Opus

0.9.14

Generated by Doxygen 1.7.1

Thu May 17 2012 15:22:05

Contents

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 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
- 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:

- Opus Encoder
- Opus Decoder
- Repacketizer
- Opus library information functions

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 Opus Encoder

Since Opus is a stateful codec, the encoding process starts with creating an encoder state.

Typedefs

- typedef struct OpusEncoder OpusEncoder

    Opus encoder state.

Functions

- int opus_encoder_get_size (int channels)
- 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 the size returned by opus_encoder_get_size.

- int opus_encode (OpusEncoder ∗st, const opus_int16 ∗pcm, int frame_size, unsigned char ∗data, int max_data_bytes)

    Encodes an Opus frame.

- int opus_encode_float (OpusEncoder ∗st, const float ∗pcm, int frame_size, unsigned char ∗data, int 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

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 assuptions 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 (1276 bytes is recommended)

opus_encode() and opus_encode_frame() 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 1 byte, 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 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 int opus_encode (OpusEncoder ∗ *st*, const opus_int16 ∗ *pcm*, int *frame_size*, unsigned char ∗ *data*, int *max_data_bytes*)

Encodes an Opus frame.

The passed frame_size must an opus frame size for the encoder's sampling rate. For example, at 48kHz the permitted values are 120, 240, 480, or 960. Passing in a duration of less than 10ms (480 samples at 48kHz) will prevent the encoder from using the LPC or hybrid modes.

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 frame of input signal

`[out]` ***data*** `char*`: Output payload (at least max_data_bytes long)

`[in]` ***max_data_bytes*** `int`: Allocated memory for payload; don't use for controlling bitrate

Returns

length of the data payload (in bytes) or Error codes

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

Encodes an Opus frame from floating point input.

The passed frame_size must an opus frame size for the encoder's sampling rate. For example, at 48kHz the permitted values are 120, 240, 480, or 960. Passing in a duration of less than 10ms (480 samples at 48kHz) will prevent the encoder from using the LPC or hybrid modes.

Parameters

`[in]` ***st*** `OpusEncoder*`: Encoder state

`[in]` ***pcm*** `float*`: Input signal (interleaved if 2 channels). length is frame_-size∗channels∗sizeof(float)

`[in]` ***frame_size*** `int`: Number of samples per frame of input signal

`[out]` ***data*** `char*`: Output payload (at least max_data_bytes long)

`[in]` ***max_data_bytes*** `int`: Allocated memory for payload; don't use for controlling bitrate

Returns

length of the data payload (in bytes) or Error codes

4.1.3.3 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 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)
- `[in]` ***channels*** `int`: Number of channels (1/2) in input signal
- `[in]` ***application*** `int`: Coding mode (OPUS_APPLICATION_VOIP/OPUS_APPLICATION_AUDIO/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 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.

4.1.3.4 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.

See also

Encoder related CTLs

4.1.3.5 void opus_encoder_destroy (OpusEncoder ∗ *st*)

Frees an OpusEncoder allocated by opus_encoder_create.

Parameters

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

4.1.3.6 int opus_encoder_get_size (int *channels*)

4.1.3.7 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 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)

`[in]` ***channels*** `int`: Number of channels (1/2) in input signal

`[in]` ***application*** `int`: Coding mode (OPUS_APPLICATION_VOIP/OPUS_-APPLICATION_AUDIO/OPUS_APPLICATION_RESTRICTED_-LOWDELAY)

Return values

OPUS_OK Success or Error codes

4.2 Opus Decoder

The decoding process also starts with creating a decoder state.

Typedefs

- typedef struct OpusDecoder OpusDecoder

  Opus decoder 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, int len, opus_-int16 ∗pcm, int frame_size, int decode_fec)

  Decode an Opus frame.

- int opus_decode_float (OpusDecoder ∗st, const unsigned char ∗data, int len, float ∗pcm, int frame_size, int decode_fec)

  Decode an opus frame 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_packet_parse (const unsigned char ∗data, int len, unsigned char ∗out_-toc, const unsigned char ∗frames[48], short 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[ ], int len)

  Gets the number of frames in an Opus packet.

- int opus_decoder_get_nb_samples (const OpusDecoder ∗dec, const unsigned char packet[ ], int len)

  Gets the number of samples of an Opus packet.

4.2.1 Detailed Description

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 or 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 assuptions 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(enc, packet, len, decoded, max_size);
```

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

4.2.2 Typedef Documentation

4.2.2.1 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

4.2.3 Function Documentation

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

Decode an Opus frame.

Parameters

`[in]` ***st*** `OpusDecoder*`: 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, if less than the maximum frame size (120ms) some frames can not be decoded

`[in]` ***decode_fec*** `int`: Flag (0/1) 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

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

Decode an opus frame 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*** `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, if less than the maximum frame size (120ms) some frames can not be decoded

`[in]` ***decode_fec*** `int`: Flag (0/1) 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 OpusDecoder* opus_decoder_create (opus_int32 *Fs*, int *channels*, int * *error*)

Allocates and initializes a decoder state.

Parameters

`[in]` ***Fs*** `opus_int32`: Sampling rate of input signal (Hz)

`[in]` ***channels*** `int`: Number of channels (1/2) in input signal

`[out]` ***error*** `int*`: OPUS_OK Success or Error codes

4.2.3.4 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.

See also

Generic CTLs

4.2.3.5 void opus_decoder_destroy (OpusDecoder * *st*)

Frees an OpusDecoder allocated by opus_decoder_create.

Parameters

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

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

Gets the number of samples of an Opus packet.

Parameters

`[in]` ***dec*** `OpusDecoder*`: Decoder state

`[in]` ***packet*** `char*`: Opus packet

`[in]` ***len*** `int`: Length of packet

Returns

Number of samples

Return values

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

4.2.3.7 int opus_decoder_get_size (int *channels*)

Gets the size of an OpusDecoder structure.

Parameters

`[in]` ***channels*** `int`: Number of channels

Returns

size

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

Initializes a previously allocated decoder state.

The state must be 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 of input signal (Hz)

`[in]` ***channels*** `int`: Number of channels (1/2) in input signal

Return values

OPUS_OK Success or Error codes

4.2.3.9 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

4.2.3.10 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

4.2.3.11 int opus_packet_get_nb_frames (const unsigned char *packet[ ]*, int *len*)

Gets the number of frames in an Opus packet.

Parameters

`[in]` ***packet*** `char*`: Opus packet

`[in]` ***len*** `int`: Length of packet

Returns

Number of frames

Return values

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

4.2.3.12 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

`[in]` ***Fs*** `opus_int32`: Sampling rate in Hz

Returns

Number of samples per frame

Return values

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

4.2.3.13 int opus_packet_parse (const unsigned char ∗ *data*, int *len*, unsigned char ∗ *out_toc*, const unsigned char ∗ *frames[48]*, short *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*** `int`: size of data

`[out]` ***out_toc*** `char*`: TOC pointer

`[out]` ***frames*** `char*[48]` encapsulated frames

`[out]` ***size*** `short[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

4.3 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)
- OpusRepacketizer ∗ opus_repacketizer_init (OpusRepacketizer ∗rp)
- OpusRepacketizer ∗ opus_repacketizer_create (void)
- void opus_repacketizer_destroy (OpusRepacketizer ∗rp)
- int opus_repacketizer_cat (OpusRepacketizer ∗rp, const unsigned char ∗data, int len)
- opus_int32 opus_repacketizer_out_range (OpusRepacketizer ∗rp, int begin, int end, unsigned char ∗data, int maxlen)
- int opus_repacketizer_get_nb_frames (OpusRepacketizer ∗rp)
- opus_int32 opus_repacketizer_out (OpusRepacketizer ∗rp, unsigned char ∗data, int maxlen)

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.

4.3.2 Typedef Documentation

4.3.2.1 typedef struct OpusRepacketizer OpusRepacketizer

4.3.3 Function Documentation

4.3.3.1 int opus_repacketizer_cat (OpusRepacketizer ∗ *rp*, const unsigned char ∗ *data*, int *len*)

4.3.3.2 OpusRepacketizer∗ opus_repacketizer_create (void)

4.3.3.3 void opus_repacketizer_destroy (OpusRepacketizer ∗ *rp*)

4.3.3.4 int opus_repacketizer_get_nb_frames (OpusRepacketizer ∗ *rp*)

4.3.3.5 int opus_repacketizer_get_size (void)

4.3.3.6 OpusRepacketizer∗ opus_repacketizer_init (OpusRepacketizer ∗ *rp*)

4.3.3.7 opus_int32 opus_repacketizer_out (OpusRepacketizer ∗ *rp*, unsigned char ∗ *data*, int *maxlen*)

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

4.4 Error codes

Defines

- #define OPUS_OK

  No error.

- #define OPUS_BAD_ARG

  One or more invalid/out of range arguments.

- #define OPUS_BUFFER_TOO_SMALL

  The mode struct passed is invalid.

- #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 Define Documentation

4.4.1.1 #define OPUS_ALLOC_FAIL

Memory allocation has failed.

4.4.1.2 #define OPUS_BAD_ARG

One or more invalid/out of range arguments.

4.4.1.3 #define OPUS_BUFFER_TOO_SMALL

The mode struct passed is invalid.

4.4.1.4 #define OPUS_INTERNAL_ERROR

An internal error was detected.

4.4.1.5 #define OPUS_INVALID_PACKET

The compressed data passed is corrupted.

4.4.1.6 #define OPUS_INVALID_STATE

An encoder or decoder structure is invalid or already freed.

4.4.1.7 #define OPUS_OK

No error.

4.4.1.8 #define OPUS_UNIMPLEMENTED

Invalid/unsupported request number.

4.5 Pre-defined values for CTL interface

Defines

- #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_SIGNAL_VOICE 3001

  Signal being encoded is voice.

- #define OPUS_SIGNAL_MUSIC 3002

  Signal being encoded is music.

- #define OPUS_BANDWIDTH_NARROWBAND

  4kHz bandpass

- #define OPUS_BANDWIDTH_MEDIUMBAND

  6kHz bandpass

- #define OPUS_BANDWIDTH_WIDEBAND

  8kHz bandpass

- #define OPUS_BANDWIDTH_SUPERWIDEBAND

  12kHz bandpass

- #define OPUS_BANDWIDTH_FULLBAND

  20kHz bandpass

4.5.1 Detailed Description

See also

Generic CTLs, Encoder related CTLs

4.5.2 Define Documentation

4.5.2.1 #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 #define OPUS_APPLICATION_RESTRICTED_LOWDELAY

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

Voice-optimized modes cannot be used.

4.5.2.3 #define OPUS_APPLICATION_VOIP

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

4.5.2.4 #define OPUS_AUTO

Auto/default setting.

4.5.2.5 #define OPUS_BANDWIDTH_FULLBAND

20kHz bandpass

4.5.2.6 #define OPUS_BANDWIDTH_MEDIUMBAND

6kHz bandpass

4.5.2.7 #define OPUS_BANDWIDTH_NARROWBAND

4kHz bandpass

4.5.2.8 #define OPUS_BANDWIDTH_SUPERWIDEBAND

12kHz bandpass

4.5.2.9 #define OPUS_BANDWIDTH_WIDEBAND

8kHz bandpass

4.5.2.10 #define OPUS_BITRATE_MAX

Maximum bitrate.

4.5.2.11 #define OPUS_SIGNAL_MUSIC 3002

Signal being encoded is music.

4.5.2.12 #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.

Defines

- #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)

  Configures VBR in the encoder.

- #define OPUS_GET_VBR(x)

  Gets the encoder's VBR configuration,.

- #define OPUS_SET_VBR_CONSTRAINT(x)

  Configures constrained VBR in the encoder.

- #define OPUS_GET_VBR_CONSTRAINT(x)

  Gets the encoder's constrained VBR configuration.

- #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 encoder's maximum bandpass allowed,.
- #define OPUS_GET_MAX_BANDWIDTH(x)

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

  Configures the encoder's bandpass,.
- #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.
- #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.
- #define OPUS_GET_DTX(x)

  Gets encoder's configured use of discontinuous transmission,.

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;

int 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 Define Documentation

4.6.2.1 #define OPUS_GET_APPLICATION(*x*)

Gets the encoder's configured application,.

See also

OPUS_SET_APPLICATION

Parameters

`[out]` ***x*** `int*`: Application value

4.6.2.2 #define OPUS_GET_BITRATE(*x*)

Gets the encoder's bitrate configuration,.

See also

OPUS_SET_BITRATE

Parameters

`[out]` ***x*** `opus_int32*`: bitrate in bits per second.

4.6.2.3 #define OPUS_GET_COMPLEXITY(*x*)

Gets the encoder's complexity configuration,.

See also

OPUS_SET_COMPLEXITY

Parameters

`[out]` ***x*** `int*`: 0-10, inclusive

4.6.2.4 #define OPUS_GET_DTX(*x*)

Gets encoder's configured use of discontinuous transmission,.

See also

OPUS_SET_DTX

Parameters

`[out]` ***x*** `int*`: DTX flag

4.6.2.5 #define OPUS_GET_FORCE_CHANNELS(*x*)

Gets the encoder's forced channel configuration,.

See also

OPUS_SET_FORCE_CHANNELS

Parameters

`[out]` ***x*** `int*`: OPUS_AUTO; 0; 1

4.6.2.6 #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*** `int*`: FEC flag

4.6.2.7 #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*** `int*`: Number of lookahead samples

4.6.2.8 #define OPUS_GET_MAX_BANDWIDTH(*x*)

Gets the encoder's configured maximum bandpass allowed,.

See also

OPUS_SET_MAX_BANDWIDTH

Parameters

`[out]` ***x*** `int*`: Bandwidth value

4.6.2.9 #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*** `int*`: Loss percentage in the range 0-100, inclusive.

4.6.2.10 #define OPUS_GET_SIGNAL(*x*)

Gets the encoder's configured signal type,.

See also

OPUS_SET_SIGNAL

Parameters

`[out]` ***x*** `int*`: Signal type

4.6.2.11 #define OPUS_GET_VBR(*x*)

Gets the encoder's VBR configuration,.

See also

OPUS_SET_VBR

Parameters

`[out]` ***x*** `int*`: 0; 1

4.6.2.12 #define OPUS_GET_VBR_CONSTRAINT(*x*)

Gets the encoder's constrained VBR configuration.

See also

OPUS_SET_VBR_CONSTRAINT

Parameters

`[out]` ***x*** `int*`: 0; 1

4.6.2.13 #define OPUS_SET_APPLICATION(*x*)

Configures the encoder's intended application.

The initial value is a mandatory argument to the encoder_create function. The supported values are:

- 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

Parameters

`[in]` ***x*** `int`: Application value

4.6.2.14 #define OPUS_SET_BANDWIDTH(*x*)

Configures the encoder's bandpass,.

See also

OPUS_GET_BANDWIDTH The supported values are:

- OPUS_AUTO (default)
- OPUS_BANDWIDTH_NARROWBAND 4kHz passband
- OPUS_BANDWIDTH_MEDIUMBAND 6kHz passband
- OPUS_BANDWIDTH_WIDEBAND 8kHz passband
- OPUS_BANDWIDTH_SUPERWIDEBAND 12kHz passband
- OPUS_BANDWIDTH_FULLBAND 20kHz passband

Parameters

`[in]` ***x*** `int`: Bandwidth value

4.6.2.15 #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_BITRATE_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.

Parameters

`[in]` ***x*** `opus_int32`: bitrate in bits per second.

4.6.2.16 #define OPUS_SET_COMPLEXITY(*x*)

Configures the encoder's computational complexity.

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

Parameters

`[in]` ***x*** `int`: 0-10, inclusive

4.6.2.17 #define OPUS_SET_DTX(*x*)

Configures the encoder's use of discontinuous transmission.

Note

This is only applicable to the LPC layer

Parameters

`[in]` ***x*** `int`: DTX flag, 0 (disabled) is default

4.6.2.18 #define OPUS_SET_FORCE_CHANNELS(*x*)

Configures mono/stereo forcing in the encoder.

This is useful when the caller knows that the input signal is currently a mono source embedded in a stereo stream.

Parameters

`[in]` ***x*** `int`: OPUS_AUTO (default); 1 (forced mono); 2 (forced stereo)

4.6.2.19 #define OPUS_SET_INBAND_FEC(*x*)

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

Note

This is only applicable to the LPC layer

Parameters

`[in]` ***x*** `int`: FEC flag, 0 (disabled) is default

4.6.2.20 #define OPUS_SET_MAX_BANDWIDTH(*x*)

Configures the encoder's maximum bandpass allowed,.

See also

OPUS_GET_MAX_BANDWIDTH The supported values are:

- OPUS_BANDWIDTH_NARROWBAND 4kHz passband
- OPUS_BANDWIDTH_MEDIUMBAND 6kHz passband
- OPUS_BANDWIDTH_WIDEBAND 8kHz passband
- OPUS_BANDWIDTH_SUPERWIDEBAND 12kHz passband
- OPUS_BANDWIDTH_FULLBAND 20kHz passband (default)

Parameters

`[in]` ***x*** `int`: Bandwidth value

4.6.2.21 #define OPUS_SET_PACKET_LOSS_PERC(*x*)

Configures the encoder's expected packet loss percentage.

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

Parameters

`[in]` ***x*** `int`: Loss percentage in the range 0-100, inclusive.

4.6.2.22 #define OPUS_SET_SIGNAL(*x*)

Configures the type of signal being encoded.

This is a hint which helps the encoder's mode selection. The supported values are:

- OPUS_SIGNAL_AUTO (default)
- OPUS_SIGNAL_VOICE
- OPUS_SIGNAL_MUSIC

Parameters

`[in]` ***x*** `int`: Signal type

4.6.2.23 #define OPUS_SET_VBR(*x*)

Configures VBR in the encoder.

The following values are currently supported:

- 0 CBR
- 1 VBR (default) The configured bitrate may not be met exactly because frames must be an integer number of bytes in length.

Warning

Only the MDCT mode of Opus can provide hard CBR behavior.

Parameters

`[in]` ***x*** `int`: 0; 1 (default)

4.6.2.24 #define OPUS_SET_VBR_CONSTRAINT(*x*)

Configures constrained VBR in the encoder.

The following values are currently supported:

- 0 Unconstrained VBR (default)
- 1 Maximum one frame buffering delay assuming transport with a serialization speed of the nominal bitrate This setting is irrelevant 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.

  Parameters

  `[in]` ***x*** `int`: 0 (default); 1

4.7 Generic CTLs

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

Defines

- #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_PITCH(x)

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

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

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

4.7.2 Define Documentation

4.7.2.1 #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*** `int*`: Bandwidth value

4.7.2.2 #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 #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)

4.7.2.4 #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.

4.8 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.

4.8.1 Function Documentation

4.8.1.1 const char* opus_get_version_string (void)

Gets the libopus version string.

Returns

Version string

4.8.1.2 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

Chapter 5

File Documentation

5.1 opus.h File Reference

Opus reference implementation API.

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

Typedefs

- typedef struct OpusEncoder OpusEncoder

  Opus encoder state.

- typedef struct OpusDecoder OpusDecoder

  Opus decoder state.

- typedef struct OpusRepacketizer OpusRepacketizer

Functions

- int opus_encoder_get_size (int channels)
- 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 the size returned by opus_encoder_get_size.

- int opus_encode (OpusEncoder ∗st, const opus_int16 ∗pcm, int frame_size, unsigned char ∗data, int max_data_bytes)

Encodes an Opus frame.

- int opus_encode_float (OpusEncoder ∗st, const float ∗pcm, int frame_size, unsigned char ∗data, int 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, int len, opus_int16 ∗pcm, int frame_size, int decode_fec)

  Decode an Opus frame.

- int opus_decode_float (OpusDecoder ∗st, const unsigned char ∗data, int len, float ∗pcm, int frame_size, int decode_fec)

  Decode an opus frame 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_packet_parse (const unsigned char ∗data, int len, unsigned char ∗out_toc, const unsigned char ∗frames[48], short 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[ ], int len)

  Gets the number of frames in an Opus packet.

- int opus_decoder_get_nb_samples (const OpusDecoder ∗dec, const unsigned char packet[ ], int len)

  Gets the number of samples of an Opus packet.

- int opus_repacketizer_get_size (void)
- OpusRepacketizer ∗ opus_repacketizer_init (OpusRepacketizer ∗rp)
- OpusRepacketizer ∗ opus_repacketizer_create (void)
- void opus_repacketizer_destroy (OpusRepacketizer ∗rp)
- int opus_repacketizer_cat (OpusRepacketizer ∗rp, const unsigned char ∗data, int len)
- opus_int32 opus_repacketizer_out_range (OpusRepacketizer ∗rp, int begin, int end, unsigned char ∗data, int maxlen)
- int opus_repacketizer_get_nb_frames (OpusRepacketizer ∗rp)
- opus_int32 opus_repacketizer_out (OpusRepacketizer ∗rp, unsigned char ∗data, int maxlen)

5.1.1 Detailed Description

Opus reference implementation API.

5.2 opus_defines.h File Reference

Opus reference implementation constants.

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

Defines

- #define OPUS_OK

  No error.

- #define OPUS_BAD_ARG

  One or more invalid/out of range arguments.

- #define OPUS_BUFFER_TOO_SMALL

  The mode struct passed is invalid.

- #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_SIGNAL_VOICE 3001

  Signal being encoded is voice.

- #define OPUS_SIGNAL_MUSIC 3002

  Signal being encoded is music.

- #define OPUS_BANDWIDTH_NARROWBAND

  4kHz bandpass

- #define OPUS_BANDWIDTH_MEDIUMBAND

  6kHz bandpass

- #define OPUS_BANDWIDTH_WIDEBAND

  8kHz bandpass

- #define OPUS_BANDWIDTH_SUPERWIDEBAND

  12kHz bandpass

- #define OPUS_BANDWIDTH_FULLBAND

  20kHz bandpass

- #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)

  Configures VBR in the encoder.

- #define OPUS_GET_VBR(x)

  Gets the encoder's VBR configuration,.

- #define OPUS_SET_VBR_CONSTRAINT(x)

  Configures constrained VBR in the encoder.

- #define OPUS_GET_VBR_CONSTRAINT(x)

  Gets the encoder's constrained VBR configuration.

- #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 encoder's maximum bandpass allowed,.

- #define OPUS_GET_MAX_BANDWIDTH(x)

  Gets the encoder's configured maximum bandpass allowed,.

- #define OPUS_SET_BANDWIDTH(x)

  Configures the encoder's bandpass,.

- #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.

- #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.

- #define OPUS_GET_DTX(x)

  Gets encoder's configured use of discontinuous transmission,.

- #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_PITCH(x)

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

- #define OPUS_GET_BANDWIDTH(x)

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

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.

5.2.1 Detailed Description

Opus reference implementation constants.

5.3 opus_multistream.h File Reference

Opus reference implementation multistream API.

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

Defines

- #define __opus_check_encstate_ptr(ptr) ((ptr) + ((ptr) - (OpusEncoder∗∗)(ptr)))
- #define __opus_check_decstate_ptr(ptr) ((ptr) + ((ptr) - (OpusDecoder∗∗)(ptr)))
- #define OPUS_MULTISTREAM_GET_ENCODER_STATE_REQUEST 5120
- #define OPUS_MULTISTREAM_GET_DECODER_STATE_REQUEST 5122
- #define OPUS_MULTISTREAM_GET_ENCODER_STATE(x, y) OPUS_MULTISTREAM_GET_ENCODER_STATE_REQUEST, __opus_check_int(x), __opus_check_encstate_ptr(y)
- #define OPUS_MULTISTREAM_GET_DECODER_STATE(x, y) OPUS_MULTISTREAM_GET_DECODER_STATE_REQUEST, __opus_check_int(x), __opus_check_decstate_ptr(y)

Typedefs

- typedef struct OpusMSEncoder OpusMSEncoder
- typedef struct OpusMSDecoder OpusMSDecoder

Functions

- OpusMSEncoder ∗ opus_multistream_encoder_create (opus_int32 Fs, int channels, int streams, int coupled_streams, unsigned char ∗mapping, int application, int ∗error)

    Allocate and initialize a multistream encoder state object.

- int opus_multistream_encoder_init (OpusMSEncoder ∗st, opus_int32 Fs, int channels, int streams, int coupled_streams, unsigned char ∗mapping, int application)

  Initialize an already allocated multistream encoder state.

- int opus_multistream_encode (OpusMSEncoder ∗st, const opus_int16 ∗pcm, int frame_size, unsigned char ∗data, int max_data_bytes)

  Returns length of the data payload (in bytes) or a negative error code.

- int opus_multistream_encode_float (OpusMSEncoder ∗st, const float ∗pcm, int frame_size, unsigned char ∗data, int max_data_bytes)

  Returns length of the data payload (in bytes) or a negative error code.

- opus_int32 opus_multistream_encoder_get_size (int streams, int coupled_-streams)

  Gets the size of an OpusMSEncoder structure.

- void opus_multistream_encoder_destroy (OpusMSEncoder ∗st)

  Deallocate a multstream encoder state.

- int opus_multistream_encoder_ctl (OpusMSEncoder ∗st, int request,...)

  Get or set options on a multistream encoder state.

- OpusMSDecoder ∗ opus_multistream_decoder_create (opus_int32 Fs, int channels, int streams, int coupled_streams, unsigned char ∗mapping, int ∗error)

  Allocate and initialize a multistream decoder state object.

- int opus_multistream_decoder_init (OpusMSDecoder ∗st, opus_int32 Fs, int channels, int streams, int coupled_streams, unsigned char ∗mapping)

  Intialize a previously allocated decoder state object.

- int opus_multistream_decode (OpusMSDecoder ∗st, const unsigned char ∗data, int len, opus_int16 ∗pcm, int frame_size, int decode_fec)

  Returns the number of samples decoded or a negative error code.

- int opus_multistream_decode_float (OpusMSDecoder ∗st, const unsigned char ∗data, int len, float ∗pcm, int frame_size, int decode_fec)

  Returns the number of samples decoded or a negative error code.

- opus_int32 opus_multistream_decoder_get_size (int streams, int coupled_-streams)

  Gets the size of an OpusMSDecoder structure.

- int opus_multistream_decoder_ctl (OpusMSDecoder ∗st, int request,...)

  Get or set options on a multistream decoder state.

- void opus_multistream_decoder_destroy (OpusMSDecoder ∗st)

Deallocate a multistream decoder state object.

5.3.1 Detailed Description

Opus reference implementation multistream API.

5.3.2 Define Documentation

5.3.2.1 #define __opus_check_decstate_ptr(*ptr*) ((ptr) + ((ptr) - (OpusDecoder∗∗)(ptr)))

5.3.2.2 #define __opus_check_encstate_ptr(*ptr*) ((ptr) + ((ptr) - (OpusEncoder∗∗)(ptr)))

5.3.2.3 #define OPUS_MULTISTREAM_GET_DECODER_STATE(*x*, *y*) OPUS_MULTISTREAM_GET_DECODER_STATE_REQUEST, __opus_check_int(x), __opus_check_decstate_ptr(y)

5.3.2.4 #define OPUS_MULTISTREAM_GET_DECODER_STATE_-REQUEST 5122

5.3.2.5 #define OPUS_MULTISTREAM_GET_ENCODER_STATE(*x*, *y*) OPUS_MULTISTREAM_GET_ENCODER_STATE_REQUEST, __opus_check_int(x), __opus_check_encstate_ptr(y)

5.3.2.6 #define OPUS_MULTISTREAM_GET_ENCODER_STATE_-REQUEST 5120

5.3.3 Typedef Documentation

5.3.3.1 typedef struct OpusMSDecoder OpusMSDecoder

5.3.3.2 typedef struct OpusMSEncoder OpusMSEncoder

5.3.4 Function Documentation

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

Returns the number of samples decoded or a negative error code.

Parameters

st Decoder state

data Input payload. Use a NULL pointer to indicate packet loss

len Number of bytes in payload

pcm Output signal, samples interleaved in channel order . length is frame_-size∗channels

frame_size Number of samples per frame of input signal

decode_fec Flag (0/1) 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.

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

Returns the number of samples decoded or a negative error code.

Parameters

st Decoder state

data Input payload buffer. Use a NULL pointer to indicate packet loss

len Number of payload bytes in data

pcm Buffer for the output signal (interleaved iin channel order). length is frame_-size∗channels

frame_size Number of samples per frame of input signal

decode_fec Flag (0/1) 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.

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

Allocate and initialize a multistream decoder state object.

Call opus_multistream_decoder_destroy() to release this object when finished.

Parameters

Fs Sampling rate to decode at (Hz)

channels Number of channels to decode

streams Total number of coded streams in the multistream

coupled_streams Number of coupled (stereo) streams in the multistream

mapping Stream to channel mapping table

error Error code

5.3.4.4 int opus_multistream_decoder_ctl (OpusMSDecoder ∗ *st,* int *request,* ...)

Get or set options on a multistream decoder state.

5.3.4.5 void opus_multistream_decoder_destroy (OpusMSDecoder ∗ *st*)

Deallocate a multistream decoder state object.

5.3.4.6 opus_int32 opus_multistream_decoder_get_size (int *streams,* int *coupled_streams*)

Gets the size of an OpusMSDecoder structure.

Returns

size

Parameters

streams	Total number of coded streams
coupled_streams	Number of coupled (stereo) streams

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

Intialize a previously allocated decoder state object.

Parameters

st	Encoder state
Fs	Sample rate of input signal (Hz)
channels	Number of channels in the input signal
streams	Total number of coded streams
coupled_streams	Number of coupled (stereo) streams
mapping	Stream to channel mapping table

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

Returns length of the data payload (in bytes) or a negative error code.

Parameters

st	Encoder state
pcm	Input signal as interleaved samples. Length is frame_size∗channels
frame_size	Number of samples per frame of input signal
data	Output buffer for the compressed payload (no more than max_data_bytes long)
max_data_bytes	Allocated memory for payload; don't use for controlling bitrate

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

Returns length of the data payload (in bytes) or a negative error code.

Parameters

st	Encoder state
pcm	Input signal interleaved in channel order. length is frame_size∗channels
frame_size	Number of samples per frame of input signal
data	Output buffer for the compressed payload (no more than max_data_bytes long)
max_data_bytes	Allocated memory for payload; don't use for controlling bitrate

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

Allocate and initialize a multistream encoder state object.

Call opus_multistream_encoder_destroy() to release this object when finished.

Parameters

Fs	Sampling rate of input signal (Hz)
channels	Number of channels in the input signal
streams	Total number of streams to encode from the input
coupled_streams	Number of coupled (stereo) streams to encode
mapping	Encoded mapping between channels and streams
application	Coding mode (OPUS_APPLICATION_VOIP/OPUS_-APPLICATION_AUDIO)
error	Error code

5.3.4.11 int opus_multistream_encoder_ctl (OpusMSEncoder ∗ *st*, int *request*, ...)

Get or set options on a multistream encoder state.

5.3.4.12 void opus_multistream_encoder_destroy (OpusMSEncoder ∗ *st*)

Deallocate a multstream encoder state.

5.3.4.13 opus_int32 opus_multistream_encoder_get_size (int *streams*, int *coupled_streams*)

Gets the size of an OpusMSEncoder structure.

Returns

size

Parameters

streams Total number of coded streams

coupled_streams Number of coupled (stereo) streams

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

Initialize an already allocated multistream encoder state.

Parameters

st Encoder state

Fs Sampling rate of input signal (Hz)

channels Number of channels in the input signal

streams Total number of streams to encode from the input

coupled_streams Number of coupled (stereo) streams to encode

mapping Encoded mapping between channels and streams

application Coding mode (OPUS_APPLICATION_VOIP/OPUS_-APPLICATION_AUDIO)

5.4 opus_types.h File Reference

Opus reference implementation types.

Defines

- #define opus_int int
- #define opus_int64 long long
- #define opus_int8 signed char
- #define opus_uint unsigned int
- #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

5.4.1 Detailed Description

Opus reference implementation types.

5.4.2 Define Documentation

5.4.2.1 #define opus_int int

5.4.2.2 #define opus_int64 long long

5.4.2.3 #define opus_int8 signed char

5.4.2.4 #define opus_uint unsigned int

5.4.2.5 #define opus_uint64 unsigned long long

5.4.2.6 #define opus_uint8 unsigned char

5.4.3 Typedef Documentation

5.4.3.1 typedef short opus_int16

5.4.3.2 typedef int opus_int32

5.4.3.3 typedef unsigned short opus_uint16

5.4.3.4 typedef unsigned int opus_uint32

Index