define opus_uint8 unsigned char

Typedefs

- typedef short opus_int16
- typedef unsigned short opus_uint16
- typedef int opus_int32
- typedef unsigned int opus_uint32

5.9.1 Detailed Description

Opus reference implementation types.

5.9.2 Macro Definition Documentation

5.9.2.1 opus_int

```
#define opus_int int /* used for counters etc; at least 16 bits */
```

5.9.2.2 opus_int64

```
#define opus_int64 long long
```

5.9.2.3 opus_int8

```
#define opus_int8 signed char
```

5.9.2.4 opus_uint

```
#define opus_uint unsigned int /* used for counters etc; at least 16 bits */
```

5.9.2.5 opus_uint64

```
#define opus_uint64 unsigned long long
```

5.9.2.6 opus_uint8

```
#define opus_uint8 unsigned char
```

5.9.3 Typedef Documentation

5.9.3.1 opus_int16

```
typedef short opus_int16
```

5.9.3.2 opus_int32

```
typedef int opus_int32
```

5.9.3.3 opus_uint16

```
typedef unsigned short opus_uint16
```

5.9.3.4 opus_uint32

```
typedef unsigned int opus_uint32
```

5.10 opus_types.h

Go to the documentation of this file.

```
00001 /* (C) COPYRIGHT 1994-2002 Xiph.Org Foundation */
00002 /* Modified by Jean-Marc Valin */
00003 /*
00004    Redistribution and use in source and binary forms, with or without
00005    modification, are permitted provided that the following conditions
00006    are met:
00007
00008    - Redistributions of source code must retain the above copyright
00009    notice, this list of conditions and the following disclaimer.
00010
00011    - Redistributions in binary form must reproduce the above copyright
00012    notice, this list of conditions and the following disclaimer in the
00013    documentation and/or other materials provided with the distribution.
00014
00015    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00016    ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00017    LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00018    A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
00019    OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00020    EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00021    PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022    PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023    LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00024    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00025    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026 */
00027 /* opus_types.h based on ogg_types.h from libogg */
00028
00033 #ifndef OPUS_TYPES_H
00034 #define OPUS_TYPES_H
00035
00036 #define opus_int         int                     /* used for counters etc; at least 16 bits */
00037 #define opus_int64       long long
00038 #define opus_int8        signed char
00039
00040 #define opus_uint        unsigned int            /* used for counters etc; at least 16 bits */
00041 #define opus_uint64      unsigned long long
00042 #define opus_uint8       unsigned char
00043
00044 /* Use the real stdint.h if it's there (taken from Paul Hsieh's pstdint.h) */
00045 #if (defined(__STDC__) && __STDC__ && defined(__STDC_VERSION__) && __STDC_VERSION__ >= 199901L) ||
      (defined(__GNUC__) && (defined(_STDINT_H) || defined(_STDINT_H_)) || defined (HAVE_STDINT_H))
00046 #include <stdint.h>
00047 #  undef opus_int64
00048 #  undef opus_int8
00049 #  undef opus_uint64
00050 #  undef opus_uint8
00051    typedef int8_t opus_int8;
00052    typedef uint8_t opus_uint8;
00053    typedef int16_t opus_int16;
00054    typedef uint16_t opus_uint16;
00055    typedef int32_t opus_int32;
00056    typedef uint32_t opus_uint32;
00057    typedef int64_t opus_int64;
00058    typedef uint64_t opus_uint64;
00059 #elif defined(_WIN32)
00060
00061 #  if defined(__CYGWIN__)
00062 #    include <_G_config.h>
00063      typedef _G_int32_t opus_int32;
00064      typedef _G_uint32_t opus_uint32;
00065      typedef _G_int16 opus_int16;
00066      typedef _G_uint16 opus_uint16;
00067 #  elif defined(__MINGW32__)
00068      typedef short opus_int16;
00069      typedef unsigned short opus_uint16;
```

```
00070       typedef int opus_int32;
00071       typedef unsigned int opus_uint32;
00072 #  elif defined(__MWERKS__)
00073       typedef int opus_int32;
00074       typedef unsigned int opus_uint32;
00075       typedef short opus_int16;
00076       typedef unsigned short opus_uint16;
00077 #  else
00078       /* MSVC/Borland */
00079       typedef __int32 opus_int32;
00080       typedef unsigned __int32 opus_uint32;
00081       typedef __int16 opus_int16;
00082       typedef unsigned __int16 opus_uint16;
00083 #  endif
00084
00085 #elif defined(__MACOS__)
00086
00087 #  include <sys/types.h>
00088    typedef SInt16 opus_int16;
00089    typedef UInt16 opus_uint16;
00090    typedef SInt32 opus_int32;
00091    typedef UInt32 opus_uint32;
00092
00093 #elif (defined(__APPLE__) && defined(__MACH__)) /* MacOS X Framework build */
00094
00095 #  include <sys/types.h>
00096    typedef int16_t opus_int16;
00097    typedef u_int16_t opus_uint16;
00098    typedef int32_t opus_int32;
00099    typedef u_int32_t opus_uint32;
00100
00101 #elif defined(__BEOS__)
00102
00103    /* Be */
00104 #  include <inttypes.h>
00105    typedef int16 opus_int16;
00106    typedef u_int16 opus_uint16;
00107    typedef int32_t opus_int32;
00108    typedef u_int32_t opus_uint32;
00109
00110 #elif defined (__EMX__)
00111
00112    /* OS/2 GCC */
00113    typedef short opus_int16;
00114    typedef unsigned short opus_uint16;
00115    typedef int opus_int32;
00116    typedef unsigned int opus_uint32;
00117
00118 #elif defined (DJGPP)
00119
00120    /* DJGPP */
00121    typedef short opus_int16;
00122    typedef unsigned short opus_uint16;
00123    typedef int opus_int32;
00124    typedef unsigned int opus_uint32;
00125
00126 #elif defined(R5900)
00127
00128    /* PS2 EE */
00129    typedef int opus_int32;
00130    typedef unsigned opus_uint32;
00131    typedef short opus_int16;
00132    typedef unsigned short opus_uint16;
00133
00134 #elif defined(__SYMBIAN32__)
00135
00136    /* Symbian GCC */
00137    typedef signed short opus_int16;
00138    typedef unsigned short opus_uint16;
00139    typedef signed int opus_int32;
00140    typedef unsigned int opus_uint32;
00141
00142 #elif defined(CONFIG_TI_C54X) || defined (CONFIG_TI_C55X)
00143
00144    typedef short opus_int16;
00145    typedef unsigned short opus_uint16;
00146    typedef long opus_int32;
00147    typedef unsigned long opus_uint32;
00148
00149 #elif defined(CONFIG_TI_C6X)
00150
```

```
00151    typedef short opus_int16;
00152    typedef unsigned short opus_uint16;
00153    typedef int opus_int32;
00154    typedef unsigned int opus_uint32;
00155
00156 #else
00157
00158    /* Give up, take a reasonable guess */
00159    typedef short opus_int16;
00160    typedef unsigned short opus_uint16;
00161    typedef int opus_int32;
00162    typedef unsigned int opus_uint32;
00163
00164 #endif
00165
00166 #endif  /* OPUS_TYPES_H */
```

Index